Politechnika Śląska
Wydział Automatyki, Elektroniki i Informatyki
Informatyka Techniczna i Telekomunikacja

Rozprawa doktorska

Korekcja danych z sekwencjonowania genomów

Maciej Długosz

Promotor: prof. dr hab. inż. Sebastian Deorowicz

Gliwice, 2023

Spis treści

Rozdział 1

Wstęp

Wśród licznych odkryć naukowych oraz osiągnięć technicznych XX i XXI wieku szczególną doniosłością charakteryzują się sukcesy w dziedzinie szeroko pojętych badań nad genetyką oraz analizą informacji genetycznej zawartej w komórkach organizmów żywych. Istotność zagadnienia, powiązana z jego „medialnością", ponoszonymi ogromnymi nakładami finansowymi oraz zaangażowaniem rzeszy specjalistów, jest efektem z jednej strony znaczącej, interdyscyplinarnej wiedzy niezbędnej do podtrzymania postępu w dziedzinie, a z drugiej niemożliwego do przecenienia utylitaryzmu osiągnięć.

W niniejszej pracy zostanie omówione zagadnienie korekcji danych uzyskanych w wyniku procesu *sekwencjonowania*, czyli odczytywania pierwszorzędowej struktury cząsteczek kwasu DNA. Sama zasada działania takiego procesu jest obszerną tematyką, obejmującą wiedzę z dziedzin takich jak biochemia, fizyka chemiczna, biologia molekularna, informatyka i innych. Rozwój sekwencjonowania następuje ciągle od lat 60. XX wieku, gdy zastosowanie wczesnych metod fizykochemicznych pozwoliło na odczytanie krótkich fragmentów kwasów nukleinowych, ewoluując do technik sekwencjonowania pierwszej generacji, które w 2001 roku dały możliwość wyznaczenia sekwencji ludzkiego genomu [124, 199]. Kolejny krok stanowiło opracowanie wysokoprzepustowych technik sekwencjonowania drugiej generacji, dających potencjał do pozyskania dużej ilości informacji w krótkim czasie oraz niskim nakładem kosztów [30, 148, 171]. Obecnie coraz większe znaczenie zyskują techniki sekwencjonowania trzeciej generacji, pozwalając na dalsze obniżenie kosztów, zwiększenie przepustowości, redukcję niedoskonałości technik drugiej generacji, choć jednocześnie wiążąc się z powstawaniem nowych wyzwań przetwarzania takich danych [31]. W obu ostatnich przypadkach proces sekwencjonowania jest przeprowadzany przy pomocy zautomatyzowanych urządzeń (*sekwenatorów*). Obecnie zdecydowanie najczęściej praktyczne zastosowanie znajdują dane z sekwenatorów marki Illumina, zaliczanych do urządzeń drugiej

generacji.

Właściwości technik drugiej i trzeciej generacji otworzyły drzwi do przeprowadzania stosunkowo łatwej analizy w przedmiocie zarówno badań podstawowych, jak też uwzględnienia jej w naukach stosowanych oraz działalności praktycznej. Spośród wielu zastosowań takich technik można wymienić choćby wyznaczanie sekwencji genomów kolejnych organizmów, w tym organizmów niemodelowych (przeprowadzanie asemblacji *de novo* [25, 107, 154]), detekcję polimorfizmów pojedynczych nukleotydów [25], identyfikację elementów funkcjonalnych genomów [97], wykrywanie oddziaływań DNA–białko [25, 107, 154], detekcję mutacji obecnych w tkankach nowotworowych [107, 154], medycynę spersonalizowaną [154], identyfikację mutacji chromosomowych [25] czy analizę metagenomu (zestawu kwasów nukleinowych wielu organizmów z danego środowiska) [107, 154].

Przetwarzanie uzyskanych w wyniku sekwencjonowania danych, mające na celu choćby wykonanie wymienionych wyżej zadań, wiąże się z koniecznością zmierzenia z szeregiem teoretycznych i praktycznych trudności informatycznych. Wymaga opracowania metod modelowania oraz statystycznej analizy danych, a także sprostania czysto technicznym wyzwaniom, będących efektem ogromnego, sięgającego tebibajtów rozmiaru danych. Przezwyciężenie tych problemów przy użyciu powszechnie stosowanych metod i narzędzi informatyki może być niemożliwe. W rezultacie niezbędne jest dokładne opracowanie algorytmów, struktur danych oraz strategii podejścia do problemu, staranna implementacja programu, a w wielu przypadkach zastosowanie odpowiednio zaawansowanego otoczenia sprzętowego. Ponadto interdyscyplinarność tematyki wiąże się z uwzględnieniem czysto organizacyjnych niuansów, rodząc wymaganie włączenia się specjalistów różnych dziedzin lub konieczności pozyskania przez twórców rozwiązań informatycznych odpowiedniej wiedzy z dziedziny problemowej. Zespół dziedzin składowych zagadnienia przetwarzania informacji biologicznej stanowił przyczynę narodzin nowej dyscypliny — *bioinformatyki*, łączącej problemy z zakresu nauk biologicznych, jak genomika, transkryptomika czy proteomika, z rozwiązaniami dostarczanymi w wyniku zastosowania technik informatycznych.

Do wymienionych aspektów komputerowego przetwarzania informacji genetycznej należy zaliczyć powszechne zjawisko występowania — nazywanych dalej błędami — niezgodności między uzyskanymi danymi sekwencjonowania a stanem faktycznym, charakteryzującym próbki materiału biologicznego. Powstawanie błędów jest efektem niedoskonałości technik przygotowania oraz sekwencjonowania materiału DNA. Stanowi echo operowania na strukturach o rozmiarach atomowych, w których pojawienie się niewielkich tylko zakłóceń, uwidocznienie się subtelnych właściwości fizycznych bądź chemicznych materiału, wystąpienie niedoskonałości aparatury lub niedokładności wykorzystywanych modeli matematycznych może powodować niewłaściwe określenie sekwencji analizowanego kwasu nukleinowego. Błędy są propagowane między kolejnymi etapami przetwarzania

danych, rodząc trudności techniczne wynikające ze wzajemnej niezgodności różnych części zestawu danych, stanowiąc czynnik ryzyka uzyskania niewłaściwych wyników lub wyciągnięcia nieprawidłowych wniosków oraz wymuszając dodatkowy wzrost zawiłości algorytmów przetwarzania, związany z dostosowaniem do obecności błędów.

Wspomniane zjawisko jest redukowane w wyniku stosowania rozmaitych podejść. Należą do nich coraz lepsze poznanie charakterystyki błędów wyników pracy różnych sekwenatorów, analiza materiału DNA poddanego amplifikacji w miejsce pojedynczych cząsteczek, zapewnienie znacznej nadmiarowości danych uzyskanych podczas sekwencjonowania, uwzględnianie mechanizmów samokontroli procesu sekwencjonowania w formie generacji współczynników jakości danych czy standardowe wyposażanie (kosztem komplikacji i wzrostu zapotrzebowania na zasoby) samych algorytmów przetwarzania w mechanizmy pozwalające na częściowe sprostanie temu wyzwaniu. Mimo to problem w dalszym ciągu nie został zażegnany. W literaturze wielokrotnie poruszano zagadnienie komputerowej korekcji sekwencji, opierającej się na wspomnianej nadmiarowości danych oraz dostępności danych kontrolnych, wykorzystującej wcześniej opracowane sekwencje genomów, przeprowadzających syntezę danych uzyskanych przy pomocy różnych technik o odmiennych własnościach oraz opartych na wiedzy dotyczącej typowej charakterystyki błędów. W tym celu zaproponowano szereg algorytmów, których zadaniem jest przetworzenie zestawu odczytów sekwencjonowania poprzez możliwie skuteczną detekcję błędów oraz ich korekcję albo eliminację obarczonych błędami sekwencji. W dużej mierze zagadnienie dotyczy danych z akwizycji przy pomocy urządzeń marki Illumina, co wynika z ich wysokiej popularności. W niniejszej pracy położono nacisk na analizę sekwencji uzyskanych techniką tego rodzaju.

Przegląd literatury pozwala na wyciągnięcie wniosku, że tematyka nie została wyczerpana. Obecnie wybór algorytmu musi wiązać się z wypracowaniem kompromisu między możliwościami oraz skutecznością algorytmu, a zapotrzebowaniem na zasoby obliczeniowe. Przeprowadzone dotychczas badania eksperymentalne nie obejmują dokładnej analizy niektórych kwestii, jak wpływu korekcji na detekcję wariantów genomów czy skuteczności często spotykanej metody oceny korekcji przy pomocy odczytów uzyskanych w wyniku komputerowej symulacji. Jednocześnie w przedstawianych w literaturze wynikach można zaobserwować niespójność, polegającą na prezentowaniu bardzo dobrych wyników danego algorytmu przez jego autorów oraz niejednokrotnie słabych — przez autorów konkurencyjnych rozwiązań. Motywacją przygotowania podjętych wysiłków jest uzupełnienie tych kwestii.

W związku z powyższym postawiono następujące tezy, których udowodnienie stanowi główny przedmiot niniejszej pracy:

1. zastosowanie niektórych nowoczesnych metod korekcji odczytów sekwencjo-

nowania genomów pozwala na redukcję liczby błędów powstałych w trakcie sekwencjonowania oraz poprawę jakości różnych wyników algorytmów przetwarzania odczytów, wliczając w to detekcję wariantów genomów przy niskiej głębokości sekwencjonowania,

2. możliwe jest opracowanie nowego algorytmu korekcji, zapewniającego lepszy od istniejących algorytmów bilans skuteczności do zapotrzebowania na zasoby, a przy tym niskie ryzyko degradacji jakości danych.

Wśród szczegółowych celów przyświecających przeprowadzonym badaniom znajduje się przygotowanie nowego algorytmu korekcji sekwencji nukleotydowych, uzyskanych z urządzeń marki Illumina w efekcie sekwencjonowania pełnego genomu. Przy jego opracowaniu wzięto pod uwagę wymagania dotyczące zakresu zastosowań, jakości uzyskiwanych wyników, szeroko pojętej wydajności oraz funkcjonalności, opierając się na doświadczeniach użytkowania innych algorytmów. Przygotowano koncepcję algorytmu, uwzględniającą dostępność pomocniczych rozwiązań i narzędzi, zostały również zaproponowane i wykorzystane autorskie rozwiązania konkretnych problemów. W rezultacie przygotowano implementację oraz przeprowadzono strojenie algorytmu oraz opracowano strategię doboru głównego parametru algorytmu, w oparciu o dane wejściowe. Jako formę teoretycznej miary szybkości działania algorytmu wykorzystano wyznaczoną złożoność obliczeniową wybranych części algorytmu.

Do istotnych celów należy zaliczyć też badania eksperymentalne, nastawione na ocenę nowego algorytmu oraz przeprowadzenie analizy porównawczej z konkurencyjnymi algorytmami. Badania zostały przeprowadzone opierając się na licznych kryteriach, pozwalając na ocenę wyników z różnych punktów widzenia. Pod uwagę wzięto jakość wyników, mierzoną skutecznością korekcji błędów, wyrażoną przy pomocy rozmaitych miar. Analizie poddano kwestię wydajności (zapotrzebowania na zasoby) algorytmów. Wykorzystano przy tym zestawy danych o różnej charakterystyce. Istotnym składnikiem tego celu jest weryfikacja potencjału zadania detekcji wariantów w roli narzędzia oceny skuteczności korekcji.

W eksperymentach wykorzystano zarówno dane uzyskane jako efekt rzeczywistego sekwencjonowania DNA, jak też wygenerowane komputerowo w procesie symulacji procesu sekwencjonowania, przeprowadzonego różnymi metodami. Dzięki temu wyznaczono poboczny cel, polegający na określeniu praktycznej użyteczności oraz realizmu uzyskanych w ten sposób sekwencji, które zostało dokonane w oparciu o uzyskane wyniki.

Zaproponowany algorytm oraz wyniki eksperymentalne zostały do tej pory upublicznione m.in. w formie następujących prac: artykułu w czasopiśmie Bioinformatics [65], rozdziałów w monografiach [64, 67] oraz dwóch plakatów przedstawionych podczas sesji konferencyjnych. Obecnie trwa również proces recenzji artykułu dla czasopisma Scientific Reports [66].

Struktura pracy jest następująca. W rozdziale 2. przedstawiono wybór informacji związanych z ewolucją i biologią w kontekście własności kwasów nukleinowych. Wiedza ta została wykorzystana do omówienia historycznych technik sekwencjonowania RNA i DNA, a także współczesnych technik sekwencjonowania DNA drugiej i trzeciej generacji. Dla tych ostatnich wyjaśniono fizykochemiczne idee procesu sekwencjonowania, mogące stanowić podstawę do uzasadnienia własności danej techniki oraz związanych z nią charakterystyki błędów.

Rozdział 3. został poświęcony omówieniu wybranych zagadnień informatycznych, w ukierunkowaniu na możliwość wykorzystania w algorytmach przetwarzania sekwencji nukleotydowych. Sformalizowano istotne pojęcia biologiczne oraz terminy związane z modelowaniem tego rodzaju danych. Zostały omówione wybrane algorytmy pełniące pomocniczą rolę w zadaniu korekcji odczytów.

W rozdziale 4. wykonano przegląd literatury pod kątem istniejących algorytmów korekcji odczytów z sekwencjonowania DNA, wraz z zasadami działania. Przedstawiono problem parametryzacji algorytmów oraz omówiono obecne w literaturze kryteria oceny algorytmów, pozwalające na dokonanie analizy porównawczej. Dokonano przeglądu metod i narzędzi komputerowej generacji odczytów sekwencjonowania.

W rozdziale 5. zawarto opis nowego algorytmu korekcji. Punkt wyjścia stanowił opis szczegółów działania oraz analiza złożoności obliczeniowej jednego z istniejących już algorytmów, którego implementacja w formie kodu źródłowego stała się podstawą opracowania nowego algorytmu. Następnie przedstawiono specyfikację wymagań nowego algorytmu, a w oparciu o nią dokonano opracowania algorytmu. Została zaproponowana strategia parametryzacji algorytmu oraz przedstawiono wyniki analizy złożoności obliczeniowej algorytmu.

Rozdział 6. poświęcono opisowi badań eksperymentalnych. Zaproponowano przebieg badań, uzasadniono dobór wykorzystanych zbiorów danych oraz zaproponowano zestaw kryteriów oceny. W dalszej części przedstawiono i omówiono wyniki eksperymentów, opierając się zarówno na dostępnych w bazach danych sekwencjach uzyskanych w wyniku sekwencjonowania DNA, jak też wygenerowanych komputerowo. Zaprezentowano wyniki jakościowe oraz ocenę wydajności. Rezultaty zostały wykorzystane do weryfikacji spełnienia wymagań nowego algorytmu, określenia praktycznej użyteczności procesu korekcji i wyróżnienia algorytmów zalecanych do użycia w innych pracach. Wykonano analizę problemu parametryzacji algorytmów. Wyciągnięto wnioski dotyczące użyteczności sekwencji uzyskanych w wyniku komputerowej symulacji w zadaniu oceny algorytmów korekcji.

Dodatkowo w załączniku A, w formie pseudokodów, zamieszczono uzupełniające specyfikacje działania niektórych algorytmów. W załączniku B zawarto szczegóły wykonania eksperymentów, przede wszystkim wiersze poleceń uruchomienia implementacji oraz wszystkie ich parametry, pozwalające na pełne po-

wtórzenie eksperymentów. W załączniku C zostały zawarte uzupełniające wyniki eksperymentów.

Rozdział 2

Genomy i ich sekwencjonowanie

2.1 Wprowadzenie

2.1.1 Kwasy DNA i RNA

Historia życia na Ziemi rozpoczęła się ok. 3,5–4 mld lat temu [204]. Z licznych badań, w szczególności genetycznych, wynika istnienie w tym okresie organizmu nazywanego *ostatnim uniwersalnym wspólnym przodkiem*, od którego pochodzą wszystkie organizmy występujące w dowolnym momencie historii świata. Cechy charakteryzujące ten organizm (*cechy fenotypowe*) były przekazywane kolejnym pokoleniom, jednocześnie ulegając powolnym, lecz sukcesywnym zmianom. Na skutek tych zmian potomne osobniki w innym stopniu mogły dostosować się do warunków panujących w środowisku, a w konsekwencji zapewniły sobie lepsze lub gorsze warunki do przeżycia. Lepiej dostosowane osobniki częściej osiągały możliwość zapoczątkowania nowego pokolenia (rozmnożenia się) i przekazania mu swoich cech, podczas gdy pozostałe wcześniej ulegały eliminacji ze środowiska na skutek wewnętrznych bądź zewnętrznych czynników. Powyższy, trwający do dziś proces nazywany jest *ewolucją*, a zestaw wszystkich cech fenotypowych charakteryzujących danego osobnika nazywany jest jego *fenotypem*.

Miliardy lat ewolucji doprowadziły do wykształcenia się ogromnej liczby organizmów charakteryzujących się niesamowicie odmiennymi cechami — rozmiarami, budową anatomiczną, strategiami przeżycia, środowiskami występowania i wieloma innymi. Taka różnorodność stanowi wyzwanie dla jednego z głównych działów biologii, jaką jest *taksonomia*. Jej celem jest wprowadzenie wyczerpującej systematyki i opisu wszystkich żyjących organizmów, a także (we współudziale *paleontologii*) organizmów wymarłych.

Budowa kwasów nukleinowych

Nośnikami umożliwiającymi przekazywanie informacji o cechach organizmów oraz przechowywanie jej w trakcie życia są *kwasy nukleinowe*. Wśród nich wyróżnia się w szczególności *kwas deoksyrybonukleinowy* (DNA) oraz *kwas rybonukleinowy* (RNA). Kwasy nukleinowe są biopolimerami zbudowanymi z monomerów zwanych *nukleotydami*. W skład nukleotydu wchodzi cząsteczka jednej z pięciu zasad azotowych: *adeniny*, *guaniny* (zasady *purynowe*), *cytozyny*, *tyminy* albo *uracylu* (zasady *pirymidynowe*). Regułą jest występowanie tyminy w nukleotydach DNA, a uracylu w nukleotydach RNA, choć spotykane są także przypadki obecności uracylu w DNA [195]. Pozostałymi składnikami nukleotydu są reszta cukrowa (odpowiednio deoksyryboza albo ryboza) oraz reszta fosforanowa. Z semantycznego punktu widzenia struktura kwasów nukleinowych jest liniowa, tzn. w obrębie cząsteczki można wyznaczyć liniowy porządek budujących go nukleotydów, jednakże wyniki najnowszych badań sugerują, że w przypadku DNA istotne jest także przestrzenne rozmieszczenie jego nukleotydów [74]. W niektórych przypadkach struktura DNA może mieć charakter cykliczny i tworzyć cząsteczkę kolistą. Pod względem fizykochemicznym nie występują żadne ograniczenia dotyczące kolejności nukleotydów w cząsteczce [101]. Rozmiar cząsteczki DNA może osiągać kilkaset milionów nukleotydów, natomiast rozmiar RNA na ogół nie przekracza kilku tysięcy nukleotydów [101]. W celu skrócenia zapisu w dalszej części pracy nukleotydy będą oznaczane symbolami A, C, G, T oznaczającymi odpowiednio nukleotyd adeninowy, cytozynowy, guaninowy, tyminowy oraz uracylowy.

Pojedynczą cząsteczkę kwasu nukleinowego określa się mianem *nici*. Nić lub jej fragment długości do kilkuset nukleotydów nosi nazwę *oligonukleotydu*, *oligomeru* lub *k-meru*, gdzie k jest liczbą nukleotydów w cząsteczce.

W budowie kwasów nukleinowych wyróżnia się dwa szczególne rodzaje wiązań chemicznych występujących między atomami różnych nukleotydów. *Wiązania fosfodiestrowe* powstają między sąsiednimi nukleotydami w nici. Atom węgla reszty cukrowej nukleotydu, oznaczany $3'$, łączy się — poprzez jego resztę fosforanową — z atomem $5'$ reszty cukrowej następnego nukleotydu. Ten sposób połączenia nukleotydów odpowiada za liniowy charakter nici. W efekcie można zdefiniować kolejność nukleotydów w cząsteczce od końca $5'$ do końca $3'$ ($5'$–$3'$) albo odwrotnie. Kolejność występowania określonych nukleotydów w nici określa się mianem jego *struktury pierwszorzędowej*.

Własności kwasów nukleinowych

Cząsteczki zasad azotowych różnych nukleotydów mogą także tworzyć między sobą *wiązania wodorowe*. Umożliwia to powstawanie połączeń między nukleotydami należących do różnych nici albo różnych fragmentów tej samej nici. Wiązania wodorowe cząsteczek adeniny i tyminy (lub uracylu) mają charakter podwójny, z kolei wiązania cytozyny i guaniny charakter potrójny. Tym samym możliwe jest

powstawanie par adenina–tymina (uracyl) albo cytozyna–guanina. Nukleotydy, których zasady azotowe mogą tworzyć powyższe dwójki, nazywamy *nukleotydami komplementarnymi*, a regułę ich parowania *zasadą komplementarności*.

Typową postacią występowania DNA jest cząsteczka sparowana, w której dwie nici (*nici komplementarne*) zbudowane z komplementarnych nukleotydów występują razem, połączone wiązaniami wodorowymi swoich zasad azotowych. Wzajemne ułożenie nici ma charakter antyrównoległy, co jest efektem przeciwnych orientacji nici — jedna nić w kolejności $5'$–$3'$, druga w odwrotnej. Struktura przestrzenna (tzw. *struktura drugorzędowa*) sparowanych nici DNA ma kształt *podwójnej helisy*. Za odkrycie przestrzennej budowy DNA Watson i Crick [200] otrzymali w 1962 roku Nagrodę Nobla w dziedzinie medycyny lub fizjologii. Ze względu na budowę sparowanej cząsteczki DNA za jednostkę długości kwasu DNA przyjmuje się *parę zasad* — pz (ang. *base pair* — bp). Jedna para zasad odpowiada nici zbudowanej z dwóch sparowanych nukleotydów komplementarnych. Ponadto, szczególnie w kontekście procesu sekwencjonowania DNA, za jednostkę ilości informacji genetycznej przyjmuje się *zasadę* — z (ang. *base* — b).

Istotnym z punktu widzenia inżynierii genetycznej przypadkiem są sparowane cząsteczki DNA posiadające końcówkę zbudowaną z krótkiego, niesparowanego odcinka. Końcówki tego typu nazywane są *lepkimi końcami* (ang. *cohesive end*).

Cząsteczki RNA często przyjmują formę jednoniciową. Powszechne jest jednak występowanie wiązań wodorowych między różnymi fragmentami tej samej nici, czego efektem jest powstawanie w strukturze drugorzędowej RNA charakterystycznych motywów, jak odcinki dwuniciowe, pętle czy wybrzuszenia [166].

Każda komórka somatyczna organizmu stanowi nośnik DNA. Jego dokładne umiejscowienie w obrębie komórki jest zależne od jednostki taksonomicznej, do której zalicza się dany organizm. W obecnie dominującym sposobie klasyfikacji organizmów, zaproponowanym w oparciu o badania genetyczne przez Woesego [204], wyróżnia się trzy główne jednostki taksonomiczne, tzw. *domeny*: *archeony*, *bakterie* oraz *eukarionty*, które obejmują wszystkie organizmy żywe, wyłączając wirusy. DNA archeonów i bakterii występuje w cytoplazmie budującej komórkę, natomiast u eukariontów zawarte jest w jądrze komórkowym a także, w mniejszych ilościach, w mitochondriach oraz — w przypadku roślin — w chloroplastach.

Ze względu na zróżnicowanie ról pełnionych przez RNA wyróżnia się kilka jego rodzajów. Należą do nich m.in. RNA *matrycowe* (mRNA), *transportujące* (tRNA) i *rybosomalne* (rRNA). mRNA pełni funkcję matrycy białka syntezowanego przez komórkę w rybosomach. Aminokwasy budujące to białko są dostarczane z cytoplazmy za pomocą krótkich nici tRNA. rRNA z kolei jest jednym z głównych składników budulcowych rybosomów.

Postacie występowania kwasów nukleinowych u wirusów są różnorodne. W zależności od szczepu nośnikiem informacji genetycznej wirusa może być zarówno

DNA, jak i RNA, w obu przypadkach w postaci jednoniciowej albo dwuniciowej.

2.1.2 Genom

Sposób zorganizowania DNA wewnątrz jądra komórkowego albo cytoplazmy jest zależny od fazy cyklu komórkowego. U eukariontów, podczas *profazy*, DNA podlega kondensacji i w efekcie, podczas *metafazy*, przyjmuje formę struktur zwanych *chromosomami*. Liczba, wielkość oraz kształt chromosomów są cechą gatunkową. Ludzkie DNA (z wyjątkiem DNA mitochondrialnego) jest zorganizowane w 23 pary chromosomów, z czego 22 pary stanowią chromosomy zwane *autosomami*. Pozostała para, nazywana *allosomami* lub *chromosomami płci*, determinuje płeć osobnika — obecność dwóch allosomów oznaczonych X wyróżnia płeć żeńską, a obecność jednego chromosomu X i jednego Y płeć męską. Podobne zasady obowiązują dla pozostałych rozdzielnopłciowych eukariontów. Kompletny zestaw chromosomów organizmu określany jest mianem *kariotypu*. Zaburzenia powyższej organizacji są przyczyną niektórych chorób genetycznych.

Obecność określonej liczby kopii chromosomów w komórce jest cechą gatunkową i nosi nazwę *ploidii*. Gatunki charakteryzujące się pojedynczym, niesparowanym garniturem chromosomowym zaliczamy do gatunków *haploidalnych*, podwójnym — do *diploidalnych*, a posiadające większą liczbę kopii chromosomów — do *poliploidalnych*. U organizmów diploidalnych każdy chromosom w parze pochodzi odpowiednio od jednego i drugiego organizmu rodzicielskiego. Odpowiadające sobie kopie chromosomów (np. w organizmach diploidalnych chromosomy z jednej pary) nazywamy *chromosomami homologicznymi*. Haploidalny zestaw informacji genetycznej nazywamy *genomem*. Genom organizmu, obok uwarunkowań środowiskowych, stanowi główne źródło jego cech fenotypowych. Rozmiary genomów wahają się od kilku tysięcy par zasad u wirusów, poprzez kilka milionów u bakterii i archeowców, po miliardy u eukariontów [101]. Ludzki genom składa się z ok. 3 miliardów par zasad [101], choć istnieją organizmy o genomach sięgających rozmiarem nawet 43 miliardów par zasad [7].

Kodowanie białek

W genomie można wyróżnić fragmenty niosące informację o budowie białek. Fragmenty te nazywane są *genami*. W obrębie genu wyróżnia się odcinki kodujące informację o kolejności aminokwasów budujących syntezowane białko (*eksony*) oraz, obecne głównie u archeonów i eukariontów, fragmenty niekodujące (*introny*). Zestaw genów pojedynczego osobnika nazywamy jego *genotypem*.

Proces tworzenia białka w oparciu o informację genetyczną jest wieloetapowy i w uproszczeniu obejmuje u eukariontów *transkrypcję*, czyli syntezę mRNA będącego kopią (*transkryptem*) kodujących odcinków genu, oraz *translację*, którą stanowi proces syntezy białka. Synteza białka stanowi jedną z form *ekspresji genu*,

której celem jest uwidocznienie cech determinowanych przez dany gen. Ekspresja jest inicjowana na skutek wystąpienia określonych czynników wewnętrznych lub zewnętrznych oddziałujących na komórkę.

Obszary kodujące zbudowane są z trójek nukleotydów zwanych *kodonami*. Jeden kodon determinuje obecność w budowanym białku jednego z dwudziestu aminokwasów. Aminokwasy pełnią funkcję oligomerów białek. Przyporządkowanie poszczególnym kodonom aminokwasów jest określane mianem *kodu genetycznego*. Charakteryzuje się on dużą uniwersalnością, tzn. prawie każdy organizm przeprowadza translację w oparciu o ten sam kod genetyczny.

Rozmiar odcinków kodujących białka w genomie jest zróżnicowany — odcinki kodujące stanowią ok. 1,1% genomu ludzkiego [199] i przeszło 80% genomu bakterii [202]. Pozostałe obszary genomu pełnią różnorodne funkcje, związane m.in. z regulacją ekspresji. Dokładna rola dużej części genomu pozostaje dotąd nieznana [70].

Zmienność genetyczna

W skład cyklu życiowego komórek organizmu wchodzą etapy, które obejmują proces powielenia (*replikacji*) DNA. Powielenie DNA umożliwia podział komórki i przekazanie do komórki potomnej właściwego materiału genetycznego. W związku z niedoskonałością procesu replikacji, a także w wyniku oddziaływania zewnętrznych czynników zwanych *mutagenami*, informacja genetyczna może ulec zmianie nazywanej *mutacją*. Mutacje mogą charakteryzować się rozmaitymi własnościami. Zakres zmian jakie mogą wywołać w genomie jest bardzo różny — od zamiany pojedynczej pary zasad na inną (*mutacja punktowa*) po przeniesienie (*transpozycję*) dużego fragmentu genomu w inne miejsce lub znaczne zmiany struktury chromosomów. Wpływ mutacji na cechy fenotypowe może być obojętny, niekorzystny, a w rzadkich przypadkach korzystny dla komórki lub organizmu potomnego. Do osobnej grupy zmian zalicza się wymianę całych fragmentów genomów, nazywaną *rekombinacją*. U eukariontów rekombinacja może przyjmować rozmaite formy, do których zaliczyć należy w szczególności niezależny wybór (*segregację*) chromosomów z pary, które zostaną przekazane do komórki rozrodczej lub zjawisko *crossing-over*, które polega na wymianie — zazwyczaj sparowanych — fragmentów chromosomów.

Mutacje i rekombinacje stanowią jedną z głównych przyczyn odmienności DNA organizmów reprezentujących różne jednostki taksonomiczne, co z kolei jest źródłem różnorodności ich fenotypów. Jednakże odmienność DNA jest charakterystyczna także dla przedstawicieli tego samego gatunku i określana jest jako wewnątrzgatunkowa *zmienność genetyczna*. Poszczególne różnice (*warianty*) genomów wynikające ze zmienności wewnątrzgatunkowej mogą mieć różny charakter, jednak dominują wśród nich *polimorfizmy pojedynczego nukleotydu* (ang. *single nucleotide polymorphism — SNP*), które w genomie ludzkim obejmują

ok. 90% różnic [198]. Pojedynczy polimorfizm jest wariantem jednej pary zasad w genomie występującym u wybranych osobników. Innym rodzajem są warianty typu *krótki indel*, które obejmują wstawienie nowego krótkiego odcinka kwasu nukleinowego do genomu lub brak takiego odcinka w genomie. Obecność różnic DNA w obrębie określonego genu jest przyczyną występowania różnych jego wariantów nazywanych *allelami*.

Przedstawiony powyżej przepływ informacji w komórce, tj. replikacja DNA, transkrypcja DNA do RNA, a następnie translacja RNA do białka nazywany jest *centralnym dogmatem biologii molekularnej*. Zasada ta w określonych sytuacjach dopuszcza także *odwrotną transkrypcję* polegającą na uzyskaniu cząsteczki *DNA komplementarnego* (cDNA) w oparciu o nić RNA, a także replikację RNA. Zjawisko uzyskiwania DNA lub RNA w oparciu o strukturę białka nie jest dopuszczalne. Za proces syntezy komplementarnej nici DNA lub RNA w oparciu o niesparowaną nić DNA oraz wolne nukleotydy są odpowiedzialne enzymy zaliczane do grup odpowiednio *polimeraz DNA* i *polimeraz RNA*. Polimerazy zazwyczaj mogą rozpocząć syntezę jedynie w miejscu zakończenia dwuniciowego fragmentu cząsteczki. Z tego powodu uprzednio do nici matrycowej jest dołączany krótki, komplementarny fragment nazywany *starterem*, który stanowi punkt inicjacji syntezy.

Substratami reakcji syntezy nici DNA są wolne nukleotydy, których grupa fosforanowa składa się z trzech atomów fosforu. W wyniku połączenia wolnego nukleotydu z nicią następuje uwolnienie grupy difosforanowej (pirofosforanowej).

Analiza genomów jest obszarem zainteresowań działu biologii molekularnej zwanej *genomiką*.

2.2 Metody odczytu sekwencji kwasów nukleinowych

W 1944 roku zostały przedstawione dowody wskazujące, że nośnikiem informacji o cechach komórek nie są białka, lecz kwasy nukleinowe [32]. Odkrycie to zainicjowało podjęcie ogromnej liczby prac mających na celu poznanie budowy i funkcji kwasów nukleinowych. Szczególnie istotnym elementem tych badań było określenie ich pierwszorzędowej struktury, czyli sekwencji budujących je nukleotydów. Takie zadanie nosi nazwę *sekwencjonowania* kwasu nukleinowego. W latach 60. i 70. XX wieku kilka grup badawczych zaproponowało różne metody sekwencjonowania DNA i RNA oraz udowodniło ich skuteczność prezentując pierwsze odczytane sekwencje [95].

Opracowanie technik sekwencjonowania było motywowane dostarczeniem nowych narzędzi poznania procesów zachodzących w komórkach. Techniki sekwencjonowania miały pomóc w określeniu ról poszczególnych fragmentów genomów, dokładnym wyjaśnieniu zasad rządzących dziedzicznością, a w konsekwencji doprowadzić do poznania mechanizmów translacji i syntezy białek [77], zbadania

fizjologii komórek zainfekowanych wirusami [78], a w dużej mierze także poznania podłoża i opracowania metod leczenia wielu chorób.

2.2.1 Techniki pomocnicze

Pierwsze skuteczne metody sekwencjonowania były ukierunkowane na poznanie sekwencji RNA. Wynikało to z mniejszych trudności technicznych — nici RNA są cząsteczkami znacznie krótszymi oraz zwykle nie posiadają nici komplementarnej. Ponadto, dzięki dostępności łatwych w hodowli organizmów modelowych opartych na RNA, istniała możliwość uzyskania dużej ilości odpowiedniego materiału genetycznego [95]. Zasada działania różnych, wczesnych metod sekwencjonowania zazwyczaj opierała się na podziale cząsteczki na bardzo krótkie fragmenty. Podział taki był najczęściej przeprowadzany przy pomocy substancji zwanych *nukleazami*, które poprzez rozerwanie wiązań fosfodiestrowych przerywają nić kwasu. Niektóre nukleazy charakteryzują się zdolnością detekcji określonego ciągu nukleotydów i przerwania nici w miejscu jego wystąpienia, co można wykorzystać jako dodatkową informację o składzie cząsteczki. Uzyskane fragmenty poddawane były zwykle separacji przy pomocy technik takich jak *elektroforeza*, *chromatografia* lub *jonoforeza*. Wykorzystują one różne własności fizyczne składników analizowanej mieszaniny w celu rozdzielenia jej składników. Separacja fragmentów pozwala na identyfikację ich własności fizycznych, a w konsekwencji określenie ich budowy.

Elektroforeza, chromatografia i jonoforeza wykorzystują własności substancji mające wpływ na szybkość jej przemieszczania się w określonych warunkach. W elektroforezie i jonoforezie tymi warunkami jest silne pole elektryczne, w którym analizowana mieszanina zostaje umieszczona. Z kolei w chromatografii mieszanina jest przepuszczana przez tzw. *złoże*. Po określonym czasie wykonuje się obserwację drogi przemieszczenia określonych frakcji. Jej długość jest podstawą do wyciągnięcia wniosków o ich właściwościach fizycznych takich jak np. masa, co z kolei daje możliwość określenia składu frakcji i całej mieszaniny.

Niektóre spośród metod opierały się na radioaktywnym znakowaniu nukleotydów. Wprowadzenie do nich atomów promieniotwórczego pierwiastka umożliwiało po separacji określenie ich lokalizacji przy pomocy *autoradiografii*, która jest techniką wykorzystującą zjawisko naświetlenia materiału światłoczułego w miejscu obecności substancji radioaktywnej.

Dodatkowo często wykorzystywanym zjawiskiem była *denaturacja DNA*, czyli efekt odwracalnego przerywania wiązań wodorowych między nukleotydami komplementarnymi, prowadzący do rozdzielenia nici. Denaturacja może zachodzić m.in. na skutek oddziaływania podwyższonej temperatury lub bardzo kwaśnego albo zasadowego środowiska.

2.2.2 Sekwencjonowanie RNA

Za pierwszą udaną próbę poznania sekwencji kwasu nukleinowego można uznać odczytanie w 1965 roku przez zespół Roberta Holleya sekwencji nici tRNA odpowiedzialnej za transport aminokwasu alaniny [102]. Nici tego typu posiadają długość 77 nukleotydów. Holley wyekstrahował je z komórek drożdży piekarniczych. W pierwszej kolejności zostały one podzielone na krótsze odcinki, a następnie poddane separacji. Identyfikacja składu odcinków długości 2 nukleotydów była możliwa poprzez obserwację ich własności w procesach chromatografii i elektroforezy. Identyfikacja budowy odcinków dłuższych oraz ustalenie względnych pozycji poszczególnych fragmentów wymagały uzyskania odcinków o innych długościach oraz ponawiania procesu identyfikacji. Pełny proces odczytu sekwencji trwał ok. 2,5 roku. Za swoje prace nad poznaniem kodu genetycznego oraz określenie roli RNA w procesie syntezy białka w 1968 roku Holley został uhonorowany Nagrodą Nobla w dziedzinie fizjologii lub medycyny.

Inna metoda sekwencjonowania RNA została zaproponowana w tym samym roku przez zespół Fredericka Sangera i polegała na radioaktywnym znakowaniu nukleotydów [181]. Proces znakowania opierał się na hodowli organizmów w warunkach obecności radioaktywnego fosforu ^{32}P, który następnie wchodził w skład nukleotydów wyhodowanych osobników. Oznakowane cząsteczki RNA poddano separacji, a ich detekcja została przeprowadzona w oparciu o autoradiografię.

W kolejnych latach różne zespoły wykorzystały metodę opracowaną przez Sangera do określenia sekwencji szeregu różnych cząsteczek tRNA oraz rRNA [95], jednak szczególnie istotnym przypadkiem jej zastosowania było przeprowadzenie w 1972 roku przez zespół Waltera Fiersa sekwencjonowania jednego pełnego genu RNA bakteriofaga MS2 [77]. Gen ten jest odpowiedzialny za kodowanie białka budującego zewnętrzną powłokę (*kapsyd*) wirusa. Kontynuacja prac zaowocowała w 1976 roku określeniem sekwencji liczącego 3569 nukleotydów genomu tego bakteriofaga [78], co stanowiło pierwszy przypadek poznania pełnego genomu RNA.

2.2.3 Sekwencjonowanie DNA

Wysiłki nad opracowaniem metody sekwencjonowania DNA były podjęte m.in. w latach 1968–1974 przez zespół Raya Wu [207]. Jego pierwsze osiągnięcie stanowiło określenie zawartości nukleotydów w lepkim końcu obecnym w DNA bakteriofaga λ [206], jednak bez wyznaczenia ich kolejności. Dalszy rozwój metody pozwolił na przeprowadzenie m.in. pełnego sekwencjonowania pierwszych ośmiu nukleotydów tego fragmentu [205]. Metoda Wu polegała na wykorzystaniu nukleotydów oznakowanych radioaktywnie przy pomocy fosforu ^{32}P, które na zasadzie komplementarności, w obecności polimerazy DNA, były dołączane do jednoniciowego fragmentu lepkiego końca DNA. Tym samym metoda ta nie pozwalała na

odczyt sekwencji DNA dwuniciowego.

W 1973 roku Walter Gilbert i Allan Maxam [82] odczytali sekwencję 24 par zasad fragmentu *operonu laktozowego*, będącego częścią genomu pałeczki okrężnicy. Zastosowana metoda polegała na znakowaniu fosforem ^{32}P ostatniego nukleotydu na końcu $5'$ sekwencjonowanych nici. Następnie, od strony końca $3'$, oddzielano od nici oligomery różnej długości. Pozostałe, zawierające oznakowany nukleotyd fragmenty były poddawane separacji w dwóch wymiarach oraz detekcji przy pomocy autoradiografii — przemieszczenie fragmentów w jednym z wymiarów separowało fragmenty o różnej długości, natomiast przemieszczenie w drugim separowało różne nukleotydy obecne na ostatniej pozycji fragmentu.

W 1975 roku Frederick Sanger i Alan Coulson zaproponowali *metodę plus i minus* [180]. Jej pierwszy etap polegał na denaturacji wejściowych cząsteczek DNA oraz syntezie komplementarnych do nich oligomerów różnej długości. Jeden spośród wykorzystywanych w syntezie nukleotydów był znakowany fosforem ^{32}P. Uzyskane nici były wykorzystywane w jednej z kolejnych faz. Faza „minus" polegała na kontynuacji syntezy nici komplementarnej, ale przy nieobecności nukleotydów opartych na jednej, wybranej zasadzie (nukleotydów jednego typu). W efekcie uzyskiwano oligomery, spośród których żaden nie kończył się nukleotydem tego typu. W fazie „plus" syntezę przeprowadzano w podobny sposób, jednak tym razem obecne były wyłącznie nukleotydy jednego typu, zatem uzyskiwano cząsteczki kończące się wyłącznie tym nukleotydem. Obie fazy wykonywano niezależnie dla każdego z czterech typów nukleotydów, a uzyskane zestawy fragmentów były poddawane separacji i obserwowane przy pomocy autoradiografii. Sekwencja DNA mogła być określona w oparciu o różnice w rezultatach obu faz. Metoda ta pozwoliła na początku 1977 roku na wykonanie sekwencjonowania genomu DNA bakteriofaga ΦX174 [182], który stanowił pierwszy w pełni poznany genom DNA.

Kolejna metoda została opracowana przez zespół Gilberta i Maxama w 1977 roku [145]. Wejściowe fragmenty DNA, po przeprowadzeniu denaturacji, były znakowane radioaktywnie poprzez dołączenie dodatkowego nukleotydu na końcu $5'$. Tak przygotowane cząsteczki były dzielone na cztery porcje. Każda z nich była poddawana działaniu jednej z odpowiednich substancji chemicznych, które przerywały nić odpowiednio po napotkaniu określonego typu nukleotydu: G, G+A (tj. G, a czasami A), C oraz C+T. Uzyskane w ten sposób oligomery były poddawane separacji i obserwacji przy pomocy autoradiografii.

W tym samym roku zespół Sangera przedstawił rozwinięcie swojej metody [183], nazywane *metodą dideoksy*. Wejściowe, zdenaturowane fragmenty rozdzielano na cztery grupy. Każdą z nich umieszczano w roztworze z wolnymi nukleotydami oraz z niewielką ilością nukleotydów jednego typu, które dodatkowo były oznakowane radioaktywnie i pozbawione grupy hydroksylowej na pozycji $3'$. Tak zmodyfikowane nukleotydy nie mogą tworzyć wiązań fosfodiestrowych z kolejny-

mi nukleotydami i powodują przerwanie syntezy przez polimerazę DNA. Moment, w którym taki terminalny nukleotyd zostawał dołączony był przypadkowy, stąd w obrębie każdego roztworu uzyskiwano odcinki różnej długości. Następnie odcinki te były poddawane separacji i obserwowane przy pomocy autoradiografii. Pozycja cząstek wyznaczała ich długość, natomiast typ ostatniego nukleotydu wynikał z typu zmodyfikowanych nukleotydów obecnych w danym roztworze. Za swoje odkrycia Sanger wraz z Gilbertem zostali w 1980 roku uhonorowani Nagrodą Nobla w dziedzinie chemii.

2.2.4 Udoskonalenia metody dideoksy

Metoda dideoksy, jako sprawniejsza i w mniejszym stopniu oparta na szkodliwej promieniotwórczości niż metoda Gilberta i Maxama [136], przez blisko 30 lat pozostawała podstawowym narzędziem sekwencjonowania DNA. W tym okresie została wielokrotnie udoskonalona, np. poprzez wprowadzenie fluorescencyjnego znakowania terminalnych nukleotydów czterema barwnikami, zastosowanie *elektroforezy kapilarnej* oraz opracowanie nowych odczynników [155]. Ulepszenia te umożliwiły zwiększenie szybkości odczytu fragmentów, przeprowadzenie reakcji w jednym roztworze zamiast w czterech, obniżenie kosztów czy rezygnację z wykorzystania radioaktywnych odczynników. W 1987 roku laboratorium Leroya Hooda, w oparciu o powyższe udoskonalenia, udostępniło urządzenie sekwencjonujące (*sekwenator*) AB370 [136], które umożliwiało częściową automatyzację procesu sekwencjonowania poprzez przeprowadzanie odczytu sekwencji przy użyciu komputera. Urządzenie to zostało skomercjalizowane przez przedsiębiorstwo Applied Biosystems. Dla odróżnienia od nowszych rozwiązań metoda dideoksy oraz jej modyfikacje są określane mianem technik sekwencjonowania *pierwszej generacji* (ang. *first-generation sequencing — FGS*).

2.3 Współczesne narzędzia i metody sekwencjonowania DNA

2.3.1 Ograniczenia metod pierwszej generacji

Metoda dideoksy pozwala na sekwencjonowanie jedynie krótkich fragmentów nici DNA nazywanych *odczytami* (ang. *reads*). Odczyty mają długość do kilkuset par zasad i są znacznie krótsze nie tylko od najprostszych genomów, ale także od wielu ich interesujących obszarów, takich jak geny czy tzw. *powtórzenia tandemowe*. Częściowym rozwiązaniem tego ograniczenia jest zastosowanie techniki *sekwencjonowania śrutowego* (ang. *shotgun-sequencing*) [166], która polega na namnożeniu (*amplifikacji*) sekwencjonowanych nici oraz wielokrotnym przeprowadzeniu sekwencjonowania fragmentów pochodzących z przypadkowych miejsc analizowanego DNA. Proces ten trwa do momentu osiągnięcia docelowej *głębo-*

kości sekwencjonowania (ang. *sequencing depth*), która jest zdefiniowana jako stosunek sumy długości wszystkich odczytów do długości genomu. Odpowiednio duża głębokość sekwencjonowania i wynikająca z niej redundancja danych w wielu przypadkach umożliwia znalezienie nakładających się fragmentów odczytów, co pozwala na określenie ich względnego położenia w nici. Sposób ten w niektórych przypadkach pozwala na poznanie postaci fragmentów nici znacznie dłuższych niż odczyt, często jest jednak niewystarczający. Przykładowo, nie umożliwia dokładnego określenia dłuższych sekwencji składających się z krótkich, powtarzających się motywów.

Do pozostałych trudności związanych z wykorzystaniem technik pierwszej generacji należy ich niska *przepustowość* (ang. *throughput*), czyli liczba nukleotydów odczytywanych w jednostce czasu. Biorąc pod uwagę konieczność dostarczenia odpowiednio dużej głębokości sekwencjonowania śrutowego oraz znaczny rozmiar wielu genomów, tempo sekwencjonowania metody dideoksy wynoszące 80 kz dla platformy 3730xl w jednym, trwającym ok. 3 godzin uruchomieniu urządzenia jest zdecydowanie niewystarczające. Istotną kwestią jest także wysoki koszt procesu sekwencjonowania [164].

Pomimo występowania powyższych oraz innych, niewymienionych wad technik pierwszej generacji, w 1990 roku zainicjowano Projekt Poznania Ludzkiego Genomu (ang. *Human Genome Project*). W jego ramach w 2001 roku dwie konkurujące instytucje wstępnie przedstawiły postać ludzkiego genomu [124, 199]. Prace te zajęły 11 lat, a w kolejnych latach ich wyniki były uzupełniane [51].

2.3.2 Techniki sekwencjonowania drugiej generacji

Kwestia niedoskonałości technik pierwszej generacji umotywowała opracowanie nowych metod i narzędzi sekwencjonowania. Są one obecnie zaliczane do grupy technik sekwencjonowania *drugiej generacji* albo *nowej generacji* (ang. *second-generation sequencing — SGS*, *next-generation sequencing — NGS*).

Zasada działania większości technik NGS polega na syntezie nici komplementarnych w oparciu o zdenaturowane nici wejściowe oraz obserwacji procesu ich tworzenia. Taka idea nosi nazwę *sekwencjonowania przez syntezę* (ang. *sequencing by synthesis*). Oparta na niej była także technika dideoksy [95].

Większość technik NGS wykorzystuje takie narzędzia i zjawiska jak *reakcja łańcuchowa polimerazy* (ang. *polymerase chain reaction — PCR*), *ligacja* czy *hybrydyzacja*. Ze względu na to, że metody NGS są stosowane do bezpośredniego sekwencjonowania DNA, a nie RNA, w niniejszym podrozdziale zawarto opis zjawisk wyłącznie w tym kontekście.

Metoda PCR jest techniką amplifikacji materiału DNA. Polega ona na cyklicznym ogrzewaniu i ochładzaniu roztworu zawierającego nici DNA, niesparowane pojedyncze nukleotydy oraz polimerazę DNA. W wyniku zwiększenia temperatury DNA ulega denaturacji, a następnie, w efekcie jej obniżenia, przy udziale

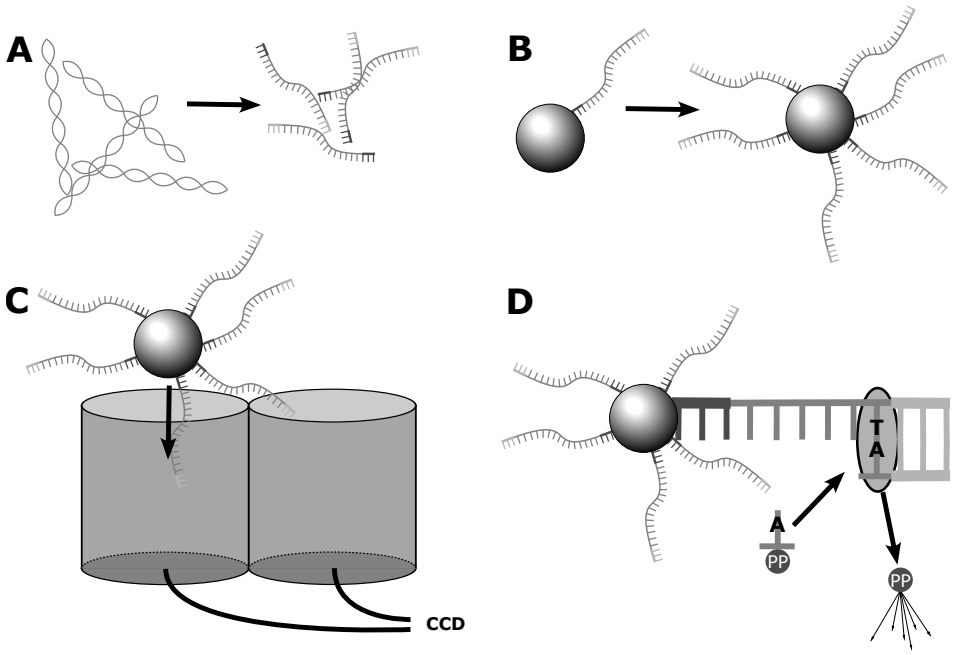
polimerazy DNA następuje na obu niciach synteza ich nici komplementarnych. Ligacja jest zjawiskiem łączenia końcówek dwóch sparowanych nici DNA. Jest ona stosowana zazwyczaj dla cząstek DNA posiadających komplementarne lepkie końce i powoduje powstanie jednej, dłuższej cząstki. Odmiennym zjawiskiem jest hybrydyzacja, która polega na uzyskaniu cząstki sparowanej poprzez połączenie dwóch komplementarnych cząstek niesparowanych.

Jedną z cech charakterystycznych współczesnych metod NGS jest przeprowadzanie jednoczesnego sekwencjonowania wielu (w niektórych sekwenatorach nawet milionów [193]) różnych fragmentów DNA wejściowego. Pozwala to na osiągnięcie bardzo wysokiej przepustowości pomimo znaczącego czasu koniecznego do przeprowadzenia pełnego cyklu odczytu jednego fragmentu. Techniki NGS charakteryzujące się dużym stopniem współbieżności zalicza się do grupy *technik wysokoprzepustowych* (ang. *high-throughput sequencing*) [30, 171].

Do istotnych zalet niektórych metod NGS należy możliwość generowania *odczytów sparowanych* (ang. *paired-end reads — PE*, *mate-pair reads — MP*). Technika ta pozwala na uzyskiwanie par odczytów, których wzajemna odległość w genomie jest w przybliżeniu znana. Istniejące podejścia do tego zagadnienia — PE i MP — różnią się sposobem przygotowania DNA wejściowego oraz odległościami między odczytami. Uzyskanie odczytów typu PE polega na sekwencjonowaniu jednego fragmentu DNA dwukrotnie: od jednego i od drugiego końca, co pozwala na określenie postaci odcinków DNA odległych o co najwyżej kilkaset par zasad. W przypadku podejścia typu MP przygotowuje się fragmenty długości kilku tysięcy par zasad. Następnie ich końce są łączone w celu uzyskania cząsteczek kolistych. Z miejsca połączenia wycina się krótszy fragment, który zostaje poddany sekwencjonowania w sposób analogiczny jak w podejściu PE. W ten sposób uzyskuje się parę odczytów odcinków oddalonych o kilka tysięcy par zasad. Do wad technik NGS należy zaliczyć możliwość generowania tylko bardzo krótkich odczytów (często krótszych niż w metodzie dideoksy) oraz ich stosunkowo niską jakość, tzn. obecność licznych różnic między odczytami a rzeczywistymi sekwencjami nukleotydów.

Techniki NGS oparte są na licznych odkryciach, które nastąpiły w ciągu ostatnich lat [30]. Pozwoliły one na opracowanie szeregu różnych rozwiązań. Kilka konkurencyjnych przedsiębiorstw wprowadziło na rynek wiele modeli urządzeń przeprowadzających sekwencjonowanie w sposób maksymalnie zautomatyzowany. Poniżej zawarto ogólny opis zasady działania i najważniejsze własności najpopularniejszych technik sekwencjonowania drugiej generacji.

Proces sekwencjonowania rozpoczyna się od uzyskania próbki materiału genetycznego poprzez jego izolację z organizmu lub środowiska. Materiał ten jest wykorzystywany do utworzenia *biblioteki DNA* (ang. *DNA library*) stanowiącej zestaw fragmentów wejściowego materiału. Fragmenty te muszą posiadać odpowiednią długość zależną m.in. od techniki sekwencjonowania, zastosowania albo



Rysunek 2.1: Zasada działania sekwenatora 454 GenomeSequencer FLX [30]

konieczności uzyskania odczytów sparowanych. Fragmenty są poddawane dalszej obróbce, która może polegać m.in. na naprawie końcówek, tworzeniu lepkich końców, łączeniu ze znanymi fragmentami DNA nazywanymi *adapterami* (ang. *adaptor*) czy amplifikacji np. przy pomocy techniki PCR [94]. Uzyskana biblioteka stanowi materiał wejściowy urządzenia sekwencjonującego.

Roche 454

W 2005 roku przedsiębiorstwo 454 Life Sciences (niedługo później zakupione przez Roche Diagnostics) udostępniło komercyjnie pierwsze urządzenie sekwencjonujące drugiej generacji — GenomeSequencer FLX [30]. Jego zasada działania oparta jest na technice sekwencjonowania przez syntezę. Sekwencjonowanie 454, ze względu na wykorzystanie własności pirofosforanów, jest nazywana *pirosekwencjonowaniem* (ang. *pyrosequencing*).

Na rys. 2.1 przedstawiono schemat działania sekwenatora GenomeSequencer FLX. Sekwencjonowane cząsteczki DNA są dzielone na fragmenty długości kilkuset par zasad. Następnie są one poddawane ligacji z adapterami (A) oraz poddawane denaturacji. Adaptery przyłączają się do indywidualnych *kulek* (ang. *bead*) pełniących funkcję podłoża. Unieruchomione w ten sposób nici są poddawane amplifikacji przy pomocy techniki PCR (B). Amplifikacja ma na celu zwiększenie intensywności emitowanego podczas procesu sekwencjonowania światła i ułatwienie jego detekcji. W rezultacie każda kulka zawiera do kilku milionów przytwierdzonych do niej kopii sekwencjonowanego odcinka. W dalszej kolejności każda

z kulek jest umieszczana w osobnym dołku mikropłytki przy końcówce światłowodu połączonego z matrycą CCD (C). Następnie do dołka wprowadzany jest zestaw cząstek polimerazy DNA oraz starterów. Proces odczytu sekwencji jest przeprowadzany poprzez cykliczne dostarczanie kolejnych wolnych nukleotydów, czego efektem jest postępująca synteza komplementarnych nici sekwencjonowanych odcinków. W momencie powstania wiązania wodorowego z dostarczonych nukleotydów uwalniane są grupy pirofosforanowe (PP), które w wyniku obecności dodatkowych substancji powodują emisję światła, wykrywanego przez matrycę CCD (D). Jeżeli aktualnie syntezowany fragment nici składa się z wielu nukleotydów opartych na tej samej zasadzie, wówczas synteza odbywa się w jednym cyklu. Powoduje to zwiększenie natężenia światła proporcjonalnie do długości jednolitego fragmentu. Po usunięciu nieprzyłączonych nukleotydów proces jest powtarzany dla nukleotydu kolejnego typu.

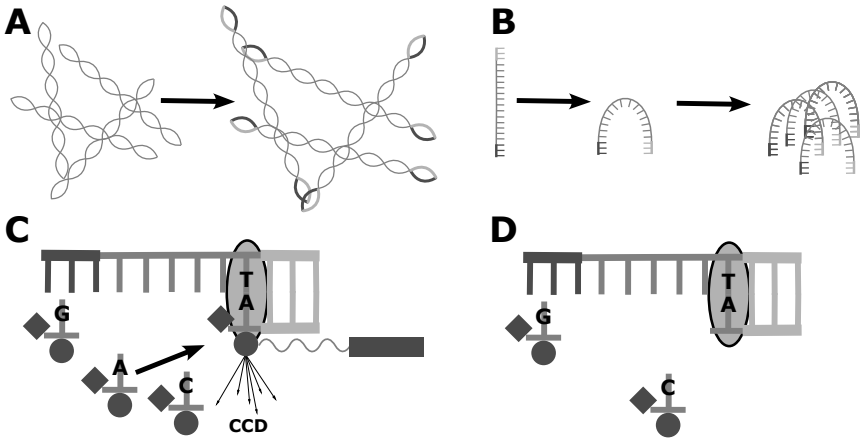
Do zalet pirosekwencjonowania należy duża, w momencie wprowadzenia techniki, przepustowość (0,7 Gz w trakcie jednego, trwającego dobę uruchomienia urządzenia w wersji FLX Titanium [136]) oraz możliwość generowania odczytów długości większej od wielu innych metod NGS. Z drugiej strony technika ta charakteryzuje się wysokim kosztem procesu sekwencjonowania oraz wrażliwością na występowanie w sekwencjonowanych fragmentach dłuższych *homopolimerów*, czyli odcinków składających się z nukleotydów tego samego typu [30]. Określenie długości takiego jednolitego fragmentu w oparciu o obserwację intensywności emitowanego światła nie jest procesem dostatecznie dokładnym [148].

Obecnie sekwenatory 454 zostały wycofane z produkcji, jednak cały czas dostępne są niezbędne odczynniki, co pozwala na pracę istniejących urządzeń [1, 2].

Illumina

Konkurencyjne rozwiązanie — sekwenator Genome Analyzer — zostało udostępnione w 2006 roku przez przedsiębiorstwo Solexa (które z kolei zostało w 2007 roku zakupione przez firmę Illumina). Sekwenatory marki Illumina także są oparte na idei sekwencjonowania przez syntezę.

Na rys. 2.2 przedstawiono zasadę działania sekwenatora Genome Analyzer. Analizowane, sparowane fragmenty są z obu końców poddawane ligacji z adapterami (A). W dalszej kolejności przeprowadzana jest ich denaturacja oraz przytwierdzenie obu końców do podłoża. W tej formie fragmenty są poddawane amplifikacji przy pomocy techniki PCR, która trwa do momentu uzyskania około 1000 kopii nici zgromadzonych w grupach (B). Następnie adaptery jednego z końców zostają odłączone od podłoża, stając się jednocześnie punktami zaczepienia starterów przy budowie drugiej nici. Roztwór jest uzupełniany starterami, polimerazą DNA oraz zestawem wolnych nukleotydów. Dodawane nukleotydy są, w zależności od typu, oznakowane jednym z czterech fluoroforów. Fluorofor jest połączony z zasadą azotową, natomiast na pozycji $3'$ nukleotydu jest obecny



Rysunek 2.2: Zasada działania sekwenatora Illumina na przykładzie Genome Analyzer [30]

usuwalny terminator (ang. *reversible terminator*), którego zadaniem jest tymczasowe uniemożliwienie utworzenia wiązania z kolejnym nukleotydem [44]. Na skutek działania polimerazy następuje synteza nici komplementarnej, jednak ze względu na obecność terminatora po dołączeniu jednego nukleotydu jest ona zatrzymywana. Próbka jest oświetlana laserem, co inicjuje emisję przez fluorofor światła w kolorze odpowiadającym typowi nukleotydu (C) — nukleotydy A i C są inicjowane laserem czerwonym, a G i T zielonym [185]. Po zarejestrowaniu światła przez matrycę CCD terminatory i fluorofory są usuwane (D) i proces jest powtarzany. W wyniku uzyskuje się serię obrazów zawierających zarejestrowane punkty światła, z których każdy reprezentuje nukleotyd osobnego odczytu.

W kolejnych latach przedstawione zostały udoskonalone modele sekwenatorów, charakteryzujące się wyższą przepustowością, lepszą dokładnością, większą długością odczytów, a także wyspecjalizowanym zastosowaniem. Należą do nich Genome Analyzer II, HiSeq, MiSeq i NovaSeq. Obecnie Illumina nie oferuje już sekwenatorów serii Genome Analyzer i Genome Analyzer II.

Technologia Illumina, obejmując 80–90% rynku, jest dominującą pod względem popularności techniką sekwencjonowania [44]. Do zalet współczesnych wersji jej sekwenatorów można zaliczyć niski koszt sekwencjonowania [164] i bardzo wysoką przepustowość wynoszącą do 6 Tz w ciągu 44 godzin dla urządzenia NovaSeq 6000 System [3]. Jednocześnie należy zwrócić uwagę na częste przypadki nieprawidłowego rozróżnienia nukleotydów A i C oraz G i T spowodowane użyciem tego samego koloru światła lasera do detekcji nukleotydów z tych par [185], a także produkcję odczytów niewielkiej długości (współcześnie 150 pz–300 pz w zależności od typu urządzenia [3]).

Applied Biosystems

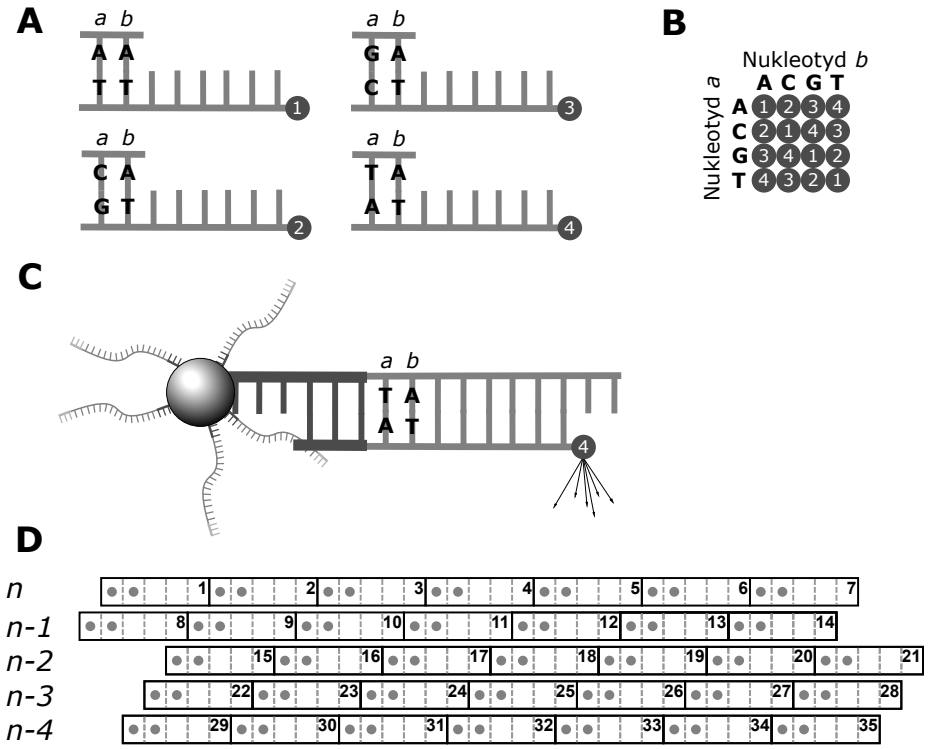
Odmienna technika została przedstawiona w 2007 roku przez przedsiębiorstwo Applied Biosystems (które wkrótce potem weszło w skład Life Technologies). Została ona udostępniona pod postacią sekwenatora ABI SOLiD. Technika ta jest klasyfikowana do grupy *sekwencjonowania przez ligację* (ang. *sequencing-by-ligation*) [30].

Na rys. 2.3 przedstawiono zasadę działania sekwenatora ABI SOLiD. Pierwszy etap procesu stanowi amplifikacja fragmentów DNA w sposób podobny jak w technice 454, tj. poprzez dołączenie ich do niewielkiej kulki i przeprowadzenie namnażania przy pomocy PCR. Następnie kulki z doczepionymi klonami fragmentu są umieszczane na płaskiej powierzchni razem ze starterem oraz zestawem 8-merów. Pierwszy i drugi nukleotyd każdego 8-meru (licząc od końca $3'$; w starszych wersjach sekwenatora zasada ta odnosiła się do czwartego i piątego nukleotydu) posiadają znaną sekwencję, która jest określana przez fluorescencyjne oznakowanie obecne w trzech ostatnich nukleotydach (A), na rysunku zaznaczone w formie kółka z cyfrą. Po dołączeniu startera długości n oraz właściwego, komplementarnego 8-meru (C) następuje emisja światła wskazująca parę nukleotydów znajdujących się na dwóch pierwszych pozycjach 8-meru. 16 możliwych par jest kodowane przy pomocy czterech kolorów światła (B). W efekcie cztery różne pary są kodowane tym samym kolorem, a tym samym informacja o budowie pary jest niepełna. Następnie trzy ostatnie spośród dołączonych nukleotydów, obejmujących także oznakowanie fluorescencyjne, są usuwane. Proces jest powtarzany kilkukrotnie, odpowiednio dla spodziewanej długości odczytu (D). W celu określenia dokładnej i pełnej postaci fragmentu w kolejnym etapie następuje usunięcie zbudowanej nici oraz rozpoczęcie jej budowy od początku, jednak przy zastosowaniu startera krótszego o jeden nukleotyd. W rezultacie następuje odczytanie par obecnych w innych miejscach nici. Po pięciokrotnym przeprowadzeniu budowy nici komplementarnej z pomocą coraz krótszych starterów, zebrane informacje umożliwiają określenie postaci całego fragmentu. Na rysunku kolejne procesy dołączania 5-merów są oznaczone liczbami.

Obecnie technika ABI SOLiD umożliwia uzyskiwanie sparowanych odczytów długości 75 pz oraz 35 pz [40], zatem wynikowe odczyty są bardzo krótkie. Przepustowość sekwenatorów sięga 2,7 Gz podczas jednego uruchomienia sekwenatora, jednak czas całego procesu sekwencjonowania sięga 10 dni [40]. Istotną zaletą jest bardzo dobra jakość uzyskiwanych danych spowodowana dwukrotnym odczytem każdego nukleotydu [30, 148] oraz stosunkowo niski koszt sekwencjonowania [90].

Ion Torrent

Technika Ion Torrent została udostępniona w 2010 roku przez przedsiębiorstwo Life Technologies (obecnie będące marką Thermo Fisher Scientific) pod postacią sekwenatora Ion PGM [171]. Jego zasada działania jest zbliżona do metody 454,



Rysunek 2.3: Zasada działania sekwenatora ABI SOLiD [8]

a istotną różnicą jest sposób detekcji liczby nukleotydów dołączonych do fragmentu. W przypadku Ion Torrent polega on na analizie zmian kwasowości roztworu, będących efektem uwalniania jonów wodorowych w momencie przyłączenia nukleotydów. Analiza jest przeprowadzana przez detektor półprzewodnikowy umieszczony na dnie dołka. W związku z rezygnacją z wykorzystania analizy światła w pracy [184] zaliczono tę technikę do grupy pośredniej między sekwencjonowaniem drugiej a trzeciej generacji.

Rezygnacja z wykorzystania matryc CCD pozwoliła na przyspieszenie procesu odczytu pojedynczego nukleotydu oraz redukcję kosztów [171]. Jednakże zmiana sposobu odczytu nie zażegnała problemu sekwencjonowania homopolimerów — obecność takich fragmentów długości większej niż 6 pz powoduje wzrost podatności na powstawanie błędów [171]. Najnowsze urządzenie sekwencjonujące Ion Torrent, Ion S5, pozwala na uzyskanie odczytów długości 200, 400, a czasem 600 pz przy przepustowości sięgającej 15 Gz w ciągu 2,5 godziny [21].

2.3.3 Techniki sekwencjonowania trzeciej generacji

W ostatnich latach daje się zaobserwować dalszy rozwój technik sekwencjonowania. Nowe rozwiązania charakteryzują się odmienną zasadą działania oraz parametrami rezultatów. Techniki te są klasyfikowane do grupy metod *sekwencjonowania trzeciej generacji* (ang. *third-generation sequencing* — *TGS*). Jako ich

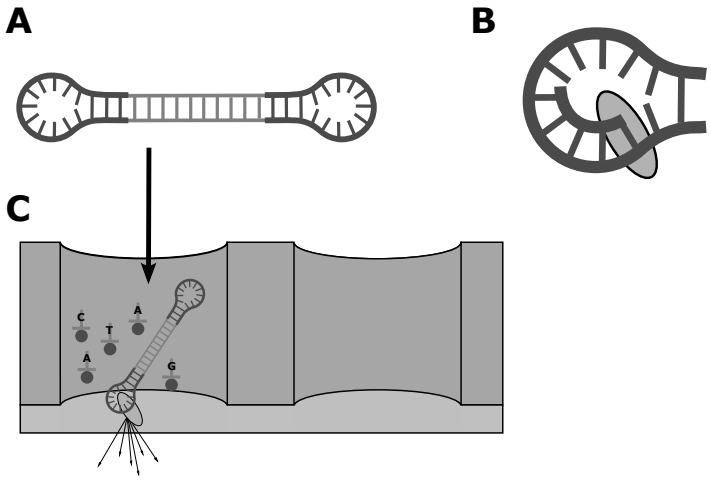
wyróżniające kryterium przyjmuje się możliwość sekwencjonowania pojedynczej cząsteczki DNA, tj. brak potrzeby wstępnej amplifikacji [184]. Takie rozwiązanie jest zaliczane do grupy *sekwencjonowania jednocząsteczkowego* (ang. *single-molecule sequencing*, *SMS*), a jego zaletą jest redukcja kosztów, uproszczenie przygotowania próbki oraz zmniejszenie liczby potencjalnych źródeł błędów poprzez rezygnację z amplifikacji metodą PCR. Kolejnymi wyznacznikami TGS są brak konieczności chwilowego zatrzymywania procesu sekwencjonowania po odczytaniu danego nukleotydu [184], a zatem także dalsze przyspieszenie procesu sekwencjonowania, oraz możliwość uzyskania odczytów długości sięgającej tysięcy par zasad [37]. Znaczna długość odczytów jest istotną zaletą i pozwala na rozwiązanie problemów, dla których odczyty NGS są zbyt krótkie [31].

Pacific Biosciences

Pierwsze urządzenie sekwencjonujące oparte na metodzie klasyfikującej się do grupy TGS, PacBio RS, zostało upublicznione w 2010 roku przez Pacific Biosciences. Zalicza się ono do rozwiązań opartych na *sekwencjonowaniu jednocząsteczkowym czasu rzeczywistego* (ang. *single-molecule-real-time sequencing — SMRT*) [171]. Jego zasada działania została przedstawiona na rys. 2.4. Wejściowe DNA, podzielone na fragmenty długości do kilkudziesięciu tysięcy par zasad, jest łączone z dwoma jednoniciowymi adapterami w taki sposób, aby cząsteczka przyjęła postać cykliczną (A). Następnie cząsteczka zostaje umieszczona w dołku pełniącym funkcję *falowodu* (ang. *zero-mode waveguides — ZMW*), o budowie maksymalnie redukującej powstawanie szumów. Na dnie dołka znajduje się unieruchomiona cząsteczka polimerazy DNA, posiadająca zdolność rozdzielania nici (ang. *strand displacing polymerase*). Proces sekwencjonowania rozpoczyna się w momencie dodania do roztworu wypełniającego dołek wolnych, fluoroscencyjnie oznakowanych nukleotydów, pozwalających na budowę nici komplementarnej (B). W momencie dołączenia nukleotydu następuje emisja światła, którego kolor podlega detekcji (C). W dalszej kolejności oznakowanie fluorescencyjne zostaje usunięte i proces jest powtarzany. Sekwencja nukleotydów jest określana w oparciu o barwę zaobserwowanych impulsów światła.

Interesującą cechą techniki PacBio jest możliwość wielokrotnego sekwencjonowania jednego fragmentu DNA — dzięki cyklicznej budowie cząsteczki polimeraza DNA może po zakończeniu odczytu jednej z nici powtórzyć proces dla jej nici komplementarnej. *Konsensusowa sekwencja* (ang. *circular consensus sequence — CCS*) uzyskanego odczytu może charakteryzować się lepszą jakością. Technika ta jest dostępna tylko dla krótkich fragmentów DNA [172].

Długość odczytów najnowszego sekwenatora PacBio RS II jest zmienna i może sięgać 60 kpz przy średniej wartości 14 kpz [171], co obok braku konieczności amplifikacji wejściowego DNA stanowi istotną zaletę tego systemu. Jednocześnie wysokie stopy błędów uzyskanych sekwencji, wysokie koszty sekwencjonowania



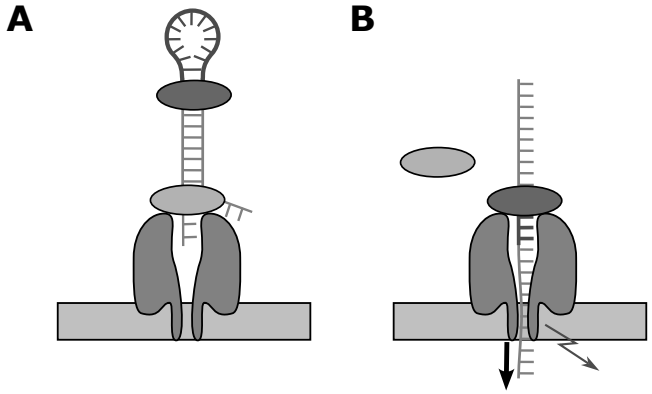
Rysunek 2.4: Zasada działania sekwenatora PacBio [172]

oraz niska przepustowość wynosząca 16 Gz w ciągu 4 godzin [15, 171] ograniczają możliwości jego wykorzystania [171].

Oxford Nanopore Technologies

Jedną z najnowszych technik TGS jest sekwencjonowanie z wykorzystaniem *nanoporów* (ang. *nanopore sequencing*) będących kanałami o nanometrowych średnicach, obecnymi w warstwie tworzywa. Rozwiązanie to zostało wdrożone w urządzeniu MinION przez Oxford Nanopore Technologies w 2014 roku. Na rys. 2.5 przedstawiono zasadę jego działania. Fragment dwuniciowego DNA długości kilkudziesięciu tysięcy par zasad jest wiązany z jednoniciowym, łączącym nici wejściowego DNA adapterem. Dodatkowo, w pobliżu adaptera dołączany jest enzym, zaznaczony na rysunku ciemnym owalem. Przy przeciwległym końcu dołączane są dwa kolejne enzymy (jasny owal), z których jeden wiąże cząsteczkę z podłożem i unieruchamia DNA obok kanału. Zadaniem sąsiedniego enzymu jest sukcesywne przerywanie wiązań wodorowych cząsteczki DNA i przesuwanie jednej z nici przez kanał (A). Przemieszczające się nukleotydy indukują przepływ prądu elektrycznego o natężeniu charakterystycznym dla danego nukleotydu. Pomiar poziomu natężenia umożliwia określenie kolejności zasad. W celu poprawy jakości wyników urządzenie MinION umożliwia sekwencjonowanie obu nici fragmentu. Po przeprowadzeniu odczytu jednej z nich, w wyniku obecności adaptera, możliwe jest wykonanie procesu dla drugiej nici (B) — uzyskane wyniki pozwalają na określenie konsensusowej sekwencji nukleotydów.

Podobnie do metody PacBio, sekwencjonowanie z wykorzystaniem nanoporów nie wymaga wstępnej amplifikacji wejściowego materiału oraz pozwala na uzyskanie bardzo długich odczytów. Typowa długość odczytu wynosi kilkadziesiąt tysięcy par zasad, a połowa łącznej długości sekwencji uzyskanych w ekspe-



Rysunek 2.5: Zasada działania sekwenatora MinION [171]

rymencie sekwencjonowania jest zawarta w odczytach długości co najmniej ok. 100 kpz [14]. Dużą zaletą tej techniki jest bardzo niski koszt i niewielki rozmiar urządzenia sekwencjonującego, z kolei koszt przygotowania próbki jest zbliżony do technik NGS [151]. Przepustowość urządzeń MinION sięga 50 Gz w trakcie jednego, trwającego 72 godzin uruchomienia urządzenia [13]. Wydajność podobnego urządzenia działającego zgodnie z tą techniką, Promethion, sięga 14 Tz w podobnym okresie, wykorzystując jednocześnie 48 płytek do przeprowadzania sekwencjonowania (ang. *flowcell*) [17]. Z drugiej strony odczyty uzyskane techniką sekwencjonowania z wykorzystaniem nanoporów charakteryzują się obecnością bardzo dużej liczby błędów [28].

2.3.4 Błędy sekwencjonowania

Do najważniejszych trudności związanych z wykorzystaniem odczytów metod NGS oraz TGS należy obecność znacznej liczby błędów sekwencjonowania. Błędy mogą mieć charakter systematyczny lub losowy, a ich własności oraz przyczyny być rozmaite i w znacznej mierze zależeć od własności danej techniki.

Potencjalne źródła błędów są obecne już w etapach wstępnego przygotowania próbki, tworzenia biblioteki bądź amplifikacji [123]. Wśród nich szczególnie istotnym problemem są błędy powstające podczas namnażania techniką PCR [184]. Jednakże liczna grupa błędów jest efektem własności konkretnej techniki, przykładowo w metodach 454 i IonTorrent trudności związane z odpowiednio dokładnym pomiarem intensywności światła albo pomiarem napięcia detektora kwasowości mogą powodować błędne określenie długości homopolimerów [30, 171]; w metodzie Illumina częstym zjawiskiem jest błędne rozróżnienie nukleotydów A i C oraz G i T spowodowane wykorzystaniem dwóch kolorów lasera — po jednym do detekcji nukleotydów każdej z tych par [123]; metody sekwencjonowania jednocząsteczkowego, jak PacBio i Oxford Nanopore, są bardziej podatne na obecność szumów tła [151, 171], stąd jakość ich odczytów jest znacząco słabsza, niż technik NGS.

Dla niektórych technik zostały przeprowadzone dokładniejsze badania charakterystyki błędów. Przykładowo, wykazały one większą podatność na błędy urządzeń Illumina przy sekwencjonowaniu motywu GGC [158] czy nadreprezentację odczytów fragmentów bogatych w nukleotydy C i G (własność znana jako *błąd GC* — ang. *GC bias*) [35]. Wykazano także spadek jakości odczytów Illumina w pobliżu końca $3'$, będącego efektem trudności w zachowaniu synchronizacji syntezy nici komplementarnych różnych klonów danego fragmentu [152, 184]; problem ten jest obecny także w innych technikach sekwencjonowania przez syntezę i ogranicza możliwą do osiągnięcia długość odczytów [148]. Ponadto istnieje grupa błędów powstających przypadkowo, będących rezultatem działania sekwenatorów na strukturach o rozmiarach atomowych, co wymaga ogromnej, a zatem trudnej do osiągnięcia dokładności. Dodatkową trudnością związaną z modelowaniem charakterystyki błędów jest wpływ na nią nie tylko techniki sekwencjonowania, ale także własności konkretnego modelu urządzenia czy sposobu przygotowania biblioteki.

Oprogramowanie syntezy odczytów w oparciu o dane wyjściowe sekwenatora, np. Phred dla metody dideoksy czy Bustard dla metody Illumina, wraz z sekwencjami symboli reprezentujących ciąg zasad azotowych zwraca *współczynniki jakości*, będące wartościami kodującymi prawdopodobieństwo błędu danego symbolu. Pozwalają one na oszacowanie średniego *współczynnika błędów* (średniego prawdopodobieństwa błędów) danego zestawu oraz na detekcję obszarów potencjalnie błędnych. Jednakże współczynniki te obejmują tylko niedoskonałości procesu odczytu symboli w oparciu o dane sekwenatora i nie uwzględniają bezpośrednio błędów powstałych w innych etapach [123]. Ponadto wykazano, że istnieją rozbieżności między rzeczywistymi stopami błędów, a oszacowanymi; różnice te są odmienne w różnych sekwenatorach [123, 148].

Podstawowy podział typów błędów sekwencjonowania uwzględnia rodzaj różnic między odczytem a rzeczywistą sekwencją i wyróżnia następujące klasy [209]:

- *substytucje* (ang. *substitution*, *mismatch*) — zamiana jednego, a rzadziej kilku symboli na inne,
- *insercje* (ang. *insertion*) — obecność w odczycie fragmentu, często o długości jednej zasady, niewystępującego w sekwencji wejściowej,
- *delecje* (ang. *deletion*) — brak w odczycie fragmentu, często długości jednej zasady, obecnego w sekwencji wejściowej,
- *błędy pokrycia* (ang. *coverage bias*) [123] — nierównomierność liczby reprezentacji różnych fragmentów sekwencji przez odczyty (nierównomierność *pokrycia odczytami* — ang. *read coverage*), w skrajnych przypadkach niektóre fragmenty mogą w ogóle nie znaleźć się w odczytach.

Błędy insercji i delecji są nazywane łącznie *indelami*. Ponadto w przypadku wystąpienia trudności z przeprowadzeniem sekwencjonowania danego nukleotydu bądź ich ciągu, jako wyjściową sekwencję oprogramowanie przyjmuje ciąg symboli N. W tabeli 2.1 przedstawiono wybór urządzeń sekwencjonujących wraz z podstawowymi własnościami ich odczytów.

2.3.5 Zastosowanie nowoczesnych technik sekwencjonowania

Rozwój technik NGS zaowocował możliwością uzyskania niskim kosztem dużej ilości informacji o pierwszorzędowej strukturze kwasów nukleinowych w krótkim czasie [30, 148]. W efekcie metody NGS stały się narzędziami użytecznymi nie tylko w procesach określania sekwencji nukleotydów DNA pełnego genomu, ale także w licznych nowych zastosowaniach, jak sekwencjonowanie rozmaitych typów RNA czy analiza wiązań DNA z białkami.

Sekwencjonowanie genomów

W zależności od potrzeb, proces sekwencjonowania DNA może być nacelowany na analizę pełnego genomu bądź jego fragmentów. Takie kryterium często wyznacza następujące strategie [156]: *sekwencjonowanie pełnego genomu* (ang. *whole-genome sequencing — WGS*), *sekwencjonowanie wszystkich eksonów* (ang. *whole-exome sequencing — WES*, *WXS*), *sekwencjonowanie wybranych eksonów* (ang. *targeted-exome sequencing — TES*).

Strategia WGS ma na celu przeprowadzenie sekwencjonowania pełnego genomu jednego organizmu w oparciu o próbkę DNA uzyskaną z komórek somatycznych. Do rozmaitych zastosowań WGS należy m.in. zadanie określenia pełnej postaci genomu (poszczególnych chromosomów) danego organizmu. Istotnym krokiem takiego procesu jest *asemblacja de novo* (ang. *de novo assembly*), nazywana dalej *asemblacją* której zadaniem jest wzajemne dopasowanie odczytów poprzez wykrycie ich wspólnych fragmentów, a w rezultacie określenie postaci odcinków dłuższych od odczytów. Asemblacja jest przeprowadzana przez specjalizowane narzędzia nazywane *asemblerami* (ang. *assembler*), których zasada działania opiera się na wykorzystaniu nadmiarowości będącej efektem dużej głębokości sekwencjonowania. Ze względu na obecność błędów w odczytach, ich niewielką długość, niedoskonałości asemblerów i inne czynniki wynikiem pracy asemblerów nie jest pełna postać genomu, ale zestaw odcinków różnej długości nazywanych *kontigami* (ang. *contig*). Wynik asemblacji może zostać uzupełniony dodatkowymi danymi, np. odczytami sparowanymi lub długimi odczytami technik TGS, w oparciu o które przeprowadza się budowę *skafoldów* (ang. *scaffold*), czyli kontigów o względnie określonym położeniu i orientacji.

W przypadku WES sekwencjonowaniu podlegają jedynie kodujące fragmenty genomu. Zakres analizowanych fragmentów można dalej zawęzić poprzez zasto-

Tabela 2.1: Charakterystyka wybranych modeli sekwenatorów NGS i TGS

Model sekwenatora	Typowa długość odczytów [pz]	Odczyty sparowane	Dominujący typ błędów	Typowy współczynnik błędu symbolu	Uwagi	Źródła
454 GS FLX Titanium	330	Tak	Insercje	0,01%–0,1%	Błędy sekwencjonowania homopolimerów	[30, 112, 148]
ABI SOLiD 3	50	Tak	—	0,1%–1%	—	[112, 148, 209]
Illumina Genome Analyzer II	75 lub 100	Tak	Substytucje	0,1%–1%	Spadek jakości w pobliżu końca 3′	[112, 148]
Illumina MiSeq	do 300	Tak	Substytucje	0,8%	Spadek jakości w pobliżu końca 3′	[3, 169]
Illumina HiSeq 2000	do 150	Tak	Substytucje	0,26%	Spadek jakości w pobliżu końca 3′	[169]
Illumina NovaSeq 6000	do 150	Tak	Substytucje	(brak danych)	—	[3]
Ion Torrent PGM 318	400	Tak	Indele	1,7%	Błędy sekwencjonowania homopolimerów	[37, 123, 169]
PacBio RS II	10 k	Nie	Indele	16,19%	—	[37, 123, 169]
Nanopore MinION	100 k	Nie	Indele	(brak danych)	Bardzo wysokie współczynniki błędów	[28]

sowanie TES, którego celem jest wybór wyłącznie eksonów tych genów, które zgodnie z oczekiwaniami mają znaczenie w analizowanym procesie. Wybór odpowiednich fragmentów poddawanych sekwencjonowaniu odbywa się poprzez ich *namnożenie* (ang. *target-enrichment*) przy wykorzystaniu jednej z dostępnych metod, takich jak różne odmiany techniki PCR albo *odwrócone sondy molekularne* (ang. *molecular inversion probes*) [140]. Opierają się one na wykorzystaniu jednoniciowych oligomerów o sekwencjach charakterystycznych dla sekwencjonowanego fragmentu. W wyniku przeprowadzenia hybrydyzacji ich z fragmentami próbki oraz usunięciu tych, które nie poddały się hybrydyzacji, możliwe jest zachowanie wyłącznie właściwych fragmentów.

Wszystkie powyższe strategie mogą być zastosowane w projektach *resekwencjonowania* (ang. *resequencing*), które polegają na przeprowadzeniu sekwencjonowania DNA organizmów, dla których znana jest typowa postać genomu nazywana *genomem referencyjnym* (ang. *reference genome*). Uzyskane odczyty zostają *dopasowane* (*zmapowane*, ang. *align*, *map*) do odpowiadających im miejsc w genomie referencyjnym, przy pomocy narzędzi nazywanych *maperami* (ang. *mapper*). W wyniku uzyskuje się, poza pozycją w genomie, także zestaw różnic między genomem referencyjnym a danym odczytem będących efektem zmienności wewnątrzgatunkowej oraz błędów sekwencjonowania. W szczególnych przypadkach dopasowywaniu mogą podlegać także inne sekwencje nukleotydowe, w tym kontigi lub skafoldy.

Powyższe metody znajdują zastosowanie w bardziej złożonych procesach przetwarzania informacji genetycznej. W zadaniu *detekcji wariantów* genomów (ang. *variant calling*) cel stanowi analiza istniejących różnic między genomami różnych osobników jednego gatunku lub określenie ich wpływu na fenotyp [148]. *Rekonstrukcja drzew filogenetycznych* (ang. *phylogenetic tree reconstruction*) pozwala na określenie genetycznych związków między różnymi jednostkami taksonomicznymi [104]. Ponadto niski koszt technik NGS umożliwia analizę DNA genomów indywidualnych osób, co pozwala na wyciągnięcie medycznych wniosków, np. postawienie diagnozy choroby nowotworowej poprzez badanie genomów podejrzanych komórek [30, 148].

Techniki oparte na sekwencjonowaniu DNA

Pośrednio techniki sekwencjonowania DNA znajdują zastosowane także w procesach, w których bezpośrednim celem nie jest określenie postaci genomu bądź jego fragmentu. Do tych procesów należą m.in. metody *RNA-Seq* i *ChIP-Seq*.

Grupa technik RNA-Seq umożliwia analizę mRNA, RNA odcinków niekodujących, *sRNA* i innych. Ich zasada działania polega na pobraniu próbki RNA oraz przeprowadzeniu odwrotnej transkrypcji w celu uzyskania cDNA, które podlega właściwemu sekwencjonowaniu. Szczególnie popularnym zastosowaniem technik RNA-Seq jest analiza ekspresji genów. W tym przypadku sekwencjonowaniu pod-

lega mRNA obecne w komórce poddanej działaniu określonego czynnika. Ekspresja danego genu może być określona w oparciu o częstość obecności jego sekwencji w odczytach. Liczniejsze występowanie sygnalizuje wyższy poziom ekspresji. Wynikiem eksperymentu jest określenie zależności między występowaniem danego czynnika a ekspresją określonych genów.

Techniki ChIP-Seq mają na celu określenie miejsc wiązań określonych białek z DNA. Analiza polega na izolacji badanych białek razem z dowiązanym DNA. DNA jest poddawane fragmentacji na odcinki długości kilkuset par zasad. Następnie białka podlegają usunięciu, a fragmenty DNA są poddawane sekwencjonowaniu.

Najważniejsze projekty sekwencjonowania

Niski koszt oraz wysoka jakość technik sekwencjonowania umożliwiły zapoczątkowanie wielu ambitnych programów badawczych mających na celu przeprowadzenie sekwencjonowania dużej grupy próbek, dostarczając tym samym porównawczych zbiorów danych.

W 2008 roku konsorcjum kilku instytucji badawczych zainicjowało Projekt 1000 Genomów (ang. *1000 Genomes Project*) [191], którego celem było przeprowadzenie w ciągu kilku lat sekwencjonowania genomów co najmniej 1000 osób należących do różnych grup etnicznych. Do planowanych efektów należało utworzenie dokładnej i dużej bazy wariantów ludzkiego genomu. Zakończony już projekt zaowocował uzyskaniem sekwencji ponad 2500 osób oraz odkryciem 88 milionów wariantów [50].

W podobnym czasie rozpoczęto Projekt 1000 Genomów Roślinnych (ang. *1000 Plant Genomes Project*) [144] nastawiony na przeprowadzenie sekwencjonowania transkryptów co najmniej tysiąca gatunków roślin. Obecnie projekt ten, w wyniku osiągnięcia celu, również został zakończony [22].

Jeszcze ambitniejsze zadania wyznaczono dla rozpoczętego w 2009 roku Projektu 10 Tysięcy Genomów (ang. *Genome 10K Project*) [93]. Jego założeniem było uzyskanie i przechowanie próbek tkanek oraz przeprowadzenie sekwencjonowania genomów kilkunastu tysięcy kręgowców. W ramach projektu zaproponowano dwa wydarzenia o charakterze konkursów [10] — Alignathon oraz Assemblathon, mające na celu ulepszenie wyników uzyskiwanych odpowiednio przez narzędzia mapowania oraz asemblacji odczytów.

Węższą, ze względu na obejmowany zakres gatunkowy, specjalizacją charakteryzował się projekt 1001 Genomów (ang. *1001 Genomes*) [26], którego celem było wykonanie sekwencjonowania genomów więcej niż 1000 osobników rzodkiewnika pospolitego (*Arabidopsis thaliana*), będącego jednym z organizmów modelowych, wraz z wyznaczeniem dla nich zestawów wariantów. Projekt zakończono wraz z uzyskaniem zestawów danych dla 1135 osobników.

Wąskie, pod względem specjalizacji, choć ambitne cele wyznaczono także Projektowi 100 000 Genomów (ang. *The 100,000 Genomes Project*) [19] zainicjowanemu w 2014 roku przez brytyjską instytucję ochrony zdrowia NHS. Objęły one przeprowadzenie sekwencjonowania 100 tysięcy genomów osób cierpiących na nowotwory lub rzadkie choroby, a także genomów członków ich rodzin. Zdobyta w ten sposób wiedza pozwoli na poprawę skuteczności diagnozy oraz leczenia. Uzyskiwanie danych sekwencjonowania zostało zakończone, choć przetwarzanie danych jest kontynuowane [19].

W efekcie rozwoju technik sekwencjonowania z upływem czasu Projekt 10 Tysięcy Genomów został rozwinięty, przyjmując nazwę Projektu Genomów Kręgowców (ang. *Vertebrate Genomes Project — VGP*), poszerzając cel działań do przeprowadzenia sekwencjonowania genomów 70 tysięcy organizmów [11].

2.3.6 Inne metody analizy kwasów nukleinowych

Wymienione powyżej metody sekwencjonowania kwasów nukleinowych w większości mają na celu określenie postaci wybranych odcinków nici DNA lub RNA. Jednocześnie znany jest szereg metod umożliwiających analizę informacji genetycznej, które nie wymagają bezpośredniego określenia sekwencji nukleotydów. Należą do nich *mikromacierze DNA* [212], *ilościowa PCR* [83], *multipleks PCR* [71], *hybrydyzacja Southerna* [194] i wiele innych.

Metody te posiadają zwykle nieco bardziej wyspecjalizowane zastosowanie i często wymagają wstępnej znajomości określonych sekwencji nukleotydów [148], np. charakterystycznych dla danego szczepu drobnoustroju albo genu, którego ekspresja ma być badana. Jako przykładowe zostaną omówione mikromacierze DNA, które m.in. ze względu na mnogość potencjalnych zastosowań, znaczną automatyzację [212] czy dużą szybkość [69] zdobyły znaczną popularność.

Mikromacierze DNA

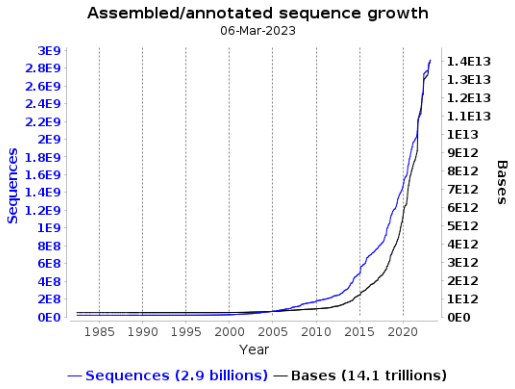
Pod pojęciem mikromacierzy DNA rozumiemy niewielką płytkę zawierającą pewną liczbę pól pokrytych licznymi, unieruchomionymi oligomerami DNA, pełniącymi funkcję sond. Oligomery, w zależności od typu mikromacierzy, mają długość kilkudziesięciu par zasad albo kilkunastu tysięcy par zasad [212] i w obrębie pola posiadają jednakową, znaną sekwencję, charakterystyczną dla rozwiązywanego problemu. W ramach eksperymentu próbka kwasu nukleinowego jest fluoroscencyjnie znakowana, dzielona, w niektórych przypadkach amplifikowana, a następnie poddawana denaturacji. Uzyskane w ten sposób nici zostają umieszczone na mikromacierzy. Wybrane fragmenty ulegają hybrydyzacji z komplementarnymi oligomerami mikromacierzy. Po pewnym czasie pola są oczyszczane z niezwiązanych fragmentów, a sama płytka jest analizowana pod kątem emisji światła przez dołączone odcinki. Intensywność światła jest proporcjonalna do liczby fragmen-

tów DNA komplementarnych do oligomerów danego pola, zatem daje podstawę do określenia względnej ilości danego fragmentu w analizowanej próbce.

Mikromacierze DNA znalazły szereg zastosowań, do których należą analiza alternatywnego składania transkryptów, badanie struktury genomu poprzez *porównawczą hybrydyzację genomową* (ang. *comparative genomic hybridization*) [212] i wiele innych. Są one użyteczne także w bezpośredniej analizie m.in. białek, antyciał i innych typów cząsteczek [69]. Jednakże jednym z ich najważniejszych zastosowań jest analiza ekspresji genów. W przypadku badań tego typu przydatne okazują się mikromacierze sondujące przy pomocy oligomerów charakterystycznych dla analizowanych genów. Intensywność światła wskazuje w tym przypadku poziom ekspresji wybranego genu. Częstym rozwiązaniem jest porównanie ekspresji dwóch próbek (badanej i kontrolnej) w ramach tzw. *znakowania dwukolorowego*. W takiej sytuacji każda z próbek jest znakowana barwnikiem o innym kolorze i analizowana na jednej płytce. Intensywność światła o określonej barwie sygnalizuje poziom ekspresji genu danej próbki.

W związku ze swoimi ograniczeniami, do których należą m.in. dopasowanie danej mikromacierzy do konkretnego zestawu genów lub fragmentów DNA, mikromacierze coraz częściej zastępowane są technikami opartymi na NGS, np. RNA-Seq. Techniki sekwencjonowania często pozwalają na przeprowadzenie analizy bez znajomości postaci genu, dzięki czemu sprawdzają się szczególnie w przypadkach występowania wariantów analizowanego odcinka [148].

2.3.7 Bazy danych biologicznych i bioinformatycznych



Rysunek 2.6: Wzrost rozmiaru bazy ENA[1]

Prowadzenie badań nad problemami biologicznymi jest ściśle związane z koniecznością przechowywania i przesyłania uzyskanych danych zarówno w pierwotnej formie, jak również danych przetworzonych. Kwestie te są związane z licznymi trudnościami, do których należą znaczny rozmiar danych i jego gwałtowny wzrost

[1]Źródło ilustracji: [18], materiały instytucji EMBL-EBI

Tabela 2.2: Wybrane bazy danych biologicznych [12, 20, 81]

Baza	Instytucja zarządzająca	Typ bazy	Zastosowanie
GenBank	NCBI	Pierwszorzędowa	Sekwencje nukleotydowe i białkowe
ENA	EMBL-EBI	Pierwszorzędowa	Sekwencje nukleotydowe i białkowe, w tym m.in. wyniki sekwencjonowania
DDBJ	NIG	Pierwszorzędowa	Sekwencje nukleotydowe i białkowe, w tym m.in. wyniki sekwencjonowania
SRA	NCBI	Pierwszorzędowa	Wyniki sekwencjonowania i mapowania
PDB	wwPDB	Pierwszorzędowa	Makromolekularne struktury biologiczne (w tym struktury białek)
dbGaP	NCBI	Drugorzędowa	Związki między ludzkim genotypem a fenotypem
Entrez	NCBI	Drugorzędowa	Mechanizm wyszukiwania w pozostałych bazach NCBI

(przykładowo: rys. 2.6), różnorodność formatów danych, konieczność dostarczenia skutecznych metod wyszukiwania, łatwość obsługi, uwzględnienie ograniczeń w dostępności do niektórych danych i wiele innych.

Wyróżniamy dwa podstawowe typy baz danych biologicznych: *pierwszorzędowe* (ang. *primary*) oraz drugorzędowe (ang. *secondary*) [16]. Bazy pierwszorzędowe zawierają dane uzyskane eksperymentalnie, udostępnione w celu dalszego przetwarzania i mające charakter archiwalny. Bazy drugorzędowe zawierają wyniki analizy danych, często łącząc zawartość różnych baz pierwszorzędowych.

Liczba baz danych podlegających zestawieniu przez czasopismo Nucleic Acids Research przekracza 2 tysiące [9]. W tabeli 2.2 przedstawiono wybór baz wraz z ich krótkim opisem.

Instytucje zarządzające bazami GenBank/SRA, ENA oraz DDBJ są związane w konsorcjum International Nucleotide Sequence Database Collaboration, którego celem jest wzajemna wymiana danych pomiędzy tymi bazami danych.

Jednoznaczna identyfikacja zestawu danych w obrębie bazy jest możliwa w oparciu o identyfikatory. W przypadku danych sekwencjonowania genomów są one określane mianem *numerów dostępowych* (ang. *accession number*) i mogą się odnosić do danych uzyskanych w wyniku jednego uruchomienia urządzenia, eksperymentu, projektu naukowego lub innych.

Rozdział 3

Kontekst informatyczny

3.1 Problemy przetwarzania informacji genetycznej

Rozwój metod sekwencjonowania DNA oraz innych gałęzi biologii molekularnej owocuje ciągłym pozyskiwaniem ogromnej ilości danych biologicznych. Akwizycja, przetwarzanie, analiza, przechowywanie, przesyłanie i zarządzanie takimi danymi wymaga zastosowania odpowiednich rozwiązań informatycznych i matematycznych. Znajdują tutaj swoje miejsce zarówno metody ogólne, stosowane także w innych dziedzinach, jak również metody specjalizowane, dopasowane do charakteru danych biologicznych [166].

Zadania przetwarzania sekwencji DNA, takie jak asemblacja, mapowanie odczytów, detekcja wariantów, rekonstrukcja drzew filogenetycznych, wizualizacja wyników, a także pomocnicze zagadnienia, jak zliczanie k-merów, przetwarzanie zbiorów k-merów i wiele innych są nietrywialnymi problemami informatycznymi. Do ich rozwiązywania stosuje się przygotowane w tym celu narzędzia będące implementacjami algorytmów bądź potokami różnych algorytmów.

Do wspomnianych technik informatycznych należy zaliczyć odpowiednie algorytmy i struktury danych oraz rozwiązania techniczne, obejmujące m.in. metody projektowania i optymalizacji oprogramowania oraz jego testowania i oceny. Ich właściwe wykorzystanie jest warunkiem koniecznym nie tylko dla uzyskania rozwiązania o oczekiwanej jakości, ale również rozsądnego gospodarowania dostępnymi zasobami, do których należą m.in. pamięć operacyjna, systemy pamięci masowej, pamięć podręczna i czas pracy procesora.

3.1.1 Definicje

Definicja 1. *Sekwencją nukleotydową nazywamy słowo* $\mathfrak{w}$ *długości* $\ell = |\mathfrak{w}|$ *nad alfabetem* $\Sigma_{\mathrm{N}} = \{A, C, G, T, N\}$.

Sekwencja nukleotydowa reprezentuje ciąg zasad azotowych budujących kolejne nukleotydy wybranego fragmentu nici DNA. Symbole A, C, G, T oznaczają kolejno nukleotyd adeninowy, cytozynowy, guaninowy i tyminowy. Z kolei symbol N odnosi się do potencjalnie obecnych w sekwencji nukleotydów o nieznanej zasadzie (ang. *uncalled bases*). W niektórych sytuacjach można się spotkać z obecnością także innych symboli (np. R — zasada purynowa, tj. A albo G), jednak w niniejszej pracy przyjęto, że są one równe N. Przy pomocy notacji $\mathfrak{w}[a]$ oznaczamy symbol o indeksie a sekwencji $\mathfrak{w}$, gdzie $0 \leqslant a \leqslant \ell - 1$. W wybranych przypadkach będziemy przyjmowali, że sekwencja nukleotydowa jest zbudowana nad alfabetem $\Sigma = \Sigma_{\mathrm{N}} \setminus \{\mathsf{N}\}$.

Definicja 2. *Podsłowem sekwencji* $\mathfrak{w}[a, b]$ *nazywamy sekwencję* $\mathfrak{w}[a]\mathfrak{w}[a{+}1] \ldots \mathfrak{w}[b]$, *gdzie* $0 \leqslant a \leqslant b \leqslant \ell - 1$.

W poniższym omówieniu struktur danych definicja podsłowa może zostać uogólniona na dowolny, skończony alfabet.

Definicja 3. *Chromosomem referencyjnym* $\mathfrak{c}$ *nazywamy sekwencję nukleotydową reprezentującą rzeczywisty chromosom wybranego organizmu lub inny fragment DNA obecny w genomie.*

W dalszej części pracy będzie przyjmowane założenie, że sekwencja chromosomu referencyjnego jest całkowicie losowa.

Definicja 4. *Genomem referencyjnym* $\mathcal{G}$ *nazywamy zbiór chromosomów referencyjnych* $\mathfrak{c}_i$, *gdzie* $i = 1, 2, \ldots, |\mathcal{G}|$.

Uwzględnienie w definicji chromosomu referencyjnego innych fragmentów wynika z istnienia sekwencji, których przynależność do konkretnych chromosomów nie została określona. W dalszej części chromosom referencyjny i genom referencyjny będą nazywane odpowiednio chromosomem i genomem.

Definicja 5. *Długością genomu referencyjnego nazywamy sumę długości tworzących go chromosomów referencyjnych:*

$$\ell_{\mathcal{G}} = \sum_{i=1}^{|\mathcal{G}|} |\mathfrak{c}_i|. \tag{3.1}$$

Definicja 6. *Odczytem* $\mathfrak{r}$ *nazywamy sekwencję nukleotydową reprezentującą odczyt sekwencjonowania DNA (*odczyt rzeczywisty*) lub imitację takiego odczytu uzyskaną w procesie symulacji (*odczyt symulowany*).*

Odczyt odpowiada sekwencji nukleotydów obecnych w jednoniciowej cząsteczce, odczytanej w kolejności $5'$–$3'$.

Definicja 7. *Współczynnikiem błędu symbolu* $\mathfrak{p}[a]$*, gdzie* $0 \leqslant a \leqslant |\mathfrak{r}| - 1$*, nazywamy szacunkowe prawdopodobieństwo, że dany symbol odpowiadającego odczytu* $\mathfrak{r}[a]$ *jest błędny, tzn. różni się od fragmentu genomu, który reprezentuje.*

Definicja 8. *Współczynnikiem jakości* $\mathfrak{q}[a]$*, gdzie* $0 \leqslant a \leqslant |\mathfrak{r}| - 1$*, nazywamy zapisaną w określonym systemie kodowania wartość prawdopodobieństwa błędu* $\mathfrak{p}[a]$ *wybranego symbolu odczytu.*

Wartości współczynników jakości $\mathfrak{q}[a]$ są elementami zbioru Q_n, gdzie $n = |Q_n|$. Wartość n określa dokładność kwantyzacji współczynników jakości, przy czym zazwyczaj $n = 42$ [79].

W wybranych przypadkach zostanie przyjęte założenie, że istnieją funkcje pozwalające na uzyskanie współczynnika błędu w oparciu o współczynnik jakości $p(q)$, gdzie q jest dowolnym współczynnikiem jakości, oraz współczynnika jakości w oparciu o współczynnik błędu $q(p)$, gdzie p jest dowolnym współczynnikiem błędu.

Definicja 9. *Zbiorem odczytów z sekwencjonowania* $\mathcal{R}$ *nazywamy zbiór par uporządkowanych* $(\mathfrak{r}_i, \mathfrak{q}_i)$*, gdzie* $i = 1, 2, \ldots, \mathfrak{n}$*, gdzie* $\mathfrak{r}_i$ *jest odczytem,* $\mathfrak{q}_i$ *jest ciągiem współczynników jakości przyporządkowanych jego symbolom, a* $\mathfrak{n} = |\mathcal{R}|$*.*

Definicja 10. *Odwróconym dopełnieniem sekwencji* $\mathfrak{w}$ *nad alfabetem* Σ *nazywamy sekwencję* $\mathrm{rev}(\mathfrak{w})$ *taką, że* $\mathrm{rev}(\mathfrak{w}) = \underline{\mathfrak{w}[\ell-1]}\,\underline{\mathfrak{w}[\ell-2]} \ldots \underline{\mathfrak{w}[0]}$*, gdzie podkreślenie oznacza przyporządkowanie symboli zgodnie z regułą:* $\underline{A} = T$*,* $\underline{C} = G$*,* $\underline{G} = C$*,* $\underline{T} = A$*.*

Definicja 11. *k-merem (oligomerem, oligonukleotydem długości k)* κ *nazywamy słowo długości* k *nad alfabetem* Σ*.*

Definicja 12. *Liczebnością* η *k-meru* κ *nazywamy liczbę jego wystąpień k-merów o sekwencji równej* κ *w określonym wielozbiorze.*

W wybranych przypadkach liczebność k-meru κ będzie określana przy pomocy notacji funkcyjnej $\eta(\kappa)$.

Definicja 13. *k-merem kanonicznym* κ_{can} *nazywamy mniejszy leksykograficznie spośród k-merów* κ *oraz jego odwróconego dopełnienia* $\mathrm{rev}(\kappa)$*.*

Definicja 14. *k-merem pokrywającym symbol* $\mathfrak{w}[a]$ *nazywamy dowolny k-mer* κ *taki, że* $\kappa = \mathfrak{w}[\mathfrak{a}, \mathfrak{b}]$*, gdzie* $0 \leqslant \mathfrak{a} \leqslant a$*,* $a \leqslant \mathfrak{b} \leqslant \ell - 1$ *oraz* $\mathfrak{b} - \mathfrak{a} + 1 = k$*.*

Definicja 15. *k-merem pokrywającym podsłowo* $\mathfrak{w}[a, b]$ *nazywamy dowolny k-mer pokrywający dowolny spośród symboli* $\mathfrak{w}[a], \mathfrak{w}[a+1], \ldots, \mathfrak{w}[b]$*.*

Rolą k-meru jest reprezentacja dowolnego fragmentu sekwencji nukleotydowej niezawierającej symboli N. Często k-mer będzie traktowany równoważnie z podsłowem pewnego odczytu $\mathfrak{r}$, przy spełnieniu warunku dotyczącego obecności symbolu N.

Wyszczególnienie k-merów kanonicznych wynika z występującego zazwyczaj braku informacji o orientacji nici DNA, której fragment dana sekwencja reprezentuje. W wybranych przypadkach będziemy przyjmowali, że k-mer jest sekwencją nad alfabetem Σ_{N}.

Definicja 16. *Błędnym k-merem nazywamy k-mer będący podsłowem odczytu zawierającym co najmniej jeden błędny symbol.*

Definicja 17. *Prawidłowym k-merem nazywamy k-mer będący podsłowem odczytu niezawierającym żadnego błędnego symbolu.*

Definicja 18. *Głębokością sekwencjonowania nazywamy wartość zdefiniowaną zgodnie ze wzorem:*

$$\mathrm{dep} = \frac{\sum_{(\mathfrak{r},\mathfrak{q})\in\mathcal{R}} |\mathfrak{r}|}{\sum_{\mathfrak{c}\in\mathcal{G}} |\mathfrak{c}|}. \tag{3.2}$$

Głębokość sekwencjonowania pełni funkcję miary stopnia redundancji danych sekwencjonowania.

3.1.2 Formaty danych

Na potrzeby komputerowej reprezentacji informacji genetycznej lub wyników jej przetwarzania opracowano szereg formatów plików. Część z nich to formaty tekstowe, w których zasady azotowe, współczynniki jakości, nagłówki odczytów oraz pozostałe dane są zakodowane w postaci symboli ASCII. Wybrane formaty posiadają swoje binarne odpowiedniki. Z powodu znacznego rozmiaru, a w przypadku plików tekstowych także ze względu na łatwość kompresji znaków ASCII, pliki zawierające sekwencje nukleotydowe zazwyczaj są przechowywane, przesyłane i przetwarzane w formie skompresowanej, najczęściej przy użyciu metod GZIP lub BZIP2, a często także BGZF, która jest rozszerzeniem GZIP przyspieszającym swobodny dostęp do skompresowanych danych [127]. Ze względu na dużą różnorodność formatów, która stanowi dodatkową trudność w opracowaniu narzędzi przetwarzania sekwencji nukleotydowych, ich wybrane przykłady zostaną omówione dokładniej.

Formaty sekwencji nukleotydowych

Podstawowym formatem reprezentacji odczytów z sekwencjonowania DNA są pliki tekstowego formatu FASTQ [47]. Zawierają one sekwencje symboli odczytów,

ciągi współczynników jakości oraz przyporządkowane im nagłówki. Każdy współczynnik jakości $\mathfrak{q}[a]$ koduje prawdopodobieństwo $\mathfrak{p}[a]$ błędu wybranego symbolu zgodnie ze wzorem:

$$\mathfrak{q}[a] = \lfloor -10 \log_{10} \mathfrak{p}[a] \rfloor. \tag{3.3}$$

Określone w ten sposób współczynniki zostają zapisane jako kody symboli ASCII poprzez dodanie odpowiedniej, zależnej od przyjętej skali przyporządkowania, wartości. Taki sposób kodowania wywodzi się z oprogramowania syntezy odczytów Phread. Format FASTQ umożliwia zapis odczytów sparowanych poprzez podział par między dwa odpowiednio nazwane pliki.

Binarnym odpowiednikiem formatu FASTQ jest SRA, który jest wykorzystywany m.in. w bazie SRA NCBI. Udostępnia on dane tego samego typu co FASTQ, jednakże natywnie reprezentuje je w postaci skompresowanej. Genomy są zazwyczaj zapisywane w tekstowych plikach FASTA, zawierających sekwencje nukleotydowe wraz z nagłówkami bez współczynników jakości. Pliki FASTA zwykle stanowią także zbiór wyjściowy procesu asemblacji.

Formaty wyników mapowania i detekcji wariantów

Standardowym sposobem zapisu wyników mapowania odczytów jest tekstowy plik SAM [130]. Każdy jego rekord odpowiada jednemu dopasowaniu odczytu do genomu i składa się z sekwencji odczytu i ciągu jego współczynników jakości, nagłówka chromosomu, pozycji w chromosomie, sekwencji *CIGAR* opisującej różnice między odczytem a odpowiadającym mu fragmentem chromosomu, ich odległości edycyjnej, współczynnika jakości dopasowania i wielu innych informacji. Binarnym odpowiednikiem formatu SAM jest BAM, przechowujący dane w postaci skompresowanej.

Wyniki procesu detekcji wariantów są zawierane w tekstowych plikach VCF [55]. W tym przypadku rekord obejmuje informacje o lokalizacji wariantu, tj. numerze chromosomu i pozycji w chromosomie. Sam wariant jest opisany swoim identyfikatorem w bazie danych bioinformatycznych, sekwencją odpowiedniego fragmentu genomu, zbiorem sekwencji wariantów tego fragmentu, współczynnikiem jakości wariantu oraz innymi wartościami.

Formaty indeksów danych

Praca algorytmów przetwarzania danych nukleotydowych niejednokrotnie wiąże się z częstym odwoływaniem się do wybranych pozycji w danych. Ze względu na ich rozmiar oraz zastosowanie kompresji, w celu osiągnięcia rozsądnego czasu wykonania takich odwołań, przeprowadza się wstępne indeksowanie danych. Uzyskane indeksy umożliwiają szybkie odnalezienie wybranej pozycji genomu, rekordu lub odczytu. Problem indeksowania sekwencji nukleotydowych oraz innych danych można rozwiązać przy pomocy narzędzi takich jak Tabix [127] czy igvto-

ols [197]. Indeksowana jest także zawartość plików BAM [197], a jeden z algorytmów mapowania odczytów, Bowtie, wykorzystuje własne narzędzie indeksujące odczyty [125]. Istnieje wiele formatów plików indeksów, a wybór odpowiedniego musi uwzględniać dopasowanie do wykorzystywanych narzędzi.

3.1.3 Trudności techniczne

Zestawy sekwencji nukleotydowych mogą osiągać znaczne rozmiary. W efekcie ich komputerowa reprezentacja może wymagać dostępności setek gibibajtów przestrzeni pamięci masowej lub operacyjnej. Rodzi to liczne trudności związane z przechowywaniem, przesyłaniem oraz przetwarzaniem takich danych.

Kluczowym zagadnieniem jest reprezentacja danych w pamięci operacyjnej podczas pracy algorytmów. W tym przypadku niezbędny jest staranny dobór struktur danych, które z jednej strony zapewnią niskie złożoności obliczeniowe operacji na nich przeprowadzanych, a z drugiej umożliwią zwięzłą reprezentację pozwalającą na wykonanie algorytmów na komputerach niewyposażonych w duże ilości kosztownej pamięci operacyjnej.

Osobną kwestię stanowi szybkość działania algorytmów. Zagadnienie to powinno być rozważane na wielu poziomach. Przede wszystkim algorytmy powinny charakteryzować się niewielką czasową złożonością obliczeniową, co jest kwestią szczególnie istotną ze względu na dużą różnorodność rozmiarów rozwiązywanych problemów — genomy eukariontów mogą być o kilka rzędów wielkości dłuższe od genomów bakterii. Z tego powodu zastosowanie algorytmu o dużej złożoności obliczeniowej, wystarczającego w niektórych przypadkach, może okazać się praktycznie niewykonalne w innych.

Ograniczenie czasu przetwarzania jest możliwe dzięki zrównolegleniu rozwiązywania zadań obliczeniowych. Współczesne komputery są wyposażone w wiele jednostek obliczeniowych (procesorów lub ich rdzeni) pozwalających na wykonywanie obliczeń w sposób równoległy. W efekcie wzrost ilości danych jest kompensowany większymi stopniami współbieżności, możliwymi do osiągnięcia przy wykorzystaniu współczesnych procesorów [141].

W dziedzinie szybkości działania istotne znaczenie mają także techniczne kwestie implementacji algorytmu. Z tego powodu niektóre narzędzia bioinformatyczne są oparte na wykorzystaniu specjalizowanych rozkazów procesora przyspieszających przetwarzanie, np. instrukcji *compare-and-swap* (CAS) do unikania potrzeby synchronizacji wątków [134, 141] lub wektorowych instrukcji rozszerzenia SSE [141].

Wybrane implementacje biorą pod uwagę zasadę działania pamięci podręcznej procesora [120, 211]. Ze względu na czasochłonność komunikacji między procesorem a pamięcią stosuje się tablicowanie pobranych danych w szybkiej, ale niewielkiej pamięci podręcznej procesora. Zakłada się tutaj czasową i przestrzenną lokalność wykorzystania danych — podczas pobierania danych z pamięci RAM

następuje skopiowanie do pamięci podręcznej także danych sąsiednich, łącznie o rozmiarze zazwyczaj 512 bitów [68], stanowiących tzw. *linię pamięci podręcznej*. Staranna optymalizacja algorytmu powinna uwzględniać także jego dopasowanie do tej własności i położyć nacisk na maksymalne wykorzystanie danych umieszczonych w bardzo bliskich obszarach pamięci oraz, ze względu na niewielki rozmiar pamięci podręcznej, na unikanie jednoczesnego przetwarzania danych z wielu odległych obszarów pamięci. Odwołanie do danych obecnych w pamięci podręcznej jest nazywane *trafieniem* (ang. *cache hit*), a odwołanie do danych obecnych tylko w pamięci operacyjnej (oraz jej skopiowanie do pamięci podręcznej) *chybieniem* (ang. *cache miss*).

Duży rozmiar danych wejściowych wiąże się z koniecznością intensywnego wykorzystania pamięci masowej, szczególnie dysków twardych. Wiele algorytmów wykorzystuje je nie tylko jako źródło danych i magazyn wyników, ale także jako magazyn danych tymczasowych, ograniczając zapotrzebowanie na pamięć operacyjną [59, 141, 173]. Szybkość transferu i czas dostępu do dysku twardego są o kilka rzędów wielkości gorsze niż do pamięci operacyjnej, wobec tego konieczne jest rozsądne korzystanie z takich rozwiązań, szczególnie poprzez zapewnienie jak największego udziału dostępu do danych w sposób sekwencyjny oraz unikanie częstych zmian wykorzystywanych obszarów dysku.

3.2 Struktury danych

Rozwiązanie problemu obliczeniowego jest możliwe przy użyciu odpowiedniego programu będącego implementacją wybranego algorytmu. Jednakże — poza algorytmem — nieodłącznym elementem programu są odpowiednie struktury danych umożliwiające praktyczną reprezentację danych wejściowych i wyjściowych algorytmu oraz jego danych tymczasowych. W dalszej części zestawiono wybrane, często wykorzystywane w bioinformatyce struktury danych.

3.2.1 Struktury drzewiaste

Pod pojęciem *drzewa* lub *struktury drzewiastej* rozumiemy zbiór V_{tree} *węzłów* spełniających następującą, rekurencyjną definicję [23, 113]:

1. wyróżniony węzeł $v_{\text{root}} \in V_{\text{tree}}$ jest korzeniem drzewa V_{tree},
2. pozostałe węzły są podzielone pomiędzy $m \geqslant 0$ rozłącznych podzbiorów $V_1, V_2, \ldots, V_m$, spośród których każdy jest drzewem; drzewa $V_1, V_2, \ldots, V_m$ są *poddrzewami* V_{tree}.

Powyższa definicja odpowiada drzewu *ukorzenionemu*. Jeśli przyjąć, że węzły drzewa są wierzchołkami grafu, wówczas alternatywną kategorię drzew *nieukorzenionych* można rozpatrywać jako spójne grafy acykliczne. Drzewo ukorzenione

także może być traktowane jako skierowany spójny graf acykliczny, dodatkowo spełniający dla korzenia v_{root} następujące warunki:

1. każdy węzeł $v \neq v_{\text{root}}$ jest węzłem początkowym krawędzi (łuku),
2. v_{root} nie jest węzłem początkowym żadnego łuku,
3. v_{root} jest korzeniem drzewa.

Krawędzie drzewa ukorzenionego reprezentują relacje rodzicielstwa między węzłami, te z kolei określają hierarchiczną strukturę węzłów drzewa [23]. Węzeł początkowy łuku jest nazywany *dzieckiem*, a węzeł końcowy *rodzicem* (bezpośrednim *przodkiem*) incydentnego węzła. Dwa różne węzły v_1 oraz v_2 nazywane są *rodzeństwem*, jeżeli posiadają tego samego rodzica. Węzeł niebędący następnikiem żadnej relacji nazywany jest *liściem*.

Powyższa definicja nakreśla ogólną strukturę danych, która w zależności od potrzeb może zostać wyspecjalizowana. Częstym przypadkiem jest ograniczenie liczby dzieci danego węzła np. do dwóch lub określenie porządku dzieci danego węzła. Takie rozwiązania są obecne w rozmaitych wariantach drzew. *Drzewa poszukiwań binarnych* (ang. *binary search tree — BST*) mogą reprezentować m.in. zbiory dynamiczne lub kolejki priorytetowe [52]. *B-drzewa* posiadają strukturę pozwalającą na wykonywanie szybkiego przetwarzania zbiorów danych umieszczonych w urządzeniach pamięci masowej, np. na dyskach twardych [52], co umożliwia zarządzanie zestawami danych o rozmiarach większych niż dostępna pamięć wewnętrzna. *Kopce* są strukturami łatwymi do reprezentacji przy pomocy tablic [23], mającymi zastosowanie w sortowaniu lub implementacji kolejek priorytetowych [52].

Osobna grupa struktur drzewiastych jest dostosowana do reprezentacji sekwencji symboli. Należą do niej m.in. *drzewa słownikowe* (ang. *trie*) [166], zwane też *drzewami wielokrawędziowymi* [54], oraz *drzewa sufiksów* [58]. Ze względu na szerokie zastosowanie w bioinformatyce drzewa sufiksów zostaną dokładniej omówione w podrozdziale 3.2.4.

3.2.2 Tablice mieszające

Tablica mieszająca jest wektorem elementów umieszczonych na pozycjach wyznaczanych przez wartości *funkcji mieszającej* $h : \mathbb{U} \to \{0, 1, \ldots, m-1\}$, gdzie $\mathbb{U}$ jest przestrzenią wartości kluczy, które stanowią kryterium wyszukiwania w strukturze danych, a m jest rozmiarem tablicy [52]. Funkcja h zazwyczaj nie jest injekcją, w efekcie czego istnieje ryzyko wystąpienia *kolizji*, czyli sytuacji, w której dla pewnych $u_1, u_2 \in \mathbb{U}, u_1 \neq u_2$ występuje $h(u_1) = h(u_2)$. W rezultacie więcej niż jeden element może zostać przyporządkowany tej samej pozycji wektora elementów. Problem rozwiązuje się w oparciu o jedną spośród *strategii przezwyciężania*

kolizji. Należą do nich *strategia łańcuchowa*, która dopuszcza zapisanie w każdej pozycji wektora więcej niż jednego elementu (np. poprzez wykorzystanie listy z dowiązaniami lub drzewa), oraz *adresowanie otwarte*[1], w której h jest funkcją dwóch zmiennych: $h : \mathbb{U} \times \{0, 1, \ldots, m-1\} \to \{0, 1, \ldots, m-1\}$, gdzie drugim argumentem jest aktualny, zmieniający wartość funkcji, numer kolizji.

Zakładając, że złożoność czasowa wartościowania funkcji h wynosi $O(1)$ a kolizje nie występują, złożoność wyszukiwania w tablicy mieszającej wynosi $O(1)$. Jednakże możliwość wystąpienia kolizji pogarsza pesymistyczną złożonością czasową do $\Theta(n)$. W praktyce jednak szybkość operacji na tablicy mieszającej jest zależna od *współczynnika wypełnienia* $\alpha = {}^{n}/_{m}$. Przyjmując, że $\alpha = O(1)$, można zachować średnią złożoność wyszukiwania w tablicy mieszającej $O(1)$ [52]. Dla spełnienia tego założenia określa się stały, zależny od strategii przezwyciężania kolizji próg wartości α, po przekroczeniu którego przeprowadza się *restrukturyzację tablicy* polegającą na zwiększeniu jej rozmiaru oraz ponownym rozmieszczeniu elementów. Restrukturyzacja jest wykonywana w średnim czasie $\Theta(n)$.

3.2.3 Filtry Blooma

Filtr Blooma jest wektorem bitowym o rozmiarze m [38], którego elementy przyjmują wartości wyznaczane przez d różnych funkcji mieszających. Jest on strukturą umożliwiającą rozwiązywanie problemów decyzyjnych polegających na wyszukiwaniu elementów w zbiorze i zwracaniu logicznej wartości sygnalizującej ich obecność. Operacje wstawiania oraz wyszukiwania obejmują wyliczenie wartości funkcji mieszających $h_i : \mathbb{U} \to \{0, 1, \ldots, m-1\}$, gdzie $i = 1, 2, \ldots, d$, a $\mathbb{U}$ jest przestrzenią wartości kluczy elementów struktury. Wstawienie elementu polega na przyporządkowaniu wartości 1 pozycjom o indeksach równym wartościom funkcji mieszających. W celu wykonania wyszukiwania przeprowadza się odczyt tych pozycji; jeżeli każda z nich zawiera wartość 1, wówczas następuje zwrócenie odpowiedzi twierdzącej, a przeczącej w przeciwnym przypadku.

Filtry Blooma dopuszczają możliwość zwrócenia nieprawidłowej odpowiedzi twierdzącej. Prawdopodobieństwo takiego błędu można oszacować wzorem [147]:

$$p_{\mathrm{FP}} \approx (1 - e^{-d\frac{n}{m}})^d. \tag{3.4}$$

Wyszukanie elementu wymaga wyliczenia wielu wartości funkcji mieszających, co wiąże się z odwołaniem do wzajemnie odległych pozycji wektora. W efekcie jeżeli wektor jest duży, takie rozwiązanie wiąże się ze słabym wykorzystaniem pamięci podręcznej procesora. W związku z tym opracowano warianty podstawowej wersji struktury dopasowane do charakterystyki pamięci podręcznej. Należy do nich m.in. *blokowy filtr Blooma* (ang. *blocked Bloom filter*) [168], który jest wektorem bitów logicznie podzielonych na bloki o rozmiarze równym rozmiarowi linii

[1] W pracy [23] mianem adresowania otwartego jest określana strategia łańcuchowa.

pamięci podręcznej. Jego funkcje mieszające pełnią dwie role: pierwsza funkcja jest odpowiedzialna za wybór bloku, a pozostałe wyznaczają obecne w obrębie bloku bity. Tym samym, zapewniając wyrównanie adresu początku wektora do rozmiaru bloku, liczba chybień pamięci podręcznej nie przekracza jednego, mogącego nastąpić w momencie odczytu wybranego bloku. Szybszym wariantem blokowego filtru Blooma jest *blokowy filtr Blooma ze wzorcami* (ang. *pattern-blocked Bloom filter*) [168], w którym zamiast wywoływać określoną liczbę funkcji mieszających wyznaczających pozycje w bloku wylicza się jedną funkcję określającą maskę (*wzorzec)* wyznaczającą bity poddawane odczytowi lub modyfikacji.

Do zastosowań filtrów Blooma należą rozwiązania problemów, w których sporadyczne występowanie błędów stanowi niższy koszt niż przechowywanie w pamięci pełnej, niejednokrotnie posiadającej znaczny rozmiar tablicy mieszającej, a jednocześnie nie występuje potrzeba odzyskania wartości elementów. Jednakże jeżeli obecność błędów nie jest akceptowalna, wówczas wykonywanie zapytań do filtra Blooma może być wspomagane przez dodatkowy, często bardziej pracochłonny, ale zwracający poprawne wyniki test, wykonywany w przypadku zwrócenia przez filtr odpowiedzi twierdzącej. Rozwiązanie to sprawdza się, gdy zdecydowana większość zapytań zwraca odpowiedź negatywną [38].

3.2.4 Drzewa i tablice sufiksów

Drzewo i *tablica sufiksów* są strukturami danych reprezentującymi sekwencje symboli w sposób umożliwiający ich szybką analizę [58, 165]. Idea tego rozwiązania opiera się na przechowywaniu poszczególnych sufiksów danej sekwencji S, tzn. sekwencji $\overset{a}{S} = S[n-a, n-1]$ dla każdego $a \in \{1, 2, \ldots, n\}$, gdzie $n = |S|$.

Drzewo sufiksów przechowuje sufiksy w drzewie spełniającym pięć warunków [58]:

1. każdy sufiks $\overset{a}{S}$ jest reprezentowany przez dokładnie jeden liści drzewa,
2. każdy z węzłów niebędących liściem lub korzeniem ma co najmniej dwóch potomków,
3. każda krawędź posiada etykietę będącą podsłowem S,
4. etykiety krawędzi łączących dany węzeł z jego potomkami muszą zaczynać się od różnych symboli,
5. konkatenacja podsłów na ścieżce między korzeniem a a-tym liściem reprezentuje $\overset{a}{S}$.

Praktycznym ograniczeniem jest, aby symbol $S[|S|-1]$ był unikalny w obrębie S. Wymaganie to jest często spełniane poprzez zwiększenie długości S o 1 oraz umieszczenie na ostatniej pozycji symbolu nienależącego do alfabetu sekwencji,

a znajdującego się w kolejności leksykograficznej później, niż wszystkie symbole tego alfabetu.

Budowa drzewa sufiksów dla sekwencji długości n jest możliwa do przeprowadzenia w czasie $O(n)$. Korzystając ze zbudowanego drzewa sufiksów zadanie wyszukania wzorca S' długości $m = |S'|$, $m \leqslant n$, można wykonać w czasie $O(m)$, natomiast zadanie wyszukania wszystkich $\bar{z}$ wystąpień wzorca w sekwencji jest możliwe w czasie $O(m + \bar{z})$. Złożoność pamięciowa drzew sufiksów wynosi $O(n)$, jednak jej współczynniki są dosyć duże [58].

Redukcja zapotrzebowania drzewa sufiksów na pamięć jest niekiedy możliwa poprzez zastąpienie go tablicą sufiksów. Struktura ta ma postać wektora zawierającego pozycje uporządkowanych leksykograficznie sufiksów sekwencji. Tablice sufiksów mają nieco mniejsze możliwości a operacje na nich charakteryzują się wyższą czasową złożonością obliczeniową, np. wyszukiwanie wzorca jest możliwe w średnim czasie $O(m \log n)$, a zliczenie liczby wystąpień w czasie $O(m \log n + \bar{z})$ [58]. Jednocześnie jednak charakteryzują się lepszymi współczynnikami pamięciowej złożoności obliczeniowej. Zarówno tablica jak i drzewo sufiksów mogą być zbudowane dla wielu sekwencji jednocześnie, co pozwala na wykonywanie dodatkowych operacji. Struktura danych tej postaci jest nazywana *uogólnionym* drzewem lub tablicą sufiksów.

Drzewa i tablice sufiksów pełnią głównie funkcję indeksów sekwencji umożliwiających ich szybkie przeszukiwanie [58, 165, 166], choć posiadają także inne zastosowania. Należą do nich m.in. wyszukiwanie najdłuższego wspólnego podsłowa dwóch sekwencji (przy użyciu uogólnionych drzew lub tablic sufiksów), wyszukiwanie najdłuższego odwróconego powtórzenia (zadanie tego rodzaju można odnieść do wyszukiwania sekwencji zasad azotowych komplementarnych nici DNA), wyznaczanie transformaty Burrowsa-Wheelera i inne [58].

3.2.5 Wybór pozostałych struktur danych

W przetwarzaniu danych bioinformatycznych znalazło zastosowanie wiele innych struktur danych, często opracowanych specjalnie w tym celu. Duże znacznie posiadają struktury indeksowe, które pozwalają na reprezentację sekwencji nukleotydowych lub białkowych w sposób umożliwiający dostatecznie szybkie wykonywanie, rozmaicie zdefiniowanego, wyszukiwania wzorców. Poza drzewami i tablicami sufiksów należą do nich m.in. indeksy FM oraz tablice Gk. Ponadto, szczególnie w procesie asemblacji, kluczową rolę odgrywają grafy de Bruijna.

Indeks FM (ang. *FM-index*) [75] jest strukturą danych wykorzystującą *transformatę Burrowsa–Wheelera* (ang. *Burrows–Wheeler transform — BWT*) [41]. BWT jest algorytmem umożliwiającym odwracalne przekształcenie sekwencji poprzez uzyskanie określonej permutacji jego symboli. W tym celu algorytm generuje wszystkie rotacje sekwencji, które następnie są poddawane sortowaniu w kolejności leksykograficznej. Wynik transformaty stanowi ciąg symboli zawartych

na ostatnich pozycjach odpowiednio każdej z rotacji. Odwracalność operacji jest zapewniana poprzez zapamiętanie indeksu pierwszego symbolu ciągu wejściowego lub dołączenie do jego końca symbolu nienależącego do alfabetu sekwencji. Sekwencja wyjściowa zazwyczaj lepiej od wejściowej poddaje się kompresji [75], dzięki czemu, wśród wielu swoich zastosowań, BWT pełni szczególnie istotną rolę w wielu algorytmach kompresji.

Indeks FM jest *indeksem pełnotekstowym* (ang. *full-text index*) umożliwiającym indeksowanie jednego, długiego ciągu symboli, np. sekwencji genomu. Jak dotąd opracowano szereg wariantów oryginalnego indeksu, charakteryzujących się m.in. lepszymi złożonościami obliczeniowymi ze względu na rozmiar alfabetu [76, 88]. Ich zasada działania polega na przekształceniu sekwencji wejściowej S przy pomocy BWT oraz, wraz z dodatkowymi informacjami, reprezentacji jej w formie skompresowanej. Algorytmy wykonujące odczyt (*zapytanie* — ang. *query*) danych z indeksu FM nie wymagają przeprowadzania dekompresji pełnego ciągu. Do typowych zapytań należą m.in. zliczanie wystąpień wzorcowego oraz wyszukiwanie pozycji wszystkich $\bar{z}$ jego wystąpień. Złożoności czasowe poszczególnych operacji różnią się w zależności od wariantu struktury danych. Złożoność pamięciowa jest zależna od entropii sekwencji S [75, 76, 88].

Tablica Gk (ang. *Gk array*) [165] jest strukturą typu *indeks słów* (ang. *word-based index*) dostosowaną do indeksowania dużych zestawów krótkich sekwencji, takich jak odczyty z sekwencjonowania. Zasada jej działania jest oparta na wykorzystaniu czterech wektorów. Podstawowy wektor, nazywany *GkSA*, jest zmodyfikowaną tablicą sufiksów ciągu będącego konkatenacją wszystkich sekwencji poddawanych indeksowaniu. Pozostałe wektory są odpowiedzialne za przechowywanie istotnych pozycji tego ciągu oraz liczebności m-merów, gdzie $m = |S'|$ jest długością wzorca będącego parametrem zapytania. Zliczenie wystąpień wzorca S' w zbiorze może być wykonane w czasie $O(1)$, a znalezienie wszystkich sekwencji (odczytów), w których występuje wzorzec w czasie $O(\bar{z}')$, gdzie $\bar{z}'$ jest liczbą sekwencji wynikowych. Złożoność pamięciowa tablic Gk wynosi $8(\ell - k + 1)\mathfrak{n} + 4(\bar{m}_{\text{un}} + 1)$, gdzie ℓ jest długością odczytu, k jest przyjętą długością oligomeru, $\mathfrak{n}$ jest liczbą sekwencji, a $\bar{m}_{\text{un}}$ jest liczbą unikalnych m-merów. Sumaryczne zapotrzebowanie na pamięć jest zwykle niższe, niż alternatywnie zastosowanej uogólnionej tablicy sufiksów.

Innym zastosowaniem charakteryzuje się struktura grafu de Bruijna [56]. Jej koncepcja polega na zapisie w węzłach grafu wszystkich istniejących sekwencji długości $k-1$ nad danym alfabetem. Sam graf przechowuje informację o wspólnych sufiksach i *prefiksach* (gdzie prefiksem sekwencji S nazywana jest każda sekwencja $\underset{a}{S} = S[0, a-1]$, gdzie $a \in \{1, 2, \ldots, |S|\}$) sekwencji długości k. Wybrane dwa węzły v_1 oraz v_2 takiego grafu są sąsiednie wtedy i tylko wtedy, gdy sufiks sekwencji v_1 jest równy prefiksowi v_2. Łuki grafu reprezentują sekwencje długości k powstałe z połączenia sekwencji v_1 z ostatnim symbolem v_2. W licznych,

w tym bioinformatycznych, zastosowaniach wykorzystuje się podgrafy grafów de Bruijna, tj. grafy de Bruijna niezawierające wybranych węzłów lub krawędzi. Reprezentują one tylko podzbiór wszystkich sekwencji długości $k-1$, np. sekwencje obecne w odczytach sekwencjonowania [48]. Jednocześnie powszechną praktyką jest nazywanie także takiej struktury grafem de Bruijna. Podgrafy takiej postaci są często wykorzystywana w algorytmach asemblacji DNA i RNA, umożliwiając znajdowanie wspólnych fragmentów odczytów, które potencjalnie mogą reprezentować sąsiednie fragmenty genomu. W takim przypadku podgraf może być wykorzystany do wyznaczania ciągów odczytów pozwalając na ich połączenie w celu uzyskania postaci danego genomu.

3.3 Wybrane techniki algorytmiczne

W rozwiązywaniu problemów dotyczących przetwarzania sekwencji nukleotydowych znajdują zastosowania różne strategie projektowania algorytmów, takie jak programowanie dynamiczne, algorytmy zachłanne, algorytmy przeszukiwania wyczerpującego czy algorytmy heurystyczne. Jednocześnie ze względu na duży rozmiar przetwarzanych danych, potencjalnie skutkujący nieakceptowalnie dużym zapotrzebowaniem na pamięć operacyjną oraz czas obliczeń, algorytmy te muszą być odpowiednio dostosowane do charakterystyki takich problemów. W niniejszym podrozdziale przestawiono wybrane techniki związane z tworzeniem algorytmów bioinformatycznych oraz przedstawiono techniki przetwarzania współbieżnego, które ze względu na rozmiar omawianych problemów są praktycznie niezbędne.

3.3.1 Algorytmy z powrotami

Pod pojęciem *algorytmu z powrotami* (ang. *backtracking algorithm*) rozumiemy algorytm poszukujący rozwiązania w zadanej przestrzeni poprzez sukcesywne dobieranie kolejnych składników rozwiązania oraz ewentualne ich wycofywanie i podejmowanie innych prób. Stanowi on udoskonaloną alternatywę *przeszukiwania siłowego* (ang. *brute force*) [91]. Algorytmy z powrotami, obok *algorytmów sita*, należą do grupy algorytmów *wyszukiwania wyczerpującego* (ang. *exhausting search*) [54].

Jeżeli dopuszczalne rozwiązanie problemu jest wektorem $X = (x_1, x_2, \ldots, x_n)$ wartości o dziedzinach odpowiednio $\mathbb{X}_1, \mathbb{X}_2, \ldots, \mathbb{X}_n$, wówczas celem algorytmu z powrotami jest wyszukanie w przestrzeni $\mathbb{X}_1 \times \mathbb{X}_2 \times \ldots \times \mathbb{X}_n$ wektora X (lub zbioru wektorów X_j) spełniającego określone dla danego problemu *kryteria poprawności* (ang. *validity criteria*). Poszukiwanie rozwiązania odbywa się poprzez przeszukiwanie wgłąb (tj. poruszając się od korzenia w kierunku wybranego liścia oraz wykonywanie powrotów w stronę korzenia) drzewa, którego poszczególne liście reprezentują rozwiązania, a pozostałe węzły rozwiązania częściowe [54].

Zagłębianie się w drzewie (rozszerzanie rozwiązania) wiąże się z przyporządkowaniem wartości kolejnym zmiennym $x_1, x_2, \ldots, x_n$ z zaznaczeniem, że dalsze rozszerzanie jest przeprowadzane tylko, gdy dotychczasowe rozwiązanie cząstkowe $(x_1, x_2, \ldots, x_i)$ dla $i < n$ spełnia kryteria poprawności. Osiągnięcie liścia spełniającego kryteria jest tożsame ze znalezieniem jednego rozwiązania. W przypadku gdy dalsze rozszerzanie rozwiązania nie jest możliwe (ze względu na znalezienie pewnego rozwiązania lub gdy żadne z dalszych rozszerzeń nie jest prawidłowe), wówczas przeprowadza się powrót do wcześniejszej zmiennej poprzez przejście w drzewie w stronę korzenia i podjęcie innej próby.

Typowym zadaniem rozwiązywanym przy pomocy algorytmów z powrotami jest *problem spełniania ograniczeń* (ang. *constraint satisfaction problem — CSP*), w którym na poszczególne zmienne $x_1, x_2, \ldots, x_n$ nałożony jest zbiór *ograniczeń* (ang. *constraint*) $\mathscr{R} = \{R_1, R_2, \ldots, R_m\}$. Każde z ograniczeń R_j, $j \in \{1, 2, \ldots, m\}$ jest relacją określającą zbiór prawidłowych wartości określonego zestawu zmiennych, nazywany ich *zakresem* (ang. *scope*) [57]. Celem CSP jest dobór wartości parametrów $x_1, x_2, \ldots, x_n$, aby spełniały wszystkie ograniczenia $\mathscr{R}$.

Algorytmy z powrotami, ze względu na potencjalnie duży rozmiar przeszukiwanej przestrzeni, w sposób bezpośredni zwykle nie znajdują zastosowania praktycznego [54]. Opracowano jednak szereg modyfikacji pozwalających na przyspieszenie ich pracy. Wykorzystują one obserwację, że podczas przechodzenia drzewa rozwiązań następuje wielokrotne sprawdzanie tych samych błędnych i prawidłowych rozwiązań cząstkowych. Wśród takich modyfikacji można wyróżnić *optymalizacje powrotów* (ang. *look-back scheme*) oraz *optymalizacje zagłębień* (ang. *look-ahead scheme*) [57].

Optymalizacje powrotów mają na celu wyeliminowanie niektórych fragmentów drzewa w chwili wykonywania powrotów do wyższych węzłów. Może to obejmować przejście do niebezpośredniego przodka bieżącego węzła (ang. *backjumping*) bądź określenie przyczyny nieznalezienia w danym momencie rozwiązania i dodanie kolejnych ograniczeń (ang. *constraint recording*) umożliwiających eliminację w przyszłości porażek o tej samej przyczynie. Inną formą optymalizacji powrotów jest przechowywanie informacji o spełnieniu lub niespełnieniu kryteriów poprawności po przypisaniu danej wartości do bieżącej zmiennej x_i, $i \in \{1, 2, \ldots, n\}$ (ang. *backmarking*). Następnie te informacje są wykorzystywane do pominięcia lub wymuszenia sprawdzenia innej części drzewa właściwej dla zmiennej x_i i jej określonej wartości [118].

Optymalizacje zagłębień są strategiami modyfikującymi przechodzenie drzewa w głąb. Mogą one obejmować kwestię wyboru zmiennej, do której zostanie w danej chwili przyporządkowana wartość. Ponadto metoda ta może uwzględniać zmianę kolejności doboru wartości mających być przypisanymi do wybranej zmiennej x_i.

Dodatkową metodą może być ograniczenie rozmiaru problemu poprzez nałożenie dodatkowych ograniczeń, zawężanie istniejących ograniczeń lub redukcję

dziedzin $\mathbb{X}_1, \mathbb{X}_2, \ldots, \mathbb{X}_n$. Wszystkie z powyższych optymalizacji wymagają mechanizmów dedukcji określających możliwość wykonania i zakres przeprowadzanych optymalizacji. Do takich strategii należą m.in. rozwiązania heurystyczne.

3.3.2 Algorytmy heurystyczne

Wiele problemów obliczeniowych posiada własności powodujące, że ich rozwiązanie przy pomocy dokładnego, bezpośrednio uzyskującego wynik algorytmu jest zadaniem praktycznie niemożliwym. W pracy [149] przedstawiono pięć takich własności, zaznaczając jednocześnie, że lista ta nie jest kompletna:

- liczba potencjalnych rozwiązań jest zbyt duża, aby problem rozwiązać metodą wyszukiwania wyczerpującego,
- stopień skomplikowania problemu wymusza jego uproszczenie,
- funkcja oceny rozwiązania jest zaszumiona lub ulega zmianom w czasie,
- na potencjalne rozwiązania są narzucone znaczne ograniczenia,
- uzyskanie rozwiązania jest utrudnione przez niedostateczne przygotowanie lub obecność barier psychologicznych osoby rozwiązującej problem.

Rozwiązanie problemu charakteryzującego się powyższymi cechami często jest możliwe przy pomocy wybranego *algorytmu heurystycznego* (*heurystyki*). Heurystyki są metodami polegającymi na przeszukiwaniu przestrzeni rozwiązań w sposób wybiórczy, ale intuicyjnie prowadzący do uzyskania jak najlepszego rozwiązania. Ich ograniczeniem jest brak gwarancji uzyskania rozwiązania optymalnego, dlatego są one dopuszczalne w przypadku, gdy rozwiązanie nieoptymalne (*suboptymalne*) jest akceptowalne i jego osiągnięcie stanowi niższy koszt niż zastosowanie algorytmu dokładnego. Z drugiej strony w wielu przypadkach algorytm jest wykonywany nie dla dokładnej komputerowej reprezentacji wybranego problemu, ale dla upraszczającego go modelu. Z tego powodu uzyskanie rozwiązania dokładnego i tak byłoby obarczone błędem niedokładności modelu i jego poszukiwanie byłoby bezcelowe [170].

Duży wpływ na jakość wyników heurystyk oraz sposobu ich uzyskania ma reprezentacja problemu [58]. Popularnym podejściem jest zastosowanie *reprezentacji binarnej*, w której potencjalne rozwiązania są wyrażane przy pomocy wektora bitów. *Reprezentacja permutacyjna* wykorzystuje wektor liczb naturalnych kodujących rozwiązanie będące określoną permutacją elementów. Liczba opracowanych reprezentacji jest długa, a jej wybór musi być uzależniony od charakterystyki problemu oraz sposobu działania heurystyki.

Praca algorytmu heurystycznego polega na przeszukiwaniu przestrzeni potencjalnych rozwiązań $\mathbb{X}$ w celu znalezienia rozwiązania najlepszego lub jako-

ściowo do niego zbliżonego. W efekcie musi zostać zdefiniowane kryterium oceny; jest ono sformalizowane przy pomocy *funkcji celu* (ang. *evaluation function*) $\text{eval} : \mathbb{X} \to \mathbb{R}$, która osiąga wartość minimalną dla optymalnego rozwiązania $x_{\text{opt}} = \arg\min_{x \in \mathbb{X}} \text{eval}(x)$. Przy takim sformułowaniu problemu można przyjąć, że jego rozwiązywanie jest tożsame z optymalizacją funkcji celu [149].

Dla każdego elementu $x \in \mathbb{X}$ definiuje się zbiór jego *sąsiedztwa*, czyli elementów znajdujących się według przyjętego kryterium dostatecznie blisko elementu x. Sąsiedztwo jest definiowane przez funkcję $N : \mathbb{X} \to 2^{\mathbb{X}}$, a jego postać jest zależna od problemu oraz przyjętego sposobu jego reprezentacji. Przykładowo, jeżeli pewne $x_0 \in \mathbb{X}$ jest zbiorem punktów przestrzeni euklidesowej, sąsiedztwo może być zdefiniowane jako $N(x) = \{y \in \mathbb{X} : \text{dist}(x, y) \leqslant \epsilon\}$, gdzie ϵ jest pewną dodatnią stałą, a $\text{dist} : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ jest odległością euklidesową x i y. Z kolei jeżeli rozwiązanie jest reprezentowane binarnie, wówczas $N(x)$ może być zbiorem elementów, których reprezentacja różni się od reprezentacji x jednym bitem [149].

Przebieg pracy heurystyki jest następujący [58]:

1. ustalenie rozwiązania początkowego $x_1 \in \mathbb{X}$,

2. generacja nowych rozwiązań poprzez uzyskanie zbioru $N(x_i)$, gdzie $i = 1, 2, \ldots, n_{\max} - 1$ jest numerem poprzedniego rozwiązania, a $n_{\max}$ jest pewnym ograniczeniem liczby iteracji algorytmu,

3. selekcja elementu x_{i+1} spośród prawidłowych elementów $N(x_i)$; x_{i+1} zastępuje rozwiązanie x_i,

4. sprawdzenie przyjętego warunku stopu.

Proste algorytmy heurystyczne często są zgodne ze strategią *lokalnych ulepszeń*, polegającą na wprowadzaniu modyfikacji do poprzedniego rozwiązania, lub z metodą *konstrukcyjną*, polegającą na sukcesywnym rozbudowaniu rozwiązania cząstkowego. Spotykane są także rozwiązania hybrydowe [119]. Wybór w każdej iteracji kolejnego rozwiązania lub jego rozszerzenia wyłącznie spośród sąsiedztwa elementu bieżącego niesie za sobą ryzyko osiągnięcia *optimum lokalnego*, czyli elementu x_{lok}, takiego że $\text{eval}(x_{\text{lok}}) \leqslant \text{eval}(x)$ dla każdego $x \in N(x_{\text{lok}})$ [149], podczas gdy istnieje pewien element x_{opt}, taki że $\text{eval}(x_{\text{opt}}) < \text{eval}(x_{\text{lok}})$. Osiągnięte optimum lokalne jest silnie zależne od przyjętego rozwiązania początkowego x_1.

Eliminacja problemu blokowania algorytmu w sąsiedztwie minimum lokalnego jest celem *metaheurystyk*, będących wysokopoziomowymi rozwiązaniami wykorzystującymi proste algorytmy heurystyczne [170] i posiadających strategie opuszczania minimów lokalnych. Realizują one dwa cele: *eksplorację*, polegającą na przeszukaniu jak największej części przestrzeni rozwiązań w celu znalezienia obiecujących regionów, oraz *eksploatację*, będącą dogłębnym przeszukiwaniem

tych regionów [149]. Do metaheurystyk należy wiele, często inspirowanych zjawiskami naturalnymi algorytmów, do których należą m.in. symulowane wyżarzanie, algorytmy genetyczne, algorytmy mrówkowe czy symulacja rojem cząstek [84].

Praca z algorytmami heurystycznymi wiąże się z zagadnieniem ich parametryzacji. Trudne do przewidzenia zależności między wartościami parametrów a uzyskanymi wynikami, zbyt duża liczba kombinacji parametrów do przeprowadzenia doboru eksperymentalnego, złożona, nieliniowa zależność między parametrami, brak zrozumienia problemu problemu przez użytkownika, zależność parametrów od konkretnego problemu i wiele innych kwestii powodują, że proces ten jest zadaniem skomplikowanym. Analityczne określanie parametrów jest możliwe tylko w wybranych, prostych przypadkach. Pewne znaczenie może mieć intuicja lub analogia do innych, podobnych problemów oraz wiedza ekspercka, choć są to metody dalece niedoskonałe. Elementem algorytmu powinna być, przynajmniej częściowa, zdolność samoadaptacji [149], jednakże w ogólnym przypadku określenie efektywności heurystyki w danych okolicznościach jest możliwe tylko na drodze eksperymentalnej [75].

3.3.3 Współbieżność

Architektury komputerów są poddawane ciągłym ulepszeniom. Obecnie szczególnie duży nacisk kładzie się na doskonalenie architektur oraz technik pozwalających na równoległe wykonywanie obliczeń [176]. Są to narzędzia umożliwiające m.in. rozwiązywanie problemów obliczeniowych o dużym stopniu skomplikowania. Podstawowym, choć nie jedynym ich celem jest przyspieszenie obliczeń poprzez dekompozycję danego problemu obliczeniowego na podproblemy (zadania) mogące zostać rozwiązane jednocześnie. Wyróżnia się kilka poziomów przetwarzania równoległego, różniących się zasadą działania, zastosowaniem, możliwościami i innymi cechami. W niniejszym podrozdziale zostanie omówiona *równoległość na poziomie wątków* (ang. *thread-level parallelism*).

Procesy współbieżne

Niech $\underline{\underline{o_i}}$ i $\overline{\overline{o_i}}$ będą odpowiednio początkiem oraz końcem pewnej operacji o_i wybranego zadania obliczeniowego realizowanego przez pewien proces (instancję programu) $\mathcal{P}$, a $\mathfrak{t}(\cdot)$ niech będzie momentem wystąpienia pewnego zjawiska. Dla *procesu sekwencyjnego* prawdziwa jest nierówność $\mathfrak{t}(\overline{\overline{o_i}}) \leqslant \mathfrak{t}(\underline{\underline{o_{i+1}}})$ dla każdej operacji o_i [53]. Zasada ta oznacza, że działanie procesu sekwencyjnego jest oparte na wykonaniu poszczególnych operacji w niezmiennej, liniowej kolejności.

Rozszerzeniem procesów sekwencyjnych, możliwym do uzyskania we współczesnych systemach operacyjnych, są *procesy współbieżne*. Są nimi takie dwa procesy $\mathcal{P}$ i $\mathcal{P}'$, dla których wzajemny porządek wykonywanych przez nie operacji jest dowolny (zakładając brak operacji synchronizujących), choć w obrębie sa-

mych procesów kolejność operacji także ma charakter liniowy. Szczególnym przypadkiem pracy procesów współbieżnych jest *jednoczesne wykonanie procesów*, polegające na wykonaniu operacji procesów $\mathcal{P}$ i $\mathcal{P}'$ przez różne procesory fizyczne w tym samym czasie. Alternatywną techniką, imitującą jednoczesne wykonanie, jest realizacja ich operacji w *przeplocie*, poprzez naprzemienne wykonanie przez jeden procesor fizyczny. Program umożliwiający równoległe wykonanie procesów, zapewniający komunikację między nimi i ich synchronizację nazywany jest *programem współbieżnym* [53]. Jeżeli dostępna jest wystarczająca liczba procesorów do zapewnienia jednoczesnego wykonania wszystkich procesów programu, będziemy go nazywali *programem równoległym* [53]. Program współbieżny jest implementacją *algorytmu współbieżnego*.

Każdy proces posiada jeden bądź wiele strumieni instrukcji mogących podlegać przetwarzaniu w sposób współbieżny. Każdy z tych strumieni nazywany jest *wątkiem*, a przetwarzanie danych przy wykorzystaniu wielu wątków nazywane jest *wielowątkowością*. W dalszej części przyjęto, że powyższe definicje współbieżności procesów są prawdziwe także dla wątków.

Sposób organizacji pracy wątków i przesyłania danych między nimi jest wyznaczany przez przyjęty *model obliczeń równoległych*. Wyróżniamy m.in. *model z pamięcią wspólną* (model *PRAM*) oraz *model sieciowy*. Pierwszy z nich stanowi uogólnienie modelu obliczeń sekwencyjnych RAM. Opiera się on na założeniu, że każdy z synchronicznie pracujących procesorów fizycznych posiada dostęp do własnego, lokalnego obszaru pamięci oraz do wspólnego, umożliwiającego komunikację między procesorami obszaru globalnego. W zależności od przyjętej strategii obsługi jednoczesnego dostępu do współdzielonych komórek pamięci wyróżniamy różne warianty modelu PRAM. Spośród nich należy wyróżnić model *wyłączny odczyt, wyłączny zapis* (ang. *exclusive read, exclusive write* — *EREW*), w którym zakłada się brak możliwego jednoczesnego dostępu wielu wątków do współdzielonej pamięci. Takie ograniczenie najbardziej odpowiada rzeczywistej pracy systemów wieloprocesorowych. Model *jednoczesny odczyt, wyłączny zapis* (ang. *concurrent read, exclusive write* — *CREW*) dopuszcza możliwość równoległego odczytu współdzielonej pamięci. Mimo nierealności tego modelu będziemy przyjmowali, że jest on możliwy do osiągnięcia, gdyż sprzętowe realizacje dostępu do pamięci ukrywają przed programistą niejednoczesny odczyt danych. W modelu sieciowym pracujące synchronicznie lub asynchronicznie, wyposażone we własne obszary pamięci lokalnej procesory mogą komunikować się ze sobą przy pomocy *sieci połączeń* o określonej topologii. Topologia ma istotny wpływ na złożoność budowy komputera oraz sprawność komunikacji między procesorami, a w konsekwencji na szybkość wykonywania obliczeń.

Model obliczeń właściwy dla danego komputera wyznacza grupę algorytmów współbieżnych możliwych do wykonania. Model z pamięcią wspólną jest przeznaczony do wykonywania algorytmów współbieżnych *z użyciem pamięci wspólnej*.

Model sieciowy reprezentuje maszynę wykonującą algorytmy *z przesyłaniem wiadomości.*

Synchronizacja

Projektowanie algorytmów współbieżnych wiąże się z rozwiązywaniem szeregu typowych problemów. Wiele z nich dotyczy kwestii synchronizacji pracy wątków, szczególnie dostępu do niepodzielnych, współdzielonych przez nie zasobów. Problem *wzajemnego wykluczenia* dotyczy ograniczenia dostępu do takiego zasobu dla jednego wątku jednocześnie i polega na zagwarantowaniu sekwencyjności dostępów do takiego zasobu. Obszar programu wymagający wzajemnego wykluczenia jest nazywany *sekcją krytyczną*. Problem ten można rozwiązać przy pomocy *semaforów* [63]. Semaforem ς nazywamy licznik dostępnych zasobów wraz z przyporządkowaną kolejką wątków oczekujących na zwolnienie zasobu. Struktura danych kolejki jest zależna od potrzeb i często jest oparta na buforze FIFO. W przypadku implementacji wzajemnego wykluczenia stosuje się *semafory binarne*, mogące przyjąć dwie wartości: 0 oznaczającą zasób niedostępny oraz 1 oznaczającą zasób wolny. Na semaforze zdefiniowano dwie główne operacje:

- Wait(ς) — jeśli $\varsigma > 0$, wówczas wykonaj $\varsigma \leftarrow \varsigma - 1$, w przeciwnym przypadku wstaw bieżący wątek do kolejki i czekaj,
- Signal(ς) — wykonaj $\varsigma \leftarrow \varsigma + 1$, jeśli istnieje wątek w kolejce, to usuń go z kolejki i wznów.

Przyjmuje się, że operacje Wait i Signal są operacjami *atomowymi*, tzn. z punktu widzenia programu ich wykonanie nie zostanie przerwane ani wykonane równolegle w innym wątku. Wykonanie operacji Wait sygnalizuje zajęcie zasobu (z ewentualnym wcześniejszym oczekiwaniem na jego zwolnienie), a Signal jego zwolnienie. W dalszej części pracy będą rozważane wyłącznie semafory binarne.

Przydzielanie zadań procesorom

Podczas projektowania algorytmu równoległego należy zwrócić uwagę na sposób dekompozycji problemu, przyjąć poziom *rozdrobnienia* (*granulacji*) obliczeń, czyli rozmiaru zadań składowych problemu, zminimalizować dodatkowy koszt obliczeniowy związany z organizacją obliczeń równoległych oraz określić strategię przydziału zadań do procesorów. Przydział ten powinien równoważyć obciążenie procesorów tak, aby ograniczyć czas, gdy część spośród procesorów pozostaje bezczynna ze względu na brak obliczeń do wykonania, a pozostałe pracują.

Do sposobów *równoważenia obciążenia* należy m.in. *metoda scentralizowana* [53], której realizacja może polegać na wyznaczeniu co najmniej jednego procesora *nadrzędnego*, którego rolą jest wyznaczanie zadań. Wykonujące zasadnicze

obliczenia procesory *podległe*, w momencie zakończenia rozwiązywania podproblemu, wysyłają żądanie do procesora nadrzędnego o przydzielenie kolejnego zadania.

Złożoności obliczeniowe algorytmów współbieżnych

Pesymistyczną czasową złożonością obliczeniową algorytmu równoległego nazywamy wartość zgodną ze wzorem:

$$T(\rho, n) = \sup_{x \in \mathbb{X}} t(\rho, x), \tag{3.5}$$

gdzie ρ jest liczbą procesorów, n rozmiarem problemu, $\mathbb{X}$ zbiorem wszystkich możliwych parametrów algorytmu, a t jest liczbą kroków obliczeniowych potrzebnych do rozwiązania danej instancji problemu.

Niech $T^*(1, n)$ oznacza pesymistyczną złożoność czasową najlepszego znanego sekwencyjnego algorytmu rozwiązującego dany problem o rozmiarze n. Miarą skrócenia czasu obliczeń w efekcie wykorzystania algorytmu równoległego jest *przyspieszenie* zdefiniowane zgodnie ze wzorem [53]:

$$T_{\text{speedup}}(\rho, n) = \frac{T^*(1, n)}{T(\rho, n)}. \tag{3.6}$$

Ze względu na niezerowy koszt organizacji obliczeń równoległych, obejmujący czas synchronizacji, dodatkowych obliczeń, niepełne wykorzystanie czasu procesorów i inne trudności, w praktyce prawidłowa jest nierówność $T_{\text{speedup}}(\rho, n) < \rho$. Ponadto przyjmując jako s część obliczeń, które są możliwe do wykonania tylko w sposób sekwencyjny (*obliczenia inherentnie sekwencyjne*) prawdziwa jest zależność:

$$T_{\text{speedup}}(\rho, n) \leqslant \frac{1}{s + \frac{(1-s)}{\rho}}. \tag{3.7}$$

Nierówność (3.7) stanowi podstawę *prawa Amdahla* [27], określającego górne ograniczenie przyspieszenia osiągniętego w efekcie zwiększania liczby procesorów. Jest ono wyznaczane zgodnie z zależnością:

$$\lim_{\rho \to \infty} \frac{1}{s + \frac{(1-s)}{\rho}} = \frac{1}{s}. \tag{3.8}$$

Rozwiązania sprzętowe

Sprzętowa realizacja obliczeń równoległych może mieć wieloraki charakter. Obliczenia z pamięcią wspólną są rozwiązaniem typowym dla komputerów wieloprocesorowych oraz komputerów wyposażonych w procesory wielordzeniowe. Technika tego rodzaju jest powszechnie wykorzystywana i często jest zgodna z architekturą SMP lub NUMA [53]. Rozwiązania z pamięcią wspólną cechują się jednak

mniejszą skalowalnością niż systemy oparte na modelu sieciowym. W tym drugim przypadku z możliwości wykonywania algorytmu równoległego przy pomocy klastra komputerów bierze się potencjał uzyskania bardzo dużej mocy obliczeniowej.

Jedną z technik umożliwiających skrócenie czasu obliczeń poprzez lepsze wykorzystanie jednostek obliczeniowych jest *wielowątkowość współbieżna* (ang. *simultaneous multi-threading —SMT*), polegająca na wykonywaniu przez jeden fizyczny rdzeń procesora współbieżnie kilku (zazwyczaj dwóch) wątków. Z poziomu środowiska użytkownika jest on widoczny jako dwa niezależne *rdzenie logiczne*. W praktyce równoległość jest tylko pozorna i wynika z wykonywania przez rdzeń fizyczny rozkazów niektórych wątków w czasie, gdy pozostałe wątki są wstrzymane ze względu na oczekiwanie na przesłanie danych z pamięci.

Istotną tematyką jest wykonywanie obliczeń poza główną jednostką obliczeniową komputera (CPU). W ostatnich latach nastąpił gwałtowny rozwój architektur hybrydowych [176], w których część obliczeń jest delegowanych do urządzeń zewnętrznych, np. procesorów GPU. Taka koncepcja jest nazywana *obliczeniami ogólnego przeznaczenia na jednostkach GPU* (ang. *general-purpose computing on graphics processing units — GPGPU*) i wykorzystuje dużą moc obliczeniową współczesnych procesorów graficznych, osiągalną jednak tylko w zadaniach o określonych cechach oraz po rozwiązaniu trudności dotyczących niestandardowego sposób wykorzystania równoległości przez GPU. Coraz częściej moc obliczeniowa CPU jest zwiększana także specjalizowanymi koprocesorami, charakteryzującymi się dużą mocą obliczeniową, ale pozbawionymi części wad GPGPU. Zastosowanie zewnętrznych urządzeń obliczeniowych niesie za sobą również nowe wyzwania, do których należą konieczność posiłkowania się specjalizowanymi narzędziami, zastosowania odpowiednich metod tworzenia oprogramowania czy uwzględnienia ograniczeń wynikających ze zbyt małej przepustowości połączenia GPU lub koprocesora z procesorem CPU [176].

3.4 Algorytmy zliczania k-merów

Wiele algorytmów oraz potoków przetwarzania informacji genetycznej obejmuje proces *zliczania k-merów* (ang. *k-mer counting*). Stanowi on zazwyczaj wstępny etap analizy wejściowej informacji. Dla wejściowego genomu $\mathcal{G}$ lub odczytów z sekwencjonowania $\mathcal{R}$ algorytm zliczania generuje bazę danych par k-merów i ich liczebności. Baza ta posiada format umożliwiający wykonanie określanych operacji na bazie, do których wliczają się iteracja po zbiorze k-merów, uzyskanie ich liczebności, sprawdzenie obecności określonego k-meru czy wykonanie zapytania o jego liczebność [59].

Baza k-merów pełni rolę modelu danych wejściowych, charakteryzującego się użytecznymi właściwościami. Może być wykorzystana w rozmaity sposób. Wyodrębnione z danych k-mery mogą służyć do budowy indeksów genomów lub

odczytów [165] — w takim przypadku określona sekwencja długości k stanowi także zapytanie indeksu. W procesie asemblacji zestaw k-merów może umożliwiać wyszukiwanie odczytów, których fragmenty w obrębie genomu nakładają się [141]; takie związki między odczytami są reprezentowane w formie grafów de Bruijna. Jednocześnie użyteczną informację stanowią liczebności k-merów. Są one podstawą do eliminacji błędów sekwencjonowania podczas asemblacji [147], pozwalają na badanie ekspresji genów w odczytach techniki RNA-Seq [87] albo na oszacowanie rozmiaru genomu w oparciu o odczyty techniki WGS [141].

Określenie użytecznej wartości k nie jest zadaniem trywialnym [45]. Musi ono uwzględniać liczne kwestie, do których należą docelowe zastosowanie bazy, rozmiar analizowanego genomu (bez względu na to, czy przedmiotem zliczania są odczyty, czy genomy), oczekiwana liczba błędów sekwencjonowania, charakterystyka sekwencji (zazwyczaj trudna do sformalizowania i nieznana *a priori*) i inne kwestie.

3.4.1 Sformułowanie problemu

Zliczaniem k-merów nazywamy zadanie uzyskania zbioru $\mathscr{K}$ par uporządkowanych (κ_i, η_i), gdzie κ_i jest k-merem, a η_i jest liczbą jego wystąpień (liczebnością) w wejściowym zbiorze sekwencji nukleotydowych $\mathcal{W}$. W zależności od problemu $\mathcal{W} = \{\mathfrak{c}_j\}$ dla każdego $\mathfrak{c}_j \in \mathcal{G}$ albo $\mathcal{W} = \{\mathfrak{r}_j\}$ dla każdej pary $(\mathfrak{r}_j, \mathfrak{p}_j) \in \mathcal{R}$. Zliczanie k-merów obejmuje:

1. wyodrębnienie wszystkich k-merów κ_ι nad alfabetem Σ_N, tzn. każdej sekwencji $\kappa_0 = \mathfrak{w}_j[0, k-1], \kappa_1 = \mathfrak{w}_j[1, k], \ldots, \kappa_{\ell-k} = \mathfrak{w}_j[\ell-k, \ell-1]$ dla każdego $\mathfrak{w}_j \in \mathcal{W}$ długości $\ell = |\mathfrak{w}_j|$; κ_ι są elementami wielozbioru $\mathcal{K}_\mathbb{N}$ mocy $\bar{k}_\mathrm{N}$,

2. uzyskanie par (κ_i, η_i) dla każdego, występującego w zbiorze $\mathcal{K}_\mathbb{N}$ przynajmniej raz, k-meru κ_i nad alfabetem Σ; pary te są elementami zbioru $\mathscr{K}$.

Zbiór $\mathscr{K}$ nazywamy *zbiorem k-merów*, zbiór $\mathcal{K}_\mathrm{un} = \{\kappa_i\}$ dla każdego κ_i w parach $(\kappa_i, \eta_i) \in \mathscr{K}$ nazywamy *zbiorem k-merów unikalnych*. Powyższe definicje zakładają, że strategia uwzględnienia obecności symbolu N w danych wejściowych polega na eliminacji k-merów zawierających ten symbol. Niektóre algorytmy zliczania stosują inne podejścia [59]. Ze względu na częsty brak informacji o orientacji k-merów narzędzia zliczające zazwyczaj domyślnie zliczają k-mery w formie kanonicznej.

3.4.2 Wybrane algorytmy zliczania k-merów

Koncepcyjnie opis zliczania k-merów jest prosty. Można zaproponować jego trywialne rozwiązanie polegające np. na utworzeniu tablicy mieszającej, której kluczami byłyby poszczególne sekwencje k-merów, a każdej sekwencji byłaby przy-

porządkowana liczebność równych jej k-merów. Ze względu na często ogromny rozmiar danych wejściowych oraz duże wartości k (np. $k = 30$) rozwiązanie takie byłoby technicznie niesatysfakcjonujące [60, 141]. Wiązałoby się bowiem ze zbyt dużym zapotrzebowaniem na pamięć oraz czas zliczania. W rezultacie opracowano szereg specjalizowanych algorytmów zliczania k-merów, udostępnianych w formie gotowych narzędzi. Należą do nich m.in. Jellyfish [141], DSK [173], KMC [59, 60, 117], Tallymer [122], BFCounter [147], khmer [147] i inne. Poniżej przedstawiono uproszczony opis działania wybranych algorytmów.

Jellyfish

Na wysokim poziomie ogólności algorytm Jellyfish jest oparty na przedstawionej wyżej idei wykorzystującej tablicę mieszającą. Sekwencje poszczególnych k-merów są kluczami przekazywanymi do funkcji mieszającej, a ich liczebności stanowią wartości umieszczone w tablicy. Kolizje są eliminowane zgodnie ze strategią adresowania otwartego. Modyfikacja elementów tablicy odbywa się przy pomocy atomowego rozkazu procesora CAS umożliwiającego współbieżną modyfikację tablicy bez potrzeby określania sekcji krytycznych oraz czasochłonnej synchronizacji pracy wątków.

Funkcja mieszająca została dobrana w sposób zapewniający jej „częściową odwracalność" polegającą na możliwości uzyskania części informacji o argumencie funkcji w oparciu o jej wartość. Dzięki temu zażegnano potrzebę przechowywania w tablicy pełnych postaci k-merów, gdyż mogą być one określone w oparciu o pozycję danego elementu w tablicy mieszającej (wartość funkcji mieszającej) oraz dodatkowe dane. Tymi danymi są bity uzupełniające informacje, wyeliminowane przez operator modulo funkcji mieszającej, a także wartości dotyczące ponownego wyliczenia wartości funkcji mieszającej w celu rozwiązania kolizji. Są one przechowywane w tablicy mieszającej, wymagając jednak znacznie mniejszej ilości pamięci niż pełna postać k-merów.

Algorytm Jellyfish wykonuje zliczanie w czasie $O(\bar{k}_{\mathrm{N}})$. Jeśli ilość dostępnej pamięci jest zbyt mała do przechowania poddawanej restrukturyzacji tablicy mieszającej, wówczas jest ona zapisywana w pamięci zewnętrznej, a w pamięci głównej tworzona jest nowa tablica. Jeśli liczba takich zapisów jest proporcjonalna do $\bar{k}_{\mathrm{N}}$, wówczas czas ten ulega pogorszeniu do $O(\bar{k}_{\mathrm{N}} \log \bar{k}_{\mathrm{N}})$. Zapisane tablice są ostatecznie poddawane scaleniu. Proces scalania, obejmujący także sortowanie elementów, jest przeprowadzany ze złożonością liniową względem łącznego rozmiaru tablic mieszających.

Uzyskana baza danych posiada format umożliwiający wykonywanie zapytań o obecność i liczebność zadanych k-merów. Zapytania są realizowane algorytmem opartym na wyszukiwaniu połówkowym. Wczesne wersje algorytmu Jellyfish obsługiwały dwie formy liczebności k-merów. Poza standardowym zliczaniem istniała możliwość określenia przyporządkowania k-merom wartości η_i zależnych nie

tylko od liczby wystąpień, ale także od ich jakości określanej przez współczynniki jakości q.

DSK

Założeniem algorytmu DSK jest znaczące ograniczenie zapotrzebowania na pamięć przy dopuszczeniu intensywnego wykorzystania systemu pamięci zewnętrznej, np. przestrzeni dyskowej. Zasada działania algorytmu umożliwia określenie górnego ograniczenia na ilość wykorzystywanej pamięci operacyjnej $\mu_{\max}$ oraz przestrzeni dyskowej $\delta_{\max}$, umożliwiając przyjęcie kompromisu w stosunku do czasu obliczeń.

Pełna analiza danych wejściowych jest przeprowadzana wielokrotnie. Liczba powtórzeń jest zależna od wartości $\delta_{\max}$ oraz $\bar{k}_{\mathrm{N}}$. Podczas każdej iteracji, wybrane w oparciu o swoją wartość funkcji mieszającej, wyodrębnione z danych wejściowych k-mery zostają zapisane do jednego z $\bar{\delta}$ plików dyskowych. Dobór docelowego pliku także jest zależny od wartości funkcji mieszającej. Po zakończeniu procesu kolejne pliki dyskowe są wczytywane, a zliczanie obecnych w nich k-merów odbywa się poprzez umieszczenie ich w tablicy mieszającej z adresowaniem otwartym.

Przy postawieniu upraszczającego założenia, że rozkład wartości funkcji mieszającej jest jednostajny, złożoność algorytmu wynosi $O\left(\frac{\bar{k}_{\mathrm{N}}^2 2\lceil\log_2 2k\rceil)}{\delta_{\max}}\right)$. DSK nie udostępnia możliwości wykonywania zapytań w uzyskanej bazie k-merów, jednak ze względu na przyjęcie jako typu pliku popularnego formatu HDF5 istnieje możliwość wykorzystania zewnętrznych narzędzi obsługujących ten format.

KMC

Obliczenia przy wykorzystaniu przestrzeni dyskowej mogą być wykonywane także przy pomocy algorytmu KMC [59, 60, 117]. Zasada jego działania jest zbliżona do DSK, jednak podstawową, mającą wpływ na zapotrzebowanie na pamięć oraz czas obliczeń różnicą w stosunku do niego jest sposób doboru pliku dyskowego, do którego dany k-mer zostanie zapisany. Metoda ta ewoluowała w kolejnych wersjach algorytmu.

Najnowsze podejście wybiera plik posiłkując się ideą *sygnatur*, będących specjalizacją *minimizerów* [174, 175]. Minimizerem nazywamy taki k'-mer ($k' < k$) będący podsłowem określonego k-meru, który spośród wszystkich jego k'-merów jest leksykograficznie najmniejszy. Funkcja mieszająca dobierająca plik przyjmuje minimizer k-meru, jednak ze względu na dużą nierównomierność liczebności różnych minimizerów, mogącą skutkować znacznym rozmiarem niektórych plików, odrzucono możliwość przekazania do funkcji minimizerów o wybranych postaciach (np. zbudowanych wyłącznie z symboli A). Minimizery nienależące do tej problematycznej grupy są nazywane sygnaturami. Postać funkcji mieszającej

jest określana heurystycznie na podstawie próbki danych poddawanej analizie we wstępnym etapie pracy algorytmu.

Super k-merem nazywamy k''-mer ($k'' \geqslant k$), którego wszystkie podsłowa długości k (k-mery) posiadają ten sam minimizer [133] lub, w przypadku KMC, tę samą sygnaturę. KMC wyodrębnia oraz umieszcza w plikach super k''-mery. Proces zliczania, w odróżnieniu od DSK, polega na wczytaniu poszczególnych plików oraz wykonaniu sortowania zawartych w nich (k, χ)-merów. Są to $k + \chi'$-mery, gdzie $\chi' \in \{0, 1, \ldots, \chi\}$ dla zadanego χ. Minimalizacja liczby wyodrębnionych (k, χ)-merów pozwala na ograniczenie zapotrzebowania na pamięć operacyjną i przyspieszenie sortowania k-merów. Sortowanie jest wykonywane specjalizowanym algorytmem sortowania pozycyjnego, działającym w formule od najbardziej znaczącego znaku (ang. *most-significant-digit — MSD*) [116]. Po przeprowadzeniu sortowania k-mery o tych samych sekwencjach znajdują się na sąsiednich pozycjach wektora, umożliwiając łatwe określenie liczebności poszczególnych k-merów.

Złożoność obliczeniowa procesu zliczania KMC wynosi [115]:

$$T^{\text{KMC3}}_{\text{równoległy}}\left(\mathfrak{n}, \ell, k, k', n_{\text{LUT}}, \bar{\delta}, p, \text{dep}\right) = O\left(\frac{\mathfrak{n}\ell k}{k - k'} + \frac{\mathfrak{n}\ell k}{p} + \bar{\delta} 4^{\max(n_{\text{LUT}}, k')} + \frac{\mathfrak{n}\ell k}{\text{dep}}\right), \tag{3.9}$$

gdzie (uwzględniono zestaw oznaczeń przyjętych w tej pracy): $\mathfrak{n}$ — liczba odczytów, ℓ — długość odczytów, k' — długość minimizera, n_{LUT} — wewnętrzny parametr algorytmu, $\bar{\delta}$ — liczba plików tymczasowych, p — liczba wątków przetwarzania równoległego, dep — głębokość sekwencjonowania.

Baza k-merów jest zapisywana w postaci umożliwiającej połówkowe wyszukanie zadanego k-meru oraz uzyskanie jego liczebności. Podstawowa implementacja algorytmu udostępnia interfejs programistyczny (ang. *application programming interface — API*) pozwalający na wykonanie takich zapytań z poziomu innego algorytmu oraz iterację po zbiorze k-merów. Proces zliczania pozwala na wprowadzenie limitu wykorzystania pamięci operacyjnej $\mu_{\max}$, jednakże nie obowiązuje on tylko na etapie zliczania i nie jest dostępny podczas wykonywania zapytań do bazy danych przy pomocy API.

Rozdział 4

Algorytmy korekcji odczytów genomów

Wynalezienie i ciągły rozwój metod sekwencjonowania drugiej generacji, a także opracowanie oprogramowania umożliwiającego wykorzystanie odczytów z sekwencjonowania do rozwiązania konkretnych problemów sprawiły, że sekwencjonowanie kwasów nukleinowych stało się często wykorzystywaną metodą badawczą. Obecnie sekwencjonowaniu poddawane są genomy rozmaitych organizmów [97].

Z drugiej strony, charakterystyka odczytów NGS skutkuje pojawieniem się nowych wyzwań, wynikających z mniejszej ich długości oraz występowaniem większej liczby błędów niż w metodzie Sangera [97, 137, 179, 189]. Błędy te mają wpływ na różne aspekty przetwarzania takich danych. W literaturze ich znaczenie najczęściej podkreślane jest w kontekście asemblacji oraz mapowania odczytów, a w niewielkim stopniu również detekcji wariantów. Do negatywnych zjawisk będących efektem błędów obecnych w odczytach, a potencjalnie możliwych do zredukowania należą:

- zwiększenie zapotrzebowania na pamięć oraz czas obliczeń [189, 192, 209],
- komplikacja algorytmów przetwarzania danych z sekwencjonowania [25, 110, 138, 142, 143, 210],
- pogorszenie jakości uzyskiwanych wyników (np. błędne dopasowania odczytów, nieprawidłowe połączenia między odczytami w asemblacji) [89, 97, 110, 137, 142, 179, 179, 187, 192, 209],
- niepełne wykorzystanie danych zawartych w odczytach [179].

Popularność technik NGS oraz konieczność ograniczenia powyższych problemów umotywowały opracowanie wielu algorytmów umożliwiających eliminację

jak największej liczby błędów, mających swoje źródło w różnych etapach procesu sekwencjonowania. Algorytmy te w wielu przypadkach wykorzystują redundancję danych, będącą efektem odpowiednio dużej głębokości sekwencjonowania [89, 110], a ujawniającą się w postaci dostatecznego pokrycia odczytami różnych fragmentów genomu, tj. liczby odczytów, które zostały uzyskane z danego fragmentu genomu. Działanie algorytmów, oparte na różnych strategiach, obejmuje modyfikację odczytów (korekcję błędów) przy wykorzystaniu wiedzy obecnej w innych, pozbawionych błędów odczytach uzyskanych z tego samego miejsca genomu [128]. Ponadto wykorzystywane są informacje zawarte we współczynnikach jakości, parametrach algorytmu, wyznaczonej charakterystyce urządzenia sekwencjonującego, odczytach uzyskanych w innych eksperymentach sekwencjonowania tej samej biblioteki DNA (np. w procesie *korekcji krzyżowej*) oraz uzyskane z innych źródeł.

W ogólnym przypadku nie jest dostępna informacja o rejonie genomu, z którego odczyt został uzyskany [138]. Często sama postać genomu także nie jest znana [179, 209]. Tym samym jednoznaczna ocena zasadności i poprawności przeprowadzonej korekcji nie jest możliwa [154], liczba potencjalnych zmian wprowadzonych do odczytu w celu poprawy jego jakości może być bardzo duża, a wiedza dotycząca charakterystyki błędów danego urządzenia ma charakter statystyczny. Ponadto proces korekcji jest skomplikowany w wyniku istnienia obszarów genomu o niskim pokryciu odczytami, obecności odczytów o znacznej liczbie błędów oraz odczytów możliwych do dopasowania do różnych obszarów genomu [25]. Algorytm powinien być wyposażony w strategię wyboru jednego spośród wielu potencjalnych ciągów modyfikacji korygujących odczyt (*ścieżek korekcji*), zapewniającego z największym prawdopodobieństwem prawidłową korekcję. W efekcie metody poszukiwania prawidłowej postaci odczytu są oparte na przesłankach intuicyjnych. Tym samym wszystkie algorytmy korekcji wykorzystują mechanizmy heurystyczne [179].

4.1 Przegląd algorytmów korekcji

W niniejszym podrozdziale omówiono wybrane spośród algorytmów korekcji odczytów. Ze względu na dużą liczbę istniejących rozwiązań skupiono się na algorytmach przeznaczonych dla odczytów z sekwencjonowania Illumina uzyskanych w strategii WGS. W dalszej części krótko omówiono algorytmy korekcji odczytów innych typów.

4.1.1 Systematyka

W pracy [209] zaproponowano systematykę algorytmów korekcji odczytów opartą na uwzględnieniu ogólnej zasady działania algorytmu. Wyróżnia ona trzy grupy:

- oparte na *spektrum k-merów* (ang. *k-spectrum-based*),
- oparte na drzewach/tablicach sufiksów (ang. *suffix tree/array-based*),
- oparte na dopasowywaniu wielu sekwencji (ang. *multiple sequence alignment-based — MSA-based*).

W pracy [97] wyróżniono także kategorię algorytmów wykorzystujących ukryte modele Markowa. Jednocześnie spotykane są rozwiązania hybrydowe, rozwiązania charakteryzujące się znaczną odmiennością w porównaniu do innych przedstawicieli swojej grupy oraz trudne do zaklasyfikowania do dowolnej z powyższych grup. Zjawisko hybrydowości można dostrzec w pewnym stopniu w wielu algorytmach. Przykładowo, w algorytmie Fiona [187] odczyty z sekwencjonowania reprezentowane są przy pomocy struktury sufiksowej. Jednocześnie korekcja jest wykonywana poprzez dopasowywanie sekwencji różnych odczytów w sposób podobny jak w algorytmach opartych na dopasowywaniu wielu sekwencji.

Zasada działania większości algorytmów zakłada prawdziwość trzech założeń [123]:

- błędy sekwencjonowania pojawiają się rzadko,
- pokrycie genomu odczytami jest równomierne,
- błędy pojawiają się z podobnym prawdopodobieństwem w różnych rejonach sekwencjonowanej próbki.

Należy jednak zauważyć, że ich prawdziwość jest słuszna tylko częściowo. Odczyty, w zależności od zastosowanego urządzenia sekwencjonującego i innych czynników, zawierają błędy systematyczne, charakterystyczne motywy błędów lub mogą być uzyskane z określonych fragmentów genomów rzadziej lub częściej niż z innych. W związku z tym algorytmy wykorzystują rozmaite strategie dopuszczające odstępstwa od takich założeń. Strategie te nierzadko stanowią cechę charakterystyczną danego algorytmu, wyróżniającą go w swojej grupie.

Zaprezentowane niżej algorytmy są przeznaczone do korekcji odczytów uzyskanych zgodnie ze strategią WGS. Dla pozostałych strategii, np. RNA-Seq, często niespełnione jest założenie dotyczące równomierności pokrycia genomu odczytami [123].

Algorytmy oparte na spektrum k-merów

Spektrum k-merów $\mathcal{K}_{\text{cut}}$ stanowi model sekwencji reprezentowanej przez odczyty. Jest definiowane jako zbiór k-merów posiadających liczebność nie mniejszą od określonego *progu obcięcia* (ang. *cutoff threshold*) η_{cut} [97, 137], tj. $\mathcal{K}_{\text{cut}} = \{\kappa_i : (\kappa_i, \eta_i) \in \mathscr{K} \wedge \eta_i \geqslant \eta_{\text{cut}}\}$, gdzie $\mathscr{K}$ jest zbiorem k-merów uzyskanych w wyniku procesu zliczania k-merów. Koncepcja algorytmów opartych na spektrum

k-merów została wprowadzona w asemblerze EULER [163] i później znalazła zastosowanie w niektórych innych asemblerach [209]. Obecnie jest dominującą techniką wykorzystywaną w procesie korekcji odczytów [123].

Algorytmy korekcji wykorzystujące spektrum k-merów są oparte na założeniu, że niewielka odległość Hamminga między wybranym k-merem zawartym w spektrum a innym k-merem, będącym fragmentem odczytu, sugeruje, że reprezentują one ten sam fragment genomu. Niezerowa odległość wskazuje obecność co najmniej jednego błędu. Błędy są eliminowane poprzez wprowadzanie modyfikacji do odczytów w celu uzyskania odczytów zawierających k-mery konsensusowe, tzn. o takiej sekwencji, aby posiadały w spektrum swój identyczny co do sekwencji odpowiednik. Założenie o wspólnym pochodzeniu podobnych (tj. o niewielkiej odległości Hamminga) k-merów nie zawsze jest prawdziwe, co może być efektem diploidalności lub poliploidalności organizmu. W tej sytuacji mogą występować różnice między sekwencjami genomu w chromosomach homologicznych, na tyle niewielkie, że uzyskane z nich k-mery zostaną sklasyfikowane przez algorytm jako podobne. Znaczenie dla prawdziwości założenia może mieć też potencjalne podobieństwo różnych fragmentów genomów organizmu, jednak odpowiedni dobór wartości k może spowodować, że spełnialność tego założenia będzie w tym przypadku satysfakcjonująca [209].

Algorytmy tej grupy zwykle klasyfikują k-mery do jednej z dwóch grup: uznawanych za pochodzące z bezbłędnych fragmentów odczytów (k-mery *zaufane* — ang. *solid*, *strong*, *trusted*) albo z fragmentów zawierających przynajmniej jeden błąd (k-mery *niezaufane* — ang. *weak*, *untrusted*). Warunkiem zaklasyfikowania k-meru do zbioru zaufanych jest zwykle jego obecność w spektrum, stąd często początkowym etapem działania algorytmów jest przeprowadzenie procesu zliczania k-merów oraz wyznaczenie progu obcięcia η_{cut}, stanowiącego graniczną wartość liczebności rozdzielającą zbiór k-merów zaufanych i niezaufanych [123].

Algorytmy oparte na drzewach/tablicach sufiksów

Wybrane algorytmy jako główną strukturę danych wykorzystują jedną ze struktur sufiksowych. Wejściowe odczyty są przedstawiane w postaci — często uogólnionej — tablicy lub drzewa sufiksów, wraz z dodatkowymi informacjami (np. liczbą odczytów zawierających sekwencję reprezentowaną danym węzłem [179]). Struktury tego typu są kosztowne pamięciowo, jednak pozwalają na bardzo szybkie wyszukiwanie oraz zastosowanie oligomerów różnych długości wraz z różnymi progami obcięcia. Często jednak struktura danych jest zoptymalizowana poprzez ograniczenie użytecznego zakresu długości oligomerów (zakresu wartości k) możliwych do odczytania ze struktury danych [123]. Grupa ta stanowi uogólnienie grupy algorytmów opartych na spektrum k-merów [25].

Przykładowy proces detekcji i korekcji błędów w oparciu o drzewo sufiksów polega na przechodzeniu drzewa w kierunku od korzenia do liści kontrolując li-

czebności zawartych w nim oligomerów lub liczby odczytów, z których dany oligomer został uzyskany. Potencjalny błąd jest wykrywany, gdy liczba wystąpień ciągu reprezentowana przez pewien węzeł v jest mniejsza od określonego progu. W takim przypadku następuje porównanie poddrzewa węzła v:

- w przypadku substytucji: z poddrzewami rodzeństwa v_i węzła v,
- w przypadku insercji: z poddrzewami rodzeństwa v_i rodzica węzła v,
- w przypadku delecji: z poddrzewami potomków v_i węzła v.

Jeżeli porównywane poddrzewa są takie same, następuje scalenie ich z poddrzewem węzła v oraz wykonanie modyfikacji odczytów, których sekwencje odpowiadają poddrzewu węzła v.

Algorytmy oparte na dopasowywaniu wielu sekwencji

Zadanie *dopasowywania* (*uliniowienia*) dwóch sekwencji polega na wzajemnym dopasowywaniu symboli spośród dwóch sekwencji nukleotydowych (lub sekwencji aminokwasów) na zasadzie ich odpowiedniości [166]. Proces dopasowywania musi uwzględniać możliwość występowania znacznych różnic między sekwencjami. Dopasowywanie sekwencji jest szeroko zbadanym problemem, często rozwiązywanym algorytmem Needlemana-Wunscha [159]. Zadanie może być uogólnione na przypadek, gdy liczba sekwencji jest większa od dwóch, jednak jest ono problemem należącym do klasy NP-trudnych.

Algorytmy korekcji tej grupy przeprowadzają wzajemne dopasowywanie odczytów, określając w ten sposób konsensusową postać ich zakładkujących się fragmentów, tj. fragmentów przypuszczalnie uzyskanych z tego samego fragmentu genomu. Obecnie, ze względu na dużą liczbę przetwarzanych odczytów, stosuje się podejście polegające na wstępnym doborze odczytów, które zostają poddane dopasowywaniu, zwykle poprzez wyszukanie odczytów posiadających wspólne podciągi (k-mery) [123]. Ponadto pojedyncze dopasowywanie jest często ograniczone do dwóch sekwencji [123, 209], np. w algorytmie Coral [179] po dopasowaniu dwóch odczytów wybranych w procesie preselekcji ich postać konsensusowa jest porównywana z potencjalnym kolejnym odczytem.

Algorytmy wykorzystujące inne techniki

Przedstawione wyżej koncepcje nie wyczerpują zestawu metod korekcji odczytów. Odmiennym, interesującym rozwiązaniem jest zastosowanie *ukrytych modeli Markowa*. Do wyjaśnienia tego pojęcia niezbędne jest zdefiniowanie łańcucha Markowa. *Łańcuchem Markowa* (*dyskretnym procesem Markowa*) nazywany jest dyskretny proces stochastyczny (rodzina zmiennych losowych Y_i, których realizacje y_i przyjmują różne wartości w funkcji czasu), którego bieżący stan (realizacja

zmiennych losowych) jest zależny wyłącznie od stanu poprzedniego [214]. Innymi słowy funkcja rozkładu prawdopodobieństwa P tych zmiennych losowych spełnia warunek:

$$P(Y_n = y_n | Y_{n-1} = y_{n-1}, Y_{n-2} = y_{n-2}, \ldots, Y_1 = y_1) = \\ = P(Y_n = y_n | Y_{n-1} = y_{n-1}). \tag{4.1}$$

Ukrytym modelem Markowa nazywamy łańcuch Markowa, którego stan w określonym momencie nie jest znany, lecz dostępny jest zbiór wartości zmiennych losowych Y_i^e nazywanych *emisjami* [167]. Wartości zbioru realizacji emisji zależą od pewnego warunkowego rozkładu prawdopodobieństwa emisji $P'(Y_1^e, Y_2^e, \ldots, Y_n^e | Y_1, Y_2, \ldots, Y_n)$.

Tym samym ukryte modele Markowa mogą być wykorzystywane do reprezentacji zjawisk charakteryzujących się pewnym zbiorem stanów, który jest obserwowalny niebezpośrednio. Z zagadnieniem tym związany jest problem określenia parametrów modelu w oparciu o ciąg emisji, który może być rozwiązany algorytmem Bauma-Welcha [167, 214]. Innym typowym problemem jest znalezienie najbardziej prawdopodobnego ciągu stanów w oparciu o ciąg emisji, które można rozwiązać algorytmem Viterbiego [167, 214] lub algorytmem Fano [210].

Metoda oparta na ukrytych modelach Markowa została wykorzystana tylko w algorytmie PREMIER [210] oraz algorytmie korekcji odczytów RNA-Seq SEECER [126], stąd w niniejszej pracy nie będzie ona wyodrębniana jako osobna grupa algorytmów.

Podobnym rozwiązaniem, tj. opartym na założeniu, że proces sekwencjonowania posiada ukryty zbiór stanów, charakteryzuje się metoda korekcji zastosowana w algorytmie SAMDUDE [79]. W tym przypadku proces sekwencjonowania jest modelowany jako kanał komunikacyjny będący źródłem szumu (błędów) w sekwencji nukleotydowej. Podobnie jak w algorytmie PREMIER metoda korekcji polega na wstępnym określeniu parametrów modelu (w tym przypadku kanału) oraz uzyskaniu w oparciu o niego oryginalnej sekwencji.

4.1.2 Pozostałe cechy algorytmów

Nierzadko algorytmy korekcji charakteryzują się cechami, które mogą stanowić podstawę wyznaczania innych klasyfikacji. Szczególnie istotną własnością jest rodzaj korygowanych błędów. Część spośród istniejących rozwiązań uwzględnia wyłącznie eliminację błędów typu substytucja, podczas gdy inne w sposób jawny korygują także błędy typu indel. Należy zwrócić uwagę, że algorytmy pierwszej z tych grup w niektórych przypadkach także są w stanie wykryć oraz skorygować błędy typu indel poprzez modyfikację wielu symboli od miejsca wystąpienia błędu aż do jednego z końców odczytu [152]. Jest to jednak zjawisko przypadkowe, obarczone licznymi ograniczeniami.

Wybrane algorytmy korekcji są dostosowane do odczytów sekwenatorów określonego typu. Jest to efekt bardzo odmiennej charakterystyki odczytów różnych sekwenatorów, szczególnie ich długości, dominującego typu błędów, charakterystyki błędów (np. prawdopodobieństwa pojawienia się błędu lub możliwej długości błędu typu indel) oraz głębokości sekwencjonowania. Jednocześnie niektóre algorytmy są zaprojektowane do obsługi odczytów różnych typów [25, 89].

Część spośród omawianych rozwiązań wykorzystuje rozmaite strategie mające na celu poprawę jakości odczytów w przypadku niepowodzeniu korekcji. Należą do nich np. eliminacja skrajnych symboli odczytów (*obcinanie* — ang. *trimming*) [209] (które stanowi alternatywę dla prostego usuwania skrajnych symboli, głównie z końca $3'$, jeżeli ich współczynniki jakości posiadają wartości poniżej zadanego progu, co poprawia jakość, ale powoduje utratę danych [209]), np. [98, 110, 138, 142], usuwanie niekorygowalnych odczytów [110], przenoszenie odczytów słabej jakości do osobnego pliku [143].

Interesującym rozwiązaniem jest dopuszczenie możliwości przeprowadzania *korekcji krzyżowej* (ang. *cross-correction*) [89], która umożliwia wykorzystanie zalet odczytów uzyskanych różnymi technikami sekwencjonowania. Przykładowo, może ono polegać na wykonaniu korekcji długich odczytów sekwenatorów Roche 454 lub sekwenatorów TGS przy użyciu informacji (np. zbioru k-merów) uzyskanych z odczytów sekwenatorów Illumina, charakteryzujących się mniejszą liczbą błędów, ale posiadającymi niewielką długość. Podobna strategia, na przykładzie odczytów TGS, zostanie omówiona w podrozdziale 4.1.5.

4.1.3 Wybór algorytmów korekcji odczytów Illumina

W niniejszym podrozdziale zawarto uszczegółowiony opis wybranych algorytmów korekcji. Ich dobór został przeprowadzony w oparciu o aktualność algorytmu, jego popularność, jakość uzyskiwanych wyników w eksperymentach opisanych w literaturze, możliwość zastosowania danego algorytmu do korekcji odczytów współczesnych urządzeń sekwencjonowania NGS. Postawiono wymaganie uniwersalności algorytmu. Położono nacisk na znalezienie przedstawicieli wszystkich grup algorytmów, starając się przedstawić jak największą różnorodność podejścia do zadania korekcji odczytów.

Jako dokładne kryterium wyboru algorytmu przyjęto, że wydanie opisującej go publikacji powinno nastąpić nie wcześniej niż w 2012 roku oraz powinna posiadać nie mniej niż 50 cytowań wg bazy Google Scholar, opierając się na stanie aktualnym w styczniu 2020 roku. Uwzględniono algorytmy dostosowane do korekcji odczytów długości ok. 100 pz–150 pz. Dodatkowo zawarto opis algorytmu Quake [110] ze względu na jego znaczną popularność (blisko 600 cytowań) oraz algorytmu Karect [25] ze względu na sygnalizowane w literaturze bardzo dobre wyniki jego pracy w kontekście poprawy jakości asemblacji [100]. Zawarto również opis algorytmu SAMDUDE [79], charakteryzującego się znaczną odmiennością

od pozostałych, a zatem nienadającego się do zaklasyfikowania do żadnej z wyżej wymienionych grup, oraz algorytmu PREMIER [210], jako przedstawiciela algorytmów wykorzystujących ukryte modele Markowa. Jednocześnie zrezygnowano z dokładnego opisu algorytmu QuorUM [142] ze względu na sygnalizowane w literaturze trudności: dużą liczbę przypadków obcinania odczytów w wyniku działania tego algorytmu [128] oraz częste przypadki nieuzyskania wyjściowych odczytów [100].

Poniższe zestawienie zawiera algorytmy w chronologicznej kolejności ukazania się wprowadzającej go publikacji. Zostało ono podsumowane w tabeli 4.1, obejmującej m.in. obligatoryjne parametry algorytmów; jeżeli algorytm przyjmuje przybliżoną wartość pewnej wielkości, oznaczono ją przy pomocy notacji „daszka", np. $\widehat{\ell}_G$ oznacza parametryzację algorytmu przybliżonym rozmiarem genomu.

Quake

Działanie algorytmu Quake [110] w pierwszym kroku obejmuje wykonanie zmodyfikowanego zliczania k-merów, w którym zamiast określenia liczebności sumowane są wagi poszczególnych k-merów. Wagi te są iloczynami współczynników błędów $p[i]$ pozycji odpowiednich symboli uzyskanych ze źródłowych odczytów. Za zliczanie k-merów jest odpowiedzialny algorytm Jellyfish [141]. Tym samym podstawowym parametrem algorytmu Quake jest długość k-meru. Algorytm jest przeznaczony do korekcji głównie odczytów sekwenatorów Illumina.

Określenie progu obcięcia jest wykonywane w oparciu o model wag k-merów. Autorzy algorytmu przyjęli, że wagi k-merów prawidłowych są dobrze modelowane rozkładem będącym połączeniem rozkładu normalnego oraz rozkładu dzeta. Z kolei jako model wag k-merów błędnych przyjęto rozkład gamma. Po postawieniu powyższych założeń próg obcięcia jest dobierany algorytmem optymalizacyjnym BFGS. Podstawowa baza danych wykorzystywana przez algorytm jest wektorem wartości binarnych reprezentujących informację o zaufaniu k-meru przyporządkowanego danej pozycji wektora.

Praca zasadniczej części algorytmu polega na wyborze z kolejnych odczytów k-merów niezaufanych oraz wyznaczaniu w oparciu o ich obecność błędnych regionów. W każdym takim regionie następuje znalezienie nukleotydu o najniższym współczynniku jakości oraz wstawienie do kolejki priorytetowej wszystkich potencjalnych zmian tej pozycji. Kluczem kolejki jest prawdopodobieństwo, że dany zestaw modyfikacji nukleotydów jest prawidłowy. Podczas pracy następuje wyjmowanie z kolejki wartości o maksymalnym kluczu. Jeśli dany zestaw modyfikacji nie prowadzi do zmodyfikowania regionu do postaci zbudowanej wyłącznie z k-merów zaufanych, wówczas następuje wybranie kolejnej pozycji o najmniejszym współczynniku jakości, wyznaczenie potencjalnych jej zmian oraz wstawienie ich do kolejki, z uwzględnieniem zestawu modyfikacji wyznaczonych wcześniej

Tabela 4.1: Zestawienie wybranych algorytmów korekcji odczytów NGS

Algorytm	Grupa algorytmów	Typy odczytów	Obligatoryjne parametry użytkownika	Główna struktura danych	Uwagi
Quake	Spektrum k-merów	Illumina	k	Wektor bitowy	
Musket	Spektrum k-merów	Illumina	k, $\widehat{k}_{\mathrm{N}}$	Filtr Blooma	
RACER	Spektrum k-merów	(brak danych)[a]	$\widehat{\ell}_{\mathcal{G}}$	Tablica mieszająca	
PREMIER	Inne (ukryte modele Markowa)	Illumina	k, λ, γ; $d_{\max}$ lub β_{bias} i δ_{step}	(brak danych)	
BLESS	Spektrum k-merów	Illumina	k	Filtr Blooma	Dopuszczalne przetwarzanie przy pomocy wielu węzłów obliczeniowych
Fiona	Hybrydowe — drzewo sufiksów oraz dopasowywanie wielu sekwencji	Dowolne	$\widehat{\ell}_{\mathcal{G}}$	Uogólnione drzewo sufiksów	Korekcja błędów typu indel
Blue	Spektrum k-merów	Ion Torrent[b]	k, $\widehat{\ell}_{\mathcal{G}}$, η_{cut}	Tablica mieszająca	Korekcja błędów typu indel
Lighter	Spektrum k-merów	(brak danych)[c]	k, p_{α}, $\widehat{\ell}_{\mathcal{G}}$	Filtr Blooma	Algorytm probabilistyczny
BFC	Spektrum k-merów	Illumina	$\widehat{\ell}_{\mathcal{G}}$	Blokowy filtr Blooma	
Karect	Dopasowywanie wielu sekwencji	Większość NGS	Ploidia, typ odległości edycyjnej	(brak danych)	Możliwe wprowadzenie ograniczenia zapotrzebowania na RAM, korekcja błędów typu indel[d]
SAMDUDE	Inne	Illumina	—	Wektor wartości, macierz wartości	Algorytm sekwencyjny

[a] W pracy [107] nie określono typu odczytów, jednak w eksperymentach autorzy wykorzystali odczyty Illumina.

[b] W pracy [89] określono tylko, że algorytm jest dostosowany do korekcji odczytów IonTorrent, jednak w eksperymentach autorzy wykorzystali odczyty Illumina i Roche 454.

[c] Zarówno w pracy [192] jak i dokumentacji programu nie określono typu odczytów, jednak w eksperymentach autorzy wykorzystali odczyty Illumina.

[d] Zgodnie z wytycznymi autorów przy korekcji odczytów sekwenatorów Illumina algorytm należy uruchomić w trybie wyznaczania odległości edycyjnej Hamminga, w której algorytm nie koryguje błędów typu indel. Stąd korekcja błędów tego rodzaju funkcjonuje tylko dla odczytów innego typu.

pozycji. W przypadku niepowodzenia korekcji Quake może przeprowadzić skrócenie lub całkowite usunięcie odczytu.

Algorytm modeluje błąd symbolu $\mathfrak{r}[a]$ definiując pewną funkcję $f(\mathfrak{r}[a], c, \mathfrak{p}[a])$, $c \in \Sigma$, określającą prawdopodobieństwo, że dany symbol odczytu, po uwzględnieniu odpowiadającego mu współczynnika błędu, w rzeczywistości powinien posiadać określoną wartość c. Wartości funkcji f są wyznaczane we wstępnym etapie poprzez przeprowadzenie korekcji próbnej grupy odczytów. Funkcja f przy zastosowaniu twierdzenia Bayesa stanowi model określający prawdopodobieństwo poprawności danego zestawu modyfikacji regionu.

Musket

W algorytmie Musket [138] zastosowanie znalazły filtry Blooma [38]. Zadaniem filtru jest umożliwienie detekcji w trakcie tworzenia spektrum k-merów posiadających liczebności równe 1. Algorytm w pierwszym przebiegu wyodrębnia z odczytów k-mery; jeżeli informacja o danym k-merze była obecna w filtrze, wówczas k-mer jest wstawiany do tablicy mieszającej, w przeciwnym wypadku następuje jego wstawienie do filtru. Tym samym tablica mieszająca zawiera k-mery o liczebnościach większych od 1, co ogranicza jej rozmiar, oraz niewielką grupę pozostałych k-merów wstawionych na skutek nieprawidłowej odpowiedzi twierdzącej filtru Blooma. W drugim przebiegu następuje określenie liczebności k-merów obecnych w tablicy mieszającej; pozostałe k-mery o liczebnościach równych 1 są wówczas eliminowane.

Korekcja odczytu jest wykonywana w dwóch etapach. W pierwszym, zachowawczym etapie przyjmowane jest założenie, że jeżeli co najmniej jeden k-mer pokrywający dany symbol jest zaufany, wówczas symbol jest prawidłowy. W przypadku znalezienia symbolu niespełniającego tego warunku następuje jego modyfikacja na taką wartość, aby skrajne k-mery pokrywające go (tj. k-mery, których pierwsza lub ostatnia pozycja odpowiada indeksowi danego symbolu w odczycie) stał się zaufane.

Korekcja zachowawcza jednego odczytu może zostać przeprowadzona wielokrotnie, jednak przy braku jej powodzenia następuje wykonanie korekcji agresywnej. W tym celu dla odczytu wyznaczana jest macierz $\mathbf{M}_{|\Sigma|\times\ell}$. W odczycie wyszukiwane są k-mery niezaufane, które określają pozycję potencjalnie błędnego symbolu. Następnie dla każdego pokrywającego tę pozycję k-meru przeprowadzana jest próba zmiany każdego jego symbolu na wszystkie symbole $c \in \Sigma$. W momencie uzyskania zmodyfikowanego zaufanego k-meru inkrementowany jest element $\mathbf{M}_{c,a}$, gdzie a jest pozycją odpowiadającą modyfikacji w odczycie. Macierz stanowi podstawę do wprowadzenia modyfikacji odczytu — pozycje macierzy o największych wartościach wskazują docelowe symbole.

RACER

Algorytm RACER [107] jest parametryzowany przybliżonym rozmiarem genomu. Rozmiar ten jest wykorzystywany do określenia długości k-meru oraz progu obcięcia η_{cut}. Metoda doboru tych zmiennych została opracowana w sposób eksperymentalny. Algorytm wykonuje zliczanie k-merów zapisując je w tablicy mieszającej. Reprezentacja poszczególnych symboli nukleotydów odbywa się przy pomocy kodu dwubitowego — taka długość kodu pozwala na zapis do 4 symboli w jednym oktecie; symbole N są zamieniane na losowo wybrane symbole zbioru Σ. Typem danych przechowujących k-mery jest typ całkowitoliczbowy o zmiennych długości 64 bitów, stąd można wnioskować, że $k \leqslant 32$. Jednocześnie każdemu k-merowi przyporządkowanych jest 8 liczników odpowiadających liczebnościom k-merów obejmujących wszystkie możliwe 4 nukleotydy poprzedzające dany k-mer, oraz 4 następujące po nim.

Algorytm sprawdza, czy kolejne k-mery posiadają liczebności poniżej η_{cut}. W takim przypadku modyfikują ostatnie ich symbole w celu uzyskania k-meru znajdującego się powyżej progu. Jeśli istnieje tylko jedna taka możliwość, wówczas jest ona przyjmowana. W pracy opisującej algorytm nie określono więcej szczegółów dotyczących jego działania.

PREMIER

Działanie algorytmu PREMIER [210] jest oparte na założeniu, że praca sekwenatora może zostać opisana przy pomocy ukrytego modelu Markowa. Własność ta wynika z obserwacji, że wartości nukleotydów w genomie wykazują silną lokalną zależność od wartości innych nukleotydów.

W algorytmie tym zbiór stanów modelu jest tożsamy zbiorowi k-merów obecnych w odczytach wejściowych. Bieżącym stanem jest k-mer będący hipotetycznym prawidłowym podsłowem danego odczytu. Dla modelu zdefiniowano rozkład emisji przekształcający bieżący stan $i = k-1, k, \ldots, \ell-1$ do pary $(\mathfrak{r}[i-k+1, i], \mathfrak{q}[i-k+1, i])$, tj. pary sekwencji: k-meru odczytu oraz ciągu współczynników jakości odpowiadających temu k-merowi. W pracy nie została określona struktura danych wykorzystana do opisu stanu modelu.

Pierwszym etapem algorytmu jest określenie parametrów modelu przy pomocy zmodyfikowanego algorytmu Bauma-Welcha. Proces ten jest parametryzowany długością k-meru oraz parametrami λ i γ, stanowiącymi współczynniki funkcji kary, której zadaniem jest modyfikacja modelu w taki sposób, aby niskie współczynniki prawdopodobieństwa przejścia między stanami uległy dalszej redukcji.

Zasadniczy proces korekcji stanowi rozwiązanie problemu *określenia najbardziej prawdopodobnej sekwencji* (ang. *maximum likelihood sequence estimation*) i może zostać przeprowadzony na jeden z dwóch sposobów: zmodyfikowanym algorytmem Viterbiego albo algorytmem Fano. W obu przypadkach w oparciu o wartość emisji (bieżący k-mer odczytu) następuje określenie wartości stanu,

czyli sekwencji k-meru potencjalnie prawidłowego. Postawione jest przy tym założenie, że pierwszy k-mer odczytu zawsze jest prawidłowy. Zmodyfikowany algorytm Viterbiego jest parametryzowany maksymalną odległością Hamminga $d_{\max}$ między odczytem prawidłowym a błędnym (wprowadzenie takiego ograniczenia umożliwia redukcję złożoności obliczeniowej algorytmu). Z kolei algorytm Fano parametryzowany jest współczynnikiem β_{bias} funkcji oceny danej ścieżki korekcji — *miary Fano* (ang. *Fano metric*), a także wartością kroku δ_{step}, określającego szybkość relaksacji minimalnej dopuszczalnej miary Fano danego ciągu stanów, gdy w danym momencie nie ma możliwości znalezienia innej ścieżki o wyższej wartości miary Fano.

BLESS

Algorytm BLESS [97, 98] w wersji 1.02 wykorzystuje filtr Blooma jako strukturę danych reprezentującą spektrum k-merów. Implementacja algorytmu jest dostosowana zarówno do uruchomienia na komputerach zgodnych z modelem z pamięcią wspólną, jak również z modelem sieciowym, umożliwiając prowadzenie obliczeń przy użyciu wielu maszyn (*węzłów obliczeniowych*).

Pierwszym etapem pracy algorytmu jest wyznaczenie piątego percentyla współczynników jakości, co jest przeprowadzane na pewnej próbie odczytów przez węzeł 1. Następnie zmodyfikowana, dostosowana do uruchomienia na wielu węzłach wersja algorytmu KMC przeprowadza zliczanie k-merów. Każdy węzeł generuje $\delta/\rho_{\text{node}}$ plików tymczasowych KMC, gdzie δ jest liczbą tymczasowych plików, a ρ_{node} jest liczbą węzłów. W rezultacie każdy węzeł posiada informacje o określonej części k-merów, które następnie są przesyłane do węzła 1 odpowiedzialnego za wybór progu η_{cut}. Przyjęta wartość progu jest przesyłana do wszystkich węzłów, które w oparciu o nią uzyskują własną część spektrum. Spektrum jest kodowane w lokalnych dla węzłów filtrach Blooma. Następnie filtry są rozsyłane do wszystkich pozostałych węzłów, które wykonują sumę binarną filtrów odebranych i lokalnych. W rezultacie wszystkie węzły zawierają identyczne kopie reprezentacji spektrum.

Korekcja jest przeprowadzana przez równolegle pracujące węzły, z których każdy jest odpowiedzialny za przetworzenie $|\mathcal{R}|/\rho_{\text{node}}$ odczytów. Pierwszym etapem korekcji jest detekcja fragmentów odczytów zbudowanych z zaufanych k-merów. Obszary te są skracane, jeżeli niezaufane obszary między nimi posiadają długość mniejszą od k. Jest to efekt obserwacji, że dowolna pozycja a w odczycie, taka że $k-1 \leqslant a \leqslant \ell - k$ jest pokrywana przez k k-merów, a w rezultacie jeden błąd substytucji symbolu na takiej pozycji generuje k błędnych k-merów. Podstawą uznania symbolu w odczycie za błędny może być także wartość współczynnika jakości jego pozycji znajdująca się poniżej piątego percentyla.

Proces korekcji fragmentów zbudowanych z nieprawidłowych k-merów jest przeprowadzany metodą z powrotami. Algorytm modyfikuje pierwszy (w kolej-

ności od obszaru prawidłowego) symbol do momentu znalezienia prawidłowego pokrywającego go k-meru. Jeśli proces się powiedzie, następuje przejście do kolejnego symbolu, który może, choć nie musi zostać zmodyfikowany w celu zapewnienia poprawności kolejnego k-meru. Po uzyskaniu prawidłowej postaci fragmentu następuje powrót w drzewie wywołań i sprawdzenie kolejnych możliwości. W przypadku znalezienia więcej niż jednego zestawu modyfikacji symboli spełniających warunek uzyskania prawidłowych k-merów, następuje wybór tego spośród nich, dla którego suma współczynników jakości poddawanych modyfikacji symboli jest minimalna.

W przypadku gdy algorytm nie jest w stanie wyznaczyć w odczycie żadnego obszaru prawidłowych k-merów, przeprowadzana jest modyfikacja pierwszego k-meru odczytu; wprowadzane są zmiany jego symboli, począwszy od pozycji o najniższych współczynnikach. W momencie, gdy zostanie znaleziony prawidłowy k-mer, następuje modyfikacja kolejnych symboli odczytu w kierunku $3'$ zgodnie z powyższą metodą z powrotami.

Jeżeli wprowadzono zmiany w symbolu znajdującym się mniej niż k pozycji od końca odczytu, następuje dołączenie do tego końca kolejnych symboli tworzących dodatkowe k-mery. Zaufanie tych k-merów jest warunkiem uznania danej modyfikacji za prawidłową. Dodatkowo w przypadku gdy znalezienie prawidłowej postaci błędnego fragmentu wiąże się z wprowadzeniem zbyt wielu zmian, wówczas przeprowadzane jest obcinanie odczytu do momentu uzyskania poziomu dopuszczalnej liczby zmian.

Wczesne wersje algorytmu (np. wersja 0.12) nie wykorzystywały do zliczania k-merów narzędzia KMC, ale przeprowadzały ten proces w oparciu o własny kod, którego działanie na wysokim poziomie abstrakcji było zbliżone do KMC: podczas wstępnego skanowania odczytów następowało wyodrębnianie k-merów, ich dystrybucja do plików dyskowych w oparciu o sekwencję k-meru, a następnie zliczanie poprzez wstawianie k-merów do tablic mieszających. Tablice te były zachowywane w plikach i po zakończeniu korekcji były wykorzystywane przez mechanizm redukujący wpływ nieprawidłowych odpowiedzi twierdzących filtru Blooma. Proces ten obejmował weryfikację, czy k-mery, które na skutek korekcji znalazłyby się w odczytach wyjściowych, faktycznie były obecne w tablicy mieszającej. W przeciwnym przypadku dana modyfikacja była wycofywana.

Fiona

Główną strukturą danych algorytmu Fiona [187] jest uogólnione drzewo sufiksów zbudowane dla sekwencji odczytów oraz ich odwróconych dopełnień. Algorytm oparto na definicji korekcji błędów określonej podobnie jak w metodach opartych na dopasowywaniu wielu sekwencji. Tym samym algorytm ten zaliczany jest do grupy algorytmów hybrydowych.

Pokrycie genomu k-merami jest modelowane przy pomocy rozkładu Poissona.

Rozkład jest estymowany w oparciu o długość genomu, długość k-meru, długość odczytu i liczbę odczytów. W oparciu o powyższy model następuje wyznaczenie prawdopodobieństwa pokrycia danej pozycji odczytu określoną liczbą k-merów, w zależności od przyjętej liczby błędów obecnych w k-merze. Stosunek takiego prawdopodobieństwa do prawdopodobieństwa pokrycia przez k-mery niezawierające błędów stanowi podstawę do określenia prawidłowości k-meru.

Detekcja błędów odbywa się poprzez wyszukiwanie podsłowa bieżącego odczytu w drzewie. Znalezienie k-meru zaklasyfikowanego przez model jako błędny skutkuje poszukiwaniem w drzewie sekwencji odczytów, które mogą zostać dopasowane do odczytu bieżącego oraz wykorzystane do wyznaczenia modyfikacji jego błędnego fragmentu. Modyfikacja odczytu odbywa się w kierunku od końca $3'$ do $5'$. Jako typ błędu, który podlega bieżącej korekcji wybierany jest taki, dla którego zostanie uzyskany najwyższy rezultat w głosowaniu, w którym biorą udział inne, dopasowane odczyty. Jednocześnie wybierany jest taki wariant modyfikacji, aby minimalizował odległość edycyjną między oryginalną, a zmodyfikowaną wersją bieżącego odczytu.

Blue

Implementacja algorytmu Blue [89] została rozdzielona na trzy odrębne narzędzia realizujące kolejne kroki: (*i*) zliczanie k-merów, (*ii*) generacja *par* k'-merów, (*iii*) korekcja. Pierwszym etapem jest zliczanie k-merów. Danymi wejściowymi mogą być zarówno odczyty poddawane korekcji, jak również zestaw innych odczytów, umożliwiając wykonanie korekcji krzyżowej. Zliczanie k-merów jest uzupełnione generacją par k'-merów, $k' < k$, domyślnie $k' = 16$. Strukturą danych przechowującą k-mery jest tablica mieszająca.

Faza korekcji polega na niezależnej analizie odczytów. Dla każdego z wyodrębnionych z nich k-merów jest określana liczebność oraz wyliczana średnia harmoniczna tych liczebności. Wykorzystanie wartości średniej liczebności (zamiast przyjęcia globalnego dla zestawu odczytów progu liczebności) umożliwia korygowanie odczytów pochodzących z fragmentów genomów o różnym pokryciu odczytami oraz uniknięcie fałszywego uznania odczytów z rejonów o słabym pokryciu za zawierające błędy. Tym samym pozwala na rezygnację z założenia o równomierności pokrycia genomu odczytami [123].

Potencjalna obecność błędu jest wykrywana przez spełnienie przynajmniej jednej z trzech przesłanek:

- obecność k-meru o liczebności mniejszej niż $1/3$ wartości średniej harmonicznej lub mniejszej niż zadany przez użytkownika próg,
- obecność k-merów o wyraźnie mniejszych wartościach liczebności niż pozostałe k-mery w odczycie,

- wykrycie końca homopolimeru (tylko przy zadaniu odpowiedniego parametru algorytmu).

W przypadku znalezienia potencjalnego błędu następuje wygenerowanie drzewa reprezentującego potencjalne modyfikacje symbolu uznanego za błędny oraz jego symboli sąsiednich. Jest ono przeszukiwane wgłąb w celu znalezienia zbioru modyfikacji o jak najwyższej ocenie, zachowując zasadę, że liczba wprowadzonych zmian powinna być jak najmniejsza. Obecność w drzewie potencjalnych modyfikacji symboli na pozycjach innych niż symbol błędny wynika z konieczności wzięcia pod uwagę wpływu zmiany na jakość pozostałej części odczytu. Modyfikacja może obejmować korekcję substytucji, insercji albo delecji długości jednego symbolu. Liczba wprowadzanych modyfikacji w celu redukcji czasu pracy algorytmu jest ograniczona do 3. Jako dodatkowe kryterium weryfikacji modyfikacji jest wykorzystywana koncepcja *par* (ang. *pair*) k'-merów. Para stanowi kontekst modyfikacji w kierunku $5'$ odczytu, umożliwiający wybranie lepszej modyfikacji. Stanowi też dodatkowe ograniczenie eliminujące niektóre ścieżki drzewa. Dokładna reguła wykorzystania par nie została wyjaśniona. Algorytm Blue nie wykorzystuje współczynników jakości odczytów, choć zmienia je dla symboli poddanych modyfikacji.

Lighter

Algorytm Lighter [192], pomimo zaliczania się do grupy algorytmów opartych na spektrum k-merów, nie wykonuje pełnego wstępnego zliczania k-merów ani nie określa progu obcięcia. Detekcja błędnych k-merów jest przeprowadzana poprzez probabilistyczne modelowanie spektrum.

Główną strukturą danych algorytmu są dwa blokowe filtry Blooma ze wzorcami. Korekcja przeprowadzana jest w trzech przebiegach. W pierwszym z odczytów wyodrębniane są wszystkie k-mery. Z prawdopodobieństwem p_α informacja o istnieniu danego k-meru jest wstawiana do filtru A. Zakładając, że typowy błędny k-mer posiada liczebność wynoszącą 1 oraz $p_\alpha < 0,5$, większość spośród błędnych k-merów nie zostanie umieszczona w filtrze. Wartość p_α stanowi parametr wejściowy algorytmu. W drugim przebiegu następuje analiza wszystkich symboli w odczytach. Dla każdego symbolu następuje wyszukanie w filtrze A wszystkich pokrywających go k-merów. Jeżeli liczba znalezionych k-merów jest większa, niż wartość określonego, dobranego w oparciu o probabilistyczny model progu, wówczas przyjmuje się, że dany symbol jest prawidłowy. Następnie w odczycie wyszukiwane są wszystkie spójne ciągi prawidłowych symboli długości k. Pokrywające je k-mery są wstawiane do filtru B.

Trzeci przebieg wykorzystuje wiedzę umieszczoną w filtrze B. W odczytach znajdowane są spójne obszary zbudowane z k-merów obecnych w tym filtrze, a następnie pozostałe obszary są poddawane modyfikacji poprzez zamianę ich

symboli na inne z wykorzystaniem algorytmu zachłannego. Spośród potencjalnego zestawu zmian wybierany jest ten, dla którego największa liczba k-merów obejmujących zmodyfikowane pozycje jest obecna w filtrze B. Powodem modyfikacji danej pozycji może być także niski współczynnik jakości danej pozycji. Próg określenia współczynnika jakości jako „niski" jest przyjmowany jako piąty percentyl współczynników jakości obecnych w próbie pierwszego miliona odczytów. Jeżeli modyfikowana pozycja znajduje się blisko końca odczytu, wówczas algorytm dołącza do odpowiedniego końca odczytu kolejne symbole tworzące dodatkowe k-mery, których obecność w filtrze B także jest traktowana jako kryterium przyjęcia modyfikacji znajdującej się w pobliżu końca.

W pracy [192] podkreślono, że odpowiedni dobór parametrów algorytmu pozwala na zachowanie w przybliżeniu stałego zapotrzebowania na pamięć, stałej dokładności oraz stałego współczynnika p_{FP} filtrów Blooma bez względu na głębokość sekwencjonowania.

BFC

Algorytm BFC [128] został opracowany w dwóch implementacjach różniących się sposobem zliczania k-merów. Wariant BFC-bf wykonuje zliczanie narzędziem KMC oraz przyjmuje $\eta_{\mathrm{cut}} = 3$. Wariant BFC-ht zlicza k-mery będące podsłowami odczytów takimi, że każdy wchodzący w ich skład symbol posiada współczynnik jakości $\mathfrak{q}[a] \geqslant 20$. W obu przypadkach baza danych k-merów jest zapisywana w blokowym filtrze Blooma.

Założeniem algorytmu jest uniknięcie przeszukiwania zachłannego. Proces korekcji polega na znalezieniu w odczycie pozycji pierwszego symbolu, który znajduje się na ostatniej pozycji pewnego k-meru niezaufanego. Następnie algorytm przystępuje do modyfikacji tego symbolu sprawdzając różne wartości zbioru Σ. Każda modyfikacja, wraz ze zbiorem modyfikacji poprzednich, sąsiadujących pozycji, jest wstawiana do kontenera Q, będącego kolejką priorytetową typu *min*. Następnie algorytm przechodzi do kolejnej pozycji w odczycie, z Q pobierany jest pierwszy element, zawierane przez niego zmiany są tymczasowo wprowadzane do odczytu, a bieżąca pozycja podlega analogicznym zmianom. Każdy element Q posiada przyporządkowaną liczbę pozycji, po modyfikacji których nie uzyskano zaufanej postaci k-meru. Liczba ta stanowi klucz kolejki. W pracy [128] zwrócono uwagę, że koncepcyjnie zmiana algorytmu pozwalająca na korekcję błędów typu indel ogranicza się do umożliwienia wstawiana do Q elementów zawierających informację o zmianie polegającej na wstawieniu bądź usunięciu symbolu. Jednakże rozwiązanie to, ze względu na rzadkość występowania błędów typu indel w odczytach Illumina, nie zostało zaimplementowane.

Jeżeli odczyt nie zawiera żadnych k-merów zaufanych, wówczas algorytm sprawdza zaufanie różnych wariantów pierwszego k-meru, różniących się od oryginalnego jedną pozycją. W przypadku znalezienia wielu zaufanych postaci pierw-

szego k-meru lub nieznalezienia żadnej, odczyt jest uznawany za niekorygowalny.

Karect

Przykładem algorytmu wykorzystującego dopasowywanie wielu sekwencji jest Karect [25]. Zasada jego działania jest następująca. Dla każdego odczytu wykonywana jest kilkuetapowa korekcja. W pierwszej kolejności następuje wyznaczenie zbioru odczytów, które poprzez dopasowanie do odczytu bieżącego będą stanowiły podstawę do jego modyfikacji. W tym celu wyszukiwane są odczyty współdzielące z odczytem bieżącym co najmniej jeden k-mer. Następnie zbiór ten jest uzupełniany odczytami współdzielącymi z odczytem bieżącym jeden k-mer, ale różniącymi się w swoim sufiksie lub prefiksie długości $k/3$ pewną określoną liczbą symboli. W dalszej kolejności jako długość oligomeru jest przyjmowana wartość $k' = 2k/3$ i cykl powtarza się. Przejście do kolejnego etapu odbywa się wyłącznie, jeżeli liczba dobranych do dopasowywania odczytów nie przekroczyła określonej wartości. W pracy [25] nie opisano struktury danych przechowującej odczyty.

Spośród znalezionych odczytów dobierane są te, które posiadają z korygowanym odczytem wspólny fragment z niewielką liczbą różnic. Ogranicza to ryzyko wykorzystania do korekcji odczytów pochodzących z innych części genomu. Wyniki dopasowywania są zapisywane w postaci grafu, którego wierzchołki reprezentują potencjalne sposoby korekcji, a krawędzie są ważone liczbą odczytów, które stanowią uzasadnienie danej zmiany. Węzły stanowiące początki krawędzi o zbyt dużych wagach są usuwane, aby uniknąć uwzględnienia w korekcji odczytów pochodzących z innych, bardzo podobnych, a mocniej pokrytych odczytami fragmentów genomu. Następnie wagi krawędzi są poddawane normalizacji, tak aby suma wag krawędzi wychodzących z wierzchołka wynosiła 1. Do korekcji wybierana jest ścieżka w grafie o jak najwyższym iloczynie wag krawędzi. Uwzględniane są także wagi odczytów, zależne od współczynników jakości oraz stopnia podobieństwa danego odczytu do odczytu korygowanego. Algorytm przyjmuje fakultatywny parametr określający maksymalną wykorzystywaną ilość pamięci; parametr ten ma wpływ na czas pracy algorytmu.

SAMDUDE

Narzędzie SAMDUDE [79] jest adaptacją algorytmu DUDE [201], będącego uniwersalnym algorytmem usuwania szumów obecnych w ciągu symboli. DUDE został oparty na założeniu istnienia jednej prawidłowej sekwencji symboli przesyłanej przez znany *dyskretny kanał bez pamięci* (ang. *discrete memoryless channel — DMC*), którego wyjście stanowi druga, zaszumiona sekwencja. Zasadą działania algorytmu jest eliminacja szumu (błędów) z drugiej sekwencji w oparciu o znajomość estymacji charakterystyki kanału.

SAMDUDE stanowi modyfikację powyższej idei. Modyfikacja ta wynika z konieczności estymacji parametrów kanału w oparciu o wiele błędnych sekwencji

(odczytów), uznawanych za próbki jednej błędnej sekwencji. Wejście algorytmu stanowi plik SAM zawierający dopasowanie odczytów z sekwencjonowania do genomu, wobec tego uruchomienie algorytmu musi być poprzedzone wykonaniem mapowania. Postawione jest założenie, że większość spośród symboli dopasowanych do danej pozycji genomu posiada wartość równą tej pozycji; w przypadku gdy występuje mniejszościowa grupa dopasowanych pozycji o odmiennych symbolach, mogą one sygnalizować obecność błędu.

Sekwencja wejściowa jest definiowana jako jeden ciąg symboli nad alfabetem Σ, z kolei wyjściowa jest zbiorem ciągów wartości nad alfabetem $\Sigma \times Q_8$, czyli zbiorem par $(\mathfrak{r}_i, \mathfrak{q}_i)$ sekwencji symboli odczytów oraz współczynników jakości o zredukowanej liczbie poziomów kwantyzacji z 42 do 8. W pierwszej kolejności zadaniem algorytmu jest estymacja parametrów kanału oraz wektora liczebności $\mathbf{v}$. Estymator kanału jest macierzą $\mathbf{M}_{|\Sigma|\times|\Sigma\times Q_8|}$ zawierającą informację o liczbie wystąpień każdej możliwej pary symbolu i współczynnika jakości (c', q), $c' \in \Sigma$, $q \in Q_8$, gdy sekwencja prawidłowa na odpowiadającej tej parze pozycji posiada wartość $c \in \Sigma$. Z kolei wektor $\mathbf{v}$ jest uzupełniany liczbą wystąpień poszczególnych elementów zbioru $\Sigma \times Q_8$ w swoim kontekście długości k, tj. wektor ten zawiera liczebności trójek $(\kappa_\mathrm{L}, (c', q), \kappa_\mathrm{R})$, gdzie κ_L i κ_R (stanowiące kontekst) są odpowiednio k-merami znajdującymi się w odczycie bezpośrednio na lewo i na prawo od symbolu c' o współczynniku jakości q.

W oparciu o powyższy model dla każdego symbolu c obecnego w odczytach z sekwencjonowania określana jest estymacja rozkładu prawdopodobieństwa, że dla kontekstu tego symbolu, jego wartości oraz współczynnika jakości powinien przyjąć określoną, skorygowaną wartość. Jeżeli maksimum tego rozkładu p_max jest osiągane dla innego symbolu c' niż c, wówczas dany symbol jest zamieniany na c', a odpowiadający mu współczynnik błędu na p_max. Wartość p_max jest wykorzystywana do modyfikacji współczynnika błędu także w przypadku, gdy symbol nie ulega zmianie. Wówczas jako nowy współczynnik błędu przyjmowana jest wartość $(1 - \frac{p+p_\mathrm{max}}{2})$, gdzie p jest oryginalnym współczynnikiem błędu tego symbolu kodowanym przez jego współczynnik jakości q. W obu przypadkach nowy współczynnik błędu zostaje zakodowany do współczynnika jakości zgodnie ze wzorem (3.3).

Możliwość przetwarzania odczytów wstępnie dopasowanych stanowi istotne ograniczenie uniwersalności algorytmu, jednak możliwość konwersji wyjściowych plików SAM do formatu FASTQ jest zadaniem prostym do wykonania przy pomocy odpowiednich narzędzi [130].

4.1.4 Inne algorytmy korekcji odczytów Illumina

W algorytmie Trowel [134] korekcja przeprowadzana jest przy silnym oparciu na współczynnikach jakości odczytów. We wstępnym etapie następuje określenie wartości 92 percentyla współczynników jakości q_min. W bazie danych umieszcza-

ne są k-mery, które zostały wyodrębnione z fragmentów odczytów zbudowanych z symboli posiadających współczynniki jakości powyżej tej wartości. W zasadniczym procesie korekcji jako pozycje odczytu potencjalnie błędne wyznaczane są te, które posiadają współczynnik jakości niższe od $q_{\min}$.

Algorytm Pollux [143] jest przeznaczony do korekcji odczytów Illumina, Ion Torrent oraz Roche 454, umożliwiając także korekcję odczytów różnego typu (uzyskanych z jednej próbki) w jednym uruchomieniu. W algorytmie następuje zliczanie k-merów, jednak bez wyznaczania progu obcięcia. Błędne k-mery w danym odczycie określane są w oparciu o występujące spadki ich liczebności. Obecność wielu sąsiednich k-merów o niskich liczebnościach (w porównaniu z pozostałym fragmentem odczytu) sygnalizuje obecność błędnego obszaru, który podlega korekcji algorytmem zachłannym — kolejne jego symbole podlegają modyfikacji, a po znalezieniu wiarygodnej zmiany, następuje przejście do kolejnej błędnej pozycji i powtórzenie procesu korekcji. Praca algorytmu nie opiera się na wykorzystaniu współczynników jakości, jednak po wprowadzeniu zmian w odczytach następuje ich modyfikacja.

Algorytm QuorUM [142] oparty jest na założeniach, że liczebność k-merów nie może być podstawowym kryterium detekcji błędów oraz że obcinanie odczytów jest nieodłącznym elementem poprawy jakości wyników. W algorytmie wykonywane jest zliczanie k-merów przy pomocy narzędzia Jellyfish. Jako próg obcięcia przyjęto wartość $\eta_{\text{cut}} = 5$. Korekcja rozpoczyna się od znalezienia pierwszego k-meru o liczebności $\eta = 3$ (tj. poniżej progu), od którego w kierunku obu końców odczytu sprawdzane są kolejne k-mery. Przy detekcji i korekcji błędów pod uwagę brane są wartości współczynników jakości oraz obecność spadków liczebności k-meru względem k-merów sąsiednich. Jeżeli liczba możliwości korekcji jest zbyt duża, następuje skrócenie danego odczytu.

Poza wymienionymi wyżej w ostatnich latach opracowano algorytmy: Ace [189], H-RACER [86], Bloocoo [36], Bcool [135], FMOE [106] oraz MuffinKmeans [24]. Do starszych rozwiązań można zaliczyć algorytmy: HiTEC [108], DecGPU [137], Coral [179], SHREC [186], Hybrid-SHREC [177], Reptile [208], Hammer [146], HiTEC [108], ECHO [109].

Odrębną grupą algorytmów korekcji są narzędzia stanowiące część asemblerów. Należą do nich moduł korekcji algorytmu ALLPATHS-LG [85], korektor asemblera SGA [190] i SOAPec, będący częścią SOAPdenovo2 [139]. Ich opis został pominięty ze względu na wyspecjalizowanie tych narzędzi do uzyskania danych o własnościach i formacie dostosowanym do jednego, wąskiego zagadnienia, co czyni je mało uniwersalnymi.

4.1.5 Algorytmy korekcji odczytów innego typu

Zagadnienie korekcji odczytów z sekwencjonowania było rozważane także w kontekście odczytów uzyskanych z sekwenatorów innych typów, tj. odczytów FGS

(szczególnie metody Sangera), innych niż Illumina metod NGS oraz metod TGS. Liczba istniejących algorytmów o takim zastosowaniu jest jednak znacznie mniejsza, niż rozwiązań dostosowanych do odczytów Illumina.

Algorytmy korekcji odczytów metody Sangera

Potrzeba przeprowadzenia korekcji została zauważona już przez autorów wczesnych algorytmów przetwarzania sekwencji nukleotydowych. Asemblery odczytów FGS EULER [162] oraz ARACHNE [34] zostały wyposażone w zintegrowane narzędzia przeprowadzające korekcję odczytów. Niedługo później opracowano autonomiczne, specjalizowane algorytmy, których zadanie stanowiła wyłącznie poprawa jakości odczytów, bez względu na ich dalsze zastosowanie. Były one oparte głównie na metodzie dopasowywania wielu sekwencji [110] charakteryzując się pewnym podobieństwem działania do niektórych późniejszych algorytmów korekcji odczytów NGS. Jednakże odmienna charakterystyka odczytów oraz ilość danych uzyskiwanych przy zastosowaniu późniejszych technik powoduje, że algorytmy te nie znajdują zastosowania dla odczytów tych technik [110].

W pracy [196] przedstawiono algorytm korekcji odczytów metody Sangera wykorzystujący *metodę zdefiniowanych pozycji nukleotydów* (ang. *defined nucleotide positions method — DNP method*). W pierwszej kolejności działanie algorytmu obejmuje skrócenie odczytów poprzez eliminację grup symboli, których średni współczynnik jakości jest poniżej zadanego progu. Następnie przeprowadzany jest dobór odczytów do wzajemnego dopasowywania. Odbywa się on w oparciu o obecność w odczytach wspólnych k-merów. Określenie, że dwa odczyty powinny zostać wzajemnie dopasowane odbywa się, jeżeli współdzielą one ze sobą co najmniej dwa niezachodzące, tj. niewspółdzielące wspólnych pozycji odczytu k-mery; k-mery te mogą się różnić określoną liczbą symboli. Relacja między pozycjami odpowiadających sobie k-merów w różnych odczytach stanowi jednocześnie wstępne wzajemne dopasowanie tych odczytów.

Uzyskane dopasowania są następnie optymalizowane przy pomocy zewnętrznego narzędzia ReAligner [29]. Optymalizacja obejmuje także maskowanie fragmentów odczytów w przypadku wykrycia, że część odczytu pochodzi z powtarzających się fragmentów genomu. Ponadto następuje weryfikacja, czy liczba różnic między mapowanymi odczytami jest podobna do oczekiwanej, tj. wynikającej z wartości współczynników jakości. Po wykonaniu mapowania znajdowana jest postać konsensusowa symboli, która jest równa najczęściej występującemu symbolowi na odpowiednich pozycjach różnych zmapowanych odczytów.

Algorytm AutoEditor [80] wykorzystuje odmienną technikę. Wejście algorytmu, poza zbiorem odczytów, stanowi genom referencyjny oraz zestaw *chromatogramów*, tj. przebiegów zmiany natężenia światła o określonym kolorze, uzyskanych przez urządzenie sekwencjonujące. W oparciu o chromatogramy oprogramowanie urządzenia sekwencjonującego przeprowadza syntezę odczytów.

W pierwszej kolejności w algorytmie wykonywane jest dopasowywanie odczytów do genomu. Następnie, dla każdego ciągłego, dopasowanego do genomu fragmentu odczytu określana jest liczba symboli, które zostały dopasowane dokładnie, czyli są one równe odpowiadającym im pozycjom w genomie. Jeśli liczba takich symboli stanowi co najmniej połowę liczby symboli fragmentu, fragment ten jest wykorzystywany do korekcji. W dalszej kolejności dopasowane odczyty są wykorzystywane do modyfikacji sekwencji genomu w celu eliminacji różnic wynikających ze zmienności wewnątrzgatunkowej.

Proces korekcji obejmuje analizę chromatogramów w miejscach, dla których w oparciu o dopasowanie występuje podejrzenie błędu. Analiza ma na celu znalezienie charakterystycznych maksimów chromatogramu, nazywanych *dobrze określonymi szczytami* (ang. *well-defined peaks*). Stanowią one maksima przebiegów, które cechują się wartościami amplitudy oraz czasu trwania maksimum. Jeżeli liczba dokładnie dopasowanych symboli we fragmencie odczytu zgadza się z liczbą dokładnie określonych szczytów na chromatogramie, wówczas symbole fragmentu uznane za potencjalnie błędne są korygowane w oparciu o zmodyfikowaną sekwencję genomu.

Algorytmy korekcji odczytów innych technik NGS

Przedstawicielem algorytmów korekcji innych technik sekwencjonowania drugiej generacji jest HECTOR [203]. Algorytm ten jest przeznaczony do korekcji odczytów Roche 454 i jest określany przez autorów jako oparty na *spektrum homopolimerów* (ang. *homopolymer spectrum based*). W pierwszym etapie zadaniem algorytmu jest konwersja odczytu do postaci ciągu par (c, n), gdzie $c \in \Sigma$ jest symbolem, a n jest liczbą sąsiadujących wystąpień c. Następnie przeprowadzane jest zliczanie *k-homopolimerów* (ang. *k-hopos*), którego idea jest zbliżona do zliczania k-merów z tą różnicą, że jako pojedynczy symbol k-homopolimeru traktowana jest cała para (c, n). W dalszej kolejności następuje określenie progu obcięcia η_{cut} oraz eliminacja k-homopolimerów o liczebnościach mniejszych od progu. Główna struktura danych jest połączeniem filtru Blooma oraz tablicy mieszającej.

Etap właściwej korekcji został oparty na algorytmie Musket i także obejmuje fazy korekcji zachowawczej i agresywnej, zdefiniowane w podobny sposób jak w algorytmie Musket. Modelowanie danych wejściowych przy pomocy k-homopolimerów jest głównym czynnikiem ułatwiającym korekcję błędów typu indel, w szczególności błędów homopolimerów. Wynika to z faktu, że korekcja błędu homopolimeru może się odbyć poprzez modyfikację liczebności n danego odczytu, bez konieczności wstawiania dodatkowego symbolu do sekwencji reprezentującej odczyt. Przykładowo, modyfikacja sekwencji AAAA do postaci AAAAAA polega na zamianie pary $(\mathsf{A}, 4)$ na $(\mathsf{A}, 6)$.

Algorytmy korekcji odczytów TGS

Urządzenia sekwencjonowania TGS są obecnie dostępne komercyjnie [37], umożliwiając powszechne wykorzystanie zalet ich odczytów. Jednocześnie istotną barierą ich efektywnego zastosowania jest obecność dużej liczby błędów sekwencjonowania, nieakceptowanej przez narzędzia przeznaczone dla przetwarzania odczytów NGS. W rezultacie długie odczyty niejednokrotnie pełnią funkcję pomocniczą, uzupełniając informacje uzyskane z odczytów NGS.

Większość algorytmów korekcji odczytów NGS nie jest dostosowana do przetwarzania odczytów TGS, nie tylko ze względu na długość odczytów różniącą się o rzędy wielkości i dużo liczniejsze, posiadające inną charakterystykę, błędy. Znaczenie ma także niska przepustowość tych urządzeń, skutkującą niewielkim pokryciem sekwencjonowanego genomu odczytami. Jak dotąd opracowano kilka algorytmów dostosowanych do korekcji odczytów TGS. Wszystkie znane autorowi algorytmy opierają się na hybrydowym wykorzystaniu odczytów krótkich odczytów NGS do korekcji długich odczytów TGS. Algorytmy te stawiają założenie, że oba rodzaje odczytów zostały uzyskane z tej samej biblioteki sekwencji [31, 178].

W pracy [31] zaproponowano algorytm LSC przeprowadzający korekcję poprzez mapowanie odczytów NGS do odczytów TGS. W pierwszej kolejności w odczytach wejściowych obu typów następuje eliminacja (*kompresja* — ang. *compression*) homopolimerów poprzez skrócenie ich do sekwencji długości $\ell = 1$ (np. sekwencja ACCCG zostaje skompresowana do postaci ACG). Autorzy algorytmu zapewniają, że powyższa zmiana wywiera niewielki wpływ na liczbę prawidłowo zmapowanych odczytów. Następnie następuje wykonanie filtracji odczytów NGS w oparciu o kryterium długości odczytu i liczbę symboli N. Odczyty TGS są poddawane wzajemnej konkatenacji z wstawieniem pomiędzy nimi ciągów symboli N, tworząc w ten sposób sekwencje imitujące chromosomy referencyjne. Proces korekcji obejmuje mapowanie odczytów NGS do utworzonych sekwencji przy pomocy zewnętrznego narzędzia Novoalign [99]. W przypadku różnic między odczytami zmapowanymi a sekwencją referencyjną określa się postać konsensusową różnic, która stanowi korekcję odpowiedniej pozycji sekwencji. Ostatnim etapem jest przywrócenie nieskorygowanych, skompresowanych homopolimerów odczytów TGS do pierwotnej postaci oraz wyodrębnienie z sekwencji odcinków odpowiadających odczytom. Przedstawione wyniki dowodzą, że algorytm LSC poprawia jakość w przypadku zastosowania strategii sekwencjonowania RNA-Seq. Eksperymenty oparto na odczytach Illumina, które wykorzystano do korekcji odczytów PacBio.

Algorytm LoRDEC [178] wykorzystuje odczyty NGS do budowy grafu de Bruijna. Z odczytów tych następuje wyodrębnianie k-merów, filtracja k-merów w oparciu o kryterium zadanego przez użytkownika progu η_{cut} oraz budowa grafu de Bruijna reprezentowanego przy pomocy filtru Blooma. Proces korekcji obejmuje wyszukiwanie w grafie k-merów wyodrębnionych z odczytów TGS. Jeżeli część

spośród takich k-merów nie zostaje znaleziona w grafie, wówczas pomiędzy węzłami grafu reprezentującymi znalezione k-mery znajdowana jest ścieżka, a obecne na niej symbole pełnią rolę korygującej odczyt modyfikacji. Odczyty NGS, dla których nie znaleziono żadnych prawidłowych k-merów są eliminowane. Ponadto odczyty, na końcach których pozostały nieskorygowane fragmenty są obcinane, a odczyty, które posiadają nieskorygowane fragmenty poza końcami są rozdzielane.

Inne rozwiązanie oparte na grafie de Bruijna zostało wykorzystane w algorytmie Jabba [150]. Ta metoda również oparta jest na budowie grafu sekwencji odczytów NGS, jednakże dopasowywanie odczytów TGS do grafu nie jest przeprowadzane z wykorzystaniem k-merów. W zamian stosowana jest metoda *rozszerzania ziarna* (ang. *seed and extend*), nie ograniczająca długości dopasowanych fragmentów do k. Po zbudowaniu grafu następuje eliminacja jego błędnych fragmentów, wykonywana w oparciu o informację o pokryciu odczytami. Następnie przeprowadzane jest wyszukanie najdłuższych wspólnych sekwencji reprezentowanych przez graf oraz podsłów odczytów TGS. Różne fragmenty odczytu TGS mogą być dopasowane niezależnie. Po uzyskaniu zbioru dopasowań następuje ich filtracja, eliminująca m.in. zbyt krótkie dopasowania. Pozostałe dopasowania są rozszerzane poprzez przechodzenie przez graf. Różnice między odczytem a ścieżką w grafie będącą rozszerzonym fragmentem stanowią podstawę do wykonania korekcji.

4.2 Parametry algorytmów korekcji

Działanie przedstawionych powyżej algorytmów jest zależne od szeregu parametrów dostrajających ich przebieg. Wartości wielu spośród parametrów, jak rozmiar filtrów Blooma lub próg obcięcia, są zwykle wewnętrznie ustalone przez autorów bądź określane automatycznie w oparciu o inne parametry lub charakterystykę danych wejściowych. Jednakże w większości przypadków wykorzystanie algorytmu wymaga zadania parametrów przez użytkownika.

Wśród tych ostatnich występuje grupa parametrów, których wartość można określić dosyć precyzyjnie analizując własności danych wejściowych (długość genomu, głębokość sekwencjonowania, liczbę k-merów określonego typu obecnych w odczytach). Z kolei inne parametry wywierają wpływ na pracę algorytmu w sposób trudny do precyzyjnego określenia, a w rezultacie ich wyznaczenie stanowi wyzwanie (długość k-meru, wartości określające prawdopodobieństwa doboru k-merów). Ich optymalny dobór wymagałby posiadania dokładnej wiedzy na temat działania algorytmu lub sprawdzenia rezultatów kolejnych etapów przetwarzania skorygowanych odczytów [187]. W takich przypadkach autorzy algorytmów proponują rozmaite strategie określania wartości parametrów, zazwyczaj opierając się na nieformalnych, uzasadnionych empirycznie zasadach. Poniżej przedsta-

wiono reguły doboru najważniejszych parametrów, które zostały przedstawione w pracach wprowadzających opisane wyżej algorytmy korekcji.

4.2.1 Dobór długości oligomeru

Długość oligomeru k to najczęściej spotykany, szczególnie wśród algorytmów opartych na spektrum k-merów, parametr wywierający wpływ na jakość wyników algorytmu. Przy jego doborze należy zwrócić uwagę, że małe wartości k pozwalają na dokładniejszą lokalizację błędu w odczycie [110], jednakże wartości zbyt małe powodują wzrost liczby przypadków takich, że różne k-mery o sekwencjach o niewielkich odległościach edycyjnych pochodzą z różnych, podobnych części genomu. Takie przypadki są trudne do odróżnienia od błędów [110]. Ponadto dla algorytmu BLESS zasygnalizowano, że zbyt małe wartości k mogą powodować wzrost liczby potencjalnych korekcji danego błędu i zwiększają prawdopodobieństwo wyboru nieprawidłowej korekcji [97]. Zwiększanie wartości k zmniejsza wpływ powtórzeń w genomie, ale jednocześnie zmniejsza przeciętną liczebność k-merów [97] powodując „wypłaszczenie" histogramu ich liczebności.

Dla algorytmu Quake zaproponowano dobór k zgodnie z równaniem $k \approx \log_4 200\widehat{\ell}_{\mathcal{G}}$, które w praktyce powoduje uzyskanie wartości z zakresu 15–19. Nowsze algorytmy powinny przyjmować wyższe wartości k, co jest obecnie wykonalne dzięki możliwości tworzenia baz danych k-merów także dla większych wartości k, a pozwala na ograniczenie wpływu powtórzeń w genomach [123].

W pracy wprowadzającej algorytm PREMIER wpływ wartości k na korekcję został omówiony podobnie, jak przedstawiono powyżej, nie podano jednak dokładnego wzorca doboru tej wartości. Do przeprowadzenia eksperymentów wybrano różne wartości k ze zbioru $\{13, 14, 15\}$.

Algorytm BLESS powinien być parametryzowany taką wartością k, aby spełnione były dwa warunki: $|\mathcal{K}_{\text{un}}|/4^k \leqslant 0{,}0001$ oraz liczba skorygowanych przez algorytm symboli była maksymalna. Należy zauważyć, że szczególnie drugi warunek nie jest możliwy do sprawdzenia *a priori* i jego określenie wymaga kilkukrotnego uruchomienia algorytmu.

Dla algorytmu BFC domyślnie przyjęto $k = 33$ nie podając uzasadnienia oraz reguły, kiedy wartość tę należy zmienić. Z kolei w algorytmie SAMDUDE przyjęto $k = 7$, stanowiące połowę długości kontekstu symbolu.

W algorytmie RACER wartość k jest określana automatycznie. Reguła jej doboru została opracowana *eksperymentalnie* i jest przeprowadzana w oparciu o długość genomu $\widehat{\ell}_{\mathcal{G}}$, jednak nie została dokładnie opisana.

Algorytm Fiona, jako korzystający ze struktur sufiksowych, nie posiada ustalonej jednej długości oligomeru, jednak w celu budowy struktury danych wymagane jest określenie zakresu długości k-merów zapisywanych w strukturze danych ograniczonym wartościami $k_{\min}$ oraz $k_{\max}$. Parametr $k_{\min}$ jest ustalany automatycznie poprzez minimalizację sumy dwóch wartości: (i) oczekiwanej liczby po-

zycji w odczytach, gdzie błędy są rozmieszczone w sposób, który uniemożliwia znalezienie k-meru pozbawionego błędu, oraz (*ii*) oczekiwanej liczby pozycji w odczytach, gdzie błąd w k-merze może spowodować odniesienie go do innej pozycji genomu. Obie te wartości są określane probabilistycznie w oparciu o oczekiwaną liczbę błędów, długość odczytów oraz długość genomu. Jednocześnie przyjmowane jest $k_{\max} = k_{\min} + 10$.

W algorytmie Karect parametr k także jest określany automatycznie i może przyjąć wartości z zakresu 27–42, w zależności od liczby odczytów, jednak nie wyjaśniono dokładnie metody doboru. Dobór wartości k dla algorytmu Blue, Lighter oraz Musket nie został wyjaśniony przez ich autorów.

4.2.2 Dobór progu obcięcia

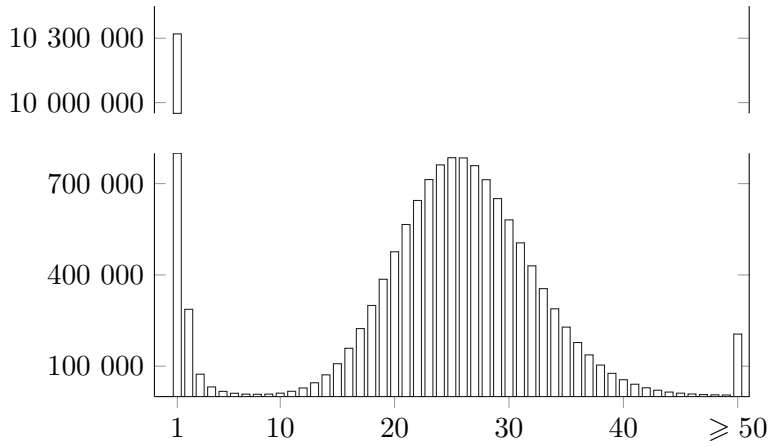
Dobór progu obcięcia η_{cut} jest zwykle przeprowadzany przy postawieniu założenia, że występowanie błędu jest rzadkim zjawiskiem, różne rodzaje błędów substytucji są równie prawdopodobne oraz pokrycie genomu odczytami jest równomierne [123]. Spośród przeanalizowanych algorytmów tylko Blue wymaga określenia tego parametru przez użytkownika.

W algorytmach BLESS oraz Musket próg obcięcia jest dobierany automatycznie w drodze analizy histogramu liczebności k-merów. Analiza polega na znalezieniu pierwszego minimum histogramu, co wynika z obserwacji, że k-mery błędne zazwyczaj posiadają niewielką liczebność. Jednocześnie, zakładając odpowiednio wysokie pokrycie sekwencjonowania, liczebność prawidłowych k-merów powinna być znacznie większa od 1. W rezultacie rejon liczebności k-merów o wartościach pośrednich powinien być reprezentowany przez niewielką liczbę k-merów i stanowić obszar poszukiwania progu obcięcia. Przykładowo, na rys. 4.1 przedstawiono histogram liczby wystąpień 25-merów wraz z rozkładami prawdopodobieństw liczby 25-merów błędnych i prawidłowych zestawu danych o głębokości sekwencjonowania dep ≈ 40. Jego pierwsze minimum określa próg obcięcia równy $\eta_{\text{cut}} = 8$.

W algorytmie Quake także przeprowadzana jest generacja histogramu liczebności, uwzględniająca dodatkowo wagi k-merów. Algorytm doboru przyjmuje parametr Ξ_{p} (domyślnie $\Xi_{\text{p}} = 200$) określający stosunek prawdopodobieństwa, że dany k-mer jest prawidłowy do prawdopodobieństwa, że jest błędny. W oparciu o to, korzystając z modelującego rozkładu prawdopodobieństwa, wyliczany jest próg obcięcia.

W algorytmie RACER wartość η_{cut} jest określana automatycznie jako funkcja rozmiaru genomu $\widehat{\ell}_{\mathcal{G}}$, jednak nie sprecyzowano dokładnie postaci tej funkcji zaznaczając tylko, że sposób doboru tej wartości został opracowany *eksperymentalnie*.

W algorytmie BFC-bf domyślnie przyjmuje się $\eta_{\text{cut}} = 3$. Dla algorytmu Blue nie zaproponowano zasady doboru progu obcięcia η_{cut}. Jednocześnie zwrócono



Rysunek 4.1: Histogram liczebności 25-merów odczytów z zestawu ERR422544

uwagę, że przyjęcie dokładnego progu dla genomu może nie być użyteczne, ze względu na fragmenty genomu o dużej liczbie powtórzeń [89].

4.2.3 Inne parametry

Pozostałe parametry zwykle stanowią dane pozwalające na wyliczenie przedstawionych wyżej parametrów lub umożliwiające określenie rozmiarów struktur danych. Algorytm Musket przyjmuje przybliżoną liczbę k-merów $\widehat{\widehat{k}}_{\mathrm{N}}$, która wg informacji udostępnionych na stronie algorytmu służy do określenia rozmiarów struktur danych.

W algorytmie Lighter parametr wejściowy stanowi wartość prawdopodobieństwa wyboru k-meru p_α. W opisującej go pracy przyjęto, że powinna być ona odwrotnie proporcjonalna do głębokości sekwencjonowania i równa 0,1 dla dep $= 70$. Ponadto algorytm jest parametryzowany przybliżonym rozmiarem genomu $\widehat{\ell}_{\mathcal{G}}$ wykorzystywanym do określenia rozmiaru filtru Blooma.

W algorytmie PREMIER maksymalna odległość Hamminga między k-merem prawidłowym i błędnym została ustalona jako $d_{\max} = 4$. Dobór pozostałych parametrów, tj. λ, γ, β_{bias} oraz δ_{step} wg autorów jest trudny i w pracy wybrano różne ich wartości bez podawania dokładnych reguł doboru, określając tylko ich przybliżony rząd wielkości.

Algorytmy BFC, RACER, Fiona oraz Blue są parametryzowane przybliżonym rozmiarem genomu $\widehat{\ell}_{\mathcal{G}}$. W przypadku algorytmu RACER jest on wykorzystywany do wewnętrznego wyliczenia innych parametrów algorytmu, a w algorytmie Fiona do określenia zakresu długości k-merów zapisywanych w strukturze danych. W przypadku BFC i Blue nie określono dokładnego zastosowania tego parametru, choć w tym ostatnim przypadku biorąc pod uwagę, iż parametr stanowi wejście etapu zliczania k-merów, można przypuszczać, że służy do określenia rozmiaru tablicy mieszającej.

Algorytm Karect musi być parametryzowany przez użytkownika tylko informacją o ploidii organizmu (haploidalny lub diploidalny) oraz typem odległości edycyjnej wykorzystywanej w dopasowywaniu sekwencji (tylko substytucje, substytucje oraz błędy typu indel z równym kosztem, substytucje oraz błędy typu indel z różnym kosztem), które powinny być dopasowane do charakterystyki odczytów. Wewnętrznie w algorytmie Karect jest wykorzystywanych kilka innych, wyliczanych automatycznie parametrów, do których należą (*i*) minimalna długość wspólnego podsłowa dwóch odczytów poddawanych dopasowywaniu, wyliczana w oparciu o długości odczytów, (*ii*) maksymalna liczba różnic między odczytami poddawanymi dopasowaniu, która jest równa $1/4$ długości ich wspólnego podsłowa oraz (*iii*) maksymalna waga krawędzi niepodlegającej usunięciu, zależna od głębokości sekwencjonowania.

W niektórych przypadkach wyjaśniono krótko sposób, w jaki określane są parametry filtru Blooma. W algorytmie BLESS rozmiar filtru jest ustalany w oparciu o wartość η_{cut} oraz docelową wartość współczynnika p_{FP}. W algorytmie BFC domyślny rozmiar filtru wynosi $m = 2^{33}$ elementów i zdefiniowano dla niego $d = 4$ funkcje mieszające.

4.3 Komputerowa symulacja procesu sekwencjonowania

Testowanie oraz ocena jakości algorytmów przetwarzających odczyty z sekwencjonowania wymaga wykorzystania odpowiednich zestawów danych. Poza odczytami uzyskanymi w wyniku eksperymentów sekwencjonowania DNA niezbędne może być oparcie się na odczytach uzyskanymi w wyniku symulacji *in silico* [105]. W literaturze można spotkać przykładowe argumenty za zastosowaniem odczytów symulowanych:

- pozycja w genomie, z której uzyskano odczyt rzeczywisty nie jest znana [103],
- może wystąpić znaczące opóźnienie między uzyskaniem danych rzeczywistych a publicznym udostępnieniem ich w bazie odczytów [103],
- symulacja umożliwia uzyskanie odczytów o określonej charakterystyce [103],
- w odczytach rzeczywistych można spotkać „zanieczyszczające" sekwencje innych organizmów [188],
- uzyskanie odczytów symulowanych jest mniej kosztowne [188].

W niektórych przypadkach autorzy prac opisujących algorytmy korekcji wykorzystują odczyty wygenerowane prostymi metodami przygotowanymi *ad hoc* do przeprowadzenia konkretnego eksperymentu [103]. Jedną z popularniejszych

metod jest zaproponowana w pracy opisującej algorytm Quake [110]. Jej zasada działania opiera się na losowym wyborze z genomu podsłów długości symulowanego odczytu oraz dopasowaniu do nich sekwencji współczynników jakości uzyskanych z rzeczywistych zestawów odczytów. Współczynniki jakości są przekształcane do współczynników błędów, w oparciu o które losowo dobierana jest nowa wartość niektórych pozycji; podczas doboru nowych symboli bierze się pod uwagę profile błędów określające prawdopodobieństwo zamiany określonego symbolu na inny (np. w technologii Illumina częstsze są błędy zamiany nukleotydu A na nukleotyd C, niż na G lub T). W niniejszej pracy metoda ta będzie nazywana *metodą Quake*, a metody nieuwzględniające odmiennych prawdopodobieństw zamiany między różnymi symbolami będą nazywane *uproszczonymi metodami Quake*.

Rozwiązania tego rodzaju, choć wygodne, nie są jednak satysfakcjonujące i mogą powodować uzyskanie niemiarodajnych wyników [89, 209]. Ponadto w wybranych przypadkach uzyskanie odczytów wyłącznie imitujących proces sekwencjonowania może być niewystarczające i symulacja musi obejmować także inne zadania, jak generacja wariantów [161] lub generacja sztucznych genomów [157]. W związku z tym opracowano szereg narzędzi umożliwiających symulację odczytów z zachowaniem określonych wymagań. Niektóre spośród nich, umożliwiające symulację procesu sekwencjonowania genomów WGS zostały poniżej pokrótce omówione.

ART Narzędzie ART [105] pierwotnie zostało przygotowane na potrzeby Projektu 1000 Genomów i jest dostosowane do generacji odczytów technologii Roche 454, Illumina i SOLiD. Umożliwia symulację błędów sekwencjonowania przeprowadzaną w oparciu o model błędów opracowany poprzez analizę dużych zbiorów odczytów.

Dla technologii Illumina dostarczony jest zestaw modeli błędów substytucji dla odczytów długości nie większej niż 75 pz. Model błędów indel został zbudowany w oparciu o odczyty długości 36 pz, a zatem w obu przypadkach dla praktycznie niespotykanych obecnie sekwenatorów, jednak narzędzie umożliwia utworzenie nowych modeli poprzez przeprowadzenie analizy rzeczywistych odczytów. Model substytucji jest oparty na wartościach współczynników jakości rzeczywistych odczytów.

Według przeglądowej pracy [73] jest to najlepsze narzędzie symulacji odczytów. W pracy tej zasygnalizowano, że ART umożliwia symulację wariantów, jednak informacja taka nie znajduje się w dokumentacji tego narzędzia.

SInC W narzędziu SInC [161] położono nacisk przede wszystkim na symulację wariantów genomów, które mogą obejmować polimorfizmy SNP, warianty liczby kopii sekwencji oraz warianty typu krótki indel. Generacja odczytów odbywa się

w oparciu o model błędów opracowany na podstawie analizy odczytów Illumina długości 100 pz i uwzględnia tylko generację błędów typu substytucja. Narzędzie pozwala na utworzenie nowych modeli błędów poprzez analizę rzeczywistych odczytów.

CuReSim Narzędzie CuReSim [42] zostało przygotowane na potrzeby porównania algorytmów mapowania. Pozwala na generowanie odczytów dowolnych typów i posiada kilka zdefiniowanych rozkładów prawdopodobieństwa, a także może być parametryzowane współczynnikami błędów oraz dowolną długością odczytu. Narzędzie dostarcza możliwość symulacji tylko błędów substytucji oraz indeli.

FASTQsim Dla narzędzia FASTQsim [188] nie określono dokładnie, dla jakiego typu odczytów narzędzie to zostało zaprojektowane. Zostało jednak wyposażone w narzędzie analizy rzeczywistych odczytów, pozwalające na analizę większej, niż w pozostałych narzędziach liczby parametrów odczytów: ich długości, rozkładu prawdopodobieństwa substytucji w zależności od pozycji oraz wartości prawidłowego symbolu, rozkładu długości indeli i innych. Autorzy zaznaczają, że taki zestaw parametrów powinien być wystarczający dla wszystkich technik sekwencjonowania. Narzędzie analizy przyjmuje, poza genomem, zestaw zmapowanych odczytów, w oparciu o które generowane są powyższe statystyki.

4.4 Kryteria oceny algorytmów korekcji

Publikacje obejmujące swoją tematyką problem korekcji odczytów zawierają także opisy metody oceny jakości procesu korekcji. Do publikacji tych należą głównie prace opisujące nowe algorytmy, jednak można wskazać również kilka prac przeglądowych [100, 154, 209] oraz nierecenzowaną [153]. Pomimo istnienia tych ostatnich trudno jest wskazać powszechnie uznany „złoty standard" kryteriów pozwalających na ewaluację danego algorytmu. W związku z tym można spotkać się z dużą różnorodnością podejścia do tego zagadnienia.

W niniejszej pracy kryteria oceny zostaną podzielone na następujące grupy:

- kryteria *jakościowe*, obejmujące wpływ procesu korekcji na jakość wyników lub pracę następnych etapów przetwarzania odczytów,
- kryteria *wydajnościowe*, obejmujące zapotrzebowanie algorytmu korekcji na czas i pamięć operacyjną,
- kryteria *techniczne*, obejmujące łatwość przygotowania danych, wykonania korekcji oraz wykorzystania wyników.

4.4.1 Kryteria jakościowe

Głównym celem korekcji odczytów jest uzyskanie odczytów, będących według różnych kryteriów lepszej jakości. Wobec tego kluczową kwestią jest ocena, w jakim stopniu dany algorytm poprawia jakość odczytów oraz czy, ze względu na niepewność jakości algorytmów heurystycznych, nie pogarsza tej jakości. W literaturze można spotkać dwie odrębne grupy testów jakościowych: opartych na odczytach symulowanych oraz rzeczywistych.

Odczyty symulowane

We wszystkich omawianych pozycjach literaturowych odczyty symulowane zostały wykorzystane do bezpośredniej oceny wpływu korekcji. Uzyskując odczyt symulowany zawierający wygenerowane błędy oraz jego pierwotną, bezbłędną postać można dokonać porównania ich z sekwencją odczytów będącą rezultatem korekcji. Odczyty są uzyskiwane zwykle w oparciu o uproszczoną metodę Quake lub jej odmiany, jak w [138, 210], lub bezpośrednio metodą Quake [110]. Z kolei w pracy [192] odczyty uzyskano przy pomocy narzędzia Mason [103], natomiast w [100] przy pomocy narzędzia ART.

Jako granulację oceny przyjmowane są pojedyncze symbole (tzn. korzystna jest prawidłowa korekcja jak największej liczby symboli odczytów), np. [138, 192, 210] lub całe odczyty (tzn. korzystna jest prawidłowa korekcjach jak największej liczby całych odczytów), np. [110]. Zalety wykorzystania danego podejścia zależą od dalszego zastosowania odczytów: w przypadku przeprowadzenia detekcji polimorfizmów SNP istotny jest wzrost liczby skorygowanych symboli, natomiast przy asemblacji wzrost liczby skorygowanych odczytów [25]. Z drugiej strony można przyjąć, że uwzględnienie granulacji symbolu przynosi mniej użyteczne wyniki, ponieważ nie kładzie nacisku na uzyskanie odczytów całkowicie prawidłowych, charakteryzujących się największą użytecznością [154].

Mierzalność oceny jest uzyskiwana przez określenie przedstawionych niżej wartości. Większość z nich w analogiczny sposób będzie stosowana również w celu dokonywania oceny jakości detekcji wariantów:

- TP — liczba odczytów zawierających błędy (lub symboli błędnych), które zostały prawidłowo skorygowane,
- FP — liczba odczytów niezawierających błędu (symboli prawidłowych), które zostały niesłusznie zmodyfikowane przez algorytm korekcji,
- FN — liczba odczytów zawierających błędy (symboli błędnych), które nie zostały skorygowane lub zostały skorygowane błędnie.

Powyższe wartości stanowią podstawę wyliczenia następujących miar:

- czułość: $\xi_{\text{sens}} = \frac{\text{TP}}{\text{TP}+\text{FN}}$,

- precyzja: $\xi_{\text{prec}} = \frac{\text{TP}}{\text{TP}+\text{FP}}$,
- zysk (ang. *gain*): $\xi_{\text{gain}} = \frac{\text{TP}-\text{FP}}{\text{TP}+\text{FN}}$,
- miara F1 (ang. *F1-score*), tj. wartość średniej harmonicznej czułości i precyzji: $\xi_{\text{F1}} = \frac{2\xi_{\text{sens}}\xi_{\text{prec}}}{\xi_{\text{sens}}+\xi_{\text{prec}}}$.

Miary te w bezpośredni sposób znajdują zastosowanie do oceny korekcji błędów substytucji. Zwykle nie jest sprecyzowana strategia ich wyliczania w przypadku usunięcia odczytu bądź jego skrócenia. W niektórych pracach jako wynik eksperymentu podawane są wartości TP oraz FP, np. w [110].

Odczyty rzeczywiste

Wykorzystanie odczytów rzeczywistych pozwala na uzyskanie lepszej wiarygodności eksperymentów, szczególnie gdy zastosowane odczyty pochodzą z organizmu modelowego ze znanym genomem dobrej jakości, ponieważ odczyty symulowane często nie odzwierciedlają rzeczywistych profili błędów [142]. Jednocześnie jednak ocena jakości ich korekcji jest trudna [89]. Ocena ta zwykle jest dokonywana poprzez wyliczenie określonych statystyk rezultatów algorytmów mapowania lub asemblacji, przetwarzających zestawy odczytów poddanych i niepoddanych korekcji. Statystyki te często są stosowane także do oceny samego procesu mapowania lub asemblacji, stąd kryterium oceny obejmuje wpływ korekcji na działanie takich algorytmów.

Asemblacja Liczba miar ewaluacji i charakterystyki procesu asemblacji jest duża, jednak istnieje grupa narzędzi wyliczających je w sposób automatyczny, do których należy popularny w ocenie korekcji Quast [92]. Narzędzie to pozwala na uzyskanie m.in. liczby *błędnych asemblacji* (ang. *misassemblies*) ξ_{misasm}, wartości *pokrycia kontigami* (ang. *genome coverage* lub *genome fraction*) ξ_{cov}, statystyk N$\mathcal{X}$, NG$\mathcal{X}$, NA$\mathcal{X}$, NGA$\mathcal{X}$, L$\mathcal{X}$, LG$\mathcal{X}$, LA$\mathcal{X}$, LGA$\mathcal{X}$, oraz liczby kontigów $\mathfrak{n}_{\text{kontig}}$.

W narzędziu Quast wykorzystano pojęcie błędnej asemblacji, zdefiniowane w narzędziu Plantagora [33] jako pozycja w kontigu taka, że:

- sekwencja kontigu znajdująca się na lewo (jego prefiks) od tej pozycji została dopasowana ponad 1 kpz od sekwencji znajdującej się na prawo (jego sufiksu), lub
- sekwencje te nachodzą na siebie na odcinku co najmniej 1 kpz, lub
- są dopasowane do osobnych nici lub innych chromosomów.

Z kolei pokrycie kontigami jest rozumiane jako stosunek liczby symboli dopasowanych do genomu do długości genomu.

Jako Nχ rozumiana jest taka długość kontigu, że suma długości kontigów posiadających długość Nχ lub większą stanowi χ% sumy długości wszystkich kontigów. Z kolei jako NGχ rozumiana jest taka długość kontigu, że suma długości kontigów posiadających długość NGχ lub większą stanowi χ% długości genomu. Wartości NAχ i NGAχ są wyliczane w taki sam sposób jak Nχ i NGχ, jednak przed ich określeniem kontigi zawierające błędy asemblacji są dzielone w miejscach wystąpienia błędów. W dalszej części pracy statystyki te będą nazywane *miarami z grupy N*.

W podobny sposób są zdefiniowane statystyki Lχ, LGχ, LAχ, LGAχ, przy czym jako ich wartości są przyjmowane nie długości kontigów, ale ich liczby (np. Lχ — taka liczba kontigów, że suma długości pewnego zbioru kontigów mocy Lχ stanowi χ% sumy długości wszystkich kontigów). W dalszej części pracy statystyki te będą nazywane *miarami z grupy L*.

W każdym przypadku $\chi \in [0, 100]$. Uzyskanie miar opartych na dzieleniu kontigów w miejscach błędów wymaga zmapowania kontigów do genomu (mapowanie jest wykonywane przez narzędzie Quast przy pomocy algorytmu MUMmer [121]), zatem wartości mogą zależeć od różnic między genomem uzyskiwanym w wyniku asemblacji a genomem referencyjnym.

Ocena jakości korekcji przez ewaluację asemblacji jest metodą pośrednią, która ze względu na obecność wewnętrznych mechanizmów korekcji obecnych w asemblerach może powodować uzyskanie niemiarodajnych wyników [154]. W literaturze do oceny jakości korekcji można spotkać wykorzystanie miar N50 [89, 110, 128, 138, 142, 143, 192], NG50 [97, 98, 110, 128, 143, 192], NGA50 [25, 100], N90, NG90 [110], a także liczby kontigów $\mathfrak{n}_{\text{kontig}}$ [110], pokrycia genomu kontigami ξ_{cov} [25, 97, 134, 138, 192], liczby błędnych asemblacji [97, 128, 134, 138, 192] i innych [25, 89, 100, 134, 192]. W niektórych przypadkach oceniono jakość nie tylko w granulacji kontigów, ale także skafoldów [25, 100, 110, 134].

Mapowanie Analizując opisy eksperymentów z wykorzystaniem mapowania można zauważyć częste przyjmowanie milczącego założenia, że genom referencyjny posiada sekwencję identyczną z genomem, który został poddany sekwencjonowaniu. Ze względu na zmienność wewnątrzgatunkową założenie to zwykle nie jest spełnione, gdyż między tymi genomami występuje pewna liczba różnic. Mimo to nie powinno prowadzić ono do faworyzowania żadnych algorytmów, pod warunkiem że nie wykonują korekcji opierając się na dopasowaniu odczytu do genomu [187]. Z drugiej strony występuje trudność z porównaniem wyników różnych eksperymentów, ponieważ różni autorzy przyjmują różne założenia np. dotyczące traktowania odczytów niedopasowanych i dopasowanych wielokrotnie [89].

W celu oceny algorytmów korekcji często wykonuje się mapowanie skorygowanych i nieskorygowanych odczytów, a liczbę różnic między genomem a dopasowanymi uznaje się za oszacowanie liczby błędów. W oparciu o te różnice następuje

wyliczenie miar podobnych do zdefiniowanych w podrozdziale 4.4.1, jak w pracach [25, 97, 100, 134, 209, 210]. Innym rozwiązaniem jest przyjęcie jako wskaźnika jakości korekcji łącznej liczby różnic między odczytami a genomem, bez dokładnego określania powyższych miar, ale z przedstawieniem w różnej formie zmiany liczby tych różnic, np. [107, 187, 192], czasem z wyróżnieniem różnych typów różnic (jak substytucja lub indel) [143]. W niektórych przypadkach przyjęto jako ocenę korekcji liczbę odczytów, dla których liczba różnic względem genomu spadła [128] lub przedstawiono histogram zmiany liczby różnic $\xi_{\mathrm{ndist}}(d_{\mathrm{e}})$ [89, 100] (gdzie $\xi_{\mathrm{ndist}}(d_{\mathrm{e}})$ jest wyrażoną procentowo częścią odczytów, których liczba różnic od podsłowa genomu do którego zostały dopasowane jest równa d_{e}; ze względu na niesprecyzowanie w literaturze pojęcia „liczby różnic", w dalszej części jako wartość d_{e} będzie przyjmowana odległość edycyjna zgodna z wartością uzyskiwaną w wyniku mapowania odczytów); w pracy [89] jako dopuszczalne wartości d_{e} przyjęto też „odczyt niezmapowany" oraz „odczyt pominięty". W wybranych przypadkach jako granulację analizy przyjęto całe odczyty i za prawidłowe odczyty uznano te, które zostały dopasowane do genomu bez żadnych różnic, np. [25, 97, 98, 128] albo były możliwe do zmapowania [134, 192].

Detekcja wariantów Wpływ korekcji na jakość wyników detekcji wariantów nie został jak dotąd dokładnie przeanalizowany. Zgodnie z wiedzą autora problem ten został poruszony tylko w trzech pracach. Autorzy algorytmu SAMDUDE [79] przeprowadzili analizę wpływu korekcji na detekcję wariantów dwóch ludzkich chromosomów poprzez określenie zmian wartości ξ_{sens}, ξ_{prec} oraz ξ_{F1} liczby wariantów przed oraz po korekcji. W pracy [110] zamieszczono wyniki wyznaczenia ξ_{sens} oraz ξ_{prec} wariantów *Escherichia coli*, jednak tylko dla oceny kryterium jakościowego algorytmu Quake, bez wykonania porównania z innymi algorytmami. Dodatkowo wykonano detekcję wariantów ludzkiego genomu, wykazując, że w wyniku korekcji pozwala ona na znalezienie większej liczby wariantów, bez ich weryfikacji. Z kolei dla oceny algorytmu BFC i innych rozwiązań zagadnienie detekcji wariantów zostało poruszone w sposób uproszczony [128] — oszacowano liczbę FP polimorfizmów pojedynczych nukleotydów poprzez porównanie zestawu wariantów uzyskanego z kontigów wygenerowanych w oparciu o skorygowane odczyty z wariantami uzyskanymi z dopasowania nieskorygowanych odczytów do genomu. We wszystkich powyższych pracach wzięto pod uwagę wyłącznie warianty typu SNP. Dodatkowo w pracy [134] przedstawiono wyniki wpływu na detekcję wariantów, jednak tylko odczytów z sekwencjonowania strategii RNA-Seq.

Narzędzia detekcji wariantów zostały wykorzystane także w eksperymentach opisanych w pracy [98], jednakże ich zadaniem była wyłącznie generacja genomu dla narzędzia mapowania, poprzez dodanie do modelowego genomu referencyjnego wariantów, a następnie eliminację ze zbioru różnic między odczytami a genomem wariantów, celem pozostawienia podzbioru będącego różnicami uznanymi

za błędy. Nie dokonano jednak żadnej oceny jakości detekcji wariantów.

Inne rozwiązania W nierecenzowanej pracy przeglądowej [153] opisano wykorzystanie techniki *Safe-SeqS* (*Safe-sequencing system*) umożliwiającej sekwencjonowanie i uzyskanie konsensusowej postaci odczytów wysokiej jakości. Technika ta opiera się na znakowaniu sekwencjonowanych nici *unikalnymi identyfikatorami cząsteczkowymi* (ang. *unique molecular identifiers — UMI*) przed rozpoczęciem namnażania metodą PCR. Identyfikatory te pozwalają na zgrupowanie namnożonych cząsteczek oraz określenie, które odczyty reprezentują ten sam fragment genomu. W rezultacie, w oparciu o różnice między tymi odczytami, możliwe jest określenie wiarygodnej postaci odczytu, stanowiącego wzór do oceny korekcji poszczególnych odczytów.

W pracy [142] zaproponowano określenie miary jakości korekcji poprzez zliczenie ile spośród 31-merów obecnych w odczytach nie jest obecnych w genomie oraz ile spośród 31-merów obecnych w genomie nie występuje w odczytach. Z kolei w pracy [154] zdefiniowano pojęcia *głębokości pokrycia* (ang. *depth of coverage*) oraz *zasięgu pokrycia* (ang. *breadth of coverage*). Definicja pierwszego pojęcia jest zbieżna z głębokością sekwencjonowania zdefiniowaną wzorem (3.2), natomiast druga wartość jest równa części genomu, który jest pokryty przez przynajmniej jeden odczyt. W obu przypadkach odczyty są uznawane za pokrywające, jeśli ich sekwencje są podsłowami sekwencji genomu. Jako kryterium oceny korekcji przyjmowana jest zmiana głębokości i zasięgu pokrycia.

4.4.2 Kryteria wydajnościowe

Prawie wszystkie spośród omówionych algorytmów zostały poddane przez autorów eksperymentom oceniającym czas pracy algorytmu τ oraz szczytowe zapotrzebowanie na pamięć operacyjną μ, w zależności od przetwarzanego zestawu danych. Zagadnienie to posiada znaczenie ze względu na duże zapotrzebowanie niektórych algorytmów na te zasoby [97]. Korekcja odczytów często stanowi tylko jeden z etapów przygotowania danych do uruchomienia potoku algorytmów. Jeżeli czas korekcji okaże się zbyt długi, może zacząć odgrywać dominującą rolę w tym potoku, opóźniając uzyskanie ostatecznych wyników. Jednocześnie zbyt duże zapotrzebowanie na pamięć operacyjną może stanowić nieprzekraczalną barierę, jeżeli w danym laboratorium nie ma dostępnej odpowiedniej stacji roboczej, a nawet jeżeli odpowiednie zasoby istnieją, duże wymagania pamięciowe mogą skutkować całkowitym zaangażowaniem takiej stacji do wykonania zadania korekcji, uniemożliwiając współbieżne wykonanie innych zadań [97].

Należy zauważyć, że w niektórych pracach postawiono, choć nie udowodniono hipotezy, że korekcja może zredukować zapotrzebowanie na zasoby kolejnych w potoku przetwarzania algorytmów. Jeżeli takie zjawisko występuje, zbyt wysokie wymagania samej korekcji mogą wyeliminować znaczenie tego zjawiska.

W pracach [98, 138] przeprowadzono analizę skalowalności, rozumianej jako wpływ zmiany liczby wątków roboczych algorytmu na czas obliczeń. Dobra skalowalność umożliwia efektywne wykorzystanie wielordzeniowych procesorów lub rozbudowanych, wieloprocesorowych stacji roboczych. Eksperymenty tego rodzaju mogą wykazać, czy algorytm posiada wąskie gardła, związane np. ze zbyt niską przepustowością wątku głównego (w modelu klient-serwer) lub czasem dostępu do pamięci masowej.

4.4.3 Kryteria techniczne

W praktyce istotnym zagadnieniem są techniczne kwestie działania algorytmów. Przykładowo, niektóre, szczególnie wczesne algorytmy zmieniają format danych wejściowych lub nie zachowują dopasowania odczytów sparowanych [89], co stanowi utrudnienie w kolejnych etapach przetwarzania odczytów. Inną pożądaną cechą algorytmu jest jego zdolność do przetwarzania odczytów różnych długości [89]. Ponadto wybrane algorytmy mogą przyjmować odczyty wejściowe w postaci plików FASTA [89]. Ze względu na rozmiar danych wejściowych i wyjściowych korzystna jest możliwość przetwarzania danych w formie skompresowanej. Duże znaczenie ma prostota użycia algorytmu, w szczególności mała liczba parametrów algorytmu, które powinny być dobierane w prosty i intuicyjny sposób, a najlepiej dobierane automatycznie przez algorytm korekcji.

Rozdział 5

Nowa metoda korekcji

5.1 Wprowadzenie

Przedstawione w rozdziale 4. algorytmy oraz ich implementacje wykorzystują różne strategie rozwiązywania trudności związanych z korekcją odczytów z sekwencjonowania Illumina uzyskanych w strategii WGS. Wiele spośród tych strategii jest wspólnych dla różnych algorytmów, co z jednej strony dowodzi dużej skali wykorzystania ich zalet, jednak z drugiej wskazuje na powszechność występowania wad lub słabości danych rozwiązań. Skalę popularności niektórych z nich można nazwać nawet „modą", szczególnie wykorzystania filtrów Blooma do przechowywania spektrum k-merów, pozwalających na jego zwięzłą reprezentację, ale także umożliwiających zachowanie wyłącznie binarnej informacji o przynależności k-meru do spektrum oraz podatnych na pojawianie się nieprawidłowych odpowiedzi twierdzących. Znaczenie takich błędów, co wynika z równania (3.4), w średnim przypadku jest niewielkie, jednak w przypadku pesymistycznym, przy niekorzystnym zbiorze k-merów wejściowych, potencjalnie może prowadzić do uzyskania nieoczekiwanie słabych wyników.

Innym przykładem popularnego rozwiązania jest silne oparcie algorytmu na współczynnikach jakości. Jak zostało wspomniane w poprzednich rozdziałach, adekwatność wartości takich współczynników do rzeczywistej skali błędów w odczytach nie jest idealna, zatem pomimo istnienia bardzo odmiennych strategii wykorzystania współczynników należy położyć nacisk na dywersyfikację źródeł informacji o jakości poszczególnych fragmentów odczytów. Ponadto niektóre spośród technik charakteryzują się mniejszą liczbą poziomów kwantyzacji, jak NovaSeq, w której są zawarte w zbiorze Q_4 mocy 4 [4], co ogranicza dokładność informacji o lokalizacji błędów.

Większość spośród omówionych w rozdziale 4. algorytmów nie została opracowana z myślą o korekcji błędów typu indel. Faktem jest, że błędy te w porównaniu

do błędów substytucji są w odczytach Illumina dużo rzadsze. Nie oznacza to jednak braku ich znaczenia. Odczyty często są wykorzystywane do rozwiązywania zadania detekcji wariantów, a liczba wariantów typu krótki indel w jednym genomie ludzkim wynosi ok. $3 \cdot 10^5$–$4 \cdot 10^5$ (przy ok. $3 \cdot 10^6$ wariantów typu SNP) [49]. Jest to zatem znaczna wartość, a obecność błędów typu indel może wpłynąć na jakość wyników poszukiwania takich wariantów. Ponadto należy mieć na uwadze, że przeprowadzenie korekcji odczytu z obecnym błędem typu indel przy pomocy algorytmu niewyposażonego w mechanizmy detekcji błędów tego rodzaju może skutkować niewłaściwym wykorzystaniem strategii algorytmu i uzyskaniem błędnych wyników lub doprowadzeniem do wyznaczenia wielu ścieżek, utrudniając wiarygodny wybór najlepszej z nich.

Różnorodność koncepcji zastosowanych w algorytmach pozwala przypuszczać, że istnieje potencjał do znalezienia kolejnych, być może lepszych strategii. Jednocześnie niedoskonałość istniejących implementacji, w tym wysokie zapotrzebowanie na zasoby (w szczególności znaczny czas obliczeń) motywują opracowanie innych, szybszych rozwiązań. W ostatnich latach rozwinięto nowe algorytmy i koncepcje, które mogą okazać się przydatne w implementacji lepszych narzędzi korekcji odczytów. Przykładem jest algorytm KMC [59, 60, 117], który znajduje zastosowanie w niektórych algorytmach korekcji jako narzędzie zliczania k-merów, jednak bez pełnego wykorzystania mechanizmów udostępnianych przez dołączone do niego KMC API.

5.2 Narzędzia KMC i KMC tools

5.2.1 Możliwości narzędzia KMC

Algorytm KMC, omówiony w podrozdziale 3.4.2, umożliwia zliczanie k-merów, generując w rezultacie ich bazę danych, zapisywaną w postaci dwóch plików dyskowych: pliku prefiksów i sufiksów. Format plików jest zbliżony do struktury danych tworzonej po wczytaniu bazy danych do pamięci operacyjnej. Odczyt plików bazy, utworzenie struktury w pamięci oraz dostęp do niej wraz z wykonaniem podstawowych operacji jest możliwy za pośrednictwem KMC API, będącego biblioteką programistyczną języka C++.

Struktura bazy danych

W narzędziu KMC posłużono się dwiema wersjami formatu bazy danych k-merów. Poniższa formalizacja tego formatu została opracowana przez autora niniejszej pracy w oparciu o dokumentację techniczną tego narzędzia. W wersji 1 (opierając się na suplemencie pracy [59]) przyjęta jest długość prefiksu k-meru równa a. Poszczególne prefiksy nie są zapisywane w pełnej postaci w bazie. Plik prefiksów zawiera ciąg (tablicę) indeksów $(\mathbf{v}_\iota^{\mathrm{ind}})$, gdzie $\iota \in 0, 1, \ldots, 4^a - 1$. Prefiks $\underset{a}{\kappa}$

danego k-meru κ jest jednoznacznie przyporządkowany elementowi ciągu $(\mathbf{v}_\iota^{\mathrm{ind}})$ przy pomocy funkcji mieszającej $h(\cdot)$, będącej bijekcją. Funkcja mieszająca gwarantuje zachowanie leksykograficznej kolejności prefiksów, tzn. $h(\mathfrak{w}) < h(\mathfrak{w}') \Leftrightarrow$ słowo $\mathfrak{w}$ jest leksykograficznie mniejsze od $\mathfrak{w}'$. Każdy indeks $\mathbf{v}_\iota^{\mathrm{ind}}$ reprezentuje jeden z kolejnych prefiksów. Plik sufiksów zawiera ciąg $(\mathbf{v}_i^{\mathrm{suf}})$ par $(\overset{k-a}{\underset{a}{\kappa}}, \eta)$, zawierających sufiks pewnego k-meru κ oraz liczebność tego k-meru. Wszystkie sufiksy danego prefiksu są obecne w tablicy na pozycjach $\mathbf{v}_i^{\mathrm{suf}}, \mathbf{v}_{i+1}^{\mathrm{suf}}, \ldots, (\mathbf{v}_j^{\mathrm{suf}} - 1)$, gdzie $i = \mathbf{v}_\iota^{\mathrm{ind}}, j = \mathbf{v}_{\iota+1}^{\mathrm{ind}}$, a ι jest indeksem prefiksu.

Powyższa struktura danych umożliwia szybkie wykonywanie m.in. dwóch operacji:

- wyszukania k-meru o zadanej sekwencji i zwrócenie binarnej informacji o jego obecności w bazie, a w przypadku odpowiedzi twierdzącej uzyskanie liczebności; zadanie to polega na wyborze elementów $i = \mathbf{v}_\iota^{\mathrm{ind}}$ (w oparciu o prefiks k-meru $\underset{a}{\kappa}$) oraz $j = \mathbf{v}_{\iota+1}^{\mathrm{ind}}$ oraz binarnym wyszukaniu w przedziale $\mathbf{v}_i^{\mathrm{suf}}, \mathbf{v}_{i+1}^{\mathrm{suf}}, \ldots, (\mathbf{v}_j^{\mathrm{suf}} - 1)$ sufiksów o odpowiedniej sekwencji; operacja wymaga obecności całej bazy danych w pamięci,

- iteracji po zbiorze k-merów, która polega na wyborze kolejnych elementów tablicy $(\mathbf{v}_\iota^{\mathrm{ind}})$, a dla każdego z nich na wyborze kolejnych, odpowiednich elementów $(\mathbf{v}_i^{\mathrm{suf}})$; ze względu na jednoznaczne przyporządkowanie prefiksu do indeksu ciągu $(\mathbf{v}_\iota^{\mathrm{ind}})$ (odwracalność funkcji $h(\cdot)$ umożliwia uzyskanie postaci całego k-meru); operacja wymaga obecności w pamięci tylko części bazy danych.

Obie te procedury mogą zostać przeprowadzone przy wprowadzeniu dodatkowych ograniczeń na wymaganą liczebność k-merów.

Struktura bazy danych w wersji 2 (generowanej przy pomocy narzędzia KMC od wersji 2), opierając się na suplemencie pracy [60] oraz dokumentacji [114], jest odmienna, co wynika z zastosowania w tej wersji algorytmu zliczania opartego na wykorzystaniu sygnatur. Różnica dotyczy formatu pliku prefiksów, który składa się z dwóch części: tablicy mapowania oraz wielu tablic indeksów. Ciąg mapowania $(\mathbf{v}_j^{\mathrm{map}})$ zawiera przyporządkowania sygnatur do ciągów indeksów. Każdy element tablicy reprezentuje jedną sygnaturę długości k'. Pozycja tego elementu jest jednoznacznie przyporządkowana sekwencji tej sygnatury w oparciu o funkcję mieszającą $h'(\cdot)$ oraz zawiera odniesienie do jednej, właściwej dla danej sygnatury, tablicy indeksów. Każdy z ciągów indeksów zawiera 4^a elementów, które reprezentują sufiksy w sposób analogiczny jak w wersji 1 bazy danych.

Struktura w tej wersji pozwala na wykonywanie podobnych działań jak w wersji 1, jednakże znalezienie k-meru musi zostać poprzedzone wyznaczeniem właściwej tablicy prefiksów poprzez odczytanie wartości z tablicy mapującej sygnaturę na prefiks. W rezultacie liczba operacji koniecznych do wyszukania może ulec zwiększeniu. Co więcej, w przypadku iteracji po zbiorze k-merów nie jest

zapewniona ich leksykograficzna kolejność, gdyż prefiksy są posortowane leksykograficznie wyłącznie w ramach jednej tablicy.

Użyteczne cechy algorytmu KMC

Algorytm zliczania jest parametryzowany nie tylko długością k-meru, ale także m.in. wartością progu obcięcia η_{cut}, domyślnie $\eta_{\text{cut}} = 2$. Ponadto domyślnie następuje zliczanie k-merów kanonicznych i wyłącznie k-mery w takiej postaci są zapisywane w bazie danych. Liczniki k-merów są inkrementowane zgodnie z arytmetyką nasyceniową, co w rezultacie skutkuje górnym ograniczeniem wartości liczników. Implementacja algorytmu umożliwia łatwą zmianę tego ograniczenia, jednak domyślna wartość wynosząca 255 powinna być wystarczająca dla wielu problemów przetwarzania odczytów z sekwencjonowania WGS, gdyż wartości liczebności (dla umiarkowanej głębokości sekwencjonowania) osiągają w większości wartości znacznie mniejsze, czego przykład można zaobserwować na rys. 4.1.

Zaletą formatu bazy KMC w stosunku do filtrów Blooma jest przechowywanie zarówno pełnej postaci zliczonych k-merów, jak też ich liczebności. Jednocześnie należy oczekiwać, że w typowym przypadku rozmiar bazy danych będzie znacznie mniejszy niż przy prostej implementacji przy pomocy tablicy mieszającej, dzięki pominięciu zapamiętywania sekwencji prefiksów.

5.2.2 Przetwarzanie baz danych KMC

Przedstawione wyżej funkcje KMC API technicznie zapewniają duże możliwości przetwarzania danych, jednak w kontekście korekcji odczytów dostęp za ich pośrednictwem do pełnej bazy danych k-merów może nie odbywać się z wystarczającą szybkością oraz stawiać zbyt duże zapotrzebowanie na pamięć. W szczególności trudności rodzi przeprowadzenie typowej operacji algorytmów opartych na spektrum k-merów, którą stanowi sprawdzanie obecności k-meru w spektrum, co poza wyszukaniem k-meru wymaga uwzględnienia jego liczebności w celu uznania za nieobecne w spektrum k-merów cechujących się liczebnością mniejszą od progu obcięcia. KMC API pozwala na pominięcie takich k-merów poprzez ustawienie ograniczeń, jednak procedura wyszukiwania k-meru wymaga obecności w pamięci całej bazy danych, zawierającej także k-mery o liczebnościach poniżej progu, których, zgodnie z rys. 4.1, może być bardzo dużo. Tym samym ich obecność może stanowić znaczny narzut pamięciowy oraz generować zwiększony koszt wyszukania w bazie.

Znaczne możliwości przetwarzania zbiorów k-merów są zapewniane przez narzędzie KMC tools [117], którego zadaniem jest szybkie (w tym wykorzystując przetwarzanie współbieżne) przetwarzanie baz danych KMC przy niskim zapotrzebowaniu na pamięć. Obejmuje ono wykonywanie matematycznych operacji na zbiorach (np. suma i przecięcie zbiorów k-merów), modyfikacje liczebności

czy filtracja zbiorów. O dużym potencjale i praktycznej użyteczności narzędzia świadczy fakt wykorzystania go wraz z KMC w potokach przetwarzania zbiorów zestawów odczytów w celu detekcji wariantów. Ich wykorzystanie w miejsce oryginalnych narzędzi umożliwia kilkukrotne skrócenie czasu przetwarzania danych oraz kilkunastokrotną redukcję zapotrzebowania na pamięć [117].

Z punktu widzenia korekcji należy stwierdzić, że spośród możliwości tego narzędzia warte uwagi są funkcje filtracji bazy k-merów w oparciu o próg obcięcia (w celu redukcji rozmiaru bazy danych wczytywanej do pamięci operacyjnej) oraz konwersji bazy z wersji 2 do wersji 1 w celu przyspieszenia wyszukiwania k-meru. Narzędzie KMC tools zostało wyposażone również w funkcję generacji histogramu bazy k-merów, która jednak nie została wykorzystana ze względu na satysfakcjonujące w tej roli działanie KMC API.

5.3 Algorytm BLESS

Punkt wyjścia opracowania nowego algorytmu stanowi algorytm BLESS w wersji 0.12, przedstawiony w pracy [97]. Tym samym algorytm ten będzie posiadał podobną strukturę oraz częściowo podobne działanie. Wynika z tego konieczność dokładnego omówienia działania algorytmu BLESS, które następnie zostanie rozwinięte w celu wyjaśnienia nowego algorytmu.

5.3.1 Dodatkowe pojęcia i oznaczenia

W celu sprecyzowania wybranych kwestii związanych z omówieniem obu algorytmów należy wprowadzić dodatkowe pojęcia. Końcem $5'$ nazywamy pierwszy ($\mathfrak{r}[0]$), a końcem $3'$ ostatni ($\mathfrak{r}[\ell-1], \ell = |\mathfrak{r}|$) symbol odczytu $\mathfrak{r}$. Korekcję (lub inne działanie) wykonywaną *w stronę końca* nazywamy czynność związaną z sukcesywnym zmniejszaniem (w stronę końca $5'$) albo zwiększaniem (w stronę końca $3'$) indeksu wskazującego na pozycję w odczycie. Dodatkowo należy przyjąć, że indeks ten nie musi przyjmować wartości z zakresu ograniczonego długością odczytu — może odnosić się również do symboli sekwencji skonkatenowanej z dowolnym końcem odczytu. W takim przypadku umieszczanie symboli na pozycjach $a < 0$ będzie nazywane *rozszerzaniem początkowego regionu* (*rozszerzaniem w lewo*), a na pozycjach $a \geqslant \ell$ *rozszerzaniem końcowego regionu* (*rozszerzaniem w prawo*).

Właściwą ścieżką korekcji nazywamy ścieżkę korekcji, która spełnia określone wymagania dotyczące zaufania pokrywających ją oligomerów. *Symbolem końcowym* k-meru κ nazywamy symbol $\kappa[0]$ lub $\kappa[k-1]$, odpowiednio w trakcie korekcji w kierunku końca $5'$ lub $3'$ (tj. korekcji kolejnych symboli o coraz niższych lub wyższych indeksach w odczycie). Symbolami ✔ oraz ✘ oznaczamy odpowiednio k-mery prawidłowe i błędne.

Obiecującym k-merem nazywamy dowolny k-mer zaufany (tj. obecny w spektrum) lub k-mer niezaufany, który jednak w algorytmie jest traktowany jak k-mer

zaufany. Sekwencje (tzn. odczyty lub k-mery) zmodyfikowane przez algorytm oznaczane są symbolem gwiazdki, np. κ^* oznacza k-mer κ, którego sekwencja została zmodyfikowana w określony sposób. *Pierwszym k-merem* nazywamy k-mer $\mathfrak{r}[0, k-1]$.

Regionem odczytu nazywamy parę uporządkowaną (a, b), gdzie $0 \leqslant a \leqslant b \leqslant \ell - k$, a ℓ jest długością odczytu ($k < \ell$). Indeks a wskazuje pierwszy symbol pierwszego k-meru (w znaczeniu podsłowa odczytu) w ciągu k-merów regionu. Z kolei b wskazuje pierwszy symbol ostatniego spośród takich k-merów. Każdy region $(0, b)$ lub $(a, \ell - k)$ nazywamy *regionem skrajnym*, odpowiednio *początkowym* i *końcowym*; pozostałe regiony nazywamy *regionami wewnętrznymi*. Symbolem pionowych kresek będzie oznaczana długość regionu: $|(a, b)| = b - a + 1$.

Działanie algorytmów zostanie omówione poprzez przedstawienie pseudokodów. Pomocniczo sprecyzowano dodatkowe procedury, które zostaną wykorzystane później w celu uproszczenia zapisu. Zostały one przedstawione w formie pseudokodów 1 oraz 2.

5.3.2 Przebieg algorytmu BLESS

Poniższy opis opracowano w oparciu o kod źródłowy implementacji algorytmu dostępnej na stronie [96]. Wszystkie wykorzystane w opisie definicje matematyczne zostały wprowadzone przez autora niniejszej pracy. Wykorzystane oznaczenia kroków korekcji są zgodne z oznaczeniami obecnymi w kodzie źródłowym implementacji. W formie pseudokodu 3 przedstawiono pomocnicze funkcje wykorzystane do wyjaśnienia algorytmu BLESS.

Parametry algorytmu

W tabeli 5.1 przedstawiono wewnętrzne stałe algorytmu. Z kolei w tabeli 5.2 przedstawiono parametry, które mogą być zadane przez użytkownika, a w tabeli 5.3 zamieszczono wybór wewnętrznych zmiennych algorytmu, wykorzystanych w pseudokodach.

Przygotowanie danych

Pierwszą część algorytmu BLESS stanowi wstępna analiza plików wejściowych, polegająca na sprawdzeniu poprawności ich formatu, oraz zliczanie k-merów. Zadana przez użytkownika długość k-meru musi być nieparzysta, ze względu na podejmowanie decyzji o wyznaczeniu odwróconego dopełnienia k-meru w oparciu o symbol środkowy, tzn. $\kappa[\lceil k/2 \rceil]$. Decyzja ta jest podejmowana w ramach sprawdzania obecności k-meru w spektrum. Etap zliczania k-merów składa się z następujących kroków: dystrybucji k-merów do plików, zliczania przy pomocy tablicy mieszającej, eliminacji k-merów o liczebności poniżej progu (próg obcię-

Pseudokod 1 Pomocnicze funkcje

Wejście: $k, \ell = |\mathfrak{r}|, \mathcal{K}_{\mathfrak{cut}}$

Require: $\psi = (a, b)$
function $\textsc{Beg}(\psi)$
    return a
end function

Require: $\psi = (a, b)$
function $\textsc{End}(\psi)$
    return b
end function

function $\textsc{Pred}(\Psi, \psi)$ ▷ Region poprzedzający region ψ
    if $\exists_{\psi_\mathrm{p} \in \Psi} (\textsc{End}(\psi_\mathrm{p}) < \textsc{Beg}(\psi) \wedge \forall_{\substack{\psi' \in \Psi \\ \textsc{Beg}(\psi') > \textsc{End}(\psi_\mathrm{p})}} (\textsc{Beg}(\psi') \geqslant \textsc{Beg}(\psi)))$ **then**
        return ψ_p
    else
        return $\varnothing$
    end if
end function

Require: $\textsc{Beg}(\psi) - \textsc{End}(\psi_\mathrm{p}) > 1$
function $\textsc{UntrustedReg}(\psi_\mathrm{p}, \psi)$
    return $(\textsc{End}(\psi_\mathrm{p}) + 1, \textsc{Beg}(\psi) - 1)$
end function

Require: $0 \leqslant a \leqslant \ell - k$
function $\kappa(\mathfrak{r}, a)$
    return $\mathfrak{r}[a, a + k - 1]$
end function

Require: $(\kappa, \eta) \in \mathcal{K}_{\mathfrak{cut}}$
function $\eta(\kappa)$
    return η
end function

Require: $a = -1 \vee a = \ell - k + 1$
function $\kappa_{\textsc{dummy}}(\mathfrak{r}, a)$ ▷ $(k-1)$-mer odczytu
    if $a = -1$ **then**
        return $\mathfrak{r}[0, k - 2]$
    else
        return $\mathfrak{r}[a, a + k - 2]$
    end if
end function

Pseudokod 2 Pomocnicze funkcje, cd.

```
30: function MinReg(Ψ)                                  ▷ Region najbliższy końcowi 5′
31:     if ∃_{ψ∈Ψ} ∀_{ψ′∈Ψ, ψ′≠ψ} (End(ψ) < Beg(ψ′)) then
32:         return ψ
33:     else
34:         return ∅
35:     end if
36: end function

37: function MaxReg(Ψ)                                  ▷ Region najbliższy końcowi 3′
38:     if ∃_{ψ∈Ψ} ∀_{ψ′∈Ψ, ψ′≠ψ} (End(ψ′) < Beg(ψ)) then
39:         return ψ
40:     else
41:         return ∅
42:     end if
43: end function

44: function Trusted(κ)
45:     if κ ∈ 𝒦_cut ∨ rev(κ) ∈ 𝒦_cut then
46:         return true
47:     else
48:         return false
49:     end if
50: end function
```

Pseudokod 3 BLESS — pomocnicza funkcja

```
1: function Apply(𝔴, Π)
2:     for all (a, c) ∈ Π do
3:         𝔴[a] ← c
4:     end for
5:     return 𝔴
6: end function
```

Tabela 5.1: Wewnętrzne stałe algorytmu BLESS

Parametr	Wartość	Znaczenie
θ_{FP}	1	Przyjęta liczba k-merów w błędnym obszarze uznanych za prawidłowe na skutek błędów filtru Blooma
$\theta_{\text{MIN}\Delta}$	10	Minimalna różnica oceny między najlepszą a drugą w kolejności ścieżką korekcji
$\theta_{\text{MN\_ERR}}$	2	Minimalna długość błędnego regionu
$\theta_{\text{MN\_SOLID}}$	2	Minimalna długość prawidłowego regionu
$\theta_{\text{MX\_ADJ}}$	4	Maksymalna liczba pozycji regionu sprawdzanych na obecność symboli niskiej jakości
$\theta_{\text{MX\_LQ}}$	3	Maksymalna liczba pozycji dla metody siłowej korekcji pierwszego k-meru
$\theta_{\text{Q\_LOW}}$	3	Górne ograniczenie wartości niskich współczynników jakości

Tabela 5.2: Wybór zewnętrznych parametrów algorytmu BLESS

Zmienna	Domyślna wartość	Znaczenie
k	Długość oligomeru	
η_{cut}	Wyznaczany w oparciu o histogram k-merów	Próg obcięcia
$\theta_{\text{MX\_E}}$	5	Maksymalna liczba pozycji rozszerzających skrajny region[1]
fp_prob	0,001	Zakładane prawdopodobieństwo nieprawidłowej odpowiedzi twierdzącej filtru Blooma
num_clusters	100	Liczba plików pomocniczych wykorzystywanych w redukcji wpływu nieprawidłowych odpowiedzi twierdzących filtru Blooma
seed	0	Ziarno generatora liczb pseudolosowych, wykorzystanego do wyznaczania funkcji mieszających filtru Blooma
verify	**true**	Binarny parametr wyznaczający weryfikację redukującą wpływ nieprawidłowych odpowiedzi twierdzących filtru Blooma

cia η_{cut} musi zostać określony przez użytkownika), zapisania uzyskanego spektrum k-merów w filtrze Blooma.

[1] W instrukcji algorytmu błędnie podano wartość 2.

Tabela 5.3: Wybór wewnętrznych zmiennych algorytmu BLESS

Zmienna	Znaczenie
$A_{\text{Q\_LOW}}$	Zbiór pozycji odczytu o niskich współczynnikach jakości
θ_{ext}	Liczba pozycji rozszerzających region dla bieżącej ścieżki
$\theta_{\text{first\_mod}}$	Najmniejsza pozycja zmodyfikowanego symbolu w ścieżce
$\theta_{\text{last\_mod}}$	Największa pozycja zmodyfikowanego symbolu w ścieżce
θ_{lq}	Liczba pozycji dla algorytmu siłowego korekcji pierwszego k-meru
Π	Ścieżka korekcji (zbiór par uporządkowanych)
Π_{n}	Nowa, skopiowana ścieżka korekcji
$\mathbf{\Pi}$	Zbiór ścieżek korekcji
ψ	Region odczytu
ψ_{p}	Poprzedni region odczytu
Ψ_{trust}	Zbiór prawidłowych regionów odczytu
Ψ_{untrust}	Zbiór błędnych regionów odczytu
sh_ns	Wartość binarna dotycząca obecności krótkiego błędnego regionu

W algorytmie przewidziano obsługę odczytów sparowanych poprzez ich niezależną korekcję (naprzemiennie odczytów z jednej pary). W implementacji nie uwzględniono pracy z plikami w formie skompresowanej. Wykonanie całości algorytmu jest przeprowadzane jednowątkowo.

Wyznaczanie regionów

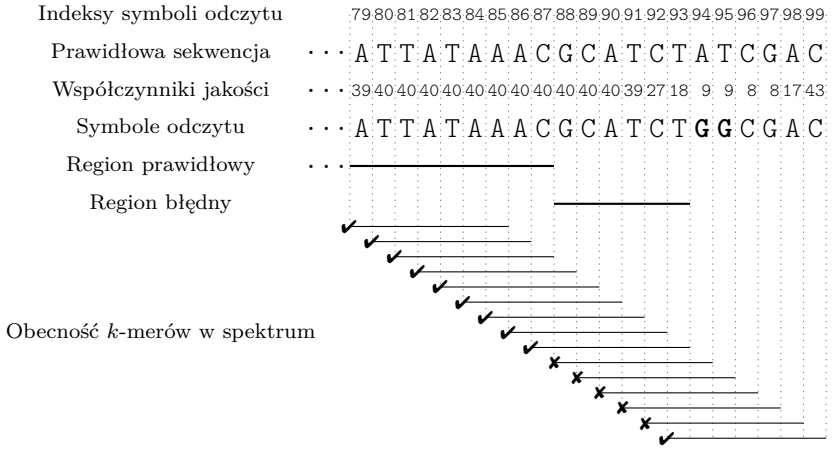
W dalszej kolejności przeprowadzana jest korekcja kolejnych odczytów. Pierwszy krok stanowi zmiana wszystkich symboli N odczytu na symbol A. Uzasadnienie wyboru takiej strategii nie zostało przez autorów podane. Następnie wyznaczane są *regiony prawidłowe* $(a_i, b_i) \in \Psi_{\text{trust}}$, a w oparciu o nie następuje wyznaczenie *regionów błędnych*. W dalszej części, jeśli nie zaznaczono inaczej, pod pojęciem *region* będziemy rozumieli region błędny.

Procedura wyznaczania regionów prawidłowych została przedstawiona na pseudokodach 43–45 zamieszczonych w dodatku A. W pierwszej kolejności (krok 0.0) następuje określenie przynależności każdego k-meru odczytu do spektrum (linie 2–17 pseudokodu). Następnie uzyskane w ten sposób wstępne regiony podlegają skróceniu lub eliminacji: w krokach 0.2 oraz 0.3 eliminowane są zbyt krótkie regiony prawidłowe i błędne poprzez zwiększenie długości sąsiadujących regionów (odpowiednio linie 18–22 oraz 23–30). Regiony błędne mogą też zostać wydłużone jeśli ich długość jest zbyt mała, co może być skutkiem błędów filtru Blooma (krok 0.4, linie 31–41). Osobno jest rozważany przypadek, gdy w odczycie wyznaczono dwa prawidłowe regiony, a region błędny między nimi jest krótszy od k. W takim przypadku może nastąpić eliminacja jednego z regionów (krok 0.5, linie 42–58). Regiony błędne mogą też mieć zwiększane długości w kierunku $5'$, jeśli k-mery znajdujące się w poprzedzających je regionach prawidłowych

zaczynają się lub kończą symbolami o niskich współczynnikach jakości (krok 0.6, linie 59–92). W kroku 0.7 (linie 93–99) następuje wykrywanie błędnych regionów krótszych niż k, których obecność skutkuje przeprowadzeniem korekcji odczytu w inny sposób.

Wyznaczanie regionów błędnych w oparciu o regiony prawidłowe odbywa się zgodnie z poniższym przykładem. Przyjmując że dla pewnego odczytu określono prawidłowe regiony (a_1, b_1) oraz (a_2, b_2), $b_1 < a_2$, wówczas każdy k-mer $\mathfrak{r}[a_1, a_1 + k - 1]$, $\mathfrak{r}[a_1 + 1, a_1 + k]$, …, $\mathfrak{r}[b_1, b_1 + k - 1]$ jest k-merem przyporządkowanym pierwszemu regionowi prawidłowemu. Z kolei k-mery $\mathfrak{r}[a_2, a_2 + k - 1]$, $\mathfrak{r}[a_2 + 1, a_2 + k]$, …, $\mathfrak{r}[b_2, b_2 + k - 1]$ są przyporządkowane regionowi drugiemu. Przyjmując że $a_1 > 0$ oraz $b_2 < \ell - k$ (tzn. symbole odczytu znajdujące się blisko końców zawierają błędy, a w konsekwencji istnieją skrajne błędne regiony; warunek ten jest spełniony tylko w niektórych odczytach) indeksy prawidłowych regionów wyznaczają także 3 regiony błędne: $(0, a_1 - 1)$, $(b_1 + 1, a_2 - 1)$ oraz $(b_2 + 1, \ell - k - 1)$, zdefiniowane w sposób analogiczny, stanowiące elementy zbioru Ψ_{untrust}. Tym samym suma zbiorów k-merów wszystkich regionów prawidłowych i błędnych stanowi zbiór wszystkich k-merów odczytu. W implementacji algorytmu nie przewidziano przechowania w pamięci zbioru regionów błędnych, gdyż wartości jego elementów mogą być wyznaczone na bieżąco w oparciu o sąsiednie regiony prawidłowe (zgodnie z funkcją `UntrustedReg` w pseudokodzie 1).

Przykładowe regiony prawidłowy oraz błędny zostały przedstawione na rys. 5.1. Część spośród 7-merów została przydzielone do regionu błędnego $(87, 93)$. Ponadto wyznaczono region prawidłowy $(a, 86)$ dla pewnego a, $0 \leqslant a \leqslant 79$. W regionie prawidłowym znajdują się wyłącznie prawidłowe 7-mery, natomiast w regionie błędnym występuje prawidłowy 7-mer o indeksie 93, zaliczony do tego regionu (np. w kroku 0.2, w którym następuje eliminacja regionów prawidłowych krótszych niż $\theta_{\text{MN\_SOLID}}$).



Rysunek 5.1: Regiony w algorytmie BLESS; $k = 7, \ell = 100$; pogrubieniem zaznaczono błędy substytucji w odczycie

Korekcja regionów

Kolejnym etapem korekcji odczytu stanowi modyfikacja symboli obecnych w błędnych regionach. Warunki wykonania poszczególnych przypadków korekcji zostały przedstawione na pseudokodzie 4, przy czym w tym etapie rozróżniane są dwa przypadki: korekcja regionów wewnętrznych oraz regionu skrajnego końcowego, które są wykonywane w kierunku $3'$ (linie 7 oraz 19), a także przypadek korekcji skrajnego początkowego regionu, który jest wykonywany w kierunku $5'$ (linia 15). W pseudokodach dla uproszczenia przyjęto, że reprezentowane przez ścieżki zmiany odczytu są wprowadzane od razu po korekcji danego regionu (np. pseudokod 5, linia 23); w oryginalnym algorytmie są one przechowywane w pliku i stosowane później po eliminacji błędów filtru Blooma.

Pseudokod 4 Korekcja regionów algorytmu BLESS

Wejście: $\Psi_{\text{trust}}, \text{sh\_ns}, k, \ell = \mathfrak{r}$

1: **if** $\Psi_{\text{trust}} \neq \emptyset \wedge \neg\text{sh\_ns}$ **then**
▷ Krok 1.1 — korekcja regionów wewnętrznych
2: **if** $|\Psi_{\text{trust}}| > 1$ **then**
3: **for** $\psi_{\text{trust}} \in (\Psi_{\text{trust}} \setminus \textsc{MinReg}(\Psi_{\text{trust}}))$ **do**
4: $\psi_{\text{p}} \leftarrow \textsc{Pred}(\Psi_{\text{trust}}, \psi_{\text{trust}})$
5: $\psi \leftarrow \textsc{UntrustedReg}(\psi_{\text{p}}, \psi_{\text{trust}})$ ▷ $\psi \in \Psi_{\text{untrust}}$, wewnętrzny
6: **if** $|\psi| \geqslant k$ **then**
7: $\textsc{Correct3}'((\textsc{Beg}(\psi), \textsc{End}(\psi) - k + 1))$
8: **end if**
9: **end for**
10: **end if**
11: $\psi_{\min} \leftarrow \textsc{MinReg}(\Psi_{\text{trust}})$
12: $\psi_{\max} \leftarrow \textsc{MaxReg}(\Psi_{\text{trust}})$
▷ Krok 1.2 — korekcja regionu skrajnego początkowego
13: **if** $\textsc{Beg}(\psi_{\min}) > 0$ **then**
14: $\psi \leftarrow (0, \textsc{Beg}(\psi_{\min} - 1))$ ▷ $\psi \in \Psi_{\text{untrust}}$, skrajny początkowy
15: $\textsc{Correct5}'(\psi)$
16: **end if**
▷ Krok 1.3 — korekcja regionu skrajnego końcowego
17: **if** $\textsc{End}(\psi_{\max}) < \ell - k$ **then**
18: $\psi \leftarrow (\textsc{End}(\psi_{\max}) + 1, \ell - k)$ ▷ $\psi \in \Psi_{\text{untrust}}$, skrajny końcowy
19: $\textsc{Correct3}'(\psi)$
20: **end if**
21: **else**
▷ Kroki 2.1, 2.2, 2.3
22: CorrectFirst
23: **end if**

Algorytm z powrotami Strategia działania algorytmu korekcji regionów jest oparta na algorytmie z powrotami, parametryzowanym błędnym regionem. Jego działanie zostanie wyjaśnione dla przypadku korekcji w kierunku $3'$, przedstawionego na pseudokodach 5 oraz 6. Sytuacja dla korekcji w kierunku $5'$ została przedstawiona na pseudokodzie 46 oraz 47, zamieszczonym w dodatku A. Ma ona charakter symetryczny.

Zasada działania korekcji polega na modyfikacji symbolu końcowego danego k-meru błędnego regionu (linia 6) do momentu, aż k-mer ten stanie się k-merem prawidłowym. Po osiągnięciu sukcesu podobna procedura jest wykonywana dla kolejnego k-meru (linia 15) po zastosowaniu w jego sekwencji uzyskanych wcześniej zmian. W razie nieosiągnięcia końca regionu czynność może być powtarzana rekursywnie (linie 36 oraz 46). Należy zauważyć, że w niektórych przypadkach wprowadzenie modyfikacji może nie być konieczne, gdyż oryginalny k-mer może być obecny w spektrum (linia 32). W momencie osiągnięcia ostatniego k-meru regionu następuje zapamiętanie bieżącej ścieżki (m.in. linia 44) oraz sprawdzenie innych modyfikacji lub powrót do wcześniejszych k-merów, dla których wybierane są kolejne wartości modyfikujące.

Korekcja pierwszego k-meru W sytuacji gdy dla odczytu nie wyznaczono żadnego regionu prawidłowego lub istnieje błędny region krótszy niż k, przeprowadzana jest osobna procedura korekcji pierwszego k-meru, przedstawiona na pseudokodach 9 oraz 10. W tym przypadku metodą siłową sprawdzane są wszystkie modyfikacje symboli o wartości współczynnika jakości mniejszej od $\theta_{\text{Q\_LOW}}$ (linia 13) lub sprawdzane niezależnie wszystkie modyfikacje kolejnych symboli k-meru (linie 15 oraz 21), w obu przypadkach będące częścią kroku 2.1. W obu podejściach celem jest uzyskanie prawidłowej sekwencji tego k-meru. Po osiągnięciu sukcesu następuje wykonanie korekcji pozostałej części odczytu w kierunku $3'$ po zastosowaniu zmian w pierwszym k-merze (linia 25) niezależnie dla każdej z uzyskanych wcześniej ścieżek, co stanowi krok 2.2.

Ocena ścieżek korekcji Uzyskane w wyniku korekcji ścieżki podlegają ocenie, która polega na wykonaniu sumowania współczynników jakości zmodyfikowanych pozycji. Jako najlepsza wybierana jest ścieżka o najmniejszej ocenie, przy czym różnica między ewentualnymi dwiema ścieżkami o najniższej ocenie musi wynosić co najmniej $\theta_{\text{MIN}\Delta}$. Dokładny przebieg oceny został przedstawiony na pseudokodzie 12.

Rozszerzanie regionów Każda ścieżka korekcji skrajnego regionu, wygenerowana w wyniku działania algorytmu z powrotami, w której zawarto modyfikacje pozycji w odczycie oddalone od jego końca o mniej niż k symboli (tj. znajdujące się na pozycji $a > \ell - k$) powoduje rozszerzanie regionu (pseudokod 7, od linii 1), które

Pseudokod 5 Korekcja regionu w kierunku $3'$ algorytmu BLESS

Wejście: $k, \ell = |\mathfrak{r}|, \mathfrak{r}$
Wyjście: $\mathfrak{r}$ poddany modyfikacji

```
1:  procedure CORRECT3'(ψ)
2:      Π ← ∅                                          ▷ Zbiór ścieżek korekcji
3:      κ ← κ(𝔯, BEG(ψ))
4:      κ* ← κ                                         ▷ Nowa sekwencja k-meru
5:      for all c ∈ Σ do
6:          κ*[k − 1] ← c
7:          if TRUSTED(κ*) then
8:              Π ← ∅                                  ▷ Zbiór par
9:              if κ* ≠ κ then
10:                 Π ← {(BEG(ψ) + k − 1, c)}
11:             end if
12:             if BEG(ψ) = END(ψ) then                ▷ Koniec regionu
13:                 Π ← Π ∪ {Π}
14:             else
15:                 Π ← CONTINUE3'(κ*, Π, (BEG(ψ + 1), END(ψ)), Π)
16:             end if
17:         end if
18:     end for
19:     if ψ jest regionem wewnętrznym then
20:         Π ← EXTENDINTERNAL(Π, ψ)
21:     else
22:         Π ← EXTEND3'(Π)
23:     end if
24:     Π ← RATE(Π)
25:     if Π ≠ ∅ then
26:         Wstaw do Π element ze zbioru Π
27:         𝔯 ← APPLY(𝔯, Π)                            ▷ Ostateczne przyjęcie zmian
28:     end if
29: end procedure

30: function CONTINUE3'(κ, Π, ψ, Π)
31:     κ* ← κ[1, k]𝔯[BEG(ψ) + k − 1]
32:     if TRUSTED(κ*) then                            ▷ Modyfikacja nie jest potrzebna
33:         if BEG(ψ) = END(ψ) then
34:             Π ← Π ∪ {Π}
35:         else
36:             Π ← CONTINUE3'(κ*, Π, (BEG(ψ) + 1, END(ψ)), Π)
37:         end if
```

Pseudokod 6 Korekcja regionu w kierunku $3'$ algorytmu BLESS, cd.

```
38:     else                                                ▷ Potrzebna modyfikacja
39:         for all c ∈ (Σ \ {κ*[k − 1]}) do
40:             κ*[k − 1] ← c
41:             if Trusted(κ*) then
42:                 Π_n ← Π ∪ {(Beg(ψ) + k − 1, c)}
43:                 if Beg(ψ) = End(ψ) then
44:                     Π ← Π ∪ {Π_n}
45:                 else
46:                     Π ← Continue3′(κ*, Π, (Beg(ψ) + 1, End(ψ)), Π_n)
47:                 end if
48:             end if
49:         end for
50:     end if
51:     return Π
52: end function
```

polega na rekursywnej konkatenacji z odczytem kolejnych symboli (linia 26) do momentu, aż symbole te wraz z końcowymi symbolami odczytu będą stanowiły sekwencję prawidłowego k-meru. Liczba rozszerzających symboli jest ograniczona do $\theta_{\text{MX\_E}}$. Jeżeli uzyskanie takiego k-meru jest niemożliwe, ścieżka zostaje odrzucana. Przeprowadzane jest również rozszerzanie regionów wewnętrznych, które polega na sprawdzeniu kolejnych k-merów między ostatnim zmodyfikowanym symbolem, a początkiem następnego prawidłowego regionu (pseudokod 8). Analogiczne rozszerzanie jest przeprowadzane w przypadku wykonania korekcji pierwszego k-meru (pseudokod 11). Na rys. 5.2 przedstawiono liczbę pozycji rozszerzających odczyt w zależności od pozycji ostatniego zmodyfikowanego symbolu. W przypadku korekcji w kierunku końca $5'$ lub korekcji pierwszego k-meru sytuacja jest symetryczna (pseudokod 11 oraz pseudokod 48 w dodatku A).

Ostatni zmodyfikowany symbol $\theta_{\text{last\_mod}}$	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
Liczba pozycji rozszerzających θ_{ext}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	3	3	3

Rysunek 5.2: Liczba pozycji rozszerzających w zależności od wartości $\theta_{\text{last\_mod}}$; $k = 7, \theta_{\text{MX\_E}} = 3, \ell = 100$

Generacja wyników

Ostatnim etapem algorytmu jest eliminacja błędów wprowadzonych do odczytów na skutek występowania nieprawidłowych odpowiedzi twierdzących filtru Blooma, które jest przeprowadzane metodą opisaną w podrozdziale 4.1.3. Wraz z wykonaniem takiej eliminacji następuje uwzględnienie w plikach wyjściowych modyfikacji, które zostały wyznaczone w poprzednim etapie. Procedura eliminacji błędów filtru Blooma nie jest już obecna w nowych wersjach algorytmu.

Pseudokod 7 Rozszerzanie końcowego regionu algorytmu BLESS

Wejście: $\mathfrak{r}, \ell = |\mathfrak{r}|, k$

```
1: function EXTEND3′(Π)
2:     for all Π ∈ Π do
3:         succ ← false
4:         if Π = ∅ then                          ▷ Istnieje ścieżka bez modyfikacji
5:             return ∅                            ▷ Pominięcie wszystkich ścieżek
6:         else
7:             θ_last_mod ← max({a : (a, c) ∈ Π}) ▷ Ostatni zmodyfikowany symbol
8:             if θ_last_mod ⩽ ℓ − k then
9:                 succ ← true
10:            else
11:                r* ← APPLY(r, Π)
12:                κ* ← κ_DUMMY(r*, ℓ − k + 1)
13:                θ_ext ← min(θ_MX_E, max(k − (ℓ − θ_last_mod), 0))
14:                succ ←CONTEXTEND3′(θ_ext, 1, κ*)
15:            end if
16:            if ¬succ then
17:                Π ← Π \ {Π}
18:            end if
19:        end if
20:    end for
21:    return Π
22: end function

23: function CONTEXTEND3′(θ_ext, a, κ*)
24:    κ* ← κ*∅                                   ▷ Konkatenacja z nowym symbolem ∅
25:    for all c ∈ Σ do
26:        κ*[k − 1] ← c
27:        if TRUSTED(κ*) then
28:            if a = θ_ext then
29:                return true
30:            else if CONTEXTEND3′(θ_ext, a + 1, κ*[1, k − 1]) then
31:                return true
32:            end if
33:        end if
34:    end for
35:    return false
36: end function
```

Pseudokod 8 Rozszerzanie regionów wewnętrznych algorytmu BLESS

Wejście: $\mathfrak{r}, \ell = |\mathfrak{r}|, k$

```
1:  function ExtendInternal(Π, ψ)
2:      for all Π ∈ Π do
3:          if Π = ∅ then                          ▷ Istnieje ścieżka bez modyfikacji
4:              return ∅                            ▷ Pominięcie wszystkich ścieżek
5:          else
6:              θ_last_mod ← max({a : (a, c) ∈ Π})  ▷ Ostatni zmodyfikowany symbol
7:              if θ_last_mod > End(ψ) then
8:                  r* ← Apply(r, Π)
9:                  n_succ ← 0
10:                 for a ← End(ψ) to θ_last_mod do
11:                     κ* ← κ(r*, a)
12:                     if Trusted(κ*) then
13:                         n_succ ← n_succ + 1
14:                     else
15:                         break
16:                     end if
17:                 end for
18:                 if n_succ < θ_last_mod − End(ψ) + 1 then
19:                     Π ← Π \ {Π}
20:                 end if
21:             end if
22:         end if
23:     end for
24: end function
```

5.3.3 Analiza czasowej złożoności obliczeniowej algorytmu BLESS

Głównym składnikiem czasu pracy algorytmu jest korekcja wyznaczonych błędnych regionów, wobec tego zostanie ona uwzględniona w analizie złożoności obliczeniowej. Złożoność zostanie wyznaczona w formie funkcji określających liczbę sprawdzeń zaufania k-merów, polegających na wykonaniu zapytań do filtru Blooma. Następnie, w twierdzeniu 3, zostanie wyznaczona pełna złożoność, w której jako operacja dominująca zostanie przyjęta liczba przypadków odwołań się do symboli k-merów podczas wyznaczania wartości funkcji mieszających w trakcie wykonania zapytań. Wszystkie złożoności obliczeniowe przedstawione poniżej zostały wyznaczone przez autora niniejszej pracy.

Pesymistyczny przypadek w korekcji odczytu ma miejsce w sytuacji, gdy nie wyznaczono żadnych prawidłowych regionów algorytmu albo istnieje błędny region krótszy niż k — w takim przypadku następuje wykonanie korekcji pierwszego k-meru, a jego kontynuacja w postaci algorytmu z powrotami jest przeprowadzana dla największej liczby symboli $\ell - k$.

Pseudokod 9 Korekcja pierwszego k-meru odczytu algorytmu BLESS

Wejście: $k, \ell = |\mathfrak{r}|, \mathfrak{q}, \mathfrak{r}$
Wyjście: $\mathfrak{r}$ poddany modyfikacji

```
1:  procedure CorrectFirst                ▷ Krok 2.1 — korekcja pierwszego k-meru
2:      Π ← ∅
3:      κ ← κ(r, 0)
4:      A_Q_LOW ← ∅                       ▷ Zbiór pozycji o niskich współczynnikach jakości
5:      for all a ∈ {0, 1, ..., |q| − 1} do
6:          if q[a] < θ_Q_LOW then
7:              A_Q_LOW ← Π_Q_LOW ∪ {a}
8:          end if
9:      end for
10:     θ_lq ← |A_Q_LOW|
11:     if 0 < θ_lq ⩽ θ_MX_LQ then
12:         Π ← ∅                                              ▷ Zbiór par
13:         Π ← FirstBruteForce(κ, Π, A_Q_LOW, Π)
14:         if Π = ∅ then
15:             Π ← FirstConsec(κ)
16:         end if
17:     else
18:         if Trusted(κ) then
19:             Π ← Π ∪ {∅}
20:         else
21:             Π ← FirstConsec(κ)
22:         end if
23:     end if
24:     Π ← FirstExtend5′(Π)

                                  ▷ Krok 2.2 — kontynuacja korekcji w stronę końca 3′
25:     Π ← FirstCont(Π)

                ▷ Krok 2.3 — ocena ścieżek korekcji pierwszego i kolejnych k-merów
26:     Π ← Rate(Π)
27:     if Π ≠ ∅ then
28:         Wstaw do Π element ze zbioru Π
29:         r ← Apply(r, Π)                                    ▷ Ostateczne przyjęcie zmian
30:     end if
31: end procedure
```

Pseudokod 10 Korekcja pierwszego k-meru odczytu algorytmu BLESS, cd.

```
32: function FirstCont(Π)
33:     Π_res ← ∅
34:     for all Π ∈ Π do
35:         κ ← κ(τ, 1)
36:         κ* ← Apply(κ, Π)
37:         Π_n ← ∅    ▷ Ścieżki zweryfikowane i uzupełnione podczas kontynuacji
38:         ψ ← (1, ℓ − k)          ▷ Region obejmujący pozostałą część odczytu
39:         Π_n ← Continue3′(κ*, Π_n, ψ, Π)
40:         Π_res ← Π_res ∪ Π_n
41:     end for
42:     Π_res ← FirstExtend3′(Π_res)
43:     return Π_res
44: end function

45: function FirstBruteForce(κ, Π, A_Q_LOW, Π)
46:     a ← min(A_Q_LOW)
47:     for all c ∈ Σ do
48:         κ* ← κ
49:         κ*[a] ← c
50:         Π_n ← Π
51:         if κ* ≠ κ then
52:             Π_n ← Π_n ∪ {(a, c)}
53:         end if
54:         if |A_Q_LOW| = 1 then          ▷ Wykorzystano wszystkie pozycje
55:             if Trusted(κ*) then
56:                 Π ← Π ∪ {Π_n}
57:             end if
58:         else
59:             Π ←FirstBruteForce(κ*, Π, A_Q_LOW \ {a}, Π_n)
60:         end if
61:     end for
62:     return Π
63: end function

64: function FirstConsec(κ)
65:     Π ← ∅
66:     for all a ∈ {0, 1, ..., k − 1} do
67:         κ* ← κ
68:         for all c ∈ Σ \ {κ[a]} do
69:             κ*[a] ← c
70:             if Trusted(κ*) then
71:                 Π ← {(a, c)}
72:                 Π ← Π ∪ {Π}
73:             end if
74:         end for
75:     end for
76:     return Π
77: end function
```

Pseudokod 11 Rozszerzanie początkowego regionu przy korekcji pierwszego k-meru algorytmu BLESS

```
1: function FirstExtend5′(Π)
2:     for all Π ∈ Π do
3:         θ_first_mod ← min({a : (a, c) ∈ Π})   ▷ Pierwszy zmodyfikowany symbol
4:         if Π ≠ ∅ ∧ θ_first_mod < k − 1 then
5:             τ* ← Apply(τ, Π)
6:             κ* ← κ_DUMMY(τ*, −1)
7:             θ_ext ← min(θ_MX_E, k − θ_first_mod − 1, κ*)
8:             if ¬ContExtend5′(θ_ext, 1, κ*) then
9:                 Π ← Π \ {Π}
10:            end if
11:        end if
12:    end for
13:    return Π
14: end function

15: function FirstExtend3′(Π)
16:    for all Π ∈ Π do
17:        θ_last_mod ← max({a : (a, c) ∈ Π})   ▷ Ostatni zmodyfikowany symbol
18:        if θ_last_mod > ℓ − k then
19:            τ* ← Apply(τ, Π)
20:            κ* ← κ_DUMMY(τ*, ℓ − k + 1)
21:            θ_ext ← min(θ_MX_E, k − (ℓ − θ_last_mod, κ*))
22:            if ¬ContExtend3′(θ_ext, 1, κ*) then
23:                Π ← Π \ {Π}
24:            end if
25:        end if
26:    end for
27:    return Π
28: end function
```

Dodatkowe oznaczenia i przyjęte założenia

W poniższych obliczeniach przyjęto dodatkowo następujące oznaczenia: $n_\psi = |\psi|, \psi \in \Psi_{\text{untrust}}$ jest długością bieżącego błędnego regionu. Wykorzystano także następujące oznaczenia zgodne z obecnymi w pseudokodach zmiennymi: θ_{ext} — aktualna liczba pozycji rozszerzających odczyt oraz θ_{lq} — liczba pozycji niskiej jakości pierwszego k-meru.

Ponadto postawiono następujące założenia, upraszczające wyprowadzanie złożoności:

1. $k > \theta_{\text{MX\_E}}$ — skrajny region może zostać rozszerzony co najwyżej o liczbę symboli mniejszą niż długość k-meru (w przypadku modyfikacji pierwszego lub ostatniego symbolu odczytu, zostaje on rozszerzony o $\theta_{\text{MX\_E}}$ symboli),

Pseudokod 12 Ocena ścieżek korekcji algorytmu BLESS

Wejście: $\mathfrak{q}$

```
1:  function RATE(Π)
2:      if |Π| < 1 then
3:          return Π
4:      else
5:          q_min ← +∞
6:          q_min 2 ← +∞
7:          Π_min ← ∅
8:          for all Π ∈ Π do    ▷ Wybór dwóch ścieżek o najmniejszych ocenach
9:              q ← Σ_{(a,c)∈Π} q(a)
10:             if q < q_min then
11:                 q_min 2 ← q_min
12:                 q_min ← q
13:                 Π_min ← Π
14:             end if
15:         end for
                                ▷ Przypuszczalnie poniższy warunek powinien
    mieć postać q_min ⩽ q_min 2 + θ_MINΔ, a jego obecna postać jest wynikiem błędu
    implementacji.
16:         if q_min ⩾ q_min 2 + θ_MINΔ then
17:             Π ← {Π_min}
18:         else
19:             Π ← ∅
20:         end if
21:         return Π
22:     end if
23: end function
```

2. $\theta_{\text{MX\_LQ}} \leqslant k - \theta_{\text{MX\_E}}$ — maksymalna liczba pozycji niskiej jakości obecnych w pierwszym k-merze wykorzystanych w korekcji metodą siłową jest mniejsza niż długość k-meru pomniejszona o maksymalną liczbę pozycji rozszerzających skrajny region,

3. $k \geqslant 10$,

4. $\ell - k \geqslant 10$,

5. istnieje możliwość, że w odczycie nie wyznaczono żadnego prawidłowego regionu (tj. jeden błędny region obejmuje cały odczyt), a jednocześnie każda k-elementowa wariacja z powtórzeniami ze zbioru Σ (tj. każdy możliwy k-mer) może wystąpić w spektrum; założenie to ma na celu znaczne uproszczenie wyznaczania liczby zapytań do spektrum oraz określania liczby wygenerowanych ścieżek,

6. złożoność obliczeniowa wyznaczenia wartości każdej funkcji mieszającej filtru Blooma wynosi $T_{\text{hash}}(k) = k$.

Wyznaczenie złożoności obliczeniowej korekcji w kierunku 3′

Lemat 1. *Pesymistyczna złożoność obliczeniowa korekcji regionu w kierunku* 3′ *algorytmu BLESS wynosi:*

$$T_{\text{B3'}}(n_\psi) = 8 \cdot 3^{n_\psi - 1} - 4 = O\left(2^{n_\psi}\right). \tag{5.1}$$

Dowód. Rozpatrywany przypadek obejmują pseudokody 5 oraz 6. W pierwszej kolejności zostanie przeanalizowane działanie funkcji `Continue3′`. Pesymistyczny przypadek obejmuje sytuację, gdy bieżący k-mer nie jest zaufany i w rezultacie następuje próba modyfikacji symbolu.

Przebieg funkcji jest następujący: w linii 32 zostaje wykonane jednokrotne zapytanie do bazy. W przypadku uzyskania negatywnego wyniku w linii 41 zapytanie zostanie wykonane trzykrotnie, podobnie przejście rekursywne dla argumentu $n_\psi - 1$ (linia 46). Tym samym złożoność obliczeniowa funkcji `Continue3′` wynosi $T'_{\text{B3'}}(n_\psi) = 3T'_{\text{B3'}}(n_\psi - 1) + 4$, gdzie gdzie $T'_{\text{B3'}}(0) = 0$, a po eliminacji rekurencji wyrażenie ulega uproszczeniu do $T'_{\text{B3'}}(n_\psi) = 2 \cdot 3^{n_\psi} - 2$.

Uwzględniając dodatkowo 4 zapytania do bazy wykonane w funkcji `Correct3′` (linia 7) oraz wywołania funkcji `Continue3′` (linia 15) ostateczna złożoność obliczeniowa przyjmuje postać $T_{\text{B3'}}(n_\psi) = 8 \cdot 3^{n_\psi - 1} - 4 = O(4^{n_\psi}) = O(2^{n_\psi})$. □

Lemat 2. *Pesymistyczna złożoność obliczeniowa rozszerzania końcowego regionu dla ścieżki korekcji w kierunku* 3′ *algorytmu BLESS wynosi:*

$$T_{\text{Bre}}(\theta_{\text{ext}}) = \frac{1}{3}\left(4^{\theta_{\text{ext}}+1} - 4\right) = O\left(2^{\theta_{\text{ext}}}\right). \tag{5.2}$$

Dowód. Rozpatrywany przypadek obejmuje funkcja `ContExtend3′` przedstawiona na pseudokodzie 7. Dla każdej pozycji rozszerzającej (tj. pozycji symbolu dołączonego do odczytu) następuje sprawdzenie co najwyżej $|\Sigma| = 4$ symboli (gdy dopiero dla ostatniego symbolu zostanie uzyskany ciąg prawidłowych k-merów), a następnie rekursywne wykonanie algorytmu dla kolejnych pozycji rozszerzających (linia 14). Stąd złożoność obliczeniowa wynosi $T_{\text{Bre}}(\theta_{\text{ext}}) = 4 + 4T_{\text{Bre}}(\theta_{\text{ext}} - 1)$, gdzie $T_{\text{Bre}}(0) = 0$, a po eliminacji rekurencji wyrażenie ulega uproszczeniu do postaci przedstawionej w tezie lematu. □

Lemat 3. *Maksymalna liczba ścieżek wygenerowanych w wyniku korekcji regionu w kierunku* 3′ *algorytmu BLESS wynosi:*

$$\pi_{\text{B3'}}(n_\psi) = 4 \cdot 3^{n_\psi - 1}. \tag{5.3}$$

Dowód. W pierwszej kolejności zostanie przeanalizowane działanie funkcji `Continue3′` (pseudokody 5 oraz 6, linia 30). W każdym jej wywołaniu może nastąpić pozytywne zweryfikowanie $|\Sigma|-1 = 3$ symboli, a dla każdego z nich rekursywne wywołanie funkcji. W rezultacie gdy funkcja jest wywoływana rekursywnie do $n_\psi - 1$ poziomów, liczba uzyskanych możliwości jest równa liczbie $(n_\psi - 1)$-elementowych wariacji z powtórzeniami ze zbioru 3-elementowego. Dodatkowo, uwzględniając fakt, że możliwe są 4 kolejne zmiany w funkcji `Correct3′` (linia 1), tj. dopuszczalne jest zachowanie oryginalnego symbolu, liczba ścieżek jest równa zależności przedstawionej w tezie lematu. □

Twierdzenie 1. *Pesymistyczna złożoność obliczeniowa korekcji w kierunku* $3'$ *algorytmu BLESS wraz z rozszerzaniem wynosi:*

$$T_{\text{B3're}}(n_\psi, \theta_{\text{MX\_E}}) = 3^{n_\psi - 2}\left(4^{\theta_{\text{MX\_E}}+2} + 8\right) - 4 = \\ = O\left(2^{n_\psi + \theta_{\text{MX\_E}}}\right). \tag{5.4}$$

Dowód. W przypadku pesymistycznym spełnione są dwa warunki: (*i*) korekcja w kierunku $3'$ odbywa się zgodnie z sytuacją jak w lemacie 1 oraz (*ii*) zostaje wygenerowana maksymalna możliwa liczba ścieżek zgodnie z sytuacją jak w lemacie 3. Ostatnia zmodyfikowana pozycja każdej ścieżki korekcji znajduje się na ostatniej pozycji regionu, ponieważ pesymistyczny przypadek złożoności funkcji `Continue3′` ma miejsce, gdy dla każdej pozycji wprowadzono pewne zmiany (zatem również dla ostatniej; pseudokod 6, linia 38). Stąd w każdym przypadku odczyt zostaje rozszerzony o $\theta_{\text{MX\_E}}$ pozycji, tzn. wartość $T_{\text{Bre}}(\cdot)$ zostanie wyznaczona dla $\theta_{\text{ext}} = \theta_{\text{MX\_E}}$. Tym samym złożoność obliczeniowa wynosi: $T_{\text{B3're}}(n_\psi, \theta_{\text{MX\_E}}) = T_{\text{B3'}}(n_\psi) + \pi_{\text{B3'}}(n_\psi) T_{\text{Bre}}(\theta_{\text{MX\_E}})$, co po podstawieniu wyrażeń oraz uproszczeniu przyjmuje postać jak w tezie twierdzenia. □

Wyznaczenie złożoności obliczeniowej korekcji pierwszego k-meru

Lemat 4. *Pesymistyczna złożoność obliczeniowa rozszerzania początkowego regionu korekcji w kierunku* $5'$ *algorytmu BLESS wynosi:*

$$T_{\text{Ble}}(\theta_{\text{ext}}) = \frac{1}{3}\left(4^{\theta_{\text{ext}}+1} - 4\right) = O\left(2^{\theta_{\text{ext}}}\right). \tag{5.5}$$

Dowód. Rozpatrywany przypadek obejmuje pseudokod 48. Przez symetryczność sytuacji dowód jest analogiczny jak w przypadku lematu 2. □

Lemat 5. *Pesymistyczna złożoność obliczeniowa siłowej metody korekcji pierwszego k-meru algorytmu BLESS wynosi:*

$$T_{\text{B1b}}(\theta_{\text{lq}}) = 4^{\theta_{\text{lq}}} = O\left(2^{\theta_{\text{lq}}}\right). \tag{5.6}$$

Dowód. Rozpatrywany przypadek obejmuje pseudokody 9 oraz 10, w szczególności funkcję `FirstBruteForce`. Liczba sprawdzanych pozycji niskiej jakości wynosi θ_{lq} (wg warunku w linii 11). Każda z tych wartości może przyjąć $|\Sigma| = 4$ wartości, wobec tego liczba wykonanych zapytań do bazy (linia 55) jest równa liczbie θ_{lq}-elementowych wariacji z powtórzeniami ze zbioru 4-elementowego, a złożoność obliczeniowa przyjmuje postać przedstawioną w tezie lematu. □

Lemat 6. *Pesymistyczna złożoność obliczeniowa korekcji pierwszego k-meru metodą modyfikacji kolejnych symboli algorytmu BLESS wynosi:*

$$T_{\mathrm{B1c}}(k) = 3k = O(k). \tag{5.7}$$

Dowód. Rozpatrywany przypadek obejmuje funkcję `FirstConsec` przedstawioną na pseudokodzie 10. Dla każdej spośród k pozycji k-meru (linia 66) jest wykonywana modyfikacja poprzez niezależne podstawianie 3 różnych symboli (tj. każdego symbolu zbioru Σ, wyłączając oryginalny symbol), a dla każdej modyfikacji wykonywane jest jedno sprawdzenie obecności k-meru w spektrum, stąd złożoność obliczeniowa przyjmuje postać przedstawioną w tezie lematu. □

Lemat 7. *Maksymalna liczba ścieżek uzyskanych w wyniku korekcji pierwszego k-meru metodą siłową algorytmu BLESS wynosi:*

$$\pi_{\mathrm{B1b}}(\theta_{lq}) = 4^{\theta_{lq}}. \tag{5.8}$$

Dowód. Sprawdzenie obecności w bazie k-merów efektu każdego zbioru modyfikacji może wiązać się z utworzeniem ścieżki. Tym samym liczba wygenerowanych ścieżek jest równa wartości złożoności wyznaczonej w lemacie 5. □

Lemat 8. *Maksymalna liczba ścieżek uzyskanych w wyniku korekcji pierwszego k-meru metodą modyfikacji kolejnych symboli algorytmu BLESS wynosi:*

$$\pi_{\mathrm{B1c}}(k) = 3k. \tag{5.9}$$

Dowód. Sprawdzenie obecności w bazie k-merów efektu każdej modyfikacji może wiązać się z utworzeniem ścieżki. Tym samym liczba wygenerowanych ścieżek jest równa wartości złożoności wyznaczonej w lemacie 6. □

Lemat 9. *W algorytmie BLESS dana pozycja $a \in \{0, 1, \ldots, k-1\}$ pełni rolę pierwszego zmodyfikowanego symbolu w maksymalnie:*

$$\pi_{\mathrm{B1c}}(a, k) = 3 \tag{5.10}$$

ścieżkach w korekcji pierwszego k-meru metodą modyfikacji kolejnych symboli.

Dowód. Każda ścieżka korekcji obejmuje modyfikację jednej pozycji. Symbol tej pozycji może przyjąć jedną z wartości ze zbioru Σ, wyłączając oryginalny symbol. Zatem modyfikacja każdej pozycji może zostać przeprowadzona na 3 sposoby. □

Lemat 10. *Pesymistyczna złożoność obliczeniowa rozszerzania początkowego regionu dla wszystkich ścieżek uzyskanych w metodzie siłowej korekcji pierwszego k-meru odczytu algorytmu BLESS wynosi:*

$$T_{\mathrm{B1ble}}(\theta_{\mathrm{MX\_E}}, \theta_{\mathrm{MX\_LQ}}) = \frac{1}{3}4^{\theta_{\mathrm{MX\_LQ}}+1}\left(4^{\theta_{\mathrm{MX\_E}}} - 1\right) = \\ = O\left(2^{\theta_{\mathrm{MX\_LQ}}+\theta_{\mathrm{MX\_E}}}\right). \tag{5.11}$$

Dowód. Rozpatrywany przypadek obejmuje pseudokod 9, gdy wykonano kod w linii 13 i w rezultacie uzyskano maksymalną możliwą liczbę ścieżek. Pesymistyczna złożoność obliczeniowa dotyczy sytuacji, gdy dla każdej ścieżki odczyt jest rozszerzany o maksymalną liczbę pozycji (linia 24). Wszystkie modyfikacje znajdują się wówczas wśród $\theta_{\mathrm{MX\_LQ}}$ pierwszych pozycji; ze względu na założenie 2, dla każdej z nich odczyt jest rozszerzany o maksymalną liczbę symboli, tj. $\theta_{\mathrm{MX\_E}}$, symetrycznie do sytuacji przedstawionej na rys. 5.2.

Złożoność może zostać wyznaczona w oparciu o lematy 4 oraz 7, przy czym równania obecne w tezach tych lematów są parametryzowane wartościami $\theta_{\mathrm{ext}} = \theta_{\mathrm{MX\_E}}$ oraz $\theta_{\mathrm{lq}} = \theta_{\mathrm{MX\_LQ}}$, tj. $T_{\mathrm{B1ble}}(\theta_{\mathrm{MX\_E}}, \theta_{\mathrm{MX\_LQ}}) = \pi_{\mathrm{B1b}}(\theta_{\mathrm{MX\_LQ}})T_{\mathrm{Ble}}(\theta_{\mathrm{MX\_E}})$, co po podstawieniu wyrażeń oraz uproszczeniu przyjmuje postać jak w tezie lematu. □

Lemat 11. *Pesymistyczna złożoność obliczeniowa rozszerzania początkowego regionu dla wszystkich ścieżek przy korekcji metodą modyfikacji kolejnych symboli pierwszego k-meru algorytmu BLESS wynosi:*

$$T_{\mathrm{B1cle}}(k, \theta_{\mathrm{MX\_E}}) = \\ = 4^{\theta_{\mathrm{MX\_E}}+1}\left(\frac{1}{3} + k + \theta_{\mathrm{MX\_E}}\right) - 4k - \frac{4}{3} = \\ = O\left((k + \theta_{\mathrm{MX\_E}})2^{\theta_{\mathrm{MX\_E}}}\right). \tag{5.12}$$

Dowód. Rozpatrywany przypadek obejmuje pseudokod 9, gdy wykonano kod w linii 15 albo w linii 21, a w rezultacie uzyskano maksymalną możliwą liczbę ścieżek. Pesymistyczna złożoność obliczeniowa dotyczy sytuacji, w której dla każdej ścieżki region jest rozszerzany o maksymalną liczbę pozycji (linia 24). Ze względu na założenie 1 liczba pozycji rozszerzających odczyt dla danej ścieżki może być różna, zgodnie z sytuacją symetryczną do przedstawionej na rys. 5.2. Tym samym liczba pozycji rozszerzających dla różnych ścieżek jest wyznaczana opierając się na lematach 4 oraz 9:

$$T_{\mathrm{B1cle}}(k, \theta_{\mathrm{MX\_E}}) = \\ = \sum_{i=1}^{\theta_{\mathrm{MX\_E}}-1} \pi_{\mathrm{B1c}}(k-i-1, k)T_{\mathrm{Ble}}(i) + \sum_{i=1}^{k-\theta_{\mathrm{MX\_E}}} \pi_{\mathrm{B1c}}(i-1, k)T_{\mathrm{Ble}}(\theta_{\mathrm{MX\_E}}),$$

co po podstawieniu wyrażeń oraz uproszczeniu przyjmuje postać jak w tezie lematu. □

Podczas korekcji pierwszego k-meru można wyróżnić trzy przypadki:

(*i*) liczba pozycji niskiej jakości $\theta_{\text{lq}} \notin (0; \theta_{\text{MX\_LQ}}]$ i oryginalny pierwszy k-mer nie jest zaufany; w efekcie zostaje wykonana tylko korekcja metodą modyfikacji kolejnych symboli (pseudokod 9, linia 21);

(*ii*) liczba pozycji niskiej jakości należy do tego przedziału; w efekcie zostaje przeprowadzona korekcja metodą siłową, w wyniku której zostaje uzyskana co najmniej jedna ścieżka korekcji (linia 13);

(*iii*) liczba pozycji niskiej jakości należy do tego przedziału, jednak korekcja metodą siłową nie zwraca żadnej ścieżki; w efekcie zostaje wykonana korekcja przez modyfikację kolejnych symboli, tzn. zostanie przeprowadzona korekcja obydwoma metodami (linie 13 oraz 15).

Wybór przypadku, dla której uzyskana złożoność korekcji pierwszego k-meru byłaby największa, jest bezcelowy, ponieważ złożoność całego odczytu zależy także od liczby ścieżek uzyskanych w wyniku korekcji pierwszego k-meru, z których każda stanowi podstawę do przeprowadzenia korekcji dalszej części odczytu. Tym samym w dalszej kolejności zostanie wyznaczona złożoność obliczeniowa korekcji całego odczytu w zależności od przypadku korekcji pierwszego k-meru.

Maksymalna liczba wygenerowanych ścieżek w przypadkach (*i*) oraz (*iii*) jest taka sama. Maksymalna złożoność obliczeniowa przypadku (*iii*) jest większa niż przypadku (*i*), pomimo wykonania w tym ostatnim jednego dodatkowego zapytania do bazy k-merów (linia 18). Tym samym znalezienie pesymistycznego przypadku uwzględniającego wszystkie przypadki wymaga porównania przypadku (*ii*) oraz (*iii*) i w kolejnym podrozdziale zostaną wyznaczone złożoności obliczeniowe dotyczące tylko tych dwóch przypadków. Przypadek (*ii*) zostanie objęty lematem 12, a przypadek (*iii*) — lematem 13.

Wyznaczenie złożoności obliczeniowej korekcji całego odczytu

Lemat 12. *Pesymistyczna złożoność obliczeniowa korekcji całego odczytu, w przypadku gdy jest wykonywana korekcja pierwszego k-meru metodą siłową, algorytmu BLESS wynosi:*

$$\begin{aligned} T_{\text{Bb}}(k, \ell, \theta_{\text{MX\_E}}, \theta_{\text{MX\_LQ}}) = \\ = \frac{1}{3} 4^{\theta_{\text{MX\_LQ}}} \left[4^{\theta_{\text{MX\_E}}+1} + 3^{\ell-k-1} \left(4^{\theta_{\text{MX\_E}}+2} + 8 \right) - 1 \right] - 4 = \\ = O\left(2^{\theta_{\text{MX\_LQ}}+\theta_{\text{MX\_E}}+\ell-k} \right). \end{aligned} \tag{5.13}$$

Dowód. Rozpatrywany przypadek obejmuje pseudokod 9. Składnikami złożoności obliczeniowej są: złożoność korekcji pierwszego k-meru metodą siłową (linia 13), złożoność rozszerzania regionu początkowego w lewo (linia 24) oraz korekcja pozostałej części odczytu (tj. regionu długości $n_\psi = \ell - k$ — linia 25),

wykonana dla wszystkich uzyskanych w wyniku korekcji pierwszego k-meru ścieżek. Opierając się na lematach 5, 7 oraz 10, a także twierdzeniu 1 sumarycznie wynosi ona: $T_{\mathrm{Bb}}(k,\ell,\theta_{\mathrm{MX\_E}},\theta_{\mathrm{MX\_LQ}}) = T_{\mathrm{B1b}}(\theta_{\mathrm{MX\_LQ}}) + T_{\mathrm{B1ble}}(\theta_{\mathrm{MX\_E}},\theta_{\mathrm{MX\_LQ}}) + \pi_{\mathrm{B1b}}(\theta_{\mathrm{MX\_LQ}})T_{\mathrm{B3're}}(\ell-k,\theta_{\mathrm{MX\_E}})$, co po podstawieniu wyrażeń oraz uproszczeniu przyjmuje postać jak w tezie lematu. □

Lemat 13. *Pesymistyczna złożoność obliczeniowa korekcji całego odczytu, w przypadku gdy jest wykonywana korekcja pierwszego k-meru metodą siłową wraz z metodą modyfikacji kolejnych symboli algorytmu BLESS, wynosi:*

$$\begin{aligned} T_{\mathrm{Bc}}(k,\ell,\theta_{\mathrm{MX\_E}},\theta_{\mathrm{MX\_LQ}}) = \\ = 4^{\theta_{\mathrm{MX\_LQ}}} + 4^{\theta_{\mathrm{MX\_E}}+1}\left(\frac{1}{3}+k+\theta_{\mathrm{MX\_E}}\right) + \\ +k\cdot 3^{\ell-k-1}\left(4^{\theta_{\mathrm{MX\_E}}+2}+8\right) - 13k - \frac{4}{3} = \\ = O\left(2^{\theta_{\mathrm{MX\_LQ}}} + (k+\theta_{\mathrm{MX\_E}})2^{\theta_{\mathrm{MX\_E}}} + k2^{\ell-k+\theta_{\mathrm{MX\_E}}}\right). \end{aligned} \tag{5.14}$$

Dowód. Rozpatrywany przypadek obejmuje pseudokod 9. Składnikami złożoności obliczeniowej są: złożoność korekcji pierwszego k-meru wykonana najpierw metodą siłową (w wyniku której nie uzyskuje się ścieżek — linia 13), a później metodą modyfikacji kolejnych symboli (linia 21), złożoność rozszerzania regionu początkowego w lewo (linia 24) oraz korekcja pozostałej części odczytu (tj. regionu długości $n_\psi = \ell - k$ — linia 25), wykonana dla wszystkich uzyskanych w wyniku korekcji pierwszego k-meru ścieżek. Opierając się na lematach 5, 6, 8 oraz 11, a także twierdzeniu 1 sumarycznie wynosi ona: $T_{\mathrm{Bc}}(k,\ell,\theta_{\mathrm{MX\_E}},\theta_{\mathrm{MX\_LQ}}) = T_{\mathrm{B1b}}(\theta_{\mathrm{MX\_LQ}}) + T_{\mathrm{B1c}}(k) + T_{\mathrm{B1cle}}(k,\theta_{\mathrm{MX\_E}}) + \pi_{\mathrm{B1c}}(k)T_{\mathrm{B3're}}(\ell-k,\theta_{\mathrm{MX\_E}})$, co po podstawieniu wyrażeń oraz uproszczeniu przyjmuje postać jak w tezie lematu. □

Określenie pesymistycznej złożoności obliczeniowej wymaga porównania złożoności stanowiących przedmiot lematów 12 oraz 13. Złożoności te są jednak zależne od czterech parametrów: k, ℓ, $\theta_{\mathrm{MX\_E}}$, $\theta_{\mathrm{MX\_LQ}}$, wobec tego rozwiązanie takiej nierówności jest zadaniem trudnym. W celu uproszczenia rozwiązania parametry $\theta_{\mathrm{MX\_E}}$ oraz $\theta_{\mathrm{MX\_LQ}}$ zostaną zastąpione wartościami, które stanowią stałe parametry implementacji algorytmu, zgodnie z tabelami 5.1 oraz 5.2.

Lemat 14. *Pesymistyczna złożoność obliczeniowa korekcji algorytmem BLESS całego odczytu jest maksymalna podczas korekcji pierwszego k-meru metodą siłową dla $k \leqslant 21$ oraz dla korekcji metodą siłową wraz z metodą modyfikacji kolejnych symboli dla $k > 21$.*

Dowód. W celu udowodnienia części tezy dotyczącej maksymalnej złożoności dla $k \leqslant 21$, należy rozwiązać nierówność obejmującą korekcję całego odczytu w obu przypadkach, których złożoności wyznaczono w dowodach lematów 12 oraz 13:

$$T_{\mathrm{Bb}}(k,\ell,\theta_{\mathrm{MX\_E}},\theta_{\mathrm{MX\_LQ}}) > T_{\mathrm{Bc}}(k,\ell,\theta_{\mathrm{MX\_E}},\theta_{\mathrm{MX\_LQ}}).$$

Lewa strona stanowi złożoność obliczeniową korekcji odczytu w przypadku (*ii*), a prawa w przypadku (*iii*).

Po podstawieniu wartości, przekształceniu i uproszczeniu nierówność przyjmuje postać:

$$4090{,}75 + 3^{\ell-k-2}(65552 - 3073{,}5k) > 255{,}1875k.$$

Na podstawie założenia 4 współczynnik $3^{\ell-k-2}$ przyjmuje wartość co najmniej 6561. Tym samym dla $k = 21$ nierówność jest prawdziwa. Ponadto dla $k < 21$ lewa strona równania przyjmuje wartości większe niż dla $k = 21$, a prawa strona — mniejsze. W związku z tym dla $k \leqslant 21$ lemat jest prawdziwy.

W celu udowodnienia części tezy dotyczącej maksymalnej złożoności dla $k > 21$, zostanie rozwiązana analogiczna nierówność, jednak z odwrotnym znakiem:

$$4090{,}75 + 3^{\ell-k-2}(65552 - 3073{,}5k) < 255{,}1875k.$$

Dla $k = 22$ lewa strona nierówności przyjmuje wartość ujemną. Ponadto dla $k > 22$ lewa strona przyjmuje wartości mniejsze niż dla $k = 22$, a prawa — mniejsze. W związku z tym dla $k > 21$ lemat jest prawdziwy. □

Należy zaznaczyć, że w lemacie 14 nie uwzględniono następującej kwestii: jeżeli w korekcji opartej na modyfikacji kolejnych symboli nie znaleziono żadnej ścieżki, wówczas liczba ścieżek uzyskanych w efekcie ewentualnej pracy algorytmu siłowego będzie mniejsza niż wynikałoby to z lematu 9, gdyż pewna grupa sprawdzanych ścieżek jest wspólna dla metody siłowej oraz metody modyfikacji kolejnych symboli. Mając jednak na celu ograniczenie liczby rozpatrywanych przypadków, w analizie pomięto tę obserwację.

Twierdzenie 2. Pesymistyczna złożoność obliczeniowa korekcji odczytu algorytmu BLESS wynosi:

$$T'_{\mathrm{B}}(k, \ell, \theta_{\mathrm{MX\_E}}, \theta_{\mathrm{MX\_LQ}}) = \begin{cases} T_{\mathrm{Bb}}(k, \ell, \theta_{\mathrm{MX\_E}}, \theta_{\mathrm{MX\_LQ}}) = O\left(2^{\ell}\right), \text{ gdy } k \leqslant 21, \\ T_{\mathrm{Bc}}(k, \ell, \theta_{\mathrm{MX\_E}}, \theta_{\mathrm{MX\_LQ}}) = O\left(k + k2^{\ell}\right), \text{ gdy } k > 21, \end{cases} \tag{5.15}$$

gdzie wartości $T_{\mathrm{Bb}}(k, \ell, \theta_{\mathrm{MX\_E}}, \theta_{\mathrm{MX\_LQ}})$ oraz $T_{\mathrm{Bc}}(k, \ell, \theta_{\mathrm{MX\_E}}, \theta_{\mathrm{MX\_LQ}})$ są zdefiniowane w równaniach (5.13) oraz (5.14).

Dowód. Jak zostało wspomniane, w przypadku pesymistycznym w wyniku wyznaczania regionów nie zostają wyznaczone żadne prawidłowe regiony lub istnieją błędne regiony krótsze niż k. Złożoność obliczeniowa korekcji całego odczytu została oparta na obserwacjach wyszczególnionych w dowodzie lematu 14.

Ze względu na postawione w tym lemacie założenie o stałości wartości $\theta_{\mathrm{MX\_E}}$ oraz $\theta_{\mathrm{MX\_LQ}}$ oraz ze względu na założenie 4, rzędy złożoności występujące w niniejszym twierdzeniu ulegają uproszczeniu w stosunku do przedstawionych w lematach 12 oraz 13. □

Twierdzenie 3. *Pesymistyczna złożoność obliczeniowa korekcji odczytu algorytmu BLESS z uwzględnieniem wykonania zapytania do filtru Blooma, przy przyjęciu jako operacji dominującej wyznaczenia wartości funkcji mieszającej, wynosi:*

$$T_{\mathrm{B}}(k,\ell,\theta_{\mathrm{MX\_E}},\theta_{\mathrm{MX\_LQ}}) = \begin{cases} dkT_{\mathrm{Bb}}(k,\ell,\theta_{\mathrm{MX\_E}},\theta_{\mathrm{MX\_LQ}}) = O\left(k2^{\ell}\right), \text{ gdy } k \leqslant 21, \\ dkT_{\mathrm{Bc}}(k,\ell,\theta_{\mathrm{MX\_E}},\theta_{\mathrm{MX\_LQ}}) = O\left(k^2 + k^2 2^{\ell}\right), \text{ gdy } k > 21, \end{cases} \tag{5.16}$$

gdzie d jest liczbą funkcji mieszających, $d = O(1)$.

Dowód. Każde zapytanie filtru Blooma wymaga wyznaczenia wartości d funkcji mieszających, gdzie każda z nich, na mocy założenia 6, charakteryzuje się złożonością rzędu $O(k)$. □

5.3.4 Podsumowanie

W wyniku analizy algorytmu stwierdzono, że posiada on następujące niedoskonałości:

- występowanie wad filtrów Blooma (błędne odpowiedzi twierdzące oraz mała ilość informacji o spektrum),
- brak skalowalności ze względu na liczbę rdzeni procesora i niska szybkość działania (w wersji 0.12),
- oparcie zasady działania algorytmu na niemal pełnym zaufaniu do danych zawartych w spektrum (przyjmowanie, że wszystkie k-mery wyjściowe powinny być zaufane, co ze względu na błędy modelowania przy pomocy spektrum nie jest pewne),
- brak wyraźnej strategii obchodzenia trudności wynikających z obecności obszarów o niskim pokryciu,
- wysoka pesymistyczna czasowa złożoność obliczeniowa wynosząca $O(k^2 + k^2 2^{\ell})$, tzn. wyższa niż wykładnicza w funkcji długości odczytu.

Należy zauważyć, że zaproponowana w algorytmie metoda eliminacji błędów filtru Blooma pozwala na uwzględnienie tylko części błędów spowodowanych fałszywymi odpowiedziami filtru, tj. błędów powstających w sytuacji, gdy w pliku wynikowym następuje umieszczenie k-meru nieobecnego w odczytach wejściowych. Może to spowodować wycofanie nieprawidłowej korekcji odczytu, który powinien być jednak skorygowany przy pomocy innej ścieżki korekcji, jednak prawidłowa korekcja takiego odczytu nie zostaje wprowadzona. Metoda ta nie wyklucza jednak spodziewanego obniżenia czułości wykrywania błędnych k-merów w odczytach wejściowych, gdyż błędny k-mer w odczycie może zostać uznany za prawidłowy i w rezultacie niepoddany korekcji.

5.4 Algorytm RECKONER

Możliwości algorytmu KMC dają potencjał do zastosowania go jako pomocniczego narzędzia w wielu innych algorytmach, także mających za zadanie wykonywanie korekcji odczytów sekwencjonowania. Szczególnie użyteczne i nowatorskie może okazać się wykorzystanie w nowym algorytmie informacji o liczebności k-merów obecnych w odczytach sekwencjonowania, a nie tylko binarnej informacji o obecności danego k-meru w spektrum (zaklasyfikowaniu go do zbioru k-merów zaufanych albo niezaufanych). Zgodnie z wiedzą autora, takie rozwiązanie jak dotąd nie zostało wdrożone w żadnym algorytmie korekcji.

Przyjęto, że w nowym algorytmie zostanie wykorzystana strategia zbliżona do algorytmów opartych na spektrum k-merów. Motywowane jest to łatwym do osiągnięcia rozsądnym zapotrzebowaniem na zasoby obliczeniowe (w stosunku do algorytmów opartych na strukturach sufiksowych — mniejszym zapotrzebowaniem na pamięć, a w stosunku do algorytmów opartych na dopasowywaniu wielu sekwencji — krótszym czasem obliczeń). Zwrócono jednak uwagę na możliwość uwzględnienia jednej z cech algorytmów opartych na strukturach sufiksowych, którą jest wykorzystanie oligomerów wielu (w przypadku nowego algorytmu dwóch) długości. Ponadto zrezygnowano z wykorzystania głównych idei algorytmów należących do grupy wykorzystujących inne techniki (jak PREMIER czy SAMDUDE) przyjmując, że złożone zagadnienie korekcji odczytów może być zbyt skomplikowane, aby precyzyjnie i w sposób praktycznie użyteczny zdefiniować je przy pomocy formalnego aparatu matematycznego, zastosowanego w koncepcji tych algorytmów.

W celu ominięcia czasochłonnej implementacji rdzenia algorytmu, podobnego dla większości algorytmów z grupy, zdecydowano się na wykorzystanie kodu istniejącego algorytmu BLESS w wersji 0.12. Jak zostało wspomniane, jest to wczesna wersja algorytmu, w której zliczanie k-merów odbywa się przy pomocy metody opisanej w podrozdziale 4.1.3 i bez obsługi wielowątkowości. W kolejnych wersjach algorytm ten został uzupełniony o liczne nowe elementy, z których tylko trzy niezależnie zostały wprowadzone w prezentowanym nowym algorytmie: zastosowanie algorytmu KMC jako narzędzia zliczania k-merów (jednakże należy zauważyć, że algorytm BLESS wykorzystuje KMC wyłącznie do zliczania k-merów i wygenerowania ich zbioru, które są później reprezentowane przy pomocy filtru Blooma, a nie bazy k-merów i KMC API), zrównoleglenie algorytmu (jednakże przy pomocy zdecydowanie innej strategii realizacji współbieżności oraz innej techniki — algorytm BLESS wykorzystuje biblioteki OpenMP oraz MPI w celu implementacji współbieżności zgodnie z modelem sieciowym), oraz obsługa plików skompresowanych (jednakże obarczona technicznymi mankamentami, co zostanie wyszczególnione w podrozdziale 6.5). Ponadto w nowych wersjach algorytmu BLESS zaproponowano metodę określania progu obcięcia, która po niewielkiej modyfikacji została zaadaptowana we wprowadzanym algo-

rytmie. Z kolei obecna w obu wersjach algorytmu BLESS metoda określania skali przyporządkowania oparta na obserwacji minimalnej i maksymalnej wartości współczynnika jakości została całkowicie zamieniona autorskim rozwiązaniem.

W związku z wykorzystaniem schematu BLESS, nowy algorytm zostanie siłą rzeczy zaklasyfikowany do grupy algorytmów z powrotami. Jednocześnie, z racji oparcia się na modelu wykorzystującym k-mery (niosących niepełną informację o budowie genomu) oraz wykorzystania intuicyjnych przesłanek w metodzie generowania i oceny ścieżek korekcji, będzie posiadał cechy algorytmu heurystycznego.

Ze względu na główną koncepcję, jako nazwę nowego algorytmu zaproponowano miano RECKONER, będącą skróconą formą od angielskiego opisu *Read Error Corrector Based on KMC* — Korektor błędów odczytów oparty na KMC. W poniższych podrozdziałach zostanie opisane działanie oraz charakterystyczne cechy algorytmu RECKONER, wyróżniające go na tle pozostałych algorytmów.

5.4.1 Przyjęty model błędów

W przygotowaniu oraz analizie algorytmu zostaną postawione następujące założenia dotyczące błędów odczytów. Każdy odczyt $\mathfrak{r}$ długości ℓ jest przekształconą formą pewnej podsekwencji genomu $\mathfrak{r}'$ długości ℓ', uzyskanej z pozycji tego genomu $\mathfrak{a}$, tj. $\mathfrak{r}' = \mathfrak{c}_i[\mathfrak{a}, \mathfrak{a} + \ell' - 1]$, gdzie $\mathfrak{c}_i \in \mathcal{G}$ oraz $0 \leqslant \mathfrak{a} \leqslant |\mathfrak{c}_i| - \ell'$, przy czym wartości $\mathfrak{a}$ w pełnym genomie (po uwzględnieniu obecności wszystkich chromosomów) są realizacją zmiennej losowej o rozkładzie jednostajnym. Przekształcenia te są przeprowadzane za pomocą następujących funkcji:

- $f_{\text{subst}} : \{0, 1, \ldots, \ell^* - 1\} \to [0, 1]$, $f_{\text{subst}}(a) = p(\mathfrak{q}^*[a])$,
- $f_{\text{del}} : \{0, 1, \ldots, \ell^* - 1\} \to \{\textbf{true}, \textbf{false}\}$, $P_{\text{r}}(f_{\text{del}}(a) = \textbf{true}) = p_{\text{del}} = \text{const}$,
- $f_{\text{ins}} : \{0, 1, \ldots, \ell^* - 1\} \to \{\textbf{true}, \textbf{false}\}$, $P_{\text{r}}(f_{\text{ins}}(a) = \textbf{true}) = p_{\text{ins}} = \text{const}$,

gdzie ℓ^* jest aktualną długością odczytu, mogącą ulegać zmianie na skutek pojawiania się błędów insercji oraz delecji, $\mathfrak{r}^*$ oraz $\mathfrak{q}^*$ są odpowiednio sekwencjami tego odczytu i ciągiem współczynników jakości, być może zmienionymi z tego samego powodu, a jest wartością z dziedziny odpowiedniej funkcji, natomiast P_{r} oznacza prawdopodobieństwo. Sekwencja $\mathfrak{q}^*$ jest ciągiem wartości będących realizacją pewnej zmiennej losowej. Ponadto zdefiniowano funkcję modelującą błędy substytucji $f_{\text{c}} : \{0, 1, \ldots, \ell^* - 1\} \to \Sigma$, przyporządkowującą pozycjom odczytu symbole zgodnie z regułą:

$$f_{\text{c}}(a) = \begin{cases} r[i], \text{ z prawdopodobieństwem } (1 - f_{\text{subst}}(a)), \\ \text{dowolne } c \in (\Sigma \setminus \{r^*[a]\}), \text{ z prawdopodobieństwem } \frac{f_{\text{subst}}(a)}{|\Sigma|-1}, \end{cases} \tag{5.17}$$

oraz funkcję modelującą symbole pojawiające się w efekcie wystąpienia błędów insercji $f_{\text{c\_ins}} : \{0, 1, \ldots, \ell^* - 1\} \to \Sigma$ przyporządkowującą pozycjom odczytu

symbole zgodnie z regułą:

$$f_{\text{c\_ins}}(a) = \text{dowolne } c \in \Sigma, \text{ z prawdopodobieństwem } \frac{1}{|\Sigma|}. \tag{5.18}$$

Znaczenie powyższych funkcji jest następujące:

- f_{subst} określa prawdopodobieństwo zamiany danego symbolu innym (wystąpienia błędu typu substytucja),
- f_{del} określa prawdopodobieństwo usunięcia danego symbolu (wystąpienia błędu typu delecja),
- f_{ins} określa prawdopodobieństwo wstawienia symbolu bezpośrednio za danym symbolem (wystąpienia błędu typu insercja),
- f_{c} określa, jaki symbol zostanie pojawi się na danej pozycji w zależności od prawdopodobieństwa błędu typu substytucja,
- $f_{\text{c\_ins}}$ określa, jaki symbol pojawi się po danej pozycji w odczycie w momencie pojawienia się błędu typu insercja.

Przyjęto, że wartość $p_{\text{ins}} \approx p_{\text{del}}$ oraz $f_{\text{subst}}(a) \gg p_{\text{ins}}$; w praktyce $\frac{f_{\text{subst}}(a)}{p_{\text{ins}}} \in [10, 100]$. Dokładne wartości współczynników są zależne od własności konkretnego zestawu odczytów, przy czym wartość $f_{\text{subst}}(a)$ może zostać wyznaczona na bieżąco w oparciu o współczynniki jakości $\mathfrak{q}^*$, natomiast wartości p_{del} oraz p_{ins} są stałe, jednak nieznane dokładnie (w każdym przypadku we właściwej dziedzinie funkcji f_{subst}).

Przekształcenie sekwencji $\mathfrak{r}'$ w sekwencję $\mathfrak{r}$ jest ciągiem $\mathfrak{S}$ wartości funkcji $f_{\text{subst}}, f_{\text{del}}, f_{\text{ins}}$ oraz odpowiadających im funkcji $f_{\text{c}}, f_{\text{c\_ins}}$, jednak z uwzględnieniem następujących reguł:

- jeżeli dla pewnego a $f_{\text{del}}(a) = \textbf{true}$, wówczas $f_{\text{del}}(a-1) = f_{\text{del}}(a+1) = \textbf{false}$ (w dziedzinie funkcji f_{del}),
- powyższa zasada jest prawdziwa także dla funkcji f_{ins},
- jeżeli dla pewnego a $f_{\text{del}}(a) = \textbf{true}$, wówczas $f_{\text{ins}}(a-5) = f_{\text{ins}}(a-4) = \ldots = f_{\text{ins}}(a+5) = \textbf{false}$ (w dziedzinie funkcji f_{ins}),
- powyższa zasada jest prawdziwa także po zamianie funkcji f_{del} na f_{ins} i na odwrót.

Powyższy model zakłada prawdziwość następujących uproszczeń:

- prawdopodobieństwo błędu insercji lub delecji jest niezależne od pozycji w odczycie,

- współczynniki jakości są prawidłowe, tzn. reprezentują rzeczywiste prawdopodobieństwo błędu substytucji,
- wystąpienie błędów substytucji na różnych pozycjach jest niezależne, tj. $P_{\mathrm{r}}(\mathfrak{r}[a]$ jest błędny$|\mathfrak{r}[b]$ jest błędny$) = P_{\mathrm{r}}(\mathfrak{r}[a]$ jest błędny$)$, gdzie $a, b \in \{0, 1, \ldots, \ell^* - 1\}, a \neq b$,
- nie występują błędy systematyczne, np. pominięte jest większe prawdopodobieństwo błędów w sekwencji GGC,
- w błędach substytucji prawdopodobieństwo błędu jest niezależne od sekwencji, tj. $\neg(f_{\mathrm{subst}}(i) \propto \mathfrak{r}[i])$,
- w błędach substytucji wartość błędnego symbolu jest niezależna od symbolu prawidłowego, tj. $\neg(f_{\mathrm{c}}(i) \propto \mathfrak{r}[i])$,
- pokrycie odczytami (a w rezultacie i k-merami) jest równomierne, tzn. pozycje poszczególnych odczytów w genomie charakteryzują się rozkładem jednostajnym,
- długość błędów typu indel wynosi 1 pz,
- jeżeli w odczycie występuje błąd typu insercja, wówczas najbliższy błąd typu delecja może wystąpić na pozycji różniącej się co najmniej o 6.

5.4.2 Analiza wymagań i celów nowego algorytmu

Postawione wymagania

Efekt działania algorytmu powinien w jak największym stopniu spełniać wymagania sformalizowane kryteriami oceny, przytoczone w podrozdziale 4.4. W szczególności przyjęto, że algorytm powinien cechować się następującymi własnościami:

1. Powinien spełniać wymagania dotyczące wykorzystania zasobów obliczeniowych. W szczególności powinien charakteryzować się niskim zapotrzebowaniem na pamięć operacyjną, gdyż spełnienie tego warunku może determinować możliwość wykonania algorytmu na danej maszynie. Dominującymi pod względem zapotrzebowania na pamięć zbiorami danych algorytmu będą spektrum k-merów oraz zestaw ścieżek korekcji aktualnego odczytu. Z obserwacji autora wynika, że pozostałe składowe, jak zbiór przetwarzanych w danej chwili odczytów czy dane konfiguracyjne, praktycznie zajmują stałą, niewielką ilość pamięci. Co więcej, zbiór ścieżek korekcji osiąga znaczne rozmiary tylko w niewielkiej liczbie odczytów (orientacyjnie rzędu jednego odczytu na milion), co jednak może mieć znaczący wpływ na szczytowe zapotrzebowanie na pamięć, przekraczając zajętość pamięci przez spektrum.

2. Powinien zapewniać odpowiednio krótki czas przetwarzania danych, rzędu kilku godzin dla ludzkiego genomu o dużej głębokości sekwencjonowania, oraz minut dla niewielkich organizmów. Cel ten powinien być osiągnięty nie tylko przez staranną implementację, ale również przez eliminację pesymistycznych przypadków złożoności obliczeniowej, które dla algorytmów z powrotami mogą być złożonościami wykładniczymi, powodując w niektórych przypadkach bardzo duży wzrost liczby rozpatrywanych ścieżek korekcji (tzw. *eksplozję kombinatoryczną*). Duża liczba możliwości może być powodowana przez występowanie powtórzeń w genomie organizmu [138]. Ponadto duża liczba ścieżek korekcji może stanowić większe wyzwanie w wyborze najlepszej z nich, zwłaszcza gdy ich jakości są zbliżone do siebie, a także wzrost zapotrzebowania na pamięć. W implementacji powinno zostać wykorzystane przetwarzanie współbieżne, zapewniające dobrą skalowalność.

3. Kluczowa jest jakość wyników ocenianych według różnych kryteriów, w szczególności wywierania wpływu na rzeczywiste zadania przetwarzania odczytów. Wpływ ten powinien obejmować eliminację możliwie dużej liczby błędów przy wyłącznie incydentalnym występowaniu przypadków wprowadzenia nowych błędów. Algorytm powinien spełniać takie wymagania dla możliwie szerokiego zakresu odczytów, w szczególności odczytów organizmów o genomach bardzo różnych rozmiarów, różnych wartościach głębokości sekwencjonowania oraz różnych charakterystykach genomów, np. rozmaitych liczbach i charakterach powtórzeń fragmentów genomów, różnych stosunkach liczby zasad A i T do C i G (wartościach *współczynnika GC* — ang. *GC content*).

4. Powinien mieć zastosowanie w korekcji odczytów obarczonych błędami podobnymi do charakterystyki błędów odczytów z sekwenatorów firmy Illumina. Odczyty tego rodzaju dominują pod względem popularności, a w konsekwencji są spotykane w większości prac badawczych.

5. Powinien w minimalnym stopniu wymagać parametryzacji przez użytkownika, szczególnie danymi o trudnych do określenia wartościach. Co więcej, wpływ parametrów na jakość wyników, szczególnie określanych przez użytkownika, powinien być — w rozsądnym zakresie — niewielki. Zapewni to mniejsze ryzyko uzyskania słabych wyników na skutek dalekiej od optymalnej parametryzacji wykonanej przez użytkownika.

6. Implementacja powinna spełniać wymagania techniczne: zapewniać obsługę danych skompresowanych (zarówno na wejściu, jak i wyjściu algorytmu), zachowywać ewentualne sparowanie odczytów, zachowywać format plików oraz pozwalać na przetwarzanie dowolnej liczby plików jednocześnie, np. w celu obsługi przypadku, gdy jedna próbka została poddana sekwencjonowaniu poprzez kilkukrotne uruchomienie urządzenia sekwencjonującego.

Implementacja powinna także zapewniać prostą instalację oraz uruchomienie algorytmu, redukując możliwość popełnienia błędu przez użytkownika. Skomplikowane użycie może być przyczyną rezygnacji z wykorzystania algorytmu (przykładowo, w pracy [210] nie uwzględniono w analizie porównawczej narzędzia BLESS ze względu na problematyczną instalację).

7. Powinien być pozbawiony wyszczególnionych w podrozdziale 5.3.4 niedoskonałości algorytmu BLESS.

Zrezygnowano z wymagania obejmującego dostarczenie możliwości parametryzowania algorytmu maksymalnym zapotrzebowaniem na pamięć, ponieważ rozmiar ten zależy przede wszystkim od własności genomu (jego rozmiaru, liczby i charakteru powtórzeń jego fragmentów) oraz od długości k-meru. Ponieważ parametry te są trudne do określenia oraz w dużej mierze zależne od własności danych wejściowych, dodanie kolejnego parametru znacząco skomplikowałoby wyznaczanie spektrum. Ponadto obserwacje autora wykazały, że dla rozsądnych wartości długości k-meru rozmiar spektrum może być odpowiedni do umieszczenia go w pamięci nawet komputerów osobistych. Wymaganie niewielkiego zapotrzebowania na przestrzeń dyskową nie ma znaczenia, ponieważ obejmuje ono wyłącznie przechowywanie plików wejściowych oraz wyjściowych, jako że przyjęta strategia zakłada, iż pliki tymczasowe, zawierające rozwiązania dla podzbiorów odczytów, będą stanowiły fragmenty plików wyjściowych. Generacja tych ostatnich polega na dołączeniu kolejnych plików tymczasowych do wynikowych, zatem łączny rozmiar plików tymczasowych oraz niepełnych wyjściowych nie może przekraczać rozmiaru gotowego zestawu plików wyjściowych.

W kolejnych podrozdziałach sprecyzowano, jakie konkretne rozwiązania zostały zastosowane w algorytmie RECKONER w celu spełnienia powyższych wymagań, odróżniając go od innych algorytmów, a które jak dotąd nie zostały zaproponowane w literaturze, były zaproponowane w innej formie, bądź były wykorzystywane w odmiennych okolicznościach. Pokrótce zwrócono też uwagę na inne podjęte próby, które jednak z różnych przyczyn nie zostały ostatecznie wdrożone.

Ogólne nowości w algorytmie

Do podstawowych nowości obecnych w algorytmie RECKONER należą ulepszenia techniczne, polegające na wprowadzeniu zliczania k-merów przy pomocy narzędzia KMC. W efekcie zrezygnowano z obecnej w algorytmie BLESS obsługi filtru Blooma, w miejsce którego wykorzystano struktury danych udostępniane przy pomocy KMC API, a w konsekwencji zaniechano także dalszego wykorzystania algorytmu rozwiązywania problemu fałszywych odpowiedzi twierdzących filtru. Wprowadzono też możliwość określenia nieparzystej długości k-meru.

Wprowadzono nową strukturę danych reprezentującą zmiany wprowadzane w odczycie, będącą *ciągiem korekcji* (w wielu przypadkach pełniącym rolę stosu

zmian, jakie kolejno są wprowadzane do regionu). Struktura ta pozwala na czytelną reprezentację ciągu operacji zmieniających region, przy czym czytelność wynika z faktu, że RECKONER jest algorytmem rekurencyjnym, a kolejne zagłębione instancje korygujących funkcji w sposób naturalny odpowiadają pozycjom ciągu. Dodatkowo zastosowanie ciągu umożliwiło eliminację technicznego wymagania, jakie występowało się w algorytmie BLESS, a polegającego na bardzo częstym kopiowaniu niepełnej ścieżki korekcji przy zagłębianiu się algorytmu (np. pseudokod 5, linia 42). Zamiast tego przyjęto dodawanie informacji o zmianach w ciągu oraz wykonywanie jego kopii wyłącznie w przypadku uzyskania właściwej ścieżki korekcji.

Poszczególnymi elementami ciągu korekcji ($\mathfrak{s}_i$), zdefiniowanego dla pewnego błędnego regionu ψ, są krotki $\mathfrak{s}_i = (\eta, c, \text{ins}, \text{del}, \text{exp\_ind})$, gdzie:

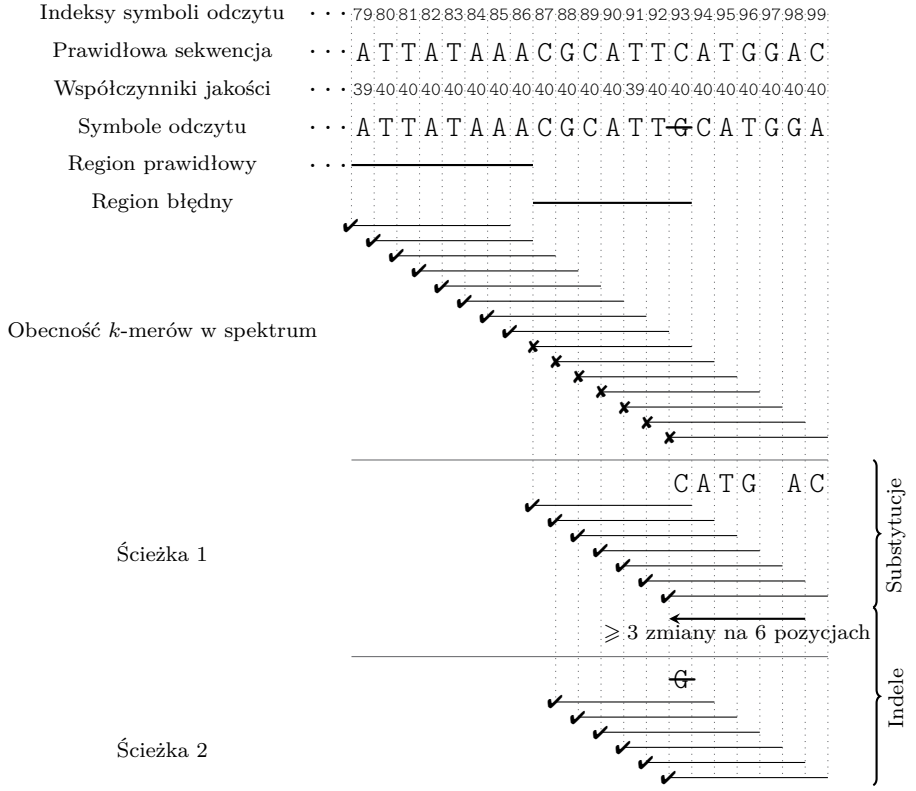
- η jest liczebnością k-meru pokrywającego daną modyfikację (w korekcji w kierunku $3'$ jest to pokrywający k-mer o najmniejszym indeksie, w korekcji w kierunku $5'$ k-mer o największym indeksie spośród wszystkich pokrywających),
- $c \in \Sigma$ jest symbolem korygującym błąd substytucji, znajdującym się na pozycji odczytu odpowiadającej danemu elementowi ciągu, albo oryginalnym symbolem tej pozycji (w przypadku gdy pozycja ta nie jest korygowana), albo symbolem wstawianym w celu korekcji błędu delecji,
- ins $\in \{\textbf{true}, \textbf{false}\}$ jest znacznikiem określającym spodziewane wystąpienie błędu insercji; po zaakceptowaniu ścieżki korekcji symbol na odpowiadającej mu pozycji zostanie usunięty,
- del $\in \{\textbf{true}, \textbf{false}\}$ jest znacznikiem określającym spodziewane wystąpienie błędu delecji; po zaakceptowaniu ścieżki korekcji przed symbolem danej pozycji zostanie wstawiony symbol c (w przypadku korekcji w kierunku $3'$) lub po nim (w przypadku korekcji w kierunku $5'$),
- exp_ind $\in \{\textbf{true}, \textbf{false}\}$ jest znacznikiem określającym, że po wykonaniu powrotu algorytmu należy na odpowiadającej mu pozycji przeprowadzić niezależnie próbę korekcji błędu typu insercja oraz błędu typu delecja, a dla każdego przypadku kontynuować korekcję w odpowiednim kierunku.

Jednocześnie dla elementów danej krotki $\mathfrak{s}_i$ spełniony musi być następujący warunek: ins $\wedge \neg$del $\wedge\, c = \varnothing \vee \neg$ins $\wedge\, c \neq \varnothing$.

Idea korekcji polega na wstawianiu kolejnych zmian do ciągu (odkładaniu na stosie) zmian i wycofywaniu ich w momencie dotarcia do uzyskania prawidłowego rozwiązania (tj. właściwej ścieżki korekcji) bądź natrafienia na moment, gdy dalsza rozbudowa prawidłowego rozwiązania nie jest możliwa. W pierwszym przypadku następuje utworzenie w oparciu o zawartość ciągu nowego ciągu zawierającego pełną ścieżkę korekcji π i dodaniu go do zbioru ścieżek $\mathbf{\Pi}$.

Nowości jakościowe

Korekcja błędów typu indel Kwestia detekcji i korekcji błędów typu indel jest istotna, a jednocześnie pomijana przez wielu twórców algorytmów korekcji. Błąd tego rodzaju czasami może zostać skorygowany poprzez wprowadzenie odpowiednio dużej liczby zmian symboli w sposób, jak przy korekcji błędów substytucji, aż do osiągnięcia końca odczytu. Przykład takiej sytuacji został przedstawiony na rys. 5.3, gdzie na pozycji 93 w wyniku błędu sekwencjonowania został wstawiony nowy symbol T (insercja). Błąd ten może zostać skorygowany poprzez zastosowanie ścieżki 1. Należy zaznaczyć, że sytuacja ta może mieć miejsce wyłącznie w skrajnych regionach. Co więcej, znaczna część spośród takich błędów pozostanie nieskorygowanych lub, jeżeli istnieje obszar genomu o podobnej sekwencji, może doprowadzić do wprowadzenia błędnej korekcji, ponieważ ścieżki tego rodzaju, ze względu na obejmowanie wielu zmian, będą posiadały niską ocenę.



Rysunek 5.3: Korekcja błędu typu insercja poprzez korekcję błędów substytucji; $k = 7, \ell = 100$; symbolem − oznaczono substytucję

Detekcja potencjalnych błędów typu indel została opracowana jako element mechanizmu przeszukiwania wyczerpującego algorytmem z powrotami. Jeżeli podczas korekcji pojawi się duża liczba następujących po sobie korekcji błędów substytucji, wówczas następuje oznaczenie pozycji początku takiej grupy jako miejsca

potencjalnego występowania błędu typu indel (znacznik exp_ind = **true**). Korekcja błędów substytucji jest kontynuowana, jednakże po jej zakończeniu (z sukcesem bądź bez) następuje powrót z jednoczesnym wyszukiwaniem znacznika pozycji potencjalnego indelu. Po jego wykryciu następuje kontynuacja korekcji od miejsca znacznika poprzez wprowadzenie jednej z 5 zmian: korekcji błędu insercji (następuje usunięcie symbolu) albo błędu delecji (następuje wstawienie jednego z czterech symboli alfabetu Σ). Dalsza praca algorytmu przebiega standardowo. Wszystkie właściwe ścieżki korekcji są zapamiętywane i poddawane ocenie w celu wyboru najlepszej.

Na przykładowym rysunku zaznaczono także alternatywną ścieżkę korekcji. Podczas analizy pozycji odczytu 93 do 100 następuje obserwacja liczby wprowadzonych zmian. Po wprowadzeniu zmiany na pozycji 98 zostają przekroczone limity oznaczające potencjalne wystąpienie błędu typu indel. Z tego powodu następuje oznaczenie pozycji początku ciągu zmian, tj. 93, jako ewentualnego błędu typu indel. Po osiągnięciu końca regionu i wygenerowaniu ścieżki 1 powrót algorytmu do coraz niższych pozycji odbywa się wraz z detekcją takiego oznaczenia. Jego napotkanie inicjuje korekcję błędu typu insercja polegającą na usunięciu symbolu G oraz kontynuacji aż do natrafienia na koniec regionu oraz — ze względu na jej powodzenie korekcji — utworzenia ścieżki 2. Ścieżka ta zawiera tylko jedną zmianę konieczną do uzyskania zaufanych k-merów w całym regionie. Region po korekcji zawiera też o jeden k-mer mniej niż przed korekcją. Na rysunku pominięto występującą później próbę korekcji błędu typu delecja, polegającą na wstawianiu nowego symbolu między pozycjami 92 i 93.

Wykorzystanie liczebności k-merów i ocena ścieżek Częstym rozwiązaniem proponowanym w literaturze i wykorzystywanym w detekcji błędów, obok stwierdzania nieobecności określonych k-merów w spektrum, jest uwzględnienie wartości współczynników jakości. W ograniczonym zakresie niosą one informację o możliwości wystąpienia błędu substytucji, nie zawierają jednak informacji o obecności błędów typu indel. Współczynniki jakości mogą być stosowane także podczas wykonywania oceny ścieżek korekcji. W obu przypadkach należy jednak zwrócić uwagę na wspomnianą wcześniej niepełną zgodność między zakodowanym prawdopodobieństwem a rzeczywistością, czy ograniczoną liczbę poziomów kwantyzacji współczynnika.

Rozszerzoną metodą zastosowania danych w obu przypadkach może być oparcie się na liczebnościach k-merów. Wiąże się to z przyjęciem założenia, że liczebność k-meru w spektrum odpowiada poziomowi pewności, że dany k-mer jest prawidłowy. Oparcie na takiej tezie powinno być jednak przeprowadzone ostrożnie (z przyczyn, które zostaną omówione w podrozdziale 5.4.2, w opisie redukcji liczebności). W algorytmie RECKONER każda z wygenerowanych właściwych ścieżek korekcji zostaje poddana ocenie, której wynik decyduje o tym,

która z nich zostanie zastosowana w modyfikacji regionu odczytu. Dla pewnego regionu $\psi = (a, b)$ ocenę ścieżki π zdefiniowano następująco, odpowiednio dla regionów korygowanych w kierunku końca $3'$ oraz w kierunku końca $5'$:

$$r_{\pi,3'}(a, b, \mathcal{K}_\pi) = \frac{\sum\limits_{\kappa \in \mathcal{K}_\pi} \text{weight}(\kappa)\eta(\kappa)}{\sum\limits_{\kappa \in \mathcal{K}_\pi} \text{weight}(\kappa)} \left(\prod_{i=a+k-1}^{b+k-1} \text{prob}(i) \right) (\theta_{\text{IND\_PROB}})^{n_{\text{ind}}}, \quad (5.19)$$

$$r_{\pi,5'}(a, b, \mathcal{K}_\pi) = \frac{\sum\limits_{\kappa \in \mathcal{K}_\pi} \text{weight}(\kappa)\eta(\kappa)}{\sum\limits_{\kappa \in \mathcal{K}_\pi} \text{weight}(\kappa)} \left(\prod_{i=a}^{b} \text{prob}(i) \right) (\theta_{\text{IND\_PROB}})^{n_{\text{ind}}}. \quad (5.20)$$

Elementami zbioru $\mathcal{K}_\pi$ są k-mery, odpowiednio $\mathfrak{r}^*[a, a+k-1], \mathfrak{r}^*[a+1, a+k], \ldots, \mathfrak{r}^*[b, b+k-1]$ albo $\mathfrak{r}^*[0, k-1], \mathfrak{r}^*[1, k], \ldots, \mathfrak{r}^*[b-k+1, b]$, w zależności od kierunku korekcji, oraz pierwszy k-mer rozszerzający o najwyższej ocenie (zgodnie z definicją rozszerzania regionu algorytmu BLESS). Rozszerzający k-mer, w przypadku gdy π nie zawiera korekcji błędów typu indel, posiada sekwencję odpowiednio $\mathfrak{r}^*[\ell-k+1, \ell-1]c$ albo $c\mathfrak{r}^*[0, k-2]$, gdzie c jest rozszerzającym symbolem, a $\mathfrak{r}^*$ sekwencją odczytu po zastosowaniu ścieżki π. Pseudokody w dalszej części pracy obejmują przypadek, gdy ścieżka zawiera również korekcje błędów typu indel. Wartość n_{ind} jest liczbą korekcji błędów typu indel, $\theta_{\text{IND\_PROB}} = 0{,}001$ jest szacunkowym prawdopodobieństwem błędu typu indel (przybliżoną wartością parametrów p_{ins} oraz p_{del}), z kolei funkcja $\text{prob}(i)$ jest zdefiniowana następująco:

$$\text{prob}(i) = \begin{cases} 1, \text{ gdy symbol odpowiadający pozycji } i \text{ w regionie nie był} \\ \quad \text{zmodyfikowany,} \\ p(\mathfrak{q}[i]) \text{ w przeciwnym razie.} \end{cases} \quad (5.21)$$

Funkcja wagowa jest zdefiniowana następująco:

$$\text{weight}(\kappa) = \begin{cases} \theta_{\text{EXT\_WEIGHT}}, \text{ gdy } \kappa \text{ jest } k\text{-merem rozszerzającym,} \\ \theta_{\text{COV\_WEIGHT}} \text{ w przeciwnym razie.} \end{cases} \quad (5.22)$$

Funkcja oceny ścieżek korekcji pełni rolę podlegającej maksymalizacji funkcji celu. Zaproponowana metoda ma na uwadze rozwiązanie wspomnianych wyżej problemów. Z jednej strony uwzględnia współczynniki błędów symboli, stanowiące niedokładną, ale bezpośrednią informację o możliwości wystąpienia błędów. Wyżej oceniane są ścieżki zawierające mniejszą liczbę modyfikacji, unikając tym samym wprowadzenia dużej liczby zmian do regionu, zmieniając jego sekwencję na inną, być może obecną w rejonie genomu o wyższym pokryciu odczytami. Tym samym za korzystną uznawana jest modyfikacja zmniejszająca odległość edycyjną oryginalnego i zmodyfikowanego regionu, co należy uznać za zjawisko pozytywne [110]. Mnożenie wartości prawdopodobieństwa, zakładając, że są to prawdopodobieństwa zdarzeń niezależnych, pozwala na szacunkowe określenie prawdopodobieństwa danego ciągu błędów obecnych w regionie. Z drugiej strony brana jest

pod uwagę liczebność k-merów, preferując ścieżki obejmujące k-mery o wyższych liczebnościach.

W niektórych przypadkach pewne wartości współczynników jakości posiadają specjalne znaczenie, np. $q = 2$ może nie oznaczać konkretnego prawdopodobieństwa błędu [152], jednak kwestię tę pominięto ze względu na brak pewności odnośnie do standaryzacji tego zjawiska oraz ze względu na fakt, że tak niskie współczynniki jakości, po zastosowaniu funkcji $p(q)$, powodują uzyskanie wysokiej wartości prawdopodobieństwa błędu, bliskiego pewności, że odpowiadający mu symbol jest błędny. Wobec tego i tak podlegają one staranniejszej korekcji niż inne symbole.

Ocena korekcji pierwszego k-meru jest przeprowadzana analogicznie, przy czym zbiór $|\mathcal{K}_\pi|$, posiadający po wykonaniu korekcji tego k-meru moc 1, jest uzupełniany o różne zestawy k-merów dla pozostałej części odczytu.

Dobór dwóch długości oligomerów W celu poprawy precyzji korekcji wprowadzono rozwiązanie polegające na weryfikacji korekcji w oparciu o oligomery długości $k'' > k$. Dodatkowa baza danych k''-merów jest generowana z progiem obcięcia $\eta''_{\text{cut}} = 1$, tworząc spektrum $\mathcal{K}''_1$. Tym samym zawiera wszystkie k''-mery występujące w sekwencjach odczytów wejściowych. Obecność w tej bazie k''-merów uzyskanych ze skorygowanego odczytu może stanowić dodatkowe kryterium wyboru ścieżek korekcji, a nawet rezygnacji z korekcji danego odczytu.

W pierwszej kolejności następuje przeprowadzenie typowej korekcji z użyciem k-merów, efektem której jest uzyskanie dla każdego błędnego regionu zestawu ścieżek korekcji wraz z przyporządkowanymi ocenami. Do odczytu stosowane są odpowiednie ścieżki o najwyższych ocenach, po czym następuje wyodrębnienie z niego wszystkich k''-merów. Jeżeli każdy z nich należy do spektrum k''-merów, wówczas dany zestaw ścieżek jest przyjmowany. W przeciwnym wypadku zmieniana jest ścieżka w jednym z regionów na posiadającą niższą ocenę. Wybór regionu i jego ścieżki do zmiany jest przeprowadzany w taki sposób, aby suma ocen dobranych ścieżek wszystkich regionów uległa jak najmniejszej redukcji. Sprawdzanie k''-merów jest powtarzane do momentu znalezienia sekwencji odczytu całkowicie pokrytej k''-merami lub wyczerpania zestawu ścieżek.

W niewielkim stopniu podobna idea była zastosowana w algorytmie HiTEC [108], gdzie wykorzystano dwie długości oligomerów: „krótkie”, mające na celu redukcję liczby niekorygowanych odczytów, oraz „długie”, mające na celu redukcję przypadków uszkodzenia odczytów [108]. Ze względu na konieczność przechowania w pamięci dwóch baz danych oligomerów oraz konieczność wykonania znacznie większej liczby czasochłonnych zapytań do baz, rozwiązanie to jest uruchamiane fakultatywnie.

Wyniki wstępnych eksperymentów wykazały, że dodatkowa weryfikacja przynosi korzystne rezultaty w przypadku korekcji odczytów poddawanych później

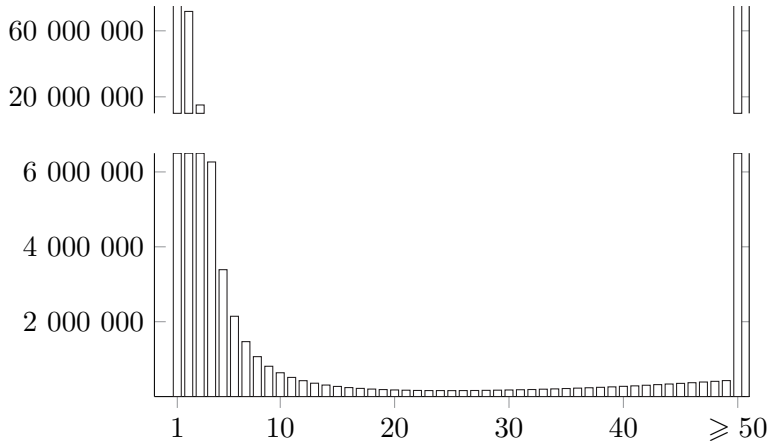
asemblacji. Jednocześnie jej wpływ na detekcję wariantów okazał się negatywny. Jako współczynnik zwiększający długości oligomeru przyjęto wartość $\theta_{\text{LONG\_RATIO}} = 1{,}5$.

Dopuszczenie niezaufanych k-merów Jak zostało wspomniane w podrozdziale 4.1.3, wystąpienie jednego błędu substytucji może skutkować wygenerowaniem k błędnych k-merów. Jednakże niektóre spośród pokrywających błąd k-merów mogą pojawić się w innych rejonach genomu, choć zjawisko to nie powinno być częste. Wobec tego większość spośród k-merów pokrywających błędną pozycję odczytu nie będzie obecnych w spektrum. Jednocześnie jednak może wystąpić zjawisko odwrotne, gdy część spośród k-merów pokrywających prawidłowy (lub prawidłowo skorygowany) symbol nie będzie znajdowała się w spektrum. Sytuacja ta może dotyczyć szczególnie przypadków, gdy liczebności k-merów danego odczytu są niewielkie i nieznacznie przekraczają wartość progu obcięcia.

W związku z tym w algorytmie dopuszczono przypadek, gdy podczas korygowania odczytu część spośród k-merów skorygowanego regionu nie znajduje się w spektrum. W tym przypadku, w razie niepowodzenia wyznaczenia zaufanego k-meru, następuje przyjęcie oryginalnego symbolu końcowego k-meru i kontynuacja algorytmu. Tym samym do grupy k-merów obiecujących wchodzących w skład właściwej ścieżki korekcji zaliczane są także k-mery niezaufane, pod warunkiem, że spełniają poniższe ograniczenie. W celu ograniczenia liczby obsługiwanych przypadków oraz ze względu na spodziewane rzadkie występowanie takiego zjawiska przyjęto warunek, że liczba dopuszczalnych niezaufanych k-merów wynosi $\theta_{\text{MX\_NON\_UNTRST}} = 5\%$ liczby symboli w oryginalnym regionie. W ocenie ścieżki liczebność niezaufanych k-merów wynosi 0, zatem preferowane są ścieżki, w których dopuszczono jak najmniej niezaufanych k-merów.

Dodatkowa korekcja symboli słabej jakości W celu zwiększenia czułości algorytmu poprzez dodatkową eksploatację przestrzeni rozwiązań, jako dodatkowe kryterium obecności błędów uwzględniane są wartości współczynników jakości. Jeżeli końcowy symbol bieżącego k-meru posiada wartość współczynnika jakości nie większą niż $\theta_{\text{Q\_LOW}} = 3$, wówczas następuje podjęcie próby korekcji polegającej na modyfikacji tego symbolu. Rozwiązanie to zapobiega sytuacji, gdy oryginalny k-mer zawierający błąd (wskazywany wyłącznie przez niski współczynnik jakości) znajduje się w spektrum, jednak następujące po nim k-mery nie. Taka sytuacja w rezultacie prowadziłaby do nieznalezienia żadnej ścieżki korekcji.

Określanie progu obcięcia Wyznaczanie wartości progu obcięcia jest przeprowadzane w sposób podobny jak w niektórych innych algorytmach, tzn. poprzez analizę histogramu liczebności k-merów, gdzie jako próg przyjmowana jest liczebność, dla której histogram przyjmuje pierwsze minimum. Zaobserwowano jednak,



Rysunek 5.4: Histogram liczebności 25-merów odczytów z zestawu SRR1945754

że w wielu zestawach danych minimum to występuje dla dużych wartości liczebności, jak np. na rys. 5.4 dla $\eta = 25$.

Przypadki takie mogą wystąpić m.in. w sytuacjach, gdy głębokość sekwencjonowania jest bardzo mała albo duża, gdy pokrycie genomu odczytami jest nierównomierne, tzn. w genomie licznie występują pozycje obecne w niewielkiej liczbie odczytów, a także w odczytach genomów diploidalnych [89]. W celu zabezpieczenia przed takim zjawiskiem, a w konsekwencji doborem zbyt dużej wartości η_{cut}, ograniczono jego wartość do $\eta_{\text{MX\_CUT}} = 5$. Przyjęto, że nawet dla małych wartości k oraz wysokiej głębokości sekwencjonowania k-mery błędne (a jednocześnie nieposiadające sekwencji równej innym, prawidłowym k-merom) osiągną taką liczebność tylko w incydentalnie małej liczbie przypadków. Badania eksperymentalne wykazały, że w wielu zestawach danych ograniczenie było stosowane (np. tabele B.20 oraz B.21).

Wybrane pozostałe rozwiązania W trakcie projektowania algorytmu zwrócono uwagę na wiele innych drobnych kwestii, z których wybrane zostaną omówione poniżej. Analizie poddano sytuację ścieżek niezawierających żadnych korekcji (co sugeruje, że region oryginalnie jest pokryty zaufanymi k-merami). Mogą one wystąpić choćby wtedy, gdy skrajny końcowy błędny region zostanie wyznaczony jako rezultat np. skrócenia prawidłowego regionu w kroku 0.6 etapu wyznaczania błędnych regionów (omówionego w podrozdziale 5.4.3). Ścieżki tego rodzaju są traktowane w sposób uprzywilejowany, ponieważ nie są weryfikowane poprzez rozszerzanie regionu. Mimo to podlegają ocenie wraz z innymi ścieżkami tym samym w przypadku uzyskania wyższej oceny, region pokryty zaufanymi także k-merami może zostać zmodyfikowany.

Przeprowadzono strojenie licznych wewnętrznych parametrów, w ramach którego stwierdzono, że optymalną liczbą symboli rozszerzających region jest $\theta_{\text{MX\_E}} =$

3. Zauważono także, że w niektórych przypadkach limity korekcji powinny być nie tylko proporcjonalne do długości regionu, ale także sztywno ograniczone z góry (np. liczba dopuszczonych w regionie korekcji błędów typu indel jest ograniczona do $\theta_{\text{MX\_IND\_RATE}} = 0{,}1$ długości regionu, ale jednocześnie nie może być większa od $\theta_{\text{MX\_IND}} = 4$). Ponieważ w ocenie ścieżki uwzględniana jest liczebność pierwszego rozszerzającego k-meru, przeprowadzono obserwację wpływu wagi liczebności tego k-meru na wyniki. Najlepsze wyniki uzyskiwane są, gdy liczebność takich k-merów jest brana do oceny z wagą mniejszą (przyjęto $\theta_{\text{EXT\_WEIGHT}} = 0{,}5$) niż liczebności pozostałych k-merów (przyjęto wagę $\theta_{\text{COV\_WEIGHT}} = 1$), co uwzględniono w równaniach (5.19) oraz (5.20). Liczebności kolejnych rozszerzających k-merów nie są brane pod uwagę, ponieważ nie przynoszą poprawy jakości.

Liczba dopuszczalnych modyfikacji wprowadzanych w regionie, $\theta_{\text{MX\_CHCK}} = 10000$, także została dobrana we wstępnych eksperymentach — zaobserwowano, że mniejsze wartości przynoszą nieco słabsze rezultaty, a większe dopuszczają do czasochłonnej analizy bardzo trudne regiony, z niską szansą na znalezienie prawidłowej korekcji. Z kolei próba zwiększenia progu wartości współczynnika jakości, który wyznacza pozycje niskiej jakości $\theta_{\text{Q\_LOW}} = 3$, w przypadku detekcji wariantów skutkowała nieznaczną poprawą wyników detekcji wariantów typu SNP, ale wyraźnym pogorszeniem dla wariantów typu krótki indel.

W stosunku do algorytmu BLESS ograniczono liczbę symboli rozszerzających regiony wewnętrzne. Wprowadzono też dodatkowy przypadek korekcji błędów substytucji pierwszego k-meru — uwzględniona została sytuacja, gdy liczba symboli niskiej jakości przekracza limit algorytmu siłowego równy $\theta_{\text{MX\_LQ}} = 4$. W takim przypadku korekcja tą metodą może zostać przeprowadzona dla dokładnie $\theta_{\text{MX\_LQ}}$ takich pozycji, będących dodatkowo najmniejszymi spośród wszystkich symboli niskich jakości. Uzupełniono także wyznaczanie regionów odczytu, poprzez uzupełnienie skracania prawidłowych regionów, jeśli w pobliżu ich końców znajdują się symbole niskiej jakości.

Należy zauważyć, że w przypadku wprowadzania wielu ulepszeń algorytmu, uzyskane rezultaty były niejednoznaczne, np. powodowały poprawę jakości asemblacji, a przy tym pogarszały wyniki detekcji wariantów. Stąd też w przypadku najbardziej wyraźnych różnic między różnymi zastosowaniami zdecydowano się na zaproponowanie konkretnych trybów pracy dla różnych zastosowań (wyróżniono specjalizowane tryby: podstawowy, z weryfikacją długimi k-merami).

Nowości wydajnościowe

Limitowanie liczby ścieżek korekcji i modyfikacji Pesymistyczna liczba ścieżek korekcji wyznaczonych podczas wprowadzania zmian do regionu może rosnąć wykładniczo w funkcji długości regionu. Osiągnięcie złożoności tego rzędu zdarza się rzadko, np. dla niewielkich wartości k lub sekwencji regionu reprezentującej fragment genomu posiadający liczne podobne sekwencje w swoich innych częściach.

Z obserwacji autora wynika, że nawet dla dużych zestawów danych przypadki takie nie są bardzo częste, jednak mogą one powodować nawet kilkukrotny wzrost czasu pracy całego algorytmu, a przede wszystkim wzrost zapotrzebowania na pamięć w związku z zachowaniem licznych ścieżek korekcji. Zapotrzebowanie to może przekraczać nawet rozmiar spektrum. Jednocześnie przy dużym zbiorze ścieżek korekcji może ulec zmniejszeniu wiarygodność metody wyboru najlepszej ścieżki, ze względu na potencjalne duże podobieństwo ocen różnych ścieżek.

Z powyższych powodów liczba ścieżek wygenerowanych w jednym regionie jest ograniczona do $\theta_{\text{MX\_PTH}} = 100$. Zrezygnowano z wykorzystania bardziej zaawansowanych strategii; możliwa rezygnacja z przeszukiwania z góry określonej części drzewa powrotów algorytmu nie zagwarantowałaby, że potencjalnie uzyskane w innej części rozwiązanie byłoby prawidłowe, a wyróżnienie części drzewa sprawiającej trudność zwykle nie jest możliwe, gdyż zazwyczaj jest to cecha całego drzewa.

Problem wykładniczej złożoności czasowej korekcji regionu został wyeliminowany przez wprowadzenie ograniczenia liczby możliwości sprawdzanych w regionie, gdzie jako jedną możliwość traktowana jest jedna eliminacja błędu dowolnego rodzaju. Przykładowo: podjęcie na określonej pozycji regionu próby korekcji błędu substytucji poprzez sprawdzenie wartości A oraz C jest traktowana jako dwie zmiany. Liczba tego rodzaju zmian jest ograniczona poprzez uwzględnienie omówionego wcześniej limitu $\theta_{\text{MX\_CHCK}} = 10000$.

Po osiągnięciu któregokolwiek z wymienionych limitów, korekcja jest przerywana, ale uzyskane do tej pory ścieżki korekcji są oceniane i mogą zostać zastosowane w odczycie. Zaobserwowano, że całkowita rezygnacja z korekcji w takich przypadkach przynosiła słabsze rezultaty.

Do dodatkowych limitów należy zaliczyć ograniczenie do wartości $\theta_{\text{MX\_LQ}}$ liczby symboli rozszerzających regiony wewnętrzne. Pierwotnie rozszerzanie było przeprowadzane poprzez weryfikację $k-1$ k-merów, jednak rezygnacja z tak starannej analizy pozwoliła na osiągnięcie nieznacznie lepszych wyników oraz ograniczenie licznych zapytań do bazy k-merów.

Przetwarzanie równoległe Przeprowadzono zrównoleglenie wykonania algorytmu przy wykorzystaniu strategii opartej na scentralizowanej metodzie przydziału zadań przez wątek nadrzędny wątkom podrzędnym. Szczegółowa zasada zrównoleglenia zostanie wyjaśniona w podrozdziale 5.4.4.

Eliminacja wstępnego sprawdzania odczytów Wstępnym etapem pracy algorytmu BLESS było sprawdzenie każdego odczytu w celu zliczenia odczytów, określenia skali mapowania oraz weryfikacji, czy odczyty są tej samej długości (co stanowiło wymaganie algorytmu). W algorytmie RECKONER etap ten pozostawiono w celu określenia skali przyporządkowania współczynników jakości do kodu ASCII

oraz wyeliminowania oczywistych błędów formatu pliku. Przyjęto, że jeżeli początek pliku jest prawidłowy, zapewne jego całość także jest prawidłowa. Dokładna kontrola jest przeprowadzana w ramach etapu korekcji odczytów w sposób równoległy. Zapobiegło to konieczności dwukrotnej iteracji przez plik, z której pierwsza była przeprowadzana jednowątkowo, oraz skrócenie czasu obliczeń.

Wykorzystanie KMC tools Jak zaznaczono w podrozdziale 5.2.2, ważną funkcją narzędzia KMC tools jest filtracja (*redukcja*, wg nazewnictwa KMC tools) zbioru k-merów w oparciu o kryterium ich liczebności. Ze względu na kilkuetapowe przetwarzanie bazy danych k-merów oraz konieczność przechowania jej pełnej wersji w pamięci, przeprowadzenie filtracji jest atrakcyjnym rozwiązaniem. Jednym ze wstępnych etapów (zgodnie z rys. 5.5) jest zliczanie wszystkich k-merów o liczebności nie mniejszej niż 2, ponieważ wszystkie unikalne k-mery są uważane za błędne. Następnie wyznaczany jest histogram liczebności, co pozwala na wyznaczenie wartości kryterium filtracji. Zastosowanie KMC tools umożliwia w tym etapie rezygnację z czasochłonnego ponownego zliczania k-merów, przetwarzając istniejącą bazę. Ponadto filtracja skutkuje uzyskaniem wynikowej bazy w wersji 1, co, jak zostało podkreślone w podrozdziale 5.2.1, może pozwolić na szybsze wykonywanie zapytań do bazy.

Kwestia ta, pomimo technicznego charakteru, jest omawiana ze względu na fakt, że struktura danych bazy KMC jest dosyć skomplikowana, dając jednak duże możliwości. Przykładowo, obie omówione wersje algorytmu BLESS przeprowadzają filtrację podczas wstawiania informacji do filtru Blooma, co można uznać za rozwiązanie proste (i wystarczające), jednak ze względu na cel polegający na przechowaniu k-merów wraz z liczebnościami, w tym zapobiegnięcie występowania fałszywie twierdzących odpowiedzi filtru, wymaga bardziej skomplikowanego rozwiązania, które jest dostarczane wraz z KMC tools.

Nowości techniczne

Określanie skali przyporządkowania Wyróżnia się dwie podstawowe skale przyporządkowania współczynników jakości odczytów Illumina [160]: Phred+33 oraz Phred+64. Wyznaczają one wartość dodawaną do współczynników jakości w celu uzyskania kodów symboli ASCII, odpowiednio 33 lub 64. Różne narzędzia syntezy odczytów są opracowane do generacji różnych z nich, a także posiadają zdefiniowany różny zakres wartości współczynników, np. $0, 1, \ldots, 40$ lub $0, 1, \ldots, 41$ [160]. Algorytm RECKONER został wyposażony w prosty mechanizm detekcji skali. W pierwszym etapie następuje odczyt współczynników jakości pierwszych odczytów w pliku oraz analiza ich wartości do momentu, aż zostanie znaleziony taki współczynnik jakości q, że $q \leqslant (33 + 5)$ (następuje przyjęcie poziomu Phred+33) albo $q \geqslant (104 - 5)$ (następuje przyjęcie poziomu Phred+64). Wartość 33 wynika

z bliskości[2] dolnego kresu możliwych wartości w skali Phred+33, a 104 z górnego kresu ($64 + 40 = 104$) w skali Phred+64. Przyjęty współczynnik tolerancji równy 5 wynika z faktu istnienia różnic między oprogramowaniem, a także ze specjalnego znaczenia współczynników jakości, np. $q = 0$, które mogą nie pojawić się w zestawie lub pojawić się rzadko, wymagając wstępnej analizy większej liczby odczytów.

Kompresja Rozmiar przetwarzanych plików odczytów uzasadnia ich kompresję, najlepiej z zapewnieniem przetwarzania skompresowanych danych „w locie", bez wstępnej dekompresji całego pliku, oraz wykonaniu kompresji wyników na bieżąco, tzn. bez tworzenia tymczasowego, nieskompresowanego pliku wyjściowego. W celu zapewnienia szybkości poprzez wykorzystanie przetwarzania równoległego, odczyt pliku wraz z łatwą obliczeniowo dekompresją odbywa się w jednym dodatkowym wątku. Z kolei czasochłonna jest przeprowadzana przez liczne wątki robocze (odpowiedzialne także za korekcję), podlegając w ten sposób zrównolegleniu. Implementacja algorytmu została wyposażona w obsługę plików w formacie GZIP, jako zdecydowanie najpopularniejszego formatu kompresji odczytów. Wzięto pod uwagę fakt, że pliki formatu GZIP mogą składać się z ciągu skonkatenowanych, niezależnie skompresowanych fragmentów. Tym samym wyniki różnych wątków mogą zostać w prosty sposób połączone w plik wynikowy.

Automatyzacja określania długości oligomeru Ze względu na istotność parametru k położono większy nacisk na jego dobór niż pozostałych parametrów. Wstępne eksperymenty wykazały, że w rozsądnym zakresie wpływ długości oligomeru k na jakość wyników algorytmu nie jest wysoki. Jednakże pozostawienie obligatoryjnego doboru tego parametru użytkownikowi może być ryzykowne oraz stanowi funkcjonalne utrudnienie. Z tego powodu opracowano prosty algorytm jego doboru, działający w oparciu o własności zestawu odczytów oraz genomu. Zgodnie z informacjami zawartymi w podrozdziale 4.2, analityczne wyznaczenie tej wartości jest praktycznie zbyt trudne, stąd zaproponowana metoda została oparta na przesłankach intuicyjnych.

Długość oligomeru, oznaczana jako k_{pred} jest wyznaczana zgodnie z następującym równaniem:

$$k_{\text{pred}} = \max(20; 0{,}9 \log_2 \widehat{\ell_{\mathcal{G}}} + 3). \tag{5.23}$$

Uzasadnienie jego postaci i doboru wartości stałych, ze względu na oparcie na danych eksperymentalnych, zostaną przedstawione w dalszej części pracy, w podrozdziale 6.3. W równaniu tym jako zmienną niezależną wykorzystano szacunkowy rozmiar genomu. Jego wyznaczenie, choć w wielu algorytmach ograniczone

[2]Minimalna wartość współczynnika w kodzie ASCII w skali Phred+33 nie musi wynosić dokładnie 33.

do konieczności określenia przez użytkownika, również zostało zautomatyzowane przez implementację następującego równania:

$$\widehat{\ell_{\mathcal{G}}} = \frac{\mathfrak{n}\bar{\ell}}{\widehat{\text{dep}}}. \tag{5.24}$$

Powyższy wzór został uzyskany poprzez przekształcenie równania 3.2, przy czym jako $\bar{\ell}$ jest przyjmowana średnia długość odczytu, wyznaczana w oparciu o podsumowanie pracy algorytmu KMC, wykonanej w celu oszacowania głębokości sekwencjonowania. Ze względu na fakt, że w celu wyznaczenia wartości k długość genomu jest poddawana logarytmowaniu, nie musi być ona określona dokładnie.

Głębokość sekwencjonowania jest szacowana w oparciu o spektrum k-merów uzyskanych z odczytów. W pierwszej kolejności następuje zliczanie 25-merów za pomocą narzędzia KMC. Następnie oszacowanie jest określane zgodnie ze wzorem:

$$\widehat{\text{dep}} = \eta_{\max}\frac{\ell}{\ell - k + 1}. \tag{5.25}$$

Przyjmując funkcję $f : \mathbb{N}_+ \to \mathbb{N}$, która każdej wartości liczebności k-merów η przyporządkowuje liczbę sekwencji k-merów posiadających taką liczebność (funkcja ta jest matematycznym opisem rozkładu przedstawionego np. na histogramie 4.1), wartość $\eta_{\max} = \arg\max_{n\in\mathbb{N}, n>\eta_{\min}} f(n)$, gdzie $\eta_{\min}$ jest pierwszym minimum funkcji f. Innymi słowy, wartość $\eta_{\max}$ jest wyznaczana jako takie η, dla którego histogram liczebności k-merów spektrum dla $k = 25$ (po usunięciu k-merów poniżej progu obcięcia) osiąga maksimum.

Wartość $\eta_{\max}$ sama może pełnić rolę estymacji pokrycia, jednak obarczoną znacznym, powodującym niedoszacowanie błędem systematycznym, wynikającym m.in. z faktu, że z odczytu długości ℓ uzyskiwanych jest tylko $\ell - k + 1$ k-merów (tym samym liczba k-merów w genomie: $\text{dep}\cdot(\ell_{\mathcal{G}} - k + 1) \approx \text{dep}\cdot(\ell_{\mathcal{G}})$, ponieważ $\ell_{\mathcal{G}} \gg k$, jest większa niż liczba k-merów uzyskanych z odczytów: $\mathfrak{n}(\ell - k + 1)$) czy z obecności błędów, które „przesuwają" histogram k-merów w lewo. Pierwsza trudność jest łatwa do rozwiązania poprzez przemnożenie tej prostej estymacji przez iloraz $\frac{\ell}{\ell-k+1}$. Zrezygnowano z uwzględnienia innych poprawek, ponieważ sam problem oszacowania głębokości sekwencjonowania jest dostatecznie złożony na przeprowadzenie osobnych badań.

Pominięte rozwiązania

W trakcie opracowania algorytmu i przeprowadzania wstępnych eksperymentów stwierdzono, że część spośród planowanych do wprowadzenia rozwiązań nie wywiera pozytywnego wpływu na jakość wyników lub czas obliczeń. Wybrane pominięte rozwiązania dotyczące przetwarzania równoległego zostaną omówione w podrozdziale 5.4.4.

W algorytmie Quake [110] uwzględniono fakt różnego prawdopodobieństwa błędów substytucji między różnymi parami wartości (np. błąd zamiany symbolu A

na C jest bardziej prawdopodobny, niż na G oraz T). Podjęto próbę uwzględnienia tej kwestii w ocenie ścieżek, jednak nie spowodowało to poprawy rezultatów.

Przeanalizowano także możliwość zastosowania w ocenie ścieżek liczebności 3-merów, poprzez zwiększenie oceny ścieżek, które powodują uzyskanie sekwencji 3-merów o wyższych liczebnościach. Idea ta jest umotywowana występowaniem różnic w liczebnościach tak krótkich oligonukleotydów, np. będących efektem trójkowości kodu genetycznego. Zmiana ta także nie przyniosła poprawy jakości wyników. Dodatkowo podjęto próbę wprowadzenia minimalnej oceny, jaką musi uzyskać przyjęta ścieżka, jednak nawet niewielkie wartości rzędu 10^{-6} nie były korzystne.

Zrezygnowano także z limitowania liczby dopuszczalnych korekcji wprowadzanych w jednym regionie (w odróżnieniu od ograniczenia liczby ścieżek oraz liczby sprawdzanych zmian). Wspomniane dwie pozostałe strategie limitowania dobrze sprawdzają się w roli zabezpieczenia przed zbytnim wzrostem czasu obliczeń. Tymczasem ograniczenie liczby zmian w regionie mogłoby utrudnić detekcję błędów typu indel (opierającą się na zliczaniu zmian w regionie). Ewentualna poprawa jakości będąca efektem ograniczenia przypadków wprowadzania wielu zmian w regionach możliwych do skorygowania małą liczbą zmian jest zapewniania przez strategię oceny, preferującej ścieżki o małej liczbie zmian (podrozdział 5.4.2).

Redukcja liczebności k-merów Wykorzystanie liczebności k-merów w ocenie ścieżek korekcji, poza zaletą wyboru k-merów pewniejszych, niesie za sobą ryzyko negatywnego wpływu k-merów uzyskanych z obszarów o wysokim pokryciu lub obszarów o sekwencjach wielokrotnie powtarzających się w genomie. Przykładowo k-mer κ_1 o $\eta(\kappa_1) = 20$, który, ze względu na wysoką liczebność z bardzo dużą dozą pewności reprezentuje prawidłowe podsłowo odczytu, może błędnie zostać zmodyfikowany do postaci k-meru κ_2 o podobnej sekwencji oraz $\eta(\kappa_2) = 255$ (tj. liczebności nasyconej, czyli oznaczającej wystąpienie k-meru w odczytach 255 lub więcej razy).

W celu ograniczenia tego zjawiska podjęto próbę redukowania wysokich liczebności k-merów. Jako granicę przyjęto dwukrotność głębokości sekwencjonowania:

$$\eta'(\kappa) = \begin{cases} \eta(\kappa), \text{ gdy } \eta(\kappa) < 2 \cdot \mathrm{dep}\,, \\ f(\eta(\kappa)) \text{ w przeciwnym razie.} \end{cases} \tag{5.26}$$

W równaniu 5.26 wartość $\eta(\kappa)$ oznacza rzeczywistą liczebność k-meru, $\eta'(\kappa)$ zredukowaną liczebność, wykorzystywaną w celu wyznaczenia oceny ścieżki, a $f : \mathbb{N} \to \mathbb{R}_+$ pewną funkcję. Przyjęto, że $f(\eta) = \log_y(\eta - 2 \cdot \mathrm{dep})$, przy czym dobrano próbny zestaw malejących podstaw logarytmu od $y = 4$ do $y = 1{,}05$. Zmniejszanie wartości y miało na celu sprawdzenie różnych poziomów redukcji — wartość $y = 4$ umożliwia silną redukcję, ponieważ funkcja logarytmiczna o takiej podstawie rośnie wolno, z kolei coraz niższe wartości y powodują coraz szybszy wzrost

funkcji, a tym samym coraz mniejszą różnicę między oryginalnymi liczebnościami $\eta(\kappa)$, a zredukowanymi $\eta'(\kappa)$. Wybór wartości granicznej równej $2 \cdot \mathrm{dep}$ jest efektem faktu, że w zakresie liczebności k-merów $[\eta_{\mathrm{cut}}, 2 \cdot \mathrm{dep}]$ występuje najwięcej k-merów prawidłowych i ich pojawienie się w odczytach wyjściowych należy uznać za pożądane. Przedstawione rozwiązanie nie spełniło jednak swojej roli, wywierając negatywny wpływ na jakość wyników korekcji.

5.4.3 Przebieg algorytmu RECKONER

Pomocnicze funkcje, wykorzystane w dalszej części w pseudokodach algorytmu, zostały przedstawione w formie pseudokodów 13 oraz 14. Dodatkowo wprowadzono oznaczenie $\ominus$ symbolizujące, że podczas modyfikacji całej krotki dany jej element nie ulega zmianie, oraz przyjęto, że symbolem pionowych kresek oznaczana jest liczba elementów ciągu, analogicznie do mocy zbioru.

Przebieg algorytmu na wysokim poziomie abstrakcji został przedstawiony na rys. 5.5. Dokładny opis działania zasadniczej części został przedstawiony w kolejnych podrozdziałach. Bloki narysowane przerywaną linią są wykonywane warunkowo. W pierwszej kolejności następuje określenie skali przyporządkowania współczynników jakości. Następnie, w przypadku nieokreślenia przez użytkownika długości oligomeru k, zostaje on wyznaczony. Zadanie to wymaga znajomości przybliżonej długości genomu $\widehat{\ell}_G$, przy czym w przypadku nieokreślenia także tego parametru, zostaje on oszacowany w oparciu o histogram zliczonych *ad hoc* 25-merów. Uzyskana długość oligomeru jest podstawą do przeprowadzania zliczania k-merów oraz — w zależności od wybranego przez użytkownika trybu pracy — zliczania k''-merów. Następnie, opierając się na histogramie k-merów, następuje wyznaczenie progu obcięcia spektrum k-merów η_{cut}. Ostatecznie k-mery posiadające liczebności mniejsze niż próg są usuwane ze spektrum w etapie filtracji, przeprowadzanym przy pomocy narzędzia KMC tools.

W zasadniczej części algorytmu w sposób równoległy następuje odczyt plików wejściowych, których podzbiory odczytów są kolejkowane przez wątek nadrzędny (odczytujący) p_{r} oraz, za pośrednictwem wątków podrzędnych p_i, $i \in \{1, 2, \ldots, \rho_{\mathrm{c}}\}$, następuje detekcja błędnych regionów tych odczytów oraz ich korekcja wraz z kompresją wyjściowych odczytów. Rezultaty są kolejkowane w formie tymczasowych plików dyskowych oraz — przy zachowaniu kolejności odczytów — łączone w pliki wyjściowe w wątku p_{w}. Pozostałe wymienione etapy, jeżeli ich wykonanie nie zostało oddelegowane do zewnętrznych narzędzi KMC i KMC tools, są wykonywane przez wątek główny procesu p_{m}.

Parametry algorytmu

W tabeli 5.4 przedstawiono wewnętrzne stałe algorytmu. Z kolei w tabeli 5.5 przedstawiono parametry, które mogą być zadane przez użytkownika, a w tabe-

Pseudokod 13 RECKONER — funkcje pomocnicze

Require: $\mathfrak{s}_i = (\eta, c, \text{ins}, \text{del}, \text{exp\_ind})$
function QUAL($\mathfrak{s}_i$)
    return η
end function

Require: $\mathfrak{s}_i = (\eta, c, \text{ins}, \text{del}, \text{exp\_ind})$
function MOD($\mathfrak{s}_i$)
    return c
end function

Require: $\mathfrak{s}_i = (\eta, c, \text{ins}, \text{del}, \text{exp\_ind})$
function INS($\mathfrak{s}_i$)
    return ins
end function

Require: $\mathfrak{s}_i = (\eta, c, \text{ins}, \text{del}, \text{exp\_ind})$
function DEL($\mathfrak{s}_i$)
    return del
end function

Require: $\mathfrak{s}_i = (\eta, c, \text{ins}, \text{del}, \text{exp\_ind})$
function EXP_INDEL($\mathfrak{s}_i$)
    return exp_ind
end function

Require: $0 \leqslant a \leqslant \ell - k''$
function $\kappa''(\mathfrak{w}, a)$
    return $\mathfrak{w}[a, a + k'' - 1]$
end function

function TRUSTEDLONG(κ'')
    if $\kappa'' \in \mathcal{K}''_1$ **then**
        return true
    else
        return false
    end if
end function

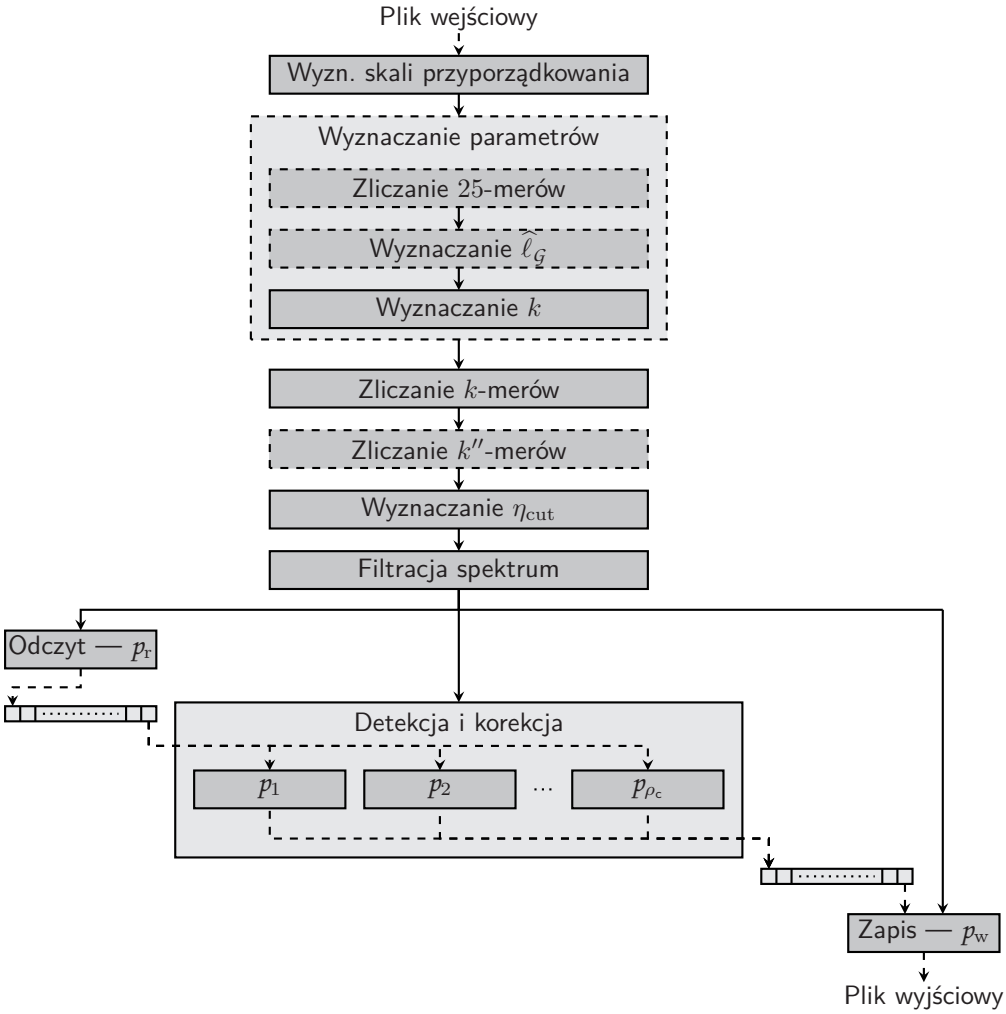
Require: $\pi_i = (a, c, \text{ins}, \text{del})$ ▷ Jeden element ścieżki
function POS(π_i)
    return a
end function

Pseudokod 14 RECKONER — funkcje pomocnicze, cd.

```
29: function CheckLong(𝔯*)
30:     for a ← 0 to |𝔯*| − k'' − 1 do
31:         κ'' ← κ''(𝔯, a)
32:         if ¬TrustedLong(κ'') then
33:             return false
34:         end if
35:     end for
36:     return true
37: end function

38: function Apply(𝔴, π)
39:     ℓ = |𝔴|
40:     for i ← 0 to |π| − 1 do
41:         (a, c, ins, del) ← π
42:         if ins then
43:             𝔴 ← 𝔴[0, a − 1]𝔴[a + 1, ℓ − 1]          ▷ Usunięcie symbolu
44:         else if del then
45:             𝔴 ← 𝔴[0, a − 1]c𝔴[a, ℓ − 1]             ▷ Wstawienie symbolu
46:         else                                          ▷ Substytucja
47:             𝔴[a] ← c
48:         end if
49:         ℓ = |𝔴|
50:     end for
51:     return 𝔴
52: end function

53: function ApplyFinal(𝔴, 𝔮, π)
54:     ℓ = |𝔴|
55:     for i ← 0 to |π| − 1 do
56:         (a, c, ins, del) ← π
57:         if ins then
58:             𝔴 ← 𝔴[0, a − 1]𝔴[a + 1, ℓ − 1]          ▷ Usunięcie symbolu
59:             𝔮 ← 𝔮[0, a − 1]𝔮[a + 1, ℓ − 1]
60:         else if del then
61:             𝔴 ← 𝔴[0, a − 1]c𝔴[a, ℓ − 1]             ▷ Wstawienie symbolu
62:             𝔮 ← 𝔮[0, a − 1]2𝔮[a, ℓ − 1]▷ Wstawienie współczynnika o wartości 2
63:         else                                          ▷ Substytucja
64:             𝔴[a] ← c
65:         end if
66:         ℓ = |𝔴|
67:     end for
68:     return (𝔴, 𝔮)
69: end function
```



Rysunek 5.5: Etapy pracy algorytmu RECKONER

li 5.6 zamieszczono wybór wewnętrznych zmiennych algorytmu, wykorzystanych w pseudokodach. Część spośród tych pierwszych została omówiona w podrozdziale 5.4.2.

Wartości parametrów zostały dobrane eksperymentalnie, poprzez osiągnięcie kompromisu między czasem obliczeń a jakością wyników. Kompromis ten ma charakter dalece nieformalny ze względu na kombinatorycznie znaczną liczbę możliwości, a także niejednoznaczny wpływ zmian strojących na oceny rezultatów uzyskiwane różnymi metodami oraz w oparciu o różne dane.

Do szczególnie istotnych należą parametry detekcji oraz korekcji błędów typu indel. Wartość parametru $\theta_{\mathrm{IND\_PROB}} = 0{,}001$ jest zgodna z przybliżonym prawdopodobieństwem wystąpienia błędu typu indel na danej pozycji odczytu Illumina. Parametr $\theta_{\mathrm{INS\_RATE}} = 10^{-6}$ został wprowadzony na potrzeby regionów, w których podjęto próbę korekcji wyłącznie błędów typu insercja, ponieważ korekcja

taka nie daje możliwości uzyskania innych informacji oceniających — po wprowadzeniu korekcji region (zgodnie z wcześniejszymi definicjami) nie jest pokrywany żadnym k-merem posiadającym liczebność, z kolei wartość współczynnika jakości przyporządkowanego symbolowi będącemu efektem błędu typu insercja według wiedzy autora nie niesie użytecznej informacji. Tym samym jako ocena ścieżki korekcji zawierającej poprawkę takiego błędu przyjmowana jest wartość tego parametru. Jest ona niewielka, dzięki czemu preferowane są ścieżki korekcji zawierające inne rozwiązania. Pozostałe parametry związane z błędami typu indel zostały dobrane w oparciu o intuicyjne przesłanki oraz wstępną weryfikację eksperymentalną.

Podstawowym zewnętrznym parametrem algorytmu jest długość oligomeru. Nieokreślenie jego wartości przez użytkownika powoduje wykonanie oszacowana zgodnie z regułą, która została przedstawiona w podrozdziale 5.4.2.

Wyznaczanie regionów

W poniższym opisie przyjęto te same definicje regionu błędnego i prawidłowego co w opisie algorytmu BLESS. Również w algorytmie RECKONER przyjęto strategię wyznaczania zbioru regionów prawidłowych Ψ_{trust}, w oparciu o który wyznaczany jest zbiór regionów błędnych Ψ_{untrust}. Procedura wyznaczania regionów prawidłowych została przedstawiona na pseudokodach 49–53 w dodatku A.

W pierwszej kolejności (krok 0.0) następuje określenie przynależności każdego k-meru do spektrum w sposób analogiczny jak w algorytmie BLESS (linie 2–17). Następnie uzyskane regiony podlegają skróceniu lub eliminacji (kroki 0.2, 0.3, 0.5, 0.7 — linie odpowiednio 78–82, 83–90, 91–107, 144–150), zgodnie z opisem w podrozdziale 5.3.2. Dodatkowo w schemacie detekcji zostały wprowadzone przedstawione niżej zmiany i uzupełnienia.

Krok 0.1 (linie 18–77) jest przeprowadzany zgodnie z koncepcją wykorzystaną w algorytmie BLESS w wersji 1.02. Regiony prawidłowe są analizowane pod kątem obecności symboli o niskich współczynnikach jakości (o mniejszych wartościach niż $\theta_{\text{Q\_LOW}} = 3$). W przypadku ich wykrycia, region jest dzielony w taki sposób, aby pozycje tych symboli znalazły się w regionach błędnych, a nowe, krótsze regiony prawidłowe posiadały długość co najmniej k.

Krok 0.4, w algorytmie BLESS (w wersji 0.12) wydłużający regiony błędne ze względu na ewentualne fałszywie pozytywne odpowiedzi filtru Blooma, nie został wykorzystany ze względu na rezygnację z zastosowania tej struktury danych.

Krok 0.5 jest przeprowadzany zgodnie z opisem zawartym w podrozdziale 5.3.2, ale po uzupełnieniu brakującego przypadku skracania od strony końca $5'$ prawidłowych regionów innych niż pierwszy, gdy ten koniec regionu zawiera symbole o niskich współczynnikach jakości.

W kroku 0.7 jako minimalna długość regionu błędnego przyjmowana jest wartość $k - 1$ zamiast k w celu umożliwienia detekcji pojedynczych błędów typu de-

Tabela 5.4: Wewnętrzne stałe algorytmu RECKONER

Parametr	Wartość	Znaczenie
		Limity
$\eta_{\text{MX\_CUT}}$	5	Maksymalny próg obcięcia
$\theta_{\text{MX\_CHCK}}$	10000	Maksymalna liczba zmian weryfikowanych w regionie
$\theta_{\text{MX\_FIRST\_PTHS}}$	30	Maksymalna liczba ścieżek uzyskanych w korekcji siłowej pierwszego k-meru
$\theta_{\text{MX\_FIRST\_RES}}$	5	Maksymalna liczba ścieżek uzyskanych podczas korekcji pierwszego k-meru
$\theta_{\text{MX\_LQ}}$	4	Maksymalna liczba symboli w korekcji pierwszego k-meru metodą siłową
$\theta_{\text{MX\_NON\_UNTRST}}$	0,05	Maksymalna część k-merów regionu, które po korekcji pozostaną niezaufane
$\theta_{\text{MX\_PTH}}$	100	Maksymalna liczba ścieżek korekcji regionu
		Związane z błędami typu indel
$\theta_{\text{IND\_PROB}}$	0,001	Przyjęte prawdopodobieństwo błędu typu indel
$\theta_{\text{INS\_RATE}}$	10^{-6}	Ocena ścieżki korekcji regionu, zawierającej wyłącznie eliminację błędów insercji
$\theta_{\text{MN\_IND\_DIST}}$	5	Minimalna odległość między korekcjami błędu typu insercja i typu delecja
$\theta_{\text{MX\_CHCK\_IND\_POS}}$	6	Liczba pozycji odczytu sprawdzanych w celu znalezienia $\theta_{\text{MX\_CHCK\_IND\_UNTRST}}$ korekcji
$\theta_{\text{MX\_CHCK\_IND\_UNTRST}}$	3	Liczba skorygowanych błędów typu substytucja, wskazujących błąd typu indel
$\theta_{\text{MX\_IND}}$	4	Maksymalna liczba błędów typu indel skorygowanych w regionie
$\theta_{\text{MX\_IND\_RATE}}$	0,1	Maksymalny stosunek liczby błędów typu indel skorygowanych w regionie do długości tego regionu
		Pozostałe
$\theta_{\text{COV\_WEIGHT}}$	1	Waga k-meru pokrywającego region
$\theta_{\text{EXT\_WEIGHT}}$	0,5	Waga pierwszego k-meru rozszerzającego
$\theta_{\text{MN\_ERR}}$	2	Minimalna długość błędnego regionu
$\theta_{\text{MN\_SOLID}}$	2	Minimalna długość prawidłowego regionu
$\theta_{\text{MX\_ADJ}}$	4	Maksymalna liczba pozycji regionu sprawdzanych na obecność symboli niskiej jakości
$\theta_{\text{MX\_LQ}}$	3	Maksymalna liczba pozycji dla metody siłowej korekcji pierwszego k-meru
$\theta_{\text{Q\_LOW}}$	3	Górne ograniczenie wartości niskich współczynników jakości

Tabela 5.5: Wybór zewnętrznych parametrów algorytmu RECKONER

Zmienna	Domyślna wartość	Znaczenie
k	—	Długość oligomeru
$\widehat{\ell}_{\mathcal{G}}$	Wyznaczana w oparciu o histogram 25-merów	Szacunkowa długość genomu
$\delta_{\max}$	4	Maksymalne zapotrzebowanie na pamięć operacyjną algorytmu KMC [GiB]
$\theta_{\text{LONG\_RATIO}}$	1,5	Iloraz długości k''-meru do długości k-meru
$\theta_{\text{MX\_E}}$	3	Maksymalna liczba pozycji rozszerzających region
ρ_{c}	Liczba rdzeni logicznych procesora	Liczba wątków korygujących
long	**false**	Binarny parametr wyznaczający weryfikację regionów k''-merami

lecja; sytuacja ta zostanie omówiona wraz z opisem korekcji regionów wewnętrznych.

Korekcja regionów

Sposób wywołania funkcji korekcji poszczególnych regionów, a także weryfikacji ścieżek korekcji przy pomocy k''-merów zostały przedstawione na pseudokodach 15–17. Rozróżniono trzy przypadki: korekcja początkowego regionu skrajnego (linia 18), końcowego regionu skrajnego (linia 24) oraz regionów wewnętrznych (linia 8). Ostatni z nich obejmuje wyróżnioną sytuację, gdy liczba błędnych k-merów regionu wynosi $k-1$ (linia 7), a ma ona miejsce, gdy region jest wyznaczony wskutek wystąpienia pojedynczego błędu delecji.

Ze względu na możliwość przeprowadzenia weryfikacji korekcji przy pomocy k''-merów, wszystkie uzyskane ścieżki korekcji dla wszystkich regionów są zapamiętywane w formie ciągu ($\mathscr{P}$), którego poszczególnymi elementami są ciągi zawierające ścieżki korekcji kolejnych błędnych regionów. Fakt tymczasowej reprezentacji ścieżek w formie ciągów oznaczonych $\boldsymbol{\pi}$ (zamiast zbiorów $\boldsymbol{\Pi}$) wynika z konieczności zachowania kolejności wprowadzania zmian w odczycie, będącej efektem korekcji błędów typu indel.

Po przeprowadzeniu generacji ścieżek następuje ich zastosowanie (linia 35), a w przypadku pracy w trybie weryfikacji k''-merów, także wyodrębnienie k''-merów oraz weryfikacja ich obecności w ${\mathcal{K}''}_1$. Weryfikacja odbywa się zgodnie z procedurą od linii 67. Ciągi ścieżek poszczególnych regionów są niezależnie sortowane w kolejności malejących ocen, a następnie pierwsze (najlepsze) ścieżki poszczególnych regionów są wprowadzane do odczytu (linia 74). Jeżeli odczyt o takiej po-

Tabela 5.6: Wybór wewnętrznych zmiennych algorytmu RECKONER

Zmienna	Znaczenie
ℓ_Δ	zmiana długości odczytu
$\mathfrak{s}_i$	i-ty element ciągu korekcji
$\mathbf{v}_{\text{Q\_LOW},i}$	i-ty element ciągu pozycji odczytu o niskich współczynnikach jakości
$\mathscr{P}$	ciąg ciągów ścieżek korekcji kolejnych regionów odczytu
Q_r	kolejka fragmentów pliku wejściowego
Q_w	kolejka fragmentów pliku wyjściowego
$\eta_{\text{max\_ext}}$	Największa liczebność pierwszego k-meru rozszerzającego region
θ_{ext}	Liczba pozycji rozszerzających region dla bieżącej ścieżki
$\theta_{\text{first\_mod}}$	Najmniejszy indeks modyfikacji w ścieżce
$\theta_{\text{last\_chck}}$	Pozycja ostatniego k-meru rozszerzającego region wewnętrzny
$\theta_{\text{last\_kmer}}$	Pozycja ostatniego k-meru w regionie podczas rozszerzania
$\theta_{\text{last\_mod}}$	Największy indeks modyfikacji w ścieżce
θ_{lq}	Liczba pozycji dla algorytmu siłowego korekcji pierwszego k-meru
$\theta_{\text{mx\_chck}}$	Pozostała liczba zmian do zweryfikowania w regionie
$\theta_{\text{mx\_ind}}$	Pozostała liczba dopuszczalnych korekcji błędów typu indel w regionie
$\theta_{\text{mx\_untrst}}$	Maksymalna liczba pozostałych dopuszczalnych błędnych k-merów w regionie lub k-merów rozszerzających
$\theta_{\text{reg\_beg}}$	Indeks początku regionu
π	Ścieżka korekcji (ciąg czwórek uporządkowanych)
ς_r	Semafor odczytu pliku
ς_w	Semafor zapisu pliku
$\boldsymbol{\pi}$	Ciąg ścieżek korekcji (ciąg ciągów czwórek uporządkowanych)
ψ	Region odczytu
ψ_p	Poprzedni region odczytu
$\boldsymbol{\Pi}$	Zbiór ścieżek korekcji
Ψ_{trust}	Zbiór prawidłowych regionów odczytu
Ψ_{untrust}	Zbiór błędnych regionów odczytu
del	Znacznik delecji w ciągu korekcji
exp_ind	Znacznik potencjalnego błędu typu indel w ciągu korekcji
fin_corr	Znacznik zakończenia korekcji odczytów pliku
fin_input	Znacznik zakończenia wczytania pliku
ins	Znacznik insercji w korekcji
sh_ns	Wartość binarna dotycząca obecności krótkiego błędnego regionu

Pseudokod 15 Korekcja regionów algorytmu RECKONER

Wejście: $\Psi_{\text{trust}}, \text{sh\_ns}, k, \mathfrak{r}, \ell = |\mathfrak{r}|$
Wejście: long ▷ Znacznik wskazujący tryb z weryfikacją w oparciu o k''-mery
Wyjście: $\mathfrak{r}$ oraz ℓ poddane modyfikacji

1: **if** $\Psi_{\text{trust}} \neq \emptyset \wedge \neg\text{sh\_ns}$ **then**
2: $\mathscr{P} \leftarrow ()$ ▷ Ciąg ciągów ścieżek korekcji kolejnych regionów

▷ Krok 1.1 — korekcja regionów wewnętrznych
3: **if** $|\Psi_{\text{trust}}| > 1$ **then**
4: **for** $\psi_{\text{trust}} \in (\Psi_{\text{trust}} \setminus \textsc{MinReg}(\Psi_{\text{trust}}))$ **do**
5: $\psi_{\text{p}} \leftarrow \textsc{Pred}(\Psi_{\text{trust}}, \psi_{\text{trust}})$
6: $\psi \leftarrow \textsc{UntrustedReg}(\psi_{\text{p}}, \psi_{\text{trust}})$ ▷ $\psi \in \Psi_{\text{untrust}}$, wewnętrzny
7: **if** $|\psi| \geqslant k - 1$ **then** ▷ Odjęcie 1 na potrzeby regionów z pojedynczą delecją
8: $\boldsymbol{\Pi} \leftarrow \textsc{CorrectInternal}(\textsc{Beg}(\psi), \textsc{End}(\psi) - k + 1)$
9: Wstaw do ciągu $(\boldsymbol{\pi})$ wszystkie elementy zbioru $\boldsymbol{\Pi}$
10: $\mathscr{P} \leftarrow \boldsymbol{\pi}\mathscr{P}$ ▷ Dołącz ciąg $(\boldsymbol{\pi})$ jako element ciągu $(\mathscr{P})$
11: **end if**
12: **end for**
13: **end if**
14: $\psi_{\min} \leftarrow \textsc{MinReg}(\Psi_{\text{trust}})$
15: $\psi_{\max} \leftarrow \textsc{MaxReg}(\Psi_{\text{trust}})$

▷ Krok 1.2 — korekcja regionu skrajnego początkowego
16: **if** $\textsc{Beg}(\psi_{\min}) > 0$ **then**
17: $\psi \leftarrow (0, \textsc{Beg}(\psi_{\min} - 1))$ ▷ $\psi \in \Psi_{\text{untrust}}$, skrajny początkowy
18: $\boldsymbol{\Pi} \leftarrow \textsc{Correct5}'(\psi)$
19: Wstaw do ciągu $(\boldsymbol{\pi})$ wszystkie elementy zbioru $\boldsymbol{\Pi}$
20: $\mathscr{P} \leftarrow \mathscr{P}\boldsymbol{\pi}$
21: **end if**

▷ Krok 1.3 — korekcja regionu skrajnego końcowego
22: **if** $\textsc{End}(\psi_{\max}) < \ell - k$ **then**
23: $\psi \leftarrow (\textsc{End}(\psi_{\max}) + 1, \ell - k)$ ▷ $\psi \in \Psi_{\text{untrust}}$, skrajny końcowy
24: $\boldsymbol{\Pi} \leftarrow \textsc{Correct3}'(\psi)$
25: Wstaw do ciągu $(\boldsymbol{\pi})$ wszystkie elementy zbioru $\boldsymbol{\Pi}$
26: $\mathscr{P} \leftarrow \mathscr{P}\boldsymbol{\pi}$
27: **end if**

Pseudokod 16 Korekcja regionów algorytmu RECKONER, cd.

```
28:     if |𝒫| > 0 then                          ▷ Uzyskano co najmniej jedną ścieżkę
29:         if long then
30:             VerifyLongKmers(𝒫)
31:         else
32:             n_reg ← |𝒫|
33:             for i ← n_reg − 1 downto 0 do
34:                 Wybierz ścieżkę π o najwyższej wartości Rate(π) spośród 𝒫_i
35:                 𝔯 ← Apply(𝔯, π)
36:             end for
37:             ℓ ← |𝔯|
38:         end if
39:     end if
40: else
                                                          ▷ Kroki 2.1, 2.2, 2.3
41:     Π ← CorrectFirst
42:     Wstaw do ciągu (π) wszystkie elementy zbioru Π
43:     if long then
44:         VerifyLongKmersFirstKmer(π)
45:     else
46:         Wybierz ścieżkę π o najwyższej wartości Rate(π) spośród elementów
    ciągu (π)
47:         𝔯 ← Apply(𝔯, π)
48:         ℓ ← |𝔯|
49:     end if
50: end if

51: procedure VerifyLongKmersFirstKmer(π)
52:     if |π| ≠ 0 then
53:         Sortuj elementy π ciągu (π) w kolejności malejących wartości Rate(π)
54:         i ← 0
55:         verified ← false                  ▷ Znaleziono ścieżkę pokrytą k″-merami
56:         repeat
57:             𝔯* ←Apply(𝔯, π_i)
58:             verified ← CheckLong(𝔯*)
59:             i ← i + 1
60:         until verified ∨ i ⩾ |π|
61:         if verified then
62:             𝔯 ←Apply(𝔯, π_{i−1})
63:             ℓ ← |𝔯|
64:         end if
65:     end if
66: end procedure
```

Pseudokod 17 Korekcja regionów algorytmu RECKONER, cd. 2

```
67: procedure VerifyLongKmers(𝒫)
68:     n_reg ← |𝒫|
69:     Sortuj każdy ciąg (𝒫_0), (𝒫_1), ..., (𝒫_{n_reg−1}) w kolejności malejących wartości Rate(π) ich elementów π
70:     ι_0, ι_1, ..., ι_{n_reg−1} ← 0        ▷ Indeksy bieżących ścieżek kolejnych regionów
71:     𝔯* ← 𝔯
72:     for i ← n_reg − 1 downto 0 do
73:         π ← 𝒫_i
74:         𝔯* ←Apply(𝔯*, π_0)▷ Zastosowanie pierwszej ścieżki z każdego regionu
75:     end for
76:     found ← true                          ▷ Czy jest nowa ścieżka do weryfikacji?
77:     while found ∧ ¬CheckLong(𝔯*) do   ▷ Leniwe wartościowanie warunku
78:         found ← false
79:         rate_MINΔ ← +∞
              ▷ Poszukiwanie regionu z którego zostanie wybrana słabsza ścieżka
80:         for i ← n_reg − 1 downto 0 do
81:             π ← 𝒫_i
82:             if |π| > ι_i then
83:                 rate_Δ ← rate(π_{ι_i}) − rate(π_{ι_i−1})
84:                 if rate_Δ < rate_MINΔ then
85:                     found ← true
86:                     rate_MINΔ ← rate_Δ
87:                     j ← i ▷ Region z którego zostanie wybrana słabsza ścieżka
88:                 end if
89:             end if
90:         end for
91:         if found then
92:             ι_j ← ι_j + 1
93:             𝔯* ← 𝔯
94:             for i ← n_reg − 1 downto 0 do
95:                 π ← 𝒫_i
96:                 𝔯* ←Apply(𝔯*, π_{ι_i})
97:             end for
98:         end if
99:     end while
100:    if found then                                  ▷ Zweryfikowano
101:        for i ← n_reg − 1 downto 0 do
102:            π ← 𝒫_i
103:            (𝔯, 𝔮) ←ApplyFinal(𝔯, 𝔮, π_{ι_i})
104:        end for
105:        ℓ ← |𝔯|
106:    end if
107: end procedure
```

staci jest pokryty zaufanymi k''-merami (linia 77), wówczas następuje ostateczne zastosowanie wybranych ścieżek (linia 103). W przeciwnym wypadku następuje wyszukiwanie takiego regionu, dla którego wybranie słabszej (kolejnej) ścieżki spowoduje najmniejszą redukcję sumy ocen ścieżek całego odczytu (linia 80). Po zmianie tej ścieżki następuje ponowne wprowadzenie zmian (linia 96) oraz kolejna weryfikacja pokrycia zaufanymi k''-merami, ale po wyborze znalezionej słabszej ścieżki jednego z regionów.

Wywołanie korekcji pierwszego k-meru wraz z weryfikacją zostało przedstawione od linii 40. W tym przypadku, po uzyskaniu zestawu ścieżek korekcji pierwszego k-meru oraz pozostałej części odczytu (linia 41) następuje ich weryfikacja poprzez wybór ścieżki o najwyższej ocenie, której zastosowanie w odczycie pozwala na pokrycie go zaufanymi k''-merami (procedura od linii 51).

Wprowadzanie zmian ścieżek (np. w pętli od linii 33) odbywa się w kolejności od regionu o najwyższym indeksie, co zapewnia łatwe wyznaczenie prawidłowych indeksów przy stosowaniu zmian, uwzględniając ewentualną zmianę długości odczytu.

Sposób wyznaczania oceny został przedstawiony w formie funkcji `Rate` na pseudokodzie 18, zgodnie z przedstawioną wyżej strategią doboru ostatecznych ścieżek korekcji. Pseudokod tej funkcji przedstawia uproszczoną implementację równań (5.19) oraz (5.20). Niezbędne wartości są zapamiętywane wraz z tworzeniem ścieżek korekcji, co zostanie wyjaśnione w dalszej części opisu.

Algorytm z powrotami Przeszukiwanie przestrzeni rozwiązań przy pomocy algorytmu z powrotami obejmuje nie tylko przypadki, gdy tworzenie kolejnych rozwiązań częściowych następuje w efekcie błędów substytucji wykrywanych w oparciu o nieobecność odpowiednich k-merów w spektrum, ale także podejmowania prób korekcji symboli niskiej jakości, zachowywania oryginalnych symboli nawet bez pokrycia prawidłowymi k-merami, a w szczególności dodawania kolejnych ciągów rozwiązań częściowych w momencie występowania potencjalnych błędów typu indel.

Działanie algorytmu zostanie wyjaśnione dla przypadku korekcji w kierunku $3'$, przedstawionego na pseudokodach 19–21. Sytuacja podobna, ale obejmująca korekcję początkowych regionów skrajnych (w kierunku $5'$) została przedstawiona na pseudokodach 54–56 w dodatku A. W celu poprawy czytelności, ze względu na znaczną liczbę parametrów funkcji rekursywnych, zdefiniowano wartości „globalne”, wspólne dla wszystkich ich wywołań.

W pierwszej kolejności, począwszy od linii 5, następuje inicjalizacja ograniczeń. Korekcja zaczyna się od modyfikacji symbolu końcowego pierwszego k-meru regionu (linia 15) do momentu, aż k-mer ten stanie się k-merem prawidłowym. Po osiągnięciu sukcesu następuje wykonanie procedury korekcji dla kolejnego k-meru (linia 24) wcześniej zmian. Na końcu następuje podjęcie próby korekcji błędu typu

Pseudokod 18 Ocena ścieżki korekcji algorytmu RECKONER

```
1:  function PathProb(π)
2:      return Iloczyn prawdopodobieństw zmodyfikowanych symboli przyporządkowany ścieżce π
3:  end function

4:  function PathWghtSum(π)
5:      return Suma wag pokrywających k-merów przyporządkowana ścieżce π
6:  end function

7:  function PathQual(π)
8:      return Suma ważonych liczebności pokrywających k-merów przyporządkowana ścieżce π
9:  end function

10: function Rate(π)
11:     n_ins ← liczba insercji w π
12:     rate ← 0
13:     if n_ins > 0 ∧ n_ins = |π| then
14:         rate ← θ_INS_RATE
15:     else
16:         rate ← PathQual(π)·PathProb(π) / PathWghtSum(π)
17:     end if
18:     return rate
19: end function
```

indel (linia 28), której przebieg zostanie omówiony w dalszej części opisu.

Zmiany budujące daną ścieżkę korekcji są umieszczane w ciągu (linia 2), którego kolejne pozycje odpowiadają kolejnym zmianom korygującym dany region. Ze względu na obecność również zmian korygujących błędy typu indel, pozycje te nie są bezpośrednio przyporządkowane pozycjom regionu. W każdym wywołaniu funkcji korygującej ciąg korekcji zostaje zmodyfikowany, przy czym typ błędu warunkuje rodzaj tych zmian: błędy substytucji wymagają zapamiętania symbolu oraz liczebności k-meru (linia 17).

Korekcja kolejnych k-merów obejmuje więcej przypadków analizy. W razie nieosiągnięcia końca regionu, czynność może być powtarzana rekursywnie (linie 45, 73 oraz 94). W pierwszej kolejności następuje weryfikacja, czy dana pozycja jest pozycją o niskiej jakości (linia 33), co ma wpływ na określenie bieżącej sytuacji. Sytuacja 1 obejmuje przypadek, gdy nie jest wymagane wprowadzenia zmian do symbolu, żeby bieżący k-mer pozostawał zaufany, a jednocześnie nie jest to symbol o niskiej jakości (linie od 34). Przeciwny warunek dotyczy sytuacji 2 (od linii 47), a obejmuje przeprowadzenie korekcji błędu substytucji (linia 58). Rezultat może być dwojakiego rodzaju: zostanie znaleziony prawidłowy k-mer

(sytuacja 2.1) albo nie zostanie znaleziony (sytuacja 2.2). W drugim przypadku, jeżeli sytuacja nie zostanie wyeliminowana na skutek przekroczenia limitu (linia 80), w ciągu jest umieszcza informacja, że liczebność danego k-meru wynosi 0 (linia 82), a pozostała część korekcji przebiega analogicznie jak w sytuacji 2.1.

We wszystkich sytuacjach: 1, 2.1 oraz 2.2, w przypadku osiągnięcia końca regionu następuje utworzenie ścieżki oraz próba rozszerzania (linie 40, 66 oraz 87). W przypadku niepowodzenia następuje detekcja błędów typu indel (linie 68 oraz 89). Detekcja ta jest przeprowadzana także na bieżąco po wprowadzenia korekcji przed osiągnięciem końca regionu (linie 74 oraz 95). Osobną czynnością jest korekcja błędów typu indel (linie 28 oraz 76), która jest przeprowadzana wyłącznie po realizacji serii prób zmiany symboli, ponieważ jest to warunek konieczny ewentualnego wykrycia na danej pozycji błędu typu indel.

Detekcja błędów typu indel (ani w konsekwencji ich korekcja) nie jest przeprowadzana w sytuacji 1 ze względu na strategię takiej detekcji, która przewiduje występowanie błędów typu indel na pozycji, dla której została podjęta próba korekcji błędu substytucji i jednocześnie podobna korekcja miała miejsce dla wielu kolejnych pozycji. Natomiast sytuacja 1 dotyczy przypadku, gdy żadna zmiana w odczycie nie nastąpiła, w rezultacie liczba korekcji kolejnych pozycji od miejsca potencjalnego błędu typu indel nie uległa zwiększeniu.

Detekcja i korekcja błędów typu indel Algorytmy detekcji błędów typu indel oraz ich korekcji w kierunku $3'$ zostały przedstawione na pseudokodach 22–24. Sytuacja podobna, ale obejmująca korekcję początkowych regionów skrajnych (w kierunku $5'$) została przedstawiona na pseudokodach 57–59. Idea detekcji polega na sprawdzeniu pewnej spójnej podsekwencji ciągu korekcji w celu wykrycia przypadków korekcji dużej liczby błędów substytucji. Zjawisko to w wielu przypadkach powinno wskazywać na występowanie błędów typu indel, w sposób wyjaśniony w podrozdziale 5.4.2 oraz na rys. 5.3.

Detekcja (procedura od linii 1) może zostać przeprowadzona w dwóch sytuacjach: gdy sprawdzanie odbywa się w momencie osiągnięcia końca regionu (`last`, parametr funkcji `Check3'Indel`, przyjmuje wartość **true**) oraz w sytuacji przeciwnej. Podział ten ma na celu rozróżnienie detekcji w regionach „krótkich”, w których ogólna zasada detekcji może być zbyt restrykcyjna ze względu na zbyt małą możliwą do wykrycia liczbę korekcji substytucji (sytuacja 1), oraz w pozostałych regionach (sytuacja 2).

Detekcja w sytuacji 1 jest przeprowadzana po spełnieniu dwóch warunków (linia 7): gdy jest to ostatnia (w kierunku przeprowadzanej korekcji) pozycja regionu oraz gdy długość regionu jest mniejsza niż przyjęta graniczna liczba błędów substytucji wskazująca na wystąpienia błędu typu indel ($\theta_{\text{MX\_CHCK\_IND\_UNTRST}} = 3$). Jeżeli liczba ta jest większa, wówczas bez względu na fakt, czy jest to koniec regionu, przeprowadzana jest detekcja w sytuacji 2. Trzeci możliwy przypadek,

Pseudokod 19 Korekcja regionu w kierunku 3′ algorytmu RECKONER

Wejście: $k, \ell = |\mathfrak{r}|, \mathfrak{r}, \mathfrak{q}$
Globalne: $\mathfrak{s}, \theta_{\text{reg\_beg}}$
Globalne: $\theta_{\text{mx\_ind}}, \theta_{\text{mx\_untrst}}$ ▷ Liczniki dwukierunkowe (dekrementowane przy zagłębianiu i inkrementowane przy powrotach algorytmu)
Globalne: $\theta_{\text{mx\_chck}}$ ▷ Licznik jednokierunkowy (tylko dekrementowany)

1: **function** CORRECT3′(ψ)
2: $\mathfrak{s} \leftarrow ()$ ▷ Pusty ciąg korekcji
3: $\mathbf{\Pi} \leftarrow \emptyset$
4: $\theta_{\text{reg\_beg}} \leftarrow \text{BEG}(\psi)$
▷ Wyznaczenie limitów
5: $\theta_{\text{mx\_ind}} \leftarrow \min(\theta_{\text{MX\_IND}}, \lceil \theta_{\text{MX\_IND\_RATE}} \cdot |\psi| \rceil)$
6: $\theta_{\text{mx\_untrst}} \leftarrow \lceil \theta_{\text{MX\_NON\_UNTRST}} \cdot |\psi| \rceil$
7: $\theta_{\text{mx\_chck}} \leftarrow \theta_{\text{MX\_CHCK}}$
8: $i \leftarrow 0$ ▷ Pozycja ciągu korekcji
9: $\kappa^* \leftarrow \kappa(\mathfrak{r}, \text{BEG}(\psi))$ ▷ Nowa sekwencja k-meru
10: **for all** $c \in \Sigma$ **do**
11: **if** $\theta_{\text{mx\_chck}} = 0$ **then**
12: **return** $\mathbf{\Pi}$
13: **end if**
14: $\theta_{\text{mx\_chck}} \leftarrow \theta_{\text{mx\_chck}} - 1$
15: $\kappa^*[k-1] \leftarrow c$
16: **if** TRUSTED(κ^*) **then**
17: $\mathfrak{s}_i \leftarrow (\eta(\kappa^*), c, \textbf{false}, \textbf{false}, \ominus)$
18: **if** BEG(ψ) = END(ψ) **then** ▷ Koniec regionu
19: $\pi \leftarrow$ CREATEPATHEXTEND3'(i, ψ)
20: **if** $\pi \neq \varnothing$ **then**
21: $\mathbf{\Pi} \leftarrow \mathbf{\Pi} \cup \{\pi\}$
22: **end if**
23: **else**
24: $\mathbf{\Pi} \leftarrow$ CONTINUE3′($\kappa^*, i+1, (\text{BEG}(\psi)+1, \text{END}(\psi)), \mathbf{\Pi}$)
25: **end if**
26: **end if**
27: **end for**
28: $\mathbf{\Pi} \leftarrow$ CORRECT3′INDEL($\kappa^*, i, \psi, \mathbf{\Pi}$)
29: **return** $\mathbf{\Pi}$
30: **end function**

Pseudokod 20 Korekcja regionu w kierunku 3′ algorytmu RECKONER, cd.

```
31: function CONTINUE3′(κ, i, ψ, Π)
32:     κ* ← κ[1, k − 1]𝔯[BEG(ψ) + k − 1]
33:     low ← 𝔮[BEG(ψ) + k − 1] < θ_Q_LOW        ▷ Czy symbol jest niskiej jakości

                ▷ Sytuacja 1 — k-mer jest zaufany i symbol nie jest niskiej jakości
34:     if ¬low ∧ TRUSTED(κ*) then                ▷ Niczego nie modyfikujemy
35:         𝔰_i ← (η(κ*), κ*[k − 1], false, false, ⊖)
36:         if BEG(ψ) = END(ψ) then
37:             if |Π| > θ_MX_PTH then
38:                 return Π
39:             end if
40:             π ← CREATEPATHEXTEND3'(i, 𝔰, ψ)
41:             if π ≠ ∅ then
42:                 Π ← Π ∪ {π}
43:             end if
44:         else
45:             Π ← CONTINUE3'(κ*, i + 1, (BEG(ψ) + 1, END(ψ)), Π)
46:         end if

            ▷ Sytuacja 2 — k-mer nie jest zaufany lub symbol jest niskiej jakości
47:     else
48:         solid ← false                         ▷ Czy znaleziono zaufany k-mer
49:         Σ′ ← Σ
50:         if ¬low then
51:             Σ′ ← Σ \ {κ*[k − 1]}              ▷ Nie ma sensu sprawdzać oryginalnego
52:         end if
53:         for all c ∈ Σ′ do
54:             if θ_mx_chck = 0 then
55:                 return Π
56:             end if
57:             θ_mx_chck ← θ_mx_chck − 1
58:             κ*[k − 1] ← c
```

Pseudokod 21 Korekcja regionu w kierunku 3′ algorytmu RECKONER, cd. 2

```
                                        ▷ Sytuacja 2.1 — znaleziono zaufany k-mer
59:             if TRUSTED(κ*) then
60:                 solid ← true
61:                 𝔰_i ← (η(κ*), c, false, false, ⊖)
62:                 if BEG(ψ) = END(ψ) then
63:                     if |Π| > θ_MX_PTH then
64:                         return Π
65:                     end if
66:                     π ← CREATEPATHEXTEND3'(i, ψ)
67:                     if π = ∅ then
68:                         CHECK3′INDEL(i, ψ, true)
69:                     else
70:                         Π ← Π ∪ {π}
71:                     end if
72:                 else
73:                     Π ← CONTINUE3′(κ*, i + 1, (BEG(ψ) + 1, END(ψ)), Π)
74:                     CHECK3′INDEL(i, ψ, false)
75:                 end if
76:                 Π ← CORRECT3′INDEL(κ*, i, ψ, Π)
77:             end if
78:         end for

                                        ▷ Sytuacja 2.2 — nie znaleziono zaufanego k-meru
79:         if ¬solid then
80:             if θ_mx_untrst > 0 then
81:                 κ*[k − 1] ← 𝔱[BEG(ψ) + k − 1]
82:                 𝔰_i ← (0, κ*[k − 1], false, false, ⊖)
83:                 if BEG(ψ) = END(ψ) then
84:                     if |Π| > θ_MX_PTH then
85:                         return Π
86:                     end if
87:                     π ← CREATEPATHEXTEND3'(i, 𝔰, ψ)
88:                     if π = ∅ then
89:                         CHECK3′INDEL(i, ψ, true)
90:                     else
91:                         Π ← Π ∪ {π}
92:                     end if
93:                 else
94:                     Π ← CONTINUE3′(κ*, i + 1, (BEG(ψ) + 1, END(ψ)), Π)
95:                     CHECK3′INDEL(i, ψ, false)
96:                 end if
97:             end if
98:         end if
99:     end if
100:    return Π
101: end function
```

Pseudokod 22 Detekcja i korekcja błędów typu indel w kierunku $3'$ algorytmu RECKONER

Globalne: $\mathfrak{s}, \theta_{\text{mx\_ind}}$

```
1:  procedure CHECK3′INDEL(i, ψ, last)
2:      if θ_mx_ind = 0 then
3:          return
4:      end if
5:      θ_n_exp_ind ← Liczba elementów s_0, s_1, ..., s_i, dla których exp_ind = true
6:      curr_pos ← BEG(ψ) + k − 1                    ▷ Pozycja aktualnego symbolu

        ▷ Sytuacja 1 — koniec krótkiego regionu, sprawdzenie wszystkich pozycji
7:      if last ∧ i + 1 < θ_MX_CHCK_IND_UNTRST then
8:          mods ← 0
9:          indel_pos ← 0
10:         for j ← i downto 0 do
11:             if INS(s_j) ∨ DEL(s_j) then
12:                 break          ▷ Przerwanie, ale nie wykluczenie błędu indel
13:             end if
14:             if MOD(s_j) ≠ r[curr_pos − i + j] then          ▷ Była modyfikacja
15:                 mods ← mods + 1
16:                 indel_pos ← j
17:             end if
18:         end for
19:         if mods ⩾ ⌈(θ_MX_CHCK_IND_UNTRST · (i+1)) / θ_MX_CHCK_IND_POS⌉ then
20:             for j ← i downto indel_pos do
21:                 EXP_INDEL(s_j) ← true                ▷ Potencjalny błąd indel
22:             end for
23:         end if
24:         return
25:     end if

                              ▷ Sytuacja 2 — sprawdzanie ograniczonej liczby pozycji
26:     if θ_n_exp_ind > 0 then
27:         return
28:     end if
```

Pseudokod 23 Detekcja i korekcja błędów typu indel w kierunku $3'$ algorytmu RECKONER, cd.

```
29:     if i + 1 ⩾ θ_MX_CHCK_IND_UNTRST then
30:         max_check ← min(i, θ_MX_CHCK_IND_POS)        ▷ Sprawdzanych pozycji
31:         mods ← 0
32:         indel_pos ← 0
33:         for j ← i downto i − max_check do
34:             if INS(𝔰_j) ∨ DEL(𝔰_j) then
35:                 break          ▷ Przerwanie, ale nie wykluczenie błędu indel
36:             end if
37:             if MOD(𝔰_j) ≠ 𝔯[curr_pos − i + j] then
38:                 mods ← mods + 1
39:                 indel_pos ← j
40:             end if
41:         end for
42:         if mods ⩾ θ_MX_CHCK_IND_UNTRST then
43:             EXP_INDEL(𝔰_indel_pos) ← true              ▷ Potencjalny błąd indel
44:         end if
45:     end if
46:     return
47: end procedure

48: function CORRECT3′INDEL(κ, i, ψ, Π)
49:     if ¬EXP_INDEL(𝔰_i) then
50:         return Π
51:     end if
52:     EXP_INDEL(𝔰_i) ← false
53:     if θ_mx_ind = 0 then
54:         return Π
55:     end if
56:     if i > 0 ∧ (INS(𝔰_{i−1}) ∨ DEL(𝔰_{i−1})) then
57:         return Π
58:     end if
59:     θ_mx_ind ← θ_mx_ind − 1
            ▷ Poniższe dwa zakresy obejmują tylko pozycje ciągu, które istnieją
60:     was_del ← wśród 𝔰_{i−θ_MN_IND_DIST}, ..., 𝔰_{i−2}, 𝔰_{i−1} jest taki, że del = true
61:     was_ins ← wśród 𝔰_{i−θ_MN_IND_DIST}, ..., 𝔰_{i−2}, 𝔰_{i−1} jest taki, że ins = true
```

Pseudokod 24 Detekcja i korekcja błędów typu indel w kierunku $3'$ algorytmu RECKONER, cd. 2

```
62:     if ¬was_del then                                          ▷ Korekcja insercji
                                  ▷ Nie sprawdzamy zaufania k-meru ani liczebności
63:         𝔰_i ← (∅, 𝔯[Beg(ψ) + k − 1], true, false, ⊖)
64:         if Beg(ψ) = End(ψ) then
65:             if |Π| > θ_MX_PTH then
66:                 𝔰_i ← (∅, ∅, false, false, ⊖)
67:                 return Π
68:             end if
69:             π ← CreatePathExtend3'(i, 𝔰, ψ)
70:             if π ≠ ∅ then
71:                 Π ← Π ∪ {π}
72:             end if
73:         else
74:             if θ_mx_chck = 0 then
75:                 𝔰_i ← (∅, ∅, false, false, ⊖)
76:                 return Π
77:             end if
78:             θ_mx_chck ← θ_mx_chck − 1
79:             κ* ← ∅κ[0, k − 2]
80:             Π ← Continue3'(κ*, i + 1, (Beg(ψ) + 1, End(ψ)), Π)
81:             𝔰_i ← (∅, ∅, false, false, ⊖)
82:         end if
83:     end if
84:     if ¬was_ins then                                          ▷ Korekcja delecji
85:         κ* ← κ                                          ▷ Nowa sekwencja k-meru
86:         for all c ∈ Σ do
87:             if θ_mx_chck = 0 then
88:                 𝔰_i ← (∅, ∅, false, false, ⊖)
89:                 return Π
90:             end if
91:             θ_mx_chck ← θ_mx_chck − 1
92:             κ*[k − 1] ← c
93:             if Trusted(κ*) then
94:                 𝔰_i ← (η(κ*), c, false, true, ⊖)
95:                 Π ← Continue3'(κ*, i + 1, ψ, Π)          ▷ Nie modyfikujemy ψ
96:             end if
97:         end for
98:         𝔰_i ← (∅, ∅, false, false, ⊖)
99:     end if
100:    θ_mx_ind ← θ_mx_ind + 1
101:    return Π
102: end function
```

gdy detekcja nie jest wykonywana na końcu regionu, a liczba pozycji jest zbyt mała, nie jest obsługiwany, ponieważ wykrycie błędu typu indel w takim regionie będzie mogło nastąpić po ponownym wejściu do funkcji, jednak po uprzednim przeprowadzeniu korekcji większej liczby błędów substytucji.

Proces detekcji w sytuacji 1 obejmuje analizę kolejnych pozycji regionu oraz zliczanie wprowadzanych zmian (linia 15). Znalezienie takiej sytuacji, że w regionie skorygowano już błąd typu indel (linia 11) powoduje przerwanie zliczania (ponieważ wcześniejsze pozycje regionu zostały już przeanalizowane pod kątem obecności takich błędów), ale dopuszczalna jest sytuacja, że wśród przeanalizowanych właśnie pozycji zostanie przeprowadzona korekcja kolejnego błędu tego typu. Ostatecznie, jeżeli liczba modyfikacji osiągnie próg (linia 19) przeskalowany względem długości regionu (iloraz długości ciągu korekcji do domyślnej liczby sprawdzanych pozycji $\theta_{\text{MX\_CHCK\_IND\_POS}} = 6$), przyjmowane jest założenie, że w regionie znajduje się błąd typu indel. W rezultacie następuje oznaczenie wszystkich przeanalizowanych pozycji jako potencjalnie zawierające błędy typu indel (linia 21). Wyznaczenie tak dużej liczby potencjalnych miejsc błędów jest rozwiązaniem zapewniającym wysoką czułość, którego koszt obliczeniowy jest jednak dostatecznie niski ze względu na niewielką długość regionu.

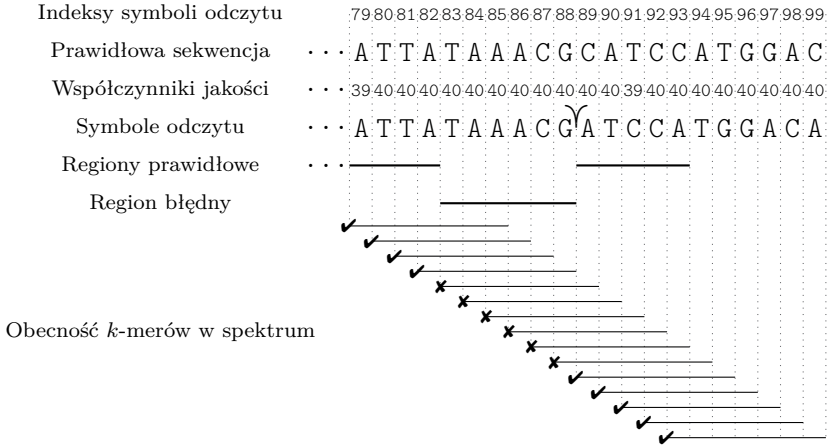
Detekcja w sytuacji 2 (od linii 29) jest przeprowadzana w podobny sposób, jednak po spełnieniu warunku, że w danym momencie żadna pozycja regionu nie jest oznaczona jako miejsce potencjalnego błędu typu indel (linia 26). Przypadek korekcji większej liczby błędów typu indel musi zostać obsłużony dwukrotnie: po wykryciu pierwszego błędu musi nastąpić jego korekcja, a następnie, podczas korekcji kolejnych błędów substytucji tego regionu, ponownie i na innej pozycji błąd typu indel musi zostać wykryty. Jednocześnie liczba skorygowanych błędów typu indel w obu sytuacjach jest limitowana (linia 2).

Zasadniczy proces korekcji (od linii 48), przeprowadzany w formie budowy nowego rozwiązania częściowego po wykonaniu powrotu algorytmu, jest ograniczony kilkoma warunkami wynikającymi z przyjętego modelu błędów oraz dodatkowymi limitami. Do ograniczeń należą limitowanie liczby indeli w regionie (linia 53), weryfikacja, czy nie następuje próba korekcji błędów indel na sąsiednich pozycjach (linia 56) oraz zapewnienie minimalnej odległości między sąsiednimi błędami typu insercja i delecja (linie 60 oraz 61). Sama korekcja jest wykonywana wyłącznie, jeśli uprzednio bieżąca pozycja została oznaczona jako miejsce wystąpienia potencjalnego błędu typu indel (linia 49).

Korekcja błędów typu insercja polega na oznaczeniu pozycji w ciągu korekcji jako zawierającej symbol do usunięcia (linia 63). Jeżeli po wprowadzeniu zmiany nie osiągnięto końca regionu (linia 73), wówczas korekcja jest kontynuowana (linia 80). Utworzenie sekwencji bieżącego k-meru, która jest niezbędna do kontynuacji (linia 79), uwzględnia fakt pominięcia jednego symbolu. Dodawany na początku symbol pusty ma na celu jedynie zapewnienie, że $|\kappa^*| = k$ i zostanie

on pominięty w funkcji `Continue3′`. Korekcja błędów typu delecja uwzględnia weryfikację czterech możliwych symboli, z których jeden w wyniku błędu mógł zostać usunięty. W tym przypadku w ciągu korekcji zostają umieszczone dwie dodatkowe informacje: znacznik konieczności wstawienia symbolu oraz wartość tego symbolu (linia 94). W tym przypadku niemożliwa jest sytuacja natychmiastowego osiągnięcia końca regionu. Określenie sekwencji roboczego k-meru dla funkcji przeprowadzającej kontynuację (linia 92) polega na prostej zamianie jego ostatniego symbolu, a w efekcie kontynuacja musi zostać przeprowadzana od tej samej pozycji w odczycie wejściowym (linia 95). W rezultacie symbol znajdujący się w regionie na aktualnie korygowanej pozycji (a przy tym w odczycie wyjściowym potencjalnie umieszczony bezpośrednio za wstawionym symbolem korygującym) zostanie do tego k-meru dołączony w funkcji `Continue3′`.

Regiony wewnętrzne Korekcja regionów wewnętrznych odbywa się przy pomocy metody zbliżonej do korekcji skrajnych regionów końcowych. Uwzględnia jednak dodatkowy przypadek, gdy region obejmuje pozycję wystąpienia tylko jednego błędu typu delecja. W większości przypadków pojawienie się jednego błędu typu substytucja albo insercja powoduje pokrycie go przez k błędnych k-merów. W omawianej sytuacji liczba tych k-merów wynosi $k-1$, co można zaobserwować na rys. 5.6 — po pozycji 88 występuje brak jednego symbolu, powodując, że $k-1$ k-merów na pozycjach 83–88 jest błędnych. Podobne zjawisko miałoby miejsce w przypadku, gdy błąd tego typu znajdowałby się na końcu regionu zawierającego większą liczbę błędów.



Rysunek 5.6: Wystąpienie jednego błędu typu delecja w regionie; $k = 7, \ell = 100$; symbolem $\curlyvee$ oznaczono delecję

Na pseudokodach 25–28 przedstawiono algorytm korekcji wewnętrznych regionów. Korekcja pojedynczego błędu typu delecja jest przeprowadzana w przypadku, gdy jest on jedynym w regionie (linia 10), co jest wykazywane małą długością regionu, a także po osiągnięciu końca regionu (linie 27 oraz 83). Sama procedura

Pseudokod 25 Korekcja wewnętrznych regionów algorytmu RECKONER

Wejście: $k, \mathfrak{q}, \mathfrak{r}$
Globalne: $\mathfrak{s}, \theta_{\text{reg\_beg}}$

```
1:  function CorrectInternal(ψ)
2:      𝔰 ← ()                                              ▷ Pusty ciąg korekcji
3:      Π ← ∅
4:      κ* ← κ(𝔯, Beg(ψ))
5:      θ_reg_beg ← Beg(ψ)
6:      i ← 0                                               ▷ Pozycja ciągu korekcji
7:      θ_mx_chck ← θ_MX_CHCK
8:      if Beg(ψ) > End(ψ) then                             ▷ Region z jedną delecją
9:          θ_mx_ind ← 1
10:         (succ, Π) ← CorrectLastIntDel(κ*, 0, true)
11:     else
12:         θ_mx_ind ← min(θ_MX_IND, ⌈θ_MX_IND_RATE · |ψ|⌉)
13:         θ_mx_untrst ← ⌈θ_MX_NON_UNTRST · |ψ|⌉
14:         for all c ∈ Σ do
15:             if θ_mx_chck = 0 then
16:                 return Π
17:             end if
18:             θ_mx_chck ← θ_mx_chck − 1
19:             κ*[k − 1] ← c
20:             if Trusted(κ*) then
21:                 𝔰_i ← (η(κ*), c, false, false, ⊖)
22:                 if Beg(ψ) = End(ψ) then
23:                     π ← CreatePathExtend3'(i, 𝔰, ψ)
24:                     if π ≠ ∅ then
25:                         Π ← Π ∪ {π}
26:                     else
27:                         (succ, Π) ← CorrectLastIntDel(κ*, i, false)
28:                         if ¬succ then
29:                             if 𝔯[Beg(ψ) + k − 1] ≠ c then   ▷ Zmodyfikowano
30:                                 Check3′Indel(i, ψ, true)
31:                             end if
32:                         end if
33:                     end if
34:                 else
35:                     Π ←ContInternal(κ*, i + 1, (Beg(ψ) + 1, End(ψ)), Π)
36:                 end if
37:             end if
38:             if 𝔯[Beg(ψ) + k − 1] ≠ c then
39:                 Π ← Correct3′Indel(κ*, i, ψ, Π)
40:             end if
41:         end for
42:     end if
43:     return Π
44: end function
```

Pseudokod 26 Korekcja wewnętrznych regionów algorytmu RECKONER, cd.

```
45: function ContInternal(κ, i, ψ, Π)
46:     κ* ← κ[1, k − 1]𝔯[Beg(ψ) + k − 1]
47:     low ← 𝔮[Beg(ψ) + k − 1] < θ_Q_LOW        ▷ Czy symbol jest niskiej jakości

            ▷ Sytuacja 1 — k-mer jest zaufany i symbol nie jest niskiej jakości
48:     if ¬low ∧ Trusted(κ*) then                ▷ Niczego nie modyfikujemy
49:         𝔰_i ← (η(κ*), κ*[k − 1], false, false, ⊖)
50:         if Beg(ψ) = End(ψ) then
51:             if |Π| > θ_MX_PTH then
52:                 return Π
53:             end if
54:             π ← CreatePathExtend3'(i, 𝔰, ψ)
55:             if π ≠ ∅ then
56:                 Π ← Π ∪ {π}
57:             end if
58:         else
59:             Π ← ContInternal(κ*, i + 1, (Beg(ψ) + 1, End(ψ)), Π)
60:         end if

            ▷ Sytuacja 2 — k-mer nie jest zaufany lub symbol jest niskiej jakości
61:     else
62:         solid ← false                         ▷ Czy znaleziono zaufany k-mer
63:         if ¬low then
64:             Σ' ← Σ \ {κ*[k − 1]}              ▷ Nie ma sensu sprawdzać oryginalnego
65:         end if
66:         for all c ∈ Σ' do
67:             if θ_mx_chck = 0 then
68:                 return Π
69:             end if
70:             θ_mx_chck ← θ_mx_chck − 1
71:             κ*[k − 1] ← c
```

Pseudokod 27 Korekcja wewnętrznych regionów algorytmu RECKONER, cd. 2

```
                                          ▷ Sytuacja 2.1 — znaleziono zaufany k-mer
72:             if TRUSTED(κ*) then
73:                 solid ← true
74:                 𝔰_i ← (η(κ*), c, false, false, ⊖)
75:                 if BEG(ψ) = END(ψ) then
76:                     if |Π| > θ_MX_PTH then
77:                         return Π
78:                     end if
79:                     π ← CREATEPATHEXTEND3'(i, 𝔰, ψ)
80:                     if π ≠ ∅ then
81:                         Π ← Π ∪ {π}
82:                     else
83:                         (succ, Π) ← CORRECTLASTINTDEL(κ*, i + 1, false)
84:                         if ¬succ then
85:                             CHECK3′INDEL(i, ψ, true)
86:                         end if
87:                     end if
88:                 else
89:                     Π ← CONTINTERNAL(κ*, i + 1, (BEG(ψ) + 1, END(ψ)), Π)
90:                     CHECK3′INDEL(i, ψ, false)
91:                 end if
92:                 Π ← CORRECT3′INDEL(κ*, i, ψ, Π)
93:             end if
94:         end for

                                          ▷ Sytuacja 2.2 — nie znaleziono zaufanego k-meru
95:         if ¬solid then
96:             if θ_mx_untrst > 0 then
97:                 κ*[k − 1] ← 𝔱[BEG(ψ) + k − 1]
98:                 𝔰_i ← (0, κ*[k − 1], false, false, ⊖)
99:                 if BEG(ψ) = END(ψ) then
100:                    if |Π| > θ_MX_PTH then
101:                        return Π
102:                    end if
103:                    π ← CREATEPATHEXTEND3'(i, 𝔰, ψ)
104:                    if π ≠ ∅ then
105:                        Π ← Π ∪ {π}
106:                    else
107:                        CHECK3′INDEL(i, ψ, true)
108:                    end if
109:                else
110:                    Π ← CONTINTERNAL(κ*, i + 1, (BEG(ψ) + 1, END(ψ)), Π)
111:                end if
112:                Π ← CORRECT3′INDEL(κ*, i, ψ, Π)
113:            end if
114:        end if
115:    end if
```

Pseudokod 28 Korekcja wewnętrznych regionów algorytmu RECKONER, cd. 3

```
116:     return Π
117: end function

                                                ▷ single — korekcja pojedynczej delecji
118: function CorrectLastIntDel(κ, i, single)
119:     if i > 0 ∧ (Ins(𝔰_{i−1}) ∨ Del(𝔰_{i−1})) then ▷ Leniwe wartościowanie warunku
120:         return true          ▷ Nie ma sensu szukać kolejnego błędu typu indel
121:     end if
122:     if θ_mx_ind = 0 then
123:         return (true, Π)                ▷ Pomiń szukanie indeli w innej gałęzi
124:     end if
125:     succ ← false                                  ▷ Powodzenie rozszerzania
126:     θ_mx_ind ← θ_mx_ind − 1
127:     κ* ← κ
128:     if ¬single then  ▷ W regionach z jedną delecją modyfikujemy wejściowy
     k-mer
129:         κ* ← κ*[1, k − 1]∅
130:     end if
131:     for all c ∈ Σ do
132:         if θ_mx_chck = 0 then
133:             𝔰_i ← (∅, ∅, false, false, ⊖)
134:             return (true, Π)
135:         end if
136:         θ_mx_chck ← θ_mx_chck − 1
137:         κ*[k − 1] ← c
138:         if Trusted(κ*) then
139:             𝔰_i ← (η(κ*), c, false, true, ⊖)
140:             if |Π| > θ_MX_PTH then
141:                 𝔰_i ← (∅, ∅, false, false, ⊖)
142:                 return (true, Π)
143:             end if
144:             π ← CreatePathExtend3'(i, 𝔰, ψ)
145:             if π ≠ ∅ then
146:                 Π ← Π ∪ {π}
147:                 succ ← true
148:             end if
149:         end if
150:     end for
151:     𝔰_i ← (∅, ∅, false, false, ⊖)
152:     θ_mx_ind ← θ_mx_ind + 1
153:     return (succ, Π)
154: end function
```

korekcji, przedstawiona od linii 118, jest podobna do korekcji pozostałych błędów typu indel, jednak zawsze po niej następuje rozszerzanie regionu (linia 144).

Korekcja pierwszego k-meru W sytuacji gdy w odczycie nie wyznaczono żadnego regionu prawidłowego lub istnieje błędny region krótszy niż $k-1$, przeprowadzana jest osobna procedura korekcji pierwszego k-meru, przedstawiona na pseudokodach 29–32. Obejmuje ona podobną strategię korekcji metodą siłową $\theta_{\text{MX\_LQ}}$ najsłabszych symboli oraz sprawdzania niezależnie wszystkich modyfikacji kolejnych symboli k-meru, jak opisano w podrozdziale 5.3.2. Algorytm został jednak uzupełniony o obsługę dodatkowych przypadków. Gdy liczba symboli jakości niskiej jakości jest większa od $\theta_{\text{MX\_LQ}}$, przeprowadzana jest korekcja siłowa dla $\theta_{\text{MX\_LQ}}$ z nich o najmniejszych wartościach współczynników (linia 25). Ponadto, przy braku powodzenia wszystkich powyższych działań, następuje próba korekcji błędu typu indel (linia 30), która polega na próbie usunięcia kolejnych symboli odczytu o indeksach $1, 2, \ldots, k-1$ (próba korekcji błędu typu insercja) albo wstawieniu przed symbolami o tych indeksach poszczególnych symboli zbioru Σ (próba korekcji błędu typu delecja). Dopuszczalne jest pojawienie się co najwyżej jednej korekcji błędu typu indel. W każdym przypadku celem jest uzyskanie prawidłowej sekwencji pierwszego k-meru odczytu. Wszystkie wymienione czynności są elementami kroku 2.1.

Po osiągnięciu sukcesu następuje wykonanie korekcji pozostałej części odczytu w kierunku $3'$ po zastosowaniu zmian w pierwszym k-merze (linia 93, krok 2.2), niezależnie dla każdej z uzyskanych ścieżek pierwszego k-meru. Ścieżki uzyskane w ramach tej kontynuacji są łączone z odpowiednią ścieżką pierwszego k-meru (linia 95), zachowując wartości wymagane do oceny pełnej, połączonej ścieżki (krok 2.3).

Tworzenie ścieżek korekcji i rozszerzanie regionów Na pseudokodzie 33 przedstawiono algorytm tworzenia ścieżki korekcji w oparciu o zawartość ciągu korekcji. Zmienne w liniach 2 oraz 3 reprezentują odpowiednio pozycję w odczycie wyjściowym i wejściowym. Zadaniem pętli od linii 8 jest modyfikacja tych zmiennych w zależności od obecności w ciągu informacji o błędach różnych typów. Sama ścieżka korekcji jest ciągiem zmian, jakie kolejno należy wprowadzić do regionu odczytu w celu przeprowadzenia jego pełnej korekcji oraz określenia składników wartości oceny korekcji ścieżki (linie 12, 16, 19, 22 oraz 29). Uzyskana ścieżka zawiera w sobie znacznik typu błędu, symbol modyfikujący (błędy typu substytucja i delecja), pozycję zmiany (zgodnie z konwencją języka C++, tzn. substytucja na pozycji a oznacza modyfikację symbolu na pozycji a, insercja na pozycji a oznacza usunięcie symbolu na pozycji a, natomiast delecja na pozycji a oznacza wstawienie symbolu bezpośrednio przed symbolem na pozycji a; konwencja ta jest zachowana także w przypadku korekcji w kierunku $5'$ (pseudokod 60 w dodatku A) oraz

Pseudokod 29 Korekcja pierwszego k-meru algorytmu RECKONER

Wejście: $k, \ell = |\mathfrak{r}|, \mathfrak{r}, \mathfrak{q}, \text{long}$
Globalne: $\mathfrak{s}$

```
1: function CorrectFirst                ▷ Krok 2.1 — korekcja pierwszego k-meru
2:     𝔰 ← ()
3:     Π ← ∅
4:     κ ← κ(𝔯, 0)
5:     v_Q_LOW ← ()                     ▷ Ciąg pozycji o niskich współczynnikach jakości
6:     for all a ∈ {0, 1, ..., |𝔮| − 1} do
7:         if 𝔮[a] < θ_Q_LOW then
8:             v_Q_LOW ← v_Q_LOW a      ▷ Dołączenie pozycji do ciągu
9:         end if
10:    end for
11:    θ_lq ← |v_Q_LOW|
12:    if 0 < θ_lq ⩽ θ_MX_LQ then
13:        Π ← FirstBruteForce(0, κ, v_Q_LOW, Π)
14:        if Π = ∅ then
15:            Π ← FirstConsec(κ)
16:        end if
17:    else
18:        if Trusted(κ) then
19:            Π ← CreatePathExtendFirst(Π, 0, 𝔰, ())  ▷ Utworzenie pustej ścieżki
20:        else
21:            if θ_lq = 0 then
22:                Π ← FirstConsec(κ)
23:            else
24:                Pozostaw w ciągu (v_Q_LOW) tylko θ_Q_LOW najsłabszych symboli
25:                Π ← FirstBruteForce(0, κ, v_Q_LOW, Π)
26:            end if
27:        end if
28:    end if
29:    if Π = ∅ then
30:        Π ← FirstIndel(Π)
31:    end if
32:    if |Π| > θ_MX_FIRST_RES then
33:        Pozostaw w Π tylko θ_MX_FIRST_RES ścieżek o najwyższych ocenach
34:    end if
35:    Π ← FirstCont(Π)
36:    return Π
37: end function
```

Pseudokod 30 Korekcja pierwszego k-meru algorytmu RECKONER, cd.

```
38: function FirstIndel(Π)
39:     for i ← 1 to k − 1 do
40:         κ* ← 𝔱[0, i − 1]𝔱[i + 1, k]                    ▷ k-mer bez pozycji a
41:         if Trusted(κ*) then
42:             𝔰0 ← (η(κ*), ∅, true, false, ⊖)
43:             Π ← CreatePathExtendFirst(Π, 1, 𝔰, (i))
44:         end if
45:     end for
46:     for i ← 1 to k − 1 do
47:         for all c ∈ Σ do
                    ▷ Gdy i > k − 2, wówczas 𝔱[i, k − 2] jest napisem długości 0
48:             κ* ← 𝔱[0, i − 1]c𝔱[i, k − 2]    ▷ k-mer z dodatkowym symbolem po
    pozycji i − 1
49:             if Trusted(κ*) then
50:                 𝔰0 ← (η(κ*), c, false, true, ⊖)
51:                 Π ← CreatePathExtendFirst(Π, 1, 𝔰, (i))
52:                 if π ≠ ∅ then
53:                     Π ← Π ∪ {π}
54:                 end if
55:             end if
56:         end for
57:     end for
58:     return Π
59: end function
```

w przypadku korekcji pierwszego k-meru (pseudokod 36)).

Rozszerzanie końcowego regionu (przedstawione na pseudokodach 34 oraz 35) jest przeprowadzane, gdy w ścieżce są obecne modyfikacje pozycji odczytu oddalone od jego końca o mniej niż k symboli (tj. znajdujące się na pozycji $a > \ell - k$, gdzie a jest pozycją ostatniej modyfikacji po zastosowaniu wszystkich modyfikacji zmieniających długość odczytu). Rozszerzanie jest wykonywane w podobny sposób, jak opisano w podrozdziale 5.3.2, z tą różnicą, że dopuszczalne jest, aby wśród rozszerzających k-merów znajdowała się część k-merów niezaufanych (linia 55). Niepowodzenie rozszerzania powoduje odrzucenie ścieżki (linia 80), przy czym ścieżki niewprowadzające żadnych zmian są przyjmowane bez rozszerzania (linia 39). W przypadku korekcji w kierunku końca $5'$ lub korekcji pierwszego k-meru sytuacja rozszerzania jest symetryczna (pseudokody 61 oraz 62 w dodatku A).

Z kolei przy rozszerzaniu regionów wewnętrznych (pseudokod 37) nie jest przeprowadzane dołączenie nowych symboli do odczytu, ale dołączanie do regionu błędnego kolejnych symboli z regionu prawidłowego, znajdującego się bezpośrednio w kierunku końca $3'$.

Pseudokod 31 Korekcja pierwszego k-meru algorytmu RECKONER, cd. 2

```
60: function FirstCont(Π)
61:     Π_res ← ∅
62:     for all π ∈ Π do
63:         𝔰 ← ()
64:         ℓ_Δ ← 0
65:         if |π| > 0 then
66:             if Ins(π_0) then
67:                 ℓ_Δ ← ℓ_Δ − 1
68:             else if Del(π_0) then
69:                 ℓ_Δ ← ℓ_Δ + 1
70:             end if
71:         end if
72:         if ℓ_Δ < 0 then     ▷ W pierwszym k-merze skorygowano błąd insercji
73:             if ℓ ⩾ k + 1 then
74:                 κ_tmp ← 𝔱[1, k + 1]
75:                 a ← 2
76:             else                                          ▷ Koniec odczytu
77:                 if Extend3'(π, ℓ_Δ) = true then
78:                     Π_res ← Π_res ∪ {π}
79:                 end if
80:                 return
81:             end if
82:         else
83:             κ_tmp ← 𝔱[0, k]
84:             a ← 1
85:         end if
86:         κ_tmp ← Apply(κ_tmp, π)
87:         Π_n ← ∅    ▷ Ścieżki uzyskane podczas kontynuacji, ale niezawierające
    korekcji pierwszego k-meru
88:         ψ ← (a, ℓ − k)
89:         θ_mx_ind ← min(θ_MX_IND, ⌈θ_MX_IND_RATE · |ψ|⌉)
90:         θ_mx_untrst ← ⌈θ_MX_NON_UNTRST · |ψ|⌉
91:         θ_mx_chck ← θ_MX_CHCK
92:         θ_reg_beg ← a

                                  ▷ Krok 2.2 — kontynuacja korekcji w stronę końca 3′
93:         Π_n ← Continue3′(κ_tmp, 0, ψ, Π_n)

    ▷ Krok 2.3 — przygotowanie do wspólnej oceny wszystkich zmian w ścieżce
94:         for all π_n ∈ Π_n do
95:             Π_res ← Π_res ∪ MergePaths(π, π_n, ℓ_Δ)
96:         end for
97:     end for
98:     return Π_res
99: end function
```

Pseudokod 32 Korekcja pierwszego k-meru algorytmu RECKONER, cd. 3

```
100: function FirstBruteForce(i, κ, v_Q_LOW, Π)
101:     j ← v_Q_LOW,i                                   ▷ i-ty element ciągu
102:     for all c ∈ Σ do
103:         if |Π| ⩾ θ_MX_FIRST_PTHS then
104:             return Π
105:         end if
106:         κ* ← κ
107:         κ*[j] ← c
108:         𝔰_i ← (⊖, c, false, false, ⊖)
109:         if i = |v_Q_LOW| − 1 then
110:             if Trusted(κ*) then
111:                 Qual(𝔰_j) ← η(κ*)
112:                 Π ← CreatePathExtendFirst(Π, i + 1, 𝔰, v_Q_LOW)
113:             end if
114:         else
115:             Π ← FirstBruteForce(Π, i + 1, κ*, v_Q_LOW, Π)
116:         end if
117:     end for
118:     return Π
119: end function

120: function FirstConsec(κ)
121:     𝔰 ← ()
122:     Π ← ∅
123:     for i ← 1 to k − 1 do
124:         κ* ← κ
125:         for all c ∈ Σ \ {κ[i]} do
126:             κ*[i] ← c
127:             if Trusted(κ*) then
128:                 𝔰_0 ← (η(κ*), c, false, false, ⊖)
129:                 Π ← CreatePathExtendFirst(Π, 1, 𝔰, (i))
130:             end if
131:         end for
132:     end for
133:     return Π
134: end function

135: function MergePaths(π_L, π_R, ℓ_Δ)
136:     W każdym elemencie π_R,i ciągu (π_R) zwiększ Pos(π_R,i) o ℓ_Δ
137:     π ← π_L π_R        ▷ Wyjściowa ścieżka powstała z połączenia dwóch ścieżek
138:     PathQual(π) ← PathQual(π_L) + PathQual(π_R)
139:     PathWghtSum(π) ← PathWghtSum(π_L) + PathWghtSum(π_R)
140:     PathProb(π) ← PathProb(π_L) · PathProb(π_R)
141: end function
```

Pseudokod 33 Tworzenie ścieżki i rozszerzanie końcowego regionu algorytmu RECKONER

Wejście: $\theta_{\text{reg\_beg}}, k, \mathfrak{q}, \mathfrak{r}$

```
1:  function CreatePathExtend3′(i, 𝔰, ψ)
2:      output_pos ← θ_reg_beg + k − 1
3:      input_pos ← θ_reg_beg + k − 1
4:      π ← ()                                              ▷ Pusta ścieżka
5:      PathQual(π) ← 0                  ▷ Suma liczebności ważonych k-merów
6:      PathProb(π) ← 1                  ▷ Iloczyn prawdopodobieństw błędów
7:      ℓ_Δ ← 0                                             ▷ Zmiana długości odczytu
8:      for j ← 0 to i do
9:          if Ins(𝔰_j) then
10:             π ← π(output_pos, ∅, true, false)           ▷ Dołączenie krotki do ciągu
11:             ℓ_Δ ← ℓ_Δ − 1
12:             PathProb(π) ← PathProb(π) · θ_IND_PROB
13:         else if Del(𝔰_j) then
14:             π ← π(output_pos, Mod(𝔰_j), false, true)
15:             ℓ_Δ ← ℓ_Δ + 1
16:             PathProb(π) ← PathProb(π) · θ_IND_PROB
17:         else if Mod(𝔰_j) ≠ 𝔯[input_pos] then              ▷ Substytucja
18:             π ← π(output_pos, Mod(𝔰_j), false, false)
19:             PathProb(π) ← PathProb(π) · p(𝔮[input_pos])
20:         end if
21:         if ¬Ins(𝔰_j) then
22:             PathQual(π) ← PathQual(π) + Qual(𝔰_j · θ_COV_WEIGHT)
23:             output_pos ← output_pos + 1
24:         end if
25:         if ¬Del(𝔰_j) then
26:             input_pos ← input_pos + 1
27:         end if
28:     end for
29:     PathWghtSum(π) ← (i + 1 − ℓ_Δ) · θ_COV_WEIGHT
30:     if ψ jest regionem wewnętrznym then
31:         return ExtendInternal(π, ℓ_Δ)
32:     else
33:         return Extend3′(π, ℓ_Δ, ψ)
34:     end if
35: end function
```

Pseudokod 34 Tworzenie ścieżki i rozszerzanie końcowego regionu algorytmu RECKONER cd.

36: **function** EXTEND3′(π, ℓ_Δ)
37: $n_{\text{mod}} \leftarrow |\pi|$
38: **if** $n_{\text{mod}} = 0$ **then**
39: **return** π
40: **end if**
▷ Wyznaczenie ostatniej zmodyfikowanej pozycji w regionie
41: $\theta_{\text{last\_mod}} \leftarrow \text{POS}(\pi_{n_{\text{mod}}-1}) + \ell_\Delta$
42: $\theta_{\text{last\_kmer}} \leftarrow \ell - k + \ell_\Delta$ ▷ Pozycja ostatniego k-meru
43: **if** $\theta_{\text{last\_mod}} \leqslant \theta_{\text{last\_kmer}}$ **then**
44: **return** π
45: **else** ▷ Wyznaczenie liczby rozszerzających pozycji
46: **if** $\theta_{\text{last\_mod}} \leqslant \theta_{\text{last\_kmer}} + \theta_{\text{MX\_E}}$ **then**
47: $\theta_{\text{ext}} \leftarrow \theta_{\text{last\_mod}} - \theta_{\text{last\_kmer}}$
48: **else**
49: $\theta_{\text{ext}} \leftarrow \theta_{\text{MX\_E}}$
50: **end if**
51: $\mathfrak{r}^* \leftarrow \text{APPLY}(\mathfrak{r}, \pi)$
52: $\ell^* \leftarrow |\mathfrak{r}^*|$
53: $\kappa^* \leftarrow \kappa_{\text{DUMMY}}(\mathfrak{r}^*, \ell^* - k + 1)\varnothing$
54: succ $\leftarrow$ **false**
55: $\theta_{\text{mx\_untrst}} \leftarrow \lceil \theta_{\text{MX\_NON\_UNTRST}} \cdot \theta_{\text{ext}} \rceil$
56: $\eta_{\text{max\_ext}} \leftarrow 0$ ▷ Maksymalna liczebność pierwszego rozszerzającego k-meru
57: **for all** $c \in \Sigma$ **do**
58: $\kappa^*[k-1] \leftarrow c$
59: solid $\leftarrow$ TRUSTED(κ^*)
60: **if** solid $\vee\ \theta_{\text{mx\_untrst}} > 0$ **then**
61: $\eta_{\text{max\_ext}} \leftarrow \max(\eta_{\text{max\_ext}}, \eta(\kappa^*))$
62: **if** $\theta_{\text{ext}} = 1$ **then**
63: **if** solid **then**
64: succ $\leftarrow$ **true**
65: **end if**
66: **else if** $\neg$succ **then**
67: **if** solid **then**
68: succ $\leftarrow$ CONTEXTEND3'($\theta_{\text{ext}}, 1, \theta_{\text{mx\_untrst}}, \kappa^*$)
69: **else**
70: succ $\leftarrow$ CONTEXTEND3'($\theta_{\text{ext}}, 1, \theta_{\text{mx\_untrst}} - 1, \kappa^*$)
71: **end if**
72: **end if**
73: **end if**
74: **end for**

Pseudokod 35 Tworzenie ścieżki i rozszerzanie końcowego regionu algorytmu RECKONER, cd. 2

```
75:         if succ then
76:             PathQual(π) ← PathQual(π) + η_max_ext · θ_EXT_WEIGHT
77:             PathWghtSum(π) ← PathWghtSum(π) + θ_EXT_WEIGHT
78:             return π
79:         else
80:             return ∅
81:         end if
82:     end if
83: end function

84: function ContExtend3′(θ_ext, a, θ_mx_untrst, κ*)
85:     κ* ← κ*[1, k − 1]∅
86:     for all c ∈ Σ do
87:         κ*[k − 1] ← c
88:         solid ← Trusted(κ*)
89:         if solid ∨ θ_mx_untrst > 0 then
90:             if a + 1 = θ_ext then
91:                 return true
92:             else
93:                 if solid then
94:                     succ ← ContExtend3′(θ_ext, a + 1, θ_mx_untrst, κ*[1, k − 1])
95:                 else
96:                     succ ← ContExtend3′(θ_ext, a+1, θ_mx_untrst −1, κ*[1, k−1])
97:                 end if
98:                 if succ then
99:                     return true
100:                end if
101:            end if
102:        end if
103:    end for
104:    return false
105: end function
```

Pseudokod 36 Tworzenie ścieżki przy korekcji pierwszego k-meru algorytmu RECKONER

Wejście: $\mathfrak{r}$

```
                                          ▷ v_pos — ciąg indeksów modyfikacji
 1: function CreatePathExtendFirst(Π, i, 𝔰, v_pos)
 2:     π ← ()                                                  ▷ Pusta ścieżka
 3:     PathQual(π) ← Qual(𝔰_{i−1}) · θ_COV_WEIGHT              ▷ Suma liczebności
    ważonych k-merów
 4:     PathProb(π) ← 1                          ▷ Iloczyn prawdopodobieństw błędów
 5:     PathWghtSum(π) ← 1 · θ_COV_WEIGHT
 6:     subst ← true              ▷ Ścieżka zawiera wyłącznie korekcje substytucji
 7:     if i = 0 then
 8:         return Π ∪ {π}
 9:     else if i = 1 then
10:         if Ins(𝔰_0) then
11:             π ← π(v_{pos,0}, ∅, true, false)
12:             PathProb(π) ← PathProb(π) · θ_IND_PROB
13:             subst ← false
14:         else if Del(𝔰_0) then
15:             π ← π(v_{pos,0}, Mod(𝔰_0), false, true)
16:             PathProb(π) ← PathProb(π) · θ_IND_PROB
17:             subst ← false
18:         end if
19:     end if
20:     if subst then
21:         for j ← 0 to i − 1 do
22:             if Mod(𝔰_j) ≠ 𝔯[v_{pos,j}] then              ▷ j-ty element ciągu
23:                 π ← π(v_{pos,j}, Mod(𝔰_j), false, false)
24:                 PathProb(π) ← PathProb(π) · p(𝔮[v_{pos,j}])
25:             end if
26:         end for
27:     end if
28:     π ←Extend5′(π)
29:     if π ≠ ∅ then
30:         Π ← Π ∪ {π}
31:     end if
32:     return Π
33: end function
```

Pseudokod 37 Rozszerzanie regionów wewnętrznych algorytmu RECKONER

Wejście: $k, \mathfrak{q}, \mathfrak{r}$
Globalne: $\theta_{\text{reg\_beg}}$

1: **function** ExtendInternal(π, ℓ_Δ, ψ)
2: $n_{\text{mod}} \leftarrow |\pi|$
3: **if** $n_{\text{mod}} = 0$ **then**
4: **return** π
5: **end if**
▷ Wyznaczenie ostatniej zmodyfikowanej pozycji w regionie
6: $\theta_{\text{last\_mod}} \leftarrow \text{End}(\psi) + \ell_\Delta$
7: $\theta_{\text{last\_chck}} \leftarrow \text{Pos}(\pi_{n_{\text{mod}}-1}) - k + 1 + \theta_{\text{MX\_E}}$ ▷ Ostatni sprawdzany k-mer
8: **if** $\theta_{\text{last\_chck}} > \theta_{\text{last\_mod}}$ **then**
9: $\mathfrak{r}^* \leftarrow \text{Apply}(\mathfrak{r}, \pi)$
10: $\eta_{\text{max\_ext}} \leftarrow 0$ ▷ Maksymalna liczebność pierwszego rozszerzającego k-meru
11: $n_{\text{succ}} \leftarrow 0$
12: $\kappa^* \leftarrow \kappa(\mathfrak{r}^*, \theta_{\text{last\_mod}} + 1)$
13: **if** Trusted(κ^*) **then**
14: $\eta_{\text{max\_ext}} \leftarrow \max(\eta_{\text{max\_ext}}, \eta(\kappa^*))$
15: $n_{\text{succ}} \leftarrow n_{\text{succ}} + 1$
16: **end if**
17: **for** $a \leftarrow \theta_{\text{last\_mod}} + 2$ **to** $\theta_{\text{last\_chck}}$ **do**
18: $\kappa^* \leftarrow \kappa(\mathfrak{r}^*, a)$
19: **if** Trusted(κ^*) **then**
20: $n_{\text{succ}} \leftarrow n_{\text{succ}} + 1$
21: **end if**
22: **end for**
23: $\theta_{\text{mx\_untrst}} \leftarrow \lceil \theta_{\text{MX\_NON\_UNTRST}} \cdot (\theta_{\text{last\_chck}} - \theta_{\text{last\_mod}} + 1) \rceil$
24: **if** $n_{\text{succ}} > (\theta_{\text{last\_chck}} - \theta_{\text{last\_mod}} + 1) - \theta_{\text{mx\_untrst}}$ **then**
25: PathQual(π) $\leftarrow$ PathQual(π) $+ \eta_{\text{max\_ext}} \cdot \theta_{\text{EXT\_WEIGHT}}$
26: PathWghtSum(π) $\leftarrow$ PathWghtSum(π) $+ \theta_{\text{EXT\_WEIGHT}}$
27: **return** π
28: **end if**
29: **else**
30: **return** π
31: **end if**
32: **end function**

Generacja wyników

Rezultatem korekcji jednego odczytu jest jego nowa (bądź oryginalna) sekwencja wraz z odpowiednio zmodyfikowanym ciągiem współczynników jakości. Wynik ten jest poddawany kompresji (jeśli dane wejściowe zostały podane w formie skompresowanej) i zapisywany do pliku tymczasowego, zawierającego zbiór odczytów jednego podzadania. Następnie pliki te są równolegle do korekcji odczytywane w kolejności zgodnie z zawartością pliku wejściowego i dołączone do pliku wynikowego, gwarantując zachowanie oryginalnej kolejności odczytów, niezbędnej podczas korekcji zestawów sparowanych. Struktura skompresowanego pliku wynikowego zawierająca skonkatenowany ciąg bloków danych jest dopuszczalna dzięki własności formatu GZIP, który pozwala na organizację pliku w formie połączonych, niezależnie skompresowanych składowych, z których każdy posiada własny nagłówek. W rezultacie kompresja poszczególnych części odbywa się w wątkach roboczych (podrozdział 5.4.4), zatem w sposób równoległy. Co więcej, taka strategia nie wymaga przechowywania danych tymczasowych w formie nieskompresowanej.

5.4.4 Przetwarzanie równoległe

Własności równoległej pracy algorytmu RECKONER wynikają z rozdzieleniu algorytmu na kilka narzędzi. Implementacje KMC oraz KMC tools posiadają własne strategie pracy równoległej, niezależne od zasadniczej części algorytmu RECKONER. W niniejszym podrozdziale zawarto opis przetwarzania tego ostatniego etapu. Na pseudokodzie 38 zawarto pomocnicze funkcje, wykorzystane w dalszej części opisu.

Przetwarzanie równoległe algorytmu korekcji zostało w pewnym stopniu oparte na metodzie zastosowanej w algorytmie KMC w wersji 1. Dla zadanej liczby wątków roboczych (przeprowadzających korekcję) ρ_c tworzonych jest $\rho_c + 3$ wątków, spośród których wątki p_r oraz p_w są odpowiedzialne za przetwarzanie plików wejściowych i wyjściowych, natomiast p_m jest wątkiem głównym, w ramach którego następuje wykonanie zadań inherentnie sekwencyjnych, utworzenie pozostałych wątków oraz oczekiwanie na ich zakończenie. Zasadnicza korekcja jest przeprowadzana w ramach wątków roboczych p_i, $i \in \{1, 2, \ldots, \rho_c\}$.

Odczyt danych do przetworzenia jest wykonywany przez wątek p_r, pełniący rolę wątku nadrzędnego (spełniając podobną funkcję jak procesor nadrzędny w scentralizowanej metodzie przydziału zadań). Wątek ten jest odpowiedzialny za odczyt bieżącego pliku dyskowego, jego dekompresję oraz buforowanie wczytanych fragmentów. Algorytm wykonywany przez wątek został przedstawiony na pseudokodzie 39. Wątki robocze (odpowiedniki procesorów podległych) p_i cyklicznie pobierają z bufora fragmenty danych, przetwarzają je oraz (potencjalnie kompresując) zapisują rezultaty w plikach tymczasowych.

Odczyt pliku w ramach osobnego wątku pozwala na wykonanie czasochłonnych, choć niewymagających znacznej mocy obliczeniowej procesora operacji dostępu do pamięci masowej. Jednocześnie umożliwia wykonywanie na bieżąco podziału zbioru odczytów wejściowych na podzbiory stanowiące zadania wątków roboczych. Jedno takie zadanie obejmuje co najwyżej 8 MiB danych zdekompresowanych. Alternatywne rozważane podejście polegało na wstępnym odczycie pliku, iteracji po jego zawartości oraz wyznaczaniu pozycji rozpoczynających części przetwarzane w sposób równoległy. Rozwiązanie to zostało zarzucone ze względu na konieczność dwukrotnego odczytu całego pliku oraz trudności w wykonywaniu skoków do określonych pozycji pliku skompresowanego (wykorzystana biblioteka programistyczna, przypuszczalnie ze względu na charakterystykę formatu GZIP, podczas wykonania skoku do pewnego miejsca w pliku wymaga dekompresji całej zawartości pliku zawartej przed tym miejscem). Rozważano także możliwość współdzielenia dostępu do pliku przez wątki robocze, tak aby odczyt odbywał się bezpośrednio przez te wątki. Pomysł zarzucono jednak ze względu na zbyt duże ryzyko znacznego wpływu synchronizacji dostępu do pliku na czas pracy wątków roboczych, w szczególności oczekiwania na zwolnienie dostępu do pliku przez inny wątek.

W zastosowanym rozwiązaniu może dojść do sytuacji, gdy wątek nadrzędny będzie stanowił wąskie gardło , co może być efektem ograniczenia przepustowości dysku lub znacznego czasu wymaganego na dekompresję przez pojedynczy wątek. Z tego powodu przeprowadzono eksperymenty określające skalowalność algorytmu, których wyniki zostaną przedstawione w podrozdziale 6.4.3. Wskazują one, że zjawisko to nie występuje nawet dla dużej liczby wątków roboczych $\rho_c = 24$.

Połączenie tymczasowych, cząstkowych plików generowanych przez wątki robocze jest przeprowadzane przez wątek scalający p_w. Jego zadaniem jest oczekiwanie na utworzenie nowego pliku tymczasowego przez któryś z wątków roboczych. W takim przypadku wykonywane jest scalenie tego pliku z plikiem wynikowym lub, gdy poprzedni fragment jeszcze nie został utworzony, oczekiwanie na jego dostępność. Algorytm działania wątku p_w został przedstawiony na pseudokodzie 40.

Wszystkie powyższe cechy współbieżności są zgodne z modelem przetwarzania z pamięcią wspólną. Zrezygnowano z możliwości wykonania obliczeń z przesyłaniem komunikatów, ze względu na spodziewane niskie przyspieszenie spowodowane koniecznością przesyłu dużej ilości informacji (odczytów oraz danych tymczasowych, zwłaszcza podczas zliczania k-merów), a także ze względu na niedostępność odpowiedniego środowiska sprzętowego pozwalającego na dogłębną ocenę takiego rozwiązania.

Synchronizacja i wzajemne wykluczenie

Zapewnienie prawidłowości wyników przetwarzania równoległego wymaga uwzględnienia wzajemnego wykluczenia dostępu do niektórych danych wspólnych przez różne wątki, a także synchronizacji pracy wątków w formie oczekiwania na dostępność danych. Problemy te zostały rozwiązane przy pomocy semaforów ς_{r} oraz ς_{w}, odpowiedzialnych za synchronizację dostępu do dwóch kolejek, odpowiednio wczytanych fragmentów pliku wejściowego Q_{r} (których odczyty powinny zostać poddane korekcji) oraz indeksów skorygowanych fragmentów pliku wyjściowego Q_{w} (które powinny zostać połączone w celu uzyskania ostatecznego wejścia).

Należy zaznaczyć, że zawarte na pseudokodach wysokopoziomowe operacje czekania na spełnienie warunków modyfikują stan tych semaforów. W momencie rozpoczęcia oczekiwania na semaforze osłaniającym daną operację oczekiwania zostaje wykonana operacja Signal, a wraz ze spełnieniem warunku — operacja Wait. Umożliwia to dostęp do odpowiednich struktur danych celem spełnienia warunków.

Pseudokod 38 Pomocnicze procedury i funkcja na potrzeby omówienia współbieżności

```
1:  procedure START(p)
2:      Rozpocznij pracę wątku p
3:  end procedure

4:  procedure JOIN(p)
5:      Oczekuj na zakończenie wątku p
6:  end procedure

7:  procedure PUSH(Q, x)
8:      Umieść wartość x w kolejce Q
9:  end procedure

10: procedure POP(Q)
11:     Usuń jedną wartość z kolejki Q
12: end procedure

13: function TOP(Q)
14:     return pierwsza wartość w kolejce Q
15: end function
```

Pseudokod 39 Przebieg wątku p_r

Globalne: Q_r — kolejka FIFO fragmentów wejściowych
Globalne: ς_r — semafor binarny zapewniający wzajemne wykluczenie dostępu do Q_r
Globalne: fin_input — znacznik zakończenia wczytania pliku

```
1:  i_in_frag ← 0                          ▷ Numer kolejnego wczytanego fragmentu pliku
2:  while nie wczytano całości pliku do
3:      Wczytaj fragment pliku do zmiennej frag, być może zdekompresuj go
4:      WAIT(ς_r)
5:      if |Q_r| ⩾ 2 · ρ_c then
6:          Czekaj na zmniejszenie liczby elementów w kolejce Q_r
7:      end if
8:      PUSH(Q_r, (frag, i_in_frag))
9:      SIGNAL(ς_r)
10:     i_in_frag ← i_in_frag + 1
11: end while
12: fin_input ← true                       ▷ Osiągnięto koniec pliku wejściowego
```

Pseudokod 40 Przebieg wątku p_w

Globalne: Q_w — kolejka priorytetowa fragmentów wyjściowych o kluczu będącym numerem fragmentu
Globalne: ς_w — semafor binarny zapewniający wzajemne wykluczenie dostępu do Q_w
Globalne: fin_corr — znacznik zakończenia korekcji odczytów pliku

```
1:  i_out_frag ← 0                         ▷ Numer kolejnego fragmentu pliku do zapisu
2:  while true do
3:      WAIT(ς_w)
4:      Czekaj na spełnienie warunku (|Q_w| ≠ 0 ∧ TOP(Q_w) = i_out_frag) ∨
    (|Q_w| = 0 ∧ fin_corr = true)
5:      if fin_corr = true ∧ |Q_w| = 0 then
6:          break
7:      else
8:          POP(Q_w)
9:      end if
10:     SIGNAL(ς_w)
11:     Dołącz plik tymczasowy o indeksie i_out_frag do pliku wynikowego
12:     i_out_frag ← i_out_frag + 1
13: end while
```

Pseudokod 41 Przebieg wątku p_{m}

Globalne: fin_corr — znacznik zakończenia korekcji odczytów pliku

```
1:  fin_corr ← false                          ▷ Czy wątki robocze zakończyły pracę
2:  START(p_w)
3:  START(p_r)
4:  for all i ∈ {1, 2, ..., ρ_c} do
5:      START(p_i)
6:  end for
7:  for all i ∈ {1, 2, ..., ρ_c} do
8:      JOIN(p_i)
9:  end for
10: WAIT(ς_w)
11: fin_corr ← true
12: SIGNAL(ς_w)
13: JOIN(p_w)
14: JOIN(p_r)
```

5.5 Analiza złożoności obliczeniowej algorytmu RECKONER

Głównym składnikiem czasu pracy algorytmu jest korekcja wyznaczonych błędnych regionów, wobec tego zostanie ona uwzględniona w analizie złożoności obliczeniowej. Złożoność zostanie wyznaczona w formie funkcji określających liczbę sprawdzeń zaufania k-merów, polegających na wykonaniu zapytań do bazy k-merów za pomocą KMC API. Następnie, w twierdzeniu 6, zostanie wyznaczona pełna złożoność, w której jako operacja dominująca zostanie przyjęta liczba przypadków porównania symboli k-merów z symbolami obecnymi w bazie.

Pesymistyczny przypadek korekcji odczytu ma miejsce w sytuacji, gdy nie wyznaczono żadnych prawidłowych regionów algorytmu albo istnieje błędny region krótszy niż k — w takim przypadku następuje wykonanie korekcji pierwszego k-meru, a jego kontynuacja w postaci algorytmu z powrotami jest przeprowadzana dla największej liczby symboli $\ell - k$.

Dodatkowe oznaczenia i przyjęte założenia

W poniższych obliczeniach przyjęto dodatkowo następujące oznaczenia: $n_\psi = |\psi|, \psi \in \Psi_{\mathrm{untrust}}$ jest długością bieżącego błędnego regionu. Wykorzystano także następujące oznaczenia zgodne z obecnymi w pseudokodach zmiennymi: θ_{ext} — aktualna liczba pozycji rozszerzających odczyt oraz θ_{lq} — liczba pozycji niskiej jakości pierwszego k-meru.

Ponadto postawiono następujące założenia, upraszczające wyznaczenie złożoności:

Pseudokod 42 Przebieg wątku p_i

Globalne: fin_input — znacznik zakończenia wczytania pliku

```
1: while true do
2:     next_frag ← GetFrag
3:     if next_frag = ∅ then
4:         break
5:     end if
6:     (i_in_frag, frag) ← next_frag
7:     Przeprowadź korekcję wszystkich odczytów we frag
8:     Zapisz w pliku tymczasowym o indeksie i_in_frag rezultat korekcji, być może
   skompresuj go
9:     wait(ς_w)
10:    Push(Q_w, i_in_frag)
11:    signal(ς_w)
12: end while

13: function GetFrag
14:    wait(ς_r)
15:    Czekaj na spełnienie warunku |Q_r| ≠ 0 ∨ fin_input = true
16:    if |Q_r| ≠ 0 then
17:        (frag, i_in_frag) ← Top(Q_r)
18:        Pop(Q_r)
19:        signal(ς_r)
20:        return (frag, i_in_frag)
21:    else
22:        signal(ς_r)
23:        return ∅
24:    end if
25: end function
```

1. $k > \theta_{\text{MX\_E}}$ — skrajny region może zostać rozszerzony co najwyżej o liczbę symboli mniejszą niż długość k-meru (w przypadku modyfikacji pierwszego lub ostatniego symbolu odczytu zostaje on rozszerzony o $\theta_{\text{MX\_E}}$ symboli),

2. w wyniku korekcji metodą siłową może zostać wygenerowanych $\theta_{\text{MX\_FIRST\_PTHS}}$ ścieżek, z których wszystkie będą miały pierwszą modyfikację wśród $k - \theta_{\text{MX\_E}}$ pozycji,

3. $k \geqslant 10$,

4. $\ell - k \geqslant 10$,

5. nie ma ograniczeń odnośnie do wzajemnych odległości korekcji błędów typu indel w ścieżce,

6. w celu podjęcia próby korekcji błędu typu indel nie jest konieczna analiza liczby wprowadzonych korekcji błędów typu substytucja; istotne jest tylko, żeby istniała możliwość wprowadzenia co najmniej jednej,

7. w wyniku zastosowania pewnej ścieżki korekcji długość regionu (w konsekwencji także odczytu) nie zmieni się, tzn. $\ell_\Delta = 0$; wynika z tego, że liczby korekcji błędów odpowiednio typu insercja i delecja w regionie są takie same,

8. symbol znajdujący się na ostatniej pozycji odczytu, tzn. na pozycji $\ell - 1$, może być efektem korekcji błędu typu delecja,

9. istnieje możliwość, że w odczycie nie wyznaczono żadnego prawidłowego regionu, a jednocześnie każda k-elementowa wariacja z powtórzeniami ze zbioru Σ może wystąpić w spektrum,

10. złożoność obliczeniowa wykonania jednego zapytania do bazy k-merów wynosi $T_{\text{kmc}}(k) = k$.

Założenie 10 jest oparte na obserwacji, że liczba wszystkich oligomerów długości k może wynosić co najwyżej $|\Sigma|^k$. Ponieważ zapytania są wykonywane do — będącej efektem działania KMC tools — bazy w wersji 1, wyszukiwanie odbywa się poprzez wyznaczenie pozycji prefiksu w czasie $O(1)$ oraz wyszukiwanie połówkowe w czasie $O(\log \bar{k})$, gdzie $\bar{k}$ jest liczbą wszystkich k-merów w bazie o wybranym sufiksie. Ograniczając złożoność całej operacji od góry, w szczególności przyjmując, że wyszukiwanie połówkowe odbywa się w całej bazie rozmiaru $|\Sigma|^k$, czas wyszukiwania wynosi $O(\log 2^k) = O(k)$.

Wyznaczenie złożoności obliczeniowej korekcji w kierunku 3′

Lemat 15. *Pesymistyczna złożoność obliczeniowa korekcji regionu w kierunku* $3'$ *algorytmu RECKONER jest nie większa niż:*

$$\begin{aligned} &T_{\text{R3'}}(n_\psi, \theta_{\text{mx\_ind}}, \theta_{\text{MX\_CHCK}}) = \\ &= \min\left(\theta_{\text{MX\_CHCK}}; \frac{20}{3}\left(19^{n_\psi+\theta_{\text{mx\_ind}}-1} - 1\right) + \frac{5}{6}\left(19^{n_\psi+\theta_{\text{mx\_ind}}-2} - 1\right) + 8\right) = \\ &= O(\theta_{\text{MX\_CHCK}}). \end{aligned} \tag{5.27}$$

Dowód. Rozpatrywany przypadek obejmują pseudokody 19–21, a także funkcja `Correct3′Indel` przedstawiona na pseudokodach 23 oraz 24. Pomocniczo zostanie wprowadzona funkcja matematyczna $T''_{\text{R3'}}(n_\psi, \theta_{\text{mx\_ind}})$, pozwalająca na analizę przypadku, gdy wpływ długości regionu n_ψ oraz pozostałej dopuszczalnej liczby korekcji błędów typu indel w regionie $\theta_{\text{mx\_ind}}$ poza funkcją `Correct3′` są rozpatrywane rozłącznie. Wykorzystane zostaną także założenia 5 oraz 6, eliminujące

konieczność analizy minimalnych odległości między skorygowanymi błędami typu indel. Następnie zostanie wprowadzona funkcja matematyczna $T'_{\mathrm{R3'}}(n_{\psi,\mathrm{mx\_ind}})$, ograniczająca poprzedni przypadek z góry. Jej postać będzie stanowiła podstawę do wyprowadzenia równania przedstawionego w tezie pracy, obejmującego również funkcję `Correct3′`.

W pierwszej kolejności zostanie przeanalizowane działanie funkcji `Continue3′`. Pesymistyczny przypadek obejmuje sytuację, gdy bez powodzenia zostanie sprawdzony warunek sytuacji 1, a następnie wykonany kod w sytuacji 2.1.

Przebieg funkcji jest następujący: w linii 34 pseudokodu 20 zostaje wykonane jednokrotne zapytanie do bazy. W przypadku uzyskania negatywnego wyniku w linii 59, zapytanie zostanie wykonane trzykrotnie, podobnie wywołanie rekursywne dla argumentu $n_\psi - 1$ (linia 73). Po zakończeniu każdego wywołania dodatkowo wywołana zostaje funkcja `Correct3′Indel` (linia 76).

Wykonanie funkcji `Correct3′Indel` wiąże się z jednokrotnym wywołaniem funkcji `Continue3′` dla argumentów $n_\psi - 1$ oraz $\theta_{\mathrm{mx\_ind}} - 1$ (pseudokod 23, linia 80), a następnie z czterokrotnym wykonaniem zapytania do bazy (linia 93) oraz czterokrotnym wywołaniem funkcji `Continue3′` dla $\theta_{\mathrm{mx\_ind}} - 1$ (linia 95). Ostatecznie złożoność obliczeniowa funkcji `Continue3′` (wraz z korekcją błędów typu indel) wynosi:

$$T''_{\mathrm{R3'}}(n_\psi, \theta_{\mathrm{mx\_ind}}) = 4T''_{\mathrm{R3'}}(n_\psi - 1, \theta_{\mathrm{mx\_ind}}) + 12T''_{\mathrm{R3'}}(n_\psi, \theta_{\mathrm{mx\_ind}} - 1) + 3T''_{\mathrm{R3'}}(n_\psi - 1, \theta_{\mathrm{mx\_ind}} - 1) + 15,$$

gdzie $T''_{\mathrm{R3'}}(n_\psi, 0) = 4T''_{\mathrm{R3'}}(n_\psi - 1, 0) + 3$, z kolei $T''_{\mathrm{R3'}}(0, \theta_{\mathrm{mx\_ind}}) = 0$, ze względu na fakt, że osiągnięcie końca regionu ($n_\psi = 0$) implikuje brak podstaw do przeprowadzenia korekcji błędu typu indel. Ponadto $T''_{\mathrm{R3'}}(0, 0) = 0$.

Według wiedzy autora równanie to w ogólnym przypadku nie jest możliwe do rozwiązania. W związku z tym jego wartość zostanie ograniczona z góry:

$$\begin{aligned} T''_{\mathrm{R3'}}(n_\psi, \theta_{\mathrm{mx\_ind}}) = \\ 4T''_{\mathrm{R3'}}(n_\psi - 1, \theta_{\mathrm{mx\_ind}}) + 12T''_{\mathrm{R3'}}(n_\psi, \theta_{\mathrm{mx\_ind}} - 1) + \\ + 3T''_{\mathrm{R3'}}(n_\psi - 1, \theta_{\mathrm{mx\_ind}} - 1) + 15 \leqslant \\ 7T''_{\mathrm{R3'}}(n_\psi - 1, \theta_{\mathrm{mx\_ind}}) + 12T''_{\mathrm{R3'}}(n_\psi, \theta_{\mathrm{mx\_ind}} - 1) + 15 \leqslant \\ T'_{\mathrm{R3'}}(n_\psi + \theta_{\mathrm{mx\_ind}}), \end{aligned}$$

dla pewnej funkcji $T'_{\mathrm{R3'}}(n_{\psi,\mathrm{mx\_ind}}) = 19T'_{\mathrm{R3'}}(n_{\psi,\mathrm{mx\_ind}} - 1) + 15$, gdzie $T'_{\mathrm{R3'}}(0) = 0$, co po eliminacji rekurencji ulega uproszczeniu do $T'_{\mathrm{R3'}}(n_{\psi,\mathrm{mx\_ind}}) = \frac{8}{9}(19^{n_{\psi,\mathrm{mx\_ind}}} - 1)$.

Prawdziwość przedstawionego ograniczenia wynika z następujących obserwacji:

- maksymalna głębokość rekurencyjnego zagnieżdżania funkcji $T''_{\mathrm{R3'}}(n_\psi, \theta_{\mathrm{mx\_ind}})$ wynosi $n_\psi + \theta_{\mathrm{mx\_ind}} + 1$,

- w przypadku, gdy w rekurencyjnym wartościowaniu tej funkcji zmienna parametryzująca $\theta_{\text{mx\_ind}}$ osiągnie wartość 0, wówczas rekurencja jest dalej wywoływana (gdy również $n_\psi > 0$), nie przekraczając jednak wspomnianego w poprzednim punkcie poziomu zagnieżdżania, ale powodując, że w kolejnych rekurencyjnych wartościowaniach liczba wywołań będzie mniejsza.

Zaproponowana funkcja $T'_{\text{R3'}}(\cdot)$ cechuje się następującymi własnościami, odnoszącymi się do obserwacji z odpowiednich powyższych punktów:

- maksymalna głębokość rekurencyjnego zagnieżdżania równania $T'_{\text{R3'}}(n_\psi + \theta_{\text{mx\_ind}})$ wynosi $n_\psi + \theta_{\text{mx\_ind}} + 1$, zatem dokładnie tyle samo, co równania $T''_{\text{R3'}}(n_\psi, \theta_{\text{mx\_ind}})$,

- w każdym rekurencyjnym wywołaniu dla $n_{\psi,\text{mx\_ind}} > 0$ liczba kolejnych rekurencyjnych wywołań wyniesie 19, podczas gdy w funkcji $T''_{\text{R3'}}(n_\psi, \theta_{\text{mx\_ind}})$ może być mniejsza, gdy $\theta_{\text{mx\_ind}} = 0$.

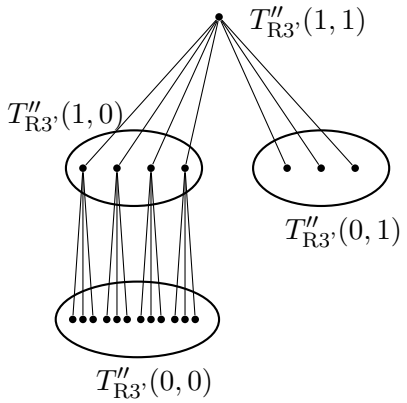
Wyjaśnienie to zilustrowano na rys. 5.7, upraszczając poprzez rezygnację z uwzględnienia błędów typu insercja oraz zaznaczenie tylko liczby wywołań funkcji. Z tego wynika, że funkcja $T'_{\text{R3'}}(\cdot)$ przy podanej parametryzacji ogranicza funkcję $T''_{\text{R3'}}$ z góry.

Uwzględniając fakt, że w funkcji `Correct3′` są wykonywane 4 zapytania do bazy (pseudokod 19, linia 16) oraz następuje czterokrotne wywołanie funkcji `Continue3′` (linia 24) i jednokrotne funkcji `Correct3′Indel` (linia 28), złożoność obliczeniowa wynosi:

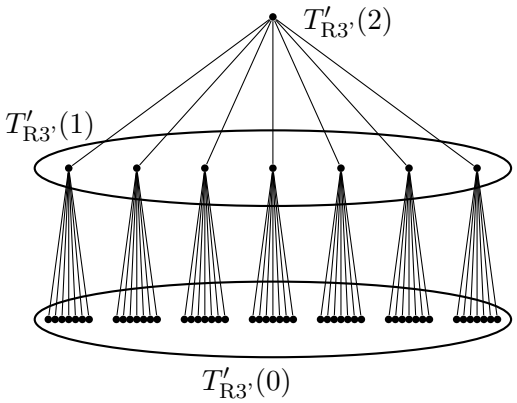
$$T_{\text{R3'}}(n_\psi, \theta_{\text{mx\_ind}}) = T''_{\text{R3'}}(n_\psi - 1, \theta_{\text{mx\_ind}}) + 4T''_{\text{R3'}}(n_\psi - 1, \theta_{\text{mx\_ind}}) + T''_{\text{R3'}}(n_\psi - 1, \theta_{\text{mx\_ind}} - 1) + 8,$$

co po uproszczeniu i podstawieniu równania funkcji $T'_{\text{R3'}}(\cdot)$, a także uwzględnieniu ograniczenia podjętych prób korekcji, które jest równe dopuszczalnej liczbie zapytań do bazy k-merów $\theta_{\text{MX\_CHCK}}$, przyjmuje postać przedstawioną w tezie twierdzenia. □

Należy zaznaczyć, że operacja poddawana zliczaniu, jako którą przyjęto wykonanie zapytania do bazy k-merów, nie musi być operacją wykonywaną największą liczbę razy. Przykładowo, zliczanie korekcji błędów substytucji (pseudokod 22, linia 37) jest przeprowadzane wielokrotnie dla każdej podjętej później próby korekcji błędu typu indel. Ze względu na fakt, że operacje tego typu wykonają się częściej co najwyżej o stały współczynnik (we wspomnianym przypadku co najwyżej $\theta_{\text{MX\_CHCK\_IND\_UNTRST}}$ więcej razy), w celu uproszczenia kwestia ta nie była brana pod uwagę.



(a) Zliczanie błędów typu indel rozłącznie, $T''_{\mathrm{R3'}}(1,1) = 20$



(b) Zliczanie błędów typu indel łącznie, $T'_{\mathrm{R3'}}(2) = 57$

Rysunek 5.7: Przykład górnego ograniczenia liczby wywołań funkcji $T''_{\mathrm{R3'}}(\cdot,\cdot)$ przy pomocy funkcji $T'_{\mathrm{R3'}}(\cdot)$ (wyłącznie dla korekcji błędów substytucji i delecji); punkty oznaczają kolejne wywołania funkcji `Continue3′`, a owale — grupy wywołań identycznie parametryzowanych funkcji

Lemat 16. *Pesymistyczna złożoność obliczeniowa rozszerzania końcowego regionu dla korekcji w kierunku 3′ algorytmu RECKONER wynosi:*

$$T_{\mathrm{Rre}}(\theta_{\mathrm{ext}}) = \frac{1}{3}\left(4^{\theta_{\mathrm{ext}}+1} - 4\right) = O\left(2^{\theta_{\mathrm{ext}}}\right). \tag{5.28}$$

Dowód. Analogiczny jak dla lematu 2, z tą różnicą, że rozpatrywany przypadek obejmują funkcje `Extend3′` oraz `ContExtend3′`, przedstawione na pseudokodach 34 oraz 35. Dopuszczenie niezaufanych k-merów w ścieżkach nie wpływa na uzyskaną złożoność pesymistyczną. □

Lemat 17. *Maksymalna liczba ścieżek algorytmu RECKONER wygenerowanych w wyniku korekcji regionu w kierunku 3′ jest nie większa niż:*

$$\pi_{\mathrm{R3'}}(n_\psi, \theta_{\mathrm{mx\_ind}}, \theta_{\mathrm{MX\_PTH}}) = \min\left(\theta_{\mathrm{MX\_PTH}}; 9 \cdot 8^{n_\psi + \theta_{\mathrm{mx\_ind}} - 1}\right). \tag{5.29}$$

Dowód. Wyznaczenie maksymalnej liczby jest utrudnione przez trudności matematyczne omówione w dowodzie lematu 15. W związku z tym zostanie podane tylko jej górne ograniczenie. Postać równania jest uzasadniona podobnie jak w dowodzie lematu 3, z tą różnicą, że jako rozmiar zbioru możliwych korekcji danej pozycji odczytu w funkcji `Continue3′` (pseudokody 19 oraz 20, linia 1) przyjęto wartość 8 ($|\Sigma| - 1 = 3$ symbole korekcji błędów substytucji, wyłączając symbol oryginalny, $|\Sigma| = 4$ symbole korekcji błędów delecji oraz 1 przypadek korekcji błędu insercji). Liczba uzyskanych w tej funkcji ścieżek korekcji jest równa liczbie $n_{\psi,\mathrm{mx\_ind}}$-elementowych wariacji z powtórzeniami ze zbioru 8-elementowego, tj. $\pi'_{\mathrm{R3'}}(n_{\psi,\mathrm{mx\_ind}}) = 8^{n_{\psi,\mathrm{mx\_ind}}}$, gdzie zmienna $n_{\psi,\mathrm{mx\_ind}}$ ma znaczenie analogicznie jak w dowodzie lematu 15.

Dodatkowo, uwzględniając fakt, że dopuszczalnych jest 9 kolejnych zmian w funkcji `Correct3'` (linia 31), tj. dodatkowo możliwe jest zachowanie oryginalnego symbolu, oraz uwzględniając ograniczenie liczby ścieżek do stałej $\theta_{\text{MX\_PTH}}$, ostateczna liczba ścieżek jest równa:

$$\begin{aligned}\pi_{\text{R3'}}(n_\psi, \theta_{\text{mx\_ind}}, \theta_{\text{MX\_PTH}}) = \\ = \min\left(\theta_{\text{MX\_PTH}}; 9\pi'_{\text{R3'}}(n_\psi + \theta_{\text{mx\_ind}} - 1)\right),\end{aligned}$$

co po podstawieniu wyrażeń oraz uproszczeniu przyjmuje postać przedstawioną w tezie twierdzenia. □

Należy zaznaczyć, że założenie 7 właściwie zmniejsza maksymalną liczbę ścieżek dla regionu ze względu na płynącą z niej konieczność wystąpienia takiej samej liczby korekcji błędów typu insercja i delecja. Mając jednak na celu ograniczenie liczby rozpatrywanych przypadków, w analizie pomięto tę obserwację.

W powyższych wyliczeniach nie uwzględniono rozszerzania odczytów, które w algorytmie RECKONER jest wykonywane wraz z tworzeniem ścieżki (np. linia 19 pseudokodu 19. Kwestia ta zostanie uwzględniona w kolejnych lematach i twierdzeniach.

Twierdzenie 4. *Pesymistyczna złożoność obliczeniowa korekcji w kierunku* $3'$ *algorytmu RECKONER wraz z rozszerzaniem wynosi:*

$$\begin{aligned}T_{\text{R3're}}(k, n_\psi, \theta_{\text{MX\_CHCK}}, \theta_{\text{MX\_E}}, \theta_{\text{MX\_IND}}, \theta_{\text{MX\_PTH}}) = \\ \min\left(\theta_{\text{MX\_CHCK}}; \frac{20}{3}\left(19^{n_\psi+\theta_{\text{MX\_IND}}-1} - 1\right) + \frac{5}{6}\left(19^{n_\psi+\theta_{\text{MX\_IND}}-2} - 1\right) + 8\right) + \\ \frac{1}{3}\min\left(\theta_{\text{MX\_PTH}}; 9 \cdot 8^{n_\psi+\theta_{\text{MX\_IND}}-1}\right)\left(4^{\theta_{\text{MX\_E}}} - 4\right) = \\ = O\left(\theta_{\text{MX\_CHCK}} + \theta_{\text{MX\_PTH}} 2^{\theta_{\text{MX\_E}}}\right).\end{aligned} \tag{5.30}$$

Dowód. Przypadek pesymistyczny jest zgodny z opisanym w dowodzie twierdzenia 1, z tą różnicą, że odwołania dotyczą odpowiednio lematów 15 oraz 17. Ponadto wyrażenia $T_{\text{R3'}}(\cdot,\cdot)$ oraz $\pi_{\text{R3'}}(\cdot,\cdot,\cdot)$ zostają sparametryzowane wartością $\theta_{\text{mx\_ind}} = \theta_{\text{MX\_IND}}$, ponieważ przypadek obejmuje korekcję całego regionu. Tym samym złożoność obliczeniowa jest równa:

$$\begin{aligned}T_{\text{R3're}}(k, n_\psi, \theta_{\text{MX\_CHCK}}, \theta_{\text{MX\_E}}, \theta_{\text{MX\_IND}}) = \\ T_{\text{R3'}}(n_\psi, \theta_{\text{MX\_IND}}) + \ \pi_{\text{R3'}}(\ell - i, n_\psi, \theta_{\text{MX\_IND}}) T_{\text{Rre}}(\theta_{\text{MX\_E}}),\end{aligned}$$

co po podstawieniu wyrażeń oraz uproszczeniu przyjmuje postać przedstawioną w tezie twierdzenia. □

Lemat 18. *Pesymistyczna złożoność obliczeniowa rozszerzania początkowego regionu w kierunku* $5'$ *algorytmu RECKONER wynosi:*

$$T_{\text{Rle}}(\theta_{\text{ext}}) = \frac{1}{3}\left(4^{\theta_{\text{ext}}+1} - 4\right) = O\left(2^{\theta_{\text{ext}}}\right). \tag{5.31}$$

Dowód. Analogiczny jak dla lematu 4, z tą różnicą, że rozpatrywany przypadek obejmują funkcje `Extend5′` oraz `ContExtend5′`, przedstawione na pseudokodach 61 oraz 62. □

Lemat 19. *Pesymistyczna złożoność obliczeniowa siłowej metody korekcji pierwszego k-meru algorytmu RECKONER wynosi:*

$$T_{\mathrm{R1b}}(\theta_{\mathrm{lq}}) = 4^{\theta_{\mathrm{lq}}} = O\left(2^{\theta_{\mathrm{lq}}}\right). \tag{5.32}$$

Dowód. Analogiczny jak dla lematu 5, z tą różnicą, że rozpatrywany przypadek obejmuje funkcję `FirstBruteForce` przedstawioną na pseudokodzie 32. □

Obecny w algorytmie warunek ograniczający liczbę ścieżek korekcji siłowej do $\theta_{\mathrm{MX\_FIRST\_PTHS}}$ nie ma wpływu na złożoność przedstawioną w lemacie 19, ponieważ w pesymistycznym przypadku limit ten nie zostanie osiągnięty, tzn. zostanie sprawdzonych wiele różnych ciągów korygujących symboli, ale mniej niż $\theta_{\mathrm{MX\_FIRST\_PTHS}}$ z nich spowoduje uzyskanie sekwencji k-meru obecnego w spektrum.

Lemat 20. *Pesymistyczna złożoność obliczeniowa korekcji pierwszego k-meru metodą modyfikacji kolejnych symboli algorytmu RECKONER wynosi:*

$$T_{\mathrm{R1c}}(k) = 3k = O(k). \tag{5.33}$$

Dowód. Analogiczny jak dla lematu 6, z tą różnicą, że rozpatrywany przypadek obejmuje funkcję `FirstConsec` przedstawioną na pseudokodzie 32. □

Lemat 21. *Pesymistyczna złożoność obliczeniowa korekcji błędów typu indel pierwszego k-meru algorytmu RECKONER wynosi:*

$$T_{\mathrm{R1i}}(k) = 5k - 5 = O(k). \tag{5.34}$$

Dowód. Rozpatrywany przypadek obejmuje funkcję `FirstIndel` przedstawioną na pseudokodzie 30. W pierwszej kolejności podejmowana jest próba usunięcia $k-1$ symboli (linia 41), a następnie wstawienia kolejnych $|\Sigma| = 4$ symboli na $k-1$ pozycjach (linia 49). Tym samym złożoność obliczeniowa przyjmuje postać przedstawioną w tezie lematu. □

Lemat 22. *Maksymalna liczba ścieżek uzyskanych w wyniku korekcji pierwszego k-meru metodą siłową algorytmu RECKONER wynosi:*

$$\pi_{\mathrm{R1b}}(\theta_{lq}) = \min\left(\theta_{\mathrm{MX\_FIRST\_PTHS}}; 4^{\theta_{lq}}\right). \tag{5.35}$$

Dowód. Sprawdzenie obecności w bazie k-merów efektu każdego ciągu modyfikacji może wiązać się z utworzeniem ścieżki. Tym samym liczba wygenerowanych ścieżek jest równa zależności jak w lemacie 19. Uwzględniając dodatkowo ograniczenie do $\theta_{\mathrm{MX\_FIRST\_PTHS}}$, liczba ścieżek jest równa zależności przedstawionej w tezie lematu. □

Lemat 23. *Maksymalna liczba ścieżek uzyskanych w wyniku korekcji pierwszego k-meru metodą modyfikacji kolejnych symboli algorytmu RECKONER wynosi:*

$$\pi_{\mathrm{R1c}}(k) = 3k. \tag{5.36}$$

Dowód. Analogiczny jak dla lematu 8. □

Lemat 24. *Maksymalna liczba ścieżek uzyskanych w wyniku korekcji błędów typu indel pierwszego k-meru algorytmu RECKONER wynosi:*

$$\pi_{\mathrm{R1i}}(k) = 5k - 5. \tag{5.37}$$

Dowód. Sprawdzenie obecności w bazie k-merów efektu każdego przypadku modyfikacji może wiązać się z utworzeniem ścieżki. Tym samym liczba wygenerowanych ścieżek jest równa zależności jak w lemacie 21, przedstawionej w odpowiedniej formie w tezie twierdzenia. □

Lemat 25. *W algorytmie RECKONER dana pozycja $a \in \{0, 1, \ldots, k-1\}$ pełni rolę pierwszego zmodyfikowanego symbolu w maksymalnie:*

$$\pi_{\mathrm{R1c}}(a, k) = 3 \tag{5.38}$$

ścieżkach w korekcji pierwszego k-meru metodą modyfikacji kolejnych symboli.

Dowód. Analogiczny jak dla lematu 9. □

Lemat 26. *W algorytmie RECKONER dana pozycja $a \in \{1, 2, \ldots, k-1\}$ pełni rolę pierwszego zmodyfikowanego symbolu w maksymalnie:*

$$\pi_{\mathrm{R1i}}(a, k) = 5. \tag{5.39}$$

ścieżkach w korekcji błędów typu indel pierwszego k-meru.

Dowód. Każda ścieżka korekcji obejmuje modyfikację jednej pozycji $a \in \{1, 2, \ldots, k-1\}$, tzn. wyłączając pozycję $a = 0$. Dla tej pozycji w ścieżce może zostać wykonana korekcja polegająca na usunięciu tego symbolu albo wstawieniu bezpośrednio wcześniej jednego symbolu ze zbioru Σ. Zatem modyfikacja każdej pozycji może zostać przeprowadzona na 5 sposobów. □

Lemat 27. *Pesymistyczna złożoność obliczeniowa rozszerzania początkowego regionu dla wszystkich ścieżek uzyskanych w metodzie siłowej korekcji pierwszego k-meru odczytu algorytmu RECKONER wynosi:*

$$\begin{aligned} T_{\mathrm{R1ble}}(\theta_{\mathrm{MX\_E}}, \theta_{\mathrm{MX\_FIRST\_PTHS}}, \theta_{\mathrm{MX\_LQ}}) = \\ = \frac{1}{3} \min\left(\theta_{\mathrm{MX\_FIRST\_PTHS}}; 4^{\theta_{\mathrm{MX\_LQ}}}\right)\left(4^{\theta_{\mathrm{MX\_E}}+1} - 4\right) = \\ = O\left(\theta_{\mathrm{MX\_FIRST\_PTHS}} 2^{\theta_{\mathrm{MX\_E}}}\right). \end{aligned} \tag{5.40}$$

Dowód. Analogiczny jak dla lematu 10, wykorzystując założenie 2, w oparciu o lematy odpowiednio 18 oraz 22. □

Lemat 28. *Pesymistyczna złożoność obliczeniowa rozszerzania początkowego regionu dla wszystkich ścieżek uzyskanych w wyniku korekcji błędów typu indel pierwszego k-meru odczytu algorytmu RECKONER wynosi:*

$$\begin{aligned} T_{\mathrm{R1ile}}(k, \theta_{\mathrm{MX\_E}}) = \\ = \frac{20}{3}\left[(k - \theta_{\mathrm{MX\_E}} - 1)\left(4^{\theta_{\mathrm{MX\_E}}} - 1\right) - \theta_{\mathrm{MX\_E}} - \frac{1}{3}4^{\theta_{\mathrm{MX\_E}}} - \frac{1}{3}\right] = \\ = O\left(k2^{\theta_{\mathrm{MX\_E}}}\right). \end{aligned} \tag{5.41}$$

Dowód. Rozpatrywany przypadek obejmuje pseudokod 29, gdy wykonano kod w linii 30, a w rezultacie uzyskano maksymalną możliwą liczbę ścieżek. Pesymistyczna złożoność obliczeniowa dotyczy sytuacji, w której dla każdej ścieżki region jest rozszerzany o maksymalną liczbę pozycji. Ze względu na założenie 1 liczba pozycji rozszerzających odczyt dla danej ścieżki może być różna, zgodnie z sytuacją symetryczną do przedstawionej na rys. 5.2. Spośród ścieżek wymagających rozszerzenia regionu $k - \theta_{\mathrm{MX\_E}} - 1$ (analogicznie do sytuacji symetrycznej jak na rys. 5.2, z tym że początkowa pozycja odczytu, w związku z warunkami pętli na pseudokodzie 30 w liniach 39 oraz 46 w tym przypadku nie będzie modyfikowana) ścieżek zostanie rozszerzonych o $\theta_{\mathrm{MX\_E}}$ symboli, natomiast $(\theta_{\mathrm{MX\_E}} - 1)$ będzie rozszerzonych o kolejno $1, 2, \ldots, \theta_{\mathrm{MX\_E}} - 1$ symboli. Tym samym, opierając się na lematach 18 oraz 26, złożoność obliczeniowa jest równa sumie:

$$\begin{aligned} T_{\mathrm{R1ile}}(k, \theta_{\mathrm{MX\_E}}) = \\ = \sum_{i=1}^{k-\theta_{\mathrm{MX\_E}}-1} \pi_{\mathrm{R1i}}(i-1, k) T_{\mathrm{Rle}}(\theta_{\mathrm{MX\_E}}) + \sum_{i=1}^{\theta_{\mathrm{MX\_E}}-1} \pi_{\mathrm{R1i}}(k - \theta_{\mathrm{MX\_E}} + i - 1, k) T_{\mathrm{Rle}}(i), \end{aligned}$$

co po podstawieniu wyrażeń oraz uproszczeniu przyjmuje postać przedstawioną w tezie. □

W korekcji pierwszego k-meru można wyróżnić cztery przypadki:

(*i*) liczba pozycji niskiej jakości $\theta_{\mathrm{lq}} = 0$ i oryginalny pierwszy k-mer nie jest zaufany; w efekcie zostaje wykonana tylko korekcja metodą modyfikacji kolejnych symboli (pseudokod 29, linia 22);

(*ii*) liczba pozycji niskiej jakości $\theta_{\mathrm{lq}} \in (0; \theta_{\mathrm{MX\_LQ}}]$; w efekcie zostaje przeprowadzona korekcja metodą siłową, w wyniku której zostaje uzyskana co najmniej jedna ścieżka korekcji (linia 13);

(*iii*) liczba pozycji niskiej jakości należy do tego przedziału, jednak korekcja metodą siłową nie zwraca żadnej ścieżki; w efekcie zostanie wykonana korekcja przez modyfikację kolejnych symboli, tzn. zostaje przeprowadzona korekcja obydwoma metodami (linie 13 oraz 15);

(*iv*) liczba pozycji niskiej jakości $\theta_{\mathrm{lq}} > \theta_{\mathrm{MX\_LQ}}$; w efekcie zostaje przeprowadzona korekcja metodą siłową dla $\theta_{\mathrm{MX\_LQ}}$ symboli najniższej jakości (linia 25).

Ponadto jeżeli dowolny z tych przypadków nie powiedzie się, zostanie wykonana korekcja błędów typu indel (linia 30).

Analogicznie jak dla algorytmu BLESS, zostaną wybrane przypadki, dla których korekcja całego odczytu będzie charakteryzowała się najwyższą złożonością obliczeniową.

Maksymalna liczba wygenerowanych ścieżek w przypadkach (*i*) oraz (*iii*) jest taka sama. Maksymalna złożoność obliczeniowa przypadku (*iii*) jest większa niż przypadku (*i*), pomimo wykonania w tym ostatnim jednego dodatkowego zapytania do bazy k-merów (linia 18). Maksymalna liczba wygenerowanych ścieżek w przypadkach (*ii*) oraz (*iv*) jest taka sama. Maksymalna złożoność obliczeniowa przypadku (*iv*) jest większa niż przypadku (*ii*), ze względu na wykonanie wspomnianego dodatkowego zapytania do bazy.

W związku z tym dalszej analizie powinny zostać poddane przypadki (*iii*) oraz (*iv*), uwzględniając wygenerowanie ewentualnych ścieżek w wyniku korekcji błędów typu indel. W przypadku (*iii*) maksymalna liczba uzyskanych ścieżek wynosi zgodnie z lematem 23 $\pi_{\mathrm{R1c}}(k) = 3k$, w związku z tym pesymistyczny przypadek korekcji odczytu ma miejsce, gdy w przypadku tym nie zostanie uzyskana żadna ścieżka i zostanie przeprowadzona korekcja błędów typu indel. Złożoność obliczeniowa będzie wtedy większa, a jednocześnie maksymalna liczba uzyskanych ścieżek (a zatem także liczba prób korekcji pozostałej części odczytu) wyniesie zgodnie z lematem 24 $\pi_{\mathrm{R1i}}(k) = 5k - 5$. Dla przypadku (*iv*) zostanie przyjęta możliwość wygenerowania maksymalnej liczby ścieżek, zatem korekcja błędów typu indel w ogóle nie zostanie wykonana. W ten sposób w porównaniu zostanie uwzględniony także potencjalnie niekorzystny przypadek wygenerowania maksymalnej liczby ścieżek w wyniku korekcji pierwszego k-meru metodą siłową. Przypadek (*iii*) zostanie objęty lematem 29, a przypadek (*iv*) — lematem 30.

Wyznaczenie złożoności obliczeniowej korekcji całego odczytu

Lemat 29. *Pesymistyczna złożoność obliczeniowa korekcji całego odczytu, w przypadku gdy jest wykonywana korekcja pierwszego k-meru metodą siłową, metodą modyfikacji kolejnych symboli oraz gdy następuje korekcja błędów typu indel, al-*

gorytmu RECKONER wynosi:

$$\begin{aligned}
&T_{\mathrm{Ri}}(k,\ell,\theta_{\mathrm{MX_CHCK}},\theta_{\mathrm{MX_E}},\theta_{\mathrm{MX_FIRST_PTHS}},\theta_{\mathrm{MX_IND}},\theta_{\mathrm{MX_LQ}}) = \\
&= 4^{\theta_{\mathrm{MX_LQ}}} + \frac{20}{3}\left[(k-\theta_{\mathrm{MX_E}}-1)\left(4^{\theta_{\mathrm{MX_E}}}-1\right) - \theta_{\mathrm{MX_E}} - \frac{1}{3}4^{\theta_{\mathrm{MX_E}}}\right] + \\
&+(5k-5)\cdot \\
&\cdot\left[\min\left(\theta_{\mathrm{MX_CHCK}};\frac{20}{3}\left(19^{\ell-k+\theta_{\mathrm{MX_IND}}-1}-1\right)+\frac{5}{6}\left(19^{\ell-k+\theta_{\mathrm{MX_IND}}-2}-1\right)+8\right)+\right. \\
&\left.\frac{1}{3}\min\left(\theta_{\mathrm{MX_PTH}};9\cdot 8^{\ell-k+\theta_{\mathrm{MX_IND}}-1}\right)\left(4^{\theta_{\mathrm{MX_E}}}-4\right)\right] + 8k - \frac{65}{9} = \\
&= O\left(2^{\theta_{\mathrm{MX_LQ}}} + (k+\theta_{\mathrm{MX_PTH}})2^{\theta_{\mathrm{MX_E}}} + k\theta_{\mathrm{MX_CHCK}}\right).
\end{aligned} \tag{5.42}$$

Dowód. Rozpatrywany przypadek obejmuje pseudokod 29. Składnikami złożoności obliczeniowej są: złożoność korekcji pierwszego k-meru wykonana najpierw metodą siłową (w wyniku której nie uzyskuje się ścieżek — linia 13), następnie metodą modyfikacji kolejnych symboli (linia 15) (również nie prowadząc do uzyskania żadnych ścieżek), a ostatecznie korekcji błędów typu indel (linia 30), złożoność rozszerzania regionu początkowego w lewo oraz korekcja pozostałej części odczytu (tj. regionu długości $n_\psi = \ell - k$ — linia 35), wykonana dla wszystkich uzyskanych w wyniku korekcji pierwszego k-meru ścieżek (wyłącznie korekcji błędów typu indel). Opierając się na lematach 19–21, 24 oraz 28, a także twierdzeniu 4 sumarycznie wynosi ona:

$$\begin{aligned}
&T_{\mathrm{Ri}}(k,\ell,\theta_{\mathrm{MX_CHCK}},\theta_{\mathrm{MX_LQ}},\theta_{\mathrm{MX_E}},\theta_{\mathrm{MX_IND}},\theta_{\mathrm{MX_FIRST_PTHS}}) = \\
&= T_{\mathrm{R1b}}(\theta_{\mathrm{MX_LQ}}) + T_{\mathrm{R1c}}(k) + T_{\mathrm{R1i}}(k) + T_{\mathrm{R1ile}}(k,\theta_{\mathrm{MX_E}}) + \\
&+ \pi_{\mathrm{R1i}}(k)T_{\mathrm{R3're}}(k,\ell-k,\theta_{\mathrm{MX_CHCK}},\theta_{\mathrm{MX_E}},\theta_{\mathrm{MX_IND}},\theta_{\mathrm{MX_FIRST_PTHS}}),
\end{aligned}$$

co po podstawieniu wyrażeń oraz uproszczeniu przyjmuje postać jak w tezie lematu. □

Lemat 30. *Pesymistyczna złożoność obliczeniowa korekcji całego odczytu, w przypadku gdy jest wykonywana korekcja pierwszego k-meru metodą siłową, algorytmu*

RECKONER wynosi:

$$\begin{aligned} T_{\mathrm{Rb}}(k,\ell,\theta_{\mathrm{MX\_CHCK}},\theta_{\mathrm{MX\_E}},\theta_{\mathrm{MX\_FIRST\_PTHS}},\theta_{\mathrm{MX\_IND}},\theta_{\mathrm{MX\_LQ}}) = \\ = \min\left(\theta_{\mathrm{MX\_FIRST\_PTHS}};4^{\theta_{\mathrm{MX\_LQ}}}\right)\cdot \\ \cdot\left[\frac{1}{3}\left(4^{\theta_{\mathrm{MX\_E}}+1}-4\right)+\right. \\ +\min\left(\theta_{\mathrm{MX\_LQ}};\frac{20}{3}\left(19^{\ell-k+\theta_{\mathrm{MX\_IND}}-1}-1\right)+\frac{5}{6}\left(19^{\ell-k+\theta_{\mathrm{MX\_IND}}-2}-1\right)+8\right)+ \\ \left.+\frac{1}{3}\min\left(\theta_{\mathrm{MX\_PTH}};9\cdot 8^{\ell-k+\theta_{\mathrm{MX\_IND}}-1}\right)\left(4^{\theta_{\mathrm{MX\_E}}}-4\right)\right]+4^{\theta_{\mathrm{MX\_LQ}}}+1 = \\ = O\left(\theta_{\mathrm{MX\_FIRST\_PTHS}}\cdot\left(\theta_{\mathrm{MX\_PTH}}2^{\theta_{\mathrm{MX\_E}}}+2^{\theta_{\mathrm{MX\_LQ}}}\right)\right). \end{aligned} \tag{5.43}$$

Dowód. Analogiczny jak dla lematu 12, w oparciu o lematy 19, 22 oraz 27, a także twierdzenie 4. Dodatkowo uwzględniono jedno zapytanie w linii 18 pseudokodu 29. □

Określenie pesymistycznej złożoności obliczeniowej wymaga porównania złożoności stanowiących przedmiot lematów 29 oraz 30. Złożoności te są jednak zależne od siedmiu parametrów: k, ℓ, $\theta_{\mathrm{MX\_CHCK}}$, $\theta_{\mathrm{MX\_E}}$, $\theta_{\mathrm{MX\_FIRST\_PTHS}}$, $\theta_{\mathrm{MX\_IND}}$, $\theta_{\mathrm{MX\_LQ}}$, wobec tego rozwiązanie takiej nierówności jest zadaniem trudnym. W celu uproszczenia rozwiązania zmienne $\theta_{\mathrm{MX\_CHCK}}$, $\theta_{\mathrm{MX\_E}}$, $\theta_{\mathrm{MX\_FIRST\_PTHS}}$, $\theta_{\mathrm{MX\_IND}}$ oraz $\theta_{\mathrm{MX\_LQ}}$ zostaną zastąpione wartościami, które stanowią stałe parametry implementacji algorytmu, zgodnie z tabelami 5.4 oraz 5.5.

Lemat 31. *Pesymistyczna złożoność obliczeniowa korekcji algorytmem RECKONER całego odczytu jest maksymalna podczas korekcji pierwszego k-meru metodą siłową, metodą modyfikacji kolejnych symboli oraz korekcji błędów indel.*

Dowód. W celu udowodnienia lematu należy rozwiązać nierówność obejmującą korekcję całego odczytu w obu przypadkach, których złożoności wyznaczono w dowodach lematów 29 oraz 30:

$$\begin{aligned} &T_{\mathrm{Rb}}(k,\ell,\theta_{\mathrm{MX\_CHCK}},\theta_{\mathrm{MX\_E}},\theta_{\mathrm{MX\_FIRST\_PTHS}},\theta_{\mathrm{MX\_IND}},\theta_{\mathrm{MX\_LQ}}) > \\ &> T_{\mathrm{Ri}}(k,\ell,\theta_{\mathrm{MX\_CHCK}},\theta_{\mathrm{MX\_E}},\theta_{\mathrm{MX\_FIRST\_PTHS}},\theta_{\mathrm{MX\_IND}},\theta_{\mathrm{MX\_LQ}}). \end{aligned}$$

W trakcie rozwiązywania czterokrotnie należy podjąć decyzję odnośnie do wyboru wartości funkcji $\min(\cdot,\cdot)$, wykorzystywanych w równaniach wspomnianych lematów. W każdym przypadku jako jej wynik zostały wybrane odpowiednie stałe, ponieważ przy przyjętym założeniu 4 alternatywne wartości byłyby większe.

Podstawienie wartości oraz rozwiązanie równania powoduje uzyskanie zależności $k <\sim 1{,}13$. Tym samym powyższe równanie jest prawdziwe tylko dla $k = 1$, co na mocy założenia 3 jest niemożliwe, zatem w żadnym przypadku złożoność

obliczeniowa korekcji całego odczytu, gdy pierwszy k-mer jest korygowany metodą siłową, nie może być maksymalny. Stąd maksymalną złożonością cechuje się przypadek, gdy dla pierwszego k-meru jest wykonywana korekcja błędów typu indel. □

Twierdzenie 5. Pesymistyczna złożoność obliczeniowa korekcji odczytu algorytmu RECKONER wynosi:

$$\begin{aligned} T'_{\mathrm{R}}(k,\ell,\theta_{\mathrm{MX\_CHCK}},\theta_{\mathrm{MX\_E}},\theta_{\mathrm{MX\_FIRST\_PTHS}},\theta_{\mathrm{MX\_IND}},\theta_{\mathrm{MX\_LQ}}) = \\ = T_{\mathrm{Ri}}(k,\ell,\theta_{\mathrm{MX\_CHCK}},\theta_{\mathrm{MX\_E}},\theta_{\mathrm{MX\_FIRST\_PTHS}},\theta_{\mathrm{MX\_IND}},\theta_{\mathrm{MX\_LQ}}) = \\ = O(k). \end{aligned} \tag{5.44}$$

gdzie wartość $T_{\mathrm{Ri}}(k,\ell,\theta_{\mathrm{MX\_CHCK}},\theta_{\mathrm{MX\_E}},\theta_{\mathrm{MX\_FIRST\_PTHS}},\theta_{\mathrm{MX\_IND}},\theta_{\mathrm{MX\_LQ}})$ jest zdefiniowana w równaniu (5.42).

Dowód. Jak zostało wspomniane, w przypadku pesymistycznym w wyniku wyznaczania regionów nie zostają wyznaczone żadne prawidłowe regiony lub istnieją błędne regiony krótsze niż k. Złożoność obliczeniowa korekcji całego odczytu została oparta na obserwacji wyszczególnionej w dowodzie lematu 31. Ze względu na postawione w tym lemacie założenie o stałości wartości $\theta_{\mathrm{MX\_CHCK}}$, $\theta_{\mathrm{MX\_E}}$, $\theta_{\mathrm{MX\_FIRST\_PTHS}}$, $\theta_{\mathrm{MX\_IND}}$ oraz $\theta_{\mathrm{MX\_LQ}}$ oraz ze względu na założenie 4, rzędy złożoności występujące w niniejszym twierdzeniu ulegają uproszczeniu w stosunku do przedstawionych w lemacie 29. □

Twierdzenie 6. *Pesymistyczna złożoność obliczeniowa korekcji odczytu algorytmu RECKONER z uwzględnieniem wykonania zapytania do bazy k-merów, przy przyjęciu jako operacji dominującej porównania symbolu k-meru z symbolem obecnym w bazie, wynosi:*

$$\begin{aligned} T_{\mathrm{R}}(k,\ell,\theta_{\mathrm{MX\_CHCK}},\theta_{\mathrm{MX\_E}},\theta_{\mathrm{MX\_FIRST\_PTHS}},\theta_{\mathrm{MX\_IND}},\theta_{\mathrm{MX\_LQ}}) = \\ = kT_{\mathrm{Ri}}(k,\ell,\theta_{\mathrm{MX\_CHCK}},\theta_{\mathrm{MX\_E}},\theta_{\mathrm{MX\_FIRST\_PTHS}},\theta_{\mathrm{MX\_IND}},\theta_{\mathrm{MX\_LQ}}) = \\ = O(k^2). \end{aligned} \tag{5.45}$$

Dowód. Na mocy założenia 10 każde zapytanie do bazy k-merów jest wykonywane ze złożonością $T_{\mathrm{kmc}}(k) = k$. W oparciu o twierdzenie 5 złożoność przyjmuje postać jak w tezie twierdzenia. □

5.5.1 Podsumowanie

Podsumowanie działania algorytmu RECKONER zostanie przeprowadzone w odniesieniu do analogicznych kwestii dotyczących algorytmu BLESS:

- algorytm nie jest oparty na filtrach Blooma, co eliminuje wszystkie niedogodności tej struktury danych; zastosowane baza k-merów KMC wymaga

jednak dodatkowego czasu na pracę zewnętrznych narzędzi tworzących i filtrujących ją oraz może wymagać wyższego zapotrzebowania na pamięć, zatem kwestie te zostaną przeanalizowane eksperymentalnie,

- algorytm dostosowano do przetwarzania równoległego, a podczas projektowania i implementacji zwrócono uwagę na szybkość działania; skuteczność tych działań zostanie zweryfikowana eksperymentalnie,
- heurystyczne przesłanki detekcji i korekcji błędów zostały oparte zarówno na spektrum k-merów, jak też na współczynnikach jakości odczytów — zwrócono także uwagę na niedoskonałość koncepcji spektrum, m.in. pozwalając na incydentalne wprowadzanie do odczytów k-merów spoza spektrum,
- częściowo zapewniono strategię rozwiązywania problemu niskiego pokrycia niektórych odczytów poprzez wspomniane dopuszczenie wprowadzania k-merów spoza spektrum oraz opcjonalną weryfikację w oparciu k''-mery,
- zredukowano pesymistyczną złożoność czasową algorytmu poprzez zastosowanie limitowania wybranych zbiorów danych oraz liczby analizowanych przypadków; pesymistyczna złożoność obliczeniowa korekcji jednego odczytu wynosi $O(k^2)$.

Pozostałe obserwacje, w szczególności odniesienie do wymagań postawionych w podrozdziale 5.4.2, zostaną omówione w rozdziale przedstawiającym wyniki eksperymentalne.

Należy podkreślić, że przedstawione w bieżącym rozdziale wyznaczanie złożoności obliczeniowych obu algorytmów jest obciążone licznymi uproszczeniami, wynikającymi przede wszystkim z przyjęcia wielu założeń. Ponadto prawdopodobieństwo wystąpienia sytuacji pesymistycznej lub zbliżonej do niej jest niepraktycznie małe. Bardzo duża liczba odczytów obecnych w typowym zestawie odczytów powoduje, że pojedyncze niekorzystne przypadki korekcji odczytów nie powinny wywrzeć znaczącego wpływu na rozwiązanie całości zadania. Dodatkowo w przypadku algorytmu RECKONER znaczna redukcja złożoności obliczeniowej została przeprowadzona poprzez wprowadzenie sztywnych limitów weryfikowanych możliwości, a nie wprowadzenie kluczowych zmian w algorytmie wyszukiwania wyczerpującego, którego złożoność bez tych ograniczeń cały czas pozostaje wykładnicza w funkcji długości odczytu.

Pomimo powyższych intuicyjnych wniosków nieprzedstawione obserwacje przebiegu pracy algorytmu RECKONER przed i po wprowadzeniu limitowania wskazują, że limitowanie spełnia swoją rolę. Jego brak w rzadkich przypadkach powodował bardzo długi czas korekcji niektórych odczytów, sięgający minut (na komputerze zgodnym ze specyfikacją podaną w tabeli B.3). Tego rodzaju zjawisko niejednokrotnie powodowało brak pamięci w związku ze zbyt dużą liczbą ścieżek. Z kolei czas obliczeń również jest redukowany na skutek limitowania, ale raczej

nie ze względu na wspomniane „trudne” odczyty, ale liczne odczyty, których korekcja była przeprowadzana nieprzeciętnie długo, ale jednocześnie znacznie krócej od tych, które wymagały najwięcej czasu.

Rozdział 6

Badania eksperymentalne

W podrozdziale 4.4 podkreślono, że do tej pory nie zaproponowano satysfakcjonującej, uniwersalnej metody oceny jakości algorytmów korekcji. Zarówno kryteria oceny, miary kryteriów, zasady doboru zbioru porównywanych algorytmów oraz ich parametrów, jak też wykorzystane zestawy danych testowych w dużej mierze różnią się w większości pozycji literaturowych. Stanowi to z jednej strony źródło różnorodnych, choć nie zawsze spójnych (ze względu na odmienność kryteriów) rezultatów, a z drugiej prezentuje mnogość możliwych podejść do problemu oraz dostarcza wyników odnoszących się do dużego spektrum aspektów. W niniejszym rozdziale przedstawiono wyniki eksperymentów stanowiących podstawę analizy porównawczej wybranej grupy algorytmów korekcji odczytów Illumina oraz eksperymentów mających na celu ocenę jakości algorytmu RECKONER. W eksperymentach częściowo wykorzystano miary obecne w pozycjach literaturowych. Wprowadzono też nowe kryteria, w tym odnoszące się do aspektów dotąd nie poruszanych przez autorów prac.

6.1 Założenia eksperymentalne

6.1.1 Cele eksperymentów

Celem eksperymentów było przeprowadzenie analizy zapotrzebowania na zasoby oraz jakości wyników poszczególnych algorytmów. Zwrócono uwagę na zastosowanie różnorodnych danych wejściowych (obejmujących organizmy o genomach charakteryzujących się różnymi cechami, w szczególności długością), odczyty różnej długości i właściwości (typowe dla sekwenatorów Illumina HiSeq, MiSeq oraz NovaSeq). Celem było także porównanie wyników wyrażonych przy pomocy rozmaitych miar. Uwzględniono odczyty rzeczywiste (co, jak zostało wyjaśnione w podrozdziale 4.4.1, stanowi pośrednią miarę jakości korekcji, jednak

odnoszącą się do realnych problemów, a nie opartą na wyidealizowanych przypadkach uzyskanych w wyniku symulacji). Jednocześnie nie zrezygnowano z analizy symulowanych zestawów danych, uważając ich korekcję za źródło bezpośredniej informacji o własnościach rezultatów pracy algorytmów dla danych różnej jakości, a także dając możliwość zweryfikowania skuteczności takiego podejścia do oceny.

Nacisk położono również na zbadanie szybkości działania oraz zapotrzebowania na pamięć implementacji, a także skalowalności przetwarzania równoległego. Część eksperymentów została przeprowadzona w celu obserwacji wpływu długości oligomeru k na uzyskana rezultaty, w tym opracowanie strategii doboru tego parametru dla algorytmu RECKONER. Wyniki podzielono zgodnie z kategoriami kryteriów, które zostały przedstawione w podrozdziale 4.4.

6.1.2 Dobór algorytmów

Spośród 11 algorytmów dokładnie opisanych w podrozdziale 4.1.3, do eksperymentów dobrano 9 z nich. Do tej grupy nie zaliczono algorytmu Quake ze względu na skupienie się autorów tego algorytmu na eksperymentach z użyciem bardzo krótkich odczytów długości 36 pz, co sugeruje przeznaczenie algorytmu do korekcji odczytów uzyskiwanych przez dawne modele sekwenatorów. We wstępnych eksperymentach stwierdzono też bardzo długi czas obliczeń, gdy wejście algorytmu stanowiły odczyty większej długości. Ponadto w pracach [89, 97, 179] sygnalizowano słabą jakość wyników tego algorytmu oraz obecność błędów.

Nie przeprowadzono eksperymentów z algorytmem PREMIER, ze względu na brak publicznie dostępnej jego implementacji, a także liczne, trudne do określenia parametry oraz przeprowadzenie oryginalnych testów dla odczytów długości 36 pz. Brak implementacji był również przyczyną pominięcia eksperymentów z jego udziałem opisanych w pracy [97]; na kwestię niedostępności implementacji zwrócono również uwagę w pracy [123].

W przypadku algorytmu BLESS zrezygnowano z eksperymentów z wykorzystaniem implementacji w wersji 0.12, stanowiącej podstawę algorytmu RECKONER, ze względu na jej jednowątkową pracę, skutkującą bardzo dużym czasem wykonania oraz znaczną nieaktualność tej wersji (wydanej w 2013 roku). W zamian do eksperymentów zakwalifikowano najnowszą wersję 1.02. Należy zauważyć, że w pracy [210] całkowicie pominięto w eksperymentach algorytm BLESS, uzasadniając to trudnościami występującymi podczas jego instalacji, wynikającymi zapewne z wykorzystania biblioteki do przetwarzania zgodnie z modelem sieciowym.

W tabeli B.1 zawarto informacje dotyczące testowanych algorytmów korekcji. Ponadto w tabeli B.2 wyszczególniono pomocnicze narzędzia i algorytmy wykorzystane do przeprowadzenia eksperymentów. W pierwszej z nich zostały wymienione algorytmy, które podlegały eksperymentom z pomiarem zapotrzebo-

wania na pamięć i czas obliczeń. Dokładne zastosowanie pomocniczych narzędzi i algorytmów zostało omówione w dalszej części rozdziału.

Ze względu na testowanie algorytmu RECKONER w dwóch trybach: zasadniczym oraz z weryfikacją w oparciu o k''-mery przyjęto, że słupki bądź wykresy obejmujące algorytm RECKONER zostaną oznaczone kolorem jasnoczerwonym (tryb zasadniczy) albo ciemnoczerwonym (tryb z weryfikacją).

6.1.3 Uruchomienie implementacji

Kompilacja

Większość algorytmów została zaimplementowana w języku C++. Ich kompilacja została wykonana przy pomocy kompilatora G++ w wersji 7.2.0, zachowując domyślne parametry kompilacji. Wyjątkiem były:

- BLESS — w celu dostarczenia implementacji interfejsu MPI zastosowano bibliotekę OpenMPI; kompilacja została wykonana przy pomocy załączonego do biblioteki kompilatora mpicxx oraz kompilatora G++ w wersji 4.9.2; uruchomienie implementacji skompilowanej przy pomocy nowszego kompilatora oraz nowszej biblioteki nie powiodło się,
- Blue — została wykorzystana wersja skompilowana przez autorów algorytmu; implementacja jest dostępna w języku C#,
- SAMDUDE — implementacja algorytmu jest dostępna w języku Python, nie wymagając kompilacji.

Środowisko testowe

W celu przeprowadzenia eksperymentów wykorzystano serwer obliczeniowy, którego parametry zostały przedstawione w tabeli B.3.

Skompilowana implementacja algorytmu Blue wymaga do pracy środowiska uruchomieniowego Microsoft .NET Framework, w związku z czym została ona uruchomiona w środowisku Mono. Z kolei implementacja algorytmu SAMDUDE została uruchomiona w środowisku Python. Algorytm BLESS, ze względu na wykorzystanie interfejsu MPI, został uruchomiony w środowisku z dostępną biblioteką OpenMPI. Pozostałe implementacje zostały uruchomione jako samodzielne pliki wykonywalne w środowisku powłoki bash.

Pomiarów zapotrzebowania na zasoby obliczeniowe dokonano przy pomocy narzędzia GNU time, pozwalającego na uruchomienie programu z wykonaniem pomiaru szeregu wartości, jak czasu obliczeń (w tym pełnego czasu od uruchomienia do zakończenia pracy programu — *wall clock*) oraz szczytowego zapotrzebowania na pamięć. W celu uniknięcia wpływu na czas obliczeń buforowania plików przez system operacyjny przy uruchomieniu różnych algorytmów korekcji

z rzędu, przed uruchomieniem każdego przypadku testowego (tj. jednego uruchomienia implementacji algorytmu lub jego potoku przetwarzania) każdorazowo przeprowadzono usunięcie plików z pamięci podręcznej systemu przy pomocy narzędzia vmtouch. Czas korekcji, asemblacji i mapowania każdego uruchomienia algorytmu został ograniczony do 24 godzin, z wyjątkiem eksperymentów oceny skalowalności, gdzie czas ten zwiększono do 48 godzin.

Analiza jakościowa wyników (jeśli nie zaznaczono inaczej), a także symulacja uproszczoną metodą Quake, zostały przeprowadzone przy pomocy przygotowanych *ad hoc* narzędzi. Szczegółowy sposób uruchomienia zewnętrznych narzędzi został zawarty w podrozdziale B.2.

Uwagi techniczne do niektórych algorytmów

Implementacja algorytmu BLESS w wersji 1.02 jest zintegrowana z narzędziem KMC w wersji 2.1.1, które posiada błąd skutkujący w niektórych środowiskach zakleszczeniem algorytmu. Implementacja KMC została zmodyfikowana przez autorów algorytmu BLESS w sposób umożliwiający wykonanie jej na wielu węzłach obliczeniowych, stąd zmiana narzędzia zliczającego k-mery na wersję działającą prawidłowo jest zadaniem skomplikowanym. W celu ominięcia ujawnienia się błędu, algorytm BLESS był uruchamiany z parametrem zwiększającym ilość pamięci dostępnej dla KMC z 4 GiB do 8 GiB. W takiej konfiguracji problem nie występował (choć nie wyeliminowało to problemu całkowicie), jednak mogło to wywrzeć pewien negatywny wpływ na zapotrzebowanie na pamięć całego potoku algorytmu BLESS.

Algorytm BLESS został przetestowany przy pomocy jednego węzła obliczeniowego, pracującego zgodnie z modelem z pamięcią wspólną. Nie podjęto próby uruchomienia na klastrze ze względu na brak odpowiedniego środowiska obliczeniowego.

Do uruchomienia algorytmu Fiona wykorzystano implementację będącą składnikiem pakietu SeqAn. Z kolei implementacja algorytmu Blue jest dostępna w postaci trzech narzędzi: zliczania k-merów (Tessel), generacji *par* k'-merów (GenerateMerPairs), właściwej korekcji (Blue). Narzędzia te były uruchamiane kolejno.

Implementacja algorytmu SAMDUDE wymaga przekazania jako wejścia zmapowanych odczytów w formacie SAM. Mapowanie przeprowadzono przy pomocy narzędzia BWA [129], poprzez wykonanie go — zgodnie z rekomendacją autorów — w trybie BWA-MEM. Plik w formacie SAM stanowi także wyjście algorytmu SAMDUDE, dlatego w przypadkach gdy eksperyment wymagał, żeby uzyskane wyniki korekcji posiadały format FASTQ (tj. na potrzeby asemblacji, mapowania oraz oceny odczytów symulowanych), przeprowadzono wyodrębnienie ich przy pomocy narzędzia SAMtools [130].

Pozostałe algorytmy przyjmują jako wejście pliki w formacie FASTQ. Jeśli było to możliwe, pliki te zawierały odczyty sparowane (wykorzystano parę pli-

ków) w skompresowanym formacie GZIP. Wyjątkiem były implementacje Blue oraz Karect, które nie obsługują plików skompresowanych, wobec tego sparametryzowano je parą plików nieskompresowanych, a także RACER, Fiona oraz BFC, które wymagały przekazania jednego pliku odczytów niesparowanych; w celu uzyskania takowych plików utworzono je dołączając drugi plik na końcu pierwszego z pary.

Niepowodzenie wykonania algorytmu

W przypadku, gdy dla danego zestawu danych wejściowych wykonanie algorytmu korekcji nie powiodło się albo przedstawienie jego wyników byłoby bezcelowe, wynik eksperymentu został oznaczony na wykresie przy pomocy jednego z symboli:

- † — wykonanie algorytmu nie zakończyło się w założonym czasie,
- ‡ — brak wystarczającej ilości pamięci operacyjnej,
- § — przerwanie programu na skutek zgłoszenia niezłapanego wyjątku albo przerwania przez system operacyjny w efekcie wystąpienia ogólnego błędu ochrony, z innego powodu niż brak wystarczającej ilości pamięci operacyjnej (np. błąd *index is out of bounds*),
- ∗ — inna przyczyna, wyszczególniona w opisie.

Dodatkowo symbolem × oznaczono przypadki, gdy nie podjęto próby wykonania algorytmu. Wszystkie te symbole odnoszą się wyłącznie do niepowodzenia odpowiedniego wykonania korekcji danego zestawu odczytów, nawet jeżeli zostały zawarte na wykresach obrazujących wyniki innych zadań, np. asemblacji.

6.1.4 Wykorzystane dane

W eksperymentach wykorzystano następujące grupy zestawów danych: zbiory odczytów, genomy referencyjne, wzorcowe zbiory wariantów pełniące funkcję prawdy podstawowej oraz zbiory regionów o wysokiej pewności dla wariantów genomu ludzkiego. Dokładne zastosowanie poszczególnych zestawów zostanie wyjaśnione wraz z opisem poszczególnych grup eksperymentów.

Genomy

Charakterystyka organizmów, których sekwencje nukleotydowe znalazły zastosowanie w eksperymentach, została zawarta w tabeli 6.1. Dane te uzyskano z bazy Genome NCBI [6], przy czym jako długość genomu została przyjęta wartość `expected_ungapped_length` zwrócona za pomocą API bazy: `https://api.ncbi.nlm.nih.gov/genome/v0/expected_genome_size?species_taxid=`, gdzie

jako parametr `species_taxid` podano odpowiedni identyfikator wyszczególniony w tabeli 6.1. Wyjątek stanowi *M. acuminata*, dla którego API nie zwracało tej długości; w tym przypadku przyjęto długość obecną w bazie Genome NCBI jako „median total length". W tabeli podano także niektóre polskie odpowiedniki łacińskiej nazwy jednostki taksonomicznej, których formy nie budziły wątpliwości.

Tabela 6.1: Charakterystyka organizmów

Identyfikator jednostki taksonomicznej w bazie NCBI	Nazwa jednostki taksonomicznej	Długość genomu	Ploidia
317	*Pseudomonas syringae*, B301D-R	6,034 Mpz	Brak informacji, przyjęto haploidalny
4932	*Saccharomyces cerevisiae*	11,63 Mpz	Zmienna, przyjęto diploidalny
3077	*Chlorella vulgaris*, UMT-M1 (chlorella zwyczajna)	37,73 Mpz	Brak informacji, przyjęto haploidalny
6239	*Caenorhabditis elegans*	102,8 Mpz	Diploidalny
3702	*Arabidopsis thaliana* (rzodkiewnik pospolity)	119,2 Mpz	Diploidalny
4641	*Musa acuminata*	461,5 Mpz	Diploidalny
4577	*Zea mays* (kukurydza zwyczajna)	2,19 Gpz	Diploidalny
9606	*Homo sapiens* (człowiek rozumny)	2,823 Gpz	Diploidalny

Przedmiotowe organizmy zostały wybrane z kilku powodów. *S. cerevisiae*, *C. elegans*, *M. acuminata*, *Z. mays* oraz *H. sapiens* reprezentują organizmy o genomach odmiennych rozmiarów, pozwalając na analizę rezultatów algorytmów podczas rozwiązywania zadań w różnej skali. Dodatkowo *Z. mays* charakteryzuje się występowaniem dużej liczby długich powtórzeń jego podsłów [72], co pozwoliło ocenić skuteczność przetwarzania tego typu sekwencji. Właściwości ludzkiego genomu zostały jak dotąd szczególnie dobrze poznane, wobec tego uznano, że istniejącą wiedzę warto uzupełnić o kontekst korekcji odczytów z sekwencjonowania *H. sapiens*. Organizmy *P. syringae* oraz *C. vulgaris* zostały wybrane ze względu na dostępność odczytów ich genomów uzyskanych w technologiach MiSeq oraz NovaSeq. Z kolei organizm *A. thaliana* został dogłębnie przeanalizowany w ramach projektu 1001 Genomów, co zaowocowało dostępnością — wykorzystanych w pracy — zestawów prawdy podstawowej jego wariantów genomu.

Odczyty sekwencjonowania

Pod pojęciem *zestawu odczytów* będzie rozumiany zbiór wszystkich odczytów (sparowanych lub nie) przetwarzanych podczas jednego uruchomienia algorytmu. W tabeli B.5 wyszczególniono zestawy odczytów wraz z roboczymi identyfikatorami[1]. Pliki odczytów zostały zaczerpnięte z bazy EBI, gdyż uzyskanie ich z bazy NCBI, korzystając z własnego oprogramowania bazy, w niektórych przypadkach powodowało uzyskanie plików o niewłaściwej strukturze. Wszystkie rzeczywiste zestawy poddane korekcji zawierają odczyty sparowane, z kolei wszystkie zestawy symulowane zawierają odczyty niesparowane. W tabeli B.7 przedstawiono zestawy odczytów wykorzystane jako wzorzec (źródło *profili błędów*) podczas symulacji odczytów. Z zestawów tych wykorzystano tylko pierwsze odczyty z pary. W drugiej części tabeli podano zestawy, które nie zostały wykorzystane bezpośrednio, lecz były składnikami wyszczególnionych w tabeli B.6 *zestawów pochodnych*. W dalszej części pracy będą wykorzystywane skrócone oznaczenia zestawów, zgodnie z podanymi w wymienionych wyżej tabelach.

Należy zwrócić uwagę na różnicę między wartościami prawdopodobieństwa błędów podanymi w tabeli B.7, a spodziewanymi, zawartymi w tabeli 2.1, zawierającej wartości typowych współczynników błędów różnych technik sekwencjonowania. Różnica jest efektem faktu, że w niniejszej pracy do wyznaczenia „jakości" zestawu odczytów, określanej jako średnie prawdopodobieństwo błędów, wykorzystano średnią arytmetyczną prawdopodobieństw wynikających ze wszystkich współczynników jakości w połowie zestawu odczytów, będących źródłem profili błędów. Ze względu na występowanie tak dużej niezgodności, we wstępnych pracach przeprowadzono weryfikację wartości średniego prawdopodobieństwa, poprzez wykonanie analizy mapowań odczytów do genomu. Uzyskano wyniki zbieżne, tzn. liczba zmian koniecznych do zmapowania odczytów wskazuje, że wyznaczone wysokie średnie uzyskane ze współczynników jakości mają uzasadnienie[2]. Podobnych obserwacji dokonano dla pozostałych zestawów odczytów.

Zestawy S1 oraz C1 zostały wcześniej wykorzystane w eksperymentach przedstawionych w pracach [154, 189]. Zestawy M1 oraz Z1, zgodnie z wiedzą autora, nie zostały jak dotąd wykorzystane w celu analizy korekcji odczytów, jednak ich wybór pozwolił na oparcie eksperymentów na genomach *M. acuminata* i *Z. mays*, użytecznych z przyczyn wymienionych wyżej. Zestaw A1 został wybrany spośród zestawów projektu 1001 Genomów mając na uwadze jego wysoką głębokość sekwencjonowania, umożliwiającą uzyskanie zestawów pochodnych o rozmaitych głębokościach. Zestawy H1_15, H2, H3 oraz H4 zawierają odczyty sekwencjono-

[1] W wybranych przypadkach elementem identyfikatora jest przybliżona głębokość sekwencjonowania.

[2] Obserwacja ta nie jest tożsama ze wspomnianą wcześniej niepełną adekwatnością między wartości współczynników jakości i rzeczywistym prawdopodobieństwem błędów symboli odczytów.

wania jednej biblioteki DNA oraz zostały uzyskane tą samą techniką, dając tym samym możliwość ich łączenia w celu uzyskania różnych głębokości sekwencjonowania. Dodatkowo, uzyskano je dla tej samej próbki DNA, co warianty zawarte w prawdzie podstawowej, wykorzystanej w ocenie, tzn. próbki NA12878, umożliwiając skuteczną ocenę detekcji wariantów. Podobna motywacja dotycząca pochodzenia próbki uzasadniła wybór zestawu H5_55. Zestawy V1 oraz P1 zostały wybrane w celu oceny korekcji odczytów uzyskanych przy pomocy sekwenatorów NovaSeq i MiSeq.

Zestaw Pr3 był wykorzystywany licznie w eksperymentach związanych z korekcją odczytów, w tym w pracach [25, 98, 107, 128, 135, 138, 192]. Pozostałe zestawy dostarczające profili błędów zostały dobrane tak, aby zapewnić odpowiednią długość odczytów (ok. 100 pz oraz 150 pz), a także reprezentować dwa odmienne poziomy jakości.

Część spośród zestawów pochodnych została uzyskana poprzez połączenie (tj. dołączenie plików odczytów na końcu odpowiedniego innego pliku) co najmniej dwóch zestawów rzeczywistych (H1–2_30, H1–3_45, H1–4_60). Z kolei zestawy H5_15, H5_30, H5_45, a także A1_10, A1_20, ..., A1_100 zostały uzyskane przez losową zamianę par odczytów w zestawie A1 oraz wybór z niego pewnej liczby par odczytów tak, aby uzyskać różne wartości głębokości sekwencjonowania. Rozwiązanie to pozwoliło na bardziej elastyczny dobór głębokości sekwencjonowania odczytów rzeczywistych. W podobny sposób uzyskano zestawy P1_60 oraz V1_60 o ograniczonej głębokości sekwencjonowania.

Dane detekcji wariantów

Analiza wpływu korekcji na detekcję wariantów wymagała dostępności zestawu wariantów uznawanego za pełny i prawidłowy, pełniącego rolę *prawdy podstawowej* (ang. *ground truth* — *GT*) [131], zawierającego najlepszą znaną wiedzę na temat wariantów obecnych w genomie analizowanego osobnika. Zestawy takie są opracowane przy wykorzystaniu różnych technik sekwencjonowania, zwiększając zaufanie do takiego zestawu. Ich zestawienie zostało zamieszczone w tabeli B.12. Dalej będą one nazywane *zestawami referencyjnymi*.

Genomy referencyjne

W tabeli B.4 wyszczególniono numery dostępowe sekwencji wchodzących w skład poszczególnych genomów referencyjnych, wykorzystanych na potrzeby mapowania, w tym mapowania wykonanego jako część procesu detekcji wariantów. Dane zostały uzyskane z bazy Genome NCBI, gdzie jako genom przyjęto zbiór sekwencji wszystkich autosomów i ewentualnych allosomów obu płci (w przypadku genomu ludzkiego), DNA mitochondrialnego oraz ewentualnego plastydu. Wyjątek stanowi genom *A. thaliana*, do którego nie wliczono sekwencji DNA mitochon-

drialnego ani plastydu ze względu na brak uwzględnienia tych sekwencji w pliku referencyjnych zestawów wariantów.

6.1.5 Parametryzacja algorytmów

Sposób doboru parametrów poszczególnych algorytmów korekcji został przedstawiony w tabeli 6.2. Parametry posiadające wartości domyślne nie były modyfikowane. Konkretne wartości przyjętych parametrów zostały przedstawione w podrozdziale B.4.

Najczęściej wymaganym parametrem algorytmów jest długość oligomeru k. Została ona dobrana poprzez wykonanie algorytmu dla wielu kolejnych wartości k (w przypadku eksperymentów związanych z detekcją wariantów oraz wykorzystujących odczyty ludzkiego genomu, wybierano tylko nieparzyste wartości k) do momentu, aż dla pewnego k_{best} kryterium oceny algorytmu osiągnęło wartość najlepszą, a jednocześnie dla dwóch wartości bezpośrednio mniejszych i większych od k_{best} uzyskano słabsze wyniki. Dodatkowo, jeżeli zmiana oceny mocno wahała się wraz ze zmianą k, weryfikowany zakres był poszerzany. Kryteria oceny algorytmów w poszczególnych grupach eksperymentów zostaną przedstawione w dalszej części pracy.

Fakt, że podczas takiego poszukiwania został sprawdzony dosyć szeroki zakres wartości, właściwie wyeliminował ryzyko utknięcia w optimum lokalnym. W praktycznym wykorzystaniu algorytmu takie rozwiązanie jest wprawdzie trudne do zaakceptowania, jednak uzyskane w ten sposób rezultaty niosą informację o potencjale algorytmu, a także pozwalają na zobrazowanie wpływu parametru k na wyniki oraz ocenę przydatności wytycznych autorów algorytmu dotyczących doboru tego parametru. Dodatkowo przyjęto, że wartość $k > 10$, co jest granicą bardzo niską, ale uzasadnioną przez część wyników. Wyjątkiem jest korekcja odczytów z zestawów H5_15, H5_30, H5_45, H5_55, która została przeprowadzona dla wartości k najlepszych dla odpowiadających pozostałych zestawów H1_15, H1–2_30, H1–3_45, H1–4_60.

Jako szacunkową długość genomu oraz ploidię organizmu przyjęto wartości zgodnie z tabelą 6.1. Liczba k-merów algorytmu Musket była określana przy pomocy KMC. Jako parametr p_α algorytmu Lighter przyjęto wartość $p_\alpha = \frac{7}{\text{dep}}$, zgodnie z uzasadnieniem podanym w podrozdziale 4.2.3. Zależy ona od głębokości sekwencjonowania sekwencjonowania dep, która z kolei była określana zgodnie z równaniem 3.2, parametryzowanym wartościami podanymi w tabeli B.5. Próg obcięcia algorytmu Blue został określony zgodnie z metodą algorytmu RECKONER. Algorytm Karect, według zaleceń autorów algorytmu, został skonfigurowany w sposób wykorzystujący odległość edycyjną Hamminga.

Tabela 6.2: Dobór parametrów algorytmów

Algorytm	Parametr algorytmu	Sposób doboru
RECKONER	k	Eksperymentalny
Musket	k	Eksperymentalny
	$\widehat{\widehat{k}}_{\mathrm{N}}$	KMC — „No. of unique k-mers”
RACER	$\widehat{\ell}_{\mathcal{G}}$	Zgodnie z tabelą 6.1
BLESS	k	Eksperymentalny
Fiona	$\widehat{\ell}_{\mathcal{G}}$	Zgodnie z tabelą 6.1
Blue	k	Eksperymentalny
	$\widehat{\ell}_{\mathcal{G}}$	Zgodnie z tabelą 6.1
	η_{cut}	Mechanizm RECKONER
Lighter	k	Eksperymentalny
	p_α	$\frac{7}{\mathrm{dep}}$
	$\widehat{\ell}_{\mathcal{G}}$	Zgodnie z tabelą 6.1
BFC	$\widehat{\ell}_{\mathcal{G}}$	Zgodnie z tabelą 6.1
Karect	Ploidia	Zgodnie z tabelą 6.1
	Typ odległości edycyjnej	Hamminga
SAMDUDE	—	—

6.1.6 Uwagi nazewnicze

W celu zachowania precyzji należy zwrócić uwagę na subtelne różnice znaczeniowe wykorzystywanych pojęć. Wariant typu krótki indel (insercja, delecja; w dalszej części pracy pomijane będzie słowo „krótki”) jest różnicą między genomem, którego sekwencjonowanie pozwoliło na uzyskanie zestawu odczytów, a zadanym genomem referencyjnym. Błąd typu indel (insercja, delecja) jest różnicą między odczytem a sekwencją genomu, z którego odczyt został uzyskany. Oba przypadki wiążą się z występowaniem różnic między sekwencją odczytu a genomem referencyjnym, choć są efektem różnych zjawisk (odmienność jest analogiczna jak w pojęciach „wariant typu SNP” oraz „błąd substytucji”). W dalszej części pracy będą stosowane znaczenia tych pojęć odpowiednie do kontekstu. Dodatkowo, w przypadkach, gdy istotne będzie rozważenie różnicy sekwencji odczytu i genomu referencyjnego, pojęcia te będą nazywane łącznie *różnicą typu indel (insercja, delecja)*.

6.2 Analiza według kryteriów jakościowych

6.2.1 Odczyty symulowane

Analizę z użyciem odczytów symulowanych przeprowadzono z dwóch wspomnianych wcześniej powodów: łatwej, bezpośredniej mierzalności uzyskanych wyników oraz elastyczności w doborze własności odczytów. Jednakże niedoskonałości (uproszczenia) procesu symulacji skutkują tym, że uzyskane w oparciu o te eksperymenty wyniki jakościowe należy traktować pomocniczo, akceptując, że nie muszą mieć one pełnego przełożenia na jakość wyników korekcji odczytów rzeczywistych.

W celu przeprowadzenia symulacji oparto się na rzeczywistych zestawach odczytów, z których uzyskano *profile błędów*, rozumiane jako charakterystyka jakości poszczególnych symboli odczytów. Symulację odczytów przeprowadzono na dwa sposoby: w oparciu o uproszczoną metodę Quake oraz przy pomocy narzędzia ART. Wybór pierwszej możliwości jest uzasadniony popularnością tej metody w literaturze, natomiast drugiej ze względu na potencjalnie użyteczną w ocenie korekcji funkcjonalność symulatora ART, obejmującą możliwość określenia oraz wykorzystania profili błędów w oparciu o rzeczywiste zestawy odczytów. Daje to potencjał nie tylko do utworzenia odczytów zadanej jakości, ale również, bardziej niż przy oparciu się na predefiniowanych przez autorów ART profilach, elastycznego doboru długości odczytu (ograniczonego właściwie tylko dostępnością odczytów wzorcowych odpowiedniej jakości). Przeprowadzenie symulacji dwoma metodami umożliwia też uzyskanie informacji na temat zgodności ich wyników.

W przypadku symulacji uproszczoną metodą Quake rolę profili pełniły ciągi współczynników jakości kolejnych odczytów (które wykorzystano zgodnie z opisem w podrozdziale 4.3), natomiast w narzędziu ART przeprowadzono wstępną analizę wzorcowych odczytów, z których narzędzie pozwoliło na uzyskanie profili, wykorzystywanych później do symulacji.

Zrezygnowano z posłużenia się konkurencyjnym względem ART (nieopisanym w niniejszej pracy) narzędziem Mason 2 [103], które posiada podobne możliwości, jednak we wstępnych eksperymentach generowane przy jego pomocy współczynniki jakości okazały się być nieadekwatne do rzeczywistych współczynników błędów. Pomimo podjęcia próby kontaktu z twórcami narzędzia, nie uzyskano informacji o wyeliminowaniu tego błędu.

Jako granulację w ocenie korekcji wybrano cały odczyt, uzasadniając to podobnie jak w suplemencie pracy [128] — niewielka część odczytów Illumina posiada dużą liczbę błędów (podczas gdy zdecydowana większość odczytów jest obarczona niewielką ich liczbą), wobec tego ta względnie mała grupa trudnych do korekcji odczytów przy zastosowaniu granulacji pojedynczego symbolu może wywrzeć znaczny wpływ na uzyskane oceny. Zwrócono także uwagę na wyższą praktyczną użyteczność „całkowicie” prawidłowych odczytów oraz większą rygo-

rystyczność takiej granulacji. Ocenę oparto na miarach zdefiniowanych w podrozdziale 4.4.1. Wykorzystana dalej notacja LxDy oznacza odczyty długości x przy głębokości sekwencjonowania równej y. Z kolei gwiazdka (*) obecna w oznaczeniach pełni rolę symbolu wieloznacznego, włączającego wszystkie możliwe kontynuacje oznaczenia (np. Q2_S1_* oznacza Q2_S1_L100D20, Q2_S1_L100D30 itd., zgodnie z wykazami zestawów symulowanych, zawartymi odpowiednio w tabelach B.8 oraz B.9. Wartości miary zysku zostały przemnożone przez 100.

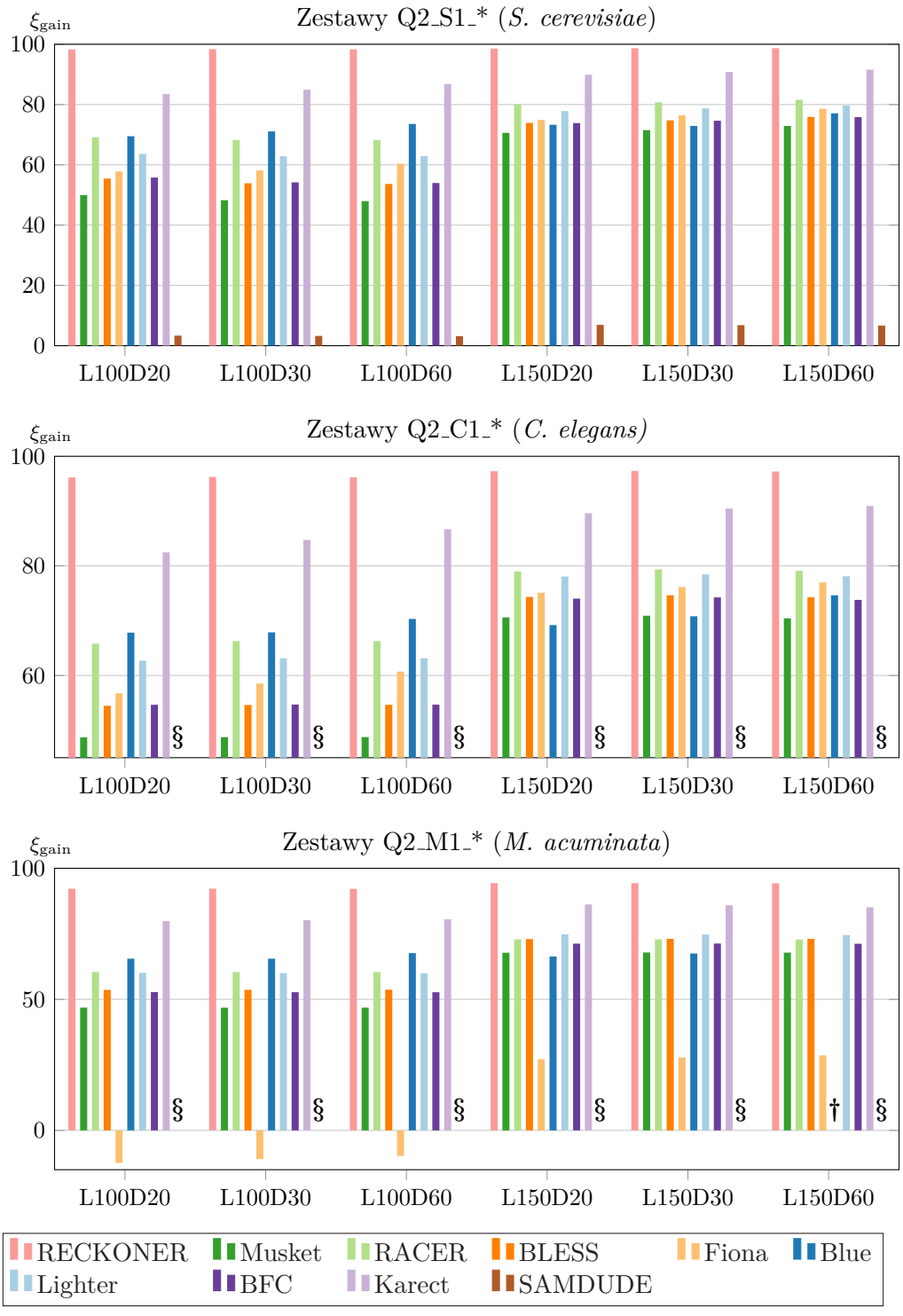
Algorytmy wymagające parametryzacji długością oligomeru k zostały uruchomione wielokrotnie z kolejnymi wartościami tego parametru dla wszystkich organizmów i obu średnich prawdopodobieństw, przy najkrótszych odczytach oraz głębokości sekwencjonowania równej 20, tzn. dla zestawów *_L100D20. Kryterium oceny i wyboru wartości k_{best} stanowiła wartość miary ξ_{gain}; tę samą wartość k_{best} zastosowano dla wszystkich pozostałych zestawów tego organizmu o obu wartościach średniego prawdopodobieństwa.

Symulacja uproszczoną metodą Quake

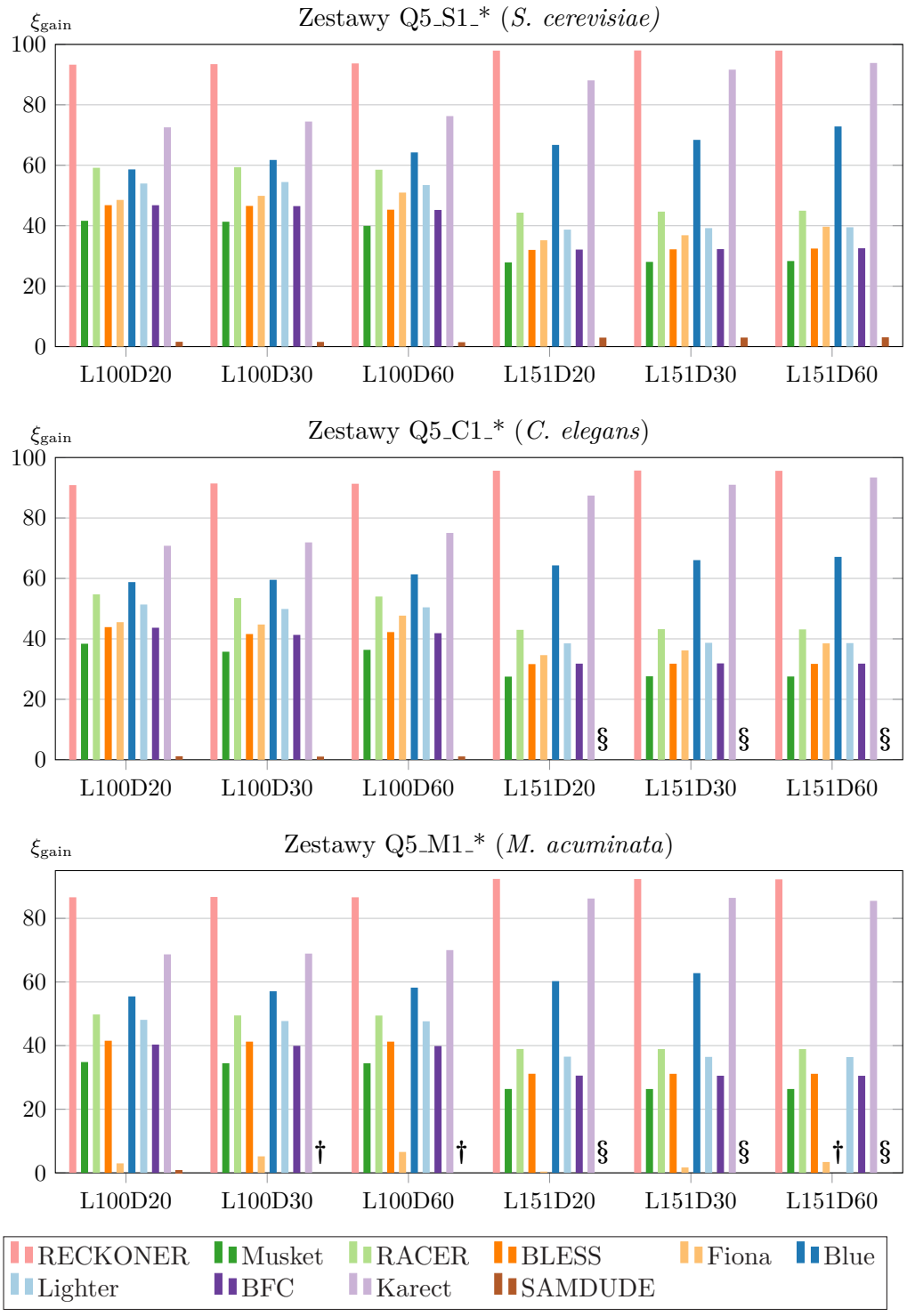
W tabeli B.8 przedstawiono szczegółowe informacje na temat wszystkich zestawów odczytów wygenerowanych uproszczoną metodą Quake. Z kolei w tabelach B.14–B.16 przedstawiono wartości parametrów algorytmów, które pozwoliły na uzyskanie najlepszych (i przedstawionych) rezultatów. Na rys. 6.1 oraz 6.2 przedstawiono wyniki korekcji wyrażone przemnożoną przez 100 miarą zysku ξ_{gain} dla poszczególnych organizmów oraz odczytów różnej jakości i długości. Wyniki wyrażone pozostałymi miarami zostały przedstawione w podrozdziale C.1.1.

Odczyty wygenerowane przy pomocy omawianej metody cechują się brakiem błędów typu indel. Ich brak z jednej strony skutkuje słabszym realizmem, ale jednocześnie pośrednio weryfikuje mechanizmy wykrywania w algorytmach błędów tego typu. Zbyt częste przypadki prób korekcji błędów typu indel spowodowałyby uzyskanie słabych wyników. Należy jednak zaznaczyć, że taka metoda oceny może powodować uprzywilejowanie algorytmów niewyposażonych w jawną korekcję błędów typu indel.

Wszystkie wyniki wskazują, że różne algorytmy charakteryzują się bardzo odmienną skutecznością. Szczególnie dobre rezultaty osiągnięto w efekcie korekcji odczytów algorytmem RECKONER. W prawie wszystkich eksperymentach pozwolił on na wyeliminowanie przeszło 95%, a w wielu niemal 100% przypadków błędnych odczytów. Łatwo dostrzegalna jest tendencja, że lepszy poziom korekcji tego algorytmu wystąpił dla odczytów długości 150 pz, przy czym głębokość sekwencjonowania nie ma wyraźnego znaczenia. Jej wpływ występuje jednak w większości pozostałych algorytmów, dla których wraz ze zwiększaniem głębokości następuje poprawa jakości wyników. Można z tego wnioskować, że algorytm RECKONER nie ulega negatywnemu wpływowi niewielkiej głębokości sekwencjonowania.



Rysunek 6.1: Wpływ korekcji na wartość ξ_{gain} dla odczytów symulowanych uproszczoną metodą Quake, $p_{\text{mean}} = 2\%$



Rysunek 6.2: Wpływ korekcji na wartość ξ_{gain} dla odczytów symulowanych uproszczoną metodą Quake, $p_{\text{mean}} = 4$–5%

Wpływ długości odczytu jest widoczny w większości algorytmów, choć zwykle wiąże się z poprawą jakości dla odczytów długości 150 pz (w porównaniu do odczytów długości 100 pz) dla niskiego średniego współczynnika błędu, a ze spadkiem dla wysokiego. Zjawisko to można dostrzec dla wszystkich algorytmów z wyjątkiem RECKONER, Karect oraz SAMDUDE. Przypuszczalnie algorytmy nie są dostosowane do radzenia sobie ze zjawiskiem występowania dużej liczby pojedynczych błędów w odczycie (często mającego miejsce w zestawach Q5_*_L151*).

W kwestii ogólnej jakości wyraźnie wyróżniają się także algorytmy Karect, Fiona oraz SAMDUDE. Karect — wyłączając RECKONER — powoduje uzyskanie najlepszych wyników, osiągając współczynnik zysku rzędu 70–80, w zależności od średniego współczynnika błędu. Z drugiej strony, jedynym algorytmem, który spowodował wygenerowanie większej liczby błędnych odczytów niż skorygowanych prawidłowo, był algorytm Fiona. W rezultacie uzyskano ujemne wartości miary ξ_{gain} tego algorytmu, choć taki efekt ma miejsce tylko dla zestawów organizmu *M. acuminata*. Algorytmy RACER, BLESS, Blue, Lighter i BFC znalazły się w środku stawki. Z kolei algorytm SAMDUDE, choć przyniósł niewielką poprawę, to znacznie mniejszą niż w przypadku pozostałych algorytmów, a jednocześnie w większości eksperymentów wykonanie algorytmu nie powiodło się. Spośród pozostałych algorytmów problemy z wykonaniem zostały zaobserwowane dla przypadków Q*_M1_L151D60 podczas korekcji algorytmem Blue.

Zaobserwowane zależności z pewnością są efektem m.in. niedoskonałości metody symulacji odczytów, w tym braku generacji błędów typu indel (choć np. w przypadku algorytmu RECKONER, wyposażonego w mechanizm ich korekcji, brak ten nie mógł wywrzeć znaczącego wpływu, na co wskazują bardzo dobre wyniki), ale też zaobserwowanej w odczytach będących źródłem profili błędów dużej liczby występujących kolejno bardzo niskich współczynników jakości, wskazujących na prawdopodobieństwo błędu wynoszące więcej niż 50% na wielu sąsiednich pozycjach. W rezultacie w wygenerowanych odczytach pojawiały się podsłowa o bardzo dużej liczbie błędów. Skuteczność radzenia sobie z podobnymi zjawiskami algorytmów może być różna, nie implikując jednak ich adekwatności dla korekcji odczytów rzeczywistych.

Symulacja narzędziem ART

W tabeli B.9 przedstawiono szczegółowe informacje na temat wszystkich zestawów odczytów wygenerowanych narzędziem ART. Należy zauważyć, że liczebności wygenerowanych odczytów w zestawach A2_M1_* oraz A5_M1_* (*M. acumianta*) są nieco mniejsze, niż uzyskane uproszczoną metodą Quake (tabela B.8). Jest to prawdopodobnie efektem błędu implementacji narzędzia ART, ponieważ było ono uruchamiane z podaniem dokładnej liczby odczytów, identycznej jak w alternatywnej metodzie. Mimo to w rezultacie uzyskano inną liczbę odczytów. Z kolei w tabelach B.17–B.19 przedstawiono wartości parametrów algorytmów, które

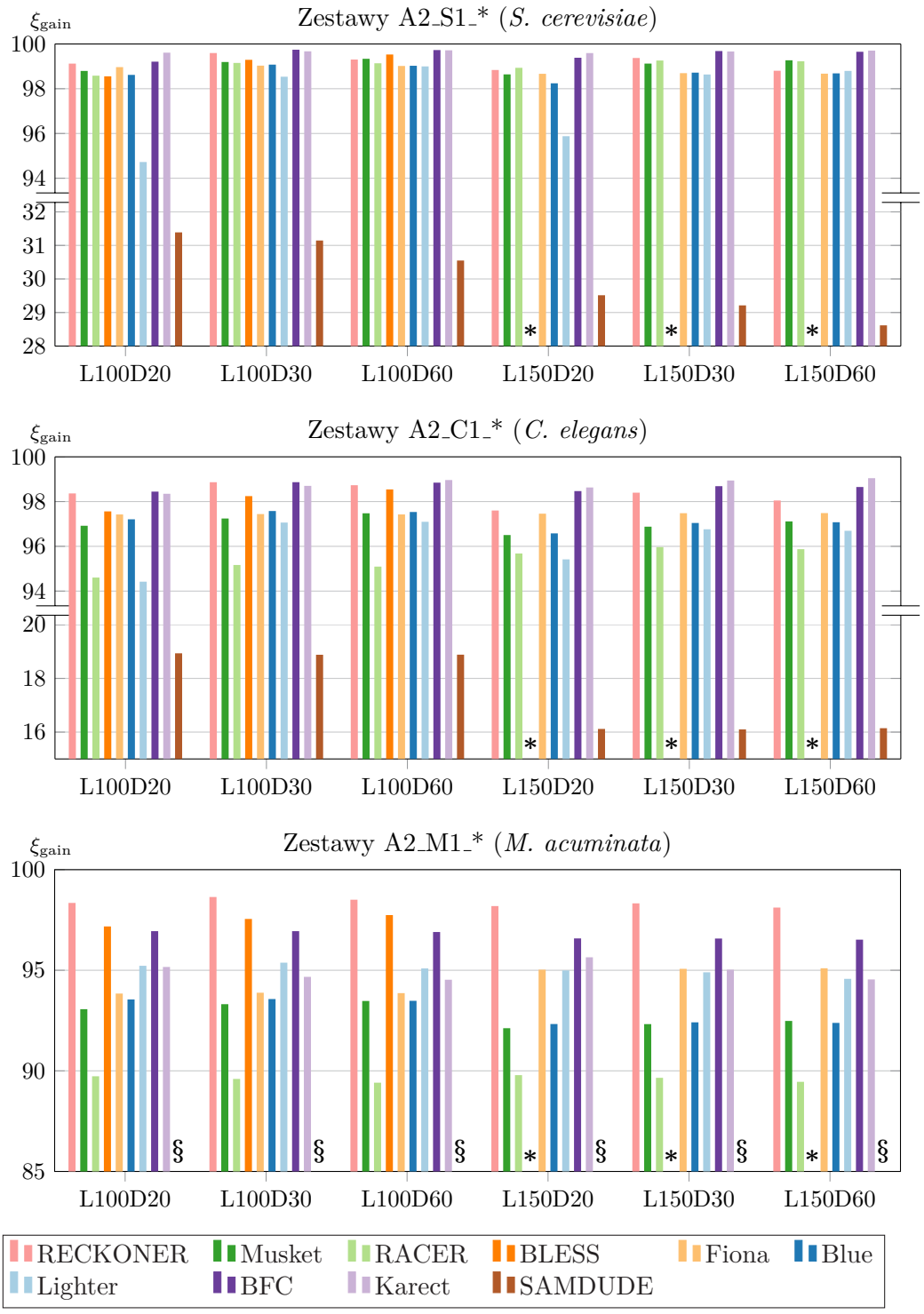
pozwoliły na uzyskanie najlepszych (i przedstawionych) rezultatów. Na rys. 6.3 oraz 6.4 przedstawiono różne wyniki korekcji według miary zysku (ξ_{gain}) dla poszczególnych organizmów oraz odczytów różnej jakości i długości. Wyniki dla pozostałych miar zostały pokazane w podrozdziale C.1.1.

Wykonanie algorytmu BLESS w niektórych przypadkach powodowało zgłoszenie komunikatu o błędnym zakresie współczynników jakości. Prawdopodobnie jest to skutkiem niewłaściwego działania mechanizmu detekcji sposobu kodowania współczynników w postaci znaków ASCII, gdy współczynniki jakości mają charakterystykę podobną jak w odczytach profilu Pr2, tzn. w zestawie występują współczynniki jakości wartości 3, a nie występują współczynniki wartości 4 i 5 (mowa o wartościach po dekodowaniu kodów ASCII w skali Phred+33).

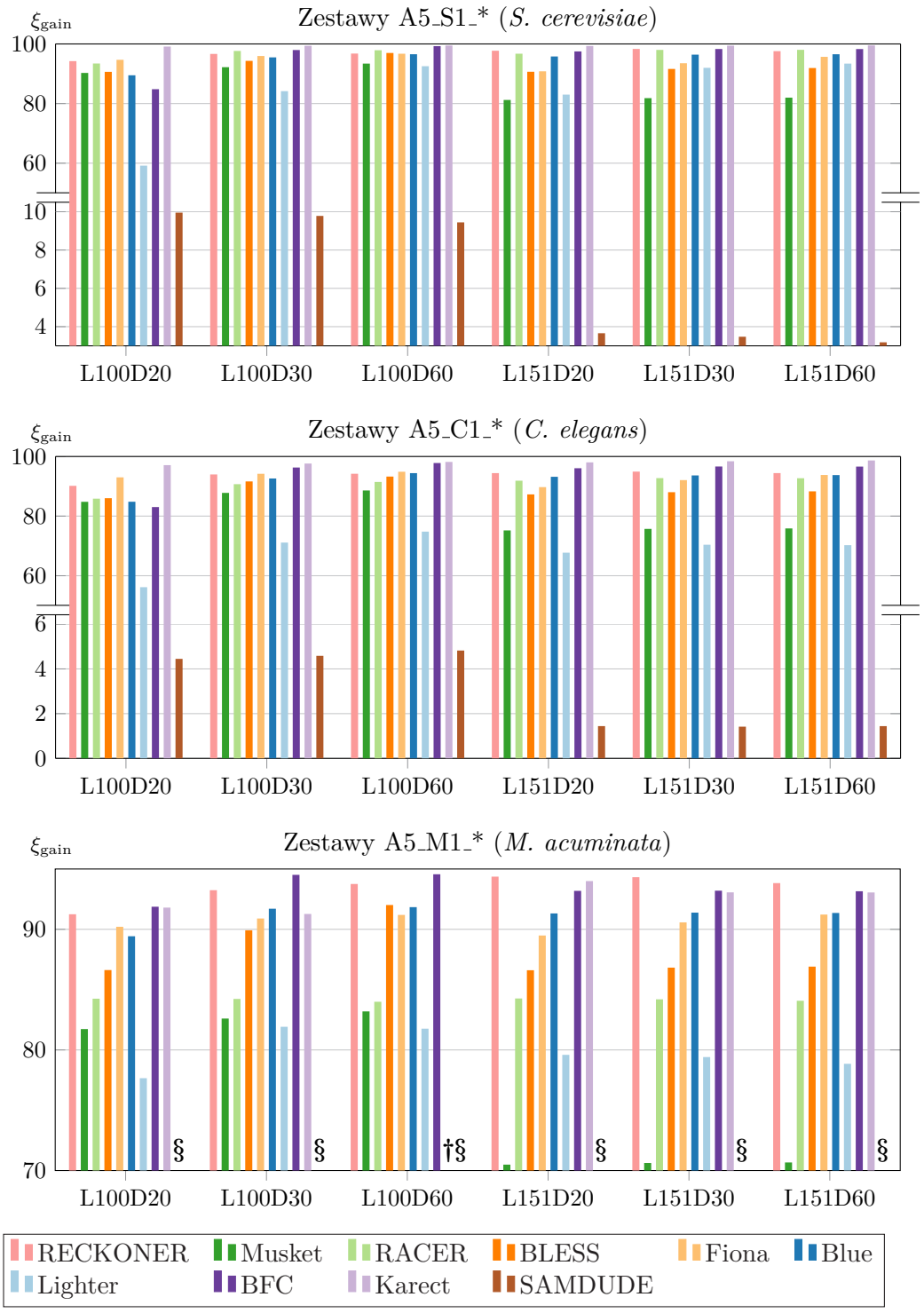
W większości przypadków uzyskanie najlepszych wyników umożliwiały algorytmy BFC i Karect, będące nieco skuteczniejsze od algorytmu RECKONER, choć w przypadku zestawu A5_S1_L100D60 algorytm Karect nie pozwolił na uzyskanie wyników w zadanym czasie. Wyjątek stanowią największe zestawy A2_M1_* oraz A5_M1_*, gdzie w większości przypadków najlepsze wyniki uzyskano dzięki zastosowaniu algorytmu RECKONER. Jego wyniki wykazały się też stabilnością wraz ze zmianą długości odczytów oraz głębokości sekwencjonowania, szczególnie w kontraście do algorytmu Lighter, choć należy zauważyć, że w większości przypadków następował dla niego niewielki spadek jakości dla głębokości równej 60 w porównaniu do głębokości 30. Zjawisko to jest spowodowane przez sposób doboru długości oligomeru, ponieważ każdy zestaw odczytów danego organizmu i jakości był korygowany przy pomocy algorytmu z tą samą zadaną długością, wyznaczoną dla głębokości sekwencjonowania równej 20×. Podobne zjawisko, ale obejmujące pełen zakres wartości głębokości sekwencjonowania, występuje w niektórych zestawach odczytów dla algorytmu Karect, np. A2_M1_* oraz A5_M1_*. W tych przypadkach przyczyna nie jest znana, a ze względu na brak przynależności tego algorytmu do algorytmów opartych na spektrum k-merów, musi być ona inna.

W większości przypadków algorytmy Blue oraz Fiona nie wyróżniały się, w szczególności nie zaobserwowanego omówionego w poprzednim podrozdziale zjawiska znacznego pogorszenia jakości odczytów w wyniku korekcji algorytmem Fiona. W bieżącym przypadku zanotowano nawet jeden przypadek A5_C1_L100D20, gdy algorytm Fiona okazał się być drugim pod względem jakości algorytmem.

Szczególnie słabe wyniki uzyskano we wszystkich przypadkach dla algorytmu SAMDUDE, dla zestawów odczytów genomu *M. acuminata* dla algorytmów Musket i RACER, a także w przypadku algorytmu Lighter dla odczytów o wyższym średnim prawdopodobieństwie błędu symolu. Ogólnie wpływ długości odczytu okazał się być nie tak znaczący jak w odczytach symulowanych uproszczoną metodą Quake, ponadto wielkość ta nie wykazuje tak wyraźnej zależności od jakości odczytów. W większości przypadków uzyskane wyniki są słabsze, ale mimo



Rysunek 6.3: Wpływ korekcji na wartość ξ_{gain} dla odczytów symulowanych narzędziem ART, $p_{\text{mean}} = 2\%$



Rysunek 6.4: Wpływ korekcji na wartość ξ_{gain} dla odczytów symulowanych narzędziem ART, $p_{\text{mean}} = 4$–5%

to powszechną sytuacją są przypadki, gdy korekcja pozwala na redukcję liczby błędnych odczytów o więcej niż 98%. Należy jednak zaznaczyć, że we wstępnych eksperymentach poddawano analizie także inne, nieomówione algorytmy, których wyniki były słabsze od zaprezentowanych.

Podsumowanie wyników dla odczytów symulowanych

Wyniki uzyskane poprzez wykorzystanie odczytów symulowanych dwiema metodami różnią się w umiarkowanym stopniu. Wykonanie większości algorytmów powiodło się, z wyjątkiem algorytmów SAMDUDE, BLESS, w dwóch przypadkach Blue oraz jednym przypadku Karect. W prawie wszystkich przypadkach najlepsze wyniki uzyskano dla algorytmu RECKONER albo Karect, a dla odczytów symulowanych narzędziem ART często także BFC. Przypuszczalnie ten ostatni algorytm jest wrażliwy na znaczące uproszczenia modelu odczytów symulowanych metodą Quake. Najlepsze algorytmy pozwalają na skuteczną korekcję więcej niż 90% odczytów zawierających błędy, w wielu przypadkach osiągając wartość miary zysku niemal 100.

Algorytmy BLESS oraz Blue w większości przypadków pozwoliły na uzyskanie przeciętnych wyników. Z kolei słabe wyniki uzyskano w rezultacie wykonania algorytmu SAMDUDE, a w przypadku odczytów symulowanych metodą Quake także Fiona. Pozwoliły one na niewielką poprawę jakości, a w przypadku Fiona korekcja skutkowała nawet pogorszeniem jakości odczytów. W wybranych sytuacjach słabymi wynikami korekcji odczytów symulowanych narzędziem ART wyróżniły się również algorytmy Musket, Lighter oraz RACER.

Zapotrzebowanie korekcji odczytów symulowanych na zasoby, ze względu na czysto eksperymentalny charakter tego zadania, zostanie przedstawione wyłącznie w dodatku, w podrozdziałach C.2.1 oraz C.2.1.

6.2.2 Odczyty rzeczywiste

Ocenę jakości korekcji rzeczywistych odczytów przeprowadzono poprzez wykonanie zadań asemblacji, mapowania i detekcji wariantów oraz analizę ich wyników. W dwóch pierwszych podejściach wykorzystano następujące zestawy odczytów spośród wyszczególnionych w tabeli B.5: P1_60, S1, V1_60, C1, M1, Z1, H1–3_45. Odczyty, po przeprowadzeniu ich korekcji niezależnie różnymi algorytmami, zostały poddane asemblacji oraz mapowaniu. Podobnie postąpiono z nieskorygowanymi odczytami, uzyskując kontrolną grupę wyników. Detekcję wariantów wykonano dla zestawów H1_15, H1–2_30, H1–3_45, H1–41_60; dla zestawów H5_15, H5_30, H5_45, H5_55 oraz zestawów A1_10, A1_20, ..., A1_100, które odpowiadają organizmom o dwóch znacznie różniących się rozmiarach.

Algorytmy wymagające parametryzacji długością oligomeru k zostały uruchomione zgodnie z opisem w podrozdziale 6.1.5, przy czym dla zestawów wy-

korzystanych w detekcji wariantów oraz asemblacji odczytów z zestawu H1–3_45 uruchomiono algorytm tylko z nieparzystymi wartościami k. Wartość k_{best} wyznaczono optymalizując długość k, według poniższych reguł oceny:

- w asemblacji — w oparciu o wartości miar oceny asemblacji N50, NA50, NG50, NGA50 (nazywane dalej *miarami z grupy N*), L50, LA50, LG50 oraz LGA50 (nazywane dalej *miarami z grupy L*) wyznaczono ranking wartości k; dla każdej miary poszczególnym przypadkom testowym przyznano ocenę rankingową: 1 dla przypadku, gdzie dana miara była najlepsza oraz kolejne wartości dla następnych w rankingu wartości miar; na koniec zsumowano oceny wszystkich miar, a jako najlepszy przypadek testowy (i najlepszą wartość k) przyjęto ten, który uzyskał najniższą sumę ocen rankingowych,
- w detekcji wariantów — przyjęto wartość pozwalającą na uzyskanie największej wartości średniej geometrycznej miar ξ_{F1} dla wariantów typu SNP oraz indel.

Mapowanie przeprowadzono dla wartości k najlepszych według kryterium oceny asemblacji. W kolejnych podrozdziałach przedstawiono wyniki dla wymienionych kryteriów oraz różnych miar.

Asemblacja

Celem asemblacji jest uzyskanie kontigów charakteryzujących się jak najlepszymi własnościami dotyczącymi ich długości oraz liczebności. Istotnym zagadnieniem jest także minimalizacja liczby błędów asemblacji. Chociaż asemblery wykazują pewną odporność na występowanie błędów sekwencjonowania [107], to jednak ich obecność wywiera znaczący wpływ na uzyskany rezultat [89]. W ramach eksperymentów asemblacja została wykonana za pomocą algorytmu Minia [46], ze względu na jego wysoką wydajność, szczególnie istotną w przypadku asemblacji dużych genomów. Do wyznaczania miar jakości asemblacji wykorzystano narzędzie Quast [92].

Wykonanie oceny jakości asemblacji jest trudnym zadaniem, między innymi na skutek dostępności licznych miar, które mogą zwracać niespójny rezultat [39]. Należy wziąć pod uwagę, że wiele miar nie daje kompletnej informacji o jakości rezultatu, np. wartość N50 uwzględnia tylko długości kontigów, ale nie ich poprawność. Z tego powodu podjęto decyzję o dokonaniu oceny poprzez wykorzystanie wielu miar, do których należą wartość pokrycia kontigami ξ_{cov}, statystyki N50, NA50, NG50, NGA50 (wartości, które powinny zostać poddane maksymalizacji po korekcji — wg suplementu pracy [92]), a także liczba błędnych asemblacji ξ_{misasm} oraz statystyki L50, LA50, LG50 oraz LGA50 (poddane minimalizacji, jak LGA50 w pracy [25]). Znaczenie tych miar zostało przedstawione w podrozdziale 4.4.1.

Na rys. 6.5–6.11 zaprezentowano uzyskane wyniki. Z kolei na rys. 6.12–6.14 przedstawiono zmianę czasu asemblacji oraz zapotrzebowania na pamięć jako rezultat korekcji wejściowych odczytów. W tabeli B.20 zawarto wartości parametrów algorytmów, które pozwoliły na uzyskanie najlepszych (i przedstawionych) rezultatów. Należy zaznaczyć, że wyniki poniższej analizy, wykonanej w oparciu o niepowiązane zestawy odczytów, nie powinny być traktowane kategorycznie i jednoznacznie, ponieważ zestawy te uzyskano nie tylko dla organizmów o genomach różnych rozmiarów, ale też ich różnej charakterystyce oraz odmiennej głębokości sekwencjonowania i różnych charakterystykach błędów odczytów.

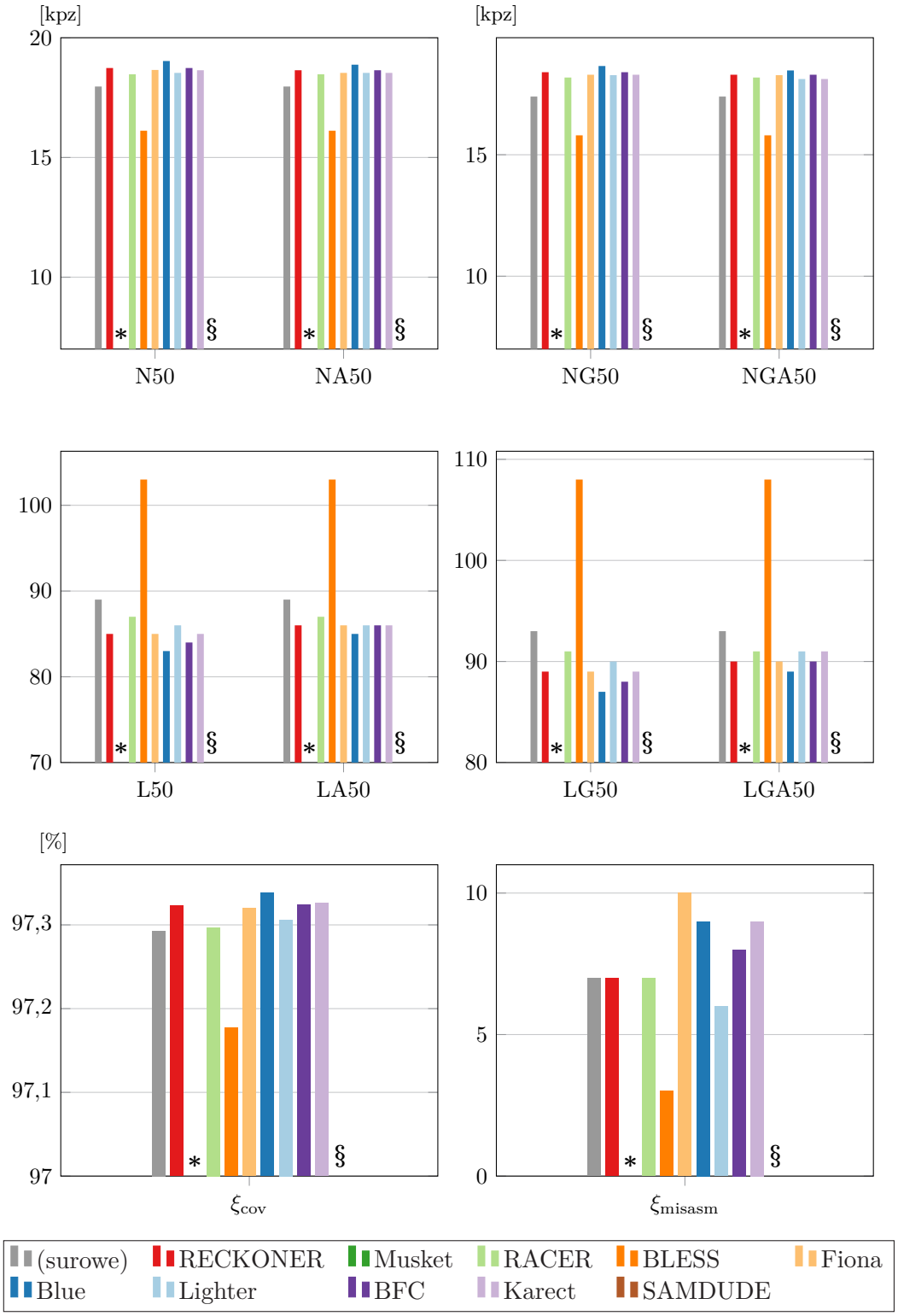
Dla zestawów M1 oraz Z1 nie wyznaczono wyników odnoszących się do połowy długości genomu, tzn. NG50, NGA50, LG50, LGA50, ponieważ uzyskane kontigi posiadały łączną długość mniejszą od zadanej długości genomu, na co wskazują wartości miary $\xi_{\text{cov}} < 50\%$.

Jakość rezultatów asemblacji w wyniku korekcji ulega zasadniczej zmianie. W wielu przypadkach uzyskane wyniki okazały się słabsze niż odczytów nieskorygowanych (oznaczonych jako „surowe"). Stąd kluczowe jest określenie, zastosowanie których algorytmów korekcji prawdopodobnie przyniesie efekt odwrotny do zamierzonego, a w przypadku których należy zachować ostrożność, ponieważ uzyskane wyniki mogą zostać poprawione, a jednocześnie istnieje ryzyko ich pogorszenia.

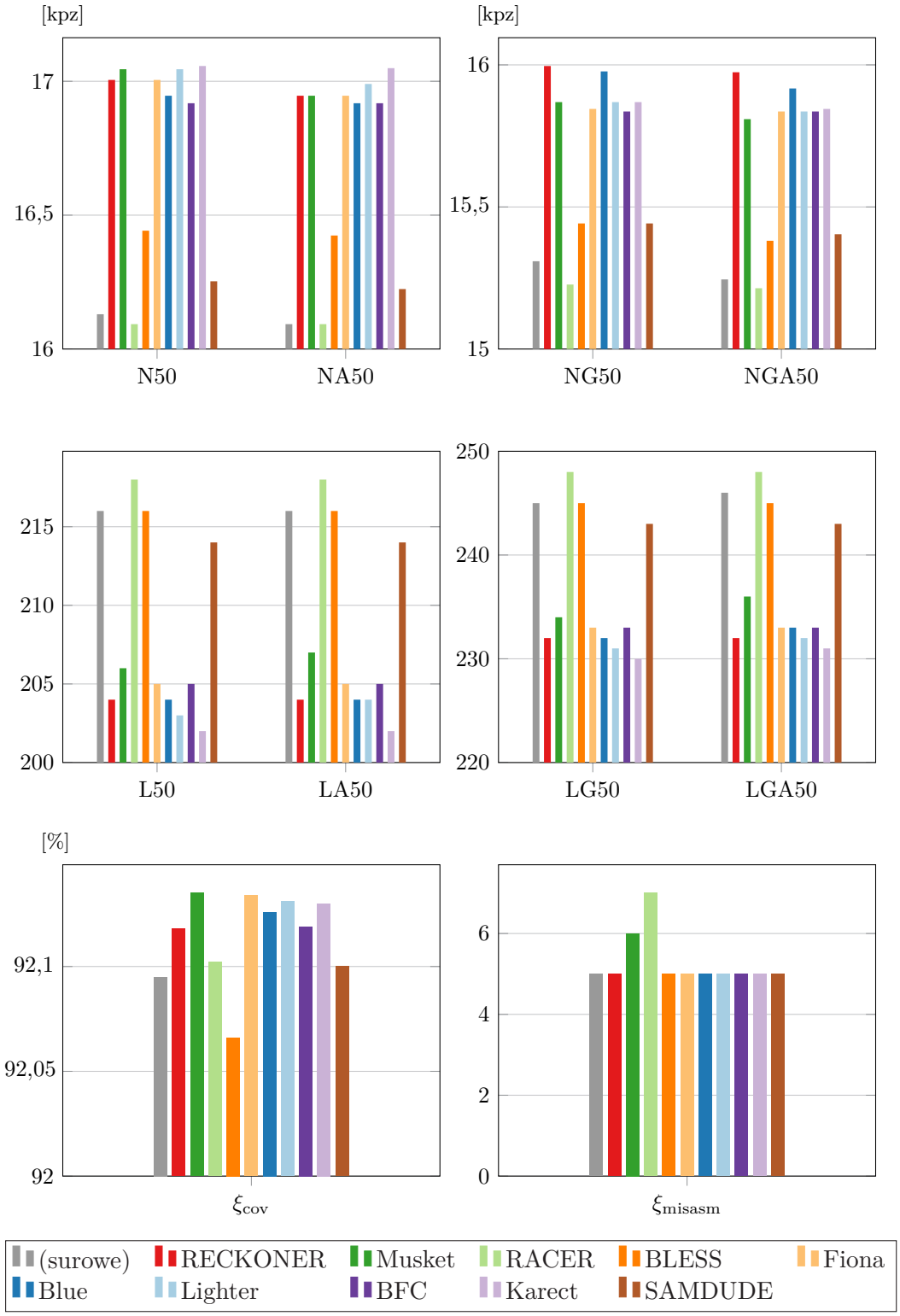
Korekcja odczytów z zestawu P1_60 pozwoliła na poprawę jakości w wyniku pracy prawie wszystkich algorytmów korekcji, przy czym nieznacznie dominujący okazał się Blue, a w dalszej kolejności: RECKONER oraz BFC. Zaobserwowano, że dla żadnej wartości parametru k nie wystąpiła zmiana sekwencji odczytów po korekcji odczytów algorytmem Musket, dlatego nie pokazano jego wyników. Z kolei dla algorytmu BLESS wystąpiło wyraźne, przeszło dziesięcioprocentowe pogorszenie większości rezultatów. Wpływ korekcji na wskaźnik ξ_{misasm} okazał się być niewielki, jednak jego bardzo małe sumaryczne wartości nie pozwalają na wyciągnięcie wniosków. Wyniki pozostałych algorytmów są do siebie dosyć podobne. Zestaw P1_60 zawierał odczyty uzyskane z sekwenatora MiSeq, zatem wyróżniające się własnościami od pozostałych zestawów.

Korekcja odczytów z zestawu S1 powiodła się w przypadku wszystkich algorytmów korekcji. Uzyskane wyniki są korzystne — prawie każdy algorytm pozwolił na poprawę wyników według większości miar. Wyjątek stanowi algorytm RACER, dla którego uzyskano wyniki nieco słabsze niż przed korekcją, ponadto w przypadku algorytmu SAMDUDE poprawa okazała się być niewielka. Z kolei dla algorytmu BLESS uzyskano poprawę wyników tylko według miar z grupy N, podczas gdy wyniki miar z grupy L oraz miary ξ_{cov} okazały się być słabsze.

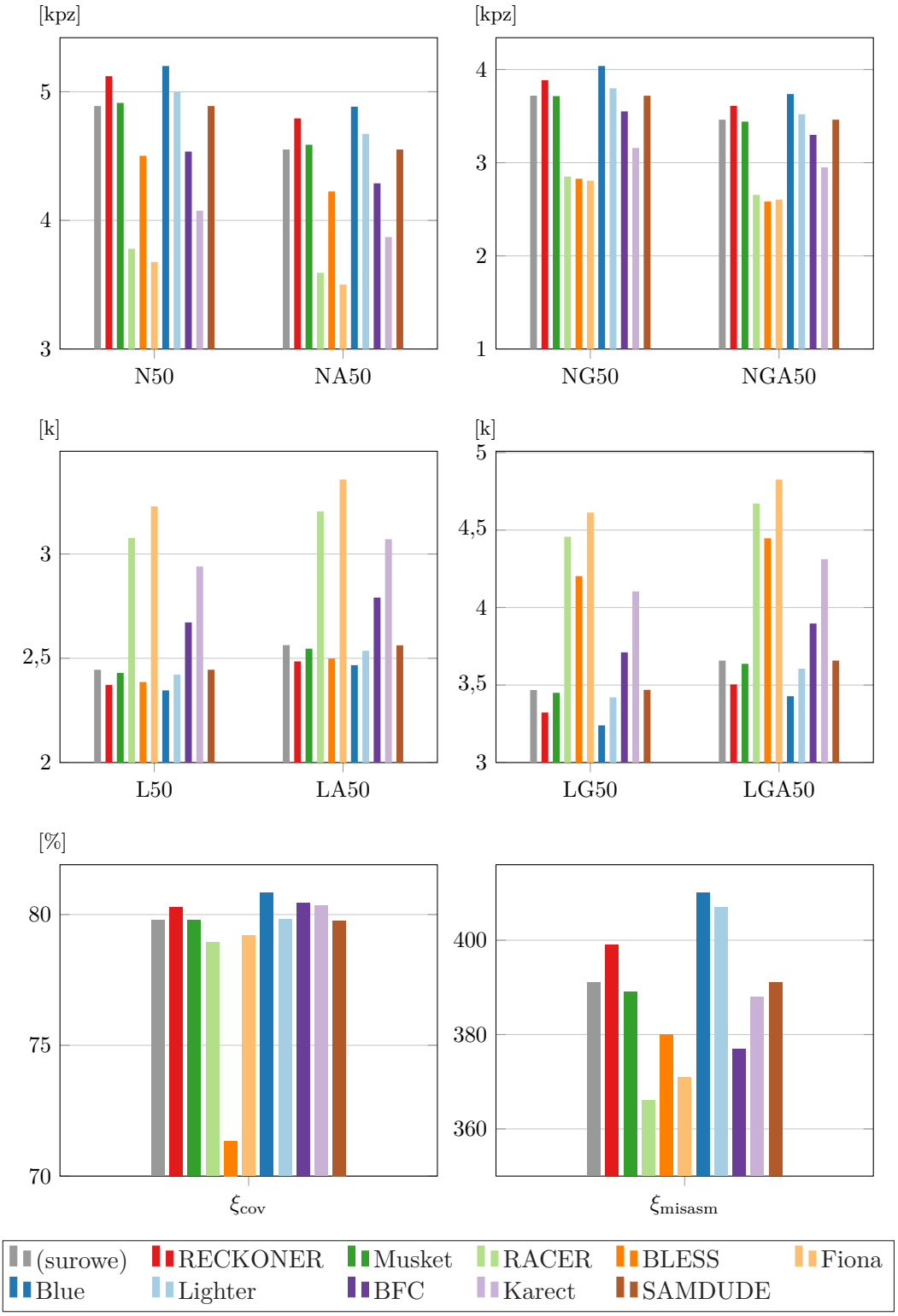
Zestaw V1_60, ze względu na objęcie odczytów ze zredukowaną liczbą poziomów kwantyzacji współczynników jakości, wymaga uwagi głównie w kontekście algorytmów, które wykorzystują wartości współczynników jakości, tj. RECKO-



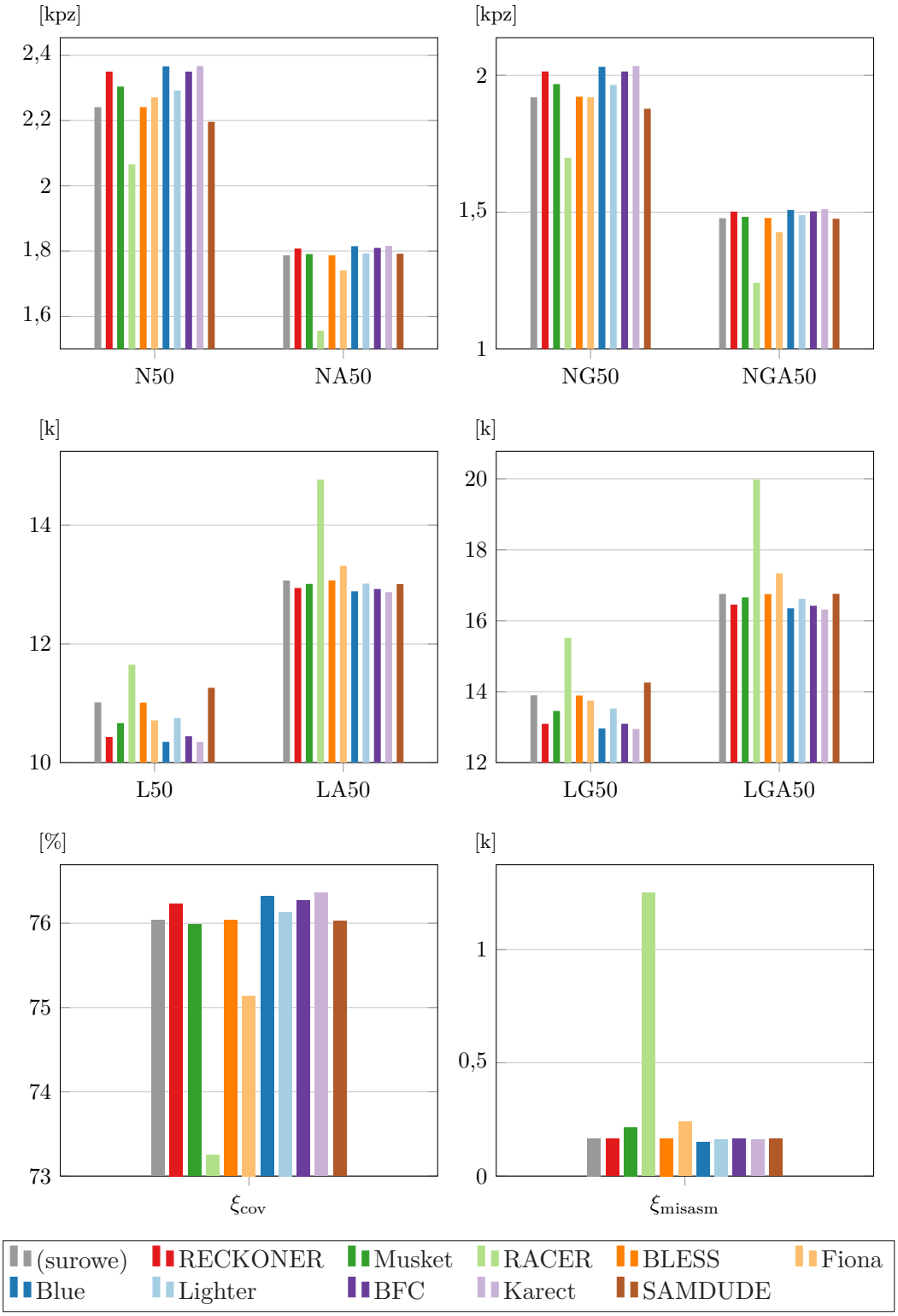
Rysunek 6.5: Wpływ korekcji na wartości miar jakości asemblacji odczytów z zestawu P1_60 (*P. syringae*)



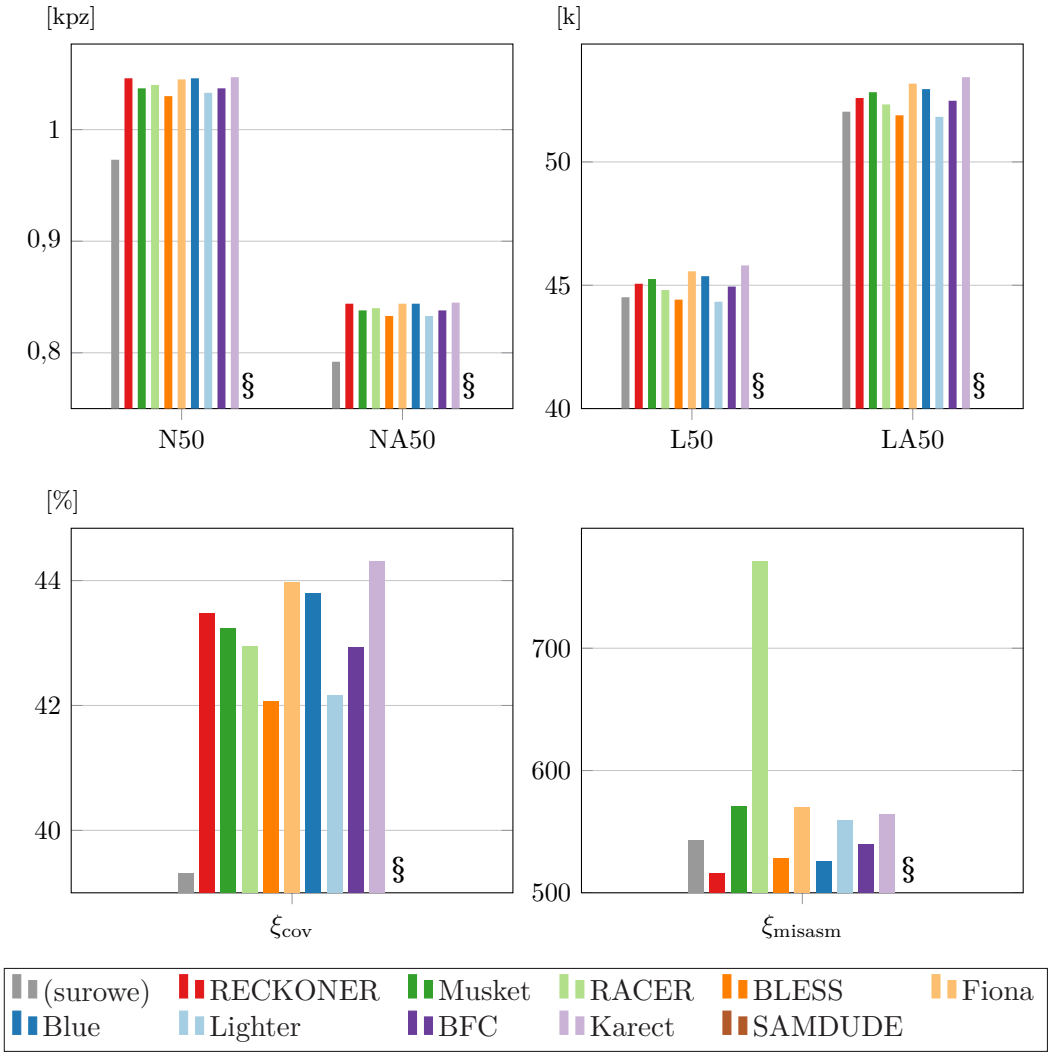
Rysunek 6.6: Wpływ korekcji na wartości miar jakości asemblacji odczytów z zestawu S1 (*S. cerevisiae*)



Rysunek 6.7: Wpływ korekcji na wartości miar jakości asemblacji odczytów z zestawu V1_60 (*C. vulgaris*)

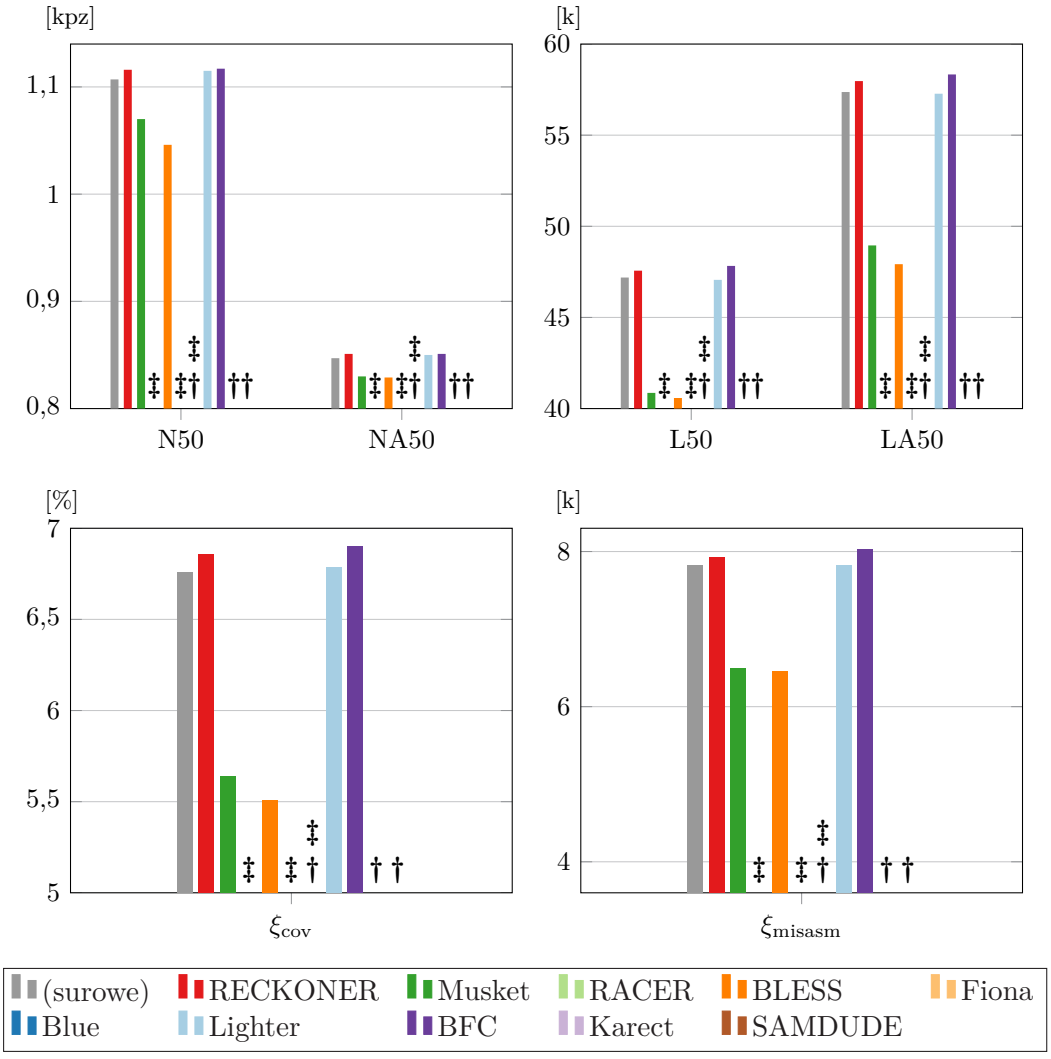


Rysunek 6.8: Wpływ korekcji na wartości miar jakości asemblacji odczytów z zestawu C1 (*C. elegans*)



Rysunek 6.9: Wpływ korekcji na wartości miar jakości asemblacji odczytów z zestawu M1 (*M. acuminata*)

NER, BLESS, Lighter i SAMDUDE. W pozostałych przypadkach współczynniki nie są wykorzystywane lub nie udostępniono takiej informacji. Spośród wymienionych algorytmów poprawa jakości jest widoczna dla algorytmu RECKONER (poza ξ_{misasm}), w niektórych przypadkach nieznacznie dla Lighter i SAMDUDE, z kolei dla BLESS obserwowane jest pogorszenie, z wyjątkiem miar L50, LA50 i ξ_{misasm}. Ponieważ ranking ten jest zgodny z obserwacjami uzyskanymi dla pozostałych zestawów, można przyjąć, że nie wykazano dużego wpływu redukcji liczby dopuszczalnych wartości współczynników jakości na rezultaty korekcji. Spośród pozostałych algorytmów uzyskane wyniki są szczególnie dobre dla algorytmu Blue oraz wyjątkowo słabe dla algorytmów RACER, Fiona i Karect. Ponownie algorytm BLESS spowodował znaczące pogorszenie parametru ξ_{cov}. Warte uwagi jest

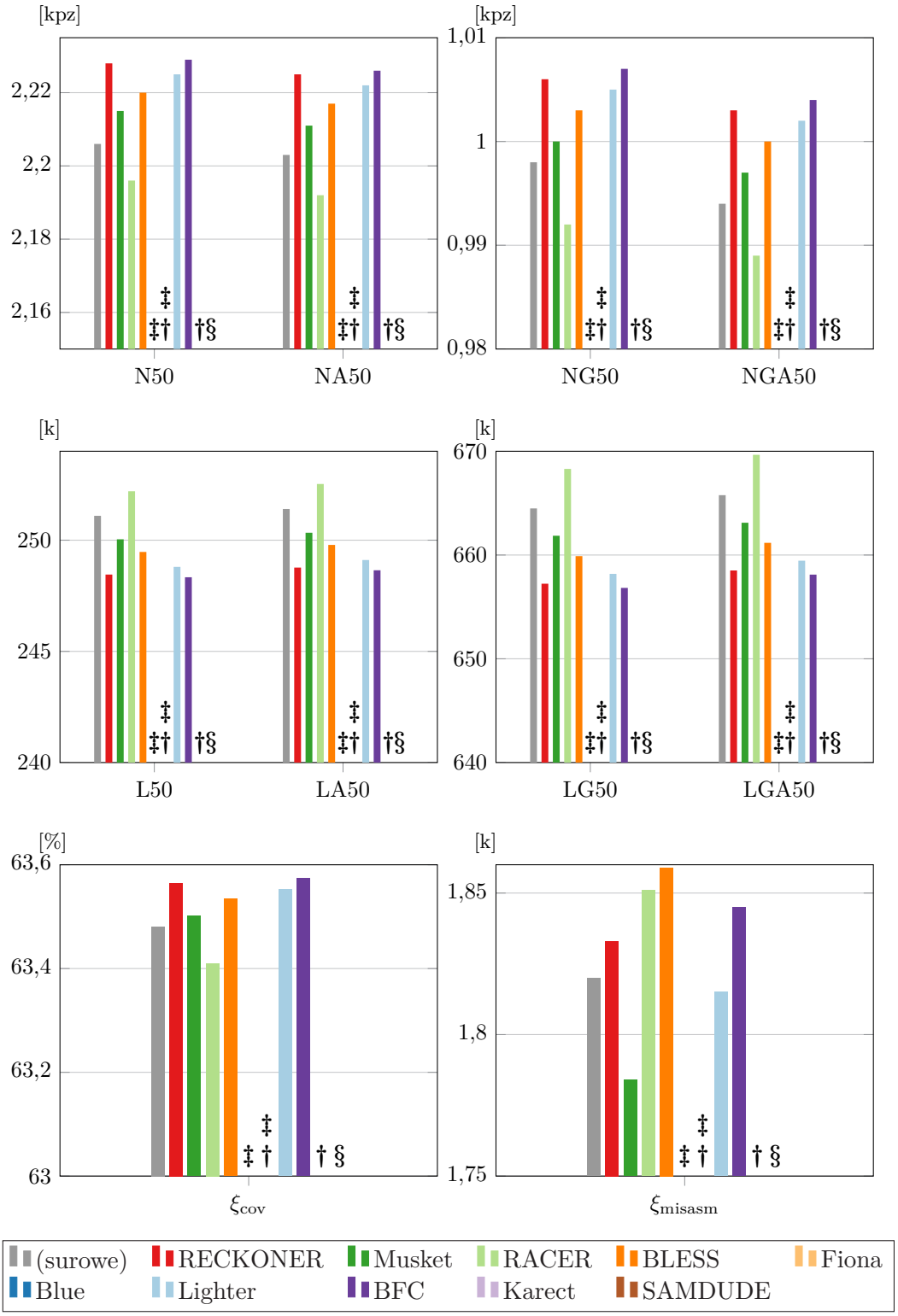


Rysunek 6.10: Wpływ korekcji na wartości miar jakości asemblacji odczytów z zestawu Z1 (*Z. mays*)

zjawisko, że najsłabsze wyniki według miary ξ_{misasm} uzyskano dla algorytmów, które według większości miar okazały się najlepsze.

Korekcja pozwoliła na poprawę jakości odczytów z zestawu C1, choć zastosowanie aż czterech algorytmów spowodowało jej pogorszenie: RACER, BLESS oraz w niewielkim stopniu Fiona i SAMDUDE. Poprawa, choć niewielka, jest obserwowalna szczególnie wyraźnie dla następujących algorytmów: RECKONER, Blue, BFC i Karect. Szczególnie znaczny jest spadek miary ξ_{cov} po zastosowaniu algorytmów RACER i BLESS, a w tym pierwszym przypadku także kilkukrotny wzrost wartości ξ_{misasm}, świadczący, że sekwencje uzyskanych kontigów mogą daleko odstawać od oczekiwanych.

W wynikach zestawu M1 zaobserwowano niespotykane we wcześniejszych ze-



Rysunek 6.11: Wpływ korekcji na wartości miar jakości asemblacji odczytów z zestawu H1–3_45 (*H. sapiens*)

stawach zjawisko. Wszystkie algorytmy korekcji (poza niepowiedzionym SAMDUDE) pozwoliły na uzyskanie poprawy wyników miar z grupy N, a jednocześnie niewielkie pogorszenie wyników miar z grupy L. Jednocześnie wszystkie pozwoliły na znaczną poprawę według miary ξ_{cov}, a RECKONER, BLESS, Blue i BFC także według ξ_{misasm}. Ponownie wyróżniają się algorytmy BLESS oraz RACER, których wykorzystanie skutkowało odpowiednio najmniejszą poprawą ξ_{cov} i największym pogorszeniem ξ_{misasm}. Należy mieć na uwadze, że zestaw ten charakteryzuje się niewielką głębokością sekwencjonowania (ok. 16×).

Zestaw Z1, ze względu na bardzo duży rozmiar, a także przypuszczalnie ze względu na dużą liczbę powtórzeń sekwencji genomu, stanowił źródło problemów w wykonaniu korekcji. Ostatecznie tylko połowa algorytmów została wykonana pomyślnie. Największa poprawa według miar z grup N i L oraz ξ_{cov} jest obserwowalna dla algorytmów RECKONER i BFC, w przypadku Lighter pozytywne zmiany są nieznaczne, a Musket i BLESS spowodowały pogorszenie wartości tych miar. Ponownie rezultaty wskaźnika ξ_{misasm} nie są zgodne z pozostałymi, choć w tym przypadku pogorszenie jakości przez algorytmy najlepsze według innych kryteriów jest minimalne, a poprawa algorytmów najsłabszych — bardzo duża.

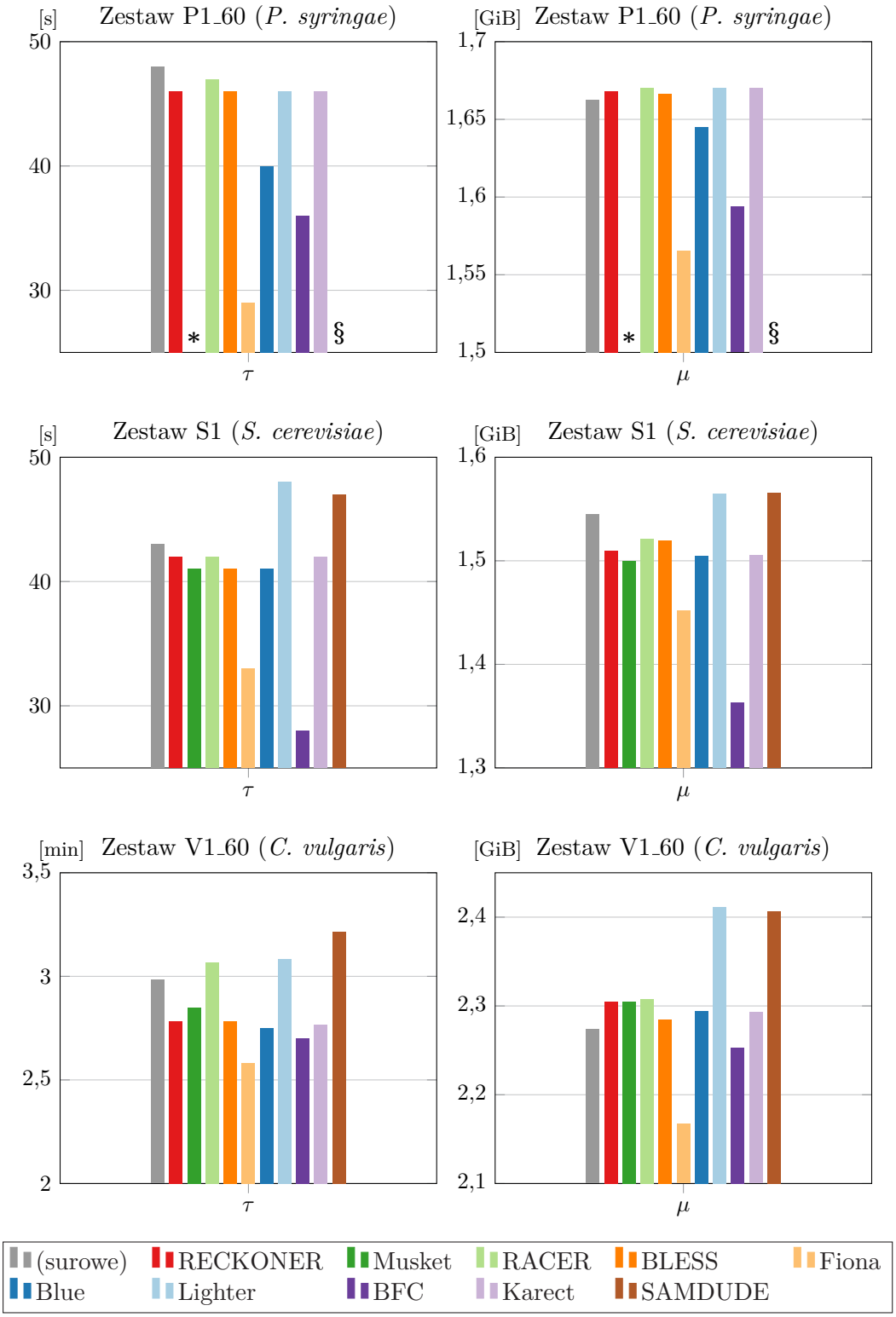
Zestaw H1–3_45 również okazał się być wyzwaniem podczas korekcji, choć w porównaniu do poprzedniego przypadku dodatkowo powiodła się korekcja algorytmem RACER, jednak powodując uzyskanie najsłabszych wyników, jako jedynych jednoznacznie wskazujących na pogorszenie jakości asemblacji. Spośród pozostałych algorytmów najlepsze wyniki uzyskano dla algorytmu RECKONER oraz BFC, choć są to zmiany na poziomie ok. 1%. Jednocześnie wśród tych dwóch wzrost liczby ξ_{misasm} dla algorytmu RECKONER jest dużo mniejszy, a największy spadek jest obserwowany dla „przeciętnego” algorytmu Musket. Największy wzrost wartości tej miary wystąpił dla algorytmu BLESS.

Spośród dziesięciu testowanych algorytmów wszystkie przypadki testowe były możliwe do wykonania tylko poprzez zastosowanie algorytmów RECKONER, Musket, BLESS, Lighter oraz BFC. Prawie zawsze wyraźną poprawę wyników można uzyskać przy pomocy algorytmów Blue (jeżeli jego wykonanie powiedzie się), RECKONER i BFC. Dla algorytmu Karect wyniki, choć nie zawsze możliwe do uzyskania, również zazwyczaj są dosyć dobre, jednak pojawiają się wątpliwości dotyczące odczytów NovaSeq — ta obserwacja wymaga dokładniejszej analizy w celu wyjaśnienia, czy słabe wyniki są efektem charakterystyki odczytów, czy działanie tego algorytmu jest jednak oparte na współczynnikach jakości i redukcja ich dokładności ma znaczący wpływ na uzyskane efekty. Algorytm Musket w większości przypadków pozwolił na niewielką poprawę jakości wyników, z kolei Lighter zazwyczaj nie wywarł wyraźnego wpływu na jakość asemblacji (w szczególności na miarę ξ_{misasm}). Niewielkim wpływem charakteryzuje się też algorytm Fiona, choć należy odnieść do niego podobną uwagę dotyczącą odczytów NovaSeq jak do algorytmu Karect. Zdecydowanie niekorzystne jest zastosowanie algoryt-

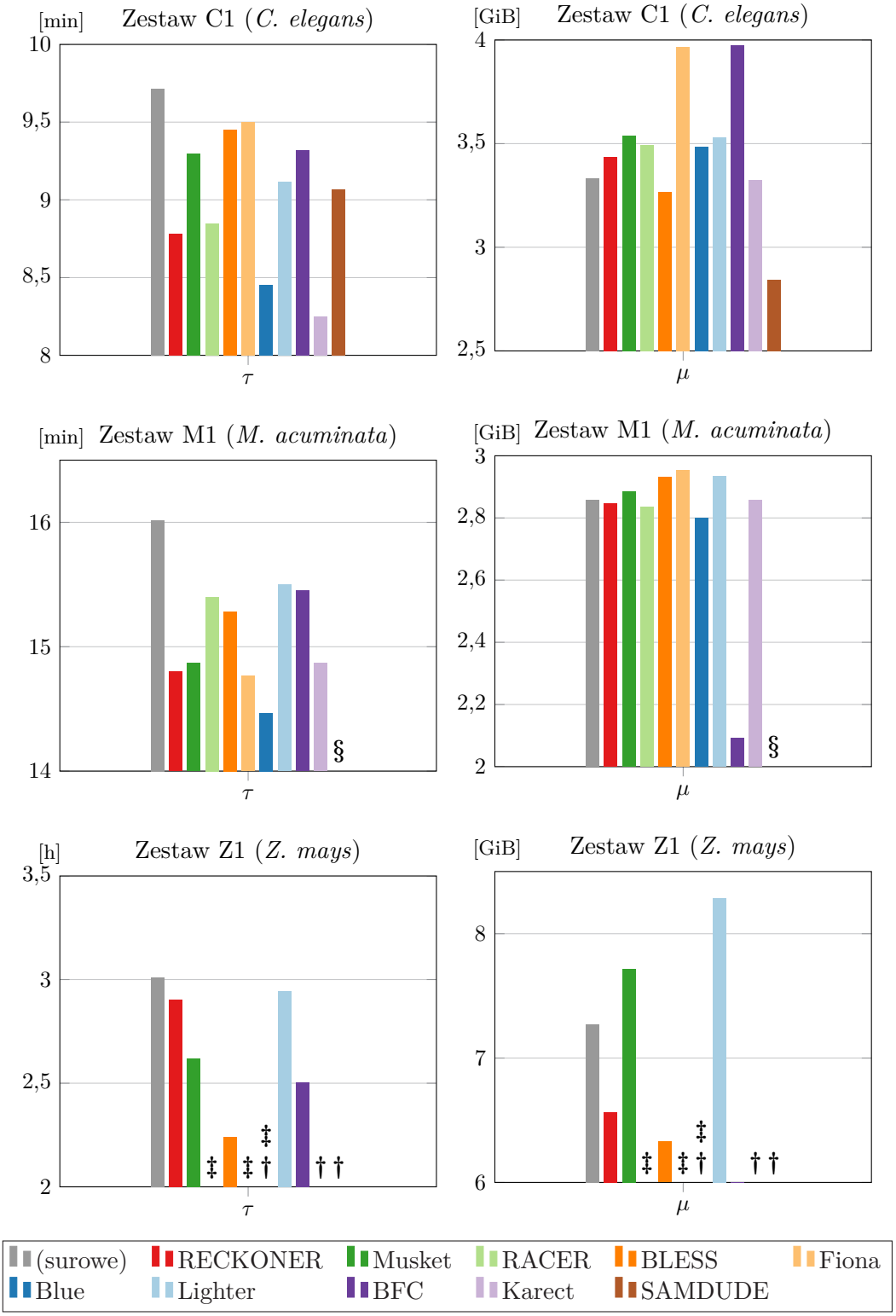
mów BLESS (zwłaszcza w kontekście wartości miary ξ_{cov}), RACER i SAMDUDE.

Warta uwagi jest także obserwacja, że często istnieje odwrotna zależność między oceną algorytmu przy pomocy miary ξ_{misasm} a pozostałymi miarami. Można by tu postawić hipotezę, że działanie słabszych algorytmów jest bardziej zachowawcze, stąd ich zastosowanie nie powoduje dużej poprawy, ale też redukuje ryzyko pojawienia się błędnych kontigów. Jednakże przeczy temu fakt, że (szczególnie dla algorytmu BLESS) korekcja powoduje spadek jakości wyników poniżej poziomu odczytów nieskorygowanych, zatem dopuszcza on wykonywanie bardziej „odważnych" zmian w odczytach.

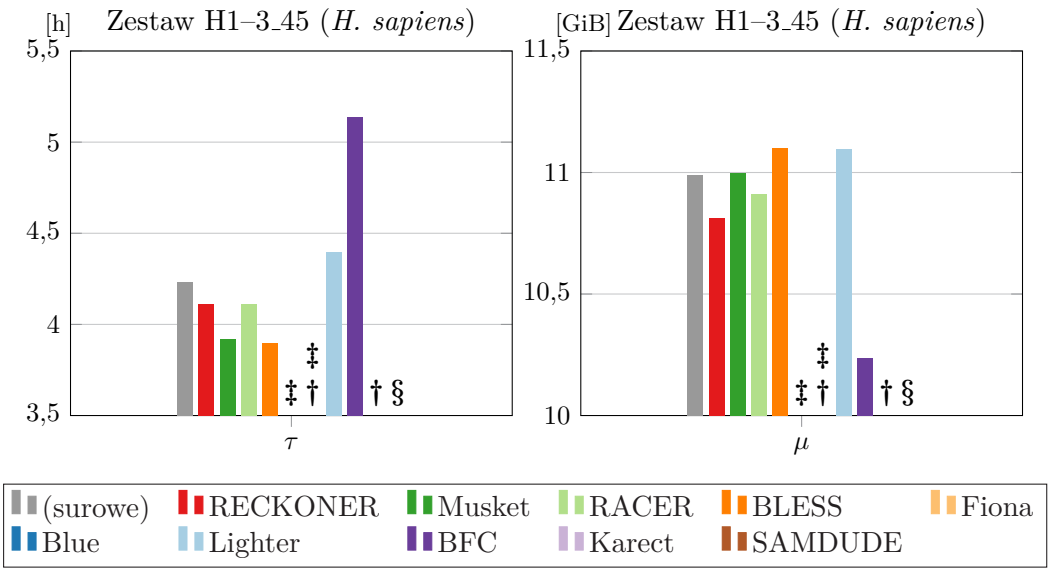
Wpływ korekcji na zapotrzebowanie asemblacji na zasoby Analiza wpływu korekcji na zapotrzebowanie na czas obliczeń τ oraz pamięć operacyjną μ asemblacji przynosi niejednoznaczne obserwacje, choć w większości przypadków można zaobserwować spadek czasu asemblacji. Dla zestawów P1_60 i S1 zmniejszenie zapotrzebowania na czas obliczeń oraz pamięć wyraźnie występuje tylko dla algorytmów Fiona i BFC, które nie wyróżniają się jakością wyników. Korekcja pozostałych algorytmów przyniosła tylko symboliczne spadki czasu, oraz, poza algorytmami Fiona, Blue i BFC, niewielki wzrost zapotrzebowania na pamięć. Dla zestawu S1 po korekcji algorytmami Lighter i SAMDUDE jest widoczny wzrost zapotrzebowania na oba zasoby, a algorytmy te pozwoliły na uzyskanie odpowiednio przeciętnych (choć lepszych w stosunku do oryginalnych odczytów) i słabych wyników. Dla zestawu V1_60 nastąpił wyraźny, choć nadal niewielki spadek zapotrzebowania na pamięć po korekcji algorytmem Fiona, a po korekcji algorytmami Lighter i SAMDUDE wzrost zapotrzebowania na pamięć (ale z mniej widocznym rezultatem w kategorii czasu). Dla zestawu C1 nie zaobserwowano znacznego odstawania wyników dla algorytmu RACER, który spowodował uzyskanie wyników słabych jakościowo, choć należy zaznaczyć, że wszystkie algorytmy spowodowały skrócenie czasu asemblacji i — w większości przypadków — niewielkie zwiększenie zapotrzebowania na pamięć (przy czym wyraźniejsze dla algorytmów Fiona i BFC, choć ponownie trudno skorelować je z jakością wyników). Dla zestawu M1 zasoby potrzebne po korekcji algorytmem RACER również nie uległy znaczącej zmianie w porównaniu do innych algorytmów. I tutaj obserwowalne jest pewne przyspieszenie asemblacji, a jednocześnie proporcjonalnie bardzo duży (choć praktycznie i tak nieznaczący) spadek zapotrzebowania na pamięć po korekcji algorytmem BFC, który jakościowo wypadł przeciętnie. W przypadku zestawu Z1 warty zainteresowania może być wynoszący nawet 45 min spadek czasu asemblacji, choć występujący dla słabego algorytmu BLESS, oraz ok. 30 min dla równie słabego Musket i jednego z najlepszych BFC. Jednocześnie wyniki te w ogóle wydają się nie korelować z zapotrzebowaniem asemblera na pamięć. W ostatnim zestawie H1–3_45 korekcja wywarła niewielki wpływ na zapotrzebowanie asemblera, choć interesujący jest wzrost czasu asemblacji o przeszło godzinę po korek-



Rysunek 6.12: Wpływ korekcji na zapotrzebowanie na zasoby asemblacji odczytów z zestawów P1_60, S1, V1_60



Rysunek 6.13: Wpływ korekcji na zapotrzebowanie na zasoby asemblacji odczytów z zestawów C1, M1, Z1



Rysunek 6.14: Wpływ korekcji na zapotrzebowanie na zasoby asemblacji odczytów z zestawu H1–3_45

cji algorytmem BFC, a przy tym dziesięcioprocentowy spadek zapotrzebowania asemblera na pamięć.

Podsumowując, analiza wpływu korekcji na zapotrzebowanie na zasoby obliczeniowe przez asembler wykazuje, że nie jest on kwestią istotną. Ewentualne próby osiągnięcia korzyści wiązałyby się z trudnym do przeprowadzenia wyborem algorytmu, a ewentualna korzyść wiązałaby się głównie z niewielkim spadkiem czasu asemblacji. Nie można jednak wykluczyć bardziej obiecującej poprawy w przypadku starszych, słabiej zoptymalizowanych pod kątem zapotrzebowania na zasoby asemblerów.

Mapowanie

Mapowanie odczytów stanowi wstępny etap niektórych innych zadań przetwarzania odczytów, w tym detekcji wariantów. Także i w tym przypadku narzędzia przetwarzające charakteryzują się częściową odpornością na błędy sekwencjonowania [89], choć należy zaznaczyć, że z punktu widzenia algorytmu mapującego błąd sekwencjonowania nie musi być odróżnialny od naturalnej zmienności. Mimo tego, jak zauważono w suplemencie pracy [128], w przypadku przeprowadzenia skutecznej korekcji dla dużej części odczytów liczba modyfikacji koniecznych do ich dopasowania powinna być mniejsza. W celu wykonania mapowania wykorzystano algorytm BWA [129], mając na uwadze jego bardzo dużą popularność.

Zestaw miar wykorzystanych w literaturze do analizy wyników mapowania odczytów został przedstawiony w podrozdziale 4.4.1. W omówionych w niniejszej pracy eksperymentach zdecydowano na wyznaczenie histogramu wartości odle-

głości edycyjnej $\xi_{\mathrm{ndist}}(d_{\mathrm{e}})$, a także dodatkowo wprowadzono miary ξ_{map1} oraz $\xi_{\mathrm{map1+}}$, oznaczające odpowiednio część odczytów, które zostały zmapowane do genomu jednokrotnie oraz wielokrotnie. Miary te niekoniecznie stanowią informację o skuteczności korekcji (np. prawidłowa korekcja pewnego odczytu może zarówno spowodować przeniesienie go z grupy zmapowanych jednokrotnie do grupy zmapowanych wielokrotnie, jak również na odwrót), jednak dają wgląd w zmiany, jakie nastąpiły w rezultatach mapowania. Tym samym przedstawione wyniki zostały uwzględnione tylko w celu zobrazowania wpływu korekcji, nie stanowiąc kryterium oceny algorytmów. Niezależnie przeprowadzono analizę liczby różnic typu indel między odczytami a podsłowami genomu, do których zostały dopasowane. W analizie wykorzystano miary ξ_{ins} oraz ξ_{del}, oznaczające stosunek liczby odczytów zawierających różnicę odpowiednio typu insercja i delecja, do liczby wszystkich odczytów.

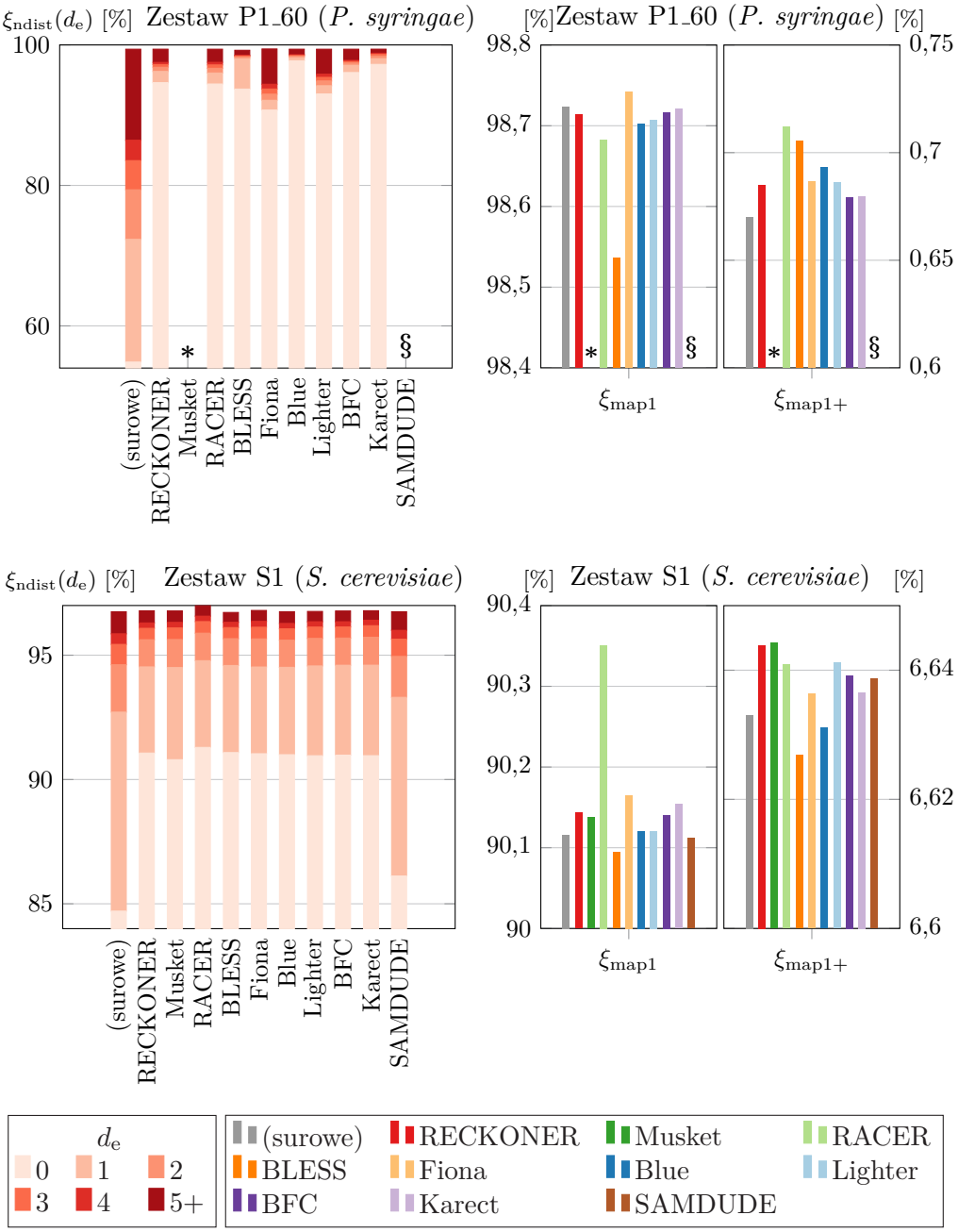
Wszystkie wykorzystane wartości zostały uzyskane w wyniku analizy atrybutów przyporządkowanych poszczególnym mapowaniom w pliku SAM[3]. W eksperymentach nie wykorzystano wyników analizy współczynników jakości mapowania, które są obecne w pliku SAM, ponieważ ich średnia arytmetyczna w wyniku korekcji ulegała bardzo niewielkim zmianom, a — potraktowana jako alternatywna dla średniej — mediana nie ulegała zmianie.

Na rys. 6.15–6.18 przedstawiono odpowiednio wyrażone procentowo wyniki $\xi_{\mathrm{ndist}}(d_{\mathrm{e}})$ dla $d_{\mathrm{e}} \in \{0, 1, 2, 3, 4, 5+\}$, gdzie 5+ oznacza odległość większą lub równą 5, a także wartości ξ_{map1} oraz $\xi_{\mathrm{map1+}}$. Mapowaniu poddano te same skorygowane odczyty, które zostały wykorzystane w celu przeprowadzenia asemblacji.

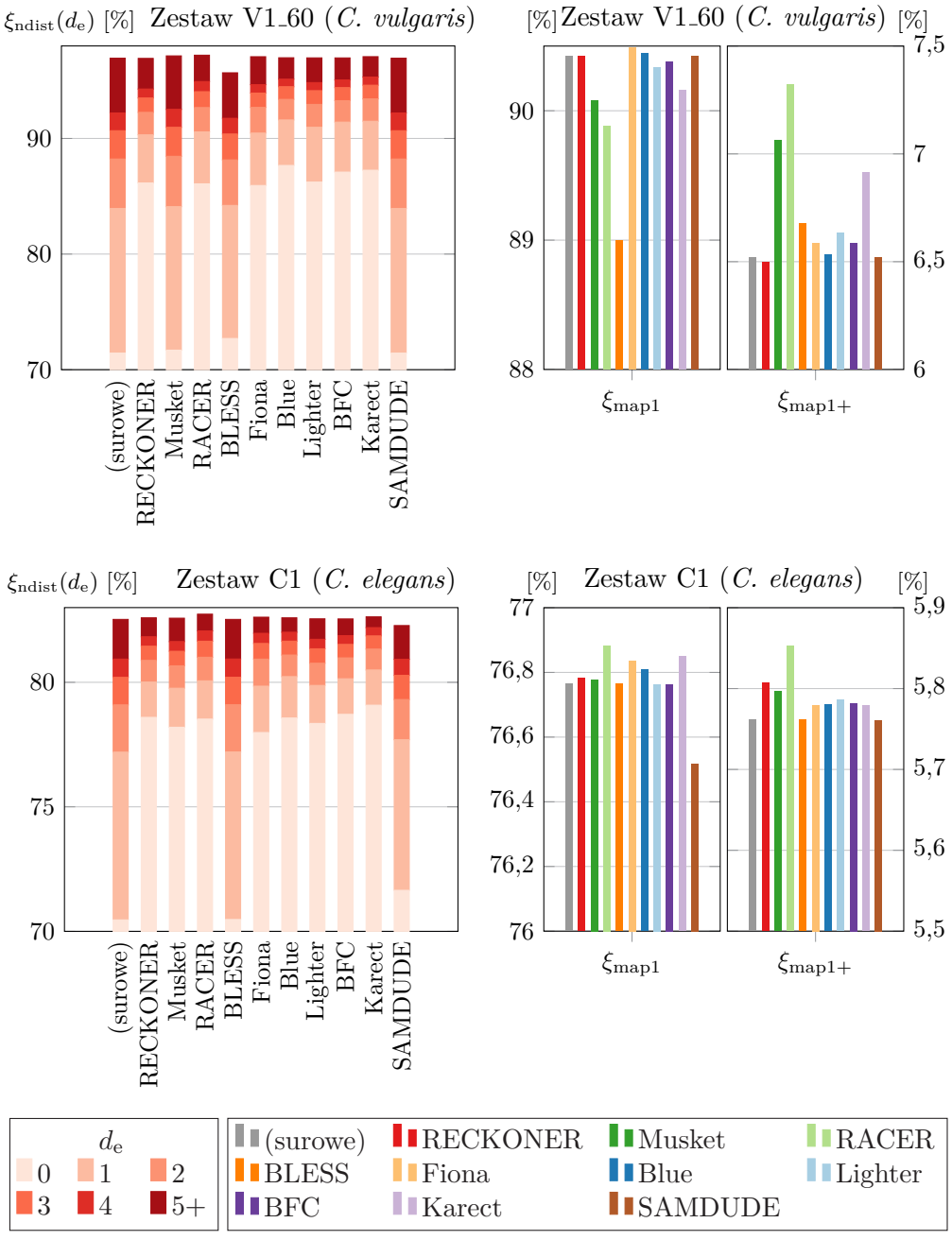
W wyniku korekcji odczytów z zestawu P1_60 wszystkie wykonane z powodzeniem algorytmy korekcji pozwoliły na znaczące zmniejszenie odległości edycyjnej między odczytami a podsłowami genomu, w tym także w sytuacjach, gdy odczyty zawierały wiele błędów (wartości 5+). Jednocześnie nie zaobserwowano wyraźnego zwiększenia liczby odczytów, które w jakikolwiek sposób zostały zmapowane (wartości ξ_{map1} oraz $\xi_{\mathrm{map1+}}$), co wiązałoby się z lepszym wykorzystaniem danych w odczytach. Wyróżniające się algorytmy to BLESS, którego zastosowanie znacząco zmniejszyło liczbę odczytów, które zostały dopasowane do jednego miejsca w genomie (prawdopodobnie dopasowane błędnie, o czym świadczą słabe wyniki asemblacji), oraz Fiona, którego wykorzystanie pozwoliło na pewne zwiększenie liczby zmapowanych z sukcesem odczytów.

W przypadku zestawu S1 poprawa jakości odczytów jest wyraźnie mniejsza, ale nadal można uznać ją za znaczącą. Wyjątkiem jest algorytm RACER, którego działanie spowodowało wzrost łącznej liczby zmapowanych odczytów o ok. 0,25 punktu procentowego (co idzie w parze ze słabymi wynikami algorytmu dla

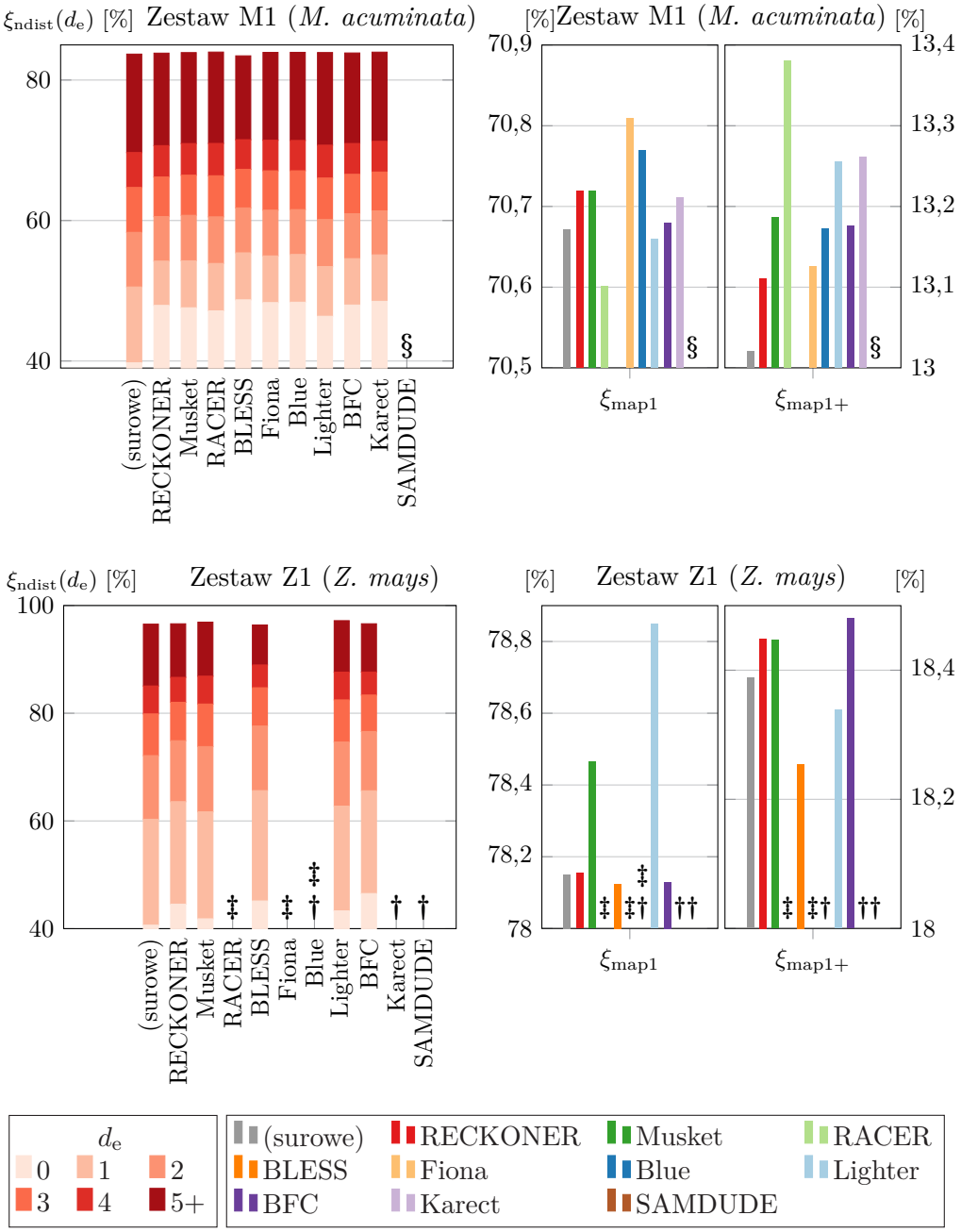
[3]Należy zaznaczyć, że odczyty w eksperymentach związanych z algorytmem SAMDUDE zostały poddane mapowaniu dwukrotnie: w celu uzyskania wejściowego algorytmu korekcji pliku SAM, a następnie, po przekształceniu wyjściowego pliku SAM do plików FASTQ, ponownie, w celu uzyskania przedstawionych wyników.



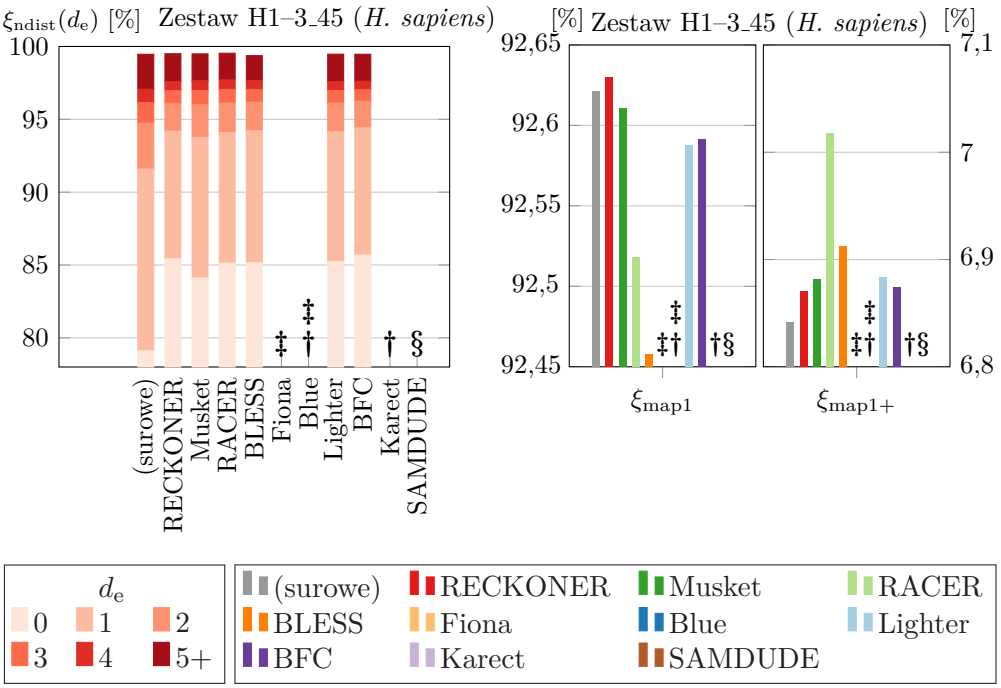
Rysunek 6.15: Wpływ korekcji na jakość mapowania odczytów z zestawów P1_60 i S1



Rysunek 6.16: Wpływ korekcji na jakość mapowania odczytów z zestawów V1_60 i C1



Rysunek 6.17: Wpływ korekcji na jakość mapowania odczytów z zestawów M1 i Z1



Rysunek 6.18: Wpływ korekcji na jakość mapowania odczytów z zestawu H1–3_45

asemblacji). Zdecydowanie mały wpływ na redukcję liczby zmian wprowadzonych przez algorytm mapujący wystąpił w efekcie korekcji algorytmem SAMDUDE.

W zestawie V1_60 wyraźne zmiany dotyczą rozmiaru grupy odczytów dopasowanych z odległością edycyjną równą 1 albo 2. Wyjątek stanowi algorytm BLESS, którego zastosowanie spowodowało zmniejszenie liczby zmapowanych jednokrotnie (a w konsekwencji — ogólnie zmapowanych) odczytów o ok. 1 punkt procentowy, przypuszczalnie korespondujące z dużym spadkiem wartości wskaźnika ξ_{cov} w wynikach asemblacji. Jednocześnie w wyniku korekcji algorytmami Musket i SAMDUDE liczba zmian wprowadzonych przez algorytm mapujący nie uległa widocznym zmianom. W przypadku wskaźnika liczby odczytów zmapowanych jednokrotnie pozytywnie wyróżnia się algorytm Fiona. Dodatkowo interesujący jest wyraźny wzrost liczby odczytów zmapowanych w więcej niż jedno miejsce genomu po korekcji algorytmami Musket, RACER i Karect. Tak duże zmiany mogą sugerować spadek jakości będącym efektem zbyt silnego wzajemnego upodobnienia odczytów z fragmentów genomu o podobnych sekwencjach. Te wyniki słabo wpasowują się w obserwacje wyników asemblacji, gdzie algorytmy RACER, Fiona i Karect spowodowały znaczny spadek jakości według miar z grup N i L, a korekcja algorytmem Musket skutkowała tylko niewielką zmianą rezultatów.

Także w przypadku zestawu C1 zmiany obejmowały głównie spadek odległości edycyjnej. Zmiany liczby zmapowanych odczytów są niewielkie, choć wszystkie algorytmy spowodowały wzrost liczby odczytów zmapowanych wielokrotnie,

szczególnie w przypadku algorytmu RACER, którego zastosowanie ponownie spowodowało ogólny wzrost liczby zmapowanych odczytów. Także w tym przypadku trudno nie zauważyć związku tego zjawiska z odpowiednimi wynikami asemblacji. Odstawanie wspomnianego algorytmu jest też dostrzegalne w wynikach asemblacji. Z kolei algorytmy BLESS oraz SAMDUDE spowodowały niewielką zmianę w porównaniu z odczytami surowymi. Należy zaznaczyć, że łącznie powiodło się mapowanie ok. 82% odczytów.

Podobnie słaby ogólny wynik mapowania odczytów jest obserwowalny dla odczytów z zestawu M1. Wyraźnym zjawiskiem jest przede wszystkim spadek w porównaniu do wcześniejszych zestawów liczby odczytów dopasowanych z odległością edycyjną równą 1, na rzecz tych z odległością równą 0. Może być to efektem dużej liczby wariantów albo trudnościami w procesie mapowania. Wyniki uzyskane dla różnych algorytmów korekcji są podobne, choć ponownie wyróżniają się algorytmu BLESS (wyraźne zmniejszenie liczby zmapowanych odczytów) oraz RACER (zwiększenie liczby odczytów, które mapują się w kilku miejscach w genomie).

Odczyty z zestawu Z1 w większości zostały zmapowane do genomu referencyjnego z odległością edycyjną równą co najmniej 1. W wyniku korekcji własność ta została ograniczona tylko nieznacznie. Jednocześnie obserwowalne jest zwiększenie liczby (jednokrotnie) zmapowanych odczytów w wyniku korekcji algorytmem Lighter o ok. 0,5 punktu procentowego oraz algorytmem Musket o ok. 0,2 punktu. Pozostałe algorytmy zwiększyły liczbę zmapowanych odczytów nieznacznie (RECKONER i BFC) lub spowodowały spadek (BLESS). Warto zaznaczyć, że wartości miary $\xi_{\text{map1+}}$ są większe niż w przypadku pozostałych zestawów, co może być efektem dużej liczby powtórzeń w genomie *Z. mays*. Wyniki te są odmienne od wyników asemblacji, gdzie efekty wyraźnie różne od pozostałych zostały uzyskane dla algorytmów Musket oraz BLESS, natomiast wyniki odczytów skorygowanych przy pomocy algorytmu Lighter były podobne do wyników uzyskanych dla odczytów nieskorygowanych.

W odczytach z zestawu H1–3_45 wystąpił duży wzrost liczby odczytów zmapowanych z odległością edycyjną równą 0, szczególnie w efekcie zmniejszenia przypadków, gdy konieczne było wprowadzenie jednej lub dwóch zmian. Jednocześnie wystąpiły niewielkie zmiany liczb odczytów zmapowanych jednokrotnie (choć wyraźny jest spadek dla algorytmów RACER i BLESS) oraz wzrost liczby odczytów zmapowanych wielokrotnie po korekcji wszystkimi algorytmami, szczególnie algorytmem RACER. W porównaniu do wyników asemblacji (poza algorytmem RACER) pozytywnie wyróżniające się w tamtym przypadku algorytmy RECKONER i BFC w eksperymentach mapowania cechują się nieco zwiększoną w stosunku do innych algorytmów liczbą odczytów zmapowanych bez różnic, choć zmiany w przypadku liczby dokonanych mapowań odczytów są różne (wzrost dla $\xi_{\text{map1+}}$ dla obu, w przypadku ξ_{map1} wzrost tylko dla RECKONER, a spadek

dla BFC).

W większości przypadków rezultatem korekcji jest wyraźne zmniejszenie odległości edycyjnej między odczytami a genomem referencyjnym, co świadczy o redukcji liczby błędów i — patrząc z perspektywy mapowania — może sugerować, że może to wywrzeć pozytywny wpływ na działanie potoków obliczeniowych wykorzystujących odczyty mapowania. Mimo to w niewielkim stopniu zmianom ulegają liczby odczytów, których (jednokrotne lub wielokrotne) mapowanie powiodło się, stąd należy ocenić, że korekcja w niewielkim stopniu może wpływać na efektywność działania samych algorytmów mapujących i skuteczność ich wykorzystania. Poprawa obu miar liczby dopasowania odczytów konsekwentnie we wszystkich przypadkach występuje dla algorytmu Fiona, choć nie jest ona duża. W przypadku większości pozostałych algorytmów również można zaobserwować wzrost sumy tych wskaźników, jednakże należy przyznać, że na bieżącym poziomie eksperymentów trudno jest zweryfikować, jakie zmiany przynależności konkretnych odczytów do grup zmapowanych lub niezmapowanych zaszły w efekcie korekcji.

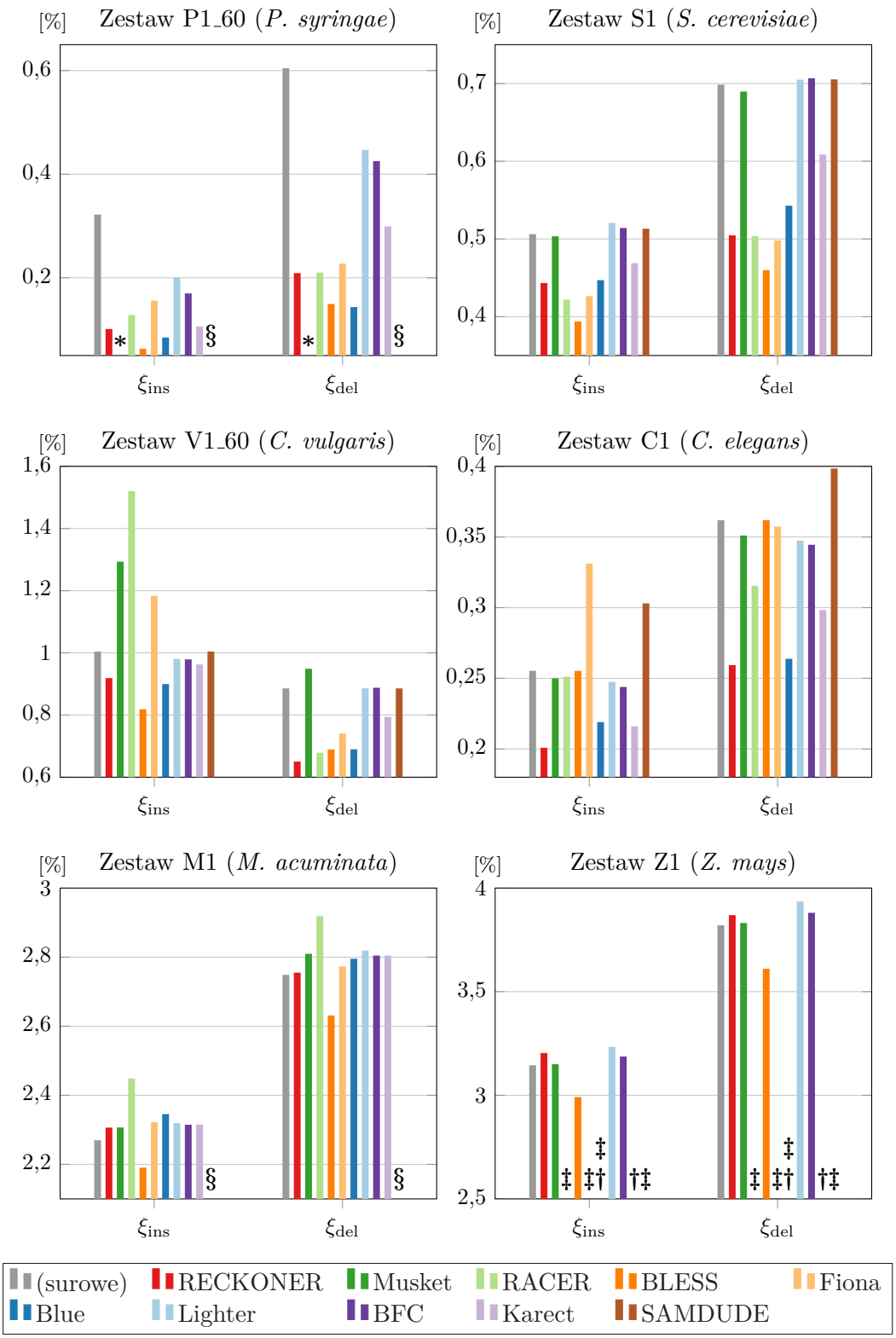
W przeprowadzonych eksperymentach jednoznacznie niekorzystnie wyróżniają się algorytmy RACER, BLESS oraz SAMDUDE. Pierwszy z nich cechuje się powodowaniem częstego znaczącego wzrostu liczby zmapowanych odczytów, choć w kontekście uzyskanych wcześniej wyników asemblacji należy przyjąć, że są to zmiany błędne. BLESS z kolei zwykle powoduje spadek liczby zmapowanych odczytów, skutkując późniejszym słabszym wykorzystaniem informacji zawartych w odczytach, których mapowanie nie powiodło się, natomiast SAMDUDE nie powoduje wyraźnej redukcji zmian wprowadzanych przez algorytm mapujący, powodując jednocześnie niewielki spadek liczby łącznie zmapowanych odczytów.

Wpływ korekcji na liczbę różnic typu indel Na rys. 6.19 oraz 6.20 przedstawiono wyrażone procentowo wartości miar ξ_{ins} oraz ξ_{del}. Uzyskane charakterystyki mają charakter uproszczony, gdyż obejmują tylko informację o obecności pewnej insercji albo delecji w danym odczycie, nie uwzględniając ich liczby. Nie wzięto pod uwagę także długości tych zmian. Należy mieć również na uwadze, że są to miary pośrednie, których wartości są obarczone charakterystyką algorytmu mapującego.

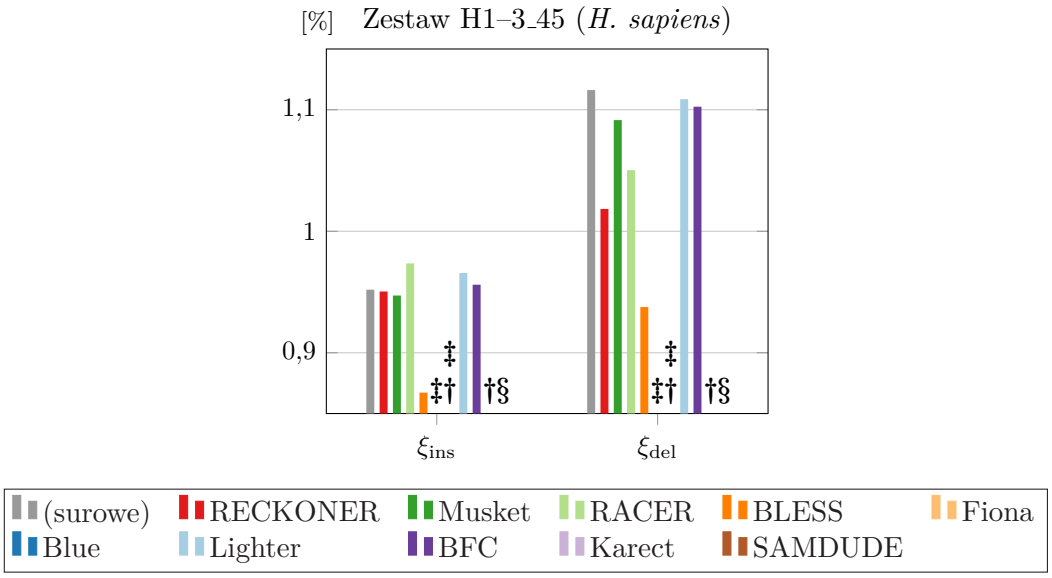
Poza algorytmem RECKONER, zgodnie z tabelą 4.1, jawna korekcja błędów typu indel została uwzględniona w algorytmach Fiona oraz Blue[4]. Stąd też analiza może posłużyć do oceny skuteczności strategii działania tego rodzaju korekcji w wymienionych algorytmach.

Dla zestawu P1_60 prawie wszystkie algorytmy pozwoliły na redukcję liczby różnic, najskuteczniej w wyniku korekcji algorytmami BLESS oraz Blue. W przypadku tego pierwszego jest to zaskakujące ze względu na brak strategii korekcji

[4]W ogólnym przypadku do tej grupy można zaliczyć też algorytm Karect, jednak w przeznaczonym dla korekcji odczytów Illumina trybie wyznaczania odległości edycyjnej Hamminga nie jest on wykorzystywany.



Rysunek 6.19: Wpływ korekcji na liczbę różnic typu indel odczytów z zestawów P1_60, S1, V1_60, C1, M1, Z1



Rysunek 6.20: Wpływ korekcji na liczbę różnic typu indel odczytów z zestawu H1–3_45

takich błędów, stąd prawdopodobnie duże znaczenie ma przewidziane w tym algorytmie obcinanie niektórych odczytów. Algorytm RECKONER również pozwolił na znaczną redukcję liczby różnic, choć uzyskane wyniki są podobne do algorytmu RACER, który w ogólnej ocenie mapowania uzyskał raczej słabe wyniki. W nieco mniejszym, choć nadal wyraźnym stopniu redukcja nastąpiła w wyniku pracy algorytmów Fiona, Lighter, BFC oraz Karect.

W przypadku zestawu S1 uzyskano wyraźną, choć mniejszą niż w poprzednim zestawie poprawę jakości dla algorytmów wyposażonych w mechanizm korekcji błędów typu indel, a także dla algorytmów BLESS, RACER oraz Karect. W pozostałych przypadkach uzyskane wyniki były podobne jak dla surowych odczytów, a czasem nieco słabsze.

W wynikach zestawu V1_60 zaobserwowano pewną poprawę wyników (RECKONER, BLESS, w przypadku błędów typu delecja także RACER, Fiona i Blue). Wśród algorytmów RACER, Musket i Fiona zanotowano wyraźne pogorszenie rezultatów po korekcji algorytmami w kontekście liczby insercji. W dwóch pierwszych przypadkach jest to o tyle interesujące, że nie przewidziano w nich korekcji błędów typu indel. Wśród pozostałych przypadków nie zaobserwowano znaczących zmian.

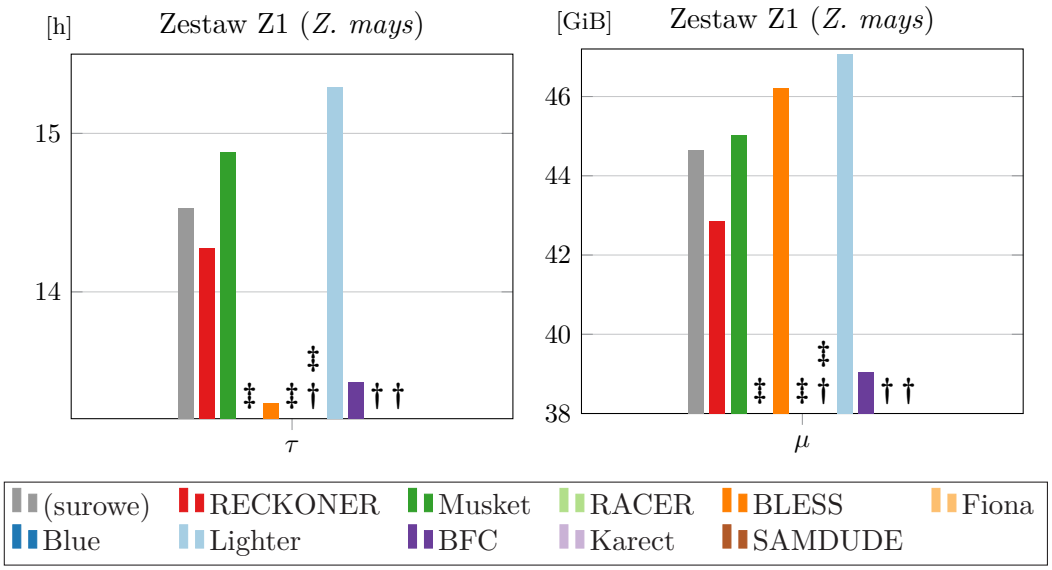
W korekcji zestawu C1 poprawę uzyskano dla algorytmów RECKONER i BLESS oraz — w nieco mniejszym stopniu — Blue i Karect. Pozostałe algorytmy nie przyniosły dużych zmian, z wyjątkiem SAMDUDE, którego wykorzystanie spowodowało wyraźne zwiększenie liczby różnic, oraz Fiona, dla którego nastąpiło pogorszenie według miary ξ_{ins}.

W wynikach zestawu M1 obserwowalny jest wzrost liczby różnic po zastosowaniu algorytmu korekcji. Wyróżniają się algorytmy BLESS (którego wykorzystanie spowodowało wyraźny spadek liczby różnic) oraz RACER (z największym wzrostem). Warto zwrócić uwagę na stosunkowo dużą liczbę różnic odczytów z tego zestawu, z którą koresponduje niewielka różnica między wynikami odczytów surowych oraz skorygowanych. Druga z tych obserwacji prawdopodobnie jest efektem niewielkiej głębokości sekwencjonowania tego zestawu (16×), będącej przeszkodą w skutecznej korekcji błędów typu indel.

Jeszcze większa ogólna liczba różnic między odczytami a genomem jest obserwowalna dla zestawu Z1, co ma o tyle duże znaczenie, że łącznie stosunek liczby różnic typu indel do liczby odczytów wynosi blisko 7%, stąd wniosek, że wszelkie przetwarzanie tego zestawu (co właściwie jest echem charakterystyki genomu) może stanowić wyzwanie. Ponownie najlepsze rezultaty uzyskano dla algorytmu BLESS, który jako jedyny pozwolił na poprawę wyników. Zmianę dla algorytmów Musket oraz BFC w ocenie według tej miary należy ocenić jako nieznaczącą, natomiast wykorzystanie algorytmów Lighter oraz RECKONER powodujące nieznaczne pogorszenie wyników.

Algorytm BLESS okazał się być najlepszy również przy korekcji odczytów z zestawu H1–3_45. Na drugim miejscu należy umieścić algorytm RECKONER, który pozwolił na redukcję szczególnie różnic typu delecja, a na kolejnym — algorytm Musket. Dla pozostałych algorytmów zanotowano niewielkie zwiększenie różnic typu insercja oraz mniejsze różnic typu delecja.

Podsumowując, wyniki wszystkich zestawów wskazują, że najskuteczniej liczbę różnic typu indel redukuje algorytm BLESS. Trudno jest to połączyć ze słabymi wynikami asemblacji oraz pozostałymi wynikami mapowania w inny sposób, niż zwracając uwagę na częste przypadki obcinania odczytów albo stawiając hipotezę, że liczba odczytów uszkodzonych w wyniku korekcji tym algorytmem jest na tyle duża, że powoduje trudności w ich mapowaniu. Spośród algorytmów wyposażonych w mechanizm korekcji błędów typu indel najlepsze wyniki uzyskano dla algorytmu RECKONER, który w większości przypadków pozwolił na redukcję liczby tych błędów lub na wzrost w minimalnym stopniu. W dalszej kolejności należy wyróżnić korzystne działanie algorytmów Blue oraz zwrócić uwagę, że algorytm z tej grupy, Fiona, pozwalał zazwyczaj na osiągnięcie słabych wyników. Jakość algorytmów Musket, RACER, Lighter, BFC wahała się w zależności od zestawu, z kolei zastosowanie algorytm Karectw większości przypadków powodowało pewną poprawę wyników. W oparciu o wszystkie wyniki można wysnuć wniosek, że dostosowanie algorytmów do korekcji błędów typu indel przynosi korzyści, choć są one niewielkie. Jedynym algorytmem, który nie pozwolił na skuteczną korekcję błędów typu indel jest SAMDUDE.



Rysunek 6.21: Wpływ korekcji na zapotrzebowanie na zasoby mapowania odczytów z zestawu Z1

Wpływ korekcji na zapotrzebowanie mapowania na zasoby Wpływ korekcji na czas mapowania τ oraz zapotrzebowanie na pamięć μ, ze względu na jego niewielkie (podobnie jak asemblacji) znaczenie, został zamieszczony w podrozdziale C.1.2. Przedstawiono wyłącznie wyniki dla zestawu Z1, gdzie są one najbardziej znaczące (rys. 6.21). W wyniku korekcji algorytmem BFC nastąpiło zmniejszenie zapotrzebowania mapera na pamięć o ok. 5 GiB, co jest znaczącą różnicą, zwłaszcza w kontekście równoczesnego skrócenia czasu mapowania o godzinę (choć mniejszego niż czas korekcji, wynoszący ok. 2,5 godziny), faktu poprawy w wyniku korekcji jakości wyników asemblacji oraz nie zaobserwowanych niepokojących zjawisk w wynikach mapowania.

Detekcja wariantów

Dokonując przeglądu literatury oraz internetowych forów branżowych zaobserwowano niewielką różnorodność istniejących referencyjnych zestawów wariantów, dostępnych na potrzeby ewaluacji procesu detekcji wariantów. Niedostatek ten znacząco ogranicza zakres możliwych do wykorzystania w eksperymentach zestawów danych. Łatwo osiągalnych jest kilka zbiorów wariantów ludzkich, w szczególności najstarszy i najbardziej popularny [131] jest zbiór siedmiu zbiorów wariantów Genome in a Bottle (GIAB) [213], a także syntetyczny, diploidalny zestaw Syndip [131].

Pomimo dostępności tych danych należy zwrócić uwagę, że detekcja wariantów ludzkich (a także korekcja odpowiednich odczytów z sekwencjonowania) stanowi wyzwanie ze względu na znaczny rozmiar genomu oraz dużą liczbę powtó-

rzeń [132]. Wobec powyższego zdecydowano się na wykorzystanie również odczytów innego organizmu, których charakterystyka mogłaby skutkować łatwiejszym wykonaniem tego zadania oraz zadania korekcji. W tym celu oparto się na zestawach danych udostępnianych w ramach projektu 1001 Genomów [26]. Zaletą wykorzystania tego projektu jest nie tylko dostępność referencyjnych zbiorów wariantów, ale także duża różnorodność zestawów odczytów, obejmująca różną głębokość sekwencjonowania oraz jakość rozumianą jako szeroki zakres wartości średnich prawdopodobieństw błędów. W przypadku *A. thaliana* wykorzystano jeden zestaw odczytów o dużej głębokości sekwencjonowania, z którego wyodrębniono odczyty w celu uzyskania zestawów A1_10, A1_20, ..., A1_100 o różnej głębokości sekwencjonowania.

W przypadku ludzkich zestawów eksperymentalnych wykorzystano odczyty (zestawy H1_15, H1–2_30, H1–3_45, H1–4_60 o różnej głębokości sekwencjonowania) uzyskane z biblioteki DNA CT8595, wygenerowanej dla wzorcowego osobnika NA12878, tego samego, dla której przygotowano jeden z referencyjnych zestawów wariantów GIAB. Uwzględniono także zestaw *regionów o wysokiej pewności* (ang. *confident call regions*, plik BED), zawierający informację, istnienie których spośród wariantów w referencyjnym zestawie zostało potwierdzone ze szczególną pewnością, a także które fragmenty genomu są, ze szczególną pewnością, pozbawione wariantów. Niezależnie przeprowadzono eksperymenty dla zestawów odczytów H5_15, H5_30, H5_45, H5_55, których wyniki porównano z zestawem referencyjnym Syndip. Ta część eksperymentów miała na celu ograniczenie wpływu konkretnego zbioru wariantów oraz narzędzia oceniającego na uzyskane rezultaty. Twórcy zbioru Syndip zwrócili uwagę, że pozostałe zestawy wzorcowe mogą cechować się nadreprezentacją wariantów łatwych do uzyskania dla technik sekwencjonowania pozwalających na uzyskanie krótkich odczytów. Zestaw Syndip powinien być pozbawiony tego rodzaju wady [131]. W tabeli B.12 zawarto wersje referencyjnych zestawów ludzkich wariantów, a także numer biblioteki DNA osobników *A. thaliana* wg numeracji projektu 1001 Genomów.

Mapowanie odczytów, będące częścią potoku przetwarzania danych, wykonano przy pomocy narzędzia BWA [129]. W celu wykonania zasadniczej detekcji wariantów wykorzystano pakiet Strelka2 [111]. Zrezygnowano z powszechnie wykorzystywanego potoku algorytmów GATK [62] ze względu na jego skomplikowaną obsługę oraz bardzo długi czas obliczeń, który stanowił techniczną barierę, szczególnie w analizie ludzkiego genomu. Uzyskane pliki mapowania były sortowane, indeksowane oraz konwertowane do formatu BAM przy pomocy narzędzia SAMtools [130].

Porównanie uzyskanych zestawów wariantów wraz z referencyjnym zestawem wykonano przy pomocy narzędzia Haplotype VCF comparison tools (hap.py) [5] (większość zestawów) oraz przy pomocy narzędzia dostępnego razem z zestawem wariantów Syndip (zestawy odczytów H5_15, H5_30, H5_45, H5_55). Narzędzia

te pozwalają na uzyskanie następujących miar jakości: czułość ξ_{sens}, precyzja ξ_{prec} oraz miara F1 ξ_{F1}, zdefiniowanych jak w podrozdziale 4.4.1. Wyniki są wyznaczane niezależnie dla wariantów typu SNP oraz indel. W wynikach hap.py wykorzystano wszystkie warianty, bez względu na zachowanie filtrów (oznaczone jako ALL). Wybór dwóch metod oceny jest umotywowany ograniczeniem wpływu algorytmu wyznaczania oceny na uzyskane rezultaty.

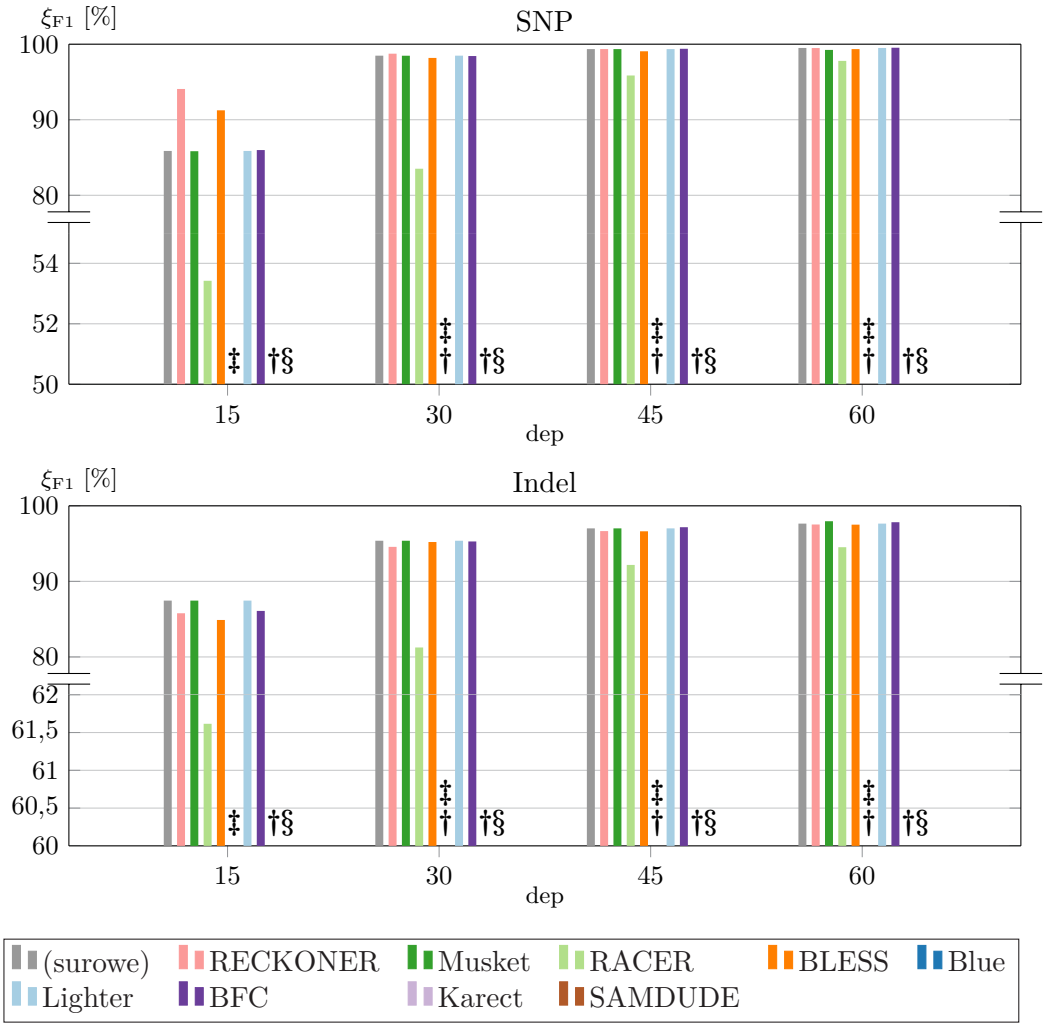
W tabeli B.21 przedstawiono wartości parametrów algorytmów, które pozwoliły na uzyskanie najlepszych wg wartości ξ_{F1} (i przedstawionych) rezultatów.

Warianty *H. sapiens* Na rys. 6.22 przedstawiono wartości ξ_{F1} detekcji dla odczytów *H. sapiens* przy różnych przybliżonych głębokościach sekwencjonowania, odpowiednio z podziałem na warianty typu SNP oraz indel, uzyskane przy pomocy narzędzia hap.py. Wyniki detekcji wariantów jednego zestawu ludzkich odczytów, ocenione według tej samej miary przy pomocy narzędzia Syndip, przedstawiono na rys. 6.23. Te ostatnie nie uwzględniają korekcji algorytmem Blue, ponieważ nie udało się go wykonać dla zestawów odczytów, których warianty były oceniane algorytmem hap.py, i w rezultacie nie uzyskano żadnych wskazówek odnośnie do doboru parametru k. Wartości wszystkich pozostałych miar zostały zaprezentowane w podrozdziale C.1.2. Wartości przemnożono przez 100%.

Korekcja odczytów ludzkiego genomu wykonana większością algorytmów nie przynosi wyraźnej zmiany jakości detekcji wariantów, gdy jest wyznaczana narzędziem hap.py. Wśród wariantów typu SNP korzystnie wyróżniają się algorytmy RECKONER oraz BLESS, które pozwalają na znaczącą poprawę detekcji przy głębokości sekwencjonowania równej 15×, w przypadku algorytmu RECKONER wynoszącą ok. 10%. Poprawa jest nieznacznie widoczna także dla algorytmu RECKONER i głębokości 30×. Sugeruje to, że zastosowanie tych algorytmów może skutecznie przynieść korzyści, gdy nie ma możliwości uzyskania wystarczającej głębokości sekwencjonowania.

W przypadku wariantów typu indel, przy głębokości sekwencjonowania równej 15× występuje spadek wartości wskaźnika ξ_{F1} w wyniku korekcji algorytmem BLESS, a w nieco mniejszym stopniu także algorytmami RECKONER i BFC. Zjawisko to jest jednak mniej istotne, ponieważ, jak zaobserwowano w pracy [61] proces detekcji wariantów typu indel dla tak niewielkiej głębokości sekwencjonowania charakteryzuje się zdecydowanie słabszymi rezultatami niż dla większych jej wartości. Dla wyższych głębokości problem zanika dla algorytmów BLESS i BFC, z kolei dla algorytmu RECKONER sukcesywnie maleje.

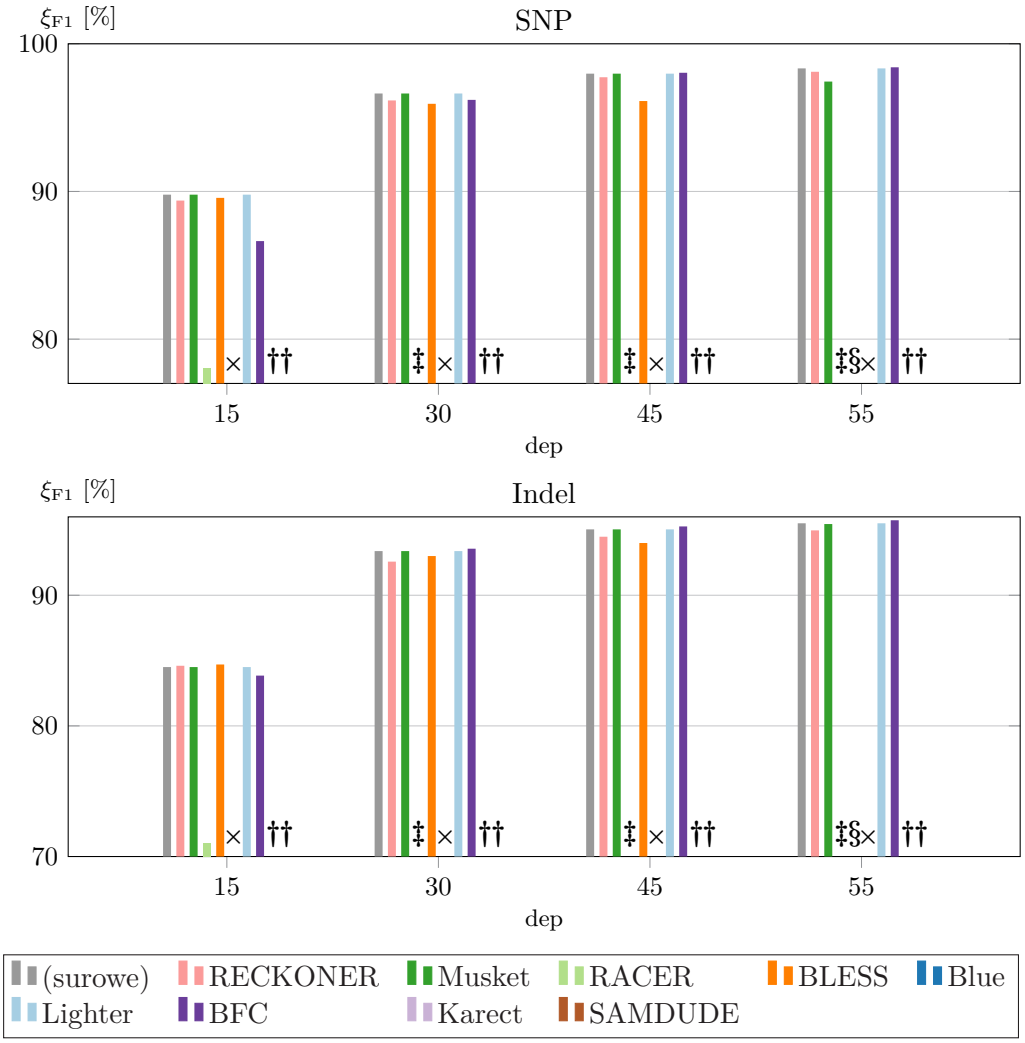
Należy mieć na uwadze, że najlepsze wyniki algorytmów Musket i Lighter uzyskano dla bardzo małych długości oligomeru, wynoszących $k = 11$ albo $k = 13$. Są to wartości nieakceptowalnie małe, nawet w przypadku genomów mniejszych i o prostszej charakterystyce powtórzeń sekwencji niż ludzki. W efekcie w wyjściowych odczytach zanotowano bardzo niewielką liczbę zmian, stąd też w oma-



Rysunek 6.22: Wpływ korekcji na wartość ξ_{F1} detekcji wariantów odczytów z zestawów H1_*, H1–2_30, H1–3_45, H1–4_60 (*H. sapiens*), różna głębokość sekwencjonowania — hap.py

wianych przypadkach wyniki tych algorytmów są bardzo zbliżone do wyników odczytów nieskorygowanych.

Wyniki uzyskane w oparciu o ocenę detekcji wariantów przy pomocy narzędzia i zestawu prawdy podstawowej wariantów Syndip okazały się być wyraźnie inne. Nie wystąpiło w nich zjawisko poprawy liczby wariantów typu SNP przy niskiej głębokości sekwencjonowania, choć jednocześnie wyniki uzyskane dla różnych głębokości sekwencjonowania wzajemnie różnią się mniej, niż w poprzednim eksperymencie. Ponadto notowany jest spadek jakości przy korekcji algorytmem BFC, który jednak zanika przy większych głębokościach. Dla algorytmu RECKONER również występuje minimalne pogorszenie wyników, które utrzymuje się bez względu na głębokość sekwencjonowania. Z kolei korekcja algorytmem BLESS



Rysunek 6.23: Wpływ korekcji na wartość ξ_{F1} detekcji wariantów odczytów z zestawów H5_* (*H. sapiens*), różna głębokość sekwencjonowania — Syndip

skutkuje spadkiem jakości przy wartościach głębokości 30× oraz 45×.

Bardzo podobny efekt dla wszystkich algorytmów wystąpił w analizie pod kątem wariantów typu indel, z tą różnicą, że dla najmniejszej głębokości algorytm RECKONER powoduje bardzo małą poprawę jakości, a dla pozostałych — większe spadki. Z kolei dla odczytów sekwencjonowania głębokości 30× oraz 45× algorytm BFC powoduje niewielką poprawę skuteczności. We wszystkich przypadkach korekcja przy pomocy algorytmu RACER spowodowała bardzo duży spadek jakości uzyskanych wyników.

Warianty *A. thaliana* Na rys. 6.24 przedstawiono wartości ξ_{sens} oraz ξ_{prec} detekcji wariantów odczytów *A. thaliana* przy pomocy narzędzia hap.py. Ze względu na

dużą różnicę między wynikami według tych miar, a zatem także niemiarodajne wartości miary F1, rezultaty przedstawiono niezależnie. Syntetyczne wyniki miary F1 zostały zaprezentowane w podrozdziale C.1.2.

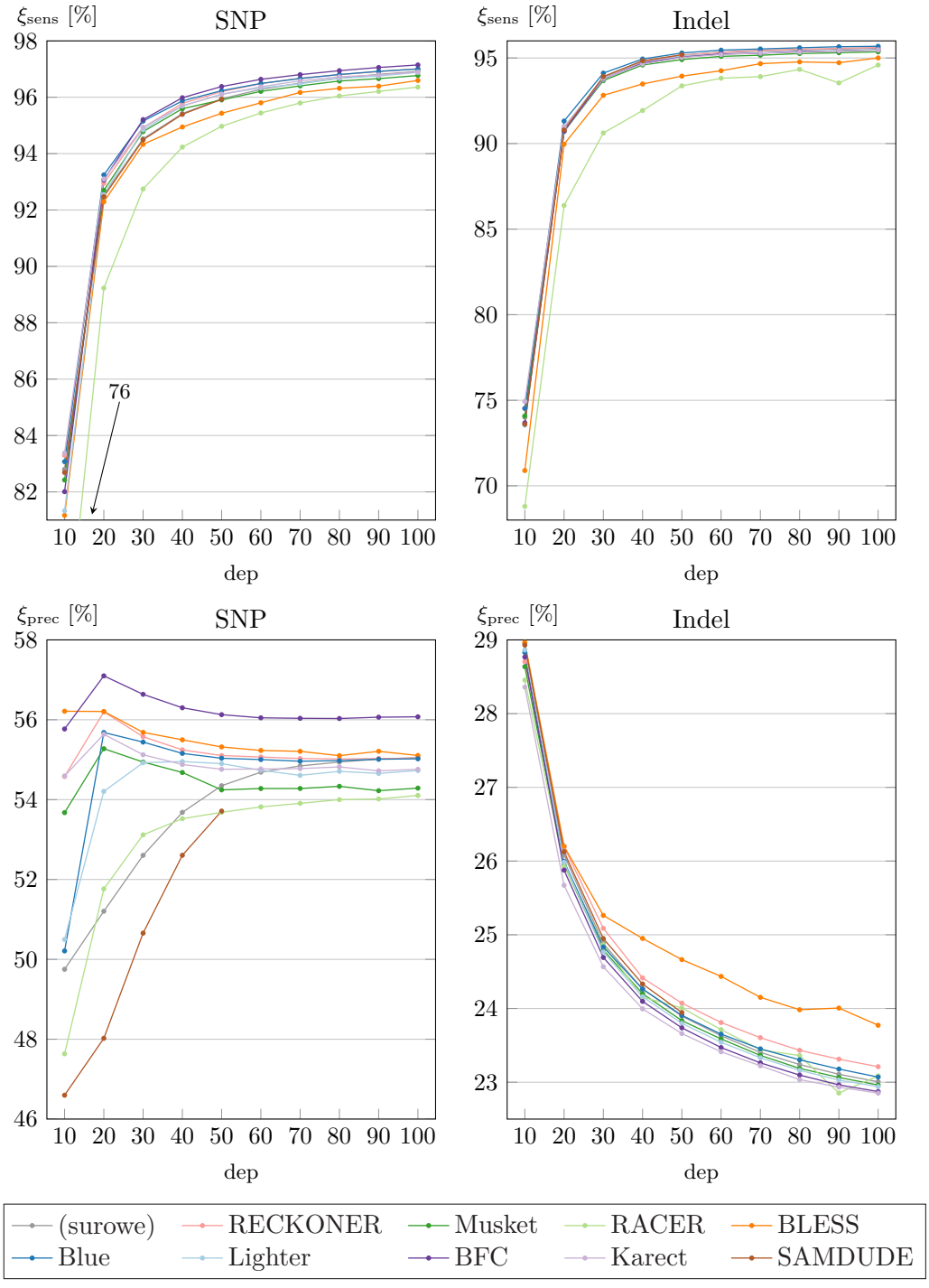
Wpływ głębokości sekwencjonowania jest zdecydowanie odmienny niż we wcześniejszych przypadkach i na pozór niedorzeczny. W przypadku czułości zgodnie z oczekiwaniami jakość wyników wzrasta wraz ze zwiększaniem głębokości sekwencjonowania. Z drugiej strony precyzja wzrasta tylko w dolnej części zakresu zmiany głębokości, a następnie powoli spada. Dla wariantów typu indel spadek następuje w pełnym zakresie. Jednocześnie uzyskane wyniki precyzji, a w konsekwencji także miary F1 są zdecydowanie niższe niż dla wariantów ludzkiego genomu.

Jako przyczynę wymienionych zjawisk można przyjąć kilka faktów. W omawianych eksperymentach nie było możliwości wykorzystania zestawu regionów o wysokiej pewności. Ponadto liczba wariantów genomu *A. thaliana* w stosunku do długości genomu jest prawie o rząd wielkości większa niż ludzkiego genomu (w oparciu o [213] oraz suplement pracy [43]), stanowiąc czynnik utrudniający przetwarzanie jego odczytów, co wbrew motywacji wyboru tego organizmu czyni to zadanie niekoniecznie prostszym.

Spadek precyzji postępujący wraz ze wzrostem głębokości sekwencjonowania jest spowodowany wzrostem liczby FP wariantów, co jest efektem wykonania oceny w oparciu o referencyjny zestaw wariantów opracowany w ramach projektu 1001 Genomów według poniższego protokołu. Warianty zostały uzyskane wykorzystując wyniki dwóch niezależnie wykonanych potoków detekcji wariantów. Z rezultatów został uzyskany iloczyn ich zbiorów, a wynikowe odczyty poddane filtracji. Opracowany w taki sposób zestaw wariantów nie jest typowym zestawem prawdy podstawowej, tylko zbiorem wariantów dosyć pewnych pod kątem prawdziwości, ale daleki od kompletności.

Wykorzystane w eksperymentach zestawy odczytów wraz ze wzrostem głębokości sekwencjonowania powodują wzrost liczby uzyskiwanych wariantów. O prawidłowości takiego zachowania świadczy sukcesywny wzrost czułości. Jednakże coraz większa liczba spośród dodatkowych wariantów nie jest obecna w zestawie referencyjnym, stąd też następuje spadek precyzji. Mimo to nie ma podstaw twierdzić, że warianty te są nieprawidłowe.

Wykorzystano kilka innych schematów oceniania wariantów, których celem miało być uzyskanie bardziej jednoznacznych wyników (dokładne wartości nie zostały przedstawione. W pierwszej kolejności podjęto próbę wykonania filtracji wariantów w sposób analogiczny do zestawu referencyjnego, eliminując te o współczynnikach jakości wariantów mniejszych od 25. Działanie nie przyniosło jednak rezultatu, ze względu na fakt, że w przeciwieństwie do wariantów referencyjnych wariantom generowanym przez narzędzie Strelka przyporządkowane są współczynniki jakości z zakresu znacznie szerszego niż standardowy wyno-



Rysunek 6.24: Wpływ korekcji na wartości ξ_{sens} oraz ξ_{prec} detekcji wariantów odczytów z zestawów A1_* (*A. thaliana*), różna głębokość sekwencjonowania — hap.py

szący ok. 0–40, sięgające wartości 3070. Jednocześnie analizując wyjściowe pliki SAM narzędzia Strelka stwierdzono, że skalowanie współczynników jakości do tego przedziału nie może być wykonane w prosty sposób.

Inne podejście do filtracji polegało na określeniu progów filtracji wariantów uzyskanych dla narzędzia Strelka w celu dopasowania poziomu jakości wyników do wariantów zestawu referencyjnego. Dla głębokości sekwencjonowania 50× oraz 100× dokonano doboru progów filtrowania tak, aby uzyskać liczbę wariantów zbliżoną do zestawu referencyjnego. Podobnie podjęto próbę takiego doboru progów, żeby uzyskać maksymalną wartość miary F1. Następnie uzyskany próg zastosowano do filtracji niezależnie dla zestawów wszystkich głębokości sekwencjonowania. W obu przypadkach uzyskane rezultaty były jednak niesatysfakcjonujące, głównie wskutek bardzo małej czułości (w wybranych przypadkach nawet mniejszej niż 10%) dla niewielkich wartości głębokości oraz dalszego występowania spadku precyzji oraz miary F1 wraz ze wzrostem głębokości sekwencjonowania.

Zaobserwowano, że zmiana średniej wartości oraz mediany wszystkich współczynników jakości zestawu zmienia się liniowo w funkcji głębokości sekwencjonowania. Fakt ten wykorzystano w celu wykonania skalowania progu filtracji w zależności od głębokości sekwencjonowania. W pierwszej kolejności wyznaczono próg filtracji dla dwóch przypadków głębokości 50× oraz 181× (pełny zestaw odczytów A1) w taki sposób, aby dopasować liczbę wariantów do rozmiaru zestawu referencyjnego. Następnie, niezależnie uzyskane dwa progi poddano skalowaniu w oparciu o współczynniki funkcji regresji liniowej, uzyskane dla zmiany średniej wartości współczynników wraz z głębokością sekwencjonowania. Wyniki dla zestawów poddanych filtracji charakteryzowały się wzrostem czułości dla niewielkich wartości głębokości, a następnie sukcesywnym spadkiem, co również jest zjawiskiem niepożądanym.

Ponadto wykonano obserwację wpływu wartości progów filtrowania dla głębokości 50× oraz 100× na wartości miary F1 wariantów. Stwierdzono, że wraz ze wzrostem progu filtracji w zakresie ok. 100–1000 następuje niewielki spadek czułości oraz wzrost precyzji. Po przekroczeniu jednak określonej wartości zależnej od głębokości sekwencjonowania następuje gwałtowny spadek tych miar, w szczególności czułości.

Pomimo wymienionych trudności uzyskane wyniki są źródłem pewnych informacji użytecznych w kontekście oceny korekcji odczytów. W większości przypadków korekcja skutkuje zwiększeniem liczby prawidłowych wariantów w stosunku do odczytów nieskorygowanych, w szczególności typu SNP, co jest obserwowalne w postaci wzrostu czułości. Wyjątek stanowią algorytmy BLESS oraz RACER, których zastosowanie powoduje spadek czułości zarówno dla wariantów typu SNP jak i indel. W pozostałych przypadkach czułość wariantów typu indel jest podobna dla odczytów surowych oraz skorygowanych przy pomocy różnych algorytmów.

Wśród wyników precyzji korzystnie wygląda wzrost w stosunku do surowych

odczytów wartości wariantów typu SNP, obserwowany dla wszystkich algorytmów korekcji z wyjątkiem RACER i SAMDUDE, co świadczy o znacznym zmniejszeniu liczby fałszywie pozytywnych wariantów. W przypadku wariantów typu indel jedynym algorytmem korekcji powodującym wyraźną zmianę w uzyskanych wynikach jest BLESS. Warianty uzyskane po korekcji odczytów tym algorytmem charakteryzują się największą precyzją, a zatem i najmniejszą liczbą fałszywie pozytywnych wariantów. Mając na uwadze wcześniejsze wyniki BLESS uzasadnione jest stwierdzenie, że może to efektem licznych przypadków uszkodzenia odczytów, powodujących powstawanie większego „szumu" informacyjnego podczas detekcji wariantów.

Podsumowując, wykorzystana metoda oceny korekcji w oparciu o detekcję wariantów odczytów genomu *A. thaliana* umożliwiła obserwację pewnych korzyści płynących z wykonania korekcji, jednak są to informacje fragmentaryczne, a sama metoda wymaga ulepszenia w wyniku dalszych, starannych prac.

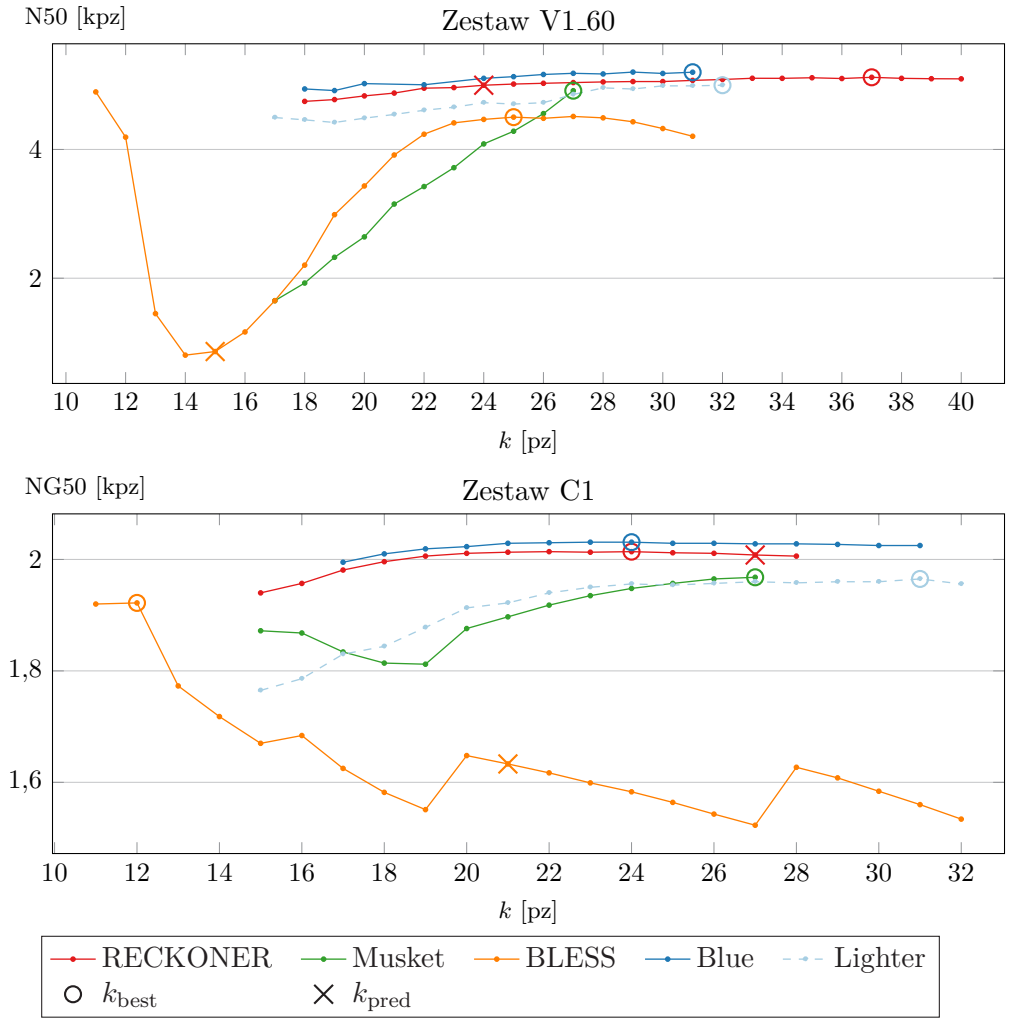
Wpływ długości oligomeru

W przedstawionych wyżej eksperymentach algorytmy parametryzowane długością oligomeru k zostały uruchomione wielokrotnie dla różnych jego wartości w celu wyboru najlepszych wyników. Takie rozwiązanie jest uciążliwe w zastosowaniu, stąd konieczne jest przyjęcie bardziej praktycznej strategii parametryzacji. Kluczowe zagadnienie stanowi ocena wpływu tego parametru, pozwalająca na stwierdzenie, jakim ryzykiem obarczone jest dokonanie wyboru.

Na rys. 6.25 oraz 6.26 przedstawiono zależność wartości miar N50 albo NG50 oceny asemblacji od wartości k algorytmów korekcji. Wyniki uzyskano dla czterech zestawów odczytów, mając na celu uzyskanie typowych lub specyficznych przebiegów zmian. Jako preferowaną miarę wyboru przyjęto NG50, jednak brak jej dla wszystkich odczytów zestawu M1 oraz wybranych przypadków zestawu V1_60 po korekcji algorytmem BLESS podyktował rezygnację z niej na rzecz N50 w analizie tych zestawów. Na wykresach wyróżniono przypadki, dla których uzyskano najlepsze wyniki asemblacji (według kryterium opartym na kilku miarach, opisanym w podrozdziale 6.2.2, stąd też przypadki te nie muszą być wartościami maksymalnymi z zakresów zobrazowanych na wykresach) oraz te, które należałoby wykorzystać postępując zgodnie z wytycznymi autorów, znanymi tylko dla algorytmów RECKONER[5] oraz BLESS. Stąd też wyniki pozwalają na uproszczoną ocenę tych wytycznych.

Spośród przetestowanych algorytmów pięć wymaga parametryzacji długością oligomeru. Nie podano wyników algorytmu Blue dla korekcji odczytów z zestawu

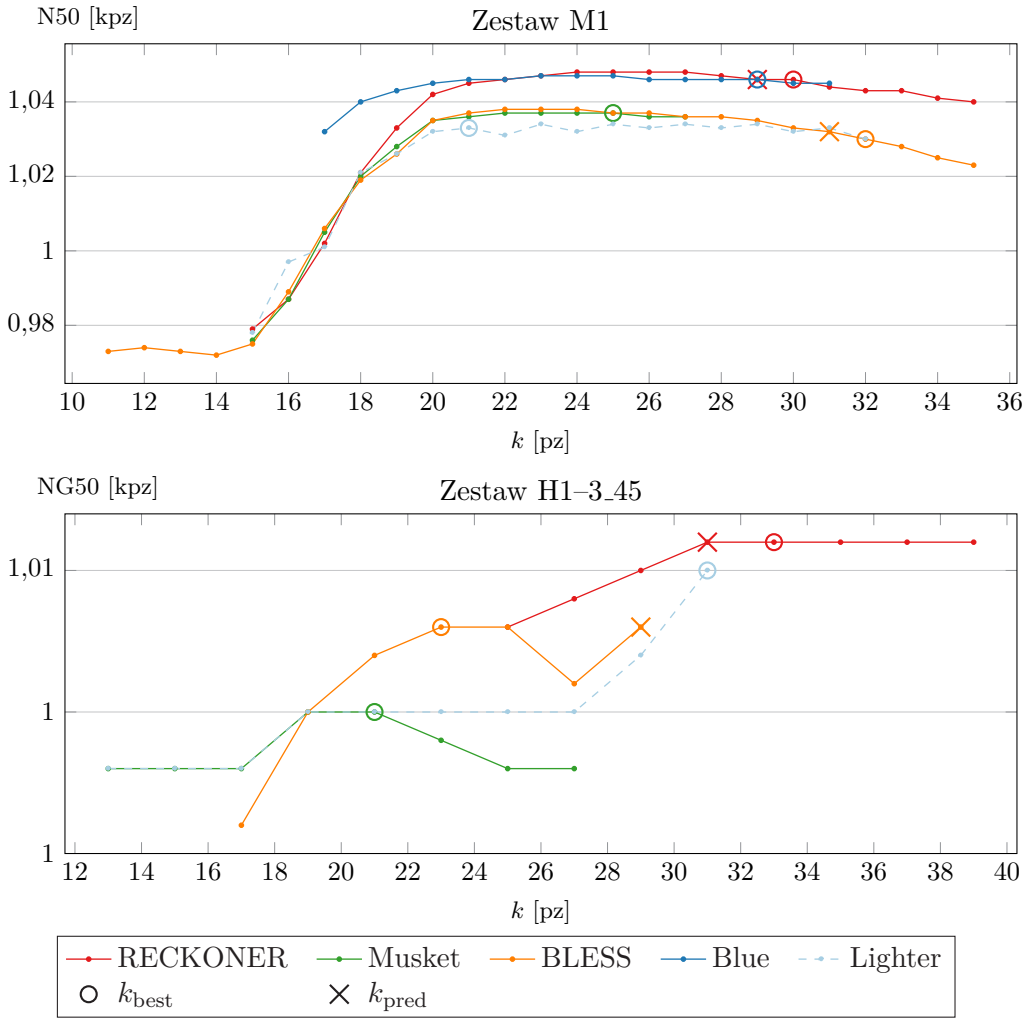
[5]Przedstawione wyniki algorytmu RECKONER zostały uprzednio wykorzystane w celu wyznaczenia reguły określenia długości oligomeru, stąd formalnie mamy do czynienia z wykorzystaniem części danych treningowych w celu przeprowadzenia testowania. Jednakże przedstawiona miara i organizmy stanowiły tylko jeden z wielu zestawów danych treningowych, dlatego uznano analizę za dopuszczalną.



Rysunek 6.25: Zależność wartości miar N50 (dla odczytów z zestawu V1_60) oraz NG50 (dla odczytów z zestawu C1) od długości oligomeru korekcji

H1–3_45, ponieważ jego wykonanie nie powiodło się. Dla algorytmów Blue oraz Lighter dopuszczalne są wartości $k \leqslant 32$, z kolei przyjęcie dla Musket $k > 27$ powoduje uzyskanie takich samych wyników jak dla $k = 27$, stąd w niektórych przypadkach możliwe jest odniesienie wrażenia, że zwiększenie wartości parametru pozwoliłoby na uzyskanie lepszych wyników. Ponadto korekcja nie powiodła się dla następujących indywidualnych przypadków: zestaw P1_60, algorytm Blue ($k = 21, 23$); zestaw C1, algorytm Blue ($k = 15, 16$); zestaw M1, algorytm Blue ($k = 15, 16$); zestaw H1–3_45, algorytm BLESS ($k = 13, 15, 31, 33$).

Wyniki pozwalają zaobserwować nie tylko różną skuteczność korekcji mierzoną wpływem na jakość asemblacji (wyższe wartości oznaczają lepszą jakość), ale także nieraz bardzo odmienny przebieg jakości wraz ze zmianą wartości k. Dla odczytów z zestawu V1_60 wpływ parametru k na pracę algorytmów Blue, REC-



Rysunek 6.26: Zależność wartości miar N50 (dla odczytów z zestawu M1) oraz NG50 (dla odczytów z zestawu H1–3_45) od długości oligomeru korekcji

KONER oraz Lighter jest bardzo niewielki, obserwowalny jest tylko niewielki spadek jakości dla małych wartości parametru. Z kolei algorytmy BLESS oraz Musket cechują się dużą wrażliwością na wartość parametru (w przypadku algorytmu BLESS wpływ jest jednak mniejszy w przedziale umiarkowanych wartości k), choć można odnieść wrażenie, że poprawny może być dobór jak największej wartości. Istotny jest fakt, że zalecany dobór parametru algorytmu BLESS jest podstawą do przyjęcia bardzo niekorzystnej wartości; eksperymentalny wyznaczenie wyników dla małych wartości parametru tego algorytmu jest spowodowane tym zaleceniem. Z kolei w przypadku algorytmu RECKONER różnica wyznaczonego oligomeru jest jeszcze większa, jednak spowodowana nią różnica jakości jest zdecydowanie mniejsza. Ponadto jest to przypadek odosobniony, gdyż pomimo niewielkiego rozmiaru genomu *C. vulgaris* najlepsza zaobserwowana długość

oligomeru jest bardzo duża (przypadek zostanie dokładniej przedstawiony w podrozdziale 6.3).

W przypadku korekcji niektórymi algorytmami odczytów z zestawu C1 dobór parametru może być bardziej skomplikowany. Duża odporność algorytmów RECKONER i Blue jest ewidentna dla $k \gtrsim 20$. W przypadku algorytmów Lighter i Musket także można przyjąć założenie o konieczności doboru dostatecznie dużych wartości k, choć dla tego drugiego występuje minimum dla $k = 19$. Sytuacja jest bardziej złożona w przypadku algorytmu BLESS. Przebieg jakości charakteryzuje się występowaniem kilku wyraźnych minimów lokalnych oraz stopniową zmianą jakości. Należy zaznaczyć, że zjawisko to jest obserwowalne także dla innych, nieprzedstawionych przypadków. Jednocześnie stosunkowo dobre wyniki tego algorytmu uzyskano dla niewielkich wartości parametru, powodujących brak zmian w odczytach oraz występowanie stosunkowo niewielkiej liczby przypadków obcięcia odczytu. W rezultacie także w tym przypadku dobór parametru k algorytmu BLESS z góry dał rezultat daleki od właściwego. W przypadku algorytmu RECKONER zaobserwowana różnica długości oligomeru oraz zmiana jakości są niewielkie.

Dla odczytów z zestawu M1 dobór zbyt małej wartości parametru wszystkich algorytmów, orientacyjnie $k \leqslant 20$, wykazał negatywny wpływ na jakość wyników, przy czym dosyć znaczny rozmiar genomu testowanych odczytów stanowi wyraźną sugestię, że dobór niewielkich wartości nie powinien nastąpić. Jednocześnie dla algorytmów RECKONER oraz BLESS jest obserwowalny powolny spadek jakości dla $k \geqslant 29$, stąd zakres rozsądnych wartości parametru jest dosyć szeroki. Zarówno dla algorytmu BLESS jak i RECKONER uzyskana najlepsza wartość k okazała się być zbliżona od wyznaczonej.

W przypadku zestawu H1–3_45 sumaryczne wahania jakości okazały się być niewielkie, prawdopodobnie na skutek znacznego stopnia skomplikowania procesu asemblacji odczytów dużego genomu, który dominuje kwestię jakości odczytów. Mimo to widać, że lepszym rozwiązaniem może być dobór dużych wartości k, zakładając, że możliwe jest wykonanie algorytmu dla takich wartości.

Podsumowując, spośród algorytmów wymagających parametryzacji długością oligomeru najmniejszą wrażliwością na nietrafiony jego dobór (w rozsądnym zakresie) charakteryzują się algorytmy RECKONER oraz Blue. Nieco bardziej ryzykownym doborem charakteryzuje się algorytmy Lighter oraz Musket, z kolei w przypadku BLESS dobór jest trudny do przeprowadzenia oraz może łatwo powodować uzyskanie nieproporcjonalnie słabych wyników. Przedstawiona w kolejnym podrozdziale strategia doboru długości oligomeru algorytmu RECKONER sprawdza się dosyć dobrze, z kolei w przypadku BLESS raczej nie spełnia swojej roli.

6.3 Dobór długości oligomeru algorytmu RECKONER

Jak wynika z podrozdziału 4.2.1, dobór długości oligomeru k nie jest zadaniem trywialnym. Parametr ten wywiera jednak wpływ na uzyskane wyniki, stąd należy poświęcić kwestii jego doboru należytą uwagę, nawet mając na uwadze fakt, że uzyskany rezultat nie będzie w pełni satysfakcjonujący.

Według wiedzy autora problem doboru długości oligomeru algorytmów korekcji nie został w literaturze dostatecznie omówiony. Jak zostało wspomniane wcześniej, nawet autorzy takich algorytmów podchodzą do tematyki ograniczając się do proponowania częściowo intuicyjnych, a częściowo opartych na obserwacjach empirycznych strategii, nie podając jednocześnie precyzyjnych przesłanek, które uzasadniałyby dane podejście.

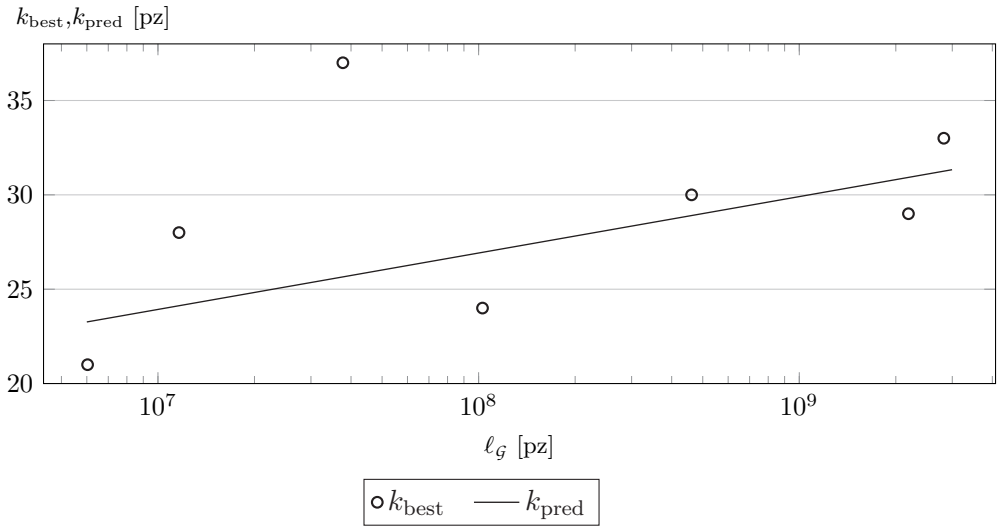
Jedną z przyczyn z pewnością jest poziom skomplikowania problemu. Uzyskany wynik powinien zależeć od długości genomu, wartości średnich współczynników jakości odczytów, charakterystyki występowania błędów typu indel, długości odczytów, charakterystyki sekwenatora, własności sekwencji genomu i innych. Z tego powodu w niniejszej pracy problem zostanie pozostawiony częściowo otwarty, a jako strategia doboru parametru zostanie zaproponowana prosta metoda, której opracowanie będzie będzie brało pod uwagę minimalizację liczby kryteriów optymalizacji wyniku. Podejście tego rodzaju można traktować jako uzasadnione w kontekście wykazanej wcześniej niewielkiej wrażliwości algorytmu RECKONER na zmianę wartości tego parametru, a także faktu, że RECKONER jest algorytmem heurystycznym, stąd też uzyskane wyniki korekcji są obarczone pewną dozą niepewności i ewentualne wahania jakości wraz ze zmianą wartości k obejmują także naturalną, niemożliwą do eliminacji zmienność.

Uwagi odnośnie do wpływu wartości k na przebieg korekcji zostały przedstawione w podrozdziale 4.2.1. Dodatkowo warto zwrócić uwagę na następujące kwestie dotyczące teoretycznego wpływu na czułość oraz precyzję korekcji (wielkości traktowane ogólniej niż w podrozdziale 4.4.1, bo obejmujące także odczyty rzeczywiste). Dobór małej wartości może powodować zmniejszenie precyzji, ze względu na wyższe prawdopodobieństwo powtórzenia danej sekwencji k-meru w sekwencji genomu, a w efekcie także w odczytach uzyskanych z różnych rejonów genomu. Analogicznie, dobór większych wartości powinien spowodować zwiększenie precyzji. Dodatkowo należy zauważyć, że dla małych wartości ulega zwiększeniu liczebność k-merów, skutkując ewentualnym zwiększeniem progu obcięcia. W efekcie k-mery o wyższej niż przy dużych wartościach k liczebności mogą być uznane za błędne, co może zwiększać czułość (w sytuacji odwrotnej rozumowanie jest analogiczne). Tym samym widać, że na dobór wartości k powinien mieć wpływ rozmiar genomu.

W celu opracowania sposobu doboru k w funkcji rozmiaru genomu należy zauważyć, że wraz z jego wzrostem (zakładając, że genom jest sekwencją losową; w literaturze zaznaczono, że dla znaczącej części genomu taka cecha występu-

je [45]) wzrasta prawdopodobieństwo, że dla zadanego k losowy k-mer będzie podsłowem tego genomu na więcej niż jednej pozycji. Tym samym, wraz ze wzrostem rozmiaru genomu należy zwiększać k, przy czym ze względu na to, że liczba możliwych sekwencji k-meru wynosi 4^k (liczba k-elementowych wariacji z powtórzeniami ze zbioru ($|\Sigma| = 4$)-elementowego), w celu zachowania liczby tych sekwencji w takim samym stosunku do długości genomu, wartość k powinna rosnąć w funkcji logarytmicznej[6]. Innymi słowy, przyjmując pewną stałą $x' \in \mathbb{R}_+, x' \ll 1$ wartość k powinna spełniać warunek $4^k = x'\ell_{\mathcal{G}}$, zatem wartość $k \propto f(\ell_{\mathcal{G}})$, gdzie f jest pewną funkcją logarytmiczną. Dodatkowo, z przyczyn praktycznych, zamiast dokładnej długości genomu $\ell_{\mathcal{G}}$ będzie przyjmowane jego przybliżenie $\widehat{\ell_{\mathcal{G}}}$. Zmiana wartości logarytmu wraz ze zmianą jej parametru jest niewielka, stąd przyjęto, że takie uproszczenie nie powinno stanowić problemu. Logarytmiczną zależność między długością genomu a długością oligomeru sugerowano też w pracy [110] oraz suplemencie pracy [134].

Analityczne wyznaczenie optymalnej zależności w oparciu o analizę działania algorytmu uznano za zadanie zbyt skomplikowane. W efekcie zdecydowano się na wyznaczenie jej empirycznie, wykorzystując regresję nieliniową i przyjmując, że celem jest parametryzacja funkcji logarytmicznej. Jako zmienna objaśniana zostanie przyjęta zmienna k, a jako zmienna objaśniająca zmienna $\widehat{\ell_{\mathcal{G}}}$. Dodatkowo, z przyczyn implementacyjnych, jako podstawę logarytmu przyjęto wartość 2. Funkcja taka przyjmuje postać: $k_{\text{pred}} = x \log_2 \widehat{\ell_{\mathcal{G}}} + y$ gdzie x i y są parametrami wyznaczonymi przy pomocy regresji.



Rysunek 6.27: Zależność najlepszej długości oligomeru od rozmiaru genomu — RECKONER

Na wykresie przedstawionym na rys. 6.27 zaznaczono wartości najlepszej dłu-

[6]Pomijając konieczność kwantyzacji wynikającą z faktu, że $k \in \mathbb{N}_+$.

gości oligomeru k_{best} od długości genomów, wyznaczone dla różnych zestawów odczytów w eksperymentach opisanych w podrozdziale 6.2.2. Zależność logarytmiczna długości oligomeru od długości genomu jest trudno dostrzegalna, w szczególności w wyniku odstającej wartości $k_{\text{best}} = 37$ dla odczytów z zestawu V1_60. Odrzucając jednak ten przypadek, wyznaczenie regresji logarytmicznej powoduje uzyskanie postaci odpowiedniej funkcji. Dodatkowo przyjęto, że minimalna uzyskana długość oligomeru powinna wynosić 20. Stąd równanie (przedstawione już w podrozdziale 5.4.2), o przybliżonych parametrach, przyjmuje postać:

$$k_{\text{pred}} = \max(20; 0{,}9 \log_2 \widehat{\ell_{\mathcal{G}}} + 3), \tag{6.1}$$

gdzie k_{pred} jest wyznaczoną długością oligomeru dla algorytmu RECKONER.

Zależność wynikająca z tego równania również została przedstawiona na wspomnianym wykresie. W celu poprawy czytelności wyznaczono go dla przeciwdziedziny $k_{\text{pred}} \in \mathbb{R}_+$. W implementacji wartość k_{pred} jest poddawana kwantyzacji poprzez zaokrąglenie do najbliższej liczby naturalnej. Przybliżona długość genomu $\widehat{\ell_{\mathcal{G}}}$ jest wyznaczana zgodnie z równaniami (5.24) oraz (5.25).

Poza powyższym przeprowadzono próbę obserwacyjnej analizy potencjalnego wpływu na uzyskaną najlepszą wartość k_{best} łatwo mierzalnych parametrów zestawów odczytów, jak głębokości sekwencjonowania, wartości średnich współczynników błędów oraz długości odczytów w zestawie. W kontekście zestawów wykorzystanych w eksperymentach nie wyciągnięto jednak jednoznacznych wniosków dotyczących ich znaczenia, stąd też w równaniu nie wzięto tych parametrów pod uwagę.

Należy zauważyć, że analizę przeprowadzono wyłącznie wykorzystując wyniki asemblacji, wykonując korekcję w trybie z weryfikacją w oparciu o k''-mery. W przypadku detekcji wariantów oraz korekcji bez tego trybu uzyskane wartości z powyższego równania wynoszą $k_{\text{pred}} = 27$ dla *A. thaliana* (w eksperymentach $k_{\text{best}} \in \{21, 23, 25\}$) oraz $k_{\text{pred}} = 31$ dla *H. sapiens* (w eksperymentach $k_{\text{best}} = 59$). Wynika z tego, że metoda powoduje uzyskanie zbyt dużej wartość parametru w pierwszym przypadku (co nie powinno znacząco wpłynąć na uzyskane wyniki). Z kolei w drugim przypadku wartość wyznaczana jest zdecydowanie zbyt mała i nawet przemnożenie wartości k_{pred} przez stałą $\theta_{\text{LONG\_RATIO}} = 1{,}5$ (różnicującą długość k-merów oraz k''-merów), nie powoduje uzyskania zbliżonej wartości. W rezultacie zostanie przedstawione zalecenie dla użytkownika, aby w wyróżniającym się zadaniu detekcji ludzkich wariantów przyjmować duże wartości parametru k.

W opinii autora warto przeprowadzić dalsze, dokładne badania mające na celu skuteczniejszy dobór długości oligomeru dla wszystkich wspomnianych algorytmów korekcji, być może obejmujących dokładne zamodelowanie zjawisk związanych z charakterystyką genomów, odczytów oraz samych algorytmów.

6.4 Analiza według kryteriów wydajnościowych

Analiza algorytmów ze względu na kryteria wydajnościowe została przeprowadzona zgodnie z informacjami z podrozdziału 4.4.2. Jako podstawowe miary zostały przyjęte czas obliczeń, zapotrzebowanie na pamięć algorytmu oraz skalowalność algorytmu wraz ze wzrostem liczby wątków.

6.4.1 Czas obliczeń i zapotrzebowanie na pamięć korekcji

Podczas uruchomienia wszystkich implementacji algorytmów korekcji przeprowadzono pomiar czasu ich pracy τ oraz szczytowego zapotrzebowania na pamięć μ. Czas ten obejmuje pracę całości implementacji, wliczając w to ewentualne narzędzia pomocnicze, które są wymagane przez algorytm (np. zliczanie k-merów). W niniejszym podrozdziale przedstawiono uzyskane wyniki dla korekcji odczytów rzeczywistych. Analogiczne wyniki dla odczytów symulowanych zamieszczono w podrozdziałach C.2.1 oraz C.2.1.

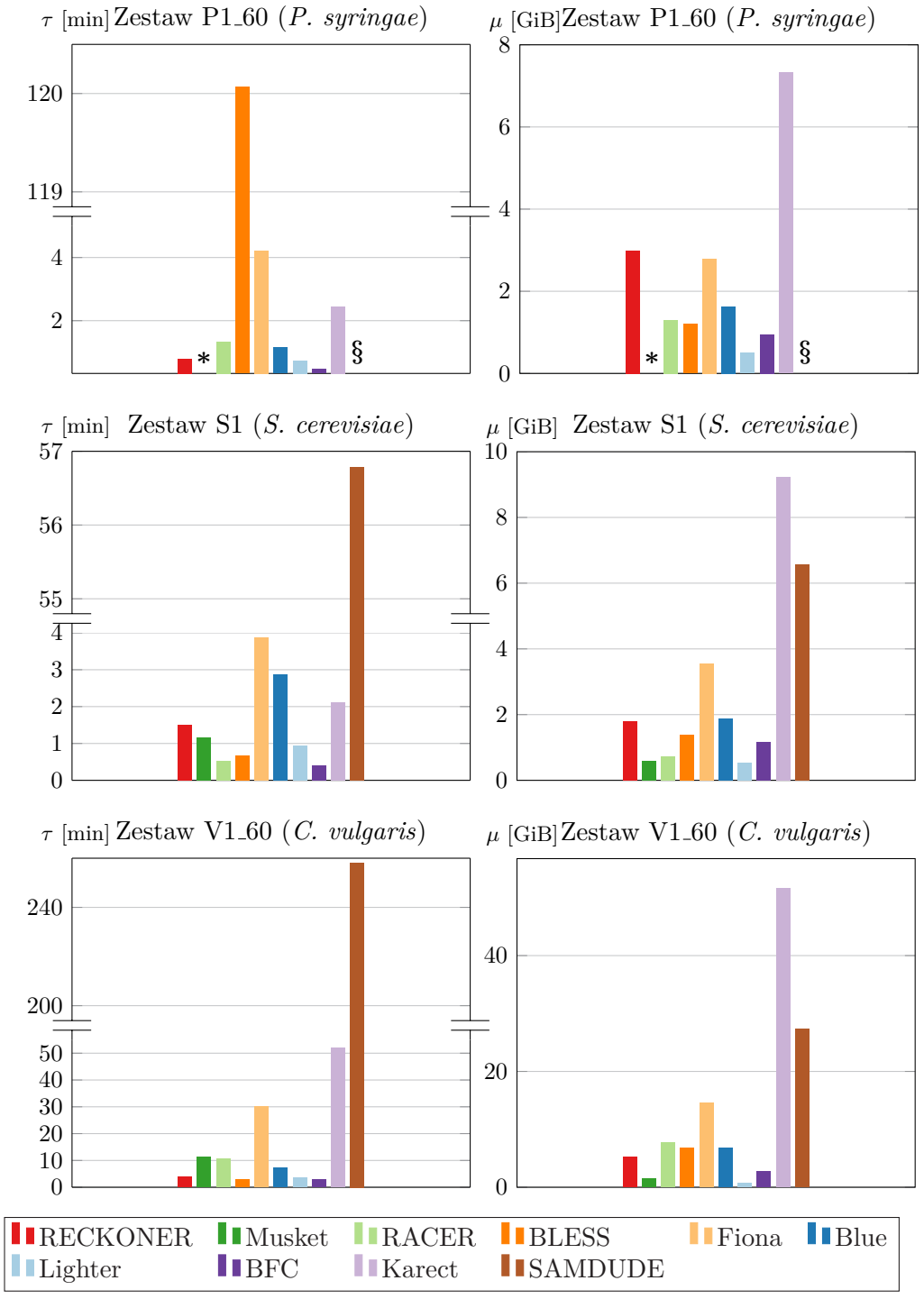
Odczyty rzeczywiste

Na rys. 6.28–6.30 przedstawiono czas obliczeń oraz zapotrzebowanie na pamięć operacyjną algorytmów korekcji, wykorzystanych do korekcji odczytów rzeczywistych. Uzyskane wyniki dotyczą przypadków, w których algorytmy były parametryzowane wartościami powodującymi uzyskanie najlepszych wyników dla asemblacji.

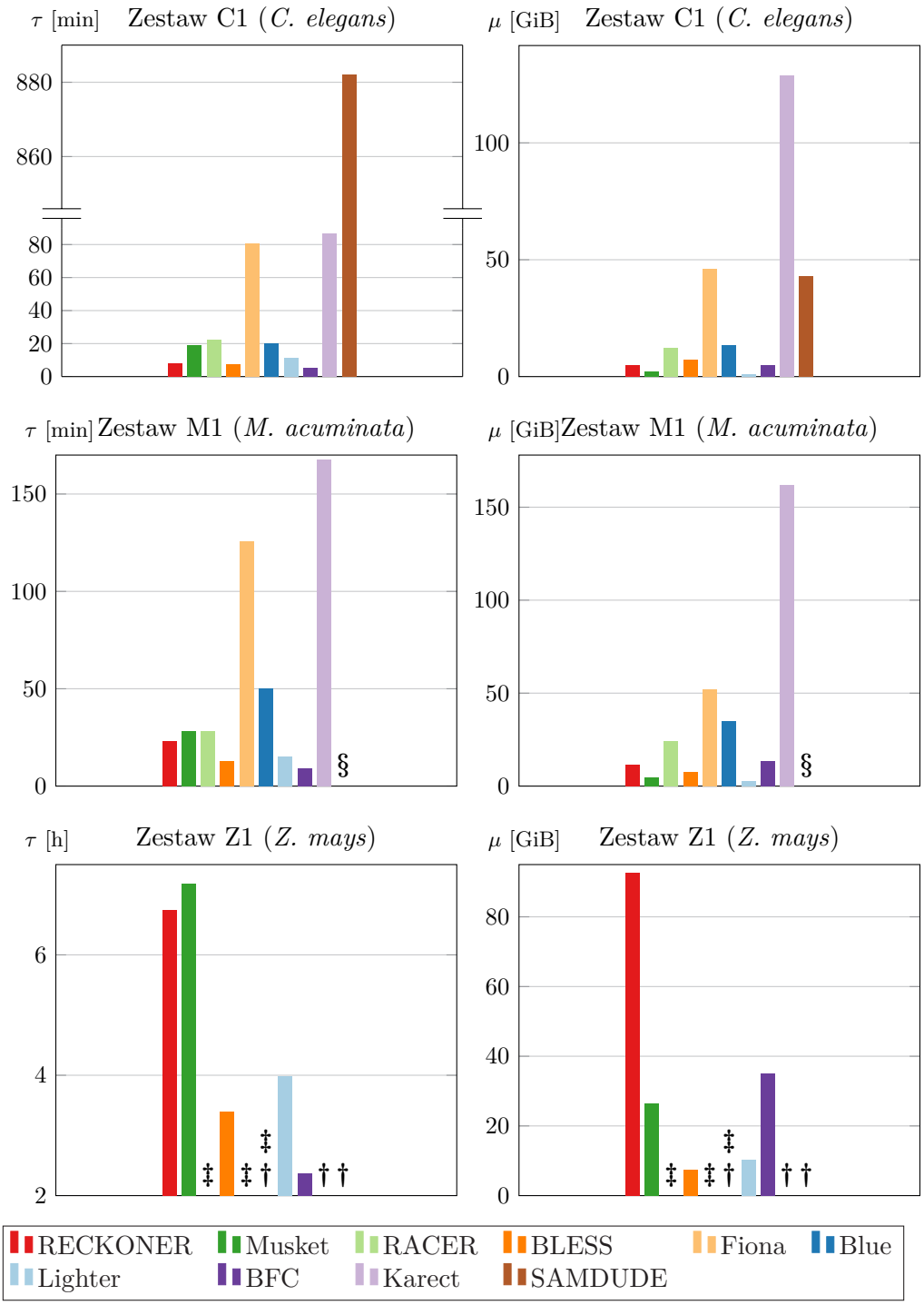
W większości zestawów względne zapotrzebowanie algorytmów na zasoby jest podobne dla różnych zestawów danych. W przypadku zestawu P1_60 większość algorytmów pozwoliła na uzyskanie wyników w ciągu kilku minut, stąd wyciągnięcie wiążących wniosków jest trudne. Wyjątek stanowi cechujący się znacznie dłuższym czasem pracy algorytm BLESS, co jest prawdopodobnie efektem charakterystyki odczytów MiSeq, w szczególności długości większej od odczytów z pozostałych zestawów.

W przypadku zestawu S1 czas wykonania prawie wszystkich algorytmów nie przekraczał 4 min, przy czym obserwowalny jest wyraźnie dominujący czas obliczeń algorytmów Fiona oraz Blue. Jednocześnie algorytmy RACER oraz BFC umożliwiają uzyskanie wyników w ciągu ok. 0,5 min. Czas działania algorytmu SAMDUDE, wynoszący prawie 1 h jest efektem wykonania sekwencyjnego.

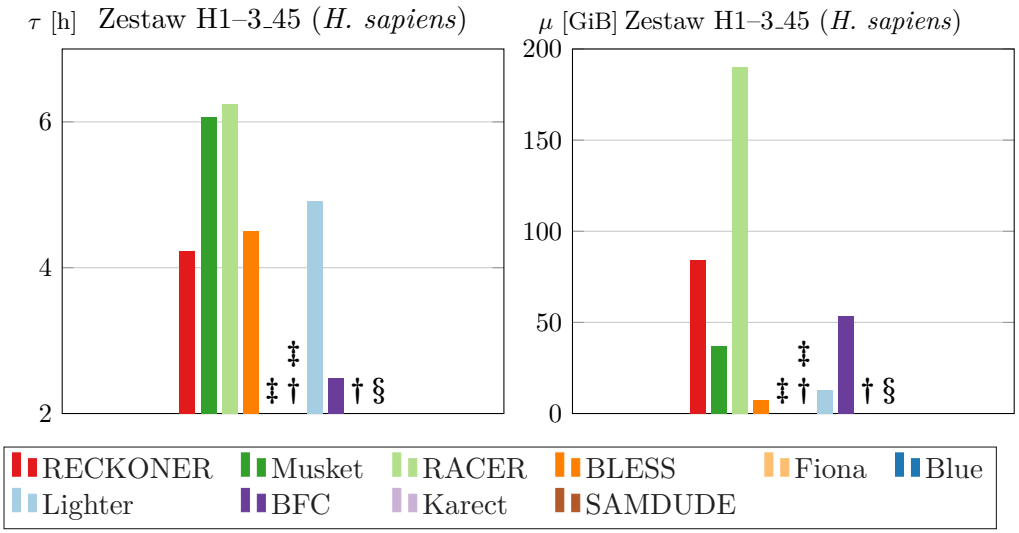
Korekcja odczytów z zestawu V1_60 przy pomocy najszybszych algorytmów RECKONER, BLESS, Lighter oraz BFC była możliwa w czasie krótszym niż 5 min. Z kolei dosyć znaczny, przekraczający 30 min czas odnotowano dla algorytmów Karect oraz Fiona. Bardzo mocno odstająca jest czasochłonność algorytmu SAMDUDE, którego wykonanie wymagało przeszło 4 h. Proporcje podobne do wymienionych są obserwowalne w efekcie korekcji odczytów z zestawów C1 i M1, przy czym w tym ostatnim przypadku z pominięciem algorytmu SAMDUDE



Rysunek 6.28: Zapotrzebowanie na zasoby korekcji odczytów z zestawów P1_60, S1, V1_60



Rysunek 6.29: Zapotrzebowanie na zasoby korekcji odczytów z zestawów C1, M1, Z1



Rysunek 6.30: Zapotrzebowanie na zasoby korekcji odczytów z zestawu H1–3_45

(którego wykonanie nie powiodło się) oraz ze zwróceniem uwagi na stosunkowo większy wzrost czasu pracy algorytmu Blue dla zestawu M1.

Czas pracy niektórych algorytmów okazał się być przeszkodą w wykonaniu korekcji odczytów z zestawu Z1. Algorytmy Karect oraz SAMDUDE, a dla niektórych długości oligomeru także algorytmu Blue nie pozwoliły na wykonanie zadania w czasie 24 h. Zdecydowanie najszybszy algorytm BFC umożliwił wykonanie korekcji w czasie nieco przekraczającym 2 h, a dalsze algorytmy, BLESS i Lighter w czasie ok. 3,5 h oraz 4 h. Algorytmy RECKONER oraz Musket wymagały 6,5 h i 7 h, co, abstrahując od jakości uzyskanych wyników, ze względu na znaczny rozmiar genomu można jednak nadal uznać za czas akceptowalny.

Korekcja odczytów z zestawu H1–3_45 również przekroczyła limit czasu w efekcie zastosowania algorytmów Karect oraz w wybranych przypadkach Blue. Brak powodzenia algorytmu SAMDUDE był skutkiem zgłoszenia błędu, jednak należy się spodziewać, że przy jego braku również korekcja nie byłaby możliwa ze względu na zbytnią czasochłonność. W przypadku zestawu H1–3_45 również najszybszy okazał się algorytm BFC, wykonany w czasie 2,5 h, choć na kolejnych miejscach znalazły się kolejno algorytmy RECKONER, BLESS oraz Lighter, których wykonanie wymagało ok. 4 h, 4,5 h i 5 h. Najbardziej czasochłonna okazała się korekcja przy pomocy algorytmów Musket oraz RACER.

Podsumowując, zdecydowanie najszybszym algorytmem jest BFC. W dalszej kolejności należy zaklasyfikować BLESS (z wyjątkiem zestawu P1_60), Lighter oraz RECKONER (z wyjątkiem zestawu Z1). Algorytmy Musket, RACER oraz Blue pozwalają na korekcję w umiarkowanym czasie, choć w ostatnim przypadku dla niektórych wartości parametru k korekcja największych zestawów odczytów w zadanym czasie nie zakończyła się. Jednocześnie algorytmy Karect, Fiona oraz

SAMDUDE charakteryzują się znacznym czasem wykonania, który może stanowić praktyczne utrudnienie w wykonaniu korekcji. W przypadku SAMDUDE należy zwrócić również uwagę na bardzo negatywne znaczenie braku przetwarzania równoległego.

Zapotrzebowanie poszczególnych algorytmów na pamięć także różni się znacząco. Dla zestawu P1_60 zapotrzebowanie nie przekracza 8 GiB dla Karect oraz ok. 3 GiB dla RECKONER oraz Fiona. Pozostałe algorytmy zmieściły się w zakresie do 2 GiB.

W przypadku zestawu S1 wyraźnie zaznaczył się względny spadek zajętości pamięci algorytmu RECKONER. Sporym zapotrzebowaniem na pamięć charakteryzuje się algorytm SAMDUDE. Wraz z dominującym algorytmem Karect oraz Fiona są jedynymi wymagającymi więcej niż 2 GiB pamięci.

Dla odczytów z zestawu V1_60 szczególnie korzystnie wyróżniają się algorytmy Lighter oraz Musket, które pozwoliły na wykonanie korekcji przy użyciu tylko odpowiednio ok. 1,4 GiB oraz 0,6 GiB. Szczególnie kontrastuje to z z algorytmami SAMDUDE oraz Karect, wymagającymi 27 GiB i 52 GiB, co dla odczytów genomu o tak małym rozmiarze należy uznać za duże wartości.

Jeszcze wyraźniejsza różnica jest dostrzegalna dla zestawu C1. Najlepsze algorytmy, Lighter i Musket, umożliwiają korekcję przy użyciu odpowiednio 1 GiB i 2 GiB pamięci, podczas gdy algorytmy Fiona oraz SAMDUDE wymagają do korekcji prawie 50 GiB pamięci. Zapotrzebowanie na pamięć algorytmu Karect osiąga ok. 130 GiB. Podobne proporcje są widoczne dla zestawu M1 (z wyjątkiem niepowodzenia korekcji przy pomocy algorytmu SAMDUDE).

Korekcja odczytów z zestawu Z1 w niektórych przypadkach nie powiodła się ze względu na brak dostatecznej ilości pamięci (wg raportu systemu operacyjnego zainstalowane było ok. 251 GiB pamięci), co miało miejsce podczas wykonania algorytmów RACER, Fiona oraz w niektórych przypadkach Blue. Najniższym zapotrzebowaniem na pamięć cechowały się (także oparte na filtrze Blooma) algorytm BLESS oraz Lighter, które pozwoliły na korekcję przy wykorzystaniu nie więcej niż 10 GiB pamięci. Wyraźnie okazało się zapotrzebowanie na pamięć algorytmów Musket (ok. 26 GiB) oraz BFC (ok. 35 GiB), co, ze względu na duży rozmiar zestawu wejściowego, należy jednak uznać za niewygórowane wymaganie. Zapotrzebowanie algorytmu RECKONER osiągnęło znaczną wartość 93 GiB, co może limitować zakres zastosowania algorytmu.

Odczyty z zestawu H1–3_45 były możliwe do skorygowania dodatkowo za pomocą algorytmu RACER, choć wymagając aż 190 GiB pamięci. Najmniejsze zapotrzebowanie zanotowano dla wszystkich trzech algorytmów opartych na filtrze Blooma, choć wahało się ono w zakresie 7 GiB–37 GiB, w zależności od algorytmu. Algorytmy BFC oraz RECKONER stawiały większe wymagania — ok. 53 GiB i 84 GiB.

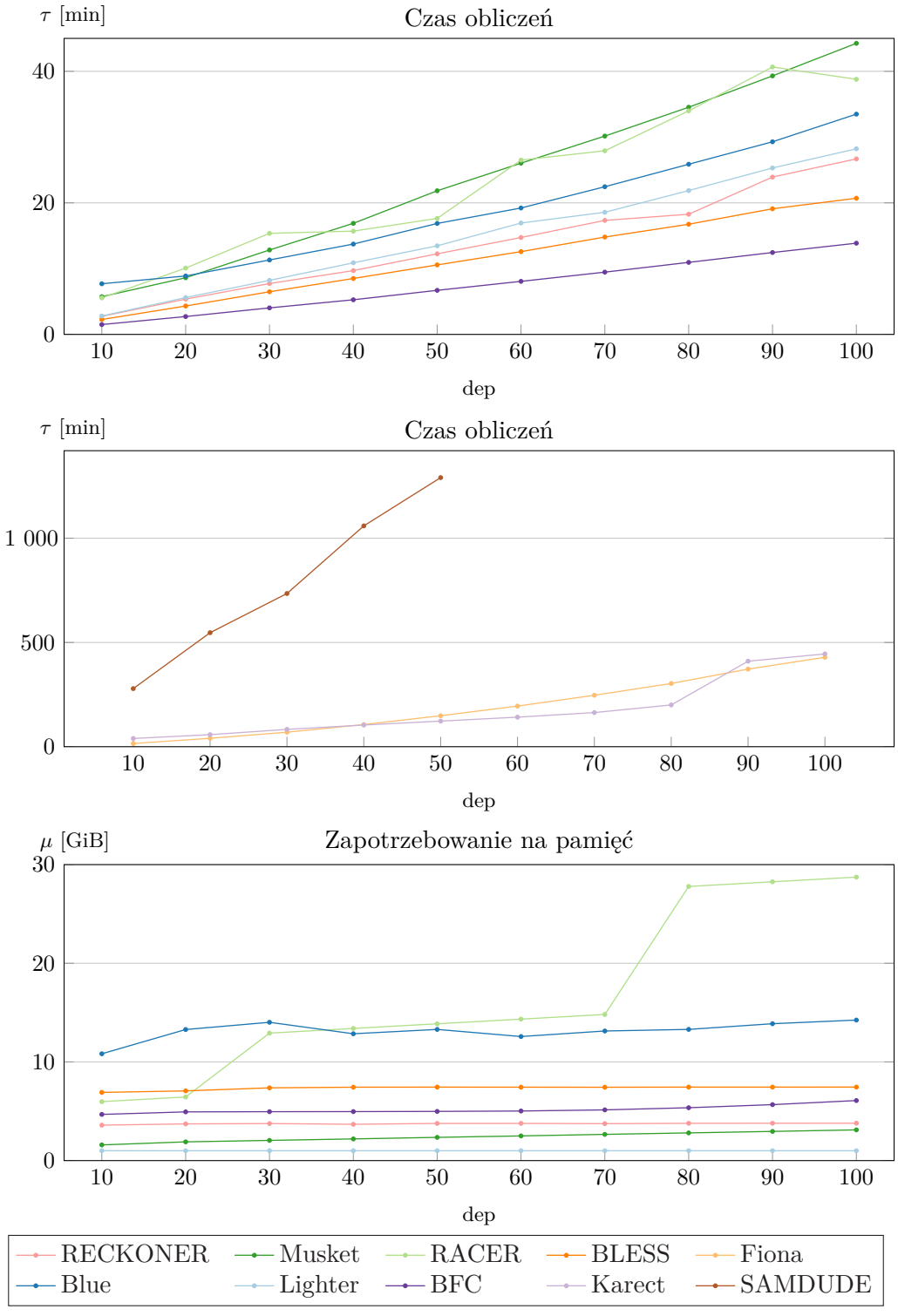
Wybór stosowanego algorytmu jest często wyraźnie ograniczony przez zapo-

trzebowanie na pamięć. Podczas gdy najbardziej oszczędne, oparte na filtrze Blooma algorytmy (Lighter, w mniejszym stopniu także Musket oraz BLESS), a także cechujący się umiarkowanym zapotrzebowaniem na pamięć BFC pozwalają w większości przypadków na korekcję przy użyciu niewielkich, łatwo dostępnych ilości pamięci, to w wyniku eksperymentów wykazano, że w niektórych przypadkach korekcja nie była możliwa nawet na typowym serwerze obliczeniowym, jak choćby w efekcie wykorzystania algorytmów RACER, Fiona i Blue. Ponadto w niektórych przypadkach zapotrzebowanie okazało się nieproporcjonalnie duże w stosunku do rozmiaru problemu, zwłaszcza w kontekście możliwości najlepszych algorytmów, co można powiedzieć o algorytmach Fiona i SAMDUDE, a także — dla największych organizmów — RECKONER. Należy jednak zauważyć, że przedstawiono wyniki dla pracy tego ostatniego algorytmu w trybie weryfikacji przy pomocy k''-merów. Rezygnacja z niego, odbywająca się kosztem jakości asemblacji, spowodowałaby przeszło dwukrotną redukcję zapotrzebowania. Dla algorytmu Karect, we wszystkich przypadkach powodzenia wykonania algorytmu, zanotowano najwyższe zapotrzebowanie na pamięć spośród wszystkich testowanych algorytmów. Algorytm jest jednak wyposażony w metodę ograniczenia zapotrzebowania na pamięć, co zostanie omówione w kolejnym podrozdziale.

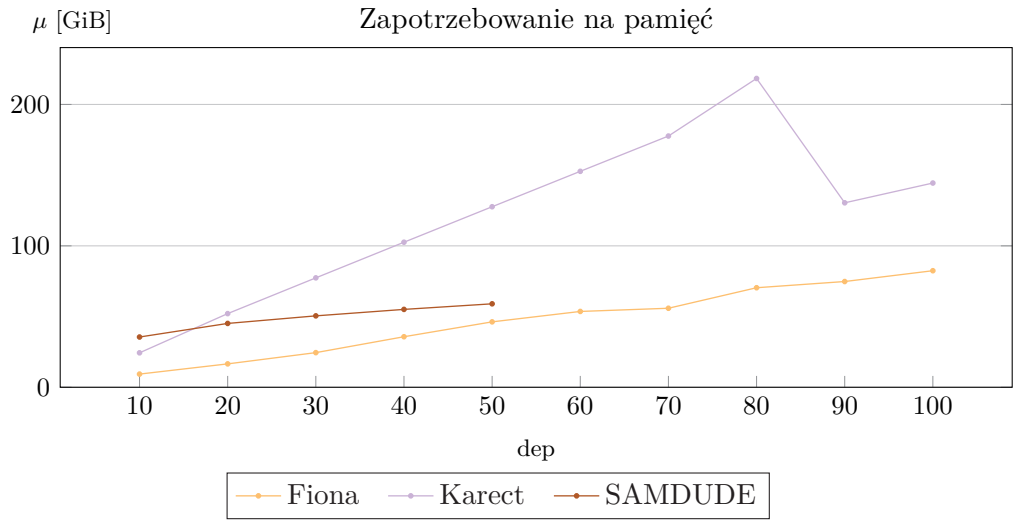
Odczyty rzeczywiste na potrzeby detekcji wariantów — wpływ głębokości sekwencjonowania

Na rys. 6.31 oraz 6.32 przedstawiono czas obliczeń τ oraz zapotrzebowanie na pamięć operacyjną μ wykonania algorytmów, wykorzystanych do korekcji odczytów rzeczywistych zestawów A1_10, A1_20, ..., A1_100 genomu *A. thaliana*. Obejmują one zmianę zapotrzebowania na zasoby wraz ze wzrostem głębokości sekwencjonowania. Wyniki pozostałych zestawów odczytów korygowanych w celu wykonania detekcji wariantów zostały zamieszczone w podrozdziale C.2.1.

Wzrost czasu korekcji większości algorytmów wraz z rozmiarem danych jest w przybliżeniu liniowy. Do wyjątków należą algorytmy Blue i Fiona, których przebiegi mogą wskazywać na występowanie mniej korzystnej zależności. Na tym etapie nie można wykluczyć, że cecha ta jest przyczyną obserwowalnego w przedstawionych wcześniej wynikach znacznego wzrostu czasu wykonania algorytmu Fiona dla kolejnych zestawów odczytów (choć przyczyna ta nie jest oczywista — dla Blue zjawisko nie jest dostrzegalne). W przypadku algorytmu RACER dają się zaobserwować pewne fluktuacje, być może spowodowane przekraczaniem wartości pewnych wewnętrznych progów zmiennych stanu algorytmu. Dla algorytmu Karect występuje wyraźny wzrost czasu obliczeń po przekroczeniu głębokości równej 80×, co najprawdopodobniej jest skutkiem parametryzacji algorytmu wartością dopuszczalnego zapotrzebowania algorytmu na pamięć, na co wskazuje przebieg wykresu jej zmian. Po przekroczeniu tego progu następuje wyraźny spadek wykorzystania pamięci, stąd można wysnuć wniosek, że w algorytmie nastąpiła próba



Rysunek 6.31: Zapotrzebowanie na zasoby korekcji odczytów z zestawów A1_*, różna głębokość sekwencjonowania



Rysunek 6.32: Zapotrzebowanie na zasoby korekcji odczytów z zestawów A1_*, różna głębokość sekwencjonowania, cd.

alokacji zbyt dużej ilości pamięci. W rezultacie nastąpiło przełączenie w tryb niższego zapotrzebowania na pamięć kosztem wzrostu czasu obliczeń.

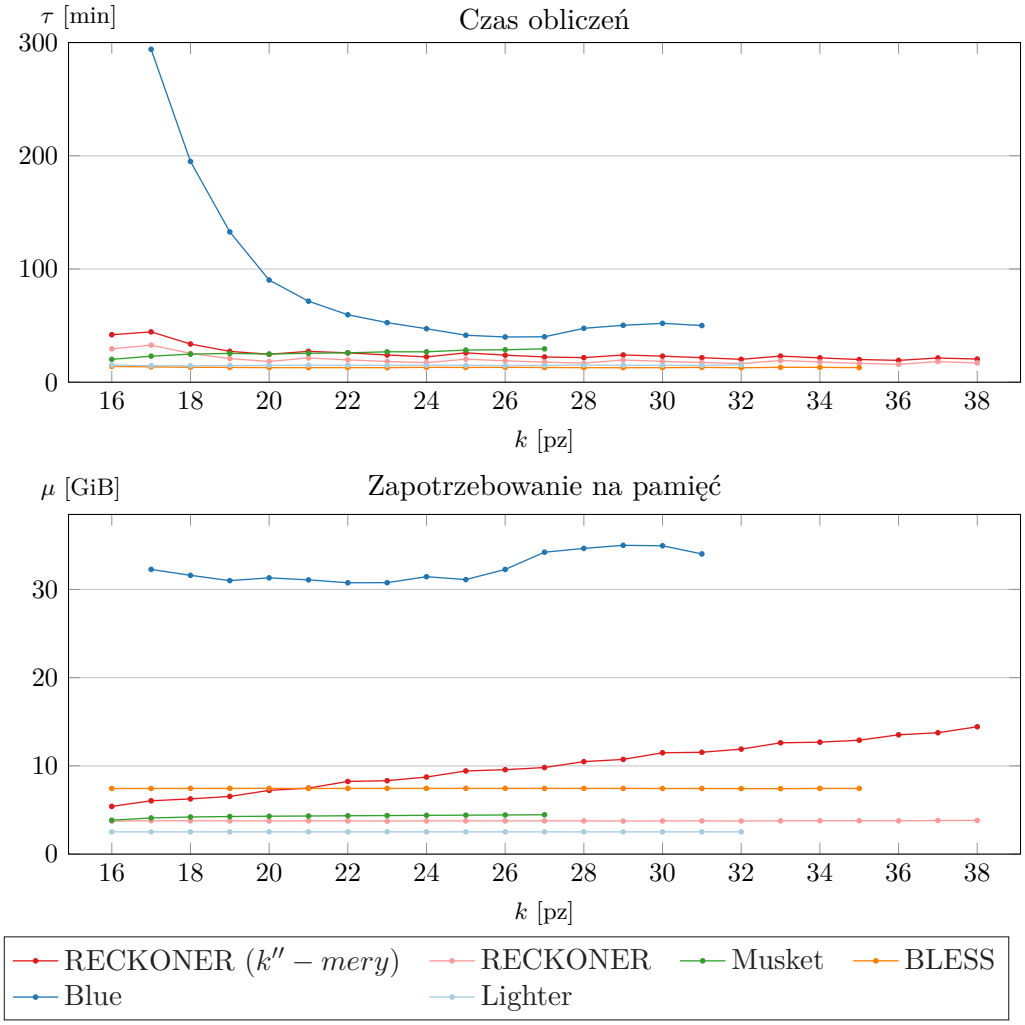
Zapotrzebowanie na pamięć większości algorytmów jest w przybliżeniu stałe bez względu na głębokość sekwencjonowania. Wyraźny, skokowy wzrost jest obserwowalny dla algorytmu RACER, osiągając prawie 30 GiB dla głębokości 100×, podczas gdy rozmiar wejściowych odczytów tego zestawu wynosi ok. 27 GiB (wraz z nagłówkami odczytów i ze współczynnikami jakości zakodowanymi w formie znaków ASCII). Niewielki wzrost jest obserwowalny dla algorytmu SAMDUDE, osiągając przeszło 50 GiB już dla głębokości 50×. Znacznie wyraźniejszy wzrost powodujący przekroczenie progu 200 GiB występuje dla algorytmu Karect, jednak możliwość określenia limitu wykorzystania pamięci oraz automatyczne znajdowanie kompromisu między czasem obliczeń a wykorzystaniem pamięci nie dyskwalifikuje tego algorytmu pomimo tak dużych wymagań (pod warunkiem, że czas obliczeń nie okaże się być zbyt duży). Wzrost zapotrzebowania na pamięć jest także obserwowalny dla algorytmów Fiona oraz SAMDUDE.

Podsumowując, własności większości algorytmów wraz ze zmianą głębokości sekwencjonowania, a zatem także ilości danych są korzystne. Wyjątek stanowią algorytmy RACER, Fiona, SAMDUDE oraz częściowo Karect, których wykorzystanie dla dużych zestawów danych o wysokim pokryciu może okazać się trudne lub niemożliwe.

6.4.2 Wpływ długości oligomeru

Na rys. 6.33 przedstawiono przedstawiono zależność czasu obliczeń oraz zapotrzebowania na pamięć korekcji w zależności od wartości k. Wyniki uzyskano dla

odczytów z zestawu M1, jako reprezentującego genom o przeciętnym rozmiarze. Algorytm RECKONER został poddany testom w dwóch trybach: podstawowym oraz wykorzystującym weryfikację w oparciu o k''-mery.



Rysunek 6.33: Zależność zapotrzebowania na zasoby korekcji od długości oligomeru dla odczytów z zestawu M1

Czas korekcji algorytmów BLESS i Lighter nie ulega znaczącemu wpływowi na skutek zmiany wartości parametru k. W przypadku pozostałych algorytmów obserwowalne są zmiany: algorytm Musket pozwala na nieznacznie szybszą korekcję dla niewielkich wartości parametru, a RECKONER — w podobnych okolicznościach na ok. dwukrotnie wolniejszą, choć zjawiska te dotyczą wartości z przedziału zbyt małego, aby rozważać je z punktu widzenia jakości wyników. Bardzo dużą wrażliwością wykazuje się algorytm Blue, którego czas obliczeń dla $k = 17$ był ok. sześciokrotnie większy niż dla $k = 26$. Wprawdzie, jak wykazano we wcześniejszej analizie, algorytm ten również nie powinien być parametryzowany małymi

wartościami k, to skala wpływu każe zwrócić uwagę, że w hipotetycznych, szczególnych przypadkach, dla których dobór niewielkiej wartości parametru byłby uzasadniony, zjawisko to może być znaczące.

Zapotrzebowanie algorytmów Musket, BLESS, Lighter (i RECKONER pracującego w zasadniczym trybie) nie wykazuje znaczącej zależności od wartości k. Jest to przypuszczalnie pozytywny efekt kodowania spektrum w formie filtrów Blooma, które mają na celu przechowanie binarnej informacji decyzyjnej, bez względu na jej charakter. Z kolei algorytm Blue cechuje się ok. dwudziestoprocentowym wzrostem zapotrzebowania przy doborze wartości parametru z przedziału wartości zbliżonych do najlepszych, a algorytm RECKONER, pracujący w trybie z weryfikacją w oparciu o k''-mery wykazuje w przybliżeniu liniową zależność tej wielkości, co jest cechą niekorzystną, ze względu na zaobserwowane uzyskiwanie korzystnych wyników dla dużych wartości parametru. Tezę o takiej własności tego trybu potwierdza również przedstawiona wcześniej analiza czasu korekcji różnych zestawów odczytów. Z drugiej strony maksymalne osiągnięte zapotrzebowanie było przeszło dwukrotnie mniejsze niż najmniejsze dla algorytmu Blue.

W przypadku algorytmów opartych na filtrach Blooma oraz algorytmu RECKONER (pracującego w zasadniczym trybie) wpływ parametru k na zapotrzebowanie na zasoby obliczeń nie jest istotny. Nieco słabiej należy ocenić algorytm Blue, choć wpływ parametru na zapotrzebowanie na pamięć nie powinien być w większości przypadków istotny. Eksperymenty wykazały, że najsłabszą charakterystyką cechuje się wykonanie algorytmu RECKONER w trybie z weryfikacją w oparciu o k''-mery, co przy korekcji zestawów odczytów dużych genomów i braku otoczenia sprzętowego o dostatecznej ilości pamięci może ograniczać jego zakres zastosowania.

6.4.3 Skalowalność

Jak zostało wspomniane w podrozdziale 3.1.3 oraz zobrazowano w przedstawionych wyżej obserwacjach czasochłonności wykonania sekwencyjnego algorytmu SAMDUDE, algorytmy korekcji powinny być dostosowane do równoległego wykonania. Oceną skuteczności przetwarzania równoległego jest skalowalność algorytmu. Lepsze rezultaty tej wielkości pozwalają na skuteczniejsze wykorzystanie dostępnych jednostek obliczeniowych oraz wskazują na potencjał ewentualnego skrócenia czasu obliczeń poprzez migrację do innego otoczenia sprzętowego. Ze względu na to, że wykorzystany w eksperymentach serwer obliczeniowy był wyposażony w procesory obsługujące wielowątkowość współbieżną (w formie technologii Hyper-threading), liczba wątków implementacji została ograniczona do wartości $\rho = 24$, równej liczbie rdzeni fizycznych.

Formalnie pod pojęciem *skalowalności* jest rozumiane zachowanie stałości ilo-

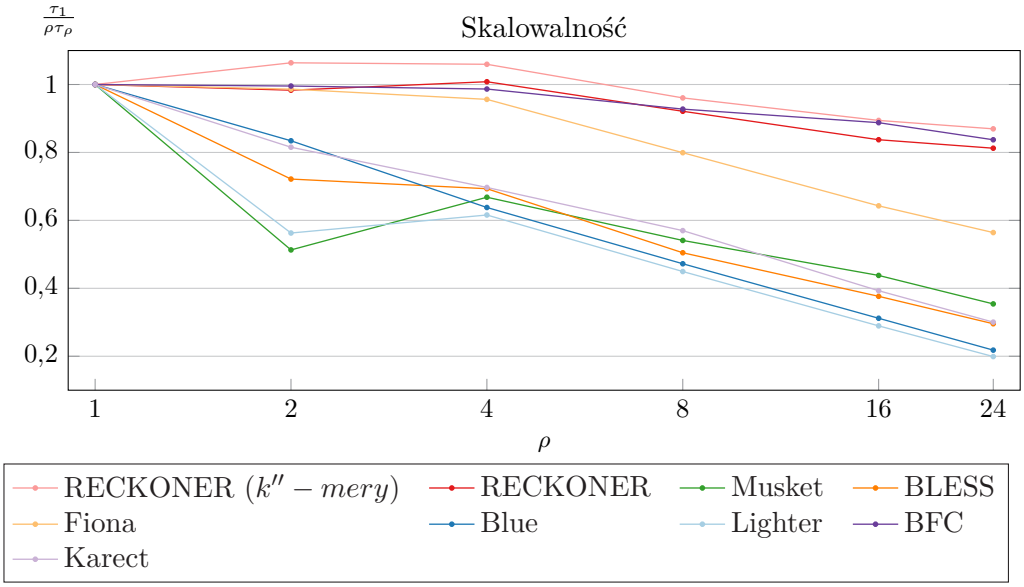
razu[7] $\frac{T_{\text{speedup}}(\rho,n)}{\rho} = \frac{T(1,n)}{\rho T(\rho,n)}$ wraz ze wzrostem rozmiaru problemu n oraz liczby procesorów ρ [53], jednak ze względu na trudność z właściwym doborem rozmiaru problemu w celu udowodnienia spełnienia tego warunku, oparciu się na danych eksperymentalnych, a nie analizie teoretycznej, a także mając na uwadze prostotę, zdecydowano na zaprezentowanie własności algorytmów dla jednego rozmiaru problemu. Mimo to, w przypadku algorytmów idealnie skalowalnych, uzyskany przebieg zmian powinien być stały. Przedstawione wyniki dotyczą przypadków, w których algorytmy były parametryzowane wartościami powodującymi uzyskanie najlepszych wyników dla asemblacji. Wyniki uzyskano dla odczytów z zestawu M1.

W celu przeprowadzenia analizy skalowalności eksperymentalnie dokonano pomiaru czasu pracy algorytmów, wykonując je w konfiguracji z różną liczbą wątków. Liczbę wątków oznaczono symbolem ρ. Jako miarę przyjęto empiryczną formę wspomnianego wyżej ilorazu, tzn. wartość $\frac{\tau_1}{\rho\tau_\rho}$, gdzie τ_ρ jest czasem działania algorytmu uruchomionego w konfiguracji pracy przy pomocy ρ wątków, w szczególności τ_1 dla $\rho = 1$. Na rys. 6.34 przedstawiono zależność wartości tej miary od liczby wątków ρ.

Algorytm RECKONER został uwzględniony dwukrotnie: w trybie pracy z weryfikacją przy wykorzystaniu k''-merów oraz w trybie zasadniczym. W eksperymentach pominięto algorytmy RACER i SAMDUDE, odpowiednio ze względu na brak udokumentowanej możliwości zadania liczby wątków oraz brak przetwarzania równoległego. Wyniki wyznaczono dla liczby wątków od 1 do 24. Należy podkreślić, że przedstawione w innych częściach pracy wyniki pomiaru czasu obejmują wykonanie przy pomocy 48 wątków, wykorzystując technologię wielowątkowości współbieżnej.

W przedstawionych wynikach należy wyjaśnić dwa zjawiska. Ze względu na ich wystąpienie podjęto próbę powtórzenia części obliczeń i pomiaru czasu, co jednak nie skutkowało wyraźną zmianą wyników. Fakt, że uzyskane wartości ilorazu w niektórych przypadkach przekraczają 1 wynika z losowych zakłóceń czasu wykonania obliczeń, jakie mogą pojawiać się w komputerach i systemach operacyjnych ogólnego przeznaczenia. W efekcie dla dobrze skalujących się algorytmów nawet niewielkie wahania (np. przypadkowe zwiększenie czasu wykonania dla $\rho = 1$) może spowodować, że iloraz uzyska wartości większe od 1. Ponadto dla algorytmów Musket oraz Lighter widoczny jest wyraźny spadek wskutek wykonania algorytmu przy pomocy $\rho = 2$ wątków. Przyczyna tego zjawiska nie jest znana, lecz prawdopodobnie jest nią albo zjawisko opisane wyżej, albo pewne wewnętrzne cechy tych algorytmów. Jednakże ze względu ujawnienie takiego efektu dla małej liczby rdzeni można przyjąć, że w praktycznym wykorzystaniu algorytmów jego wpływ nie będzie miał dużego znaczenia.

[7] W równaniu 3.6 zamiast wartości $T(1,n)$ występuje $T^*(1,n)$. Zmiana wynika z faktu, że jako przypadek odniesienia nie jest teraz brany najlepszy algorytm sekwencyjny, ale sekwencyjna wersja algorytmu równoległego.



Rysunek 6.34: Skalowalność algorytmów korekcji na przykładzie odczytów z zestawu M1

Analizowane algorytmy charakteryzują się bardzo różną skalowalnością. Zdecydowanie najlepsze wyniki uzyskano dla algorytmów RECKONER oraz BFC. Pracując przy pomocy $\rho = 24$ wątków wartość ilorazu przekracza dla nich 0,8, co świadczy o bardzo efektywnym wykorzystaniu procesorów. Różnice w wynikach obu trybów pracy algorytmu RECKONER nie są duże i mogą wynikać z większego zapotrzebowania na pamięć w trybie z wykorzystaniem dwóch baz KMC i w rezultacie ze słabszego wykorzystania pamięci podręcznej procesora. Uzyskane wyniki stanowią jedną z przyczyn zaobserwowanej wcześniej dużej szybkości algorytmów RECKONER oraz BFC, ponieważ dzięki dobrej skalowalności możliwe było skuteczne wykorzystanie dużej liczby jednostek obliczeniowych komputera.

Dalszymi w kolejności algorytmami są kolejno Fiona, BLESS, Musket oraz Karect. Z kolei algorytmy Blue oraz Lighter w pracy przy pomocy $\rho = 24$ wątków osiągnęły wartość ilorazu ok. 0,2, co oznacza, że 80% czasu pracy jednostek obliczeniowych procesora była poświęcona na oczekiwanie, synchronizację lub inne zadania związane z organizacją obliczeń równoległych. Jednocześnie ogólny bardzo krótki czas pracy algorytmu Lighter częściowo niweluje mankamenty słabej skalowalności.

6.5 Analiza według kryteriów technicznych

W trakcie przygotowania eksperymentów zwrócono uwagę na kwestie techniczne, omówione w podrozdziale 4.4.3. Własności te wywierają wpływ na łatwość uruchomienia danej implementacji, przy czym „łatwość” oznacza nie tylko wygodę,

ale też brak poważnych trudności, które w niektórych sytuacjach potencjalnie mogłoby uniemożliwić uruchomienie implementacji, wykorzystanie wyników lub spowodować niewłaściwe użycie algorytmu.

W trakcie przygotowania eksperymentów dużego wysiłku wymagała kompilacja oraz uruchomienie implementacji algorytmu BLESS, co było efektem oparcia jej na interfejsie MPI. Celem zastosowania tej biblioteki przez autorów algorytmu było przyspieszenie przetwarzania poprzez bardziej masowe zrównoleglenie. Jednakże biorąc pod uwagę, że wykonanie algorytmu tylko w jednym przypadku osiągnęło ok. 7,5 h[8], można podać w wątpliwość, czy zysk uzyskany z przetwarzania w modelu sieciowym przyniósłby korzyści większe, niż koszt przygotowań, obejmujący nie tylko wspomniane komplikacje instalacyjne, ale również wysiłek związany z konfiguracją węzłów. Mimo to, jeśli przebieg prac wymagałby wielokrotnego uruchomienia algorytmu np. z różną parametryzacją, można by rozważyć niezależne uruchomienie implementacji na różnych węzłach obliczeniowych.

W przypadku implementacji algorytmu Fiona oraz BFC nie zapewniono zachowania oryginalnych współczynników jakości odczytów. Można przypuszczać, że takie zachowanie jest spowodowane brakiem prostej strategii określania wartości tych współczynników dla zmodyfikowanych symboli. Jednakże, jak wykazały eksperymenty, całkowity brak informacji jest rozwiązaniem gorszym (czasem wręcz wykluczającym) niż przyjęcie umownej strategii.

We wszystkich przetestowanych implementacjach zapewniono zachowanie wejściowego formatu plików odczytów, przy czym w implementacjach BFC oraz Musket zmianie ulegają nagłówki odczytów w pliku FASTQ. Usuwana jest z nich część informacji, a w przypadku BFC nagłówek jest uzupełniany informacjami o przeprowadzonej korekcji. Implementacje nie posiadają funkcji pozwalających na zachowanie oryginalnych nagłówków. Implementacje algorytmów RACER, Fiona, BFC nie obsługują plików odczytów sparowanych. Tym samym w przypadku konieczności korekcji odczytów tego rodzaju, niezbędne jest np. połączenie plików odczytów, poddanie ich korekcji oraz ponowne rozdzielenie.

Implementacje algorytmów RACER, Fiona, Blue, BFC, Karect nie obsługują plików skompresowanych. W efekcie wymaga to dostępności odpowiednio większej przestrzeni dyskowej (przykładowo: para plików zestawu H1–4_60 w formie skompresowanej posiada rozmiar ok. 135 GiB, podczas gdy w formie nieskompresowanej ok. 416 GiB; w rezultacie na przeprowadzenie samej korekcji — wraz z późniejszą niezależną od korekcji kompresją wyników — potrzeba niemal 1 TiB przestrzeni dyskowej. Takie wielkości mogą sprawić trudność w praktyce, ponieważ ulegają one dalszemu rozszerzeniu na skutek faktu, że korekcja jest tylko jednym z etapów przetwarzania danych, podczas gdy pozostałe etapy także mogą wymagać znacznej przestrzeni dyskowej. Ponadto konieczność dekompresji oraz

[8]Zestaw H1–4_60, $k = 17$. Wprawdzie dla zestawu P1_60 przy $k = 20$ algorytm nie wykonał się w ciągu 24 h, jednak ze względu na fakt, że występują duże wahania w czasie korekcji odczytów z tego zestawu wraz ze zmianą k, należy przyjąć występowanie błędu w implementacji.

kompresji stanowi utrudnienie uruchomienia algorytmu oraz przynosi dodatkowy narzut czasowy. Implementacja algorytmu BLESS pozwala tylko na wygenerowanie wynikowych plików w formie nieskompresowanej, które na końcu działania algorytmu zostają poddane kompresji, tym samym na dysku w pewnym momencie muszą się znaleźć obie postacie pliku wynikowego.

W przypadku algorytmu SAMDUDE pewną trudność stanowi konieczność wcześniejszego zmapowania odczytów. Fakt ten wynika wprawdzie z zasady działania algorytmu, jednak jest dodatkowym wymaganiem praktycznym. Co więcej, wynikowe odczyty również są reprezentowane w formie mapowań. W niektórych przypadkach jest to zjawisko korzystne, ponieważ dopasowania bezpośrednio po korekcji mogą zostać wykorzystane (np. w detekcji wariantów), jednak w wielu przypadkach, np. w celu przeprowadzenia asemblacji, konieczne jest wyodrębnienie odczytów do postaci pliku FASTQ.

Algorytmy SAMDUDE oraz w mniejszym stopniu Blue charakteryzują się słabą niezawodnością, objawiającą się częstymi niepowodzeniami korekcji. Stanowi to praktyczną uciążliwość, ponieważ trudno jest z góry przewidzieć, kiedy algorytm pozwoli na wykonanie zadania. Ponadto eksperymenty wykazały bardzo duże zapotrzebowanie na przestrzeń dyskową algorytmu Karect, wynoszące kilkakrotność rozmiaru danych wejściowych.

Znaczącym aspektem praktycznym i koncepcyjnym jest parametryzacja algorytmów. Najczęściej spotykanym parametrem algorytmów jest długość k-meru. Dobór tego parametru, nawet w przypadku zaproponowania przez autorów odpowiedniej strategii, sprawia pewne problemy. Nie w pełni jest także z góry wiadome, w jakim stopniu parametr ten może wpływać na jakość uzyskanych wyników, a zatem jak duży wysiłek powinien być włożony w dobór oraz jakie ryzyko niesie ewentualna pomyłka bądź nieprzewidywalność tej wartości. Tym samym w algorytmach BLESS, Musket, Blue oraz Lighter należałoby rozważyć utworzenie automatycznego mechanizmu doboru tego parametru. Być może w niektórych przypadkach wystarczającą strategią byłoby przyjęcie stałej wartości, jak ma to miejsce w algorytmie BFC, choć też w obu przypadkach weryfikacji wymagałaby skuteczność takich metod.

Pozostałe obligatoryjne parametry algorytmów są zwykle dosyć proste do określenia (w przypadku algorytmu Musket liczba k-merów jest dosyć trudna do oszacowania z góry, ale po zastosowaniu odpowiedniego zewnętrznego narzędzia jest możliwa do określenia dokładnie, choć stanowi to dodatkową trudność). Do skomplikowanych w parametryzacji należy z kolei algorytm PREMIER, który, jak zostało wspomniane, m.in. ze względu na mnogość parametrów nie został poddany ocenie. Wpływ jego parametrów na działanie mógłby być przedmiotem osobnej, rozbudowanej analizy.

6.6 Podsumowanie

Przedstawione wyżej eksperymenty przyniosły wiele obserwacji i wniosków dotyczących szeroko pojętej użyteczności różnych algorytmów korekcji. Wprawdzie zakres badań może być jeszcze znacząco poszerzony, w szczególności o uwzględnienie innych algorytmów oraz dokładniejszą analizę niejednoznacznych przypadków, jednak skala przedsięwzięcia wymagałoby zaangażowania dużych środków.

Eksperymentalnie wykazano, że część algorytmów daje możliwość poprawy jakości zadania asemblacji o różnej skali, zależnej od zestawu odczytów. Dokładne wyniki nie są wprawdzie w pełni jednoznaczne, ponieważ zaobserwowano przypadki niespójności między grupami miar oceny asemblacji (np. miarami z grup N oraz miary ξ_{misasm}), zatem decydując się na wykorzystanie korekcji należy zwrócić uwagę na cel zastosowania wyników. Mimo to w wielu przypadkach zaobserwowano poprawę jakości, także dla odczytów z sekwenatorów MiSeq oraz w niektórych przypadkach NovaSeq. Bezpośredni wpływ na skuteczność mapowania odczytów jest niejasny w ocenie, ponieważ wyniki wskazują raczej na skuteczność korekcji niż poprawę jakości mapowania. Z kolei wpływ na kroki przetwarzania odczytów niebezpośrednio następujące po korekcji jest niewielki. Wyniki wskazują, że poprawa jakości detekcji ludzkich genomów jest możliwa w określonych przypadkach (głównie przy niewielkiej głębokości sekwencjonowania). Wpływ korekcji na detekcję wariantów *A. thaliana*, przy znacznych ograniczeniach metody korekcji można uznać za obiecujący, choć nie pozwala na wyciągnięcie wiążących wniosków. Hipoteza o potencjale korekcji w istotnej redukcji zapotrzebowania na zasoby kosztownych obliczeniowo algorytmów asemblacji i mapowania nie znajduje solidnych podstaw.

Analiza korekcji odczytów symulowanych w porównaniu do rzeczywistych wskazuje, że symulacja jako składnik ewaluacji algorytmów korekcji jest użyteczna w umiarkowanym stopniu. Z jednej strony algorytmy pozwalające na skuteczną korekcję odczytów symulowanych dosyć dobrze sprawdzają się w korekcji odczytów rzeczywistych, a algorytmy najsłabsze przeciwnie. Jednak zależność ta nie ma bezpośredniego przełożenia między rodzajami odczytów, szczególnie w kontekście skali, ponieważ eksperymenty wykazały, że korekcja odczytów symulowanych może być przeprowadzona z nierealistycznie wysoką skutecznością. Należy zauważyć także, że bardziej prawdopodobne wyniki uzyskano dla odczytów symulowanych specjalizowanym algorytmem niż uproszczoną metodą Quake.

Wykorzystanie korekcji jako dodatkowego, ewentualnie możliwego do pominięcia kroku w przetwarzaniu sekwencji DNA jest obarczone ryzykiem wyboru algorytmu nieskutecznego lub, co gorsza, powodującego pogorszenie wyników. Jest też obarczone trudnościami technicznymi, związanymi z wysiłkiem uruchomienia algorytmu, dodatkowym czasem koniecznym do poświęcenia w celu wykonania korekcji oraz możliwością niepowodzenia korekcji, będącej efektem dużego zapotrzebowania na pamięć lub niestabilnością pracy algorytmu. Stąd też konieczne

jest staranne dokonanie wyboru algorytmu. Poniżej przedstawiono krótkie podsumowanie zaobserwowanych własności różnych rozwiązań.

6.6.1 RECKONER

Algorytm wykazuje bardzo wysoką skuteczność korekcji odczytów symulowanych — dla odczytów uzyskanych przy pomocy uproszczonej metody Quake osiągając najlepsze wyniki spośród testowanych algorytmów, a dla symulowanych narzędziem ART — najlepsze lub jedne z najlepszych. Algorytm pozwala na poprawę jakości asemblacji we wszystkich analizowanych przypadkach, z wyjątkiem pewnego pogorszenia wskaźnika ξ_{misasm} (choć generalnie jest to cecha algorytmów, które poprawiają pozostałe miary). Wpływ ten jest pozytywny także dla odczytów z sekwenatorów NovaSeq oraz MiSeq. Skuteczności korekcji można także dowieść przez obserwację wyników mapowania, ponieważ korzystnie wpływa na liczbę różnic między odczytami a genomem oraz w wielu przypadkach pozwala na powodzenie mapowania większej liczby odczytów. Algorytm umożliwia także skuteczną redukcję liczby błędów typu indel. Znaczenie korekcji algorytmem RECKONER dla detekcji wariantów ludzkiego genomu jest pozytywne wyłącznie dla niewielkiej głębokości sekwencjonowania, gdy celem jest detekcja wariantów typu SNP w odczytach dla zestawu referencyjnego GIAB, w pozostałych wpływ jest neutralny lub w niewielkim stopniu niekorzystne.

Korekcja jest przeprowadzana szybko, porównywalnie do innych dobrych algorytmów, jednocześnie algorytm jest bardzo łatwo skalowalny ze względu na liczbę jednostek obliczeniowych komputera. Zapotrzebowanie na pamięć jest niewielkie przy korekcji odczytów niewielkich genomów, jednak wzrasta znacząco dla odczytów genomów dużych rozmiarów, choć mankament ten występuje tylko w trybie z weryfikacją w oparciu o k''-mery, który jest zalecany wyłącznie dla korekcji mającej na celu wykorzystanie odczytów w asemblacji; tryb ten może zostać w razie potrzeby wyłączony, skutkując jednak spadkiem jakości. W algorytmie przewidziano strategię doboru głównego parametru i eksperymenty wykazały jej skuteczność. Jednocześnie pozostawiono możliwość doboru wartości parametru użytkownikowi. Algorytm wykazuje się niewielką wrażliwością na wartość tego parametru.

Spełnienie postawionych wymagań

Postawione w podrozdziale 5.4.2 wymagania dotyczące własności algorytmu w większości zostały spełnione. Algorytm zazwyczaj jest wykonywany szybko, niewiele odstając od najszybszych algorytmów, a jednocześnie w żadnym eksperymencie nie wymagając nieproporcjonalnie dużo czasu. Wyjątek stanowi korekcja zestawów odczytów największych genomów, jednak w tych przypadkach uzyskany czas obliczeń także należy uznać za rozsądny, a jednocześnie nie odstający znacząco

od konkurencyjnych rozwiązań. Dodatkowo wykazano, że algorytm bardzo dobrze skaluje się wraz ze zmianą liczby wątków, stąd jego przyspieszenie w wyniku zastosowania lepszego otoczenia sprzętowego powinno być skuteczne. W rozdziale 5 wykazano analitycznie, że dzięki wprowadzeniu wewnętrznych ograniczeń w działaniu algorytmu po spełnieniu postawionych założeń nie występuje ryzyko drastycznego wzrostu obliczeń ze względu na pesymistyczne własności danych wejściowych.

W podobny sposób można scharakteryzować zapotrzebowanie na pamięć — na ogół jest ono niewielkie, z wyjątkiem korekcji odczytów dwóch zestawów odczytów z największych genomów. Podczas gdy w omawianej skali czas obliczeń nie musi być barierą w uzyskaniu wyników obliczeń, to zapotrzebowanie na pamięć może to uniemożliwić, choć, jak zostało wspomniane, istnieje możliwość zmiany trybu pracy algorytmu, kosztem jakości, gdy celem wykorzystani odczytów jest asemblacja *de novo*. Wzrost zapotrzebowania na pamięć względem algorytmu BLESS częściowo jest rekompensowany ominięciem zjawiska fałszywych odpowiedzi twierdzących zapytań filtru Blooma.

Algorytm spełnia swoje podstawowe zadanie jakim jest poprawa jakości odczytów. Eksperymenty wykazały, że w większości przypadków sprawdza się dobrze w korekcji odczytów sekwenatorów Illumina o różnej charakterystyce: uzyskanych z różnych modeli urządzeń, o różnej głębokości sekwencjonowania i o różnej jakości. Typową sytuacją jest, że algorytm znajduje się w czołówce rozwiązań konkurencyjnych, a w niektórych przypadkach pozwala na uzyskanie najlepszych wyników. Eksperymenty nie wykazały istnienia znaczącego ryzyka, że zastosowanie algorytmu przyniesie negatywne skutki dla wyników, z wyjątkiem zadania detekcji ludzkich wariantów, choć ogólnie jest to zadanie trudne, w którym zastosowanie korekcji (poza wybranymi przypadkami) ma wątpliwe znaczenie.

Praktyczne wykorzystanie implementacji algorytmu nie powinno być źródłem problemów. Algorytm pozwala na zadanie głównego parametru, a jednocześnie został wyposażony w mechanizm automatycznego doboru. Jego działanie jest wprawdzie obarczone wadami, jednakże ryzyko popełnienia znaczącego błędu w konfiguracji jest niewielkie. Implementacja jest dostosowana do obsługi różnych konfiguracji plików wejściowych (sparowane lub niesparowane), w formie skompresowanej lub nie. Jednocześnie zadbano o prostą i intuicyjną instalację implementacji oraz jego uruchomienie.

6.6.2 Musket

Większość eksperymentów pozwala zaklasyfikować algorytm Musket jako przeciętny. Wyniki korekcji odczytów symulowanych okazały się słabe, choć w praktycznych zastosowaniach, w szczególności we wspomaganiu asemblacji i detekcji wariantów organizmu o średnim rozmiarze genomu uzyskane wyniki były przeciętne. Jednocześnie korekcja prawie zawsze powoduje zwiększenie liczby różnic

typu indel w odczytach. Algorytm nie pozwolił na przeprowadzenie korekcji odczytów sekwenatora NovaSeq. Wyzwaniem jest korekcja odczytów dużych genomów. Czas korekcji jest typowy, choć jednocześnie zapotrzebowanie na pamięć we wszystkich przypadkach okazało się być bardzo małe. Algorytm wymaga parametryzacji długością oligomeru, choć poza przypadkami dużych genomów (dla których algorytmu lepiej nie wykorzystywać) strategia doboru może ograniczyć się do wybrania największej możliwej wartości $k = 27$. Jednocześnie algorytm wymaga przekazania przybliżonej liczby k-merów w odczytach wejściowych, co stanowi proste do rozwiązania, ale potencjalnie uciążliwe wymaganie.

6.6.3 RACER

Wyniki eksperymentów przy użyciu odczytów symulowanych wskazują, że algorytm RACER jest algorytmem o przeciętnej jakości, choć wykazano też istnienie przypadków słabej korekcji. Analiza w oparciu o odczyty rzeczywiste skłania raczej ku tej drugiej możliwości — spośród wszystkich eksperymentów algorytm pozwolił na poprawę jakości asemblacji tylko dla dwóch zestawów danych, a w wielu przypadkach powodował ich pogorszenie. Szczególnie niepokojące są wyniki mapowania, gdzie pomimo przeciętnych wyników redukcji liczby różnic między odczytami a genomem często powodował wzrost liczby odczytów, które mapowały się do genomu wielokrotnie, z kolei skuteczność korekcji błędów typu indel mocno waha się w zależności od zestawu odczytów. Spośród testowanych algorytmów RACER jako jeden z dwóch wykazał się szczególnie wyraźnym negatywnym wpływem niewielkiej głębokości sekwencjonowania na jakość wyników. W przypadku wpływu na jakość detekcji wariantów ludzkiego genomu wykazano bardzo duże pogorszenie jakości.

Czas pracy algorytmu jest przeciętny, z kolei zapotrzebowanie na pamięć bardzo mocno wzrasta wraz z rozmiarem danych wejściowych (zarówno rozmiaru genomu, jak też głębokości sekwencjonowania) stawiając wymagania nie pozwalające na wykonanie części obliczeń. Do zalet algorytmu należy prostota użycia, polegająca na parametryzacji wyłącznie szacunkowym rozmiarem genomu, choć jednocześnie implementacja pozwala na przetwarzania wyłącznie nieskompresowanych plików niesparowanych odczytów.

6.6.4 BLESS

Eksperymenty wykorzystujące odczyty symulowane wskazują, że algorytm BLESS charakteryzuje się przeciętną skutecznością korekcji, choć w przypadku odczytów uzyskanych uproszczoną metodą Quake wyniki są zbliżone raczej do najsłabszych algorytmów. W większości przypadków wpływ korekcji na jakość asemblacji jest wyraźnie negatywny. Algorytm jest dosyć szybki i cechuje się niewielkim zapotrzebowaniem na pamięć, z wyjątkiem korekcji odczytów z sekwenatora No-

vaSeq, korekcja których wymagała bardzo dużej ilości czasu. Zaobserwowanym zjawiskiem jest bardzo duża liczba przypadków obcinania odczytów, w niektórych przypadkach (np. w korekcji zestawu P1_60) liczba usuniętych symboli była o dwa rzędy wielkości większa niż skorygowanych. Praktycznym ograniczeniem wykorzystania algorytmu jest niejasna metoda doboru długości oligomeru, mająca trudny do przewidzenia wpływ na uzyskane wyniki. Zaproponowana przez autorów algorytmu strategia jest niepraktyczna i nieskuteczna. Istotnymi mankamentami algorytmów są kwestie techniczne. Uruchomienie algorytmu wymaga włożenia sporego wysiłku w celu przeprowadzenia kompilacji w środowisku biblioteki MPI. Ponadto implementacja, choć pozwalająca na przetwarzanie danych skompresowanych, powoduje uzyskanie tymczasowych nieskompresowanych odczytów wynikowych, które są na samym końcu poddawane kompresji. W rezultacie w końcowym etapie pracy konieczne jest przechowanie na dysku odczytów wejściowych, wyjściowych oraz wyjściowych nieskompresowanych, zwiększając zapotrzebowanie na przestrzeń dyskową.

6.6.5 Fiona

Ocena jakości algorytmu Fiona w oparciu o odczyty symulowane znacząco różni się w zależności od metody symulacji: w przypadku uproszczonej metody Quake wyniki okazały się być słabe, z kolei w przypadku ART — przeciętne albo dobre. W wynikach asemblacji uwagę przykuwają dobre wyniki korekcji odczytów genomów niewielkich rozmiarów oraz przeciętnych lub dosyć dobrych wyników odczytów dużych genomów. Wyjątek stanowią odczyty uzyskane przy pomocy urządzenia NovaSeq, które okazały się być słabe. Wpływ korekcji na mapowanie odczytów we wszystkich przypadkach spowodował niewielki wzrost liczby zmapowanych odczytów. Algorytm, pomimo wyposażenia go w strategię korekcji błędów typu indel, w większości przypadków spowodował zwiększenie liczby różnic tego typu, choć można wskazać przypadki, gdy ich eliminacja okazała się być przeprowadzona skutecznie. Wykonanie algorytmu wymaga dosyć dużej ilości czasu oraz znacznego zapotrzebowania na pamięć, w tym ostatnim przypadku ograniczając zakres wykonanych eksperymentów. Co więcej, algorytm wykazuje niekorzystny wpływ na zapotrzebowanie na zasoby wraz ze zmianą głębokości sekwencjonowania. Z kolei skalowalność algorytmu jest przeciętna. Istotnym mankamentem implementacji okazało się wprowadzanie zmian do wyjściowych współczynników jakości, uniemożliwiając wykorzystanie odczytów w celu detekcji wariantów przy pomocy narzędzi wykorzystanych w eksperymentach. Ponadto algorytm nie pozwala na przetwarzanie skompresowanych plików odczytów. Jednocześnie jedynym obligatoryjnym parametrem algorytmu jest łatwo dostępny szacunkowy rozmiar genomu.

6.6.6 Blue

Korekcja odczytów symulowanych przy pomocy algorytmu Blue powoduje uzyskanie wyników nieco lepszych od przeciętnych. Z kolei wyniki asemblacji wskazują na wysoką jakość korekcji, w niektórych przypadkach uzyskane wyniki były najlepsze spośród wszystkich testowanych algorytmów. W przypadku mapowania nie zaobserwowano znaczącej zmiany liczby uzyskiwanych wyników, choć należy zauważyć, że zazwyczaj korekcja w małym stopniu zwiększa liczbę zmapowanych odczytów. Liczba różnic typu indel w mapowaniu również w wyniku korekcji zostaje zwykle nieco zredukowana. Jakość uzyskanych wyników w niewielkim stopniu ulegała zmianom długości oligomeru. Zapotrzebowanie algorytmu na zasoby jest umiarkowane, choć wyraźnie wzrasta wraz z korekcją odczytów dużych genomów. Zjawisko jest o tyle istotne, że w wielu przypadkach niemożliwe było wykonanie eksperymentów ze względu na przekroczenie limitu czasu lub brak dostępnej ilości pamięci. Algorytm słabo się skaluje, choć wpływ długości oligomeru oraz głębokości sekwencjonowania na jego zapotrzebowanie na zasoby należy ocenić jako dobry. Algorytm nie został wyposażony w automatyczną strategię doboru progu obcięcia, stąd wykonanie algorytmu wymaga jego parametryzacji. Do niedogodności technicznych implementacji algorytmu należy zaliczyć możliwość przetwarzania wyłączenie nieskompresowanych zestawów odczytów, a także konieczność zaopatrzenia w środowisko Microsoft .NET Framework.

6.6.7 Lighter

Możliwości korekcji odczytów symulowanych przy pomocy algorytmu Lighter różnią się w zależności od sposobu symulacji: w przypadku uproszczonej metody Quake uzyskane wyniki są przeciętne, z kolei w przypadku symulacji narzędziem ART — zdecydowanie słabe. Wyniki wpływu na asemblację wskazują na dobrą skuteczność korekcji. Wpływ korekcji na mapowanie zazwyczaj powodował zmniejszenie liczby odczytów zmapowanych jednokrotnie na rzecz zmapowanych wielokrotnie, choć zjawisko to nie jest szczególnie duże. Z kolei wpływ na liczbę różnic typu indel w odczytach także była niewielka, a ich kierunek różni się w zależności od eksperymentu. Wpływ korekcji na detekcję wariantów ludzkiego genomu praktycznie nie istnieje. Jakość wyników w zależności od długości oligomeru w rozsądnym zakresie zmienia się nieznacznie. Algorytm jest dosyć szybki, a jednocześnie najbardziej oszczędny pamięciowo ze wszystkich testowanych. Jednocześnie wpływ długości oligomeru oraz głębokości sekwencjonowania na zapotrzebowanie na zasoby należy ocenić pozytywnie. Algorytm w porównaniu do konkurencyjnych rozwiązań skaluje się bardzo słabo, choć ze względu na ogólnie dużą szybkość działania nie powinno stanowić to istotnego problemu. Algorytm wymaga parametryzacji prawdopodobieństwem konfigurującym główną strukturę danych. Autorzy algorytmu zaproponowali jednak konkretną metodę

jego doboru.

6.6.8 BFC

Skuteczność korekcji algorytmem BFC odczytów symulowanych zależy od sposobu ich uzyskania: dla symulacji narzędziem ART są one bardzo dobre, wskazując, że algorytm wykazuje bardzo wysoką skuteczność w korekcji błędów. Z drugiej strony interesująca jest umiarkowana skuteczność korekcji mniej realistycznych, ale „prostszych" odczytów uzyskanych uproszczoną metodą Quake. Jednocześnie zastosowanie algorytmu bardzo skutecznie poprawia jakość asemblacji, z wyjątkiem korekcji odczytów sekwenatorów NovaSeq. Wysoka skuteczność eliminacji błędów jest także widoczna w wynikach mapowania, choć nie dotyczy to błędów typu indel, które są korygowane w niewielkim stopniu. Zastosowanie algorytmu w potokach detekcji wariantów można zalecić dla dostatecznie dużej głębokości sekwencjonowania (co najmniej 45×), co może pozwolić na niewielką poprawę detekcji wariantów typu indel, jednak dla małych wartości głębokości wykorzystanie algorytmu może skutkować pogorszeniem rezultatów.

Korekcja algorytmem BFC jest przeprowadzana bardzo szybko — w zdecydowanej większości przypadków jest on najszybszym algorytmem, a jednocześnie cechuje się rozsądnie umiarkowanym zapotrzebowaniem na pamięć. Ponadto algorytm bardzo dobrze skaluje się wraz ze zwiększaniem liczby dostępnych jednostek obliczeniowych. Algorytm pracuje też bardzo stabilnie, kończąc się z powodzeniem we wszystkich eksperymentach. Jedyne zaobserwowane trudności techniczne obejmują tylko brak obsługi plików skompresowanych oraz odczytów sparowanych.

6.6.9 Karect

Wszystkie grupy eksperymentów wykazały bardzo pozytywny wpływ korekcji algorytmem Karect. Algorytm koryguje bardzo dużą liczbę błędów, na co wskazują zarówno wyniki eksperymentów z użyciem odczytów symulowanych, jak też ocena wpływu korekcji na mapowanie — skorygowane odczyty wykazują mniejszą liczbę różnic w stosunku do genomu, w porównaniu do nieskorygowanych. Pomimo niewykorzystania trybu korekcji błędów typu indel, algorytm pozwala na umiarkowaną ich redukcję. Korekcja algorytmem Karect wykazuje też wyraźnie korzystny wpływ na wyniki asemblacji, z wyjątkiem odczytów z sekwenatorów MiSeq. Algorytm nie wymaga też uciążliwej parametryzacji długością oligomeru.

Dobre wyniki są jednak okupione znacznym zapotrzebowaniem na zasoby. Wprawdzie algorytm daje możliwość określenia limitu pamięci (kosztem zwiększenia czasu obliczeń), jednak nawet przy określeniu dosyć liberalnego progu niektóre obliczenia mogą nie wykonać się w czasie doby, uniemożliwiając przeprowadzenie analizę wpływu korekcji na detekcję wariantów ludzkiego genomu. Ograniczenie

to może być trudne do wyeliminowania nawet w wyniku zwiększenia liczby jednostek obliczeniowych komputera ze względu na słabą skalowalność algorytmu. Jednocześnie w celu poprawy użyteczności należałoby wprowadzić poprawki implementacyjne, pozwalające na obsługę danych skompresowanych oraz redukcję zapotrzebowania na przestrzeń dyskową.

6.6.10 SAMDUDE

Próby praktycznego zastosowania algorytmu SAMDUDE należy uznać za bezcelowe. Implementacja sprawia problemy techniczne, stawia wysokie wymagania dotyczące zasobów, a uzyskane wyniki w niemal wszystkich przypadkach są słabe, prawie zawsze powodujące uzyskanie gorszych rezultatów niż w efekcie wykorzystania nieskorygowanych odczytów.

6.6.11 Ranking algorytmów

W celu podsumowania powyższych obserwacji i wniosków, wykonano ścisły ranking algorytmów. W tym celu niezależnie dokonano oceny siedmiu grup eksperymentów, w ramach których wyznaczono osobne rankingi cząstkowe. Ich wyniki zostały poddane sumowaniu. W każdym rankingu cząstkowym algorytm mógł uzyskać liczbę punktów z zakresu 1–10, przy czym mniejsza liczba punktów oznacza algorytm lepszy. W sytuacjach równych wyników dwóch algorytmów, dopuszczalne były także punkty ułamkowe, np. sytuacja: algorytm A — 1 pkt, B — 2,5 pkta, C — 2,5 pkta oznacza, że algorytm A jest najlepszy, natomiast algorytmy B i C należy zaklasyfikować na drugim miejscu.

Rankingi cząstkowe zostały przeprowadzone dla następujących grup eksperymentów: (*i*) odczyty symulowane metodą Quake (wyłącznie głębokość sekwencjonowania 60×), (*ii*) odczyty symulowane narzędziem ART (wyłącznie głębokość sekwencjonowania 60×), (*iii*) asemblacja (według miary NGA50 lub, jeśli była niedostępna, NA50), (*iv*) detekcja wariantów (wyłącznie głębokość sekwencjonowania 60×, według średniej geometrycznej miar F1 detekcji wariantów typu SNP i indel), (*v*) czas korekcji odczytów rzeczywistych, (*vi*) zapotrzebowanie na pamięć korekcji odczytów rzeczywistych, (*vii*) skalowalność (zgodnie z sytuacją przedstawioną na rys. 6.34, dla $\rho = 24$).

Wyniki przedstawiono w pierwszej części tabeli 6.3. Pozwalają one na stworzenie syntetycznego podsumowania, zgodnie z którym najlepszym algorytmem jest RECKONER, nieznacznie wyprzedzając BFC. Potwierdzeniem tych wyników jest drugi zaproponowany ranking, w którym wymienione wyżej eksperymenty zostały ocenione niezależnie od siebie, a następnie poddane sumowaniu. Uzyskane sumy zostały przedstawione w drugiej części tabeli. W tym przypadku również najlepszymi algorytmami są RECKONER oraz BFC, z niewielką przewagą na korzyść tego pierwszego.

Tabela 6.3: Ranking algorytmów w oparciu o grupy eksperymentów (pierwsza część tabeli): (*i*) odczyty symulowane metodą Quake, (*ii*) odczyty symulowane narzędziem ART, (*iii*) asemblacja, (*iv*) detekcja wariantów, (*v*) czas korekcji, (*vi*) zapotrzebowanie na pamięć, (*vii*) skalowalność; ranking algorytmów w oparciu o niezależne eksperymenty (druga część tabeli)

Grupa	RECKONER	Musket	RACER	BLESS	Fiona	Blue	Lighter	BFC	Karect	SAMDUDE
i	1	9	4	6,5	6,5	3	5	8	2	10
ii	3	8	7	4	6	5	9	1	2	10
iii	1	6	8	9	7	2,5	4,5	2,5	4,5	10
iv	3	4,5	7	2	9,5	6	4,5	1	8	9,5
v	4	6	5	2	7	8	3	1	9	10
vi	5	2	6	3	8	7	1	4	10	9
vii	1	4	9,5	6	3	7	8	2	5	9,5
Suma	**18**	**39,5**	**46,5**	**32,5**	**47**	**38,5**	**35**	**19,5**	**40,5**	**68**

	RECKONER	Musket	RACER	BLESS	Fiona	Blue	Lighter	BFC	Karect	SAMDUDE
Suma	**116**	**200,5**	**213,5**	**178,5**	**241,5**	**197**	**163,5**	**123,5**	**210**	**336**

Rozdział 7

Podsumowanie

W niniejszej pracy przedstawiono szereg zagadnień związanych z komputerową korekcją odczytów sekwencjonowania genomów. Wprowadzeniem do analizy przedmiotu było przytoczenie podstawowej wiedzy biologicznej, obejmującej kontekst zagadnienia budowy kwasu DNA. Wykonano również krótkie podsumowanie historii rozwoju technik i metod sekwencjonowania RNA i DNA, przy czym szczególny nacisk położono na analizę współczesnych, wysokoprzepustowych technik sekwencjonowania drugiej generacji. Wspomniano również o najnowszych rozwiązaniach, klasyfikowanych do grupy sekwencjonowania technik trzeciej generacji.

Nietrywialny technicznie charakter problemu komputerowego przetwarzania sekwencji nukleotydowych umotywował dokonanie przeglądu zagadnień algorytmicznych i technicznych, stosowanych lub potencjalnie nadających się do zastosowania w rozwiązaniu tego rodzaju zadań. Wprowadzono formalne definicje wybranych pojęć związanych z przetwarzaniem sekwencji nukleotydowych.

W dalszej kolejności dokonano przeglądu istniejących algorytmów korekcji odczytów z sekwencjonowania DNA, w szczególności uzyskanych w wyniku pracy urządzeń sekwencjonujących marki Illumina, przytoczono również skrócony przegląd metod korekcji odczytów innych rodzajów. Ponadto zwrócono uwagę na niektóre dostępne narzędzia komputerowej generacji sekwencji imitujących odczyty sekwencjonowania DNA.

W oparciu o dokonany przegląd wyciągnięto wniosek o możliwości zaproponowania nowego algorytmu korekcji odczytów z sekwenatorów Illumina. W tym celu przeprowadzono dokładną analizę działania algorytmu BLESS, którego ogólna zasada działania została zaadaptowana do opracowania i implementacji nowego algorytmu, klasyfikowanego jako algorytm oparty na spektrum k-merów — RECKONER. Dla obu algorytmów wyznaczono złożoności obliczeniowe wybranych przypadków korekcji odczytu.

Algorytm RECKONER wraz z dziewięcioma innymi algorytmami został pod-

dany eksperymentalnym testom, weryfikującym zapotrzebowanie algorytmów na zasoby oraz jakość uzyskanych rezultatów w zadaniu korekcji odczytów z sekwenatorów marki Illumina. Położono nacisk na uwzględnienie możliwie różnorodnych metod oceny, pozwalających na wyciągnięcie precyzyjnych wniosków oraz redukcję ryzyka niezaobserwowania wybranych aspektów działania i zastosowań algorytmów. Oparto się zarówno na rzeczywistych odczytach, udostępnionych w publicznych bazach danych jako efekt sekwencjonowania próbek DNA, jak też odczytów symulowanych komputerowo. Wykorzystano przy tym niektóre strategie oceny przedstawione w literaturze, a także wprowadzono nowe, w szczególności uwzględniające wpływ korekcji na proces detekcji wariantów genomów. Wyniki eksperymentów dały też wgląd w nietrywialną tematykę parametryzacji algorytmów.

Wśród celów, jakie przyświecały wykonanym zadaniom, należy wymienić opracowanie nowego algorytmu, który miał pozwalać na korekcję odczytów z sekwenatorów Illumina, cechować się dobrą jakością wyników, zachowując przy tym wysoką szybkość i umiarkowane zapotrzebowanie na pamięć. Przyjęto, że algorytm powinien być mało wrażliwy na efekty niedoskonałości modelowania danych sekwencjonowania oraz cechować się dobrą pesymistyczną czasową złożonością obliczeniową. Jednocześnie wykorzystanie algorytmu nie powinno wiązać się z nieadekwatnymi trudnościami technicznymi. Cel ten został osiągnięty, zapewniając przynajmniej częściowe spełnienie wymagań. Opracowany algorytm pozwala na uzyskanie dobrych, w niektórych przypadkach najlepszych spośród analizowanych algorytmów wyników. Co istotne, wyniki eksperymentów wykazały, że zastosowanie algorytmu nie powinno nieść za sobą znacznego ryzyka degradacji danych sekwencjonowania, w ogólnym przypadku realnego w niekorzystnych okolicznościach. Jego implementacja charakteryzuje się dosyć szybkim działaniem i bardzo dobrą skalowalnością ze względu na przetwarzanie równoległe, przy zapotrzebowaniu na pamięć współmiernym do rozmiaru genomu poddanego sekwencjonowaniu. Algorytm oparto na reprezentacji modelu odczytów w formie spektrum k-merów, wprowadzono tryb weryfikacji korekcji przy pomocy oligomerów dwóch długości (k-merów oraz k''-merów), a sam proces korekcji oparto zarówno na spektrum, jak też na informacjach o jakości, dostarczanych wraz z wejściowymi odczytami. Tym sposobem zredukowano zagrożenia płynące z uproszczeń modelu oraz kluczowość jego jakości. Udowodniono, że przy spełnieniu założeń złożoność czasowa algorytmu w zadaniu korekcji jednego odczytu nie przekracza pod względem rzędu kwadratu długości k-meru. W przygotowaniu oraz implementacji algorytmu uwzględniono kwestię łatwości użycia, w tym częściowego zautomatyzowania doboru głównego parametru oraz eliminację problemów dostrzeżonych w innych algorytmach.

W algorytmie zaproponowano szereg nowych rozwiązań, wliczając do nich m.in. sposób pamięciowej reprezentacji rozwiązań częściowych, korekcję błędów

typu indel przy pomocy metody z powrotami, uwzględnienie liczebności k-merów w odczytach wejściowych, wykorzystywanych w szczególności w opracowanej metodzie oceny i wyboru spośród wielu potencjalnych rozwiązań, dopuszczenie w wynikowych odczytach sekwencji k-merów nie należących do danych wejściowych oraz dodatkową, staranną korekcję w momencie natrafienia na symbole o niewielkich współczynnikach jakości.

Wysoka szybkość algorytmu została osiągnięta m.in. w drodze wprowadzenia limitów potencjalnie licznych rozpatrywanych przypadków lub uzyskiwanych wyników częściowych, przeprowadzenia zrównoleglenia algorytmu, wykonanego w sposób eliminujący konieczność wstępnego sprawdzania odczytów, oraz oddelegowania wykonania zadań przetwarzania zbiorów k-merów do specjalizowanych narzędzi KMC oraz KMC tools. Opracowano metody automatycznego doboru skali przyporządkowania współczynników jakości odczytów oraz wprowadzono możliwość kompresji wyników.

Ponadto wykorzystano rozwiązania, które w różnej formie były obecne w istniejących algorytmach, mając na celu osiągnięcie lepszych rezultatów poprzez synergiczne ich połączenie, w szczególności wspomniane wykorzystanie w procesie korekcji i weryfikacji wyników oligomerów dwóch długości oraz adaptację spotykanej w literaturze metody określania progu obcięcia spektrum k-merów.

Kolejny cel stanowiło dokonanie eksperymentalnej oceny działania wielu algorytmów korekcji. W oparciu o wyniki wykazano, że korekcja pozwala na poprawę rezultatów pracy typowych algorytmów wykorzystania odczytów sekwencjonowania, w szczególności asemblacji *de novo*, choć niejednokrotnie powodując uzyskanie sprzecznych wniosków w efekcie oparcia się na różnych kryteriach oceny wyników asemblacji. Pozytywny wpływ jest też obserwowalny w przypadku mapowania odczytów w zadaniach resekwencjonowania. Należy podkreślić, że wykorzystanie niektórych algorytmów niesie za sobą duże ryzyko uzyskania odczytów, których jakość negatywnie odbiega od odczytów niepoddanych korekcji. Zapotrzebowanie algorytmów na zasoby obliczeniowe charakteryzuje się bardzo dużym zróżnicowaniem, przy czym niektóre algorytmy mogą zostać wykonane nawet przy pomocy skromnego, domowego komputera. Ponadto nie potwierdzono hipotezy, zgodnie z którą korekcja odczytów może znacząco zredukować zapotrzebowanie na zasoby obliczeniowe algorytmów, wykorzystujących takie odczyty.

W ramach prac eksperymentalnych zaproponowano prototypowe metody oceny korekcji przy pomocy narzędzi detekcji wariantów genomów. Wykazano, że proces ten może pełnić rolę wskaźnika skuteczności korekcji, choć jest to podejście bardzo kosztowne obliczeniowo. Jakość detekcji wariantów w efekcie korekcji w pewnym stopniu ulega zmianom, w wybranych przypadkach zmiany te są korzystne. Mimo to zagadnienie wymaga dalszej analizy oraz rozszerzenia zakresu prac.

Osiągnięto cel porównania wyników uzyskanych przy pomocy dwóch metod

komputerowej symulacji z wynikami dla odczytów rzeczywistych. W ramach prac wykazano, że korekcja odczytów symulowanych cechuje się wyraźnie wyższą skutecznością, w niektórych przypadkach sięgającą 100%. Z drugiej strony takie wyniki są częściowo miarodajne w kontekście wykazania względnej skuteczności algorytmów korekcji.

W efekcie obserwacji wyników eksperymentów można postawić zalecenie wykorzystania w realnych warunkach pięciu spośród algorytmów poddanych analizie: RECKONER, Blue, Lighter, BFC oraz Karect. Żaden z algorytmów nie okazał się być najlepszy według wszystkich kryteriów, jednak w ocenie autora wybór któregokolwiek z nich powinien zazwyczaj przynieść korzyści bez istotnego ryzyka pogorszenia wyników. Jednocześnie zaobserwowane niedoskonałości nie powinny być kluczowe, a ich ewentualne wystąpienie wyraźnie zasugerować użytkownikowi wybór innego rozwiązania. W przypadku algorytmu RECKONER należy mieć na uwadze ryzyko wzrostu zapotrzebowania na pamięć przy korekcji odczytów organizmów o bardzo dużych genomach. Implementacja algorytmu Blue cechuje się niską pewnością, skutkując niemożliwością wykonania niektórych zadań. Algorytm Lighter powoduje uzyskanie dobrych wyników, ale słabszych od innych pozytywnie ocenionych algorytmów. Spośród słabości algorytmu BFC wykazano wyłącznie umiarkowaną skuteczność korekcji odczytów z sekwenatora NovaSeq. Z kolei algorytm Karect przede wszystkim wymaga dosyć dużego zapotrzebowania na pamięć, a jego wykonanie — znacznego czasu.

Zaproponowany zestaw algorytmów nie jest w pełni zgodny z rankingiem przedstawionym w podrozdziale 6.6.11, w szczególności wskutek uzyskania wysokiej noty przez algorytm BLESS. Jednakże ze względu na jego bardzo słabe wyniki w kontekście wpływu korekcji na asemblację *de novo* oraz trudności techniczne uruchomienia implementacji, zdecydowano się wykluczyć go z tej grupy.

W świetle powyższego podsumowania pierwszą tezę przedstawioną w rozdziale 1 należy uznać za udowodnioną. Wyróżnione wyżej algorytmy pozwalają na redukcję liczby błędów sekwencjonowania, co w sposób bezpośredni zostało wykazane w eksperymentach opartych na odczytach symulowanych. Z drugiej strony zaobserwowano korzystny wpływ korekcji na wyniki zadań przetwarzania sekwencji uzyskanych w wyniku rzeczywistej pracy urządzeń sekwencjonujących, w tym, zakładając niewielką głębokość sekwencjonowania, na detekcję wariantów genomów.

Opracowany algorytm RECKONER również był przedmiotem szeregu eksperymentów, które pozwoliły na zweryfikowanie postawionych mu wymagań. Uzyskane wyniki wskazują, że skuteczność algorytmu we wszystkich przypadkach jest dobra, z kolei przypadki pogorszenia jakości danych były tylko incydentalne. Wyniki uzyskano przy zachowaniu umiarkowanego zapotrzebowania algorytmu na zasoby, co stanowi dowód prawdziwości drugiej tezy pracy. Całościowe podsumowanie pozwala stwierdzić, że algorytm ten pozwala na osiągnięcie najbardziej

korzystnego kompromisu między jakością wyników a zapotrzebowaniem na zasoby obliczeniowe.

Przedstawiony w pracy algorytm oraz uzyskane wyniki eksperymentalne pozwoliły na przygotowanie artykułu w czasopiśmie [65], drugiego artykułu, znajdującego się obecnie w trakcie recenzji [66], rozdziałów w monografiach [64, 67] oraz plakatów konferencyjnych.

Pomimo szerokiego i w niektórych przypadkach głębokiego zakresu pracy, należy stwierdzić, że prace nad niektórymi podjętymi tematami są warte kontynuacji. W szczególności gwałtowny rozwój technik sekwencjonowania, owocujący możliwością uzyskania odczytów o innej charakterystyce, które przypuszczalnie w najbliższej przyszłości zdominują przedmiotową tematykę, może wymusić pogłębienie wiedzy na temat związanych z nimi błędów sekwencjonowania oraz odpowiednich technik korekcji. Jest to zagadnienie istotne mimo widocznych braków w wyczerpaniu dotychczasowej tematyki: podczas gdy liczba algorytmów korekcji odczytów sekwencjonowania Illumina jest duża, to w dalszym ciągu wiele z nich, w szczególności najnowszych, nie zostało poddanych niezależnej analizie.

Poruszone zagadnienie detekcji wariantów z pewnością wymaga dalszego opracowania. Interesującym kierunkiem może być zaproponowanie protokołu skutecznej oceny przy pomocy danych udostępnionych w ramach projektu 1001 Genomów, pozwalając na wykorzystanie bogactwa zgromadzonego w jego ramach informacji. Na uwadze należy mieć także możliwość pojawienia się nowych, starannie przygotowanych zestawów prawdy podstawowej dla innych niż człowiek organizmów modelowych, co pozwoliłoby na wykorzystanie protokołów oceny, na których w niniejszej pracy oparto się w celu przeprowadzenia analizy wykorzystując zestaw wariantów.

Problemy korekcji odczytów Illumina wielokrotnie były analizowane w literaturze, jednak należy mieć na uwadze, że skupiano się głównie na danych z sekwenatorów generujących odczyty długości 100 pz–150 pz, jak HiSeq 2000 lub HiSeq 2500. Dokładniejszego przeglądu wymaga częściowo poruszona w niniejszej pracy kwestia odczytów o innej charakterystyce sekwencji, tj. uzyskanych w wyniku pracy sekwenatorów z grupy MiSeq oraz NovaSeq. Jednocześnie należy mieć na uwadze obecność na rynku najnowszych urządzeń, jak NextSeq 2000 czy NovaSeq X, a także produktów innych producentów.

W pracy podjęto próbę wypracowania strategii doboru długości k-meru, zapewniającej uzyskanie jak najlepszych wyników. Zadanie to było poruszane w literaturze, jednak jak dotąd prawdopodobnie nie opracowano wnikliwego omówienia problemu, w szczególności umożliwiającego względnie dokładne wyznaczanie takiego parametru. I chociaż osiągnięcie pełnej ścisłości dla różnych algorytmów może być zagadnieniem bardzo trudnym, to istotność oraz wszechstronność tematyki, dotykającej również innego rodzaju problemów przetwarzania sekwencji nukleotydowych, są czynnikami stanowiącymi silne uzasadnienie wysiłków uszcze-

gółowienia istniejącej wiedzy.

Dokładna weryfikacja zaproponowanej strategii doboru długości k-meru jest istotna także w kontekście samego algorytmu RECKONER. Może ona obejmować wprowadzenie innej metody lub modyfikację stałych wartości strojących obecną. Może to zostać osiągnięte np. poprzez analizę większego zestawu odczytów, optymalizację pod kątem innych zastosowań algorytmu lub bardziej dogłębną analizę teoretyczną. Ponadto rozważaniom warto poddać poprawę skuteczności korekcji, szczególnie mając na uwadze nie w pełni satysfakcjonujące wyniki detekcji wariantów ludzkiego genomu. Można rozważyć przeprowadzenie starannego studium korekcji zbioru konkretnych odczytów. Ponadto można rozważyć zastosowanie k-merów dwóch długości już podczas korekcji odczytów, a nie tylko weryfikacji wyników. W ten sposób uzyskano by część zalet algorytmów opartych na drzewach/tablicach sufiksów. Warta podjęcia prac jest także redukcja zapotrzebowania na pamięć wykorzystywaną do przechowywania spektrum k-merów, co wiązałoby się z wprowadzeniem zmian w strukturach danych KMC.

Bibliografia

[1] Roche Shutting Down 454 Sequencing Business — GenomeWeb. https://www.genomeweb.com/sequencing/roche-shutting-down-454-sequencing-business, [dostęp: 20.09.2017]. [cytowanie na str. 20]

[2] Six Years After Acquisition, Roche Quietly Shutters 454 - Bio-IT World. http://www.bio-itworld.com/2013/10/16/six-years-after-acquisition-roche-quietly-shutters-454.html, [dostęp: 20.09.2017]. [cytowanie na str. 20]

[3] Sequencing Platforms — Compare NGS platforms (benchtop, production-scale). https://www.illumina.com/systems.html, [dostęp: 26.09.2017]. [cytowanie na str. 21, 29]

[4] NovaSeq™6000 System Quality Scores and RTA3 Software. https://www.illumina.com/content/dam/illumina-marketing/documents/products/appnotes/novaseq-hiseq-q30-app-note-770-2017-010.pdf, [dostęp: 12.12.2020]. [cytowanie na str. 97]

[5] hap.py. https://github.com/Illumina/hap.py, [dostęp: 29.08.2020]. [cytowanie na str. 245]

[6] Home - Genome - NCBI. https://www.ncbi.nlm.nih.gov/genome, [dostęp: 02.01.2021]. [cytowanie na str. 205, 335]

[7] The World's Largest Animal Genome. https://www.uni-wuerzburg.de/en/news-and-events/news/detail/news/the-worlds-largest-animal-genome/, [dostęp: 09.06.2023]. [cytowanie na str. 10]

[8] SOLiD™ System Accuracy. https://assets.thermofisher.com/TFS-Assets/LSG/Application-Notes/cms_057511.pdf, [dostęp: 05.10.2017]. [cytowanie na str. 23]

[9] Database Summary Paper Alpha List. http://www.oxfordjournals.org/nar/database/a/, [dostęp: 02.11.2017]. [cytowanie na str. 34]

[10] Genome 10K Project — Genome 10K Project. https://genome10k.soe.ucsc.edu/, [dostęp: 31.10.2017]. [cytowanie na str. 31]

[11] Project to read genomes of all 70,000 vertebrate species reports first discoveries. https://news.ucsc.edu/2021/04/vertebrate-genomes.html, [dostęp: 11.03.2023]. [cytowanie na str. 32]

[12] Home - dbGaP - NCBI, https://www.ncbi.nlm.nih.gov/gap, [dostęp: 02.11.2017]. [cytowanie na str. 34]

[13] MinION — Oxford Nanopore Technologies. https://nanoporetech.com/products/minion, [dostęp: 10.06.2023]. [cytowanie na str. 26]

[14] Advantages of nanopore sequencing. https://nanoporetech.com/how-it-works/advantages-nanopore-sequencing, [dostęp: 10.06.2023]. [cytowanie na str. 26]

[15] PacBio RS II - Pacific Biosciences. http://www.pacb.com/products-and-services/pacbio-systems/rsii/, [dostęp: 27.09.2017]. [cytowanie na str. 25]

[16] Primary and secondary databases — EMBL-EBI Train online. https://www.ebi.ac.uk/training/online/course/bioinformatics-terrified/what-database/relational-databases/primary-and-secondary-databases, [dostęp: 02.11.2017]. [cytowanie na str. 34]

[17] PromethION — Oxford Nanopore Technologies. https://nanoporetech.com/products/promethion, [dostęp: 10.06.2023]. [cytowanie na str. 26]

[18] ENA Browser. https://www.ebi.ac.uk/ena/about/statistics, [dostęp: 10.03.2023]. [cytowanie na str. 33]

[19] 100,000 Genomes Project — Genomics England. https://www.genomicsengland.co.uk/the-100000-genomes-project/, [dostęp: 11.03.2023]. [cytowanie na str. 32]

[20] The Sequence Read Archive (SRA) Overview. https://www.ncbi.nlm.nih.gov/sra/docs/, [dostęp: 02.11.2017]. [cytowanie na str. 34]

[21] Run Sequence — Thermo Fisher Scientific. https://www.thermofisher.com/pl/en/home/life-science/sequencing/next-generation-sequencing/ion-torrent-next-generation-sequencing-workflow/ion-torrent-next-generation-sequencing-run-sequence.html, [dostęp: 12.10.2017]. [cytowanie na str. 23]

[22] 1000 Plants. https://sites.google.com/a/ualberta.ca/onekp/, 2017. [dostęp: 31.10.2017]. [cytowanie na str. 31]

[23] A. V. Aho, J. E. Hopcroft oraz J. D. Ullman. Data structures and algorithms, 1983. [cytowanie na str. 41, 42, 43]

[24] A. Alic, A. Tomás, J. S. Torres, I. Medina oraz I. Blanquer. Robust Error Correction for De Novo Assembly via Spectral Partitioning and Sequence Alignment. In *IWBBIO*, pages 1040–1048, 2014. [cytowanie na str. 79]

[25] A. Allam, P. Kalnis oraz V. Solovyev. Karect: accurate correction of substitution, insertion and deletion errors for next-generation sequencing data. *Bioinformatics*, 31(21):3421–3428, 2015. [cytowanie na str. 2, 61, 62, 64, 67, 77, 90, 92, 93, 208, 220]

[26] C. Alonso-Blanco, J. Andrade, C. Becker, F. Bemm, J. Bergelson, K. M. Borgwardt, J. Cao, E. Chae, T. M. Dezwaan, W. Ding, et al. 1,135 genomes reveal the global pattern of polymorphism in Arabidopsis thaliana. *Cell*, 166(2):481–491, 2016. [cytowanie na str. 31, 245]

[27] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM, 1967. [cytowanie na str. 54]

[28] R. Ammar, T. A. Paton, D. Torti, A. Shlien oraz G. D. Bader. Long read nanopore sequencing for detection of HLA and CYP2D6 variants and haplotypes. *F1000Research*, 4, 2015. [cytowanie na str. 26, 29]

[29] E. L. Anson and E. W. Myers. ReAligner: a program for refining DNA sequence multi-alignments. *Journal of Computational Biology*, 4(3):369–383, 1997. [cytowanie na str. 80]

[30] W. J. Ansorge. Next-generation DNA sequencing techniques. *New biotechnology*, 25(4):195–203, 2009. [cytowanie na str. 1, 18, 19, 20, 21, 22, 26, 28, 29, 30]

[31] K. F. Au, J. G. Underwood, L. Lee oraz W. H. Wong. Improving PacBio long read accuracy by short read alignment. *PloS one*, 7(10):e46679, 2012. [cytowanie na str. 1, 24, 82]

[32] O. T. Avery, C. M. MacLeod oraz M. McCarty. Studies on the chemical nature of the substance inducing transformation of Pneumococcal types. *Journal of experimental medicine*, 79(2):137–158, 1944. [cytowanie na str. 12]

[33] R. Barthelson, A. J. McFarlin, S. D. Rounsley oraz S. Young. Plantagora: modeling whole genome sequencing and assembly of plant genomes. *PLoS One*, 6(12), 2011. [cytowanie na str. 91]

[34] S. Batzoglou, D. B. Jaffe, K. Stanley, J. Butler, S. Gnerre, E. Mauceli, B. Berger, J. P. Mesirov oraz E. S. Lander. ARACHNE: a whole-genome shotgun assembler. *Genome research*, 12(1):177–189, 2002. [cytowanie na str. 80]

[35] Y. Benjamini and T. P. Speed. Summarizing and correcting the GC content bias in high-throughput sequencing. *Nucleic acids research*, 40(10):e72–e72, 2012. [cytowanie na str. 27]

[36] G. Benoit, D. Lavenier, C. Lemaitre oraz G. Rizk. Bloocoo, a memory efficient read corrector. 2014. [cytowanie na str. 79]

[37] C. Bleidorn. Third generation sequencing: technology and its potential impact on evolutionary biodiversity research. *Systematics and biodiversity*, 14(1):1–8, 2016. [cytowanie na str. 24, 29, 82]

[38] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970. [cytowanie na str. 43, 44, 70]

[39] K. R. Bradnam, J. N. Fass, A. Alexandrov, P. Baranay, M. Bechner, I. Birol, S. Boisvert, J. A. Chapman, G. Chapuis, R. Chikhi, et al. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *GigaScience*, 2 (1):2047–217X, 2013. [cytowanie na str. 220]

[40] H. Buermans and J. Den Dunnen. Next generation sequencing technology: advances and applications. *Biochimica et Biophysica Acta (BBA)-Molecular Basis of Disease*, 1842(10):1932–1941, 2014. [cytowanie na str. 22]

[41] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, 1994. [cytowanie na str. 45]

[42] S. Caboche, C. Audebert, Y. Lemoine oraz D. Hot. Comparison of mapping algorithms used in high-throughput sequencing: application to Ion Torrent data. *BMC genomics*, 15(1):264, 2014. [cytowanie na str. 89]

[43] J. Cao, K. Schneeberger, S. Ossowski, T. Günther, S. Bender, J. Fitz, D. Koenig, C. Lanz, O. Stegle, C. Lippert, et al. Whole-genome sequencing of multiple Arabidopsis thaliana populations. *Nature genetics*, 43(10):956–963, 2011. [cytowanie na str. 249]

[44] F. Chen, M. Dong, M. Ge, L. Zhu, L. Ren, G. Liu oraz R. Mu. The history and advances of reversible terminators used in new generations of sequencing technology. *Genomics, proteomics & bioinformatics*, 11(1):34–40, 2013. [cytowanie na str. 21]

[45] R. Chikhi and P. Medvedev. Informed and automated k-mer size selection for genome assembly. *Bioinformatics*, 30(1):31–37, 2013. [cytowanie na str. 56, 257]

[46] R. Chikhi and G. Rizk. Space-efficient and exact de Bruijn graph representation based on a Bloom filter. *Algorithms for Molecular Biology*, 8(1):1–9, 2013. [cytowanie na str. 220]

[47] P. J. Cock, C. J. Fields, N. Goto, M. L. Heuer oraz P. M. Rice. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic acids research*, 38(6):1767–1771, 2009. [cytowanie na str. 38]

[48] P. E. Compeau, P. A. Pevzner oraz G. Tesler. How to apply de Bruijn graphs to genome assembly. *Nature biotechnology*, 29(11):987, 2011. [cytowanie na str. 47]

[49] . G. P. Consortium et al. A map of human genome variation from population-scale sequencing. *Nature*, 467(7319):1061, 2010. [cytowanie na str. 98]

[50] . G. P. Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68–74, 2015. [cytowanie na str. 31]

[51] I. H. G. S. Consortium et al. Finishing the euchromatic sequence of the human genome. *Nature*, 431(7011):931–945, 2004. [cytowanie na str. 17]

[52] T. Cormen, C. Leiserson, R. Rivest oraz C. Stein. *Wprowadzenie do algorytmów*. WNT, 2001. [cytowanie na str. 42, 43]

[53] Z. Czech. Wprowadzenie do obliczeń równoległych, wydanie II, 2013. [cytowanie na str. 51, 52, 53, 54, 269]

[54] Z. Czech, S. Deorowicz oraz P. Fabian. *Algorytmy i struktury danych: wybrane zagadnienia*. Wydawnictwo Politechniki Śląskiej, 2010. [cytowanie na str. 42, 47, 48]

[55] P. Danecek, A. Auton, G. Abecasis, C. A. Albers, E. Banks, M. A. DePristo, R. E. Handsaker, G. Lunter, G. T. Marth, S. T. Sherry, et al. The variant call format and VCFtools. *Bioinformatics*, 27(15):2156–2158, 2011. [cytowanie na str. 39]

[56] N. G. De Bruijn. A combinatorial problem. In *Nederl. Akad. Wetensch. Proc.*, volume 49, pages 758–764, 1946. [cytowanie na str. 46]

[57] R. Dechter and D. Frost. Backjump-based backtracking for constraint satisfaction problems. *Artificial Intelligence*, 136(2):147–188, 2002. [cytowanie na str. 48]

[58] A. Dedbudaj-Grabysz, S. Deorowicz oraz J. Widuch. *Algorytmy i struktury danych: wybór zaawansowanych metod*. Wydawnictwo Politechniki Śląskiej, 2010. [cytowanie na str. 42, 44, 45, 49, 50]

[59] S. Deorowicz, A. Debudaj-Grabysz oraz S. Grabowski. Disk-based k-mer counting on a PC. *BMC bioinformatics*, 14(1):160, 2013. [cytowanie na str. 41, 55, 56, 57, 58, 98]

[60] S. Deorowicz, M. Kokot, S. Grabowski oraz A. Debudaj-Grabysz. KMC 2: fast and resource-frugal k-mer counting. *Bioinformatics*, 31(10):1569–1576, 2015. [cytowanie na str. 57, 58, 98, 99]

[61] S. Deorowicz, A. Debudaj-Grabysz, A. Gudyś oraz S. Grabowski. Whisper: read sorting allows robust mapping of DNA sequencing data. *Bioinformatics*, 35(12): 2043–2050, 2019. [cytowanie na str. 246]

[62] M. A. DePristo, E. Banks, R. Poplin, K. V. Garimella, J. R. Maguire, C. Hartl, A. A. Philippakis, G. Del Angel, M. A. Rivas, M. Hanna, et al. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature genetics*, 43(5):491, 2011. [cytowanie na str. 245]

[63] E. W. Dijkstra. Cooperating sequential processes. In *The origin of concurrent programming*, pages 65–138. Springer, 1968. [cytowanie na str. 53]

[64] M. Długosz. Genome variant calling in context of sequencing reads correction. In *Recent Advances in computational oncology and personalized medicine*, pages 89–98. Springer, 2021. doi: https://doi.org/10.34918/83567. [cytowanie na str. 4, 287]

[65] M. Długosz and S. Deorowicz. Reckoner: read error corrector based on KMC. *Bioinformatics*, 33(7):1086–1089, 2017. [cytowanie na str. 4, 287]

[66] M. Długosz and S. Deorowicz. Illumina reads correction—evaluation and improvements. 2023. doi: https://doi.org/10.21203/rs.3.rs-2715541/v1. [cytowanie na str. 4, 287]

[67] M. Długosz, S. Deorowicz oraz M. Kokot. Improvements in DNA Reads Correction. In *Man-Machine Interactions 5: 5th International Conference on Man-Machine Interactions, ICMMI 2017 Held at Kraków, Poland, October 3-6, 2017*, pages 115–124. Springer, 2018. [cytowanie na str. 4, 287]

[68] U. Drepper. What every programmer should know about memory. *Red Hat, Inc*, 11:2007, 2007. [cytowanie na str. 41]

[69] M. Dufva. Introduction to microarray technology. *DNA Microarrays for Biomedical Research: Methods and Protocols*, pages 1–22, 2009. [cytowanie na str. 32, 33]

[70] S. R. Eddy. The C-value paradox, junk DNA and ENCODE. *Current biology*, 22 (21):R898–R899, 2012. [cytowanie na str. 11]

[71] E. M. Elnifro, A. M. Ashshi, R. J. Cooper oraz P. E. Klapper. Multiplex PCR: optimization and application in diagnostic virology. *Clinical microbiology reviews*, 13(4):559–570, 2000. [cytowanie na str. 32]

[72] S. J. Emrich, S. Aluru, Y. Fu, T.-J. Wen, M. Narayanan, L. Guo, D. A. Ashlock, and P. S. Schnable. A strategy for assembling the maize (Zea mays L.) genome. *Bioinformatics*, 20(2):140–147, 2004. [cytowanie na str. 206]

[73] M. Escalona, S. Rocha oraz D. Posada. A comparison of tools for the simulation of genomic next-generation sequencing data. *Nature Reviews Genetics*, 17(8):459, 2016. [cytowanie na str. 88]

[74] B. Eslami-Mossallam, R. D. Schram, M. Tompitak, J. van Noort oraz H. Schiessel. Multiplexing genetic and nucleosome positioning codes: a computational approach. *PloS one*, 11(6):e0156905, 2016. [cytowanie na str. 8]

[75] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398. IEEE, 2000. [cytowanie na str. 45, 46, 51]

[76] P. Ferragina, G. Manzini, V. Mäkinen oraz G. Navarro. An alphabet-friendly FM-index. In *International Symposium on String Processing and Information Retrieval*, pages 150–160. Springer, 2004. [cytowanie na str. 46]

[77] W. Fiers, G. Haegeman, D. Iserentant oraz W. Min Jou. Nucleotide sequence of the gene coding for the bacteriophage MS2 coat protein. *Nature*, 237, 1972. [cytowanie na str. 12, 14]

[78] W. Fiers, R. Contreras, F. Duerinck, G. Haegeman, D. Iserentant, J. Merregaert, W. M. Jou, F. Molemans, A. Raeymaekers, A. Van den Berghe, et al. Complete nucleotide sequence of bacteriophage MS2 RNA: primary and secondary structure of the replicase gene. *Nature*, 260(5551):500–507, 1976. [cytowanie na str. 13, 14]

[79] I. Fischer-Hwang, I. Ochoa, T. Weissman oraz M. Hernaez. Denoising of Aligned Genomic Data. *Scientific reports*, 9(1):1–11, 2019. [cytowanie na str. 37, 66, 67, 77, 93]

[80] P. Gajer, M. Schatz oraz S. L. Salzberg. Automated correction of genome sequence errors. *Nucleic acids research*, 32(2):562–569, 2004. [cytowanie na str. 80]

[81] M. Y. Galperin, X. M. Fernández-Suárez oraz D. J. Rigden. The 24th annual Nucleic Acids Research database issue: a look back and upcoming changes. *Nucleic acids research*, 45(D1):D1–D11, 2017. [cytowanie na str. 34]

[82] W. Gilbert and A. Maxam. The nucleotide sequence of the lac operator. *Proceedings of the National Academy of Sciences*, 70(12):3581–3584, 1973. [cytowanie na str. 15]

[83] A. Git, H. Dvinge, M. Salmon-Divon, M. Osborne, C. Kutter, J. Hadfield, P. Bertone oraz C. Caldas. Systematic comparison of microarray profiling, real-time PCR oraz next-generation sequencing technologies for measuring differential microRNA expression. *Rna*, 16(5):991–1006, 2010. [cytowanie na str. 32]

[84] F. W. Glover and G. A. Kochenberger. *Handbook of metaheuristics*, volume 57. Springer Science & Business Media, 2006. [cytowanie na str. 51]

[85] S. Gnerre, I. MacCallum, D. Przybylski, F. J. Ribeiro, J. N. Burton, B. J. Walker, T. Sharpe, G. Hall, T. P. Shea, S. Sykes, et al. High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proceedings of the National Academy of Sciences*, 108(4):1513–1518, 2011. [cytowanie na str. 79]

[86] S. Gomaa, N. A. Belal oraz Y. El-Sonbaty. H-RACER: Hybrid RACER to Correct Substitution, Insertion oraz Deletion Errors. In *International Conference on Bioinformatics and Biomedical Engineering*, pages 62–73. Springer, 2017. [cytowanie na str. 79]

[87] M. G. Grabherr, B. J. Haas, M. Yassour, J. Z. Levin, D. A. Thompson, I. Amit, X. Adiconis, L. Fan, R. Raychowdhury, Q. Zeng, et al. Full-length transcriptome assembly from RNA-Seq data without a reference genome. *Nature biotechnology*, 29(7):644, 2011. [cytowanie na str. 56]

[88] S. Grabowski, G. Navarro, R. Przywarski, A. Salinger oraz V. Mäkinen. A simple alphabet-independent FM-index. *International Journal of Foundations of Computer Science*, 17(06):1365–1384, 2006. [cytowanie na str. 46]

[89] P. Greenfield, K. Duesing, A. Papanicolaou oraz D. C. Bauer. Blue: correcting sequencing errors using consensus and context. *Bioinformatics*, 30(19):2723–2732, 2014. [cytowanie na str. 61, 62, 67, 69, 74, 86, 88, 91, 92, 93, 95, 138, 202, 220, 233]

[90] A. K. Gupta and U. Gupta. Next Generation Sequencing and Its Applications-Chapter 19. *Animal Biotechnology*, pages 345–367, 2014. [cytowanie na str. 22]

[91] E. Gurari. CIS 680: Data Structures: Chapter 19: Backtracking Algorithms, 1999. [cytowanie na str. 47]

[92] A. Gurevich, V. Saveliev, N. Vyahhi oraz G. Tesler. QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, 2013. [cytowanie na str. 91, 220]

[93] D. Haussler, S. J. O'Brien, O. A. Ryder, F. K. Barker, M. Clamp, A. J. Crawford, R. Hanner, O. Hanotte, W. E. Johnson, J. A. McGuire, et al. Genome 10K: a proposal to obtain whole-genome sequence for 10 000 vertebrate species. *Journal of Heredity*, pages 659–674, 2009. [cytowanie na str. 31]

[94] S. R. Head, H. K. Komori, S. A. LaMere, T. Whisenant, F. Van Nieuwerburgh, D. R. Salomon oraz P. Ordoukhanian. Library construction for next-generation sequencing: overviews and challenges. *Biotechniques*, 56(2):61, 2014. [cytowanie na str. 19]

[95] J. M. Heather and B. Chain. The sequence of sequencers: the history of sequencing DNA. *Genomics*, 107(1):1–8, 2016. [cytowanie na str. 12, 13, 14, 17]

[96] Y. Heo. BLESS 0.12 — kod źródłowy. https://sourceforge.net/projects/bless-ec/files/bless.v0p12.tgz. [dostęp: 14.05.2020]. [cytowanie na str. 102]

[97] Y. Heo, X.-L. Wu, D. Chen, J. Ma oraz W.-M. Hwu. BLESS: bloom filter-based error correction solution for high-throughput sequencing reads. *Bioinformatics*, 30 (10):1354–1362, 2014. [cytowanie na str. 2, 61, 63, 72, 84, 92, 93, 94, 101, 202]

[98] Y. Heo, A. Ramachandran, W.-M. Hwu, J. Ma oraz D. Chen. BLESS 2: accurate, memory-efficient and fast error correction method. *Bioinformatics*, 32(15):2369–2371, 2016. [cytowanie na str. 67, 72, 92, 93, 95, 208]

[99] C. Hercus. Novocraft short read alignment package. *Website http://www. novocraft. com*, 2009. [cytowanie na str. 82]

[100] M. Heydari, G. Miclotte, P. Demeester, Y. Van de Peer oraz J. Fostier. Evaluation of the impact of Illumina error correction tools on de novo genome assembly. *BMC bioinformatics*, 18(1):374, 2017. [cytowanie na str. 67, 68, 89, 90, 92, 93]

[101] E. Holak, L. Hoppe, W. Lewiński, I. Lipka oraz B. Ruda-Groborz. *Vademecum maturalne 2008*. Operon, 2008. [cytowanie na str. 8, 10]

[102] R. W. Holley, J. Apgar, G. A. Everett, J. T. Madison, M. Marquisee, S. H. Merrill, J. R. Penswick oraz A. Zamir. Structure of a ribonucleic acid. *Science*, pages 1462–1465, 1965. [cytowanie na str. 14]

[103] M. Holtgrewe. Mason: a read simulator for second generation sequencing data. 2010. [cytowanie na str. 87, 90, 211]

[104] M. Howison, F. Zapata oraz C. W. Dunn. Toward a statistically explicit understanding of de novo sequence assembly. *Bioinformatics*, 29(23):2959–2963, 2013. [cytowanie na str. 30]

[105] W. Huang, L. Li, J. R. Myers oraz G. T. Marth. ART: a next-generation sequencing read simulator. *Bioinformatics*, 28(4):593–594, 2012. [cytowanie na str. 87, 88]

[106] Y.-T. Huang and Y.-W. Huang. An efficient error correction algorithm using FM-index. *BMC bioinformatics*, 18(1):524, 2017. [cytowanie na str. 79]

[107] L. Ilie and M. Molnar. RACER: Rapid and accurate correction of errors in reads. *Bioinformatics*, 29(19):2490–2493, 2013. [cytowanie na str. 2, 69, 71, 93, 208, 220]

[108] L. Ilie, F. Fazayeli oraz S. Ilie. HiTEC: accurate error correction in high-throughput sequencing data. *Bioinformatics*, 27(3):295–302, 2011. [cytowanie na str. 79, 136]

[109] W.-C. Kao, A. H. Chan oraz Y. S. Song. ECHO: a reference-free short-read error correction algorithm. *Genome research*, 21(7):1181–1192, 2011. [cytowanie na str. 79]

[110] D. R. Kelley, M. C. Schatz oraz S. L. Salzberg. Quake: quality-aware detection and correction of sequencing errors. *Genome biology*, 11(11):R116, 2010. [cytowanie na str. 61, 62, 67, 68, 80, 84, 88, 90, 91, 92, 93, 135, 143, 257]

[111] S. Kim, K. Scheffler, A. L. Halpern, M. A. Bekritsky, E. Noh, M. Källberg, X. Chen, Y. Kim, D. Beyter, P. Krusche, et al. Strelka2: fast and accurate calling of germline and somatic variants. *Nature methods*, 15(8):591–594, 2018. [cytowanie na str. 245]

[112] M. Kircher and J. Kelso. High-throughput DNA sequencing–concepts and limitations. *Bioessays*, 32(6):524–536, 2010. [cytowanie na str. 29]

[113] D. E. Knuth. *Sztuka programowania. T. 1, Algorytmy podstawowe*. Wydawnictwa Naukowo-Techniczne, 2002. [cytowanie na str. 41]

[114] M. Kokot. KMC API documentation. https://github.com/refresh-bio/KMC/blob/v3.0.0/API.pdf, . [dostęp: 05.06.2020]. [cytowanie na str. 99]

[115] M. Kokot. Wyznaczanie zbiorów podsłów sekwencji nukleotydowych w danych z sekwencjonowania genomów. . [cytowanie na str. 59]

[116] M. Kokot, S. Deorowicz oraz A. Debudaj-Grabysz. Sorting data on ultra-large scale with RADULS. In *International Conference: Beyond Databases, Architectures and Structures*, pages 235–245. Springer, 2017. [cytowanie na str. 59]

[117] M. Kokot, M. Długosz oraz S. Deorowicz. KMC 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, 2017. [cytowanie na str. 57, 58, 98, 100, 101]

[118] G. Kondrak and P. Van Beek. A theoretical evaluation of selected backtracking algorithms. *Artificial Intelligence*, 89(1-2):365–387, 1997. [cytowanie na str. 48]

[119] C. Koulamas. A new constructive heuristic for the flowshop scheduling problem. *European Journal of Operational Research*, 105(1):66–71, 1998. [cytowanie na str. 50]

[120] T. Kowalski, S. Grabowski oraz S. Deorowicz. Indexing arbitrary-length k-mers in sequencing reads. *PloS one*, 10(7):e0133198, 2015. [cytowanie na str. 40]

[121] S. Kurtz, A. Phillippy, A. L. Delcher, M. Smoot, M. Shumway, C. Antonescu oraz S. L. Salzberg. Versatile and open software for comparing large genomes. *Genome biology*, 5(2):R12, 2004. [cytowanie na str. 92]

[122] S. Kurtz, A. Narechania, J. C. Stein oraz D. Ware. A new method to compute K-mer frequencies and its application to annotate large repetitive plant genomes. *BMC genomics*, 9(1):517, 2008. [cytowanie na str. 57]

[123] D. Laehnemann, A. Borkhardt oraz A. C. McHardy. Denoising DNA deep sequencing data—high-throughput sequencing errors and their correction. *Briefings in bioinformatics*, 17(1):154–179, 2015. [cytowanie na str. 26, 27, 29, 63, 64, 65, 74, 84, 85, 202]

[124] E. S. Lander, L. M. Linton, B. Birren, C. Nusbaum, M. C. Zody, J. Baldwin, K. Devon, K. Dewar, M. Doyle, W. FitzHugh, et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, 2001. [cytowanie na str. 1, 17]

[125] B. Langmead and S. L. Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature methods*, 9(4):357–359, 2012. [cytowanie na str. 40]

[126] H.-S. Le, M. H. Schulz, B. M. McCauley, V. F. Hinman oraz Z. Bar-Joseph. Probabilistic error correction for RNA sequencing. *Nucleic acids research*, 41(10): e109–e109, 2013. [cytowanie na str. 66]

[127] H. Li. Tabix: fast retrieval of sequence features from generic TAB-delimited files. *Bioinformatics*, 27(5):718–719, 2011. [cytowanie na str. 38, 39]

[128] H. Li. BFC: correcting Illumina sequencing errors. *Bioinformatics*, 31(17):2885–2887, 2015. [cytowanie na str. 62, 68, 76, 92, 93, 208, 211, 233]

[129] H. Li and R. Durbin. Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009. [cytowanie na str. 204, 233, 245]

[130] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis oraz R. Durbin. The sequence alignment/map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, 2009. [cytowanie na str. 39, 78, 204, 245]

[131] H. Li, J. M. Bloom, Y. Farjoun, M. Fleharty, L. Gauthier, B. Neale oraz D. MacArthur. A synthetic-diploid benchmark for accurate variant-calling evaluation. *Nature methods*, 15(8):595–597, 2018. [cytowanie na str. 208, 244, 245]

[132] R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, et al. De novo assembly of human genomes with massively parallel short read sequencing. *Genome research*, 20(2):265–272, 2010. [cytowanie na str. 245]

[133] Y. Li et al. MSPKmerCounter: a fast and memory efficient approach for k-mer counting. *arXiv preprint arXiv:1505.06550*, 2015. [cytowanie na str. 59]

[134] E.-C. Lim, J. Müller, J. Hagmann, S. R. Henz, S.-T. Kim oraz D. Weigel. Trowel: a fast and accurate error correction module for Illumina sequencing reads. *Bioinformatics*, 30(22):3264–3265, 2014. [cytowanie na str. 40, 78, 92, 93, 257]

[135] A. Limasset, J.-F. Flot oraz P. Peterlongo. Toward perfect reads: self-correction of short reads via mapping on de bruijn graphs. *arXiv preprint arXiv:1711.03336*, 2017. [cytowanie na str. 79, 208]

[136] L. Liu, Y. Li, S. Li, N. Hu, Y. He, R. Pong, D. Lin, L. Lu oraz M. Law. Comparison of next-generation sequencing systems. *BioMed Research International*, 2012, 2012. [cytowanie na str. 16, 20]

[137] Y. Liu, B. Schmidt oraz D. L. Maskell. DecGPU: distributed error correction on massively parallel graphics processing units using CUDA and MPI. *BMC bioinformatics*, 12(1):85, 2011. [cytowanie na str. 61, 63, 79]

[138] Y. Liu, J. Schröder oraz B. Schmidt. Musket: a multistage k-mer spectrum-based error corrector for Illumina sequence data. *Bioinformatics*, 29(3):308–315, 2012. [cytowanie na str. 61, 62, 67, 70, 90, 92, 95, 130, 208]

[139] R. Luo, B. Liu, Y. Xie, Z. Li, W. Huang, J. Yuan, G. He, Y. Chen, Q. Pan, Y. Liu, et al. SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. *Gigascience*, 1(1):2047–217X, 2012. [cytowanie na str. 79]

[140] L. Mamanova, A. J. Coffey, C. E. Scott, I. Kozarewa, E. H. Turner, A. Kumar, E. Howard, J. Shendure oraz D. J. Turner. Target-enrichment strategies for next-generation sequencing. *Nature methods*, 7(2):111–118, 2010. [cytowanie na str. 30]

[141] G. Marçais and C. Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6):764–770, 2011. [cytowanie na str. 40, 41, 56, 57, 68]

[142] G. Marçais, J. A. Yorke oraz A. Zimin. QuorUM: an error corrector for Illumina reads. *PLoS One*, 10(6), 2015. [cytowanie na str. 61, 67, 68, 79, 91, 92, 94]

[143] E. Marinier, D. G. Brown oraz B. J. McConkey. Pollux: platform independent error correction of single and mixed genomes. *BMC bioinformatics*, 16(1):10, 2015. [cytowanie na str. 61, 67, 79, 92, 93]

[144] N. Matasci, L.-H. Hung, Z. Yan, E. J. Carpenter, N. J. Wickett, S. Mirarab, N. Nguyen, T. Warnow, S. Ayyampalayam, M. Barker, et al. Data access for the 1,000 Plants (1KP) project. *GigaScience*, 3(1):17, 2014. [cytowanie na str. 31]

[145] A. M. Maxam and W. Gilbert. A new method for sequencing DNA. *Proceedings of the National Academy of Sciences*, 74(2):560–564, 1977. [cytowanie na str. 15]

[146] P. Medvedev, E. Scott, B. Kakaradov oraz P. Pevzner. Error correction of high-throughput sequencing datasets with non-uniform coverage. *Bioinformatics*, 27 (13):i137–i141, 2011. [cytowanie na str. 79]

[147] P. Melsted and J. K. Pritchard. Efficient counting of k-mers in DNA sequences using a bloom filter. *BMC bioinformatics*, 12(1):333, 2011. [cytowanie na str. 43, 56, 57]

[148] M. L. Metzker. Sequencing technologies–the next generation. *Nature reviews. Genetics*, 11(1):31, 2010. [cytowanie na str. 1, 20, 22, 27, 28, 29, 30, 32, 33]

[149] Z. Michalewicz and D. B. Fogel. *How to solve it: modern heuristics*. Springer Science & Business Media, 2013. [cytowanie na str. 49, 50, 51]

[150] G. Miclotte, M. Heydari, P. Demeester, P. Audenaert oraz J. Fostier. Jabba: Hybrid error correction for long sequencing reads using maximal exact matches. In *International Workshop on Algorithms in Bioinformatics*, pages 175–188. Springer, 2015. [cytowanie na str. 83]

[151] A. S. Mikheyev and M. M. Tin. A first look at the Oxford Nanopore MinION sequencer. *Molecular ecology resources*, 14(6):1097–1102, 2014. [cytowanie na str. 26]

[152] A. E. Minoche, J. C. Dohm oraz H. Himmelbauer. Evaluation of genomic high-throughput sequencing data generated on Illumina HiSeq and genome analyzer systems. *Genome biology*, 12(11):R112, 2011. [cytowanie na str. 27, 66, 136]

[153] K. Mitchell, I. Mandric, J. Brito, Q. Wu, S. Knyazev, S. Chang, L. S. Martin, A. Karlsberg, E. Gerasimov, R. Littman, et al. Benchmarking of computational error-correction methods for next-generation sequencing data. *bioRxiv*, page 642843, 2019. [cytowanie na str. 89, 94]

[154] M. Molnar and L. Ilie. Correcting Illumina data. *Briefings in bioinformatics*, 16 (4):588–599, 2014. [cytowanie na str. 2, 62, 89, 90, 92, 94, 207]

[155] O. Morozova and M. A. Marra. Applications of next-generation sequencing technologies in functional genomics. *Genomics*, 92(5):255–264, 2008. [cytowanie na str. 16]

[156] S. Munchel, Y. Hoang, Y. Zhao, J. Cottrell, B. Klotzle, A. K. Godwin, D. Koestler, P. Beyerlein, J.-B. Fan, M. Bibikova, et al. Targeted or whole genome sequencing of formalin fixed tissue samples: potential applications in cancer genomics. *Oncotarget*, 6(28):25943, 2015. [cytowanie na str. 28]

[157] G. Myers. A dataset generator for whole genome shotgun sequencing. In *ISMB*, pages 202–210, 1999. [cytowanie na str. 88]

[158] K. Nakamura, T. Oshima, T. Morimoto, S. Ikeda, H. Yoshikawa, Y. Shiwa, S. Ishikawa, M. C. Linak, A. Hirai, H. Takahashi, et al. Sequence-specific error profile of Illumina sequencers. *Nucleic acids research*, 39(13):e90–e90, 2011. [cytowanie na str. 27]

[159] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970. [cytowanie na str. 65]

[160] C. Ortutay and Z. Ortutay. *Molecular Data Analysis Using R*. Wiley Online Library, 2017. [cytowanie na str. 141]

[161] S. Pattnaik, S. Gupta, A. A. Rao oraz B. Panda. SInC: an accurate and fast error-model based simulator for SNPs, Indels and CNVs coupled with a read generator for short-read sequence data. *BMC bioinformatics*, 15(1):40, 2014. [cytowanie na str. 88]

[162] P. A. Pevzner, H. Tang oraz M. S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the national academy of sciences*, 98(17): 9748–9753, 2001. [cytowanie na str. 80]

[163] P. A. Pevzner, H. Tang oraz M. S. Waterman. A new approach to fragment assembly in DNA sequencing. In *Proceedings of the fifth annual international conference on Computational biology*, pages 256–267. ACM, 2001. [cytowanie na str. 64]

[164] S. Pfeifer. From next-generation resequencing reads to a high-quality variant data set. *Heredity*, 2016. [cytowanie na str. 17, 21]

[165] N. Philippe, M. Salson, T. Lecroq, M. Leonard, T. Commes oraz E. Rivals. Querying large read collections in main memory: a versatile data structure. *BMC bioinformatics*, 12(1):242, 2011. [cytowanie na str. 44, 45, 46, 56]

[166] A. Polański. Podstawy bioinformatyki, 2010. [cytowanie na str. 9, 16, 35, 42, 45, 65]

[167] A. Polanski and M. Kimmel. Bioinformatics, 2007. [cytowanie na str. 66]

[168] F. Putze, P. Sanders oraz J. Singler. Cache-, hash-and space-efficient bloom filters. *Experimental Algorithms*, pages 108–121, 2007. [cytowanie na str. 43, 44]

[169] M. A. Quail, M. Smith, P. Coupland, T. D. Otto, S. R. Harris, T. R. Connor, A. Bertoni, H. P. Swerdlow oraz Y. Gu. A tale of three next generation sequencing platforms: comparison of Ion Torrent, Pacific Biosciences and Illumina MiSeq sequencers. *BMC genomics*, 13(1):341, 2012. [cytowanie na str. 29]

[170] M. G. Resende and J. P. de Sousa. *Metaheuristics: computer decision-making*, volume 86. Springer Science & Business Media, 2013. [cytowanie na str. 49, 50]

[171] J. A. Reuter, D. V. Spacek oraz M. P. Snyder. High-throughput sequencing technologies. *Molecular cell*, 58(4):586–597, 2015. [cytowanie na str. 1, 18, 22, 23, 24, 25, 26]

[172] A. Rhoads and K. F. Au. PacBio sequencing and its applications. *Genomics, proteomics & bioinformatics*, 13(5):278–289, 2015. [cytowanie na str. 24, 25]

[173] G. Rizk, D. Lavenier oraz R. Chikhi. DSK: k-mer counting with very low memory usage. *Bioinformatics*, 29(5):652–653, 2013. [cytowanie na str. 41, 57]

[174] M. Roberts, W. Hayes, B. R. Hunt, S. M. Mount oraz J. A. Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18): 3363–3369, 2004. [cytowanie na str. 58]

[175] M. Roberts, B. R. Hunt, J. A. Yorke, R. A. Bolanos oraz A. L. Delcher. A preprocessor for shotgun assembly of large genomes. *Journal of computational biology*, 11 (4):734–752, 2004. [cytowanie na str. 58]

[176] K. Rojek, Ł. Szustak oraz R. Wyrzykowski. *Zrównoleglanie i automatyczne dostosowanie algorytmów numerycznych do architektur hybrydowych z akceleratorami GPU*. PWN, 2015. [cytowanie na str. 51, 55]

[177] L. Salmela. Correction of sequencing errors in a mixed set of reads. *Bioinformatics*, 26(10):1284–1290, 2010. [cytowanie na str. 79]

[178] L. Salmela and E. Rivals. LoRDEC: accurate and efficient long read error correction. *Bioinformatics*, 30(24):3506–3514, 2014. [cytowanie na str. 82]

[179] L. Salmela and J. Schröder. Correcting errors in short reads by multiple alignments. *Bioinformatics*, 27(11):1455–1461, 2011. [cytowanie na str. 61, 62, 64, 65, 79, 202]

[180] F. Sanger and A. R. Coulson. A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase. *Journal of molecular biology*, 94(3): 441IN19447–446IN20448, 1975. [cytowanie na str. 15]

[181] F. Sanger, G. Brownlee oraz B. Barrell. A two-dimensional fractionation procedure for radioactive nucleotides. *Journal of molecular biology*, 13(2):373IN1–398IN4, 1965. [cytowanie na str. 14]

[182] F. Sanger, G. M. Air, B. G. Barrell, N. L. Brown, A. R. Coulson, J. Fiddes, C. Hutchison, P. M. Slocombe oraz M. Smith. Nucleotide sequence of bacteriophage φX174 DNA. *Nature*, 265(5596):687–695, 1977. [cytowanie na str. 15]

[183] F. Sanger, S. Nicklen oraz A. R. Coulson. DNA sequencing with chain-terminating inhibitors. *Proceedings of the national academy of sciences*, 74(12):5463–5467, 1977. [cytowanie na str. 15]

[184] E. E. Schadt, S. Turner oraz A. Kasarskis. A window into third-generation sequencing. *Human molecular genetics*, 19(R2):R227–R240, 2010. [cytowanie na str. 23, 24, 26, 27]

[185] M. Schirmer, U. Z. Ijaz, R. D'Amore, N. Hall, W. T. Sloan oraz C. Quince. Insight into biases and sequencing errors for amplicon sequencing with the Illumina MiSeq platform. *Nucleic acids research*, 43(6):e37–e37, 2015. [cytowanie na str. 21]

[186] J. Schröder, H. Schröder, S. J. Puglisi, R. Sinha oraz B. Schmidt. SHREC: a short-read error correction method. *Bioinformatics*, 25(17):2157–2163, 2009. [cytowanie na str. 79]

[187] M. H. Schulz, D. Weese, M. Holtgrewe, V. Dimitrova, S. Niu, K. Reinert oraz H. Richard. Fiona: a parallel and automatic strategy for read error correction. *Bioinformatics*, 30(17):i356–i363, 2014. [cytowanie na str. 61, 63, 73, 83, 92, 93]

[188] A. Shcherbina. FASTQSim: platform-independent data characterization and in silico read generation for NGS datasets. *BMC research notes*, 7(1):533, 2014. [cytowanie na str. 87, 89]

[189] S. Sheikhizadeh and D. de Ridder. ACE: accurate correction of errors using K-mer tries. *Bioinformatics*, 31(19):3216–3218, 2015. [cytowanie na str. 61, 79, 207]

[190] J. T. Simpson and R. Durbin. Efficient de novo assembly of large genomes using compressed data structures. *Genome research*, 22(3):549–556, 2012. [cytowanie na str. 79]

[191] N. Siva. 1000 Genomes project, 2008. [cytowanie na str. 31]

[192] L. Song, L. Florea oraz B. Langmead. Lighter: fast and memory-efficient sequencing error correction without counting. *Genome biology*, 15(11):509, 2014. [cytowanie na str. 61, 69, 75, 76, 90, 92, 93, 208]

[193] W. W. Soon, M. Hariharan oraz M. P. Snyder. High-throughput sequencing for biology and medicine. *Molecular systems biology*, 9(1):640, 2013. [cytowanie na str. 18]

[194] E. Southern. Southern blotting. *Nature protocols*, 1(2):518–526, 2006. [cytowanie na str. 32]

[195] I. Takahashi and J. Marmur. Replacement of thymidylic acid by deoxyuridylic acid in the deoxyribonucleic acid of a transducing phage for Bacillus subtilis. *Nature*, 197(4869):794–795, 1963. [cytowanie na str. 8]

[196] M. T. Tammi, E. Arner, E. Kindlund oraz B. Andersson. Correcting errors in shotgun sequences. *Nucleic acids research*, 31(15):4663–4672, 2003. [cytowanie na str. 80]

[197] H. Thorvaldsdóttir, J. T. Robinson oraz J. P. Mesirov. Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration. *Briefings in bioinformatics*, 14(2):178–192, 2013. [cytowanie na str. 40]

[198] M. A. Varela and W. Amos. Heterogeneous distribution of SNPs in the human genome: microsatellites as predictors of nucleotide diversity and divergence. *Genomics*, 95(3):151–159, 2010. [cytowanie na str. 12]

[199] J. C. Venter, M. D. Adams, E. W. Myers, P. W. Li, R. J. Mural, G. G. Sutton, H. O. Smith, M. Yandell, C. A. Evans, R. A. Holt, et al. The sequence of the human genome. *science*, 291(5507):1304–1351, 2001. [cytowanie na str. 1, 11, 17]

[200] J. D. Watson, F. H. Crick, et al. Molecular structure of nucleic acids. *Nature*, 171 (4356):737–738, 1953. [cytowanie na str. 9]

[201] T. Weissman, E. Ordentlich, G. Seroussi, S. Verdú oraz M. J. Weinberger. Universal discrete denoising: Known channel. *IEEE Transactions on Information Theory*, 51 (1):5–28, 2005. [cytowanie na str. 77]

[202] E. Westhof. The amazing world of bacterial structured rnas. *Genome biology*, 11 (3):108, 2010. [cytowanie na str. 11]

[203] A. Wirawan, R. S. Harris, Y. Liu, B. Schmidt oraz J. Schröder. HECTOR: a parallel multistage homopolymer spectrum based error corrector for 454 sequencing data. *BMC bioinformatics*, 15(1):131, 2014. [cytowanie na str. 81]

[204] C. R. Woese, O. Kandler oraz M. L. Wheelis. Towards a natural system of organisms: proposal for the domains Archaea, Bacteria oraz Eucarya. *Proceedings of the National Academy of Sciences*, 87(12):4576–4579, 1990. [cytowanie na str. 7, 9]

[205] R. Wu. Nucleotide sequence analysis of DNA: I. partial sequence of the cohesive ends of bacteriophage λ and 186 DNA. *Journal of molecular biology*, 51(3):501–521, 1970. [cytowanie na str. 14]

[206] R. Wu and A. Kaiser. Structure and base sequence in the cohesive ends of bacteriophage lambda DNA. *Journal of molecular biology*, 35(3):523–537, 1968. [cytowanie na str. 14]

[207] Y. Xue, Y. Wang oraz H. Shen. Ray Wu, fifth business or father of DNA sequencing? *Protein & cell*, 7(7):467–470, 2016. [cytowanie na str. 14]

[208] X. Yang, K. S. Dorman oraz S. Aluru. Reptile: representative tiling for short read error correction. *Bioinformatics*, 26(20):2526–2533, 2010. [cytowanie na str. 79]

[209] X. Yang, S. P. Chockalingam oraz S. Aluru. A survey of error-correction methods for next-generation sequencing. *Briefings in bioinformatics*, 14(1):56–66, 2012. [cytowanie na str. 27, 29, 61, 62, 64, 65, 67, 88, 89, 93]

[210] X. Yin, Z. Song, K. Dorman oraz A. Ramamoorthy. PREMIER—PRobabilistic error-correction using Markov inference in errored reads. In *2013 IEEE International Symposium on Information Theory*, pages 1626–1630. IEEE, 2013. [cytowanie na str. 61, 66, 68, 71, 90, 93, 131, 202]

[211] J. Zhang, K. Kobert, T. Flouri oraz A. Stamatakis. PEAR: a fast and accurate Illumina Paired-End reAd mergeR. *Bioinformatics*, 30(5):614–620, 2013. [cytowanie na str. 40]

[212] A. Żmieńko, L. Handschuh, M. Góralski oraz M. Figlerowicz. Zastosowanie mikromacierzy DNA w genomice strukturalnej i funkcjonalnej. *Biotechnologia*, 4(83): 39–53, 2008. [cytowanie na str. 32, 33]

[213] J. M. Zook, D. Catoe, J. McDaniel, L. Vang, N. Spies, A. Sidow, Z. Weng, Y. Liu, C. E. Mason, N. Alexander, et al. Extensive sequencing of seven human genomes to characterize benchmark reference materials. *Scientific data*, 3(1):1–26, 2016. [cytowanie na str. 244, 249]

[214] P. W. Zwiernik. Wstęp do ukrytych modeli Markowa i metody Bauma–Welcha. [cytowanie na str. 66]

Dodatki

Dodatek A

Wybrane pseudokody

Na pseudokodach 43–45 przedstawiono przebieg wyznaczania prawidłowych regionów odczytu algorytmu BLESS. Na pseudokodach 46 oraz 47 przedstawiono przebieg korekcji regionu w kierunku $5'$ algorytmu BLESS, z kolei na pseudokodzie 48 przedstawiono przebieg rozszerzania początkowego regionu algorytmu BLESS.

Na pseudokodach 49–53 przedstawiono przebieg wyznaczania prawidłowych regionów odczytu algorytmu RECKONER. Na pseudokodach 54–56 przedstawiono przebieg korekcji regionu w kierunku $5'$ algorytmu RECKONER. Na pseudokodach 57–59 przedstawiono przebieg detekcji i korekcji błędów typu indel w kierunku $5'$ algorytmu RECKONER, z kolei na pseudokodach 60–62 przedstawiono przebieg tworzenia ścieżki korekcji i rozszerzania początkowego regionu w kierunku $5'$ algorytmu RECKONER.

Pseudokod 43 Wyznaczanie prawidłowych regionów odczytu algorytmu BLESS

Wejście: $\mathfrak{r}, \ell = |\mathfrak{r}|, k$
Wyjście: $\Psi_{\text{trust}}, \text{sh\_ns}$

▷ Krok 0.0 — określenie zaufania k-merów

$\psi \leftarrow \varnothing$ ▷ Para indeksów
prev_trust $\leftarrow$ **false**
for $a \leftarrow 0$ **to** $\ell - k - 1$ **do**
  $\kappa \leftarrow \kappa(\mathfrak{r}, a)$
  if $\textsc{Trusted}(\kappa)$ **then**
    if prev_trust = **false then**
      $\textsc{Beg}(\psi) \leftarrow a$
      prev_trust = **true**
    end if
  else
    if prev_trust = **true then**
      $\textsc{End}(\psi) \leftarrow a$
      $\Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \cup \{\psi\}$
      prev_trust $\leftarrow$ **false**
    end if
  end if
end for

▷ Krok 0.1 nie został wykorzystany w algorytmie

▷ Krok 0.2 — eliminacja krótkich regionów prawidłowych

for all $\psi \in \Psi_{\text{trust}}$ **do**
  if $|\psi| = \theta_{\text{MN\_SOLID}}$ **then**
    $\Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \setminus \{\psi\}$
  end if
end for

▷ Krok 0.3 — eliminacja krótkich regionów błędnych

for all $\psi \in \Psi_{\text{trust}}$ **do**
  $\psi_{\text{p}} \leftarrow \textsc{Pred}(\Psi_{\text{trust}}, \psi)$
  if $\psi_{\text{p}} \neq \varnothing \wedge |\textsc{UntrustedReg}(\psi_{\text{p}}, \psi)| < \theta_{\text{MN\_ERR}}$ **then**
    $\Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \setminus \{\psi, \psi_{\text{p}}\}$
    $\psi_{\text{n}} \leftarrow (\textsc{Beg}(\psi_{\text{p}}), \textsc{End}(\psi))$ ▷ Nowy region o większej długości
    $\Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \cup \{\psi_{\text{n}}\}$
  end if
end for

Pseudokod 44 Wyznaczanie prawidłowych regionów odczytu algorytmu BLESS, cd.

▷ Krok 0.4 — uwzględnienie wpływu błędów filtru Blooma na długość regionów

31: **for all** $\psi \in \Psi_{\text{trust}}$ **do**
32: $\quad \psi_{\text{p}} \leftarrow \textsc{Pred}(\Psi_{\text{trust}}, \psi)$
33: $\quad$ **if** $\psi_{\text{p}} \neq \varnothing \wedge k - \theta_{\text{FP}} \leqslant |\textsc{UntrustedReg}(\psi_{\text{p}}, \psi)| < k$ **then**
34: $\quad\quad$ **if** $|\psi| > \theta_{\text{FP}}$ **then**
35: $\quad\quad\quad \textsc{Beg}(\psi_{\text{p}}) \leftarrow \textsc{Beg}(\psi_{\text{p}}) + \theta_{\text{FP}}$
36: $\quad\quad$ **end if**
37: $\quad\quad$ **if** $|\psi_{\text{p}}| > \theta_{\text{FP}}$ **then**
38: $\quad\quad\quad \textsc{End}(\psi_{\text{p}}) \leftarrow \textsc{End}(\psi_{\text{p}}) - \theta_{\text{FP}}$
39: $\quad\quad$ **end if**
40: $\quad$ **end if**
41: **end for**

▷ Krok 0.5 — rozszerzanie krótkiego błędnego regionu do końca odczytu

42: **if** $|\Psi_{\text{trust}}| = 2$ **then**
43: $\quad \psi_{\text{p}} \leftarrow \textsc{MinReg}(\Psi_{\text{trust}})$
44: $\quad \psi \leftarrow \textsc{MaxReg}(\Psi_{\text{trust}})$
45: $\quad$ **if** $\textsc{Beg}(\psi_{\text{p}}) = 0$ **then**
46: $\quad\quad$ **if** $|\textsc{UntrustedReg}(\psi_{\text{p}}, \psi)| < k$ **then**
47: $\quad\quad\quad \Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \setminus \{\psi\}$
48: $\quad\quad$ **end if**
49: $\quad$ **else if** $\textsc{End}(\psi) = \ell - k$ **then**
50: $\quad\quad$ **if** $|\textsc{UntrustedReg}(\psi_{\text{p}}, \psi)| < k$ **then**
51: $\quad\quad\quad$ **if** $|\psi| \geqslant 0,1\ell$ **then**
52: $\quad\quad\quad\quad$ **if** $|\psi_{\text{p}}| < 0,1\ell$ **then**
53: $\quad\quad\quad\quad\quad \Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \setminus \{\psi_{\text{p}}\}$
54: $\quad\quad\quad\quad$ **end if**
55: $\quad\quad\quad$ **end if**
56: $\quad\quad$ **end if**
57: $\quad$ **end if**
58: **end if**

Pseudokod 45 Wyznaczanie prawidłowych regionów odczytu algorytmu BLESS, cd. 2

▷ Krok 0.6 — skracanie prawidłowych regionów kończących się symbolami niskiej jakości
59: **if** $\Psi_{\text{trust}} \neq \emptyset$ **then**
60: **for all** $\psi \in \Psi_{\text{trust}} \setminus \{\textsc{MaxReg}(\Psi_{\text{trust}})\}$ **do** ▷ Skracanie od strony końca 3′
61: **if** $|\psi| > \theta_{\text{MX\_ADJ}} + 1$ **then**
62: **for** $a \leftarrow \textsc{End}(\psi)$ **downto** $\textsc{End}(\psi) - \theta_{\text{MX\_ADJ}} + 1$ **do**
63: **if** $\mathfrak{q}[a + k - 1] < \theta_{\text{Q\_LOW}} \vee \mathfrak{q}[a] < \theta_{\text{Q\_LOW}}$ **then**
64: $\textsc{End}(\psi) \leftarrow \textsc{End}(\psi) - 1$
65: **break**
66: **end if**
67: **end for**
68: **end if**
69: **end for**
70: $\psi \leftarrow \textsc{MaxReg}(\Psi_{\text{trust}})$ ▷ Skracanie ostatniego regionu
71: **if** $\textsc{End}(\psi) < \ell - k$ **then**
72: **if** $|\psi| > \theta_{\text{MX\_ADJ}} + 1$ **then**
73: **for** $a \leftarrow \textsc{End}(\psi)$ **downto** $\textsc{End}(\psi) - \theta_{\text{MX\_ADJ}} + 1$ **do**
74: **if** $\mathfrak{q}[a + k - 1] < \theta_{\text{Q\_LOW}}$ **then**
75: $\textsc{End}(\psi) \leftarrow \textsc{End}(\psi) - 1$
76: **break**
77: **end if**
78: **end for**
79: **end if**
80: **end if**
81: $\psi \leftarrow \textsc{MinReg}(\Psi_{\text{trust}})$ ▷ Skracanie pierwszego regionu
82: **if** $\textsc{Beg}(\psi) > 0$ **then**
83: **if** $|\psi| > \theta_{\text{MX\_ADJ}} + 1$ **then**
84: **for** $a \leftarrow \textsc{Beg}(\psi)$ **to** $\textsc{Beg}(\psi) + \theta_{\text{MX\_ADJ}} - 1$ **do**
85: **if** $\mathfrak{q}[a + k - 1] < \theta_{\text{Q\_LOW}}$ **then**
86: $\textsc{Beg}(\psi) \leftarrow \textsc{Beg}(\psi) + 1$
87: **break**
88: **end if**
89: **end for**
90: **end if**
91: **end if**
92: **end if**

▷ Krok 0.7 — detekcja błędnych regionów krótszych od k
93: sh_ns $\leftarrow$ **false** ▷ Istnieje błędny region krótszy niż k
94: **for all** $\psi \in \Psi_{\text{trust}}$ **do**
95: $\psi_{\text{p}} \leftarrow \textsc{Pred}(\Psi_{\text{trust}}, \psi)$
96: **if** $\psi_{\text{p}} \neq \varnothing \wedge |\textsc{UntrustedReg}(\psi_{\text{p}}, \psi)| < k$ **then**
97: sh_ns $\leftarrow$ **true**
98: **end if**
99: **end for**

Pseudokod 46 Korekcja regionu w kierunku $5'$ algorytmu BLESS

Wejście: $k, \ell = |\mathfrak{r}|, \mathfrak{r}$
Wyjście: $\mathfrak{r}$ poddany modyfikacji

```
1:  procedure CORRECT5'(ψ)
2:      Π ← ∅                                          ▷ Zbiór ścieżek korekcji
3:      κ ← κ(r, END(ψ))
4:      κ* ← κ                                         ▷ Nowa sekwencja k-meru
5:      for all c ∈ Σ do
6:          κ*[0] ← c
7:          if TRUSTED(κ*) then
8:              Π ← ∅                                  ▷ Zbiór par
9:              if κ* ≠ κ then
10:                 Π ← Π ∪ {(END(ψ), c)}
11:             end if
12:             if END(ψ) = 0 then                     ▷ Koniec regionu
13:                 Π ← Π ∪ {Π}
14:             else
15:                 Π ← CONTINUE5'(κ*, (BEG(ψ), END(ψ) − 1), Π)
16:             end if
17:         end if
18:     end for
19:     Π ← EXTEND5'(Π)
20:     Π ← RATE(Π)
21:     if Π ≠ ∅ then
22:         Wstaw do Π element ze zbioru Π
23:         r ← APPLY(r, Π)                            ▷ Ostateczne przyjęcie zmian
24:     end if
25: end procedure

26: function CONTINUE5'(κ, ψ, Π)
27:     κ* ← r[END(ψ)]κ[0, k − 1]
28:     if TRUSTED(κ*) then                            ▷ Modyfikacja nie jest potrzebna
29:         if END(ψ) = 0 then
30:             Π ← Π ∪ Π
31:         else
32:             Π ← CONTINUE5'(κ*, (BEG(ψ), END(ψ) − 1, Π)
33:         end if
```

Pseudokod 47 Korekcja regionu w kierunku $5'$ algorytmu BLESS, cd.

```
34:     else                                                    ▷ Potrzebna modyfikacja
35:         for all c ∈ (Σ \ {κ*[0]}) do
36:             κ*[0] ← c
37:             if Trusted(κ*) then
38:                 Π_n ← Π ∪ (End(ψ), c)
39:                 if End(ψ) = 0 then
40:                     Π ← Π ∪ {Π_n}
41:                 else
42:                     Π ← Continue5'(κ*, (Beg(ψ), End(ψ) − 1, Π_n)
43:                 end if
44:             end if
45:         end for
46:     end if
47:     return Π
48: end function
```

Pseudokod 48 Rozszerzanie początkowego regionu algorytmu BLESS

Wejście: $\mathfrak{r}, \ell = |\mathfrak{r}|, k$

```
1:  function EXTEND5′(Π)
2:      for all Π ∈ Π do
3:          succ ← false
4:          if Π = ∅ then                          ▷ Istnieje ścieżka bez modyfikacji
5:              return ∅                            ▷ Pominięcie wszystkich ścieżek
6:          else
7:              θ_first_mod ← min({a : (a, c) ∈ Π})        ▷ Pierwszy zmodyfikowany symbol
8:              if θ_first_mod ⩾ k − 1 then
9:                  succ ← true
10:             else
11:                 r* ← APPLY(r, Π)
12:                 κ* ← κ_DUMMY(r*, −1)
13:                 θ_ext ← min(θ_MX_E, max(k − θ_first_mod − 1, 0))
14:                 succ ←CONTEXTEND5′(θ_ext, 1, κ*)
15:             end if
16:             if ¬succ then
17:                 Π ← Π \ {Π}
18:             end if
19:         end if
20:     end for
21:     return Π
22: end function

23: function CONTEXTEND5′(θ_ext, a, κ*)
24:     κ* ← ∅κ*                                   ▷ Konkatenacja z nowym symbolem ∅
25:     for all c ∈ Σ do
26:         κ*[0] ← c
27:         if TRUSTED(κ*) then
28:             if a = θ_ext then
29:                 return true
30:             else if CONTEXTEND5′(θ_ext, a + 1, κ*[0, k − 2]) then
31:                 return true
32:             end if
33:         end if
34:     end for
35:     return false
36: end function
```

Pseudokod 49 Wyznaczanie prawidłowych regionów odczytu algorytmu RECKONER

Wejście: $\mathfrak{r}, \ell = |\mathfrak{r}|, k$
Wyjście: $\Psi_{\text{trust}}, \text{sh\_ns}$

▷ Krok 0.0 — określenie zaufania k-merów
$\psi \leftarrow \varnothing$ ▷ Para indeksów
prev_trust $\leftarrow$ **false**
for $a \leftarrow 0$ **to** $\ell - k - 1$ **do**
    $\kappa \leftarrow \kappa(\mathfrak{r}, a)$
    if TRUSTED(κ) **then**
        if prev_trust = **false then**
            BEG(ψ) $\leftarrow a$
            prev_trust = **true**
        end if
    else
        if prev_trust = **true then**
            END(ψ) $\leftarrow a$
            $\Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \cup \{\psi\}$
            prev_trust $\leftarrow$ **false**
        end if
    end if
end for

Pseudokod 50 Wyznaczanie prawidłowych regionów odczytu algorytmu RECKONER, cd.

▷ Krok 0.1 — skracanie prawidłowych regionów zawierających symbole niskiej jakości
if $|\Psi_{\text{trust}}| > 1$ **then**
  short_dist $\leftarrow$ **false**
  for all $\psi \in \Psi_{\text{trust}} \setminus \{\textsc{MinReg}(\Psi_{\text{trust}})\}$ **do**
    $\Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \setminus \{\psi\}$
    $\psi_{\text{p}} \leftarrow \textsc{Pred}(\Psi_{\text{trust}}, \psi)$
    if $\psi_{\text{p}} \neq \varnothing \wedge |\textsc{UntrustedReg}(\psi_{\text{p}}, \psi)| < k$ **then**
      short_dist $\leftarrow$ **true**
    end if
  end for
  if short_dist **then**
    for all $\psi \in \Psi_{\text{trust}}$ **do**
      num_low $\leftarrow 0$
      for $a \leftarrow \textsc{Beg}(\psi)$ **to** $\textsc{End}(\psi) + k - 1$ **do**
        if $\mathfrak{q}[a] < \theta_{\text{Q\_LOW}}$ **then**
          num_low $\leftarrow$ num_low $+ 1$
          if num_low $= 1$ **then**
            if $a \geqslant \textsc{Beg}(\psi) + k$ **then**
              $\Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \cup \{(\textsc{Beg}(\psi), a - k)\}$
            end if
          else
            if $a -$ prev_low $> k$ **then**
              $\Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \cup \{(\text{prev\_low} + 1, a - k)\}$
            end if
          end if
          prev_low $\leftarrow a$
        end if
      end for
      if num_low > 0 **then**
        if $\textsc{End}(\psi) \geqslant$ prev_low $+ k$ **then**
          $\Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \cup \{(\text{prev\_low} + k, \textsc{End}(\psi))\}$
        end if
      else
        $\Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \cup \{\psi\}$ ▷ Ponowne wstawienie ψ
      end if
    end for
  end if

Pseudokod 51 Wyznaczanie prawidłowych regionów odczytu algorytmu RECKONER, cd. 2

54: **else if** $|\Psi_{\text{trust}}| = 1 \wedge (\textsc{Beg}(\textsc{MinReg}(\Psi_{\text{trust}})) \neq 0 \vee \textsc{End}(\textsc{MaxReg}(\Psi_{\text{trust}})) \neq \ell - k)$ **then**
55: $\psi \leftarrow$ jedyny element zbioru Ψ_{trust}
56: $\Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \setminus \{\psi\}$ ▷ Usunięcie jedynego elementu
57: num_low $\leftarrow 0$
58: **for** $a \leftarrow \textsc{Beg}(\psi)$ **to** $\textsc{End}(\psi) + k - 1$ **do**
59: **if** $\mathfrak{q}[a] < \theta_{\text{Q\_LOW}}$ **then**
60: num_low $\leftarrow$ num_low $+ 1$
61: **if** num_low $= 1$ **then**
62: **if** $a - \textsc{Beg}(\psi) \geqslant k + \theta_{\text{MN\_SOLID}} - 1$ **then**
63: $\Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \cup \{(\textsc{Beg}(\psi), a - k)\}$
64: **end if**
65: prev_low $\leftarrow a$
66: **else**
67: **if** $a -$ prev_low $\geqslant k + \theta_{\text{MN\_SOLID}}$ **then**
68: $\Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \cup \{(\text{prev\_low} + 1, a - k)\}$
69: **end if**
70: prev_low $\leftarrow a$
71: **end if**
72: **end if**
73: **end for**
74: **if** $\Psi_{\text{trust}} = \emptyset$ **then**
75: $\Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \cup \{\psi\}$ ▷ Ponowne wstawienie ψ
76: **end if**
77: **end if**

▷ Krok 0.2 — eliminacja krótkich regionów prawidłowych
78: **for all** $\psi \in \Psi_{\text{trust}}$ **do**
79: **if** $|\psi| = \theta_{\text{MN\_SOLID}}$ **then**
80: $\Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \setminus \{\psi\}$
81: **end if**
82: **end for**

▷ Krok 0.3 — eliminacja krótkich regionów błędnych
83: **for all** $\psi \in \Psi_{\text{trust}}$ **do**
84: $\psi_{\text{p}} \leftarrow \textsc{Pred}(\Psi_{\text{trust}}, \psi)$
85: **if** $\psi_{\text{p}} \neq \varnothing \wedge |\textsc{UntrustedReg}(\psi_{\text{p}}, \psi)| < \theta_{\text{MN\_ERR}}$ **then**
86: $\Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \setminus \{\psi, \psi_{\text{p}}\}$
87: $\psi_{\text{n}} \leftarrow (\textsc{Beg}(\psi_{\text{p}}), \textsc{End}(\psi))$ ▷ Nowy region o większej długości
88: $\Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \cup \{\psi_{\text{n}}\}$
89: **end if**
90: **end for**

▷ Krok 0.4 nie został wykorzystany w algorytmie

Pseudokod 52 Wyznaczanie prawidłowych regionów odczytu algorytmu RECKONER, cd. 3

▷ Krok 0.5 — rozszerzanie krótkiego błędnego regionu do końca odczytu
if $|\Psi_{\text{trust}}| = 2$ **then**
  $\psi_{\text{p}} \leftarrow \textsc{MinReg}(\Psi_{\text{trust}})$
  $\psi \leftarrow \textsc{MaxReg}(\Psi_{\text{trust}})$
  if $\textsc{Beg}(\psi_{\text{p}}) = 0$ **then**
    if $|\textsc{UntrustedReg}(\psi_{\text{p}}, \psi)| < k$ **then**
      $\Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \setminus \{\psi\}$
    end if
  else if $\textsc{End}(\psi) = \ell - k$ **then**
    if $|\textsc{Untrusted}(\psi_{\text{p}}, \psi)| < k$ **then**
      if $|\psi| \geqslant 0,1\ell$ **then**
        if $|\psi_{\text{p}}| < 0,1\ell$ **then**
          $\Psi_{\text{trust}} \leftarrow \Psi_{\text{trust}} \setminus \{\psi_{\text{p}}\}$
        end if
      end if
    end if
  end if
end if

▷ Krok 0.6 — skracanie prawidłowych regionów kończących się symbolami niskiej jakości
if $\Psi_{\text{trust}} \neq \emptyset$ **then**
  for all $\psi \in \Psi_{\text{trust}} \setminus \{\textsc{MaxReg}(\Psi_{\text{trust}})\}$ **do**▷ Skracanie od strony końca $3'$
    max_adj $\leftarrow \min(\theta_{\text{MX\_ADJ}}, |\psi| - 1)$
    for $a \leftarrow \textsc{End}(\psi)$ **downto** $\textsc{End}(\psi) - \theta_{\text{MX\_ADJ}} + 1$ **do**
      if $\mathfrak{q}[a + k - 1] < \theta_{\text{Q\_LOW}} \vee \mathfrak{q}[a] < \theta_{\text{Q\_LOW}}$ **then**
        $\textsc{End}(\psi) \leftarrow \textsc{End}(\psi) - 1$
        break
      end if
    end for
  end for
  for all $\psi \in \Psi_{\text{trust}} \setminus \{\textsc{MinReg}(\Psi_{\text{trust}})\}$ **do**▷ Skracanie od strony końca $5'$
    max_adj $\leftarrow \min(\theta_{\text{MX\_ADJ}}, |\psi| - 1)$
    for $a \leftarrow \textsc{Beg}(\psi)$ **to** $\textsc{Beg}(\psi) + \theta_{\text{MX\_ADJ}} - 1$ **do**
      if $\mathfrak{q}[a + k - 1] < \theta_{\text{Q\_LOW}} \vee \mathfrak{q}[a] < \theta_{\text{Q\_LOW}}$ **then**
        $\textsc{Beg}(\psi) \leftarrow \textsc{Beg}(\psi) + 1$
        break
      end if
    end for
  end for

Pseudokod 53 Wyznaczanie prawidłowych regionów odczytu algorytmu RECKONER, cd. 4

```
127:     ψ ← MaxReg(Ψ_trust)                          ▷ Skracanie ostatniego regionu
128:     max_adj ← min(θ_MX_ADJ, |ψ| − 1)
129:     for a ← End(ψ) downto End(ψ) − θ_MX_ADJ + 1 do
130:         if q[a + k − 1] < θ_Q_LOW then
131:             End(ψ) ← End(ψ) − 1
132:             break
133:         end if
134:     end for
135:     ψ ← MinReg(Ψ_trust)                          ▷ Skracanie pierwszego regionu
136:     max_adj ← min(θ_MX_ADJ, |ψ| − 1)
137:     for a ← Beg(ψ) to Beg(ψ) + θ_MX_ADJ − 1 do
138:         if q[a + k − 1] < θ_Q_LOW then
139:             Beg(ψ) ← Beg(ψ) + 1
140:             break
141:         end if
142:     end for
143: end if

                                  ▷ Krok 0.7 — detekcja błędnych regionów krótszych od k
144: sh_ns ← false                                    ▷ Istnieje błędny region krótszy niż k
145: for all ψ ∈ Ψ_trust do
146:     ψ_p ← Pred(Ψ_trust, ψ)
147:     if ψ_p ≠ ∅ ∧ |Untrusted(ψ_p, ψ)| < k − 1 then
148:         sh_ns ← true
149:     end if
150: end for
```

Pseudokod 54 Korekcja regionu w kierunku 5′ algorytmu RECKONER

Wejście: $k, \ell = |\mathfrak{r}|, \mathfrak{r}, \mathfrak{q}$
Globalne: $\mathfrak{s}, \theta_{\text{reg\_beg}}$
Globalne: $\theta_{\text{mx\_ind}}, \theta_{\text{mx\_untrst}}$ ▷ Liczniki dwukierunkowe (dekrementowane przy zagłębianiu i inkrementowane przy powrotach algorytmu)
Globalne: $\theta_{\text{mx\_chck}}$ ▷ Licznik jednokierunkowy (tylko dekrementowany)

1: **function** CORRECT5′(ψ)
2: $\mathfrak{s} \leftarrow ()$ ▷ Pusty ciąg korekcji
3: $\mathbf{\Pi} \leftarrow \emptyset$
4: $\theta_{\text{reg\_beg}} \leftarrow \text{BEG}(\psi)$
▷ Wyznaczenie limitów
5: $\theta_{\text{mx\_ind}} \leftarrow \min(\theta_{\text{MX\_IND}}, \lceil \theta_{\text{MX\_IND\_RATE}} \cdot |\psi| \rceil)$
6: $\theta_{\text{mx\_untrst}} \leftarrow \lceil \theta_{\text{MX\_NON\_UNTRST}} \cdot |\psi| \rceil$
7: $\theta_{\text{mx\_chck}} \leftarrow \theta_{\text{MX\_CHCK}}$
8: $i \leftarrow 0$ ▷ Pozycja ciągu korekcji
9: $\kappa^* \leftarrow \kappa(\mathfrak{r}, \text{END}(\psi))$ ▷ Nowa sekwencja k-meru
10: **for all** $c \in \Sigma$ **do**
11: **if** $\theta_{\text{mx\_chck}} = 0$ **then**
12: **return** $\mathbf{\Pi}$
13: **end if**
14: $\theta_{\text{mx\_chck}} \leftarrow \theta_{\text{mx\_chck}} - 1$
15: $\kappa^*[0] \leftarrow c$
16: **if** TRUSTED(κ^*) **then**
17: $\mathfrak{s}_i \leftarrow (\eta(\kappa^*), c, \textbf{false}, \textbf{false}, \ominus)$
18: **if** END(ψ) $= 0$ **then** ▷ Koniec regionu
19: $\pi \leftarrow \text{CREATEPATHEXTEND5'}(i, \mathfrak{s}, \psi)$
20: **if** $\pi \neq \varnothing$ **then**
21: $\mathbf{\Pi} \leftarrow \mathbf{\Pi} \cup \{\pi\}$
22: **end if**
23: **else**
24: $\mathbf{\Pi} \leftarrow \text{CONTINUE5'}(\kappa^*, i+1, (\text{BEG}(\psi), \text{END}(\psi) - 1), \mathbf{\Pi})$
25: **end if**
26: **end if**
27: **end for**
28: $\mathbf{\Pi} \leftarrow \text{CORRECT5'INDEL}(\kappa^*, i, \psi, \mathbf{\Pi})$
29: **return** $\mathbf{\Pi}$
30: **end function**

Pseudokod 55 Korekcja regionu w kierunku $5'$ algorytmu RECKONER, cd.

```
31: function CONTINUE5′(κ, i, ψ, Π)
32:     κ* ← 𝔯[END(ψ)]κ[0, k − 1]
33:     low ← 𝔮[END(ψ)] < θ_Q_LOW                ▷ Czy symbol jest niskiej jakości

                ▷ Sytuacja 1 — k-mer jest zaufany i symbol nie jest niskiej jakości
34:     if ¬low ∧ TRUSTED(κ*) then               ▷ Niczego nie modyfikujemy
35:         𝔰_i ← (η(κ*), κ*[0], false, false, ⊖)
36:         if END(ψ) = 0 then
37:             if |Π| > θ_MX_PTH then
38:                 return Π
39:             end if
40:             π ← CREATEPATHEXTEND5'(i, 𝔰, ψ)
41:             if π ≠ ∅ then
42:                 Π ← Π ∪ {π}
43:             end if
44:         else
45:             Π ← CONTINUE5'(κ*, i + 1, (BEG(ψ), END(ψ) − 1), Π)
46:         end if

            ▷ Sytuacja 2 — k-mer nie jest zaufany lub symbol jest niskiej jakości
47:     else
48:         solid ← false                        ▷ Czy znaleziono zaufany k-mer
49:         Σ′ ← Σ
50:         if ¬low then
51:             Σ′ ← Σ \ {κ*[0]}                 ▷ Nie ma sensu sprawdzać oryginalnego
52:         end if
53:         for all c ∈ Σ′ do
54:             if θ_mx_chck = 0 then
55:                 return Π
56:             end if
57:             θ_mx_chck ← θ_mx_chck − 1
58:             κ*[0] ← c
```

Pseudokod 56 Korekcja regionu w kierunku 5′ algorytmu RECKONER, cd. 2

▷ Sytuacja 2.1 — znaleziono zaufany k-mer

```
59:            if TRUSTED(κ*) then
60:                solid ← true
61:                𝔰_i ← (η(κ*), c, false, false, ⊖)
62:                if END(ψ) = 0 then
63:                    if |Π| > θ_MX_PTH then
64:                        return Π
65:                    end if
66:                    π ← CREATEPATHEXTEND5'(i, 𝔰, ψ)
67:                    if π = ∅ then
68:                        CHECK5'INDEL(i, ψ, true)
69:                    else
70:                        Π ← Π ∪ {π}
71:                    end if
72:                else
73:                    Π ← CONTINUE5'(κ*, i + 1, (BEG(ψ), END(ψ) − 1), Π)
74:                    CHECK5'INDEL(i, ψ, false)
75:                end if
76:                Π ← CORRECT5'INDEL(κ*, i, ψ, Π)
77:            end if
78:        end for
```

▷ Sytuacja 2.2 — nie znaleziono zaufanego k-mer

```
79:        if ¬solid then
80:            if θ_mx_untrst > 0 then
81:                κ*[0] ← 𝔱[END(ψ)]
82:                𝔰_i ← (0, κ*[0], false, false, ⊖)
83:                if END(ψ) = 0 then
84:                    if |Π| > θ_MX_PTH then
85:                        return Π
86:                    end if
87:                    π ← CREATEPATHEXTEND5'(i, 𝔰, ψ)
88:                    if π = ∅ then
89:                        CHECK5'INDEL(i, ψ, true)
90:                    else
91:                        Π ← Π ∪ {π}
92:                    end if
93:                else
94:                    Π ← CONTINUE5'(κ*, i + 1, (BEG(ψ), END(ψ) − 1), Π)
95:                    CHECK5'INDEL(i, ψ, false)
96:                end if
97:            end if
98:        end if
99:    end if
100:    return Π
101: end function
```

Pseudokod 57 Detekcja i korekcja błędów typu indel w kierunku $5'$ algorytmu RECKONER

Globalne: $\mathfrak{s}, \theta_{\text{mx\_ind}}$

```
1:  procedure CHECK5′INDEL(i, ψ, last)
2:      if θ_mx_ind = 0 then
3:          return
4:      end if
5:      θ_n_exp_ind ← Liczba elementów s_0, s_1, ..., s_i, dla których exp_ind = true
6:      curr_pos ← END(ψ)                          ▷ Pozycja aktualnego symbolu

        ▷ Sytuacja 1 — koniec krótkiego regionu, sprawdzenie wszystkich pozycji
7:      if last ∧ i + 1 < θ_MX_CHCK_IND_UNTRST then
8:          mods ← 0
9:          indel_pos ← 0
10:         for j ← i downto 0 do
11:             if INS(s_j) ∨ DEL(s_j) then
12:                 break          ▷ Przerwanie, ale nie wykluczenie błędu indel
13:             end if
14:             if MOD(s_j) ≠ t[curr_pos + i − j] then          ▷ Była modyfikacja
15:                 mods ← mods + 1
16:                 indel_pos ← j
17:             end if
18:         end for
19:         if mods ⩾ ⌈(θ_MX_CHCK_IND_UNTRST · (i+1)) / θ_MX_CHCK_IND_POS⌉ then
20:             for j ← i downto indel_pos do
21:                 EXP_INDEL(s_j) ← true                  ▷ Potencjalny błąd indel
22:             end for
23:         end if
24:         return
25:     end if

                              ▷ Sytuacja 2 — sprawdzanie ograniczonej liczby pozycji
26:     if θ_n_exp_ind > 0 then
27:         return
28:     end if
```

Pseudokod 58 Detekcja i korekcja błędów typu indel w kierunku $5'$ algorytmu RECKONER, cd.

```
29:     if i + 1 ⩾ θ_MX_CHCK_IND_UNTRST then
30:         max_check ← min(i, θ_MX_CHCK_IND_POS)        ▷ Sprawdzanych pozycji
31:         mods ← 0
32:         indel_pos ← 0
33:         for j ← i downto i − max_check do
34:             if INS(𝔰_j) ∨ DEL(𝔰_j) then
35:                 break          ▷ Przerwanie, ale nie wykluczenie błędu indel
36:             end if
37:             if MOD(𝔰_j) ≠ 𝔯[curr_pos + i − j] then
38:                 mods ← mods + 1
39:                 indel_pos ← j
40:             end if
41:         end for
42:         if mods ⩾ θ_MX_CHCK_IND_UNTRST then
43:             EXP_INDEL(𝔰_indel_pos) ← true              ▷ Potencjalny błąd indel
44:         end if
45:     end if
46:     return
47: end procedure

48: function CORRECT5′INDEL(κ, i, ψ, Π)
49:     if ¬EXP_INDEL(𝔰_i) then
50:         return Π
51:     end if
52:     EXP_INDEL(𝔰_i) ← false
53:     if θ_mx_ind = 0 then
54:         return Π
55:     end if
56:     if i > 0 ∧ (INS(𝔰_{i−1}) ∨ DEL(𝔰_{i−1})) then
57:         return Π
58:     end if
59:     θ_mx_ind ← θ_mx_ind − 1
            ▷ Poniższe dwa zakresy obejmują tylko pozycje ciągu, które istnieją
60:     was_del ← wśród 𝔰_{i−θ_MN_IND_DIST}, ..., 𝔰_{i−2}, 𝔰_{i−1} jest taki, że del = true
61:     was_ins ← wśród 𝔰_{i−θ_MN_IND_DIST}, ..., 𝔰_{i−2}, 𝔰_{i−1} jest taki, że ins = true
```

Pseudokod 59 Detekcja i korekcja błędów typu indel w kierunku $5'$ algorytmu RECKONER, cd. 2

```
62:     if ¬was_del then
                                  ▷ Nie sprawdzamy zaufania k-meru ani liczebności
63:         s_i ← (∅, r[End(ψ)], true, false, ⊖)
64:         if End(ψ) = 0 then
65:             if |Π| > θ_MX_PTH then
66:                 s_i ← (∅, ∅, false, false, ⊖)
67:                 return Π
68:             end if
69:             π ← CreatePathExtend5'(i, s, ψ)
70:             if π ≠ ∅ then
71:                 Π ← Π ∪ {π}
72:             end if
73:         else
74:             if θ_mx_chck = 0 then
75:                 s_i ← (∅, ∅, false, false, ⊖)
76:                 return Π
77:             end if
78:             θ_mx_chck ← θ_mx_chck − 1
79:             κ* ← κ[1, k − 1]∅
80:             Π ← Continue5'(κ*, i + 1, (Beg(ψ), End(ψ) − 1), Π)
81:             s_i ← (∅, ∅, false, false, ⊖)
82:         end if
83:     end if
84:     if ¬was_ins then
85:         κ* ← κ                                  ▷ Nowa sekwencja k-meru
86:         for all c ∈ Σ do
87:             if θ_mx_chck = 0 then
88:                 s_i ← (∅, ∅, false, false, ⊖)
89:                 return Π
90:             end if
91:             θ_mx_chck ← θ_mx_chck − 1
92:             κ*[k − 1] ← c
93:             if Trusted(κ*) then
94:                 s_i ← (η(κ*), c, false, true, ⊖)
95:                 Π ← Continue5'(κ*, i + 1, ψ, Π)        ▷ Nie modyfikujemy ψ
96:             end if
97:         end for
98:         s_i ← (∅, ∅, false, false, ⊖)
99:     end if
100:    θ_mx_ind ← θ_mx_ind + 1
101:    return Π
102: end function
```

Pseudokod 60 Tworzenie ścieżki i rozszerzanie początkowego regionu w kierunku $5'$ algorytmu RECKONER

Wejście: $\theta_{\text{reg\_beg}}, k, \mathfrak{q}, \mathfrak{r}$

```
1:  function CreatePathExtend5′(i, 𝔰, ψ)
2:      pos ← θ_reg_beg
3:      π ← ()                                              ▷ Pusta ścieżka
4:      PathQual(π) ← 0                  ▷ Suma liczebności ważonych k-merów
5:      PathProb(π) ← 1                  ▷ Iloczyn prawdopodobieństw błędów
6:      ℓ_Δ ← 0                                             ▷ Zmiana długości odczytu
7:      for j ← 0 to i do
8:          if Ins(𝔰_j) then
9:              π ← π(pos, ∅, true, false)                  ▷ Dołączenie krotki do ciągu
10:             ℓ_Δ ← ℓ_Δ − 1
11:             PathProb(π) ← PathProb(π) · θ_IND_PROB
12:         else if Del(𝔰_j) then
13:             π ← π(pos + 1, Mod(𝔰_j), false, true)
14:             ℓ_Δ ← ℓ_Δ + 1
15:             PathProb(π) ← PathProb(π) · θ_IND_PROB
16:         else if Mod(𝔰_j) ≠ 𝔯[pos] then                  ▷ Substytucja
17:             π ← π(pos, Mod(𝔰_j), false, false)
18:             PathProb(π) ← PathProb(π) · p(𝔮[pos])
19:         end if
20:         if ¬Ins(𝔰_j) then
21:             PathQual(π) ← PathQual(π) + Qual(𝔰_j · θ_COV_WEIGHT)
22:         end if
23:         if ¬Del(𝔰_j) then
24:             pos ← pos − 1
25:         end if
26:     end for
27:     PathWghtSum(π) ← (i + 1 − ℓ_Δ) · θ_COV_WEIGHT
28:     return Extend5′(π, ℓ_Δ, ψ)
29: end function
```

Pseudokod 61 Tworzenie ścieżki i rozszerzanie początkowego regionu w kierunku $5'$ algorytmu RECKONER, cd.

30: **function** EXTEND5′(π, ℓ_Δ)
31: $n_{\text{mod}} \leftarrow |\pi|$
32: **if** $n_{\text{mod}} = 0$ **then**
33: **return** π
34: **end if**
▷ Wyznaczenie ostatniej zmodyfikowanej pozycji w regionie
35: $\theta_{\text{first\_mod}} \leftarrow \text{POS}(\pi_{n_{\text{mod}}-1})$
36: **if** $\theta_{\text{last\_mod}} \geqslant k - 1$ **then**
37: **return** π
38: **else** ▷ Wyznaczenie liczby rozszerzających pozycji
39: **if** $\theta_{\text{last\_mod}} \geqslant k - \theta_{\text{MX\_E}} - 1$ **then**
40: $\theta_{\text{ext}} \leftarrow k - \theta_{\text{last\_mod}} - 1$
41: **else**
42: $\theta_{\text{ext}} \leftarrow \theta_{\text{MX\_E}}$
43: **end if**
44: $\mathfrak{r}^* \leftarrow \text{APPLY}(\mathfrak{r}, \pi)$
45: $\ell^* \leftarrow |\mathfrak{r}^*|$
46: $\kappa^* \leftarrow \varnothing\kappa_{\text{DUMMY}}(\mathfrak{r}^*, -1)$
47: succ $\leftarrow$ **false**
48: $\theta_{\text{mx\_untrst}} \leftarrow \lceil \theta_{\text{MX\_NON\_UNTRST}} \cdot \theta_{\text{ext}} \rceil$
49: $\eta_{\text{max\_ext}} \leftarrow 0$ ▷ Maksymalna liczebność pierwszego rozszerzającego k-meru
50: **for all** $c \in \Sigma$ **do**
51: $\kappa^*[0] \leftarrow c$
52: solid $\leftarrow$ TRUSTED(κ^*)
53: **if** solid $\vee\ \theta_{\text{mx\_untrst}} > 0$ **then**
54: $\eta_{\text{max\_ext}} \leftarrow \max(\eta_{\text{max\_ext}}, \eta(\kappa^*))$
55: **if** $\theta_{\text{ext}} = 1$ **then**
56: **if** solid **then**
57: succ $\leftarrow$ **true**
58: **end if**
59: **else if** $\neg$succ **then**
60: **if** solid **then**
61: succ $\leftarrow$ CONTEXTEND5'($\theta_{\text{ext}}, 1, \theta_{\text{mx\_untrst}}, \kappa^*$)
62: **else**
63: succ $\leftarrow$ CONTEXTEND5'($\theta_{\text{ext}}, 1, \theta_{\text{mx\_untrst}} - 1, \kappa^*$)
64: **end if**
65: **end if**
66: **end if**
67: **end for**

Pseudokod 62 Tworzenie ścieżki i rozszerzanie początkowego regionu w kierunku $5'$ algorytmu RECKONER, cd. 2

```
68:         if succ then
69:             PathQual(π) ← PathQual(π) + η_max_ext · θ_EXT_WEIGHT
70:             PathWghtSum(π) ← PathWghtSum(π) + θ_EXT_WEIGHT
71:             return π
72:         else
73:             return ∅
74:         end if
75:     end if
76: end function

77: function ContExtend5′(θ_ext, a, θ_mx_untrst, κ*)
78:     κ* ← ∅κ*[0, k − 2]
79:     for all c ∈ Σ do
80:         κ*[0] ← c
81:         solid ← Trusted(κ*)
82:         if solid ∨ θ_mx_untrst > 0 then
83:             if a + 1 = θ_ext then
84:                 return true
85:             else
86:                 if solid then
87:                     succ ← ContExtend5′(θ_ext, a + 1, θ_mx_untrst, κ*[0, k − 2])
88:                 else
89:                     succ ← ContExtend5′(θ_ext, a+1, θ_mx_untrst −1, κ*[0, k−2])
90:                 end if
91:                 if succ then
92:                     return true
93:                 end if
94:             end if
95:         end if
96:     end for
97:     return false
98: end function
```

Dodatek B

Szczegółowe informacje o eksperymentach

B.1 Algorytmy i otoczenie sprzętowe

B.1.1 Wersje algorytmów

W tabeli B.1 przedstawiono testowane algorytmy korekcji. W tabeli B.2 przedstawiono pomocnicze narzędzia i algorytmy wykorzystane do przeprowadzenia eksperymentów. W jej pierwszej części zostały zamieszczone algorytmy, które podlegały testom wydajnościowym.

B.1.2 Serwer obliczeniowy

W tabeli B.3 przedstawiono specyfikację serwera wykorzystanego w eksperymentach.

B.2 Wywołanie algorytmów

W niniejszym podrozdziale przedstawiono sposób wywołania wszystkich narzędzi wykorzystanych w eksperymentach.

B.2.1 Algorytmy korekcji

Znaczenie poszczególnych wartości:

- `<alfa>` — prawdopodobieństwo p_α wyboru k-meru (Lighter),
- `<fastq_we>`, `<fastq_we1>`, `<fastq_we2>` — pliki wejściowe FASTQ (odpowiednio: plik odczytów niesparowanych, pliki odczytów sparowanych),

- `<fastq_wy>` — plik wyjściowy FASTQ (plik odczytów niesparowanych),
- `<fastq_wy_pref>` — prefiks plików wyjściowych (BLESS),
- `<genom>` — plik genomu referencyjnego FASTA,
- `<k>` — długość oligomeru,
- `<kmerow>` — liczba k-merów w odczytach wejściowych,
- `<nazwa>` — nazwa eksperymentu,
- `<obciecie>` — próg obcięcia,
- `<pamiec>` — limit pamięci (250 — Karect),
- `<ploidia>` — ploidia (`haploid` albo `diploid`, odpowiednio dla haploidalnych oraz diploidalnych — Karect),
- `<sam_we>` — plik wejściowy SAM (uprzednio zmapowane odczyty — SAMDUDE),
- `<sam_wy>` — plik wyjściowy SAM (SAMDUDE),
- `<watkow>` — liczba wątków (48 lub mniej w przypadku analizy skalowalności).

W przypadku korekcji odczytów symulowanych (niesparowanych) implementacje, jeśli nie zaznaczono inaczej, uruchamiano analogicznie, przekazując tylko jeden plik wejściowy.

RECKONER

Odczyty symulowane oraz rzeczywiste dla detekcji wariantów:
`reckoner -kmerlength <k> -threads <watkow> <fastq_we1> <fastq_we2>`
Pozostałe odczyty rzeczywiste:
`reckoner -longkmer -kmerlength <k> -threads <watkow> <fastq_we1> <fastq_we2>`

Musket

`musket -k <k> <kmerow> -p <watkow> -inorder -omulti <nazwa> <fastq_we1> <fastq_we2>`

RACER

`RACER_Linux_parallel <fastq_we> <fastq_wy> <genom>`

BLESS

Odczyty rzeczywiste:
`bless -kmerlength <k> -read1 <fastq_we1> -read2 <fastq_we2> -smpthread <watkow> -prefix <fastq_wy_pref> -gzip -max_mem 8`
Odczyty symulowane:
`bless -kmerlength <k> -read1 <fastq_we1> -read2 <fastq_we2> -smpthread <watkow> -prefix <fastq_wy_pref> -gzip -max_mem 8 -notrim`

Fiona

```
fiona -nt <watkow> -g <genom> <fastq_we> <fastq_wy>
```

Blue

```
mono Tessel.exe -k <k> -g <genom> -t <watkow> -f fastq -tmp . <nazwa> <fastq_we1> <fastq_we2>
mono GenerateMerPairs.exe -t <watkow> <nazwa>.cbt <fastq_we1> <fastq_we2>
mono Blue.exe -m <obciecie> -t <watkow> -r o -o . -f fastq <nazwa>.cbt <fastq_we1> <fastq_we2>
```

Lighter

```
lighter -r <fastq_we1> -r <fastq_we2> -k <k> <genom> <alfa> -t <watkow>
```

BFC

```
bfc -s <genom> -t <watkow> <fastq_we> > <fastq_wy>
```

Karect

```
karect -correct -threads=<watkow>-memory=<pamiec> -celltype=<ploidia> -matchtype=hamming -inputfile=<fastq_we1> -inputfile=<fastq_we1>
```

SAMDUDE

```
python3 run_denoiser.py <sam_we> <sam_wy>
```

B.2.2 Asemblacja i mapowanie

Znaczenie poszczególnych wartości:

- `<fasta>` — wyjściowy plik kontigów,
- `<fastq_we1>`, `<fastq_we2>` — pliki wejściowe FASTQ (pliki odczytów sparowanych),

- <nazwa> — nazwa eksperymentu,
- <pamiec> — limit pamięci (250 — Minia),
- <ref> — plik genomu referencyjnego,
- <sam> — wyjściowy plik zmapowanych odczytów,
- <watkow> — liczba wątków (48).

BWA — jednokrotne indeksowanie genomu

```
bwa index <ref>
```

BWA — mapowanie

```
bwa mem -t <watkow> -M <ref> <fastq_we1> <fastq_we2> > <sam>
```

Minia

```
minia -in <fastq_we1> -in <fastq_we2> -max-memory <pamiec> -out <fasta>
```

Quast

```
python quast.py <fasta> -o <nazwa> -R <ref> --threads <watkow>
```

B.2.3 Detekcja wariantów

Znaczenie poszczególnych wartości:

- <bam> — posortowany plik zmapowanych odczytów,
- <bam_dir> — katalog pomocniczy (Strelka),
- <bam_raw> — nieposortowany plik zmapowanych odczytów,
- <bed> — plik regionów wysokiej pewności (BED),
- <fastq_we1>, <fastq_we2> — pliki wejściowe FASTQ (pliki odczytów sparowanych),
- <gt_vcf> — plik prawdy podstawowej,
- <nazwa> — nazwa eksperymentu,
- <ref> — genom referencyjny,
- <sam> — wyjściowy plik zmapowanych odczytów,
- <watkow> — liczba wątków (48).

BWA — jednokrotne indeksowanie genomu

```
bwa index <ref>
```

Detekcja wariantów

```
bwa mem -t <watkow> -M <ref> <fastq_we1> <fastq_we2> > <sam>
samtools view -@ <watkow> -b <sam> > <bam_raw>
samtools sort -@ <watkow> <bam_raw> > <bam>
samtools index <bam>
python configureStrelkaGermlineWorkflow.py --bam <bam> --ref <ref> --runDir
<bam_dir> python <bam_dir>/runWorkflow.py -m local -j <watkow>
```

hap.py

```
python hap.py <gt_vcf> <nazwa>.vcf.gz -o <nazwa>.happy.nist -r <ref>
```

hap.py — z zestawem regionów wysokiej pewności

```
python hap.py <gt_vcf> <nazwa>.vcf.gz -f <bed> -o <nazwa>.happy.nist
-r <ref>
```

Syndip

```
run-flt -o <nazwa> <nazwa>.vcf run-eval -g 38 <nazwa>.flt.vcf.gz | sh
```

B.2.4 Pozostałe algorytmy i narzędzia

Znaczenie poszczególnych wartości:

- <dlugosc> — długość odczytów,
- <fastq_we> — wzorcowy dla profilu plik odczytów,
- <fastq_wy> — plik odczytów symulowanych,
- <fastq_wy1>, <fastq_wy2> — pliki odczytów sparowanych FASTQ uzyskane z odczytów pliku <sam>,
- <katalog> — katalog, którego wielkość podlegała pomiarom,
- <nazwa> — nazwa profilu,
- <liczba_odczytow> — liczba odczytów do wygenerowania,
- <plik> — plik FASTQ lub SAM, wykorzystany w algorytmie poddawanym pomiarowi czasu obliczeń,

- <polecenie_parametry> — polecenie uruchamiające narzędzie podlegającą pomiarom, wraz z jego własnymi parametrami,
- <ref> — genom referencyjny,
- <sam> — plik SAM, z którego następuje wyodrębnienie odczytów,
- <wy> — plik wyjściowy,
- <wylaczenia> — plik tekstowy zawierający listę nazw plików niepodlegających analizie (wszystkie pliki wejściowe i wyjściowe implementacji algorytmu).

ART — wyznaczanie profili błędów

Narzędzie wykorzystywane do generacji odczytów symulowanych. `art_profiler_illumina <fastq_we> <nazwa> fastq`

ART — symulacja odczytów

```
art_illumina -sam -1 <nazwa> -l <dlugosc> -i <ref> -c <liczba_odczytow>
-o <fastq_wy> -rs 0 -na
```

du

Narzędzie wykorzystywane do pomiaru zapotrzebowania implementacji na przestrzeń dyskową.

```
du --max-depth=1 -b --exclude-from=<wylaczenia> <katalog> | cut -f1
>> <wy>
```

GNU time

Narzędzie wykorzystywane do pomiaru zapotrzebowania implementacji na pamięć oraz do pomiaru czasu obliczeń. `time -v -o <wy> <polecenie_parametry>`

vmtouch

Narzędzie wykorzystywane przed każdym wykonaniem korekcji, asemblacji lub mapowania dla każdego pliku wejściowego.

```
vmtouch -fe <plik>
```

SAMtools

Narzędzie wykorzystane w celu wyodrębnienia zmapowanych odczytów do plików FASTQ.

```
samtools fastq <sam> -1 <fastq_wy1> -2 <fastq_wy2>
```

B.3 Pozostałe wykorzystane dane

W tabeli B.4 przedstawiono numery dostępowe sekwencji wchodzących w skład poszczególnych genomów referencyjnych. Dane zostały uzyskane z bazy Genome NCBI [6]. Nagłówki pliku FAST genomu *A. thaliana* zostały zmodyfikowane poprzez oznaczenie ich kolejnymi liczbami naturalnymi w celu zachowania zgodności z oznaczeniem nagłówków w pliku prawdy podstawowej.

Tabela B.4: Genomy referencyjne; MT — DNA mitochondrialne, Pltd — DNA plastydu

Chromosom	Numer dostępowy sekwencji	Chromosom	Numer dostępowy sekwencji
P. syringae		*C. vulgaris*	
-	GCF_000585725.1	1	GCA_008119945.1
S. cerevisiae			
1	NC_001133.9	10	NC_001142.9
2	NC_001134.8	11	NC_001143.9
3	NC_001135.5	12	NC_001144.5
4	NC_001136.10	13	NC_001145.3
5	NC_001137.3	14	NC_001146.8
6	NC_001138.5	15	NC_001147.6
7	NC_001139.9	16	NC_001148.4
8	NC_001140.6	MT	NC_001224.1
9	NC_001141.2		
C. elegans			
1	NC_003279.8	5	NC_003283.11
2	NC_003280.10	10	NC_003284.9
3	NC_003281.10	MT	NC_001328.1
4	NC_003282.8		
A. thaliana			
1	NC_003070.9	4	NC_003075.7
2	NC_003071.7	5	NC_003076.8
3	NC_003074.8		
M. acuminata			
1	NC_025202.1	7	NC_025208.1
2	NC_025203.1	8	NC_025209.1
3	NC_025204.1	9	NC_025210.1
4	NC_025205.1	10	NC_025211.1

5	NC_025206.1	11	NC_025212.1
6	NC_025207.1		
	Z. mays		
1	NC_024459.2	7	NC_024465.2
2	NC_024460.2	8	NC_024466.2
3	NC_024461.2	9	NC_024467.2
4	NC_024462.2	10	NC_024468.2
5	NC_024463.2	MT	NC_007982.1
6	NC_024464.2	Pltd	NC_001666.2
	H. sapiens		
1	NC_000001.11	14	NC_000014.9
2	NC_000002.12	15	NC_000015.10
3	NC_000003.12	16	NC_000016.10
4	NC_000004.12	17	NC_000017.11
5	NC_000005.10	18	NC_000018.10
6	NC_000006.12	19	NC_000019.10
7	NC_000007.14	20	NC_000020.11
8	NC_000008.11	21	NC_000021.9
9	NC_000009.12	22	NC_000022.11
10	NC_000010.11	X	NC_000023.11
11	NC_000011.10	Y	NC_000024.10
12	NC_000012.12	MT	NC_012920.1
13	NC_000013.11		

W tabeli B.5 przedstawiono zestawy odczytów rzeczywistych, wykorzystanych w eksperymentach bezpośrednio lub jako zestawy pochodne.

W tabeli B.6 przedstawiono zestawy pochodne odczytów rzeczywistych, będących podzbiorami zestawów lub połączeniem zestawów w tabeli B.5.

W tabeli B.7 przedstawiono zestawy odczytów rzeczywistych, wykorzystanych do uzyskania profili błędów na potrzeby generacji odczytów symulowanych.

W tabelach B.8 oraz B.9 przedstawiono własności zestawów odczytów symulowane uproszczoną metodą Quake oraz narzędziem ART.

W tabelach B.10 oraz B.11 przedstawiono adresy dostępowe do rzeczywistych zestawów odczytów.

W tabeli B.12 przedstawiono zestawy prawdy podstawowej (*ground truth*), wykorzystane w celu oceny detekcji wariantów wraz z ich adresami dostępowymi.

B.4 Wartości parametrów algorytmów korekcji

W niniejszym podrozdziale przedstawiono wszystkie parametry, które zostały przekazane do algorytmów korekcji w celu uzyskania najlepszych wyników.

W tabeli B.13 przedstawiono parametry zależne od genomu źródłowego odczytów, które nie ulegały zmianie wraz z rodzajem odczytów (tzn. odczyty rzeczywiste, symulowane), ich własnościami ani dalszym zastosowaniem (tzn. asemblacja, mapowanie, detekcja wariantów).

B.4.1 Odczyty symulowane — metoda Quake

W tabelach B.14–B.16 przedstawiono wartości parametrów algorytmów korekcji, wykorzystane w celu korekcji odczytów symulowanych narzędziem Quake.

B.4.2 Odczyty symulowane — metoda ART

W tabelach B.17–B.19 przedstawiono wartości parametrów algorytmów korekcji, wykorzystane w celu korekcji odczytów symulowanych narzędziem ART.

B.4.3 Odczyty rzeczywiste — asemblacja i mapowanie

W tabeli B.20 przedstawiono parametry algorytmów korekcji, wykorzystane w celu korekcji odczytów rzeczywistych, gdzie miarą jakości był wpływ odpowiednio na asemblację *de novo*.

B.4.4 Odczyty rzeczywiste — detekcja wariantów

W tabeli B.21 przedstawiono parametry algorytmów korekcji, wykorzystane w celu korekcji odczytów rzeczywistych, gdzie miarą jakości był wpływ na detekcję wariantów.

Tabela B.1: Wersje algorytmów korekcji

Algorytm	Wersja	Źródło
RECKONER	2.0	https://github.com/refresh-bio/RECKONER
BLESS	1.02	https://sourceforge.net/projects/bless-ec/files/
Musket	1.1	http://musket.sourceforge.net/homepage.htm
RACER	—	https://www.csd.uwo.ca/~ilie/RACER/
Fiona	2.4.0	https://github.com/seqan/seqan
Blue	1.1.3	https://bioinformatics.csiro.au/public/files/
Lighter	1.1.2	https://github.com/mourisl/Lighter/releases
BFC	BFC-ht, version r181	https://github.com/lh3/bfc
Karect	1.0	https://github.com/aminallam/karect
SAMDUDE	wersja z dnia 2 maja 2018 r.	https://github.com/irenatfh/SAMDUDE

Tabela B.2: Wersje pomocniczych algorytmów, narzędzi i bibliotek

Narzędzie	Zastosowanie	Wersja	Źródło
	Narzędzia (algorytmy) podlegające analizie wydajności		
BWA	Mapowanie odczytów	0.7.15-r1140	https://github.com/lh3/bwa/releases/tag/v0.7.15
Minia	Asemblacja	3.2.4	https://github.com/GATB/minia
	Narzędzia niepodlegające analizie wydajności		
ART	Symulacja odczytów	2.5.8	https://www.niehs.nih.gov/research/resources/software/biostatistics/art/index.cfm
bash	Środowisko uruchomieniowe	4.4.12(1)-release	repozytorium Debian
G++	Kompilator języka C++ (kompilacja BLESS)	4.9.2	repozytorium Debian
G++	Kompilator języka C++	7.2.0	https://gcc.gnu.org
GNU time	Pomiar zapotrzebowania na pamięć oraz czas obliczeń	1.7	repozytorium Debian
hap.py	Ocena wariantów	0.3.9	https://github.com/Illumina/hap.py/releases
Mono	Środowisko uruchomieniowe Blue	4.6.2	repozytorium Debian
OpenMPI	Biblioteka na potrzeby BLESS	1.8.7	https://www.open-mpi.org/software/ompi/v1.8
Python	Środowisko języka Python	2.7.13	repozytorium Debian
Python 3	Środowisko języka Python 3	3.5.3	repozytorium Debian
Quast	Ocena asemblacji	5.0.2	https://sourceforge.net/projects/quast/files
SAMtools	Przetwarzanie plików SAM	1.9	https://github.com/samtools/samtools
Strelka	Detekcja wariantów	2.9.9, centos6_x86_64	https://github.com/Illumina/strelka/releases
Syndip oraz run-flt	Ocena wariantów	0.4	https://github.com/lh3/CHM-eval
vmtouch	Zarządzanie pamięcią podręczną systemu operacyjnego	1.3.0	https://github.com/hoytech/vmtouch

Tabela B.3: Konfiguracja sprzętowa i programowa wykorzystanego serwera

Parametr	Wartość
Procesory	2 × Intel Xeon E5-2670 v3, 2,3 GHz 12 dwuwątkowych rdzeni każdy
RAM	256 GB (wg. raportu systemu operacyjnego ok. 251 GiB), wyłączona partycja wymiany
Pamięć masowa	6 × HDD 1 TB, 7200 rpm, pamięć podręczna 64 MB, interfejs SAS 6 Gbps, połączone w sprzętową macierz RAID-5 w konfiguracji strip size 256 kB
System operacyjny	Linux Debian 9.2.1 64-bit

Tabela B.5: Rzeczywiste zestawy odczytów wykorzystane w eksperymentach

Id.	Organizm	Numer dostępu	Liczba odczytów	Długość odczytu	Sekwenator	Głębokość sekw.	Wykorzystanie
S1	*S. cerevisiae*	ERR422544	4,777 mln	100 pz	HiSeq 2000	41	Asemblacja, mapowanie
C1	*C. elegans*	SRR543736	57,72 mln	101 pz	HiSeq 2000	57	Asemblacja, mapowanie
M1	*M. acuminata*	ERR204808	67,18 mln	108 pz	Genome Analyzer IIx	16	Asemblacja, mapowanie
Z1	*Z. mays*	SRR1575513	1,014 mld	100 pz	HiSeq 2000	46	Asemblacja, mapowanie
			Zestawy wykorzystane do uzyskania zestawów pochodnych				
P1	*P. syringae*	SRR1119292	2,577 mln	średnio 248 pz	MiSeq	106	Asemblacja, mapowanie
V1	*C. vulgaris*	SRR9478717	75,46 mln	151 pz	NovaSeq 6000	294	Asemblacja, mapowanie
A1	*A. thaliana*	SRR1945754	213,6 mln	101 pz	HiSeq 2000	181	Detekcja wariantów
H1_15[a]	*H. sapiens*	ERR174324	447,1 mln	101 pz	HiSeq 2000	16	Detekcja wariantów, asemblacja, mapowanie
H2	*H. sapiens*	ERR174325	431,9 mln	101 pz	HiSeq 2000	15	Detekcja wariantów, asemblacja, mapowanie
H3	*H. sapiens*	ERR174326	425,6 mln	101 pz	HiSeq 2000	15	Detekcja wariantów, asemblacja, mapowanie
H4	*H. sapiens*	ERR174327	423,4 mln	101 pz	HiSeq 2000	15	Detekcja wariantów
H5_55	*H. sapiens*	ERR1341796	1,029 mld	151 pz	HiSeq X Ten	55	Detekcja wariantów

[a]Głębokość równa 15 jest przybliżona.

Tabela B.6: Zestawy pochodne — rzeczywiste zestawy odczytów uzyskane z połączenia zestawów w tabeli B.5 lub będących podzbiorami odczytów jednego z nich

Id.	Organizm	Numery dostępu	Liczba odczytów	Długość odczytu	Sekwenator	Głębokość sekw.	Wykorzystanie
P1_60	*P. syringae*	SRR1119292	2,577 mln	średnio 248 pz	MiSeq	60	Asemblacja, mapowanie
V1_60	*C. vulgaris*	SRR9478717	14,99 mln	151 pz	NovaSeq 6000	60	Asemblacja, mapowanie
A1_10	*A. thaliana*	SRR1945754	11,8 mln	101 pz	HiSeq 2000	10	Detekcja wariantów
A1_20	*A. thaliana*	SRR1945754	23,6 mln	101 pz	HiSeq 2000	20	Detekcja wariantów
A1_30	*A. thaliana*	SRR1945754	35,41 mln	101 pz	HiSeq 2000	30	Detekcja wariantów
A1_40	*A. thaliana*	SRR1945754	47,21 mln	101 pz	HiSeq 2000	40	Detekcja wariantów
A1_50	*A. thaliana*	SRR1945754	59,01 mln	101 pz	HiSeq 2000	50	Detekcja wariantów
A1_60	*A. thaliana*	SRR1945754	70,81 mln	101 pz	HiSeq 2000	60	Detekcja wariantów
A1_70	*A. thaliana*	SRR1945754	82,61 mln	101 pz	HiSeq 2000	70	Detekcja wariantów
A1_80	*A. thaliana*	SRR1945754	94,41 mln	101 pz	HiSeq 2000	80	Detekcja wariantów
A1_90	*A. thaliana*	SRR1945754	106,2 mln	101 pz	HiSeq 2000	90	Detekcja wariantów
A1_100	*A. thaliana*	SRR1945754	118 mln	101 pz	HiSeq 2000	100	Detekcja wariantów
H1–2_30	*H. sapiens*	ERR174324 ERR174325	879 mln	101 pz	HiSeq 2000	31	Detekcja wariantów
H1–3_45	*H. sapiens*	ERR174324 ERR174325 ERR174326	1,305 mld	101 pz	HiSeq 2000	47	Detekcja wariantów, asemblacja, mapowanie
H1–4_60	*H. sapiens*	ERR174324 ERR174325 ERR174326 ERR174327	1,728 mld	101 pz	HiSeq 2000	62	Detekcja wariantów
H5_15	*H. sapiens*	ERR1341796	280,4 mln	151 pz	HiSeq X Ten	15	Detekcja wariantów
H5_30	*H. sapiens*	ERR1341796	560,9 mln	151 pz	HiSeq X Ten	30	Detekcja wariantów
H5_45	*H. sapiens*	ERR1341796	841,3 mln	151 pz	HiSeq X Ten	45	Detekcja wariantów

Tabela B.7: Rzeczywiste zestawy odczytów wykorzystane do uzyskania profili błędów; uwzględniono wyłącznie pierwsze odczyty z par

Identyfikator	Numery dostępu	Liczba odczytów	Długość odczytu	p_{mean}
Pr1	DRR031158	10,31 mln	100 pz	2,02% ≈ 2%
Pr2	SRR1802178	33,81 mln	150 pz	1,84% ≈ 2%
Pr3	SRR065390	33,62 mln	100 pz	5,46% ≈ 4–5%
Pr4	SRR650760	8,17 mln	151 pz	4,16% ≈ 4–5%

Tabela B.8: Symulowane zestawy odczytów wykorzystane w eksperymentach — uproszczona metoda Quake

Organizm	Głębokość sekw.	$p_{\text{mean}} = 2\%$ Identyfikatory Q2_*	Liczba odczytów	Długość odczytu	Źródło profilu	$p_{\text{mean}} = 4$–5% Identyfikatory Q5_*	Liczba odczytów	Długość odczytu	Źródło profilu
S. cerevisiae	20	S1_L100D20	2,326 mln	100 pz	Pr1	S1_L100D20	2,326 mln	100 pz	Pr3
	30	S1_L100D30	3,49 mln	100 pz	Pr1	S1_L100D30	3,49 mln	100 pz	Pr3
	60	S1_L100D60	6,979 mln	100 pz	Pr1	S1_L100D60	6,979 mln	100 pz	Pr3
	20	S1_L150D20	1,55 mln	150 pz	Pr2	S1_L151D20	1,54 mln	151 pz	Pr4
	30	S1_L150D30	2,326 mln	150 pz	Pr2	S1_L151D30	2,311 mln	151 pz	Pr4
	60	S1_L150D60	4,653 mln	150 pz	Pr2	S1_L151D60	4,622 mln	151 pz	Pr4
C. elegans	20	C1_L100D20	20,57 mln	100 pz	Pr1	C1_L100D20	20,57 mln	100 pz	Pr3
	30	C1_L100D30	30,85 mln	100 pz	Pr1	C1_L100D30	30,85 mln	100 pz	Pr3
	60	C1_L100D60	61,7 mln	100 pz	Pr1	C1_L100D60	61,7 mln	100 pz	Pr3
	20	C1_L150D20	13,71 mln	150 pz	Pr2	C1_L151D20	13,62 mln	151 pz	Pr4
	30	C1_L150D30	20,57 mln	150 pz	Pr2	C1_L151D30	20,43 mln	151 pz	Pr4
	60	C1_L150D60	41,13 mln	150 pz	Pr2	C1_L151D60	40,86 mln	151 pz	Pr4
M. acuminata	20	M1_L100D20	92,31 mln	100 pz	Pr1	M1_L100D20	92,31 mln	100 pz	Pr3
	30	M1_L100D30	138,5 mln	100 pz	Pr1	M1_L100D30	138,5 mln	100 pz	Pr3
	60	M1_L100D60	276,9 mln	100 pz	Pr1	M1_L100D60	276,9 mln	100 pz	Pr3
	20	M1_L150D20	61,54 mln	150 pz	Pr2	M1_L151D20	61,13 mln	151 pz	Pr4
	30	M1_L150D30	92,31 mln	150 pz	Pr2	M1_L151D30	91,7 mln	151 pz	Pr4
	60	M1_L150D60	184,6 mln	150 pz	Pr2	M1_L151D60	183,4 mln	151 pz	Pr4

Tabela B.9: Symulowane zestawy odczytów wykorzystane w eksperymentach — narzędzie ART

Organizm	Głębokość sekw.	$p_{\text{mean}} = 2\%$ Identyfikatory A2_*	Liczba odczytów	Długość odczytu	Źródło profilu	$p_{\text{mean}} = 4$–5% Identyfikatory A5_*	Liczba odczytów	Długość odczytu	Źródło profilu
S. cerevisiae	20	S1_L100D20	2,326 mln	100 pz	Pr1	S1_L100D20	2,326 mln	100 pz	Pr3
	30	S1_L100D30	3,49 mln	100 pz	Pr1	S1_L100D30	3,49 mln	100 pz	Pr3
	60	S1_L100D60	6,979 mln	100 pz	Pr1	S1_L100D60	6,979 mln	100 pz	Pr3
	20	S1_L150D20	1,55 mln	150 pz	Pr2	S1_L151D20	1,54 mln	151 pz	Pr4
	30	S1_L150D30	2,326 mln	150 pz	Pr2	S1_L151D30	2,311 mln	151 pz	Pr4
	60	S1_L150D60	4,653 mln	150 pz	Pr2	S1_L151D60	4,622 mln	151 pz	Pr4
C. elegans	20	C1_L100D20	20,57 mln	100 pz	Pr1	C1_L100D20	20,57 mln	100 pz	Pr3
	30	C1_L100D30	30,85 mln	100 pz	Pr1	C1_L100D30	30,85 mln	100 pz	Pr3
	60	C1_L100D60	61,7 mln	100 pz	Pr1	C1_L100D60	61,7 mln	100 pz	Pr3
	20	C1_L150D20	13,71 mln	150 pz	Pr2	C1_L151D20	13,62 mln	151 pz	Pr4
	30	C1_L150D30	20,57 mln	150 pz	Pr2	C1_L151D30	20,43 mln	151 pz	Pr4
	60	C1_L150D60	41,13 mln	150 pz	Pr2	C1_L151D60	40,86 mln	151 pz	Pr4
M. acuminata	20	M1_L100D20	79,4 mln	100 pz	Pr1	M1_L100D20	79,4mln	100 pz	Pr3
	30	M1_L100D30	119,1 mln	100 pz	Pr1	M1_L100D30	119,1mln	100 pz	Pr3
	60	M1_L100D60	238,2 mln	100 pz	Pr1	M1_L100D60	238,2mln	100 pz	Pr3
	20	M1_L150D20	52,79 mln	150 pz	Pr2	M1_L151D20	52,44 mln	151 pz	Pr4
	30	M1_L150D30	79,19 mln	150 pz	Pr2	M1_L151D30	78,67 mln	151 pz	Pr4
	60	M1_L150D60	158,4 mln	150 pz	Pr2	M1_L151D60	157,3 mln	151 pz	Pr4

Tabela B.10: Odnośniki do rzeczywistych zestawów danych

Identyfikator	Numer dostępu	Adres
P1	SRR1119292	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR111/002/SRR1119292/SRR1119292_1.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR111/002/SRR1119292/SRR1119292_2.fastq.gz
S1	ERR422544	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR422/ERR422544/ERR422544_1.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR422/ERR422544/ERR422544_2.fastq.gz
V1	SRR9478717	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR947/007/SRR9478717/SRR9478717_1.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR947/007/SRR9478717/SRR9478717_2.fastq.gz
C1	SRR543736	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR543/SRR543736/SRR543736_1.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR543/SRR543736/SRR543736_2.fastq.gz
A1	SRR1945754	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR194/004/SRR1945754/SRR1945754_1.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR194/004/SRR1945754/SRR1945754_2.fastq.gz
M1	ERR204808	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR204/ERR204808/ERR204808_1.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR204/ERR204808/ERR204808_2.fastq.gz
Z1	SRR1575513	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR157/003/SRR1575513/SRR1575513_1.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR157/003/SRR1575513/SRR1575513_2.fastq.gz

Tabela B.11: Odnośniki do rzeczywistych zestawów danych, cd.

Identyfikator	Organizm dostępu	Adres
H1_15	ERR174324	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174324/ERR174324_1.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174324/ERR174324_2.fastq.gz
H2	ERR174325	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174325/ERR174325_1.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174325/ERR174325_2.fastq.gz
H3	ERR174326	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174326/ERR174326_1.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174326/ERR174326_2.fastq.gz
H4	ERR174327	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174327/ERR174327_1.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR174/ERR174327/ERR174327_2.fastq.gz
H5_55	ERR1341796	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR134/006/ERR1341796/ERR1341796_1.fastq.gz ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR134/006/ERR1341796/ERR1341796_2.fastq.gz
Pr1	DRR031158	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/DRR031/DRR031158/DRR031158_1.fastq.gz
Pr2	SRR1802178	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR180/008/SRR1802178/SRR1802178_1.fastq.gz
Pr3	SRR065390	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR065/SRR065390/SRR065390_1.fastq.gz
Pr4	SRR650760	ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR650/SRR650760/SRR650760_1.fastq.gz

Tabela B.12: Zestawy prawdy podstawowej (*ground truth*) dla oceny detekcji wariantów

Organizm	Identyfikator	Zestaw GT	Odnośnik
H. sapiens	H1_15 H2 H3 H4	NA12878 (HG001), wersja 3.3.2, w tym BED	A
	H5_55	Zestaw dostępny z Syndip, wersja 0.4	—
A. thaliana	A1_10	intersection 6904, wersja 3.1	B

Odnośnik	Adres
A	ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/NA12878_HG001/NISTv3.3.2/GRCh38/HG001_GRCh38_GIAB_highconf_CG-IllFB-IllGATKHC-Ion-10X-SOLID_CHROM1-X_v.3.3.2_highconf_PGandRTGphasetransfer.vcf.gz ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/NA12878_HG001/NISTv3.3.2/GRCh38/HG001_GRCh38_GIAB_highconf_CG-IllFB-IllGATKHC-Ion-10X-SOLID_CHROM1-X_v.3.3.2_highconf_PGandRTGphasetransfer.vcf.gz.tbi ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/NA12878_HG001/NISTv3.3.2/GRCh38/HG001_GRCh38_GIAB_highconf_CG-IllFB-IllGATKHC-Ion-10X-SOLID_CHROM1-X_v.3.3.2_highconf_nosomaticdel_noCENorHET7.bed
B	https://1001genomes.org/data/GMI-MPI/releases/v3.1/intersection_snp_short_indel_vcf/intersection_6904.vcf.gz

Tabela B.13: Wartości parametrów algorytmów korekcji odczytów, wspólne dla wybranych zestawów odczytów

		Jednostki taksonomiczne wraz z oznaczeniem zestawów							
		P. syringae	*S. cerevisiae*	*C. vulgaris*	*C. elegans*	*A. thaliana*	*M. acuminata*	*Z. mays*	*H. sapiens*
Algorytm	Parametr algorytmu	P1_60	S1 Q2_S1_* Q5_S1_* A2_S1_* A5_S1_*	V1_60	C1 Q2_C1_* Q5_C1_* A2_C1_* A5_C1_*	A1	M1 Q2_M1_* Q5_M1_* A2_M1_* A5_M1_*	Z1	H1_15 H1–2_30 H1–3_45 H1–4_60
RACER Fiona Blue Lighter BFC	$\widehat{\ell}_{\mathcal{G}}$	6033803	11631719	37731351	102831857	119197130	461539000	2190096555	2823132909
Algorytm	Parametr algorytmu	P1_60	S1	V1_60	C1	A1	M1	Z1	H1_15 H1–2_30 H1–3_45 H1–4_60
Karect	Ploidia	haploid	diploid	haploid	diploid	diploid	diploid	diploid	diploid
Algorytm	Parametr algorytmu	—	Q2_S1_* Q5_S1_* A2_S1_* A5_S1_*	—	Q2_C1_* Q5_C1_* A2_C1_* A5_C1_*	—	Q2_M1_* Q5_M1_* A2_M1_* A5_M1_*	—	—
Karect	Ploidia	—	haploid	—	haploid	—	haploid	—	—

Tabela B.14: Parametry algorytmów korekcji dobrane wg miary ξ_{gain} — odczyty z grupy Q2_* (symulowane metodą Quake)

	Algorytm	Parametr	*_L100D20	*_L100D30	*_L100D60	*_L150D20	*_L150D30	*_L150D60
Q2_S1_*	RECKONER	k	25					
	Musket	k	27					
		$\widehat{\bar{k}}_{\text{N}}$	172149456	258224184	516448294	192311104	288466656	576933312
	BLESS	k	28					
	Blue	k	17					
		η_{cut}	3	5	5	3	5	5
	Lighter	k	30					
		p_α	0,175	0,1166667	0,0583333	0,175	0,1166667	0,0583333

	Algorytm	Parametr	*_L100D20	*_L100D30	*_L100D60	*_L150D20	*_L150D30	*_L150D60
Q2_C1_*	RECKONER	k	34					
	Musket	k	27					
		$\widehat{\bar{k}}_{\text{N}}$	1521911454	2282867218	4565734436	1700153336	2550230004	5100460132
	BLESS	k	39					
	Blue	k	21					
		η_{cut}	3	5	5	4	5	5
	Lighter	k	32					
		p_α	0,175	0,1166667	0,0583333	0,175	0,1166667	0,0583333

Tabela B.15: Parametry algorytmów korekcji dobrane wg miary ξ_{gain} — odczyty z grup Q2_*, Q5_* (symulowane metodą Quake), cd.

	Algorytm	Parametr	*_L100D20	*_L100D30	*_L100D60	*_L150D20	*_L150D30	*_L150D60
Q2_M1_*	RECKONER	k	47					
	Musket	k	27					
		$\widehat{\widehat{k}}_{\text{N}}$	5895963883	8844011645	17688619518	6586544956	9880213805	19759170207
	BLESS	k	54					
	Blue	k	21					
		η_{cut}	5	5	5	5	5	5
	Lighter	k	32					
		p_α	0,175	0,1166667	0,0583333	0,175	0,1166667	0,0583333

	Algorytm	Parametr	*_L100D20	*_L100D30	*_L100D60	*_L151D20	*_L151D30	*_L151D60
Q5_S1_*	RECKONER	k	24					
	Musket	k	27					
		$\widehat{\widehat{k}}_{\text{N}}$	172149456	258224184	516448294	192578125	288867250	577734375
	BLESS	k	25					
	Blue	k	17					
		η_{cut}	3	5	5	3	5	5
	Lighter	k	30					
		p_α	0,175	0,1166667	0,0583333	0,175	0,1166667	0,0583333

Tabela B.16: Parametry algorytmów korekcji dobrane wg miary ξ_{gain} — odczyty z grupy Q5_* (symulowane metodą Quake), cd. 2

	Algorytm	Parametr	*_L100D20	*_L100D30	*_L100D60	*_L151D20	*_L151D30	*_L151D60
Q5_C1_*	RECKONER	k	33					
	Musket	k	27					
		$\widehat{\widehat{k}}_{\text{N}}$	1521911454	2282867218	4565734436	1702514250	2553771250	5107542625
	BLESS	k	37					
	Blue	k	19					
		η_{cut}	3	5	5	4	5	5
	Lighter	k	32					
		p_α	0,175	0,1166667	0,0583333	0,175	0,1166667	0,0583333

	Algorytm	Parametr	*_L100D20	*_L100D30	*_L100D60	*_L151D20	*_L151D30	*_L151D60
Q5_M1_*	RECKONER	k	43					
	Musket	k	27					
		$\widehat{\widehat{k}}_{\text{N}}$	5896148059	8844175773	17689152885	6596222121	9893960765	19787562835
	BLESS	k	53					
	Blue	k	24					
		η_{cut}	5	5	5	5	5	5
	Lighter	k	32					
		p_α	0,175	0,1166667	0,0583333	0,175	0,1166667	0,0583333

Tabela B.17: Parametry algorytmów korekcji dobrane wg miary ξ_{gain} — odczyty z grupy A2_* (symulowane metodą ART)

	Algorytm	Parametr	*_L100D20	*_L100D30	*_L100D60	*_L150D20	*_L150D30	*_L150D60
A2_S1_*	RECKONER	k	24					
	Musket	k	25					
		$\widehat{\widehat{k}}_{\text{N}}$	176802144	265203216	530406356	195412896	293119344	586238688
	BLESS	k	25					
	Blue	k	28					
		η_{cut}	3	5	4	4	5	5
	Lighter	k	20					
		p_α	0,175	0,1166667	0,0583333	0,175	0,1166667	0,0583333

	Algorytm	Parametr	*_L100D20	*_L100D30	*_L100D60	*_L150D20	*_L150D30	*_L150D60
A2_C1_*	RECKONER	k	31					
	Musket	k	27					
		$\widehat{\widehat{k}}_{\text{N}}$	1521911454	2282867218	4565734436	1700153336	2550230004	5100460132
	BLESS	k	31					
	Blue	k	31					
		η_{cut}	3	5	5	4	5	5
	Lighter	k	25					
		p_α	0,175	0,1166667	0,0583333	0,175	0,1166667	0,0583333

Tabela B.18: Parametry algorytmów korekcji dobrane wg miary ξ_{gain} — odczyty z grup A2_*, A5_* (symulowane metodą ART), cd.

	Algorytm	Parametr	*_L100D20	*_L100D30	*_L100D60	*_L150D20	*_L150D30	*_L150D60
A2_M1_*	RECKONER	k	45					
	Musket	k	27					
		$\widehat{\widehat{k}}_{\text{N}}$	5875378296	8812991150	17625154684	6546521968	9819842347	19639790713
	BLESS	k	41					
	Blue	k	31					
		η_{cut}	5	5	5	5	5	5
	Lighter	k	32					
		p_α	0,175	0,1166667	0,0583333	0,175	0,1166667	0,0583333

	Algorytm	Parametr	*_L100D20	*_L100D30	*_L100D60	*_L151D20	*_L151D30	*_L151D60
A5_S1_*	RECKONER	k	19					
	Musket	k	19					
		$\widehat{\widehat{k}}_{\text{N}}$	190760208	286140312	572280542	204903125	307354754	614709375
	BLESS	k	19					
	Blue	k	17					
		η_{cut}	4	5	5	4	5	5
	Lighter	k	17					
		p_α	0,175	0,1166667	0,0583333	0,175	0,1166667	0,0583333

Tabela B.19: Parametry algorytmów korekcji dobrane wg miary ξ_{gain} — odczyty z grupy A5_* (symulowane metodą ART), cd. 2

	Algorytm	Parametr	*_L100D20	*_L100D30	*_L100D60	*_L151D20	*_L151D30	*_L151D60
A5_C1_*	RECKONER	k	23					
	Musket	k	23					
		$\widehat{\widehat{k}}_{\text{N}}$	1604176938	2406265446	4812530869	1756994706	2635491930	5270983989
	BLESS	k	23					
	Blue	k	22					
		η_{cut}	3	5	5	4	5	5
	Lighter	k	18					
		p_α	0,175	0,1166667	0,0583333	0,175	0,1166667	0,0583333

	Algorytm	Parametr	*_L100D20	*_L100D30	*_L100D60	*_L151D20	*_L151D30	*_L151D60
A5_M1_*	RECKONER	k	33					
	Musket	k	26					
		$\widehat{\widehat{k}}_{\text{N}}$	5954427948	8931797223	17863848198	6607956977	9912309228	19824517925
	BLESS	k	33					
	Blue	k	31					
		η_{cut}	4	5	5	5	5	5
	Lighter	k	20					
		p_α	0,175	0,1166667	0,0583333	0,175	0,1166667	0,0583333

Tabela B.20: Wartości parametrów algorytmów korekcji — najlepsze wyniki dla asemblacji

Algorytm	Parametr algorytmu	P1_60	S1	V1_60	C1	M1	Z1	H1–3_45
RECKONER	k	21	28	37	24	30	29	33
Musket	k	27	20	27	27	25	17	21
	$\widehat{\widehat{k}}_{\mathrm{N}}$	324103443	386870788	1874059371	4326883814	5613969652	84922741931	105583172060
BLESS	k	18	24	25	12	32	21	23
Blue	k	29	30	31	24	29	—	—
	η_{cut}	5	5	5	5	3	—	—
Lighter	k	31	19	32	31	21	20	31
	p_α	0,0583334	0,0853659	0,0583333	0,0614035	0,21875	0,0760870	0,0744681

Tabela B.21: Wartości parametrów algorytmów korekcji — najlepsze wyniki dla detekcji wariantów

Algorytm	Parametr algorytmu	A1_10	A1_20	A1_30	A1_40	A1_50	A1_60	A1_70
RECKONER	k	21	21	21	25	25	25	25
Musket	k	27	27	27	27	27	27	27
	$\widehat{k}_{\mathrm{N}}$	882822008	1765634528	2648440858	3531262068	4414069446	5296885472	6179717566
BLESS	k	39	23	25	35	37	37	41
Blue	k	21	31	31	31	31	31	31
	η_{cut}	2	3	4	5	5	5	5
Lighter	k	31	31	31	31	29	31	31
	p_α	0,35	0,175	0,1166667	0,0875	0,07	0,0583333	0,05

Algorytm	Parametr algorytmu	A1_80 H5_15	A1_90 H5_30	A1_100 H5_45	H1_15 H5_55	H1–2_30	H1–3_45	H1–4_60
RECKONER	k	23	21	21	59	59	59	59
Musket	k	27	27	27	13	13	13	23
	$\widehat{k}_{\mathrm{N}}$	7062571008	7945389220	8828215498	39747974419	78165991096	116026013123	136390504558
BLESS	k	29	39	29	21	15	17	17
Blue	k	31	31	31	—	—	—	—
	η_{cut}	5	5	5	—	—	—	—
Lighter	k	31	31	29	11	13	11	11
	p_α	0,04375	0,0388889	0,035	0,21875	0,1129032	0,0744681	0,0564516

Dodatek C

Dodatkowe wyniki eksperymentów

C.1 Ocena wg kryteriów jakościowych

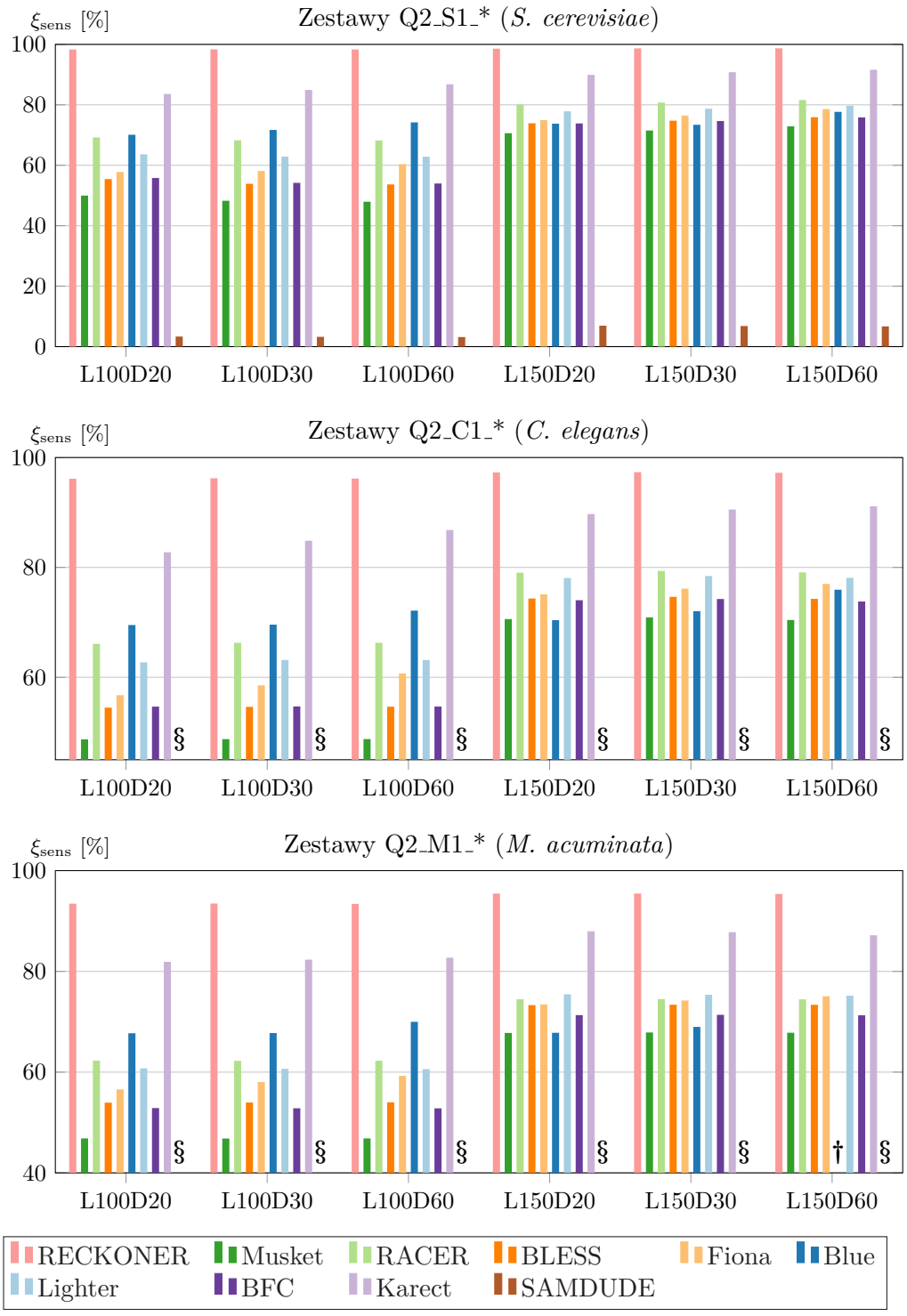
C.1.1 Odczyty symulowane

Symulacja metodą Quake

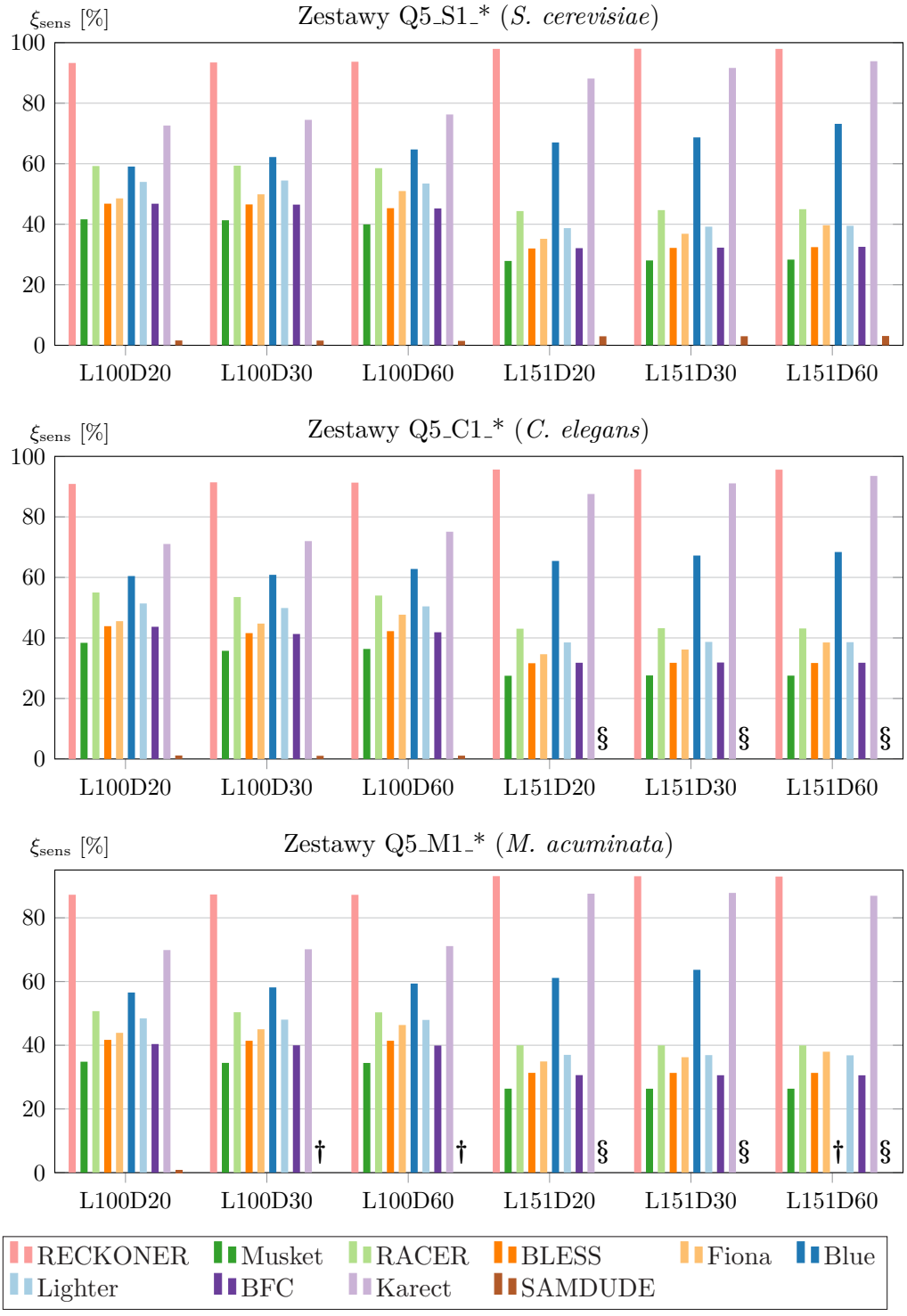
Na rysunkach C.1 oraz C.3 przedstawiono wyrażone procentowo wyniki korekcji według miary czułości (ξ_{sens}) i precyzji (ξ_{prec}) dla poszczególnych organizmów oraz odczytów różnej długości, o średnim prawdopodobieństwie błędu $p_{\text{mean}} = 2\%$, uzyskane dla wartości k_{best} maksymalizującej miarę zysku (ξ_{gain}).Z kolei na rysunkach C.2 oraz C.4 przedstawiono analogiczne wyniki dla $p_{\text{mean}} = 4$–5%. Wartości miar zostały przemnożone przez 100%.

Symulacja narzędziem ART

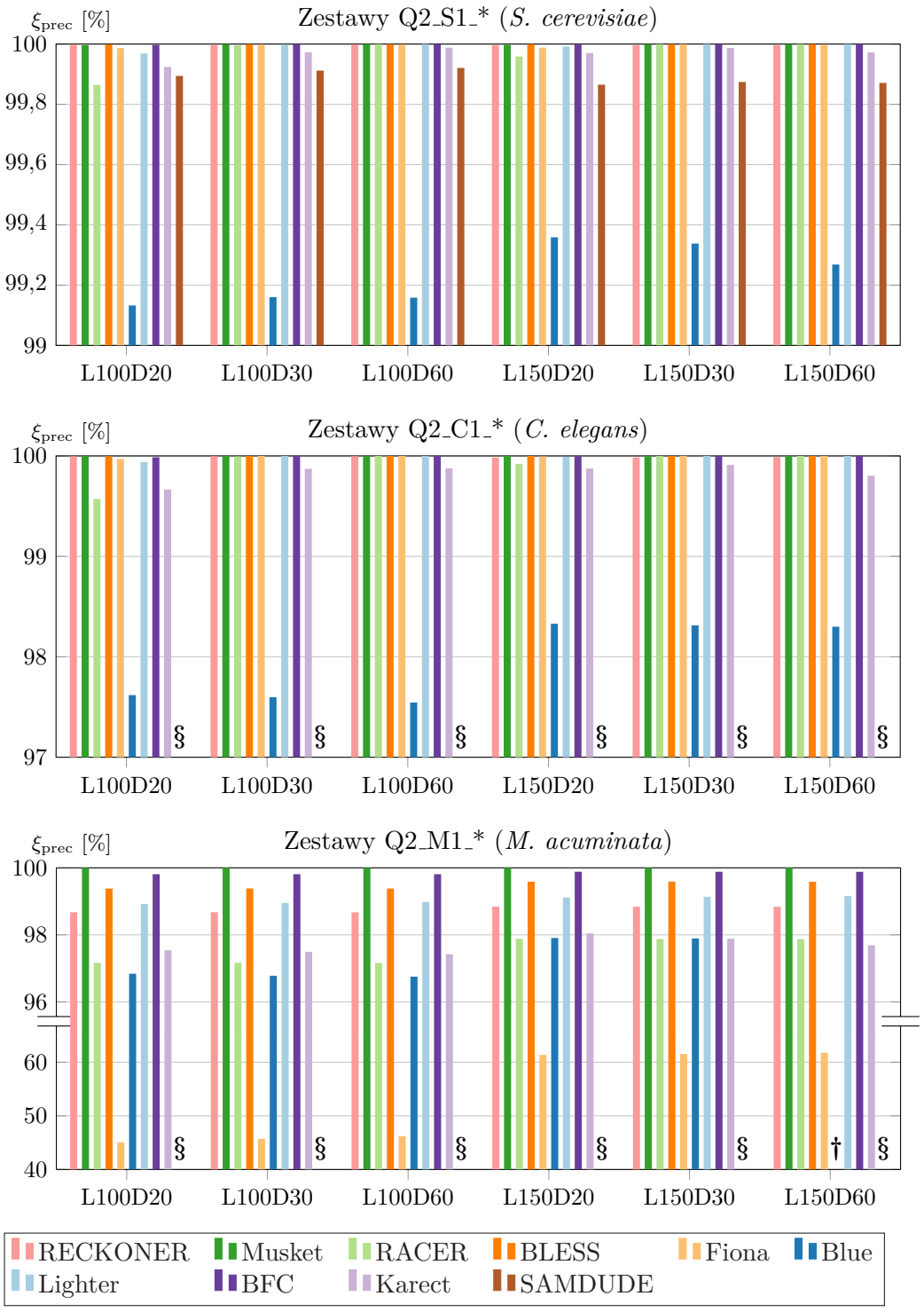
Na rysunkach C.5 oraz C.7 przedstawiono wyrażone procentowo wyniki korekcji według miary czułości (ξ_{sens}) i precyzji (ξ_{prec}) dla poszczególnych organizmów oraz odczytów różnej długości, o średnim prawdopodobieństwie błędu $p_{\text{mean}} = 2\%$, uzyskane dla wartości k_{best} maksymalizującej miarę zysku (ξ_{gain}). Z kolei na rysunkach C.6 oraz C.8 przedstawiono analogiczne wyniki dla $p_{\text{mean}} = 4$–5%. Wartości miar zostały przemnożone przez 100%.



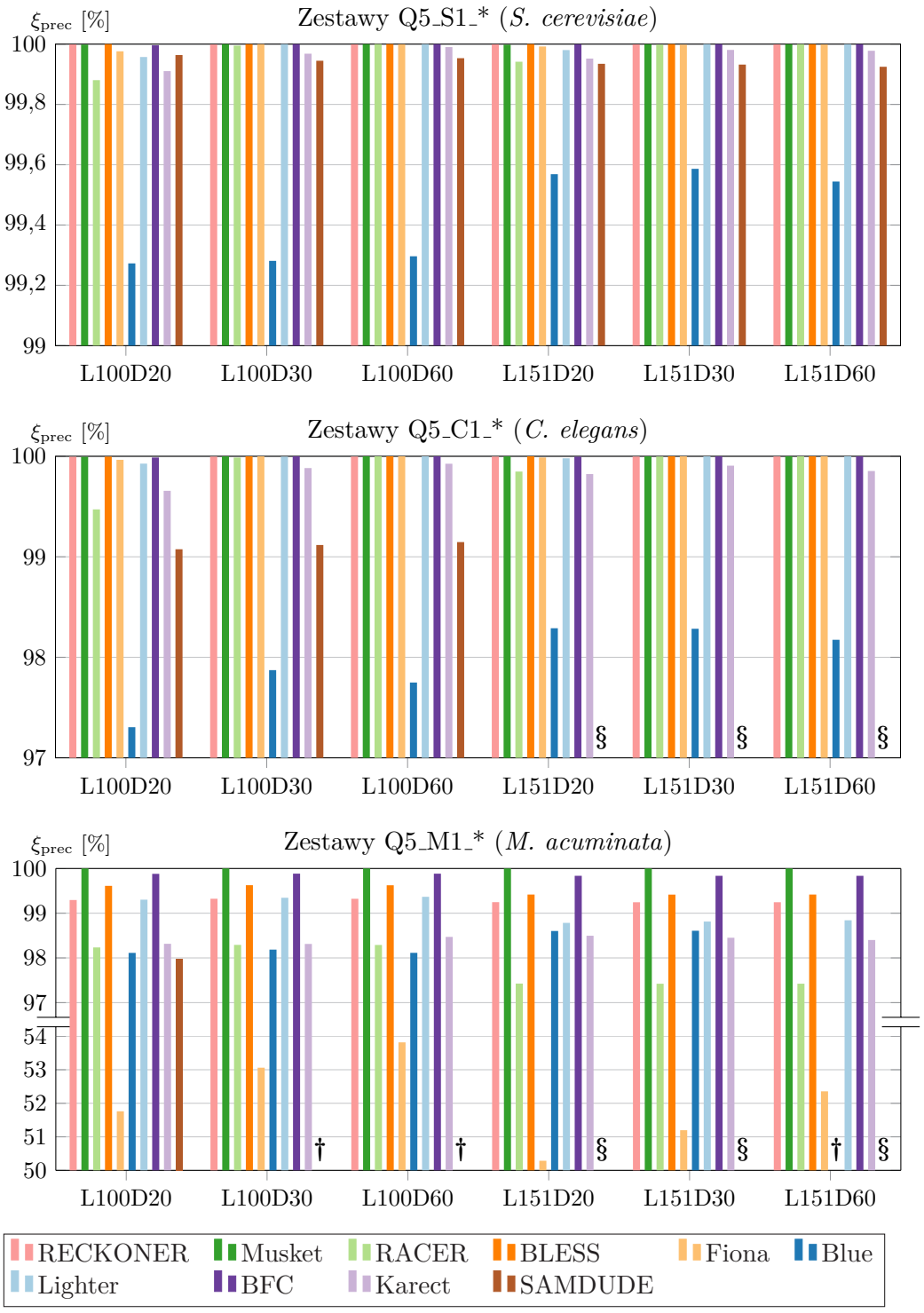
Rysunek C.1: Wpływ korekcji na wartość ξ_{sens} dla odczytów symulowanych uproszczoną metodą Quake, $p_{\text{mean}} = 2\%$



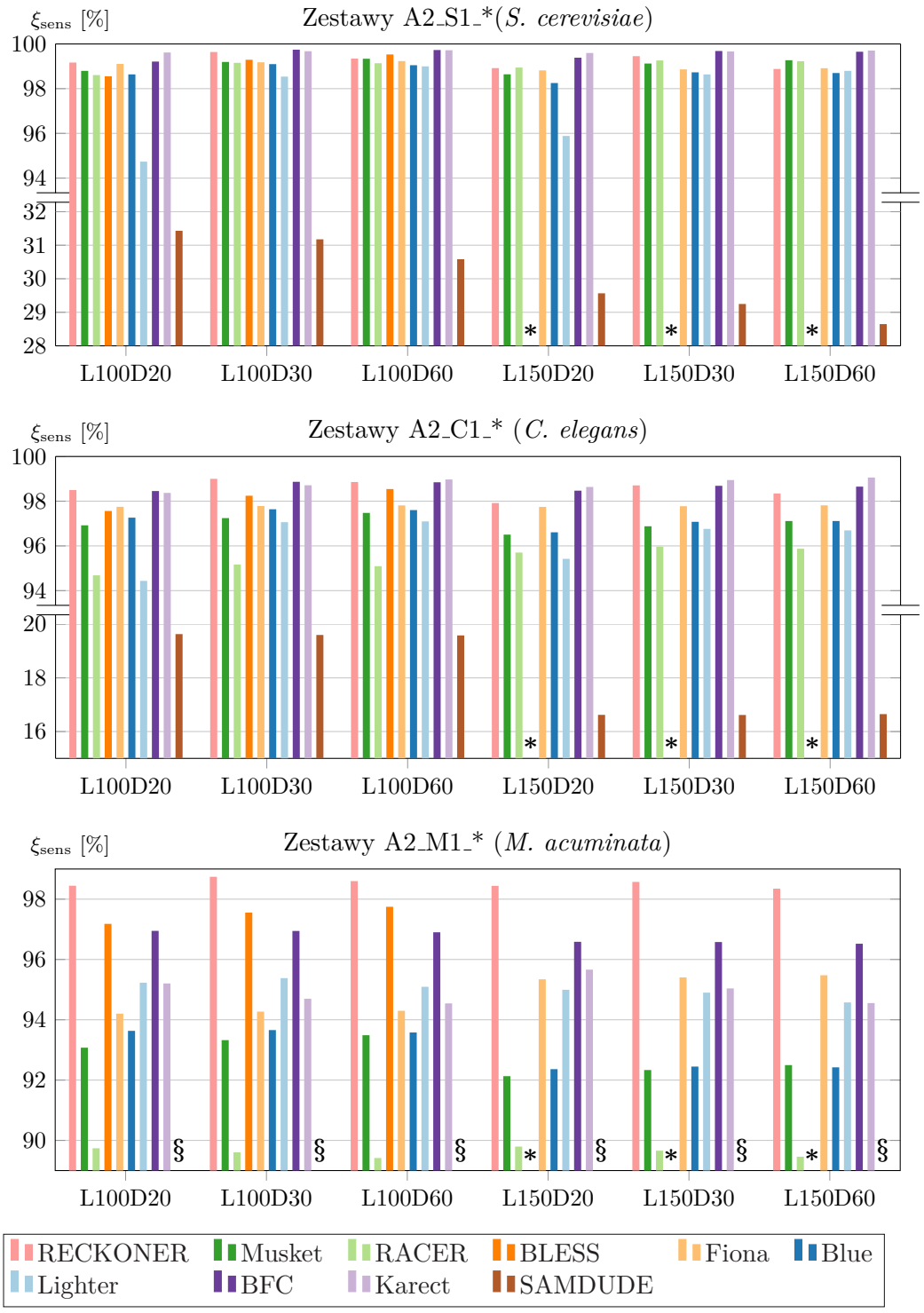
Rysunek C.2: Wpływ korekcji na wartość ξ_{sens} dla odczytów symulowanych uproszczoną metodą Quake, $p_{\text{mean}} = 4$–5%



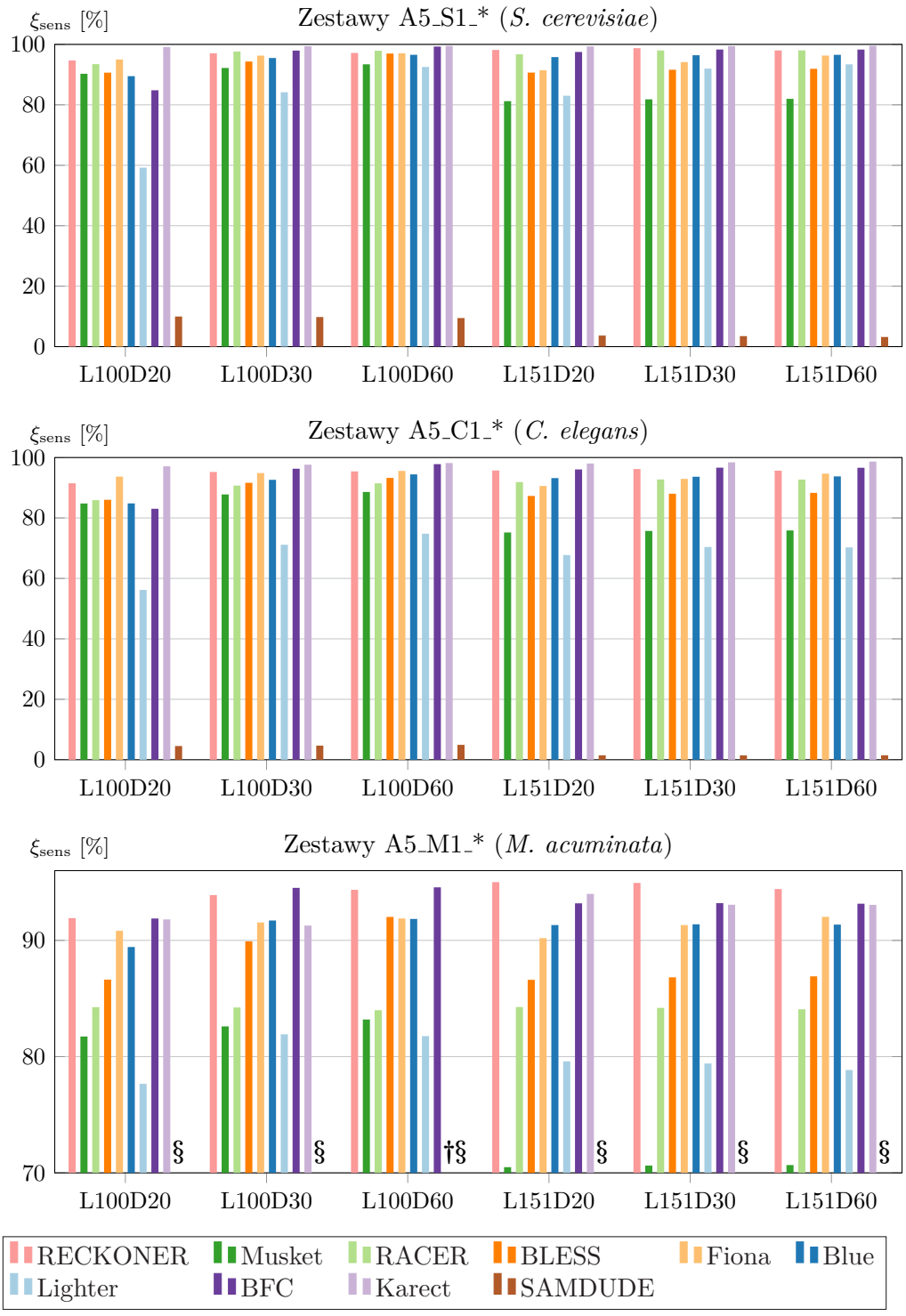
Rysunek C.3: Wpływ korekcji na wartość ξ_{prec} dla odczytów symulowanych uproszczoną metodą Quake, $p_{\text{mean}} = 2\%$



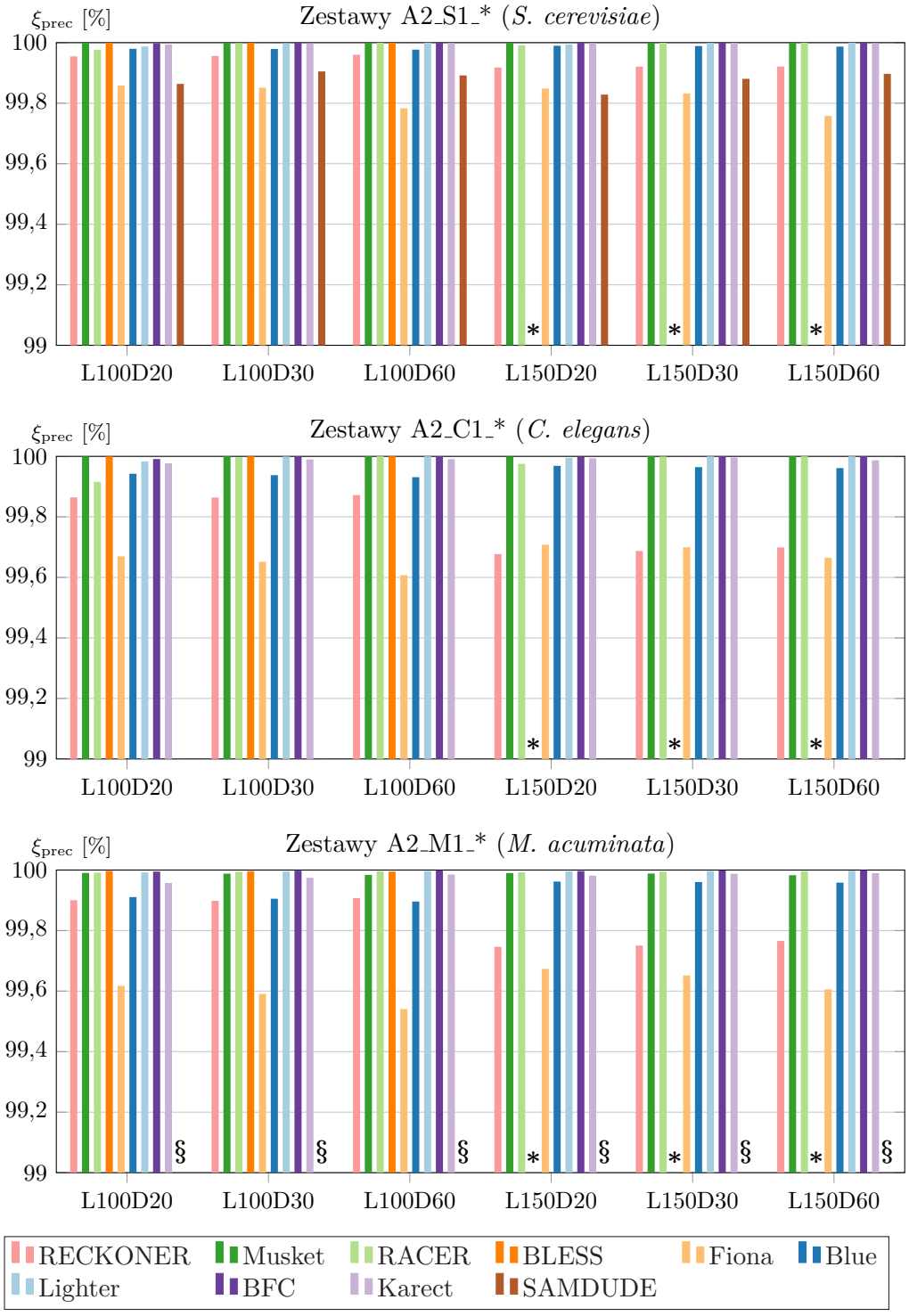
Rysunek C.4: Wpływ korekcji na wartość ξ_{prec} dla odczytów symulowanych uproszczoną metodą Quake, $p_{\text{mean}} = 4$–5%



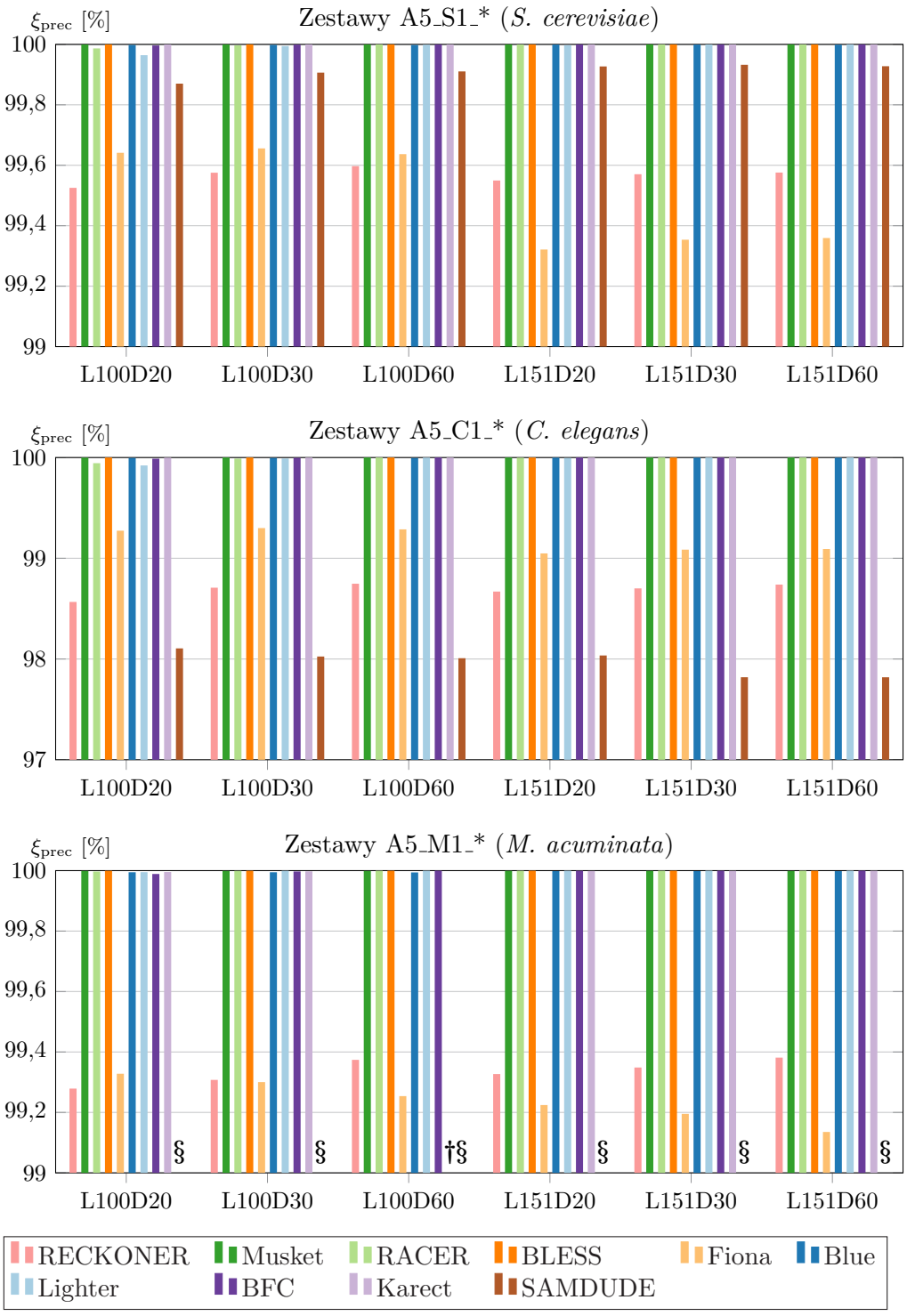
Rysunek C.5: Wpływ korekcji na wartość ξ_{sens} dla odczytów symulowanych narzędziem ART, $p_{\text{mean}} = 2\%$



Rysunek C.6: Wpływ korekcji na wartość ξ_{sens} dla odczytów symulowanych narzędziem ART, $p_{\text{mean}} = 4$–5%



Rysunek C.7: Wpływ korekcji na wartość ξ_{prec} dla odczytów symulowanych narzędziem ART, $p_{\text{mean}} = 2\%$



Rysunek C.8: Wpływ korekcji na wartość ξ_{prec} dla odczytów symulowanych narzędziem ART, $p_{\text{mean}} = 4$–5%

C.1.2 Odczyty rzeczywiste

Mapowanie

Na rys. C.9 oraz C.10 zaprezentowano zmianę czasu mapowania oraz zapotrzebowania na pamięć jako rezultat korekcji wejściowych odczytów.

Detekcja wariantów

Na rys. C.11 przedstawiono wartości ξ_{sens} detekcji dla odczytów *H. sapiens* dla różnych głębokości sekwencjonowania, odpowiednio z podziałem na liczbę wariantów typu SNP oraz indel, uzyskane przy pomocy narzędzia hap.py. Na rys. C.12 analogicznie przedstawiono wartości ξ_{prec}. Wszystkie wartości zostały przemnożone przez 100%.

Na rys. C.13 przedstawiono wartości ξ_{F1} detekcji dla odczytów *A. thaliana* dla różnych głębokości sekwencjonowania.

Wyniki detekcji wariantów jednego zestawu ludzkich odczytów, ocenione według tych samych miar przy pomocy narzędzia Syndip, przedstawiono na rys. C.14 oraz C.15.

C.2 Analiza według kryteriów wydajnościowych

C.2.1 Czas obliczeń i zapotrzebowanie na pamięć korekcji

Odczyty symulowane uproszczoną metodą Quake

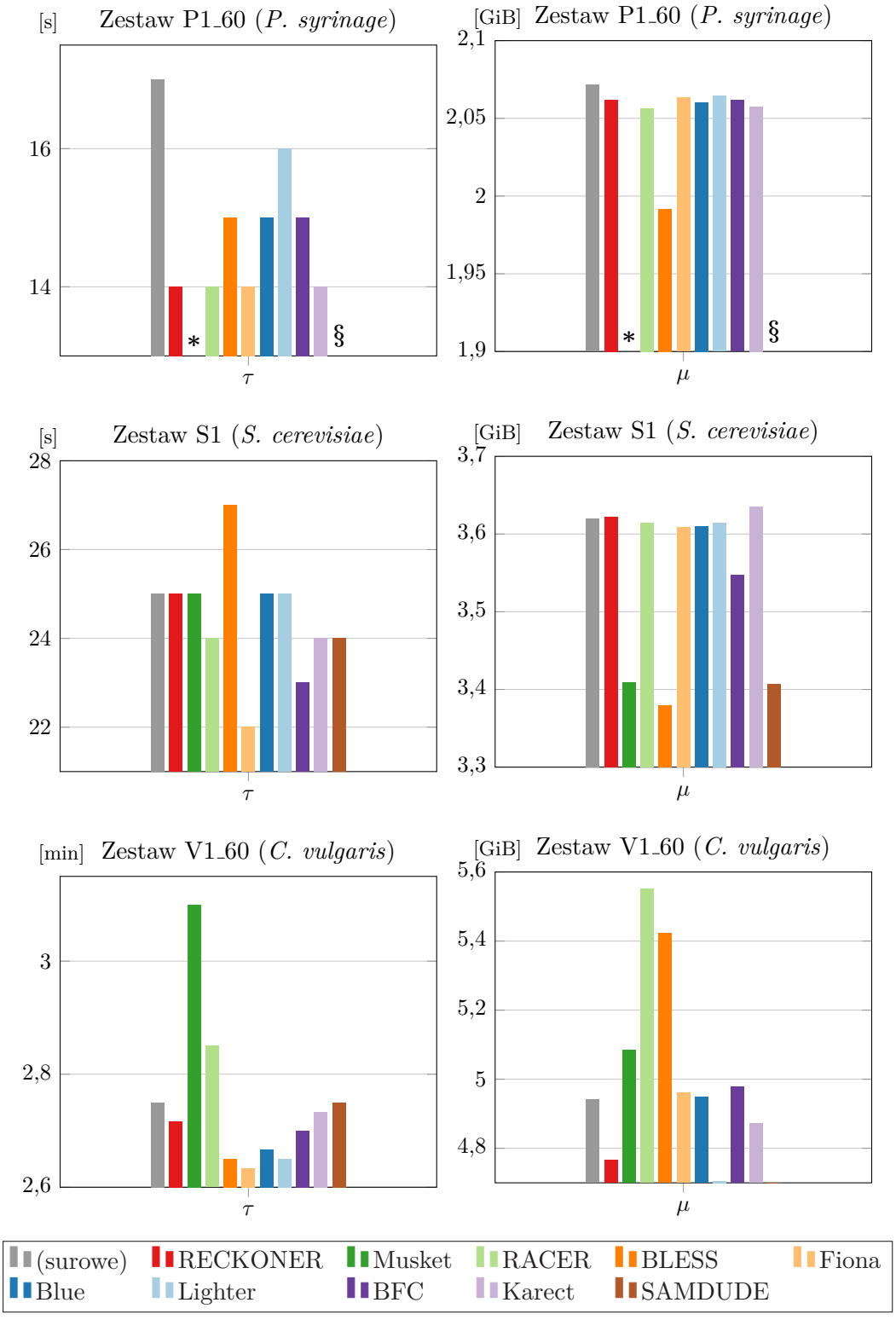
Na rys. C.16–C.19 przedstawiono czas obliczeń oraz zapotrzebowanie na pamięć operacyjną algorytmów korekcji, wykorzystanych do korekcji odczytów symulowanych uproszczoną metodą Quake.

Odczyty symulowane narzędziem ART

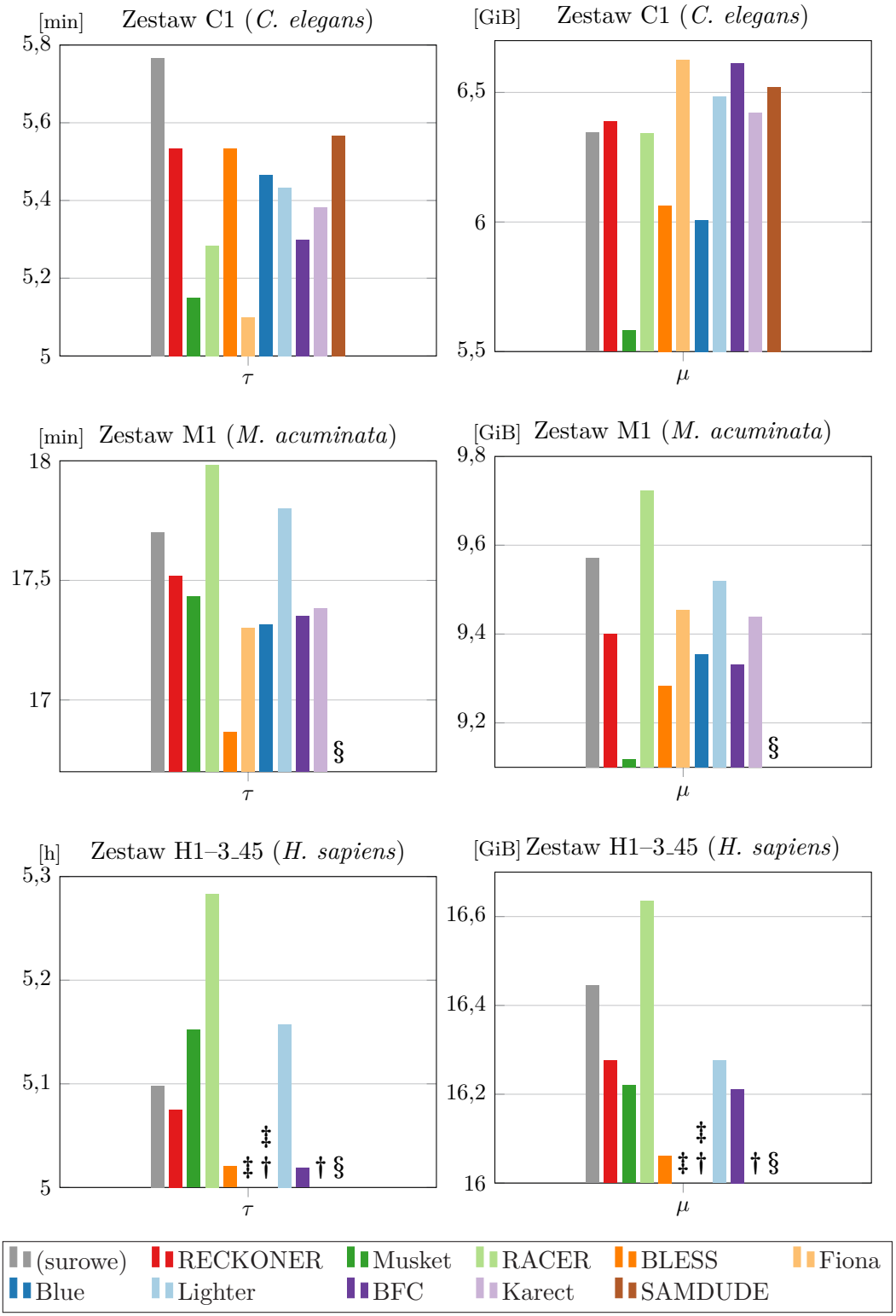
Na rys. C.20–C.23 przedstawiono czas obliczeń oraz zapotrzebowanie na pamięć operacyjną algorytmów korekcji, wykorzystanych do korekcji odczytów symulowanych narzędziem ART.

Odczyty rzeczywiste na potrzeby detekcji wariantów

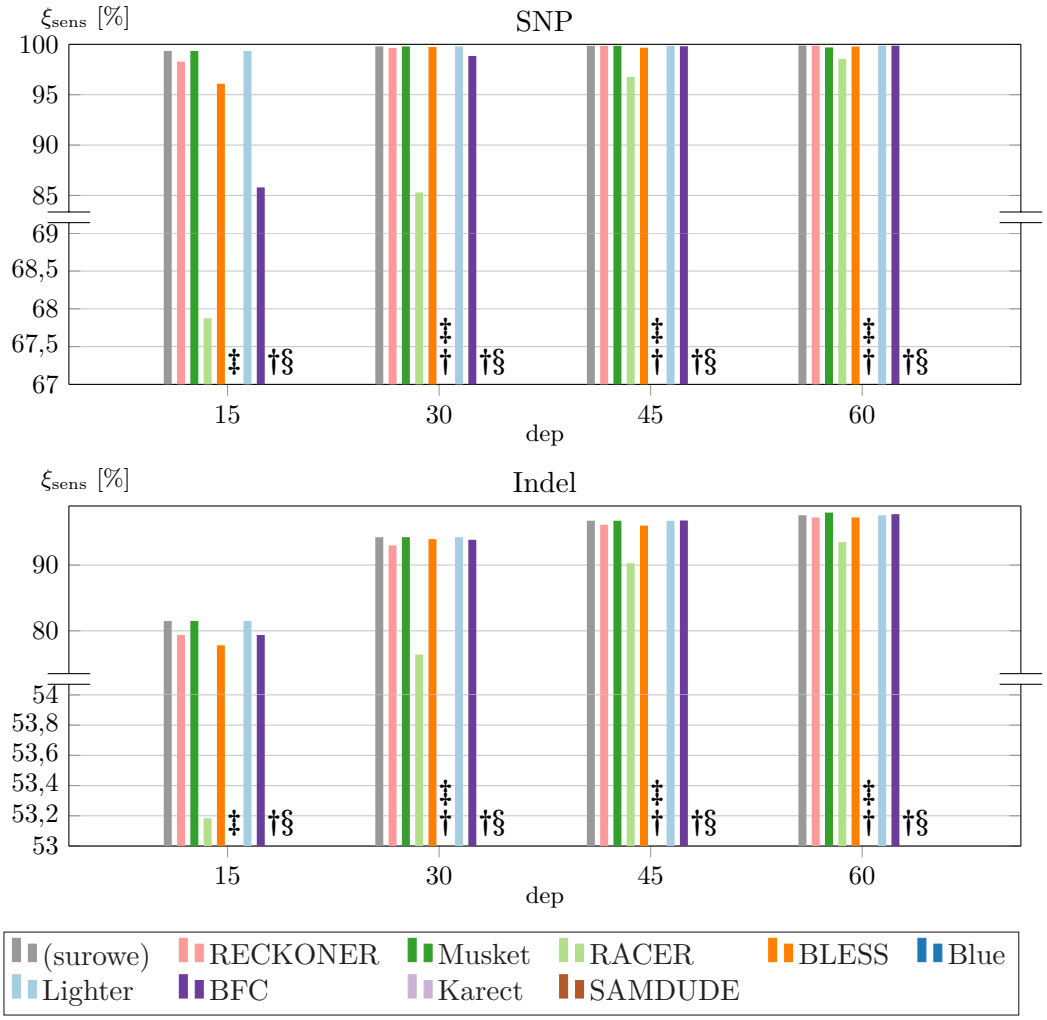
Na rys. C.24 oraz C.25 przedstawiono czas obliczeń oraz zapotrzebowanie na pamięć operacyjną algorytmów korekcji, wykorzystanych do korekcji odczytów rzeczywistych, zastosowanych w zadaniu detekcji wariantów.



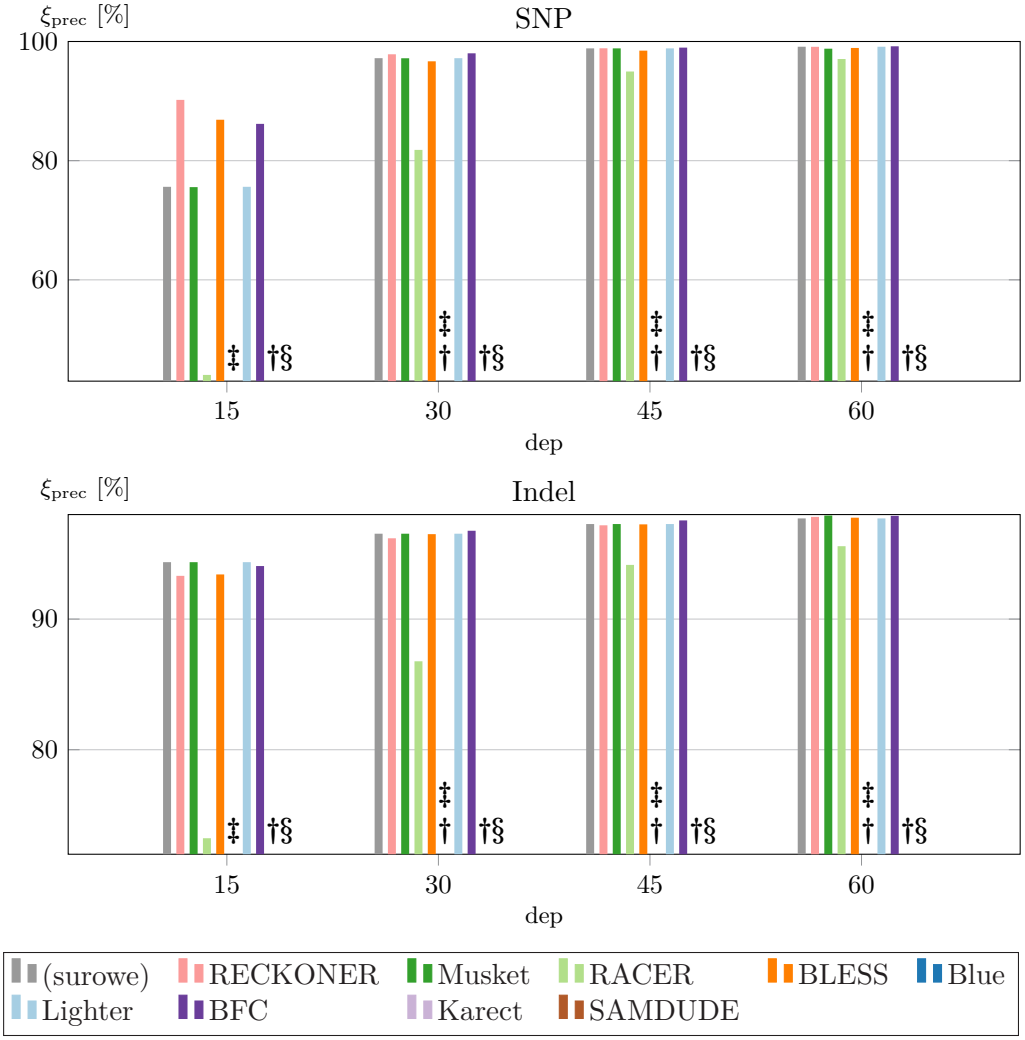
Rysunek C.9: Wpływ korekcji na zapotrzebowanie na zasoby mapowania odczytów z zestawów P1_60, S1, V1_60



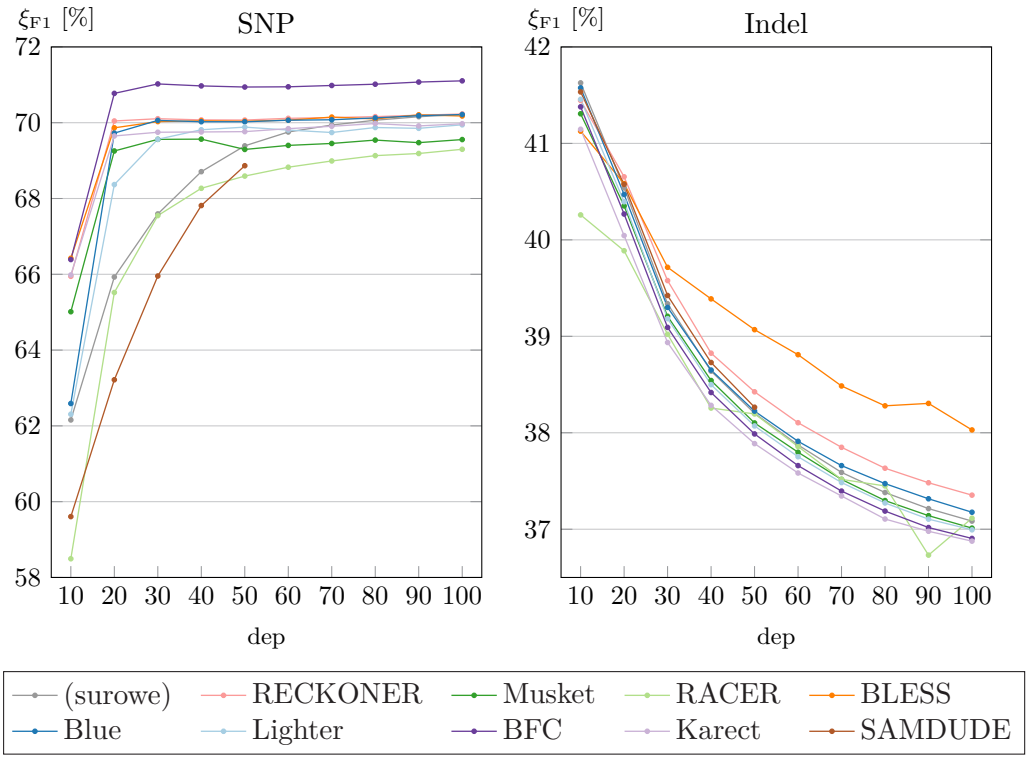
Rysunek C.10: Wpływ korekcji na zapotrzebowanie na zasoby mapowania odczytów z zestawów C1, E1, H1–3_45



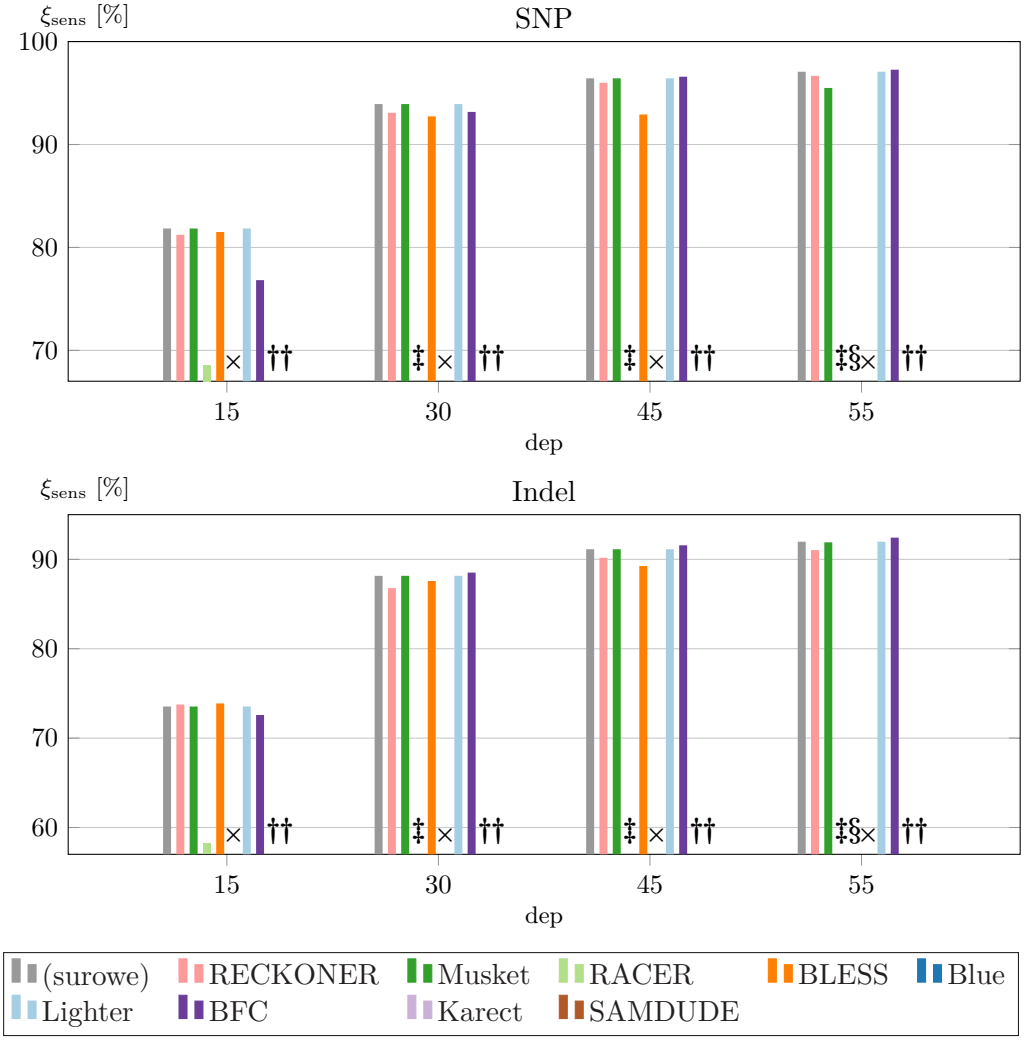
Rysunek C.11: Wpływ korekcji na wartość ξ_{sens} detekcji wariantów odczytów z zestawów H1_15, H1–2_30, H1–3_45, H1–4_60 (*H. sapiens*), różna głębokość sekwencjonowania — hap.py



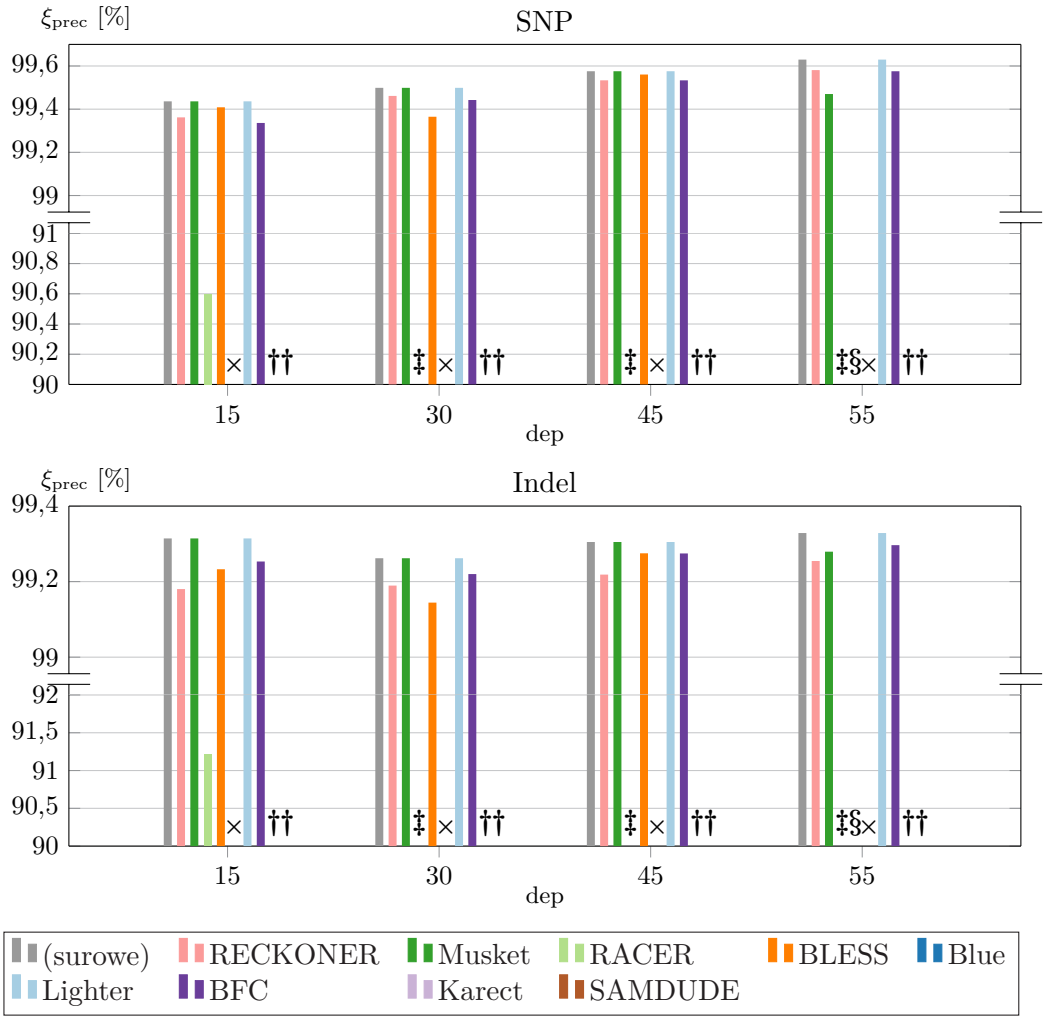
Rysunek C.12: Wpływ korekcji na wartość ξ_{prec} detekcji wariantów odczytów z zestawów H1_15, H1–2_30, H1–3_45, H1–4_60 (*H. sapiens*), różna głębokość sekwencjonowania — hap.py



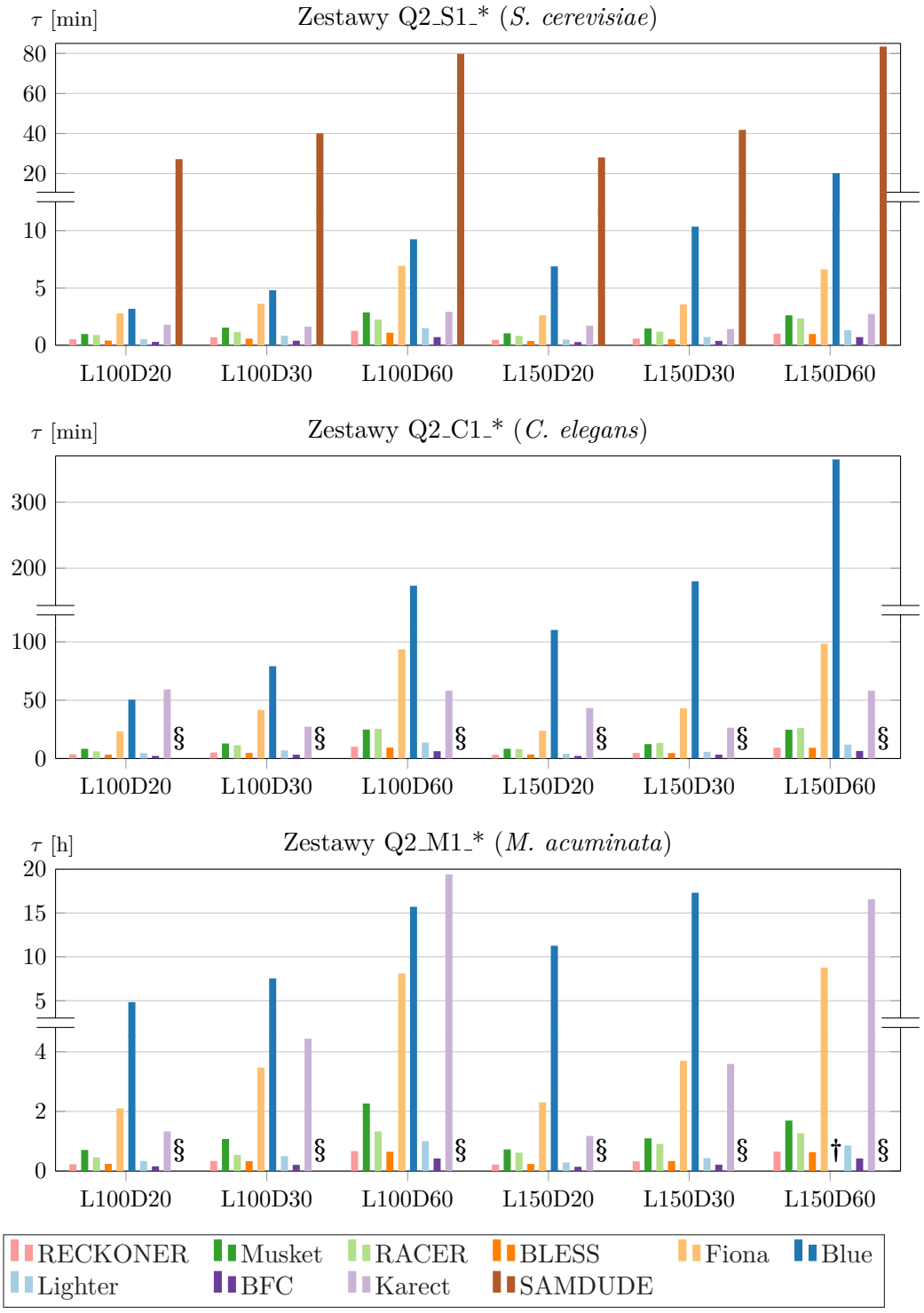
Rysunek C.13: Wpływ korekcji na wartość ξ_{F1} detekcji wariantów odczytów z zestawów A1_* (*A. thaliana*), różna głębokość sekwencjonowania — hap.py



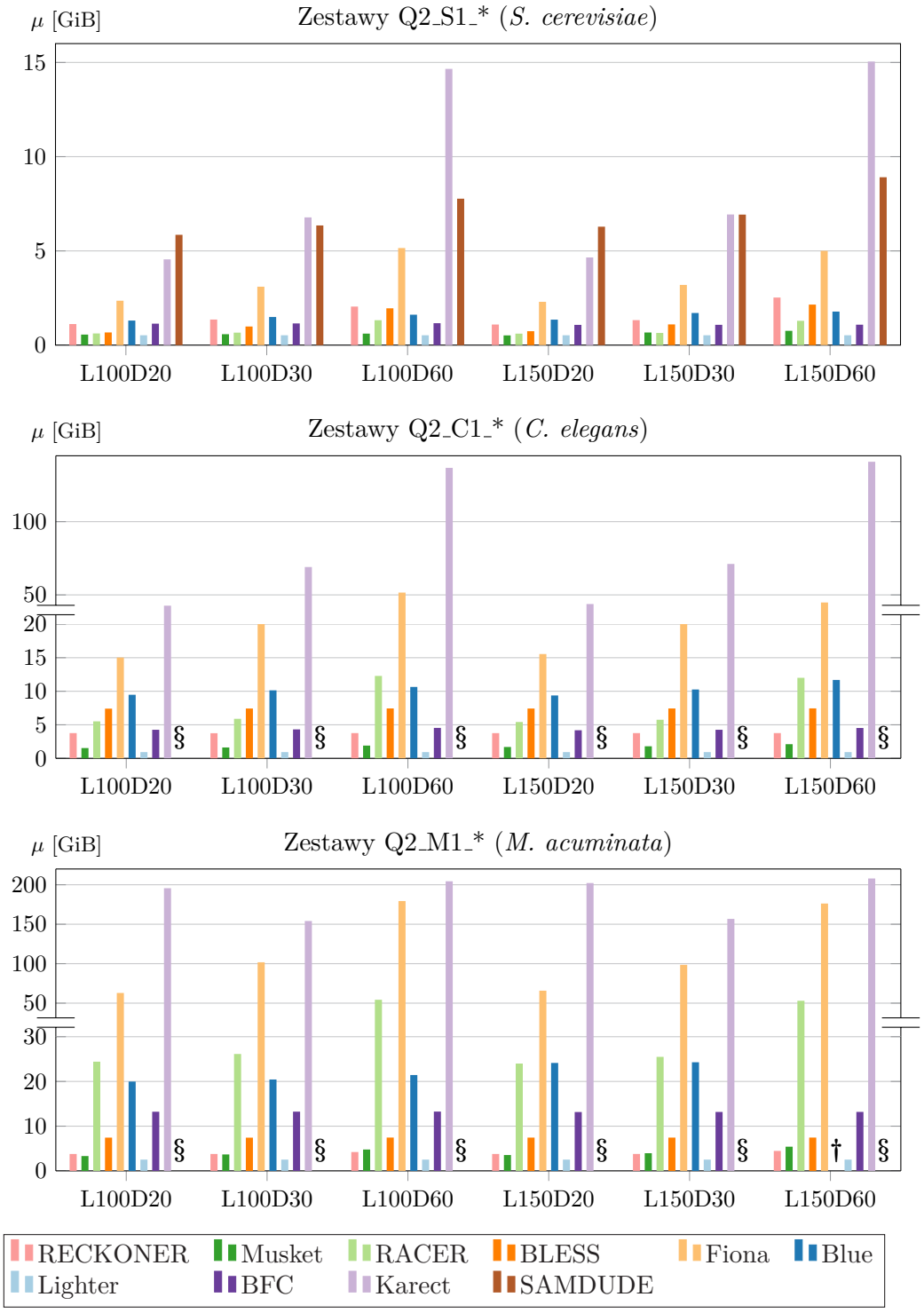
Rysunek C.14: Wpływ korekcji na wartość ξ_{sens} detekcji wariantów odczytów z zestawów H5_* (*H. sapiens*), różna głębokość sekwencjonowania — Syndip



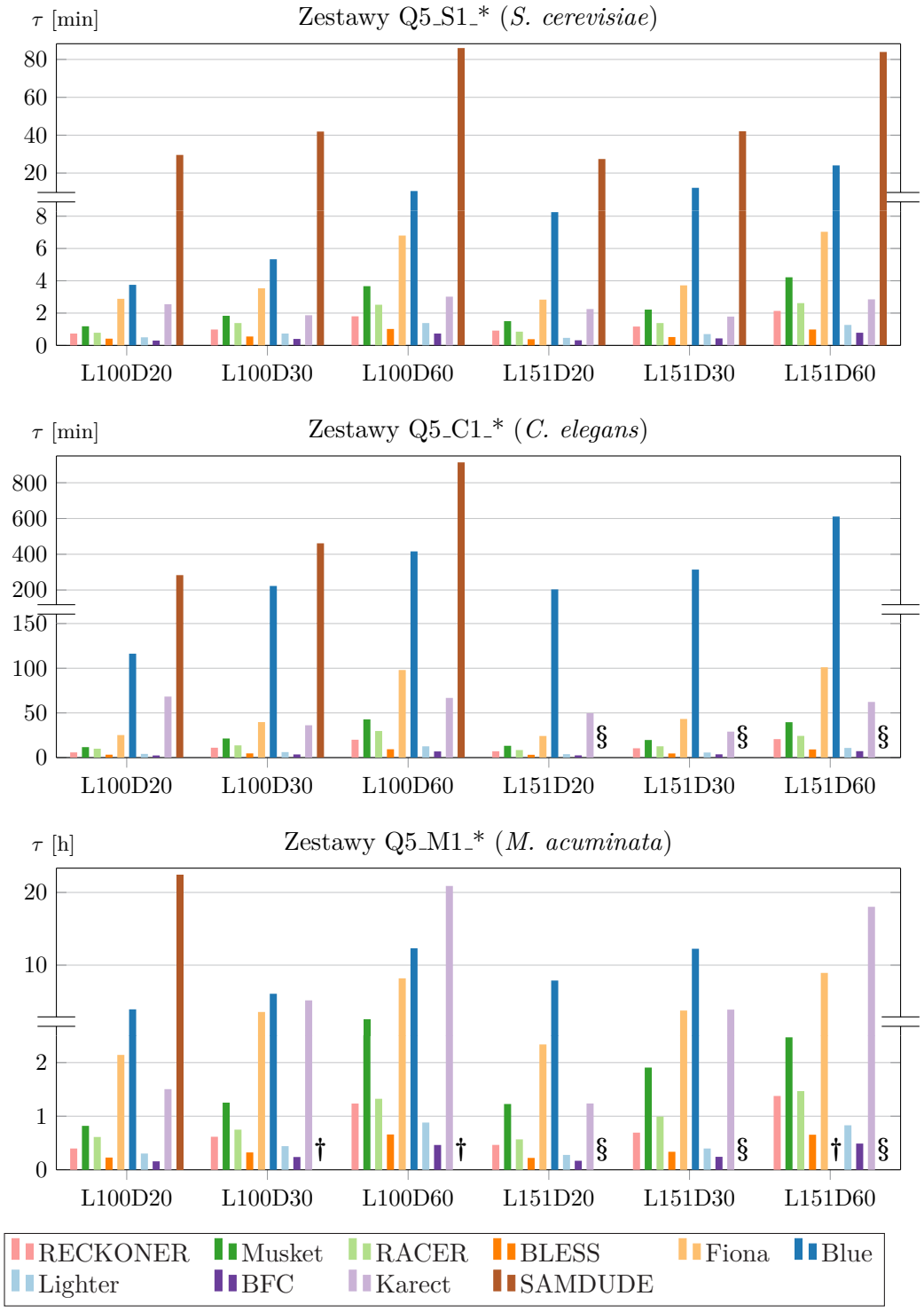
Rysunek C.15: Wpływ korekcji na wartość ξ_{prec} detekcji wariantów odczytów z zestawów H5_* (*H. sapiens*), różna głębokość sekwencjonowania — Syndip



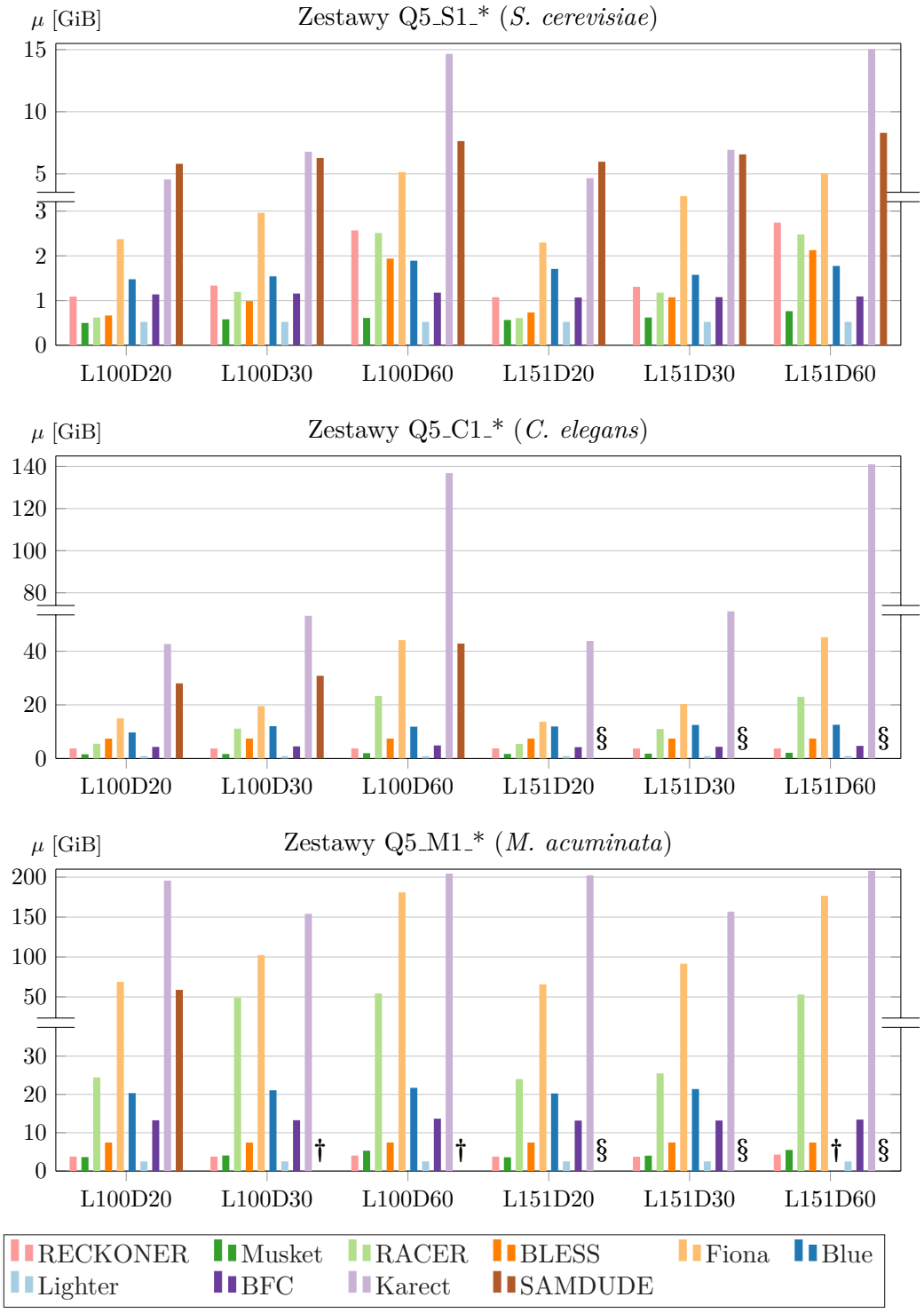
Rysunek C.16: Czas korekcji odczytów symulowanych uproszczoną metodą Quake, $p_{\text{mean}} = 2\%$



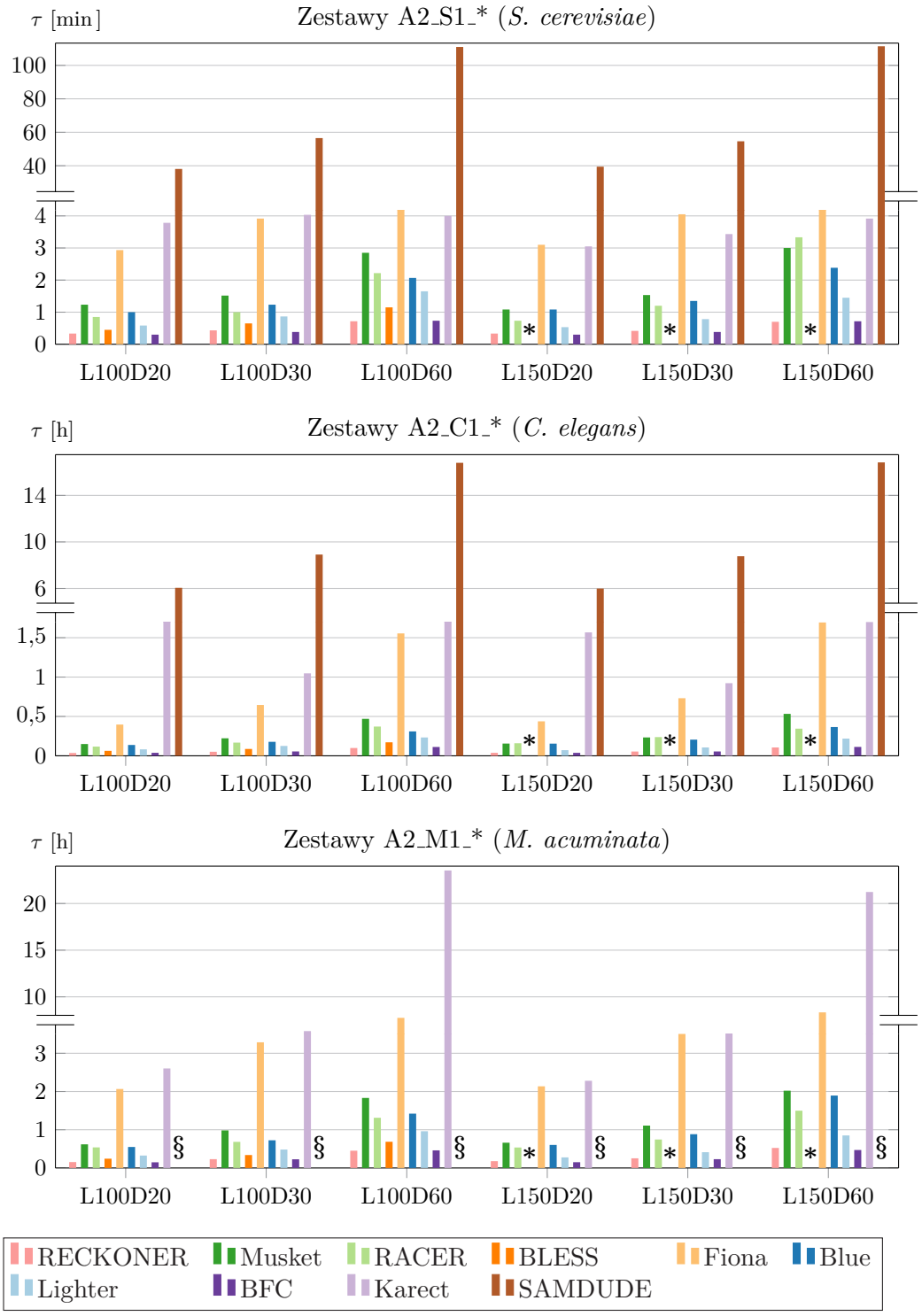
Rysunek C.17: Zapotrzebowanie na pamięć korekcji odczytów symulowanych uproszczoną metodą Quake, $p_{\text{mean}} = 2\%$



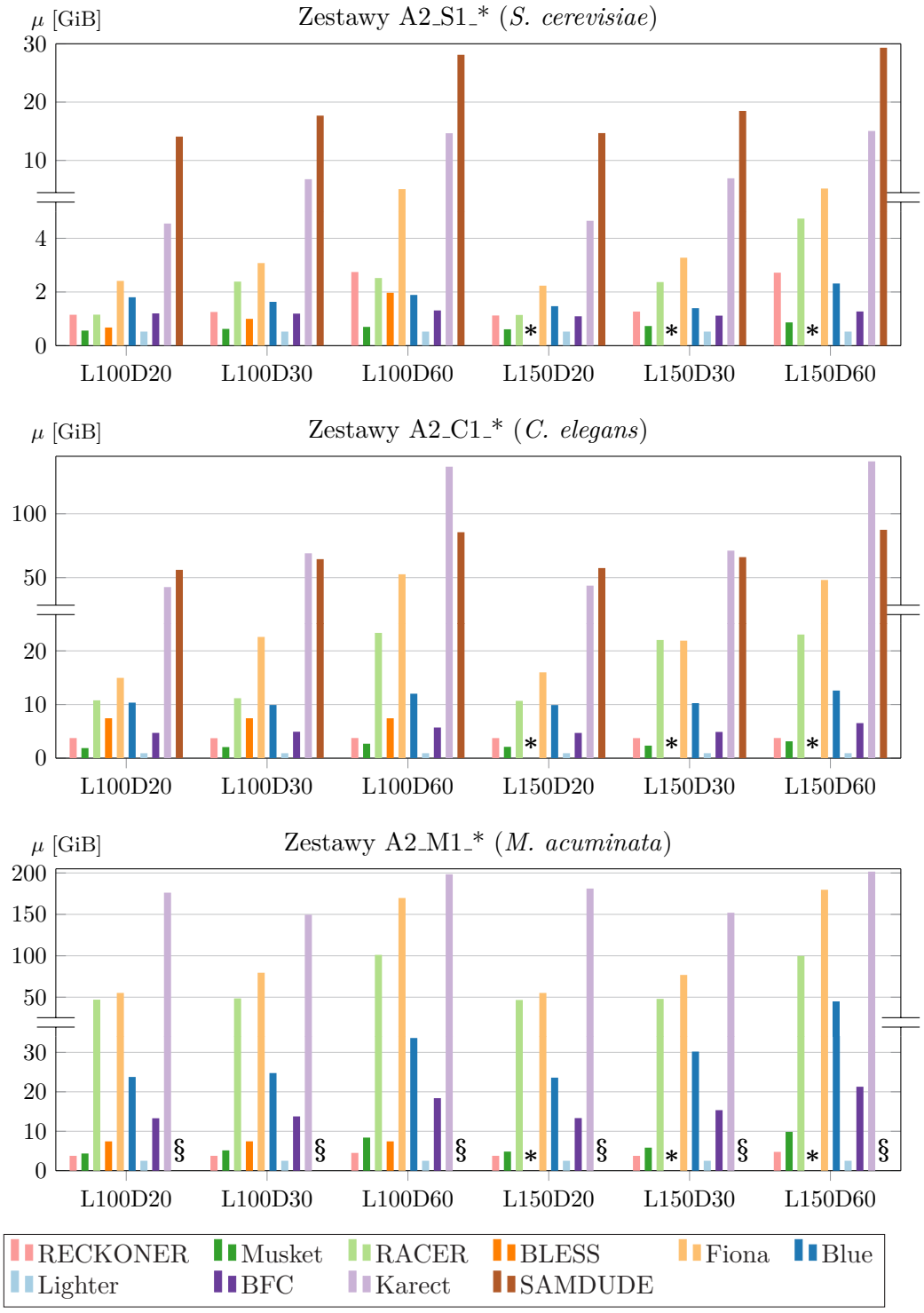
Rysunek C.18: Czas korekcji odczytów symulowanych uproszczoną metodą Quake, $p_{\text{mean}} = 5\%$



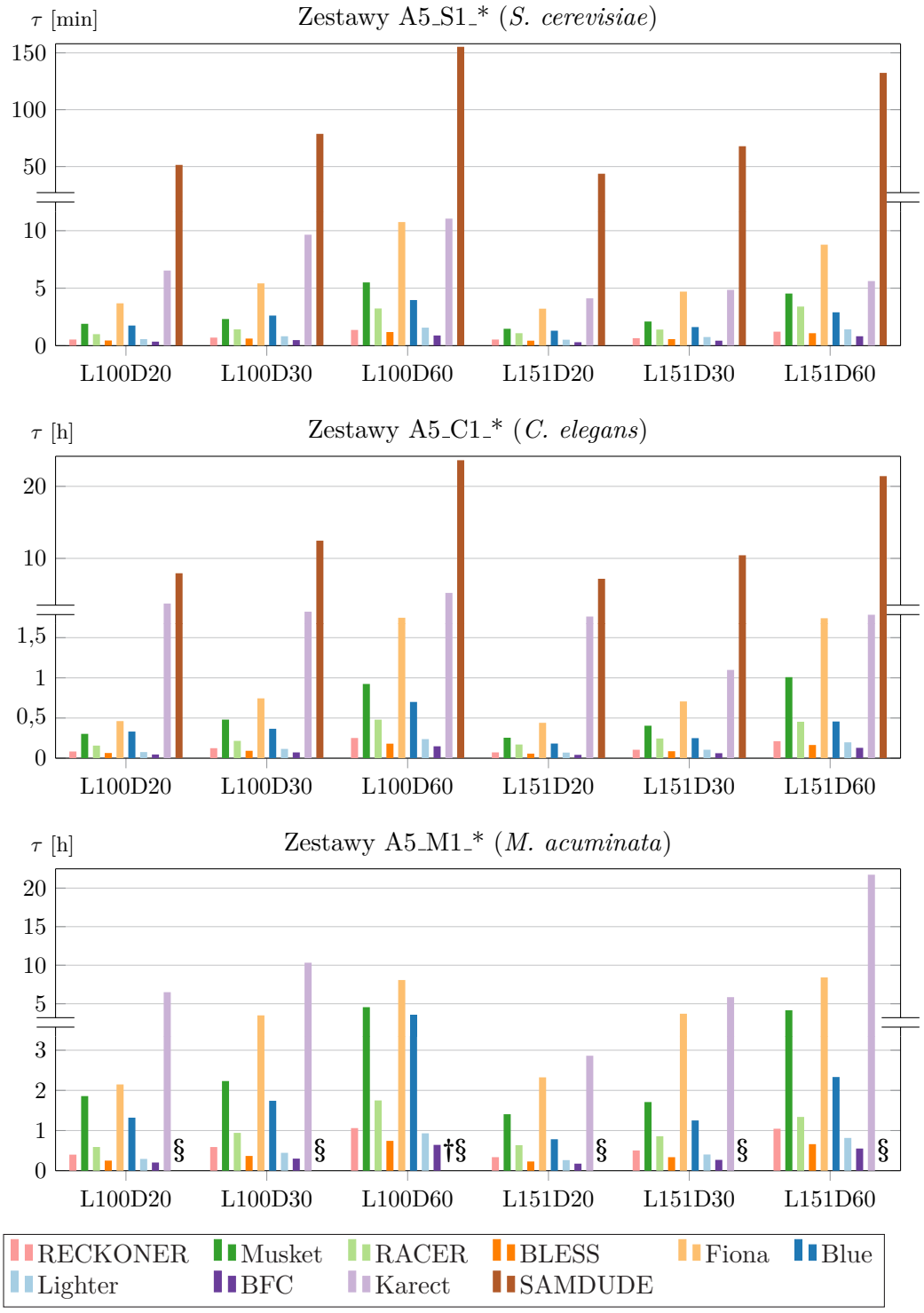
Rysunek C.19: Zapotrzebowanie na pamięć korekcji odczytów symulowanych uproszczoną metodą Quake, $p_{\text{mean}} = 5\%$



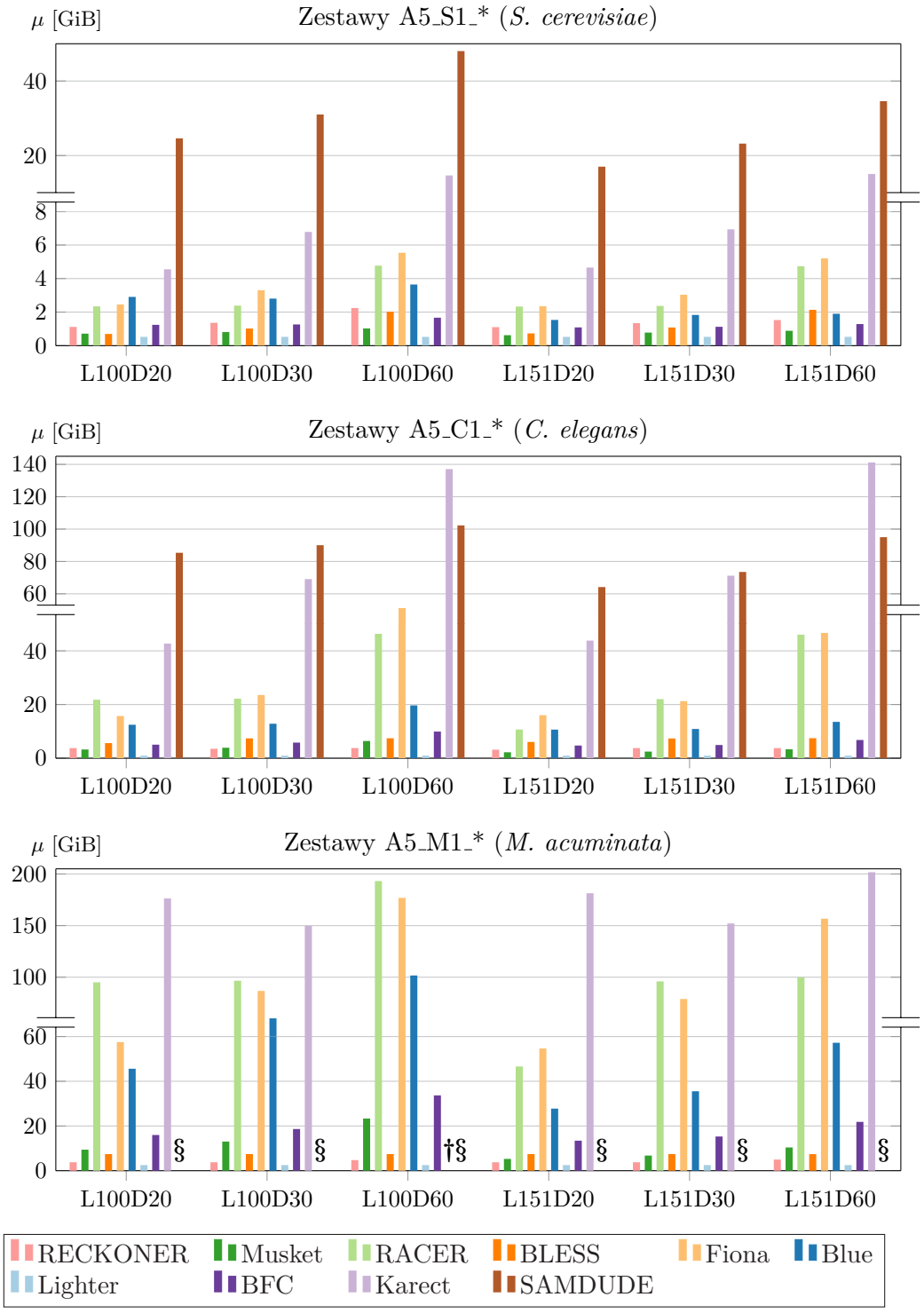
Rysunek C.20: Czas korekcji odczytów dla odczytów symulowanych narzędziem ART, $p_{\text{mean}} = 2\%$



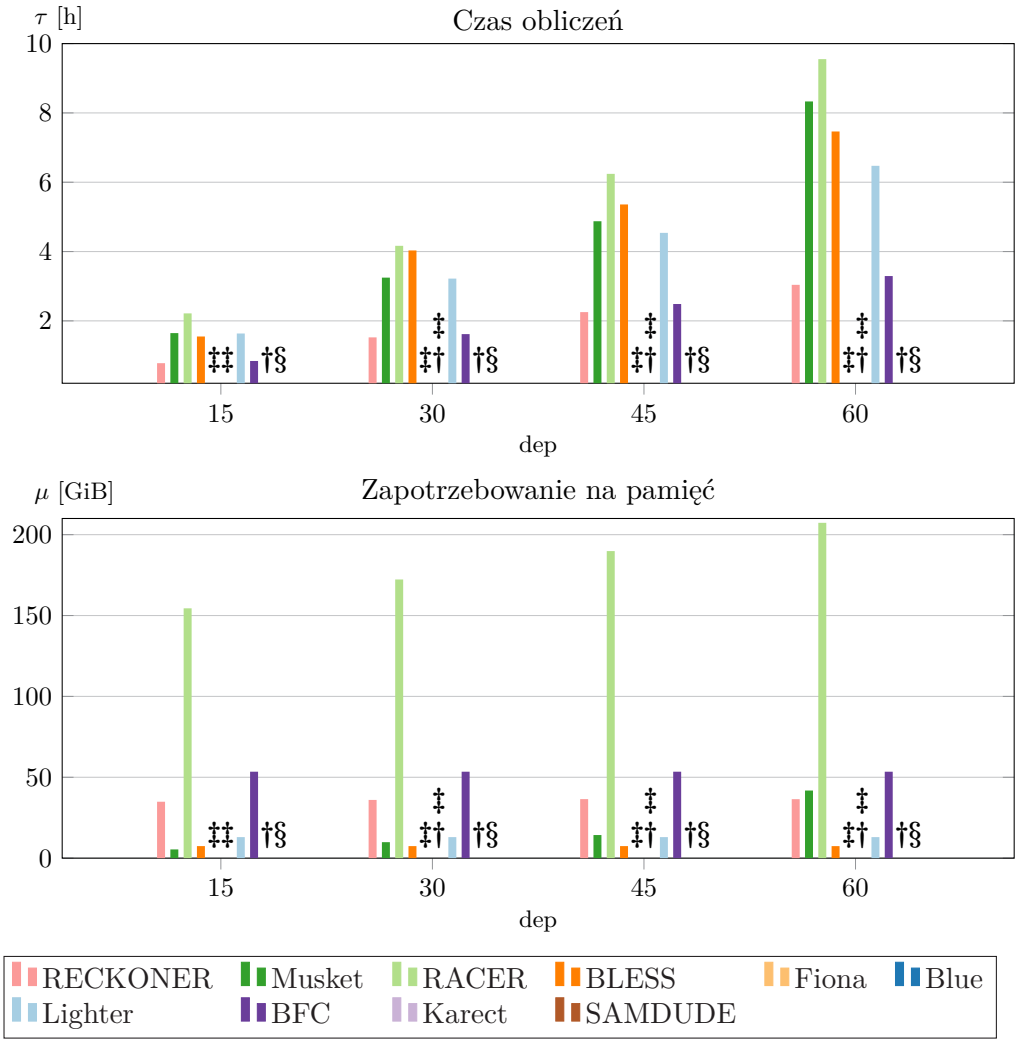
Rysunek C.21: Zapotrzebowanie na pamięć korekcji odczytów symulowanych narzędziem ART, $p_{\text{mean}} = 2\%$



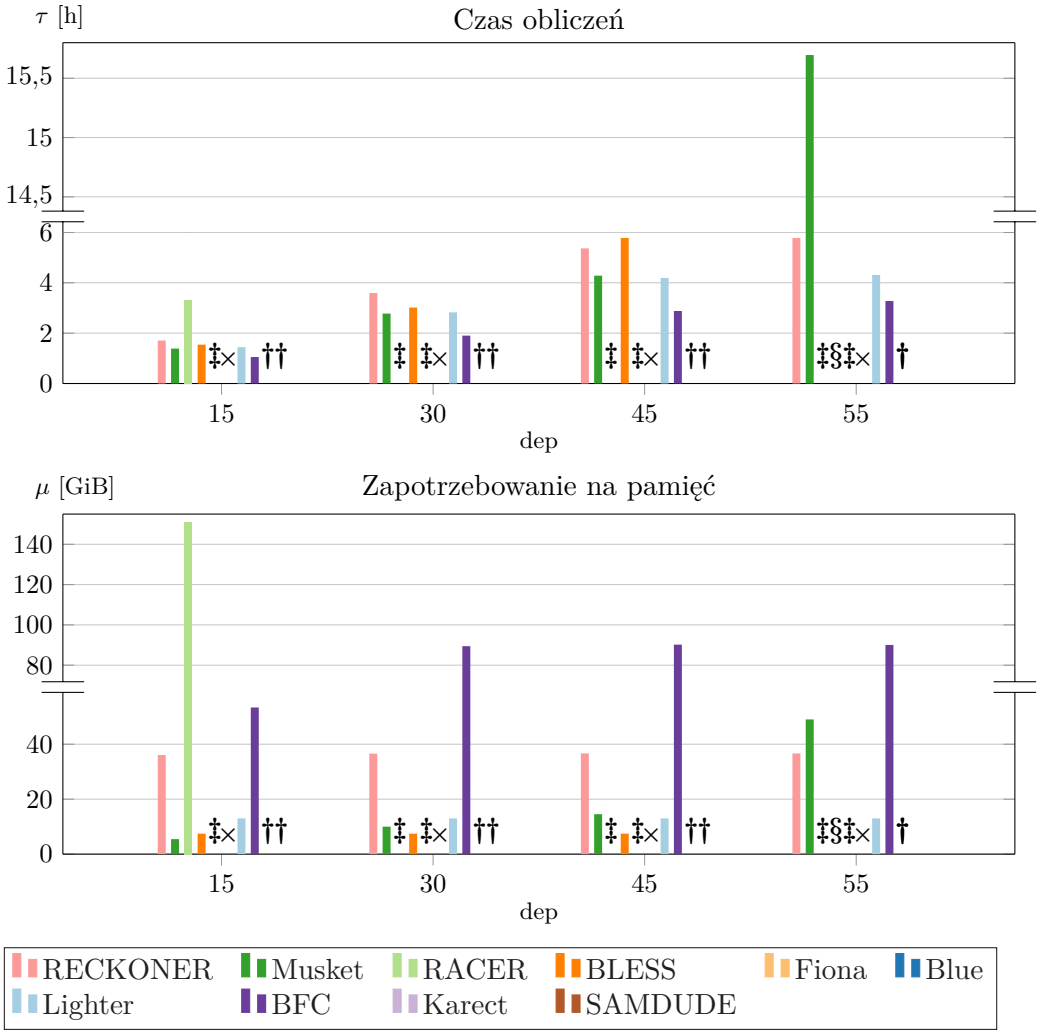
Rysunek C.22: Czas korekcji odczytów symulowanych narzędziem ART, $p_{\text{mean}} = 5\%$



Rysunek C.23: Zapotrzebowanie na pamięć korekcji odczytów symulowanych narzędziem ART, $p_{\text{mean}} = 5\%$



Rysunek C.24: Zapotrzebowanie na zasoby korekcji odczytów z zestawów H1_15, H1–2_30, H1–3_45, H1–4_60, różna głębokość sekwencjonowania



Rysunek C.25: Zapotrzebowanie na zasoby korekcji odczytów z zestawów H5_*, różna głębokość sekwencjonowania

Spis symboli i skrótów

Symbol	Opis
$\emptyset$	zbiór pusty
$\varnothing$	pusta wartość lub pusty symbol
$\ominus$	element krotki nie ulegający zmianie podczas przypisania jej nowej wartości
$\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}$	nukleotyd: adeninowy, cytozynowy, guaninowy, tyminowy
N	nieznany nukleotyd
$\underline{\mathsf{A}}$	nukleotyd komplementarny do nukleotydu adeninowego
$\widehat{\cdot}$	estymacja wartości, zwykle parametr algorytmu
$\overline{\overline{o_i}}$	koniec operacji o_i
$\underline{\underline{o_i}}$	początek operacji o_i
$\lvert\cdot\rvert$	moc zbioru, długość ciągu, długość sekwencji symboli, długość regionu
$\overset{a}{S}$	prefiks sekwencji S długości a
$\underset{a}{S}$	sufiks sekwencji S długości a
pz	para zasad (jednostka długości cząsteczki DNA)
z	zasada (jednostka ilości informacji genetycznej)
$a, b, \mathfrak{a}, \mathfrak{b}, a_i, b_i$	indeksy symboli sekwencji
c, c'	dowolny symbol
$\mathfrak{c}, \mathfrak{c}_i$	chromosom referencyjny
d	liczba funkcji mieszających filtru Blooma
$d_{\max}$	maksymalna odległość Hamminga (PREMIER)
d_{e}	odległość edycyjna odczytu i fragmentu genomu
e	element dowolnego zbioru
$f(\cdot)$	dowolna, określona funkcja
$f_{\mathrm{c}}(\cdot)$	funkcja modelująca symbol błędu substytucji
$f_{\mathrm{c\_ins}}(\cdot)$	funkcja modelująca symbol błędu insercji
$f_{\mathrm{del}}(\cdot)$	funkcja modelująca prawdopodobieństwo wystąpienia błędu delecji

Symbol	Opis
$f_{\text{ins}}(\cdot)$	funkcja modelująca prawdopodobieństwo wystąpienia błędu insercji
$f_{\text{subst}}(\cdot)$	funkcja modelująca prawdopodobieństwo wystąpienia błędu substytucji
$h(\cdot)$, $h(\cdot,\cdot)$, $h'(\cdot)$, $h_i(\cdot)$	funkcje mieszające
i, j, ι	indeksy elementów zbiorów lub iteracji
k	długość oligomeru (k-meru)
k'	długość oligomeru, gdy $k' < k$ (KMC, Blue)
k''	długość oligomeru, gdy $k'' > k$ (KMC, RECKONER)
$\bar{k}$	moc zbioru k-merów o zadanym prefiksie nad alfabetem Σ
$\bar{k}_{\text{N}}$	moc zbioru k-merów nad alfabetem Σ_N
$k_{\min}$	minimalna długość oligomeru (Fiona)
$k_{\max}$	minimalna długość oligomeru (Fiona)
k_{best}	długość oligomeru, dla którego uzyskano najlepszą ocenę algorytmu
k_{pred}	wyznaczona długość oligomeru dla algorytmu RECKONER
ℓ	długość sekwencji nukleotydowej
ℓ'	długość bezbłędnego odczytu sekwencjonowania
ℓ^*	długość zmodyfikowanego odczytu sekwencjonowania
$\bar{\ell}$	średnia długość sekwencji nukleotydowych (odczytów)
$\ell_{\mathcal{G}}$	długość genomu referencyjnego
ℓ_{Δ}	zmiana długości odczytu
m	moc podzbioru lub długość podciągu
$\bar{m}_{\text{un}}$	liczba unikalnych oligomerów, których długość jest równa długości wzorca m (tablice Gk)
n	moc określonego zbioru, długość ciągu, liczba operacji, rozmiar problemu lub długość startera
n_{LUT}	wewnętrzny parametr algorytmu KMC, wywierający wpływ na jego złożoność obliczeniową
$\mathfrak{n}$	liczba odczytów, liczba sekwencji zawartych w strukturze danych (tablice Gk)
$\mathfrak{n}_{\text{kontig}}$	liczba kontigów
$n_{\max}$	maksymalna liczba iteracji algorytmu
n_{ind}	liczba korekcji błędów typu indel w ścieżce korekcji
n_{ψ}	długość błędnego regionu (BLESS, RECKONER)
$n_{\psi,\text{mx\_ind}}$	łączna liczba korekcji, które można jeszcze wprowadzić w regionie (RECKONER)
o_i	operacja (krok algorytmu)
p	współczynnik błędu

Symbol	Opis
$p(q)$	współczynnik błędu odpowiadający współczynnikowi jakości q
$\mathfrak{p}$	sekwencja współczynników błędów
p_{del}	prawdopodobieństwo wystąpienia błędu delecji
p_{ins}	prawdopodobieństwo wystąpienia błędu insercji
$p_{\max}$	maksymalna wartość rozkładu prawdopodobieństwa (SAMDUDE)
p_{mean}	średnie prawdopodobieństwo błędu symbolu (substytucji) odczytu
p_{FP}	prawdopodobieństwo nieprawidłowej odpowiedzi twierdzącej filtru Blooma
p_α	prawdopodobieństwo zliczenia k-meru (Lighter)
p_i	i-ty wątek podległy (roboczy — RECKONER)
p_{m}	główny wątek (RECKONER)
p_{r}	wątek odczytujący (nadrzędny — RECKONER)
p_{w}	wątek zapisujący (RECKONER)
q	współczynnik jakości
$q(p)$	współczynnik jakości odpowiadający współczynnikowi błędu p
$\mathfrak{q}, \mathfrak{q}_i$	sekwencja współczynników jakości
$\mathfrak{q}^*$	sekwencja współczynników jakości o zmodyfikowanej długości
$q_{\min}$	graniczna wartość współczynnika jakości (92 percentyl) (Trowel)
$\mathfrak{r}, \mathfrak{r}_i$	odczyt z sekwencjonowania
$\mathfrak{r}'$	bezbłędna sekwencja odczytu z sekwencjonowania
$\mathfrak{r}^*$	zmodyfikowany odczyt z sekwencjonowania
s	inherentnie sekwencyjna część algorytmu
$(\mathfrak{s}_i)$	ciąg korekcji (RECKONER)
$t(\cdot, \cdot)$	liczba kroków obliczeniowych potrzebnych do rozwiązania konkretnej instancji danego problemu przy danej liczbie procesorów
$\mathfrak{t}(\cdot)$	moment zdarzenia
u_1, u_2	dowolne wartości klucza tablicy mieszającej
v, v_i	dowolne węzły drzewa
v_1, v_2	węzły grafu, węzły drzewa
v_{root}	korzeń drzewa
$\mathbf{v}$	wektor (SAMDUDE)
$(\mathbf{v}_\iota^{\mathrm{ind}})$	ciąg indeksów prefiksów (KMC)
$(\mathbf{v}_j^{\mathrm{map}})$	ciąg mapowania sygnatur do ciągów indeksów (KMC)
$(\mathbf{v}_i^{\mathrm{suf}})$	ciąg sufiksów (KMC)
$\mathbf{v}_{\mathrm{Q\_LOW}}$	ciąg pozycji odczytu o niskich współczynnikach jakości (RECKONER)
$\mathfrak{w}$, $\mathfrak{w}_j$, $\mathfrak{w}'$	sekwencja nukleotydowa

Symbol	Opis
x, y	dowolne wartości, w szczególności stała równania funkcji
x'	dowolna stała równania funkcji
x, x_0, x_i, y	elementy zbioru $\mathbb{X}$ lub $\mathbb{X}_i$
x_i	poszukiwany parametr
x_{lok}	lokalne minimum
x_{opt}	globalne minimum
$\mathcal{X}$	procentowa część długości sekwencji
y	podstawa logarytmu w eksperymentalnym limitowaniu liczebności k-merów
y_i	realizacja zmiennej losowej
$\bar{z}$	liczba wystąpień wyszukiwanego wzorca
$\bar{z}'$	liczba sekwencji (odczytów) zawierających wzorzec (tablice Gk)
A	dowolny zbiór
$A_{\text{Q\_LOW}}$	zbiór pozycji odczytu o niskich współczynnikach jakości (BLESS)
$\mathcal{G}$	genom referencyjny
$\mathcal{K}_{\text{cut}}$	spektrum k-merów
$\mathcal{K}''_1$	spektrum k''-merów
$\mathcal{K}_{\text{N}}$	zbiór k-merów nad alfabetem Σ_N
$\mathcal{K}_{\text{un}}$	zbiór unikalnych k-merów
$\mathcal{K}_{\pi}$	zbiór k-merów wykorzystywanych w ocenie ścieżki korekcji π
$\mathscr{K}$	zbiór par k-merów z liczebnościami
$\mathbf{M}$	macierz (Musket, SAMDUDE)
$N(\cdot)$	sąsiedztwo rozwiązania
$\mathbb{N}_+$	zbiór liczb naturalnych dodatnich
$P(\cdot)$	rozkład prawdopodobieństwa zmiennej losowej
$P\mathscr{P}$	ciąg ciągów ścieżek korekcji kolejnych regionów odczytu (RECKONER)
$P_{\text{r}}(\cdot)$	prawdopodobieństwo pewnego zdarzenia
$P_{\text{r}}(\cdot\|\cdot)$	prawdopodobieństwo warunkowe pewnego zdarzenia
$P'(\cdot)$	rozkład prawdopodobieństwa emisji ukrytego modelu Markowa
$\mathcal{P}, \mathcal{P}'$	procesy (instancje programu)
Q	kolejka priorytetowa (BFC)
Q_n	zbiór dopuszczalnych współczynników jakości mocy n
Q_{r}	kolejka fragmentów pliku wejściowego (RECKONER)
Q_{w}	kolejka fragmentów pliku wyjściowego (RECKONER)
R_j	ograniczenie rozwiązania

Symbol	Opis
$\mathbb{R}_+$	zbiór liczb rzeczywistych dodatnich
$\mathcal{R}$	zbiór par: odczytów z sekwencjonowania i współczynników błędów
$\mathscr{R}$	zbiór ograniczeń rozwiązań
S	dowolna sekwencja symboli
$\mathfrak{S}$	ciąg przekształceń odczytu (model błędu odczytu)
S'	wyszukiwana sekwencja wzorcowa
$T(\cdot,\cdot)$	pesymistyczna czasowa złożoność obliczeniowa
$T^*(\cdot,\cdot)$	pesymistyczna czasowa złożoność pewnego najlepszego znanego algorytmu sekwencyjnego
$T_{\mathrm{kmc}}(\cdot)$	złożoność obliczeniowa zapytania do bazy KMC (RECKONER)
$T_{\mathrm{hash}}(\cdot)$	złożoność obliczeniowa funkcji mieszającej (BLESS)
$T_{\mathrm{speedup}}(\cdot,\cdot)$	przyspieszenie algorytmu
$T_{\mathrm{B}}(\cdot,\cdot,\cdot,\cdot)$	pesymistyczna złożoność obliczeniowa korekcji odczytu z uwzględnieniem wykonania zapytań do filtru Blooma (BLESS)
$T'_{\mathrm{B}}(\cdot,\cdot,\cdot,\cdot)$	pesymistyczna złożoność obliczeniowa korekcji odczytu (BLESS)
$T_{\mathrm{B1b}}(\cdot)$	pesymistyczna złożoność obliczeniowa siłowej korekcji pierwszego k-meru (BLESS)
$T_{\mathrm{B1ble}}(\cdot,\cdot)$	pesymistyczna czasowa złożoność rozszerzania w lewo wszystkich ścieżek uzyskanej w korekcji metodą modyfikacji kolejnych symboli pierwszego k-meru (BLESS)
$T_{\mathrm{B1cle}}(\cdot,\cdot)$	pesymistyczna czasowa złożoność rozszerzania w lewo wszystkich ścieżek uzyskanej w metodzie siłowej korekcji pierwszego k-meru (BLESS)
$T_{\mathrm{B1c}}(\cdot)$	pesymistyczna złożoność obliczeniowa modyfikacji kolejnych symboli pierwszego k-meru (BLESS)
$T_{\mathrm{Bb}}(\cdot,\cdot,\cdot,\cdot)$	pesymistyczna złożoność obliczeniowa korekcji całego odczytu, w przypadku gdy jest wykonywana korekcja pierwszego k-meru metodą siłową (BLESS)
$T_{\mathrm{Bc}}(\cdot,\cdot,\cdot,\cdot)$	pesymistyczna złożoność obliczeniowa korekcji całego odczytu, w przypadku gdy jest wykonywana korekcja pierwszego k-meru metodą siłową oraz metodą modyfikacji kolejnych symboli (BLESS)
$T_{\mathrm{B3'}}(\cdot)$	pesymistyczna złożoność obliczeniowa korekcji w kierunku $3'$ (BLESS)
$T'_{\mathrm{B3'}}(\cdot)$	pesymistyczna złożoność obliczeniowa korekcji w kierunku $3'$ — wartość pośrednia (BLESS)
$T_{\mathrm{B3're}}(\cdot,\cdot)$	pesymistyczna złożoność obliczeniowa korekcji w kierunku $3'$ wraz z rozszerzaniem (BLESS)

Symbol	Opis
$T_{\mathrm{Ble}}(\cdot)$	pesymistyczna złożoność obliczeniowa rozszerzania jednej ścieżki korekcji w lewo (BLESS)
$T_{\mathrm{Bre}}(\cdot)$	pesymistyczna złożoność obliczeniowa rozszerzania jednej ścieżki korekcji w prawo (BLESS)
$T^{\mathrm{KMC3}}_{\text{równoległy}}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot)$	pesymistyczna złożoność obliczeniowa zliczania k-merów przy pomocy algorytmu KMC
$T'_{\mathrm{R}}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot)$	pesymistyczna złożoność obliczeniowa korekcji odczytu (RECKONER)
$T_{\mathrm{R}}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot)$	pesymistyczna złożoność obliczeniowa korekcji odczytu z uwzględnieniem wykonania zapytań do bazy k-merów (RECKONER)
$T_{\mathrm{R1b}}(\cdot)$	pesymistyczna złożoność obliczeniowa siłowej korekcji pierwszego k-meru (RECKONER)
$T_{\mathrm{R1ble}}(\cdot,\cdot,\cdot)$	pesymistyczna czasowa złożoność rozszerzania w lewo wszystkich ścieżek uzyskanej w korekcji metodą modyfikacji kolejnych symboli pierwszego k-meru (RECKONER)
$T_{\mathrm{R1ile}}(\cdot,\cdot)$	pesymistyczna czasowa złożoność rozszerzania w lewo wszystkich ścieżek uzyskanej w korekcji błędów typu indel pierwszego k-meru (RECKONER)
$T_{\mathrm{R1c}}(\cdot)$	pesymistyczna złożoność obliczeniowa modyfikacji kolejnych symboli pierwszego k-meru (RECKONER)
$T_{\mathrm{R1i}}(\cdot)$	pesymistyczna złożoność obliczeniowa korekcji błędów typu indel pierwszego k-meru (RECKONER)
$T_{\mathrm{R3'}}(\cdot,\cdot,\cdot)$	pesymistyczna złożoność obliczeniowa korekcji w kierunku $3'$ (RECKONER)
$T'_{\mathrm{R3'}}(\cdot)$	pesymistyczna złożoność obliczeniowa korekcji w kierunku $3'$ — wartość pośrednia (RECKONER)
$T''_{\mathrm{R3'}}(\cdot,\cdot)$	pesymistyczna złożoność obliczeniowa korekcji w kierunku $3'$ — wartość, gdy długość regionu i liczba dopuszczalnych korekcji błędów typu indel są traktowane rozłącznie (RECKONER)
$T_{\mathrm{R3're}}(\cdot,\cdot,\cdot,\cdot,\cdot)$	pesymistyczna złożoność obliczeniowa korekcji w kierunku $3'$ wraz z rozszerzaniem (RECKONER)
$T_{\mathrm{Rb}}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot)$	pesymistyczna złożoność obliczeniowa korekcji całego odczytu, w przypadku gdy jest wykonywana korekcja pierwszego k-meru metodą siłową (RECKONER)

Symbol	Opis
$T_{\mathrm{Ri}}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot)$	pesymistyczna złożoność obliczeniowa korekcji całego odczytu, w przypadku gdy jest wykonywana korekcja pierwszego k-meru metodą siłową, metodą modyfikacji kolejnych symboli oraz gdy następuje korekcja błędów typu indel (RECKONER)
$T_{\mathrm{Rle}}(\cdot)$	pesymistyczna złożoność obliczeniowa rozszerzania jednej ścieżki korekcji w lewo (RECKONER)
$T_{\mathrm{Rre}}(\cdot)$	pesymistyczna złożoność obliczeniowa rozszerzania jednej ścieżki korekcji w prawo (RECKONER)
$\mathbb{U}$	przestrzeń możliwych wartości klucza elementu tablicy mieszającej
V_{tree}	dowolne drzewo
V_i	poddrzewo wybranego drzewa
X	wektor poszukiwanych parametrów
X_j	jeden z wielu wektorów poszukiwanych parametrów
Y_i	zmienna losowa
Y_i^e	emisja ukrytego modelu Markowa
$\mathbb{X}$	przestrzeń wartości argumentu algorytmu
$\mathbb{X}_i$	przestrzeń wartości i-tego parametru rozwiązania
$\mathcal{W}$	zbiór sekwencji nukleotydowych
α	współczynnik wypełnienia tablicy mieszającej
β_{bias}	współczynnik funkcji oceny ścieżki korekcji (PREMIER)
γ	współczynniki funkcji kary (PREMIER)
δ	zapotrzebowanie algorytmu na przestrzeń dyskową
$\delta_{\max}$	maksymalne zapotrzebowanie na przestrzeń dyskową (DSK)
δ_{step}	szybkość relaksacji minimalnej dopuszczalnej miary Fano (PREMIER)
$\bar{\delta}$	liczba plików tymczasowych (DSK, BLESS)
ϵ	dowolna nieujemna stała
η, η_i	liczebność wybranego k-meru
$\eta(\kappa)$	liczebność k-meru κ
η_{cut}	próg obcięcia
$\eta_{\max}$	drugie maksimum histogramu liczebności k-merów
$\eta_{\mathrm{max\_ext}}$	największa liczebność pierwszego k-meru rozszerzającego region (RECKONER)
$\eta_{\min}$	pierwsze minimum histogramu liczebności k-merów
η''_{cut}	próg obcięcia zbioru k''-merów
$\eta_{\mathrm{MX\_CUT}}$	maksymalny próg obcięcia (RECKONER)
θ_{ext}	liczba pozycji rozszerzających dla bieżącej ścieżki (BLESS, RECKONER)

Symbol	Opis
$\theta_{\text{first\_mod}}$	najmniejszy indeks modyfikacji w ścieżce (BLESS, RECKONER)
$\theta_{\text{last\_chck}}$	pozycja ostatniego k-meru rozszerzającego region wewnętrzny (RECKONER)
$\theta_{\text{last\_kmer}}$	pozycja ostatniego k-meru w regionie podczas rozszerzania (RECKONER)
$\theta_{\text{last\_mod}}$	największy indeks modyfikacji w ścieżce (BLESS, RECKONER)
θ_{lq}	liczba pozycji dla algorytmu siłowego korekcji pierwszego k-meru (BLESS, RECKONER)
$\theta_{\text{mx\_chck}}$	pozostała liczba zmian do zweryfikowania w regionie (RECKONER)
$\theta_{\text{mx\_ind}}$	pozostała liczba dopuszczalnych korekcji błędów typu indel w regionie (RECKONER)
$\theta_{\text{mx\_untrst}}$	maksymalna liczba pozostałych dopuszczalnych błędnych k-merów w regionie lub k-merów rozszerzających (RECKONER)
$\theta_{\text{reg\_beg}}$	indeks początku regionu (RECKONER)
$\theta_{\text{COV\_WEIGHT}}$	waga k-meru pokrywającego region (RECKONER)
$\theta_{\text{EXT\_WEIGHT}}$	waga pierwszego k-meru rozszerzającego (RECKONER)
θ_{FP}	przyjęta liczba k-merów w błędnym obszarze uznanych za prawidłowe na skutek błędów filtru Blooma (BLESS)
$\theta_{\text{IND\_PROB}}$	przyjęte prawdopodobieństwo błędu typu indel (RECKONER)
$\theta_{\text{INS\_RATE}}$	ocena ścieżki korekcji regionu, zawierającej wyłącznie eliminację błędów insercji (RECKONER)
$\theta_{\text{LONG\_RATIO}}$	iloraz długości k''-meru do długości k-meru (RECKONER)
$\theta_{\text{MIN}\Delta}$	minimalna różnica oceny między najlepszą a drugą w kolejności ścieżką korekcji (BLESS)
$\theta_{\text{MN\_ERR}}$	minimalna długość błędnego regionu (BLESS, RECKONER)
$\theta_{\text{MN\_IND\_DIST}}$	minimalna odległość między korekcjami błędu typu insercja i typu delecja (RECKONER)
$\theta_{\text{MN\_SOLID}}$	minimalna długość prawidłowego regionu (BLESS, RECKONER)
$\theta_{\text{MX\_ADJ}}$	maksymalna liczba pozycji regionu sprawdzanych na obecność symboli niskiej jakości (BLESS, RECKONER)
$\theta_{\text{MX\_CHCK}}$	maksymalna liczba zmian weryfikowanych w regionie (RECKONER)

Symbol	Opis
$\theta_{\text{MX\_CHCK\_IND\_POS}}$	Liczba pozycji odczytu sprawdzanych w celu znalezienia $\theta_{\text{MX\_CHCK\_IND\_UNTRST}}$ korekcji (RECKONER)
$\theta_{\text{MX\_IND\_UNTRST}}$	Liczba skorygowanych błędów typu substytucja, wskazujących błąd typu indel (RECKONER)
$\theta_{\text{MX\_E}}$	maksymalna liczba pozycji rozszerzających region (BLESS, RECKONER)
$\theta_{\text{MX\_FIRST\_PTHS}}$	maksymalna liczba ścieżek uzyskanych w korekcji siłowej pierwszego k-meru (RECKONER)
$\theta_{\text{MX\_FIRST\_RES}}$	maksymalna liczba ścieżek uzyskanych w wyniku korekcji pierwszego k-meru (RECKONER)
$\theta_{\text{MX\_IND}}$	maksymalna liczba błędów typu indel skorygowanych w regionie (RECKONER)
$\theta_{\text{MX\_IND\_RATE}}$	Maksymalny stosunek liczby błędów typu indel skorygowanych w regionie do długości tego regionu (RECKONER)
$\theta_{\text{MX\_LQ}}$	maksymalna liczba symboli w korekcji pierwszego k-meru metodą siłową (BLESS, RECKONER)
$\theta_{\text{MX\_NON\_UNTRST}}$	maksymalna liczba ścieżek korekcji regionu (RECKONER)
$\theta_{\text{MX\_PTH}}$	maksymalna liczba symboli w korekcji pierwszego k-meru metodą siłową (BLESS, RECKONER)
$\theta_{\text{Q\_LOW}}$	górne ograniczenie wartości niskich współczynników jakości (BLESS, RECKONER)
κ	dowolny oligomer
κ_1, κ_2	konkretne oligomery
κ^*	zmodyfikowany oligomer
κ_{can}	dowolny k-mer kanoniczny
κ_i	wybrany k-mer
κ_{L}, κ_{R}	lewy i prawy k-mer kontekstu symbolu (SAMDUDE)
λ	współczynniki funkcji kary (PREMIER)
μ	zapotrzebowanie algorytmu na pamięć
μ_{max}	maksymalne zapotrzebowanie na RAM (DSK, KMC)
ξ_{map1}	część odczytów zmapowanych do genomu jednokrotnie
$\xi_{\text{map1+}}$	część odczytów zmapowanych do genomu wielokrotnie
ξ_{cov}	pokrycie genomu kontigami
ξ_{del}	stosunek liczby różnic typu delecja między odczytem a odpowiednim podsłowem genomu do liczby odczytów

Symbol	Opis
ξ_{ins}	stosunek liczby różnic typu insercja między odczytem a odpowiednim podsłowem genomu do liczby odczytów
ξ_{misasm}	liczba błędnych asemblacji
$\xi_{\mathrm{ndist}}(\cdot)$	procentowa część odczytów o zadanej odległości edycyjnej od genomu
ξ_{sens}	czułość korekcji lub detekcji wariantów
ξ_{prec}	precyzja korekcji lub detekcji wariantów
ξ_{gain}	zysk korekcji
ξ_{F1}	miara F1 jakości detekcji wariantów
π	ścieżka korekcji (RECKONER)
$\boldsymbol{\pi}$	ciąg ścieżek korekcji (RECKONER)
$\pi_{\mathrm{B1b}}(\cdot)$	maksymalna liczba ścieżek uzyskanych w wyniku korekcji pierwszego k-meru metodą siłową (BLESS)
$\pi_{\mathrm{B1c}}(\cdot)$	maksymalna liczba ścieżek uzyskanych w wyniku korekcji pierwszego k-meru metodą modyfikacji kolejnych symboli (BLESS)
$\pi_{\mathrm{B1c}}(\cdot,\cdot)$	liczba ścieżek pełniących rolę pierwszego zmodyfikowanego symbolu w wyniku korekcji pierwszego k-meru metodą modyfikacji kolejnych symboli (BLESS)
$\pi_{\mathrm{B3'}}(\cdot)$	maksymalna liczba ścieżek uzyskanych w wyniku korekcji regionu w kierunku $3'$ (BLESS)
$\pi_{\mathrm{R1b}}(\cdot)$	maksymalna liczba ścieżek uzyskanych w wyniku korekcji pierwszego k-meru metodą siłową (RECKONER)
$\pi_{\mathrm{R1c}}(\cdot)$	maksymalna liczba ścieżek uzyskanych w wyniku korekcji pierwszego k-meru metodą modyfikacji kolejnych symboli (RECKONER)
$\pi_{\mathrm{R1c}}(\cdot,\cdot)$	liczba ścieżek pełniących rolę pierwszego zmodyfikowanego symbolu w wyniku korekcji pierwszego k-meru metodą modyfikacji kolejnych symboli (RECKONER)
$\pi_{\mathrm{R1i}}(\cdot)$	maksymalna liczba ścieżek uzyskanych w wyniku korekcji błędów typu indel pierwszego k-meru (RECKONER)
$\pi_{\mathrm{R1i}}(\cdot,\cdot)$	liczba ścieżek pełniących rolę pierwszego zmodyfikowanego symbolu w wyniku korekcji błędów typu indel pierwszego k-meru (RECKONER)
$\pi_{\mathrm{R3'}}(\cdot,\cdot,\cdot)$	maksymalna liczba ścieżek uzyskanych w wyniku korekcji regionu w kierunku $3'$ (RECKONER)
$\pi'_{\mathrm{R3'}}(\cdot)$	maksymalna liczba ścieżek uzyskanych w wyniku korekcji regionu w kierunku $3'$ — wartość pośrednia (RECKONER)

Symbol	Opis
χ	maksymalna wartość χ' (KMC)
χ'	różnica długości fragmentu super k-meru oraz k-meru χ (KMC)
ρ	liczba procesorów lub wątków procesu
ρ_{node}	liczba węzłów obliczeniowych (BLESS)
ρ_{c}	liczba wątków roboczych (RECKONER)
ς	semafor
ς_{r}	semafor odczytu pliku (RECKONER)
ς_{w}	semafor zapisu pliku (RECKONER)
τ	czas obliczeń algorytmu
τ_ρ	czas obliczeń algorytmu uruchomionego w konfiguracji pracy z ρ wątkami
τ_1	czas obliczeń algorytmu uruchomionego w konfiguracji pracy z jednym wątkiem
ψ	dowolny region odczytu (BLESS, RECKONER)
ψ_{p}	poprzedni region odczytu (BLESS, RECKONER)
Ξ_{p}	stosunek prawdopodobieństw (Quake)
Π	ścieżka korekcji (BLESS)
Π_{n}	nowa ścieżka korekcji (BLESS)
$\mathbf{\Pi}$	zbiór ścieżek korekcji (BLESS, RECKONER)
Σ	alfabet symboli nukleotydów
Σ_{N}	alfabet symboli nukleotydów, wliczając symbol N
Ψ_{trust}	zbiór prawidłowych regionów odczytu (BLESS, RECKONER)
Ψ_{untrust}	zbiór błędnych regionów odczytu (BLESS, RECKONER)
dep	głębokość sekwencjonowania
del	znacznik błędu typu delecja (RECKONER)
$\mathrm{dist}(\cdot,\cdot)$	odległość rozwiązań
$\mathrm{eval}(\cdot)$	funkcja celu
fin_corr	znacznik zakończenia korekcji odczytów pliku (RECKONER)
fin_input	znacznik zakończenia wczytania pliku (RECKONER)
exp_ind	znacznik potencjalnego błędu typu indel (RECKONER)
ins	znacznik błędu typu insercja (RECKONER)
long	parametr wyznaczający weryfikację regionów k''-merami
$\mathrm{prob}(\cdot)$	funkcja pomocnicza wyznaczania oceny ścieżki korekcji
sh_ns	wartość binarna dotycząca obecności krótkiego błędnego regionu (BLESS, RECKONER)
$\mathrm{rev}(\cdot)$	odwrócone dopełnienie oligomeru

Symbol	Opis
Signal$(\cdot)$	operacja zwolnienia zasobu zabezpieczonego semaforem
Wait$(\cdot)$	operacja zajęcia zasobu zabezpieczonego semaforem
L$\mathcal{X}$	liczba kontigów jako miara jakości asemblacji
LA$\mathcal{X}$	liczba kontigów jako miara jakości asemblacji, po podziale kontigów
LG$\mathcal{X}$	liczba kontigów jako miara jakości asemblacji, w odniesieniu do rozmiaru genomu
LGA$\mathcal{X}$	liczba kontigów jako miara jakości asemblacji, w odniesieniu do rozmiaru genomu, po podziale kontigów
N$\mathcal{X}$	długość kontigu jako miara jakości asemblacji
NA$\mathcal{X}$	długość kontigu jako miara jakości asemblacji, po podziale kontigów
NG$\mathcal{X}$	długość kontigu jako miara jakości asemblacji, w odniesieniu do rozmiaru genomu
NGA$\mathcal{X}$	długość kontigu jako miara jakości asemblacji, w odniesieniu do rozmiaru genomu, po podziale kontigów
TP	liczba odczytów lub symboli prawidłowo skorygowanych
FP	liczba odczytów lub symboli uszkodzonych w wyniku korekcji
FN	liczba odczytów lub symboli nieskorygowanych lub skorygowanych błędnie

Skrót	Opis
API	interfejs programistyczny aplikacji
BWT	transformata Burrowsa–Wheelera
FGS	sekwencjonowanie pierwszej generacji
MP	odczyty sparowane o dużej wzajemnej odległości
MSD	sortowanie pozycyjne od najbardziej znaczącego znaku
NGS (SGS)	sekwencjonowanie nowej (drugiej) generacji
PCR	reakcja łańcuchowa polimerazy
PE	odczyty sparowane o niewielkiej wzajemnej odległości
SNP	polimorfizm pojedynczego nukleotydu
TGS	sekwencjonowanie trzeciej generacji
WGS	sekwencjonowanie pełnego genomu

Skrót	Opis
WES (WXS)	sekwencjonowanie wszystkich eksonów

Spis rysunków

Spis tabel

Spis pseudokodów