Politechnika Śląska
Wydział Automatyki, Elektroniki i Informatyki

Rozprawa doktorska

Wyznaczanie zbiorów podsłów sekwencji nukleotydowych w danych z sekwencjonowania genomów

Streszczenie (autoreferat)

Marek Kokot

Promotor: prof. dr hab. inż. Sebastian Deorowicz

Katowice, 2020

Spis treści

Spis treści 2

1 Wprowadzenie 3

2 Opracowane algorytmy 5

3 Analiza złożoności obliczeniowej 8

4 Wyniki eksperymentów 9

5 Wnioski 13

Bibliografia 16

Wykaz publikacji autora 18

1 Wprowadzenie

Zliczanie k-merów jest zadaniem polegającym na znalezieniu wszystkich podsłów długości k w zadanej sekwencji (w zadanych sekwencjach). Dodatkowo dla każdego takiego podsłowa należy wyznaczyć liczebność z jaką występuje w sekwencji (sekwencjach). W wyniku zliczania powstaje więc zbiór par, w których pierwszym elementem jest k-mer, a drugim jego liczebność. Zliczanie k-merów jest pierwszym krokiem wielu analiz danych pozyskanych z eksperymentów sekwencjonowania genomów. Jest wykorzystywane m.in. w trakcie asemblacji *de novo*, detekcji powtórzeń, szybkim dopasowaniu wielu sekwencji czy klasyfikacji danych metagenomowych. Chociaż zadanie to jest koncepcyjnie proste jego realizacja nie jest trywialna, zwłaszcza gdy weźmie się pod uwagę charakterystykę danych. Przede wszystkim dane te obarczone są błędami. Pojedynczy błąd substytucji, polegający na przekłamaniu jednego symbolu, może wprowadzić do k błędnych (niewystępujących w genomie badanego organizmu) k-merów. Powstałe w ten sposób błędne k-mery w większości przypadków będą miały liczebność jeden. W przypadku prostego podejścia do zliczania k-merów, polegającego na użyciu tablicy mieszającej, błędy te znacznie zwiększają zapotrzebowanie na pamięć operacyjną. W największym z zestawów danych użytych w eksperymentach, zawierającym dane sekwencjonowania genomu człowieka, składającym się z 736,4 G symboli, całkowita liczba różnych 28-merów wynosi 19,7 G, podczas gdy liczba różnych 28-merów, które wystąpiły co najmniej dwukrotnie to 3,65 G. Algorytm zliczania k-merów musi być więc oszczędny pamięciowo. Wziąwszy pod uwagę rozmiar danych wejściowych pożądane jest także, aby czas działania algorytmu był jak najkrótszy. O tym, że zadanie to nie jest proste może świadczyć również liczba algorytmów zaproponowanych w literaturze. Wśród nich wyróżnić można: Jellyfish [10], DSK [14], KMC 1 [1], Gerbil [3], BFCounter [12], Turtle [15], KCMBT [9], Tallymer [11], MSPKmerCounter [8]. Warto zaznaczyć, że są to rozwiązania dedykowane dla systemów z pamięcią wspólną (UMA/NUMA), dające wyniki dokładne. Poza nimi istnieje również szereg rozwiązań dedykowanych dla systemów z pamięcią rozproszoną lub dających jedynie przybliżone wyniki.

Punktem wyjścia prac przedstawionych w rozprawie jest algorytm KMC 1. Działa on dwuetapowo. W pierwszym etapie, dla każdego k-mera wyznaczany jest numer kubełka, do którego zostanie zapisany. Wykorzystywana jest do tego pewna nieiniekcyjna funkcja. Każdemu kubełkowi odpowiada plik na dysku twardym. Po zapisaniu wszystkich k-merów do odpowiednich kubełków, każdy kubełek może być przetworzony dalej niezależnie od pozostałych w drugim etapie. Do zliczania k-merów w jednym kubełku wykorzystywane jest sortowanie (posortowane identyczne k-mery będą na sąsiadujących pozycjach, dzięki czemu ich zliczenie jest możliwe za pomocą liniowego przejścia). Istotnym aspektem, wpły-

wającym na czas działania, jest więc szybkość samego użytego algorytmu sortowania. Chociaż w algorytmice sortowanie jest problemem szeroko przebadanym i opracowanych zostało wiele rozwiązań, wciąż pojawiają się nowe. Związane jest to z szerokim zakresem zastosowań i nowymi możliwościami oferowanymi przez architektury sprzętowe. Współcześnie, zwykłe stacje robocze, a nawet komputery przenośne, bądź urządzenia mobilne wyposażone są w co najmniej kilka rdzeni. W konsekwencji wydajne algorytmy, w tym algorytmy sortowania, powinny efektywnie wykorzystywać mechanizmy wielowątkowości. Ponadto duży wpływ na szybkość działania może mieć zastosowanie optymalizacji niskopoziomowych, związanych na przykład z realizacją warunkowych rozgałęzień, a także efektywne wykorzystanie pamięci podręcznej procesora. W przypadku sortowania k-merów wszystkie rekordy są tej samej długości, można więc zastosować sortowanie pozycyjne. Dzięki temu, że nie bazuje ono na porównaniach elementów, możliwe jest osiągnięcie złożoności czasowej $\Theta(p)$, gdzie p to liczba sortowanych rekordów.

Funkcja przyporządkowująca k-mery do kubełków zastosowana w algorytmie KMC 1 ma tę cechę, że dwa sąsiadujące k-mery z wejściowej sekwencji z dużym prawdopodobieństwem przypisane zostaną do różnych kubełków. W kontekście tego zastosowania rozpatrywać należy to jako wadę, gdyż przez to każdy k-mer zapisany musi zostać niezależnie od sąsiadujących z nim. Zastosowanie funkcji, w której prawdopodobieństwo przypisania tego samego kubełka sąsiednim k-merom jest wyższe jest pożądane. W takiej sytuacji możliwy jest zapis kilku k-merów w postaci dłuższej sekwencji, nazwanej super-k-merem. Funkcja taka wykorzystana została w algorytmie MSPKmerCounter i bazuje na tzw. minimizerach. Minimizer to najmniejszy leksykograficznie m-mer w k-merze ($m < k$). Prawdopodobieństwo, że dwa sąsiednie k-mery posiadają ten sam minimizer jest na tyle wysokie, że pozwala zapisać k-mery w kubełkach w dużo bardziej zwartej postaci. Minimizery posiadają jednak pewne wady. Pierwszą z nich jest to, że największy z kubełków ma stosunkowo duży rozmiar w stosunku do pozostałych. Drugą z wad jest to, że liczba uzyskanych super-k-merów jest stosunkowo duża. W ogólności minimalizacja obu tych kryteriów nie jest zadaniem łatwym. W algorytmie KMC 2 [2], któremu poświęcono znaczną część rozprawy, zaproponowane zostało uogólnienie minimizerów — sygnatury. W przypadku stosowania sygnatur niektóre z m-merów traktowane są jako zabronione i jeżeli wystąpią w k-merze to są pomijane. Jak się okazuje, dzięki sygnaturom, możliwe jest w pewnym stopniu lepsze zrównoważenie rozmiarów kubełków, jak również zmniejszenie całkowitej liczby super-k-merów. Zapis w kubełkach super-k-merów, poza zmniejszeniem ilości danych, które muszą zostać zapisane na dysku ma jeszcze jedną zaletę. Na etapie sortowania, rekordami zamiast k-merów, mogą być nieco dłuższe sekwencje — (k, x)-mery. Ściślej (k, x)-mer to każdy $(k + x')$-mer dla $x' \in \{0, 1, ..., x\}$. W ten sposób możliwe jest zredukowanie liczby sortowanych rekordów (a więc także zapotrzebowania na pamięć operacyjną) i czasu potrzebnego na wykonanie

sortowania.

Poza wymienionymi wcześniej zastosowaniami wyników algorytmów zliczania k-merów, istnieją także zastosowania bazujące na bezpośrednim zestawianiu ze sobą zbiorów k-merów. Wśród nich wyróżnić można algorytmy DIAMUND [16] i NIKS [13]. Pierwszy z nich ma na celu znalezienie mutacji odpowiedzialnych za nieodziedziczone choroby genetyczne. W swoim działaniu zestawia on zbiory k-merów chorego osobnika i jego zdrowych rodziców. Algorytm NIKS również pozwala znaleźć mutacje pomiędzy dwoma blisko spokrewnionymi organizmami. Operacje na zbiorach k-merów w tych algorytmach wykonywane są w sposób bardzo prosty. Zastosowanie dedykowanego narzędzia mogłoby skrócić czas działania i zmniejszyć zapotrzebowanie na pamięć operacyjną algorytmów bezpośrednio zestawiających zbiory k-merów.

Celem pracy było wykazanie następujących tez:

1. Zastosowanie sygnatur oraz (k, x)-merów znacznie skraca czas działania oraz zapotrzebowanie na pamięć operacyjną i dyskową algorytmu zliczania k-merów.

2. Zastosowanie dedykowanego narzędzia wykonującego operacje na zbiorach k-merów pozwala na skrócenie czasu działania i zapotrzebowania na pamięć operacyjną algorytmów, których działanie bazuje na zestawianiu ze sobą zbiorów k-merów.

3. Wykorzystanie informacji o aktualnych rozmiarach podtablic i wybór odpowiedniej procedury sortowania w trakcie przebiegów sortowania pozycyjnego pozwala opracować szybki hybrydowy algorytm sortowania, który umożliwia skrócenie czasu działania algorytmu zliczania k-merów.

Opisane w pracy badania zostały wcześniej częściowo opisane w artykułach naukowych, których współautorem jest Marek Kokot [2], [7], [5], [6].

2 Opracowane algorytmy

W celu wykazania tez opracowano i zaimplementowano pięć algorytmów: KMC 2, KMC 3, RADULS 1, RADULS 2 oraz KMC tools. Zostaną one pokrótce opisane.

KMC 2

KMC 2 jest algorytmem zliczania k-merów opracowanym na bazie algorytmu KMC 1. Dzięki zastosowaniu sygnatur i (k, x)-merów, opisanych już we wprowadzeniu, możliwe było znaczne zredukowanie czasu wykonywania obliczeń, zapotrzebowania na pamięć operacyjną i dyskową. W kontekście tej ostatniej dla stosunkowo dużych wartości k możliwe było zredukowanie zapotrzebowania na pamięć dyskową o rząd wielkości. Przykładowo dla jednego z zestawów danych, dla $k = 55$ zapotrzebowanie na pamięć dyskową zostało zmniejszone z 279 GB do 29 GB.

RADULS 1

RADULS 1 jest hybrydowym, równoległym algorytmem sortowania opartym na sortowaniu pozycyjnym w wariancie działającym od najbardziej znaczącej cyfry. Podstawową wadą sortowania pozycyjnego jest duża liczba chybień w pamięć podręczną (ang. *cache miss*). W pracy [17] zaproponowane zostało podejście znacznie redukujące ową liczbę. Opiera się ono na wykorzystaniu dodatkowego programowego bufora niewielkiego rozmiaru. Rekordy zamiast zostać zapisane na swoje docelowe pozycje są najpierw zapisywane na odpowiednie pozycje w tym buforze. Dopiero gdy skończy się miejsce związane z daną wartością cyfry wykonywany jest zapis do pamięci głównej. W ten sposób na raz (w jednym transferze danych) zapisywanych jest więcej rekordów. Pomysł ten został wykorzystany w algorytmie RADULS 1. Istotną cechą sortowania pozycyjnego działającego od najbardziej znaczącej cyfry jest fakt, że możliwa jest obserwacja jak podtablice w kolejnych wywołaniach rekurencyjnych się zawężają. Umożliwia to wybór najlepszej w danych okolicznościach procedury sortowania. W algorytmie RADULS 1 wprowadzono klasyfikację podtablic (kubełków) w zależności od ich rozmiaru i numeru aktualnie przetwarzanej cyfry. Wyróżniane są: pierwsza cyfra, duże kubełki, średnie kubełki oraz malutkie kubełki. Gdy rozmiary podtablic stają się bardzo małe, klasyfikowane są jako malutkie kubełki i wybierany jest algorytm ogólnego przeznaczenia, przykładowo sortowanie przez wstawianie. Dzięki temu w wielu przypadkach możliwe jest zakończenie działania algorytmu bez konieczności rekurenycjnego dotarcia do najmniej znaczącej cyfry.

KMC 3

Algorytm RADULS 1 został wykorzystany w algorytmie KMC 3 stanowiącym udoskonalenie algorytmu KMC 2. Pozwoliło to na znaczne zmniejszenie czasu obliczeń drugiego etapu algorytmu, zwłaszcza gdy $k > 32$. Przykładowo, dla największego z przebadanych zestawów danych czas potrzebny na zliczenie 40-merów za pomocą KMC 3 wynosi nieco ponad 2,5 godziny, podczas gdy w przypadku KMC 2 jest to nieco ponad 4 godziny. Poza zastosowaniem algorytmu RADULS 1

wprowadzono szereg innych usprawnień. Jednym z nich jest usprawniony odczyt danych, gdy te są w postaci skompresowanej. Dla tego samego zestawu danych wyznaczenie 40-merów, w danych skompresowanych, za pomocą algorytmu KMC 2 trwało ponad 3,5 godziny, podczas gdy algorytm KMC 3 uzyskał wyniki w czasie poniżej 1 godziny i 40 minut. W algorytmie KMC 3 zmieniony został również sposób przydziału wątków w trakcie drugiego etapu oraz zrównoleglono pozyskiwanie k-merów i ich liczebności z posortowanych (k, x)-merów.

RADULS 2

W trakcie sortowania za pomocą algorytmu RADULS 1 od 10% do 50% czasu wykonania przeznaczone jest na sortowanie kubełków sklasyfikowanych jako malutkie. W ramach prac rozwojowych nad algorytmem RADULS 1 przeprowadzono więc badania związane z sortowaniem tablic o rozmiarach do kilkuset elementów. W ich ramach udało się opracować szybki hybrydowy algorytm dedykowany do sortowania malutkich tablic. Ponadto zastosowano specjalną procedurę wyrównywania rekordów umożliwiającą wykonanie szybszych przesyłów buforów w pamięci. Wprowadzono także dodatkową kategorię — małe kubełki, dedykowaną dla tych kubełków, które nie zostały jeszcze zaklasyfikowane jako malutkie, a jednak ich rozmiar jest na tyle niewielki, że użycie procedury dla średnich kubełków ma pewne wady.

KMC tools

Narzędzie KMC tools umożliwia wykonanie szeregu operacji na parze zbiorów k-merów. Ponadto umożliwia przekształcenia pojedynczego zbioru k-merów oraz zdefiniowanie złożonej operacji, dzięki której możliwe jest wykonanie na raz operacji na wielu zbiorach k-merów. Wejściem narzędzia są zbiory k-merów powstałe wskutek zliczania z wykorzystaniem algorytmu KMC 1, KMC 2 lub KMC 3. Główną motywacją dla stworzenia KMC tools była obserwacja, że pojawiają się algorytmy zestawiające ze sobą zbiory k-merów, a istnieje niewiele narzędzi umożliwiających wykonanie operacji na zbiorach k-merów (jedynym znalezionym rozwiązaniem jest GenomeTester4 [4]).

Optymalizacje sygnatur

Zaproponowane w algorytmie KMC 2 sygnatury, stanowiące uogólnieniem minimizerów, powstały na podstawie pewnych obserwacji co do tego, które z m-merów wpływają niekorzystnie na liczbę super-k-merów i rozmiar największego z plików tymczasowych. W rozprawie rozpatrzono poszukiwanie lepszego zestawu sygnatur. W tym celu wykorzystano szereg heurystyk, poczynając od prostych podejść, na metaheurystyce symulowanego wyżarzania kończąc. W ten sposób uzyskano zestaw sygnatur lepiej minimalizujący wspomniane kryteria. Ponieważ jednak

jako wejście heurystyk wykorzystany został wycinek danych będących wynikiem sekwencjonowania genomu ludzkiego, opracowany zestaw sygnatur obarczony jest ryzykiem dopasowania do danych i może okazać się gorszy od zestawu sygnatur zaproponowanego w KMC 2, jeżeli zostanie użyty dla danych sekwencjonowania genomów innych organizmów.

3 Analiza złożoności obliczeniowej

W rozprawie przeprowadzono szczegółową analizę złożoności czasowej i pamięciowej zaproponowanych algorytmów. Szczególnie interesująca wydaje się analiza prowadząca do uzyskania wartości oczekiwanej liczby wystąpień super-k-mera, w którym liczba k-merów wynosi γ. Jak zostało pokazane, jeżeli $1 \leqslant \gamma < k-m+1$ wynosi ona:

$$E^{\text{liczba}}_{\text{super}_\text{k}}(n,k,m,\gamma) = (n-k+1)\left(\frac{w-\gamma+2}{w+\gamma} + \frac{w-\gamma}{w+\gamma+2} - 2\frac{w-\gamma+1}{w+\gamma+1}\right),$$

natomiast gdy $\gamma = k-m+1$, to $E^{\text{liczba}}_{\text{super}_\text{k}}(n,k,m,\gamma) = (n-k+1)/(2k-2m+1)$. W zależności tej $w = k-m$, przez n oznaczona jest długość sekwencji, m oznacza długość minimizera. Zależność ta jest wprawdzie wyznaczona dla pewnego uproszczonego modelu, jednak dość dobrze przybliża ona sytuację, w której stosowane są, zaproponowane w algorytmie KMC 2, sygnatury. Wyznaczenie tej zależności było istotnym elementem w wyznaczeniu złożoności czasowej algorytmu KMC 2 (a także KMC 3 bo jak pokazano złożoności te są takie same), która wynosi:

$$T^{\text{KMC 3}}_{\text{równoległy}}(r,n,k,m,n_{\text{LUT}},n_\text{p},t) = \mathcal{O}\left(\frac{rnk}{k-m} + \frac{rnk}{t} + n_\text{p}4^{\max(n_{\text{LUT}},m)} + \frac{rnk}{c}\right).$$

Przez r oznaczona jest liczba sekwencji, t oznacza liczbę wątków, c to pokrycie eksperymentu sekwencjonowania, n_{LUT} stanowi wewnętrzny parametr algorytmu związany z oszczędniejszym zapisem wynikowego zbioru k-merów, n_p to liczba plików tymczasowych, w których zapisywane są super-k-mery. Pokazano również, że zapotrzebowanie na pamięć dyskową algorytmu KMC 2 wynosi $\mathcal{O}(r(n-m+k(n-k))/(k-m))$, a wyrażona w bajtach złożoność pamięciowa wynosi:

$$S_{\text{KMC}}(L_{\text{pam}},k,k_\text{m},r,n,n_\text{p},x) = \max\left(L_{\text{pam}}, 16\left\lceil\frac{k}{k_\text{m}}\right\rceil\frac{r(n-k+1)}{n_\text{p}\left(2-\left(\frac{1}{2}\right)^x\right)}\right).$$

W zależności tej L_{pam} to wyrażony w bajtach limit pamięci będący jednym z parametrów algorytmów KMC 2 i KMC 3, a k_{m} to liczba symboli k-mera możliwych do zapisania na jednym słowie maszynowym (równym 8 B), natomiast x to maksymalna liczba symboli dodatkowych (powyżej k) zapisanych w (k, x)-merze.

W rozprawie pokazano również, że złożoność czasowa algorytmu RADULS 2 wynosi:

$$T^{\mathrm{RADULS\,2}}_{\mathrm{równoległy}}(p, R, B, \tau, n_1, \xi, \vartheta, t) = \mathcal{O}\left(\tau\left(n_1RB + \frac{p}{t}(\vartheta + \xi)\right) + \frac{R^{\vartheta+1}}{t} + tn_1R\right),$$

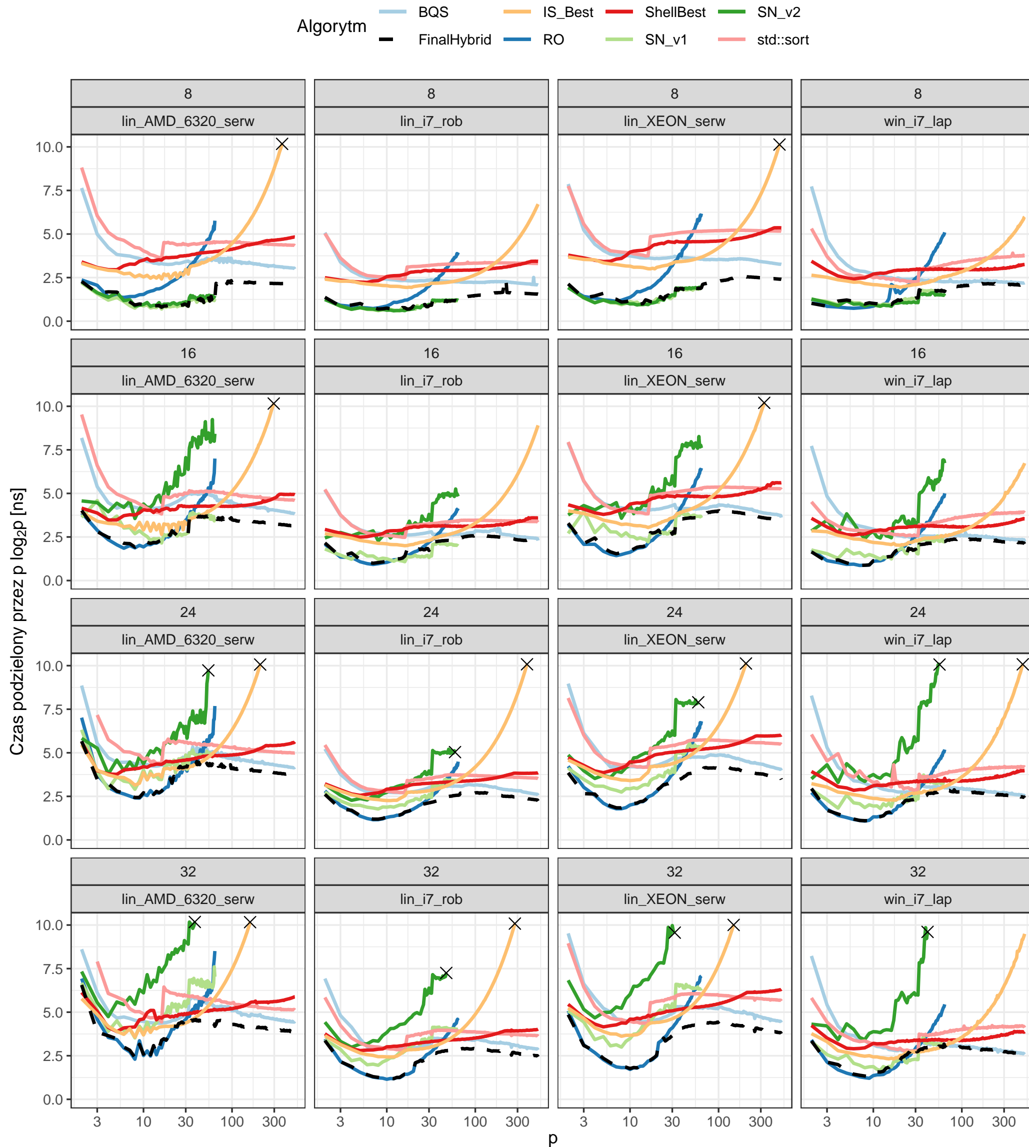
gdzie τ oznacza rozmiar rekordu w bajtach, p rozmiar sortowanej tablicy w rekordach, R to podstawa systemu liczbowego cyfr z jakich zbudowane są rekordy. Wartość ξ oznacza liczbę rekordów w każdym kubełku sklasyfikowanym jako mały bądź malutki, a ϑ to liczba cyfr, dla których odpowiadające im kubełki nie są sklasyfikowane jako małe ani malutkie. W trakcie równoległego przetwarzania najbardziej znaczącej cyfry przez t wątków, tablica dzielona jest na tn_1 przedziałów. Liczba rekordów przypadających na pojedynczy (dla konkretnej wartości cyfry) bufor w pamięci podręcznej oznaczona została przez B.

Niezaprzeczalną wadą przedstawionych złożoności jest trudność ich interpretacji i niewielka przydatność praktyczna. Niemniej jednak sam proces ich wyznaczenia pozwolił jeszcze dokładniej zrozumieć dane wejściowe i zachowanie opracowanych algorytmów.

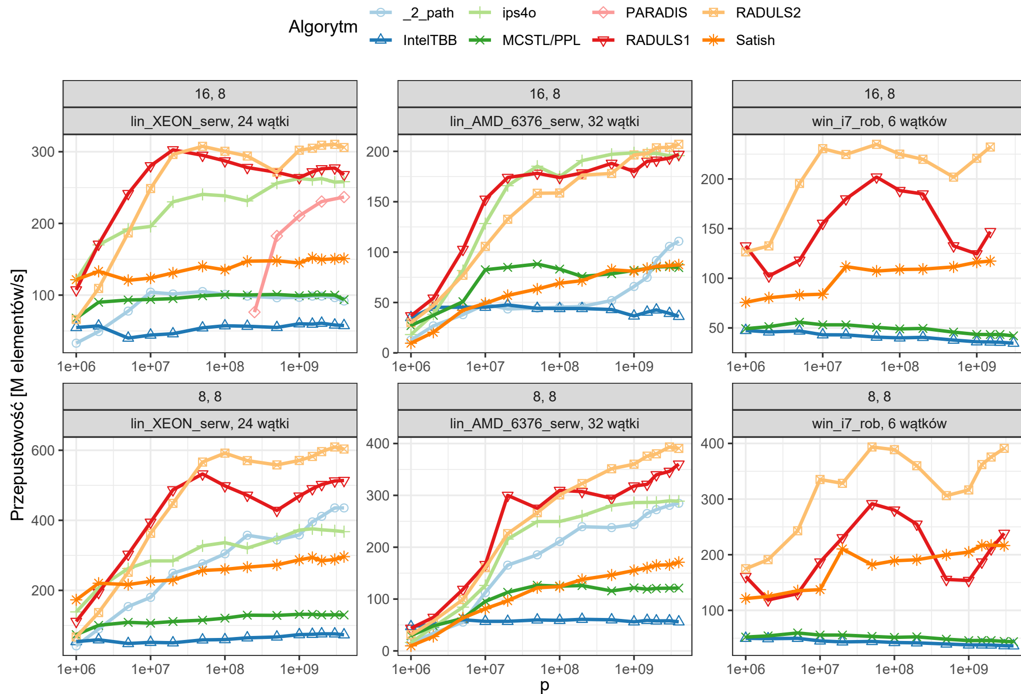
4 Wyniki eksperymentów

Sortowanie

Wyniki badań związanych z opracowaniem algorytmu specjalizowanego do sortowania tablic o rozmiarach do kilkuset elementów zaprezentowane zostały na rysunku 1. Szczegóły na temat użytych platform sprzętowo-programowych oraz algorytmów referencyjnych znaleźć można w rozprawie. Warto mieć na uwadze, że pomiar czasu wykonania dla sortowania tak małych tablic wyrażony jest w nanosekundach. Pomiary tak małych wartości są obarczone dużym ryzykiem błędu stąd są to wyniki uśrednione z wielu wykonań (szczegóły w rozprawie). Zdecydowano się również na nieprzedstawianie bezpośrednich czasów, a ich wartości podzielonych przez $p \log_2 p$, z tego powodu, że dla czasów przedstawionych bezpośrednio wykresy stają się całkowicie nieczytelne. Nie stanowi to dużego problemu, jako że głównym celem tych wykresów jest porównanie algorytmów, a nie



Rysunek 1: Porównanie algorytmów sortowania malutkich tablic dla rekordów o kluczu 8 B. W górnej części każdego wykresu podano rozmiar rekordu, poniżej podano platformę sprzętowo-programową. Wartości na osi rzędnych zostały ograniczone do wartości 10,2, serie danych dla których spowodowało to usunięcie części wyników zostały zakończone znacznikiem $\times$. Zaproponowany algorytm hybrydowy został nazwany FinalHybrid.



Rysunek 2: Porównanie przepustowości algorytmów sortowania dla danych generowanych z rozkładem jednostajnym. W górnej części każdego wykresu podano oddzielone przecinkiem kolejno rozmiar rekordu w bajtach i rozmiar jego klucza w bajtach.

bezpośredni odczyt czasów ich wykonania. Jak widać opracowane rozwiązanie hybrydowe uzyskuje najkrótsze czasy wykonania.

Na rysunku 2 przedstawiono porównanie przepustowości sortowania dla różnych rozmiarów sortowanych tablic. Algorytmy RADULS 1 i RADULS 2 w wielu przypadkach cechują się największą przepustowością, zwłaszcza gdy liczba rekordów jest stosunkowo duża. W większości przypadków algorytm RADULS 2 sprawuje się lepiej niż jego poprzednia wersja. Zdarzają się jednak sytuacje, w których tak nie jest. Spośród istniejących rozwiązań konkurencyjnych na uwagę zasługuje algorytm ips^{4}o, który osiąga w niektórych sytuacjach najwyższe przepustowości, a warto mieć na uwadze, że jest to algorytm oparty na porównaniach elementów.

Zliczanie k-merów

W tabeli 1 przedstawiono porównanie zapotrzebowania na zasoby algorytmów KMC 1 i KMC 2 dla wybranych zestawów danych wejściowych. Tabela ta jasno pokazuje wyraźną przewagę nowej wersji algorytmu. Warto zwrócić uwagę, na to że w przypadku większego z zestawów, dla $k = 55$, algorytm KMC 1 nie zakończył

Tabela 1: Porównanie algorytmów KMC 1 i KMC 2. Rozmiar danych wejściowych skompresowanych (gzip) podany w nawiasie. Liczba wątków została ustawiona na liczbę wirtualnych rdzeni (12). Obliczenia wykonano na komputerze wyposażonym w procesor Intel Core i7-4930 @ 3.4 HGz oraz 64 GiB pamięci operacyjnej. Wyniki HDD zostały uzyskane dla dwóch dysków HDD skonfigurowanych w RAID 0 (355 MB/s) oraz jednego dysku SSD (512 MB/s). Limit przestrzeni dyskowej został ustawiony na 650 GB.

Algorytm	$k = 28$			$k = 55$		
	RAM [GB]	Dysk [GB]	Czas [s]	RAM [GB]	Dysk [GB]	Czas [s]
M. balbisiana (49,1 GB)						
SSD						
KMC 1	13	165	1 229	15	279	2 622
KMC 2	12	41	755	12	29	834
HDD						
KMC 1	13	165	2 194	15	279	3 367
KMC 2	12	41	960	12	29	1 041
***H. sapiens* 2** (105,8 GB)						
SSD						
KMC 1	17	396	2 998	*pamięć dyskowa przekroczona*		
KMC 2	12	101	1 615	13	70	2 038
HDD						
KMC 1	17	369	4 898	*pamięć dyskowa przekroczona*		
KMC 2	12	101	2 259	13	70	2 640

obliczeń z powodu przekroczenia ustalonego limitu pamięci dyskowej, podczas gdy algorytm KMC 2 potrzebował jedynie 70 GB pamięci dyskowej. Zmniejszenie zapotrzebowania na przestrzeń dyskową ma znaczny wpływ na redukcję czasu wykonania. Wynika to z faktu, że operacje wejścia/wyjścia związane z dyskiem twardym są czasochłonne w porównaniu z operacjami wykonywanymi na pamięci operacyjnej.

W rezultacie badań nad sygnaturami udało się uzyskać lepszy ich zestaw dedykowany dla danych pochodzących z eksperymentów sekwencjonowania genomu człowieka. Porównując algorytm KMC 3 z innymi rozwiązaniami rozpatrzono także zmodyfikowaną wersję korzystającą z tego zbioru sygnatur. Ponadto rozpatrzono wersję algorytmu KMC 3, w której zamiast algorytmu RADULS 1 użyty jest algorytm RADULS 2. Wyniki zapotrzebowania na zasoby dla największego z użytych w badaniach zestawów danych przedstawione zostały w tabeli 2. Jak widać, w porównaniu do algorytmu KMC 2, algorytm KMC 3 cechuje się dużo krótszym czasem wykonania, zwłaszcza w sytuacji, w której $k > 32$ (to zasługa użycia algorytmu RADULS 1) oraz gdy dane wejściowe są skompresowane (to zasługa usprawnionego odczytu danych). Jeżeli chodzi o wprowadzone modyfikacje, to widoczne jest, że w przypadku tego zestawu, zastosowanie algorytmu RADULS 2 w większości sytuacji spowodowało zwiększenie czasu obliczeń.

Dla innych zestawów danych, zaprezentowanych w rozprawie, różnice w czasach działania są mniejsze. Nie jest jasne dlaczego zastosowanie algorytmu RADULS 2 nie przyniosło poprawy czasu działania całego algorytmu. Wykorzystanie opracowanego za pomocą metaheurystyki symulowanego wyżarzania zestawu sygnatur pozwoliło na nieznaczne zmniejszenie zapotrzebowania na pamięć dyskową oraz zachowanie narzuconego limitu pamięci (32 GB) w większej liczbie przypadków niż w sytuacji, w której stosowane są sygnatury zaproponowane w KMC 2. Najważniejszym wnioskiem jest jednak to, że w znaczącej większości przypadków algorytmy KMC 2 i KMC 3 sprawują się lepiej od rozwiązań konkurencyjnych. W szczególności istotne wydaje się uzyskanie wyraźnie lepszych czasów działania i zapotrzebowania na pamięć operacyjną niż algorytm Jellyfish 2, który (bazując na liczbie cytowań, wynoszącej na dzień dwudziestego ósmego maja 2020, wg Google Scholar 1 328) jest najpopularniejszym algorytmem wykonującym zliczanie k-merów. Dla porównania, liczby cytować algorytmów KMC 2 i KMC 3 wg Google Scholar to odpowiednio: 181 i 77.

Operacje na zbiorach k-merów

W ramach ewaluacji narzędzia KMC tools użyto go w potoku algorytmu DIAMUND. Wyniki przedstawione zostały w tabeli 3. Dla porównania użyto również istniejącego rozwiązania oferującego podobną funkcjonalność — GenomeTester4. Ponieważ nie wszystkie wymagane operacje były możliwe z jego wykorzystaniem, obliczenia wykonano jedynie dla części kroków. Na podstawie tabeli widać, że w znacznym zakresie udało się zredukować czas wykonania i zapotrzebowanie na pamięć operacyjną.

5 Wnioski

W ramach pracy opracowano nowe, wydajne algorytmy realizujące zdanie zliczania k-merów. Badania eksperymentalne pokazały, że mają one wyraźnie niższe zapotrzebowanie na zasoby niż istniejące rozwiązania. W przypadku opracowanego najpierw algorytmu KMC 2 jest to możliwe dzięki zastosowaniu sygnatur i (k, x)-merów, tak więc pierwsza z tez została udowodniona. Opracowane narzędzie dedykowane do wykonywania operacji na zbiorach k-merów, KMC tools, pozwoliło na znaczne zredukowanie czasu działania i zapotrzebowania na pamięć algorytmu DIAMUND, którego zasadniczą częścią jest bezpośrednie porównanie spokrewnionych osobników poprzez zestawianie zbiorów k-merów. W rozprawie

Tabela 2: Porównanie algorytmów zliczania k-merów dla zestawu *H. sapiens* 3, którego skompresowany (gzip) rozmiar to 614 GB. Czasy podane są dla wejścia nieskompresowanego („Czas”) oraz skompresowanego „Czas-gz”). „—” oznacza, że tryb jest niewspierany. KMC 3-H to zmodyfikowana wersja algorytmu KMC 3, w której wykorzystany został wypracowany heurystycznie zestaw sygnatur. KMC 3-R2 to wersja, w której zamiast algorytmu RADULS 1 użyty został algorytm RADULS 2.

Algorithm	$k = 28$				$k = 55$			
	RAM [GB]	Dysk [GB]	Czas [s]	Czas-gz [s]	RAM [GB]	Dysk [GB]	Czas [s]	Czas-gz [s]
Xeon HDD (12 wątków)								
DSK 2	*czas przekroczony (>12h)*				*czas przekroczony (>12h)*			
Gerbil	29	523	11 994	12 730	62	364	11 968	12 469
GTester4	*czas przekroczony (>12h)*				*niewspierane k*			
Jellyfish 2	84	251	38 338	20 284	104	636	31 783	31 345
KCMBT	*pamięć przekroczona*				*niewspierane k*			
KMC 2 (64GB)	64	551	10 777	9 036	72	381	13 774	11 804
KMC 3 (32GB)	33	596	8 812	5 860	32	389	8 103	5 050
KMC 3-H (32GB)	32	545	8 658	5 377	32	387	8 151	4 971
KMC 3-R2 (32GB)	33	596	8 871	5 822	32	389	8 120	4 977
Opteron HDD (16 wątków)								
DSK 2	*czas przekroczony (>12h)*				*czas przekroczony (>12h)*			
Gerbil	29	523	11 447	18 980	62	364	12 559	15 549
GTester4	*czas przekroczony (>12h)*				*niewspierane k*			
Jellyfish 2	159	0	23 828	22 168	200	473	22 993	24 266
KCMBT	*pamięć przekroczona*				*niewspierane k*			
KMC 2 (64GB)	65	551	11 294	9 066	72	381	16 701	14 654
KMC 3 (32GB)	33	596	8 621	6 630	32	389	8 290	6 345
KMC 3-H (32GB)	33	545	8 252	6 498	32	387	8 254	7 192
KMC 3-R2 (32GB)	33	596	8 513	7 481	33	389	8 350	7 028

pokazano również, że narzędzie to pozwala zmniejszyć zapotrzebowanie na pamięć operacyjną i czas wykonania algorytmu NIKS. W ten sposób udowodniona została druga z tez. W ramach badań nad rozwojem algorytmu KMC 2 udało się opracować bardzo szybkie hybrydowe algorytmy sortowania pozycyjnego, RADULS 1 i RADULS 2. Zasadniczą cechą ich działania jest podejmowanie decyzji dotyczącej wyboru procedury sortowania na podstawie aktualnej sytuacji (numeru aktualnie przetwarzanej cyfry i rozmiaru aktualnej podtablicy). Algorytm RADULS 1 został użyty jako jedno z usprawnień w algorytmie KMC 3. Na podstawie przedstawionych badań można stwierdzić, że trzecia z tez również została udowodniona.

Tabela 3: Porównanie potoku DIAMUND, służącego do wykrywania mutacji dla zestawu BH1019. Czasy dla oryginalnego algorytmu DIAMUND, algorytmu DIAMUND, w którym część kroków zastąpiona została przez KMC 3 i KMC tools, algorytmu DIAMUND, w którym część krókow zastąpiona została przez GenomeTester4. We wszystkich przypadkach użyto 8 wątków. '—' oznacza, że krok nie jest wymagany.

Krok	DIAMUND czas [s]	KMC i KMC tools czas [s]	GenomeTester4 czas [s]
Krok wstępny	583	—	—
Zliczanie k-merów potomka	219	156	281
Konwersja i sortowanie k-merów potomka	1 597	—	—
Zliczanie k-merów rodzica 1	159	150	278
Konwersja i sortowanie k-merów rodzica 1	3 375	—	—
Zliczanie k-merów rodzica 2	199	193	330
Konwersja i sortowanie k-merów rodzica 2	4 179	—	—
Usunięcie k-merów wektora z potomka	102		12
Usunięcie k-merów egzonu referencyjnego z potomka	217	38	6
Usunięcie k-merów rodziców z potomka	1 011		9
Filtrowanie odczytów	2 134	2 310	niewspierane
Reszta potoku	3 928	3 811	
Zastąpione kroki	13 775	2 847	
Cały potok	17 703	6 658	
RAM [GB]	105	7	41

Bibliografia

[1] Sebastian Deorowicz, Agnieszka Debudaj-Grabysz, and Szymon Grabowski. Disk-based k-mer counting on a pc. *BMC bioinformatics*, 14(1):160, 2013. [cytowanie na str. 3]

[2] Sebastian Deorowicz, Marek Kokot, Szymon Grabowski, and Agnieszka Debudaj-Grabysz. Kmc 2: fast and resource-frugal k-mer counting. *Bioinformatics*, 31(10):1569–1576, 2015. [cytowanie na str. 4, 5]

[3] Marius Erbert, Steffen Rechner, and Matthias Müller-Hannemann. Gerbil: a fast and memory-efficient k-mer counter with gpu-support. *Algorithms for Molecular Biology*, 12(1):9, 2017. [cytowanie na str. 3]

[4] Lauris Kaplinski, Maarja Lepamets, and Maido Remm. Genometester4: a toolkit for performing basic set operations-union, intersection and complement on k-mer lists. *Gigascience*, 4(1):58, 2015. [cytowanie na str. 7]

[5] Marek Kokot, Sebastian Deorowicz, and Agnieszka 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. 5]

[6] Marek Kokot, Sebastian Deorowicz, and Maciej Długosz. Even faster sorting of (not only) integers. In *International Conference on Man–Machine Interactions*, pages 481–491. Springer, 2017. [cytowanie na str. 5]

[7] Marek Kokot, Maciej Długosz, and Sebastian Deorowicz. Kmc 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, 2017. [cytowanie na str. 5]

[8] Yang Li et al. Mspkmercounter: a fast and memory efficient approach for k-mer counting. *arXiv preprint arXiv:1505.06550*, 2015. [cytowanie na str. 3]

[9] Abdullah-Al Mamun, Soumitra Pal, and Sanguthevar Rajasekaran. Kcmbt: a k-mer counter based on multiple burst trees. *Bioinformatics*, 32(18):2783–2790, 2016. [cytowanie na str. 3]

[10] Guillaume Marçais and Carl Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6):764–770, 2011. [cytowanie na str. 3]

[11] Allan M Maxam and Walter Gilbert. A new method for sequencing dna. *Proceedings of the National Academy of Sciences*, 74(2):560–564, 1977. [cytowanie na str. 3]

[12] Pall Melsted and Jonathan K Pritchard. Efficient counting of k-mers in dna sequences using a bloom filter. *BMC bioinformatics*, 12(1):333, 2011. [cytowanie na str. 3]

[13] Karl JV Nordström, Maria C Albani, Geo Velikkakam James, Caroline Gutjahr, Benjamin Hartwig, Franziska Turck, Uta Paszkowski, George Coupland, and Korbinian Schneeberger. Mutation identification by direct comparison of whole-genome sequencing data from mutant and wild-type individuals using k-mers. *Nature biotechnology*, 31(4):325, 2013. [cytowanie na str. 5]

[14] Guillaume Rizk, Dominique Lavenier, and Rayan Chikhi. Dsk: k-mer counting with very low memory usage. *Bioinformatics*, 29(5):652–653, 2013. [cytowanie na str. 3]

[15] Rajat Shuvro Roy, Debashish Bhattacharya, and Alexander Schliep. Turtle: Identifying frequent k-mers with cache-efficient algorithms. *Bioinformatics*, 30(14):1950–1957, 2014. [cytowanie na str. 3]

[16] Steven L Salzberg, Mihaela Pertea, Jill A Fahrner, and Nara Sobreira. Diamund: Direct comparison of genomes to detect mutations. *Human mutation*, 35(3):283–288, 2014. [cytowanie na str. 5]

[17] Nadathur Satish, Changkyu Kim, Jatin Chhugani, Anthony D Nguyen, Victor W Lee, Daehyun Kim, and Pradeep Dubey. Fast sort on cpus and gpus: a case for bandwidth oblivious simd sort. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 351–362. ACM, 2010. [cytowanie na str. 6]

Wykaz publikacji autora

Artykuły w czasopismach

1. Deorowicz Sebastian, Kokot Marek, Grabowski Szymon and Debudaj-Grabysz Agnieszka. KMC 2: Fast and resource-frugal k-mer counting, 2015. BIOINFORMATICS, vol. 31, no. 10, pp. 1569-1576,

2. Kokot Marek, Długosz Maciej and Deorowicz Sebastian: KMC 3: counting and manipulating k-mer statistics, 2017. BIOINFORMATICS, vol. 33, no. 17, pp. 2759–2761,

3. Deorowicz Sebastian, Gudyś Adam, Długosz Maciej, Kokot Marek, Danek Agnieszka. Kmer-db: instant evolutionary distance estimation, 2019, BIOINFORMATICS, vol. 35, no. 1, pp. 133–136.

Artykuły w materiałach konferencyjnych

1. Kokot Marek, Deorowicz Sebastian and Debudaj-Grabysz Agnieszka. Sorting Data on Ultra-Large Scale with RADULS: New Incarnation of Radix Sort, 2017. 13th International Conference Beyond Databases Architectures and Structures (BDAS 2017), 30/05/2017-02/06/2017, Ustroń, Polska, pp. 235-245 Beyond Databases, Architectures and Structures. Towards Efficient Solutions for Data Analysis and Knowledge Representation, ISBN: 978-3-3195-8273-3, pp. 235–245,

2. Kokot Marek, Deorowicz Sebastian, and Długosz Maciej. Even faster sorting of (not only) integers. International Conference on Man–Machine Interactions (ICMMI 2017), pp. 481–491,

3. Długosz Maciej, Deorowicz Sebastian, and Kokot Marek. Improvements in DNA Reads Correction. In International Conference on Man–Machine Interactions (ICMMI 2017), pp. 115–124.

4. Kokot Marek, Długosz Maciej and Deorowicz Sebastian. Operations on k-mers' sets, 2015. 8th Symposium of the Polish Bioinformatics Society (PTBI2015), 17/09/2015-19/09/2015, Lublin, Polska. (Plakat)