STUDIA INFORMATICA 2008
Volume 29 Number 4A (80)

Tadeusz KRASIŃSKI
Uniwersytet Łódzki, Wydział Matematyki i Informatyki
Sebastian SAKOWSKI
Politechnika Śląska, Instytut Informatyki

PRZEGLĄD MODELI I PRAKTYCZNYCH IMPLEMENTACJI *DNA* OBLICZEŃ

Streszczenie. W artykule omówiono różne możliwości przetwarzania informacji za pomocą łańcuchów *DNA*. Przedstawiono wyniki niektórych prac doświadczalnych oraz teoretyczne modele obliczeń za pomocą *DNA*.

Słowa kluczowe: algorytm, teoretyczny model, *DNA* obliczenia

A REVIEW OF MODELS AND PRACTICAL IMPLEMENTATIONS OF *DNA* COMPUTATION

Summary. In this paper we discuss various possibilities of using *DNA* to information processing. We describe some practical implementations and theoretical models of computation built on *DNA*.

Keywords: algorithm, theoretical model, *DNA* computing

1. Wstęp

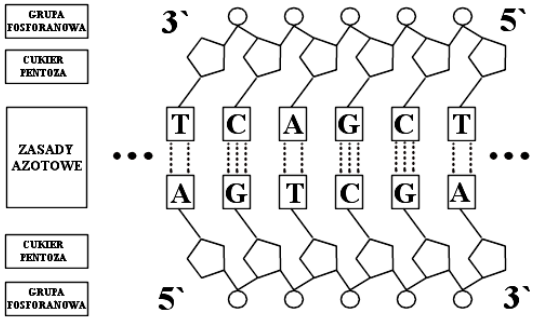
Początek badań nad informatyką jako dziedziną nauki o przetwarzaniu i ruchu informacji rozpoczął się od konstrukcji teoretycznych (w głowach logików matematycznych). Znacznie później powstały praktyczne maszyny cyfrowe wykorzystywane do automatycznego wykonywania obliczeń. Ich działanie było oparte na przepływie elektronów. Powstał przemysł komputerowy, który dąży do ciągłego ulepszania komputerów elektronicznych. Obecnie prowadzone badania nad nowymi technologiami, mogącymi zastąpić komputery oparte na elektronice, koncentrują się na trzech kierunkach: komputery kwantowe, komputery optyczne oraz komputery oparte na łańcuchach *DNA*. W pracy omówimy trzeci kierunek („*DNA* Com-

puting"). Dynamiczny rozwój tej dziedziny, umiejscowionej na pograniczu informatyki, matematyki i biologii, rozpoczął się od eksperymentu Adlemana, który jako pierwszy użył łańcuchów *DNA* do znanego problemu kombinatorycznego (istnienie drogi Hamiltona w grafie). Doświadczenie to pokazało, jak dużą moc obliczeniową i masową równoległość przetwarzania mogą mieć systemy informatyki wzorowane na organizmach żywych. Od tego czasu dziedzina ta znacznie rozwinęła się. Pojawiły się różne praktyczne implementacje obliczeń za pomocą *DNA* oraz zostały opracowane różne teoretyczne modele obliczeń opartych na *DNA*. Celem pracy jest przeglądowe omówienie wybranych modeli obliczeń za pomocą *DNA* oraz ich praktycznych (laboratoryjnych) zastosowań.

Praca została podzielona na cztery części. Pierwsza omawia skrótowo budowę *DNA* oraz przybliża podstawowe operacje na *DNA*, używane w praktycznych implementacjach. Teoretyczne modele obliczeń opartych na *DNA* przedstawia druga część artykułu. W części trzeciej omówione zostały ważniejsze praktyczne implementacje obliczeń opartych na *DNA*. Ostatnia część pracy porównuje te praktyczne implementacje.

2. Budowa i podstawowe operacje na DNA

Rozkwit badań nad *DNA* jako polimeru kodującego informacje rozpoczął się w 1953 roku, kiedy to James Watson i Francis Crick, na podstawie zdjęć krystalografii rentgenowskiej, opracowali model łańcucha zwanego helisą *DNA*. Od tego momentu organizmy żywe zaczęto postrzegać jako bardzo złożone procesy przetwarzające informacje [24]. Systemy informatyczne, dzięki którym organizmy żywe mogą istnieć, zaczęto nazywać biologicznymi systemami informatyki [27, 28]. *DNA* jest nośnikiem informacji genetycznej określającej funkcje i budowę organizmów żywych. Podstawowym elementem budowy organizmów żywych są białka zbudowane z 20 różnych aminokwasów. Białka pełnią wiele funkcji w organizmach żywych, np.: funkcję budulcową, funkcję katalityczną, funkcję sterującą itd. Łańcuch *DNA* zawiera wszystkie informacje potrzebne do syntezy różnych białek. Samo *DNA* jest polimerem zbudowanym z nukleotydów. Pojedynczy nukleotyd zbudowany jest z zasady azotowej, cukru oraz z grupy fosforanowej. Nukleotydy różnią się między sobą zasadą azotową. Zatem, właściwym nośnikiem informacji genetycznej w *DNA* są cztery zasady azotowe: adenina (*A*), guanina (*G*), tymina (*T*) i cytozyna (*C*). Zasady azotowe w nukleotydach łączą się ze sobą „komplementarnie", tzn.: adenina łączy się zawsze z tyminą (dwa wiązania wodorowe), a guanina z cytozyną (trzy wiązania wodorowe). Watson i Crick zaproponowali model *DNA*, w którym dwa łańcuchy nukleotydów oplatają się helikalnie dzięki „komplementarności" zasad azotowych.



Rys. 1. Budowa *DNA*
Fig. 1. The structure of *DNA*

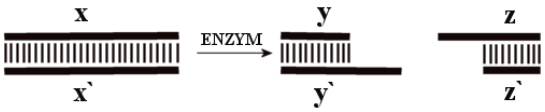
DNA nie jest bezpośrednią matrycą do syntezy białek, funkcję tę pełni *RNA*. Wyróżniamy informacyjny *RNA* (*mRNA*), transportujący *RNA* (*tRNA*) oraz rybosomowy *RNA* (*rRNA*). Informacja zapisana w *DNA* jest poddawana procesowi transkrypcji (przepisywania) z *DNA* na informacyjny *RNA*. Pozostałe dwa rodzaje *RNA* (*tRNA* oraz *rRNA*) stanowią część mechanizmu syntezy białek. Kodowanie informacji w *RNA* odbywa się, podobnie jak w *DNA*, za pomocą zasad azotowych z tą różnicą, że zamiast tyminy (*T*) w *RNA* występuje uracyl (*U*).

Podstawową jednostką określającą informację genetyczną jest gen, czyli określony odcinek *DNA*. Geny nie kodują informacji w sposób ciągły, wyróżniamy odcinki kodujące informacje („eksony") oraz odcinki niekodujące („introny"). Zespół genów nazywamy genomem, który wraz z procesami przetwarzającymi informację w nim zawartą stanowi bardzo złożony system informatyczny. W biologicznych systemach informatyki *DNA* pełni rolę zewnętrznej pamięci, zawierającej informacje o sposobie porządkowania swobodnych atomów i molekuł w użyteczne cząsteczki [28]. Biorąc pod uwagę, że biologiczne systemy informatyki rozwijały się miliardy lat, są one prawdopodobnie najbardziej rozwiniętymi systemami informatyki znanymi człowiekowi. W biologicznych systemach informatyki podstawowa informacja reprezentowana jest w postaci bitów molekularnych, które są reprezentowane czterema zasadami azotowymi (adeniną, guaniną, tyminą, cytozyną w *DNA* oraz dodatkowo uracylem w *RNA*).

W latach siedemdziesiątych ubiegłego stulecia nastąpił rozwój metod sztucznej rekombinacji *DNA*, czyli wymiany fragmentów *DNA* na inne (w szczególności syntezę dowolnego łańcucha *DNA*). Umożliwiło to sterowanie informacją zapisaną w *DNA* i powstanie nowej dziedziny zwanej inżynierią genetyczną. Możliwe to było dzięki opracowaniu technik działania na *DNA*.

Wyróżniamy następujące podstawowe operacje (działania) na *DNA*:

- **Cięcie**. Związki chemiczne zwane enzymami restrykcyjnymi umożliwiają precyzyjne cięcie łańcuchów *DNA* (rys. 2) w określonym miejscu (np. enzym *BseXI* rozpoznaje następującą sekwencję nukleotydów *GCAGC*, a następnie tnie łańcuch *DNA* w odległości 8 nukleotydów od znalezionej sekwencji).

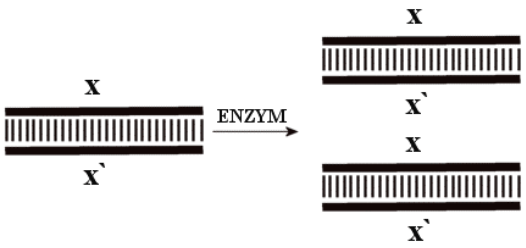


Rys. 2. Cięcie *DNA*
Fig. 2. Cutting *DNA*

- **Łączenie**. Enzymy ligazy umożliwiają łączenie za sobą łańcuchów *DNA* (rys. 3), np. połączenie dwóch łańcuchów *DNA* z komplementarnymi odcinkami w jeden długi.

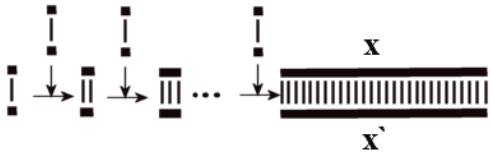
Rys. 3. Łączenie łańcuchów *DNA*
Fig. 3. Linking *DNA*

- **Kopiowanie**. Enzymy polimerazy kopiują informacje z jednego łańcucha *DNA* na drugi (rys. 4), np. polimeraza *DNA* tworzy kopię komplementarną do nici matrycy



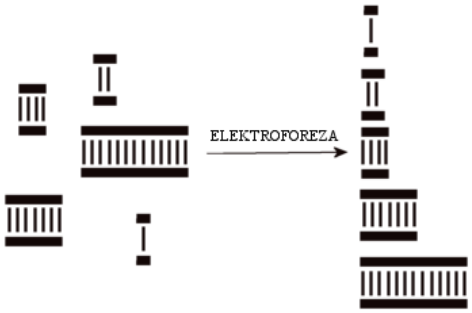
Rys. 4. Kopiowanie *DNA*
Fig. 4. Copying *DNA*

- **Synteza**. Współczesne laboratoria biologii molekularnej umożliwiają syntezę wcześniej zaprojektowanego łańcucha *DNA* o dowolnej sekwencji zasad (rys. 5).



Rys. 5. Synteza *DNA*
Fig. 5. Synthesizing *DNA*

- **Sortowanie.** Do rozdzielania łańcuchów *DNA* o różnej długości wykorzystuje się elektroforezę na żelu. Analizowany roztwór *DNA* nanosi się na płytkę uformowaną z żelu elektroforetycznego. Wzdłuż krawędzi płytki biegną elektrody. Pod wpływem stałego napięcia elektrycznego łańcuchy *DNA* ulegają stopniowemu rozdzieleniu i sortowaniu ze względu na długość łańcuchów *DNA*, czyli ilość nukleotydów (rys. 6).



Rys. 6. Sortowanie *DNA*
Fig. 6. Sorting *DNA*

3. Teoretyczne modele komputerów zbudowanych z DNA

3.1. Model oparty na "filtrowaniu"

Pierwszym problemem kombinatorycznym rozwiązanym eksperymentalnie za pomocą *DNA* była problem drogi Hamiltona w grafie. Jest to znany problem NP-zupełny, polegający na znalezieniu drogi przechodzącej przez wszystkie wierzchołki grafu, przy założeniu że każdy wierzchołek występuje tylko raz na tej drodze. Eksperyment ten opracowany został przez L. Adlemana [1]. Wykazał on wyjątkowe możliwości wykorzystania *DNA* do wykonywania obliczeń. W swojej pracy nie podał jednak formalnego modelu komputera opartego na *DNA*. Następnie w 1995 roku R. Lipton [11] opracował formalny model przetwarzania informacji oparty na idei Adlemana. Użył on tych samych operacji co Adleman i rozwiązał inny znany problem NP-zupełny – 3SAT.

Idea metody Adlemana jest następująca. Wszystkie dane wejściowe grafu (wierzchołki i krawędzie), zakodowane w postaci odpowiednich łańcuchów *DNA*, mieszamy ze sobą. W roztworze łańcuchy te łączą się ze sobą na wszystkie możliwe sposoby, tworząc różne łańcuchy *DNA*. Jeżeli istnieje rozwiązanie rozpatrywanego problemu (przy tych danych wejściowych), to wśród tych łańcuchów istnieją takie, które reprezentują to rozwiązanie. Problem polega na „wykryciu" tych rozwiązań w roztworze. Metodą użytą do tego jest „filtrowanie" (filtering model), którego teoretyczny model możemy opisać następująco. Dany jest zbiór łańcuchów *DNA* (dokładnie wielozbiór, gdyż każdy łańcuch *DNA* może występować w wielu kopiach). Na tym wielozbiorze wykonujemy następujące operacje (wzorowane na operacjach wykonywanych na *DNA*):

$input(T)$ – funkcja wejścia; dany jest wejściowy zbiór (lub wielozbiór) T.

$separate(T,S)$ – funkcja rozdzielania; dany jest zbiór T (łańcuchów *DNA*) oraz S inny zbiór łańcuchów *DNA*. Utwórz dwa nowe zbiory $+(T,S)$ oraz $-(T,S)$, gdzie $+(T,S)$ są

wszystkimi łańcuchami *DNA* w T, zawierającymi S, natomiast $-(T,S)$ są wszystkimi łańcuchami w T niezawierającymi S.

$merge(T_1,T_2,...,T_n)$ – funkcja łączenia; dane są zbiory $T_1,T_2,...,T_n$, utwórz ich sumę $\bigcup\{T_1,...,T_n\} = T_1 \cup T_2 \cup ... \cup T_n$.

$detect(T)$ – funkcja detekcji; dany jest zbiór T. Wyjściem jest prawda, gdy T jest niepuste, w przeciwnym przypadku wyjściem jest fałsz.

Przykład 1

Dla danego T np.: $T = \{A,B,C\}$ dany algorytm daje w wyniku prawdę, gdy dane wejściowe zawierają A:

$$\begin{aligned} &input(T)\\ &T \leftarrow -(T,B)\\ &T \leftarrow -(T,C)\\ &detect(T) \end{aligned} \tag{1}$$

3.2. Model oparty na "splataniu"

Kolejnym teoretycznym modelem *DNA* obliczeń jest „system splatania” (*splicing system*) opisany po raz pierwszy w 1987 roku przez T. Heada [8]. Model oparty na splataniu został rozwinięty i dokładnie opisany przez G. Păuna, G. Rozenberga, A. Salomaa w monografii [14]. Reguły tego systemu dokładnie odpowiadają operacjom cięcia oraz łączenia *DNA*. W modelu tym dwuniciowe *DNA* są reprezentowane jako pojedyncze słowa z alfabetu $\{A,T,G,C\}$ (ze względu na komplementarność Watsona-Cricka jedna nić jednoznacznie wyznacza drugą). Autorzy opisali wiele różnych systemów splatania, od prostych akceptujących języki regularne, do skomplikowanych opartych na wielozbiorach, akceptujących języki rekursywnie przeliczalne. Jednym z modeli opartych na „splataniu” jest H-system (H od twórcy systemu – Thomasa Heada [8]). Jest to układ

$$S = (V,\Sigma,A,R) \tag{2}$$

składający się z następujących elementów:

V – zbiór skończony zwany alfabetem S,

$\Sigma \subset V$ – zbiór skończony zwany alfabetem symboli terminalnych (końcowych),

$A \subset V^*$ – zbiór słów z alfabetu V; słowa te nazywamy aksjomatami systemu S,

$R \subset V^* \times V^* \times V^* \times V^*$ – zbiór reguł splatania.

Każdą regułę $r = (u_1,u_2,u_3,u_4) \in R$ zapisujemy inaczej z użyciem dodatkowych symboli $\#$ (oznaczający miejsce cięcia łańcuchów *DNA*) oraz $\$$ (rozdzielający łańcuchy *DNA*).

$$r = u_1 \# u_2 \$ u_3 \# u_4 \tag{3}$$

Do określenia języka generowanego przez S definiujemy relację $\Rightarrow_r$ dla każdej reguły $r = u_1 \# u_2 \$ u_3 \# u_4 \in R$. Jeżeli $x, y, z \in V^*$, to

$$(x, y) \Rightarrow_r z, \tag{4}$$

gdy $x = x_1 u_1 u_2 x_2$, $y = y_1 u_3 u_4 y_2$ oraz $z = x_1 u_1 u_4 y_2$ dla pewnych $x_1, x_2, y_1, y_2 \in V^*$. Wejściowe słowa x oraz y są cięte w miejscach określonych przez regułę splatania r i następnie łączona jest pierwsza część pierwszego słowa z drugą częścią drugiego słowa. Jeżeli x, y przebiegają wszystkie aksjomaty z A, to otrzymamy pierwszy poziom zbioru słów $\sigma(A)$, tzn. definiujemy:

$$\sigma(A) = \{z \in V^* : (x, y) \Rightarrow_r z \text{ dla pewnych } x, y \in A \text{ oraz } r \in R\}. \tag{5}$$

Iterując ten proces, otrzymujemy drugie, trzecie itd. poziomy zbiorów słów

$$\begin{aligned} &\sigma^0(A) = A, \\ &\sigma^{i+1}(A) = \sigma^i(A) \cup \sigma(\sigma^i(A)), \\ &i = 0,\ 1,\ 2,\ \ldots \end{aligned} \tag{6}$$

Ostatecznie

$$\begin{aligned} &\sigma^*(A) = \bigcup_{i \geq 0} \sigma^i(A), \\ &L(S) = \sigma^*(A) \cap \Sigma^*. \end{aligned} \tag{7}$$

Język $L(S)$ jest nazywany językiem generowanym przez system splatania S. Słowo x z symboli terminalnych Σ (tzn.: $x \in \Sigma^*$) należy do $L(S)$, jeżeli x może być otrzymane z aksjomatów przez użycie reguł splatania w skończonej ilości.

Moc obliczeniowa takich systemów splatania, tzn. klasa języków generowanych przez systemy splatania, nie jest duża. Zachodzi następujące twierdzenie ([15], Theorem 7.6).

Twierdzenie. Klasa języków generowanych przez systemy splatania o skończonym zbiorze aksjomatów i reguł splatania jest równa klasie języków regularnych.

Przykład 2

Prostym przykładem systemu splatania jest np. dodawanie dwóch liczb. Definiujemy $S = (V, \Sigma, A, R)$, gdzie alfabetem systemu jest $V = \{a, b, c\}$, natomiast zbiór symboli terminalnych to $\Sigma = \{a\}$. Zakładamy, że aksjomatami systemu są dwa słowa $\{a^n b, c a^m\}$ dla pewnych $n, m \in \mathrm{N}$. Reguła działająca na aksjomatach to $r = a \# b \$ c \# a$. W wyniku działania reguły r na słowach $(a^n b, c a^m)$ jest a^{n+m}, gdyż w powyższych oznaczeniach mamy:

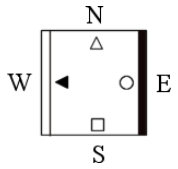
$$(a^n b, c a^m) = (\underbrace{a}_{x_1}^{n-1} \underbrace{a}_{u_1} \underbrace{b}_{u_2} \underbrace{\Lambda}_{x_2}, \underbrace{\Lambda}_{y_1} \underbrace{c}_{u_3} \underbrace{a}_{u_4} \underbrace{a}_{y_2}^{m-1}) \Rightarrow_r \underbrace{a}_{x_1}^{n-1} \underbrace{a}_{u_1} \underbrace{a}_{u_4} \underbrace{a}_{y_2}^{m-1} = a^{n+m}, \tag{8}$$

gdzie przez Λ oznaczamy słowo puste (tzn. słowo niezawierające żadnego symbolu).

3.3. Model oparty na "samoskładaniu"

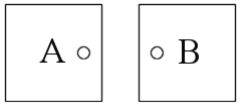
Kolejnym teoretycznym modelem, mającym swoje źródło w reakcjach chemicznych związanych z *DNA*, jest model oparty na „samoskładaniu” (*tile assembly model*). Jego idea polega na łączeniu się „płytek” w zależności od etykiet i „mocy wiązania” przypisanych bokom płytek. Wzorowane jest to na samoskładaniu się związków chemicznych, takich jak kryształy, łańcuchy *DNA* itd. Twórcami tego modelu byli P. Rothemund oraz E. Winfree [17]. Autorzy wykazują, że model ten jest równoważny z uznanym teoretycznym modelem obliczeń, czyli maszyną Turinga. W tym modelu każda „płytka” (jest ich skończona ilość, ale każda dostępna w potencjalnie nieskończonej liczbie kopii) ma przypisaną do każdego boku etykietę oraz moc wiązania. Na przykład dla płytki z rysunku 7 bok:

- zachodni (W) ma etykietę ◄ i moc łączenia 2 (podwójna linia),
- północny (N) ma etykietę Δ i moc 1 (jedna linia),
- południowy (S) ma etykietę □ i moc 1 (jedna linia),
- wschodni (E) ma etykietę ○ i moc łączenia 0 (pogrubiona linia).



Rys. 7. Przykładowy wygląd płytki
Fig. 7. An example of a tile

Proces samoskładania rozpoczyna się od specyficznej wyróżnionej płytki „zarodkowej”. Dołączenie kolejnej płytki jest możliwe, gdy łączone boki mają te same etykiety i łączna moc wiązania (tzn. suma wspólnych mocy łączonych boków) przekracza z góry określoną liczbę τ (odpowiednik temperatury w reakcjach chemicznych). Na przykład, gdy $\tau = 1$, to płytki z rysunku 8 połączą się, gdyż łączone boki mają tę samą etykietę (kółko) oraz moc wiązania $1 \geq \tau$. Gdy $\tau = 2$, to płytki te nie połączą się.



Rys. 8. Płytki łączące się przy $\tau = 1$
Fig. 8. Binding tiles for $\tau = 1$

Wynikiem działania modelu są kolejno powstające możliwe tablice płytek lub, przy innej definicji, tablice końcowe, do których nie można już dokładać żadnej płytki.

Formalna definicja systemu *R* opartego na samoskładaniu jest następująca

$$R = (T, S, g, \tau) \tag{9}$$

gdzie:

T – jest skończonym zbiorem „płytek”,

S – jest zbiorem „płytek zarodkowych” (na ogół jest to jedna),

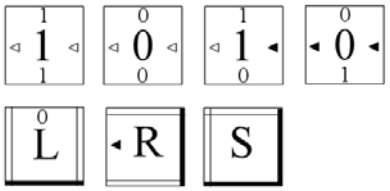
g – moc wiązania przypisana bokom płytek,

$\tau > 0$ – jest odpowiednikiem temperatury.

System oparty na samoskładaniu jest definiowany jako relacja $\rightarrow_T$ między konfiguracjami (tablicami) A, B powstającymi z płytek: $A \rightarrow_T B$, jeżeli istnieje płytka $t \in T$ i miejsce (x, y) w tablicy A takie, że B powstaje z A przez dołączenie (uprawnione) płytki t.

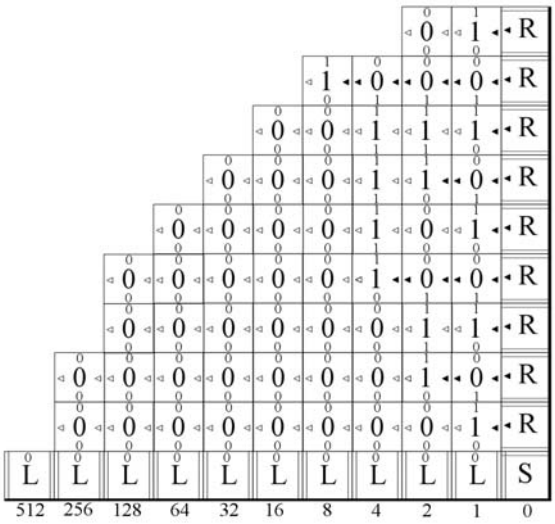
Przykład 3

Rozważmy następujący system samoskładania złożony z siedmiu płytek (rys. 9).



Rys. 9. Płytki użyte w procesie samoskładania
Fig. 9. Tiles used in the self-assembly system

Przyjmujemy, że S jest płytką zarodkową i $\tau = 2$. Dokładna analiza wszystkich możliwości prowadzi do wniosku, że powstanie tylko następująca tablica (zawierająca płytki oznaczone 0 i 1 – rys. 10), która może dalej rozbudowywać się, dając w kolejnych wierszach (liczonych od dołu) binarne przedstawienie kolejnych liczb naturalnych. Kolumny są wartościami kolejnych pozycji danej liczby naturalnej zapisanej w systemie dwójkowym. Dla ułatwienia pod każdą kolumną zapisujemy potęgę liczy 2, odpowiadającą pozycjom w systemie dwójkowym.



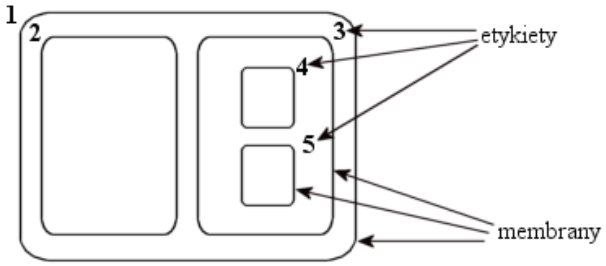
Rys. 10. Proces wzrastania struktury w modelu samoskładania
Fig. 10. Process of growing up structure in the self-assembly system

3.4. Model oparty na "membranach"

Ogólna idea modelu opartego na „membranach” (*membrane model*) wzorowana jest na błonach komórkowych występujących w organizmach żywych [15, 16]. Podstawowym elementem modelu są membrany, które oddzielają dwa regiony, skończony wewnętrzny od nieskończonego zewnętrznego. Oba regiony (zewnętrzny oraz wewnętrzny) mogą się ze sobą komunikować. W modelu tym operujemy na zbiorach obiektów, a właściwie na wielozbiorach oddzielonych membranami. Wielozbiór może być reprezentowany w różny sposób, lecz najlepszą postacią jest struktura ciągów. Przykładowo, jeżeli obiekty a,b,c są reprezentowane w ilościach 5, 2, 6 każda, to reprezentujemy ten wielozbiór słowem $a^5b^2c^6$. Wszystkie inne permutacje tego ciągu są tym samym wielozbiorem.

Reakcje biochemiczne w organizmach żywych są niedeterministyczne, ale prawie zawsze paralelne (tzn. działające na tych samych kopiach tego samego związku chemicznego). Zatem, operacje na wielozbiorach dokładnie odpowiadają temu schematowi. Możemy zatem mówić, tak jak to jest w organizmach żywych, o przetwarzaniu informacji.

W modelu tym struktura membran jest hierarchiczna, membrany zawierają wewnątrz inne membrany.



Rys. 11. Budowa membran
Fig. 11. The structure of membrane

Każda membrana ma określoną etykietę (np. numer) i komponenty wewnątrz: wielozbiory, reguły działania. Za pomocą tych reguł przekształceniu ulegają nie tylko wielozbiory w tych membranach, ale również same membrany. Wyróżniamy następujące rodzaje reguł:

- operujące na wielozbiorach, np. reguła

$$ab \to ac \tag{10}$$

(wszystkie pary ab w danej membranie zostają zamienione na pary ac – wtedy a nazywamy katalizatorem),

- komunikacyjne między membranami, np. reguła

$$d \to (e, out) \tag{11}$$

(każdy symbol d zostaje zamieniony na e i przeniesiony na zewnątrz membrany, tzn. do membrany o jeden poziom wyżej),

- operujące na membranach, np. reguła

$$c \to a^2 \delta \tag{12}$$

(zastosowanie tej reguły powoduje zmianę c na a^2 i zniknięcie tej membrany; otrzymany wielozbiór przechodzi do membrany o jeden poziom wyżej).

Wybór reguł jest niedeterministyczny, ale paralelny, tzn. w jednym ruchu zastosowany do wszystkich możliwych elementów danego wielozbioru. Wynikiem działania systemu jest liczba symboli w wielozbiorze, który znajdzie się na zewnątrz wszystkich membran (w środowisku) po zatrzymaniu się wszystkich działań.

Formalna definicja modelu opartego na membranach, zwanego również P systemem, jest następująca:

$$\Pi = (O, C, \mu, \omega_1, \omega_2, ..., \omega_m, R_1, R_2, ..., R_m, i_0) \tag{13}$$

gdzie:

O – jest alfabetem skończonym i niepustym,

$C \subset O$ – zbiór katalizatorów,

μ – jest strukturą membran, zawierającą m membran etykietowanych przez 1, 2, 3, …, m; mówi się, że system jest stopnia m,

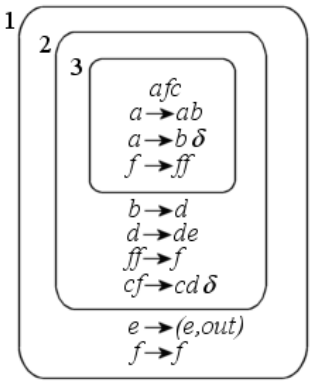
$\omega_1, \omega_2, ..., \omega_m$ – są ciągami z O reprezentującymi wielozbiory w regionach 1, 2, … , m w strukturze membran,

$R_1, R_2, ..., R_m$ – są zbiorami reguł skojarzonymi z regionami 1, 2, … , m,

i_0 – wyjściowy region systemu (i_0 jest jedną z etykiet, zwykle jest to region zewnętrzny).

Przykład 4

Rozważmy następujący system membran P zadany graficznie na rys. 12.



Rys. 12. System membran
Fig. 12. A membrane system

W regionie 3 (ograniczonym membraną nr 3) afc jest wielozbiorem (tzn. symbole a, f, c są dane w pojedynczych kopiach), a $a \to ab$, $a \to b\delta$, $f \to ff$ regułami działania i podobnie w regionach 2 oraz 1. Jako region wyjściowy przyjmujemy zewnętrze membran. Tak określony system membran oblicza kwadraty liczb naturalnych, tzn.

$$L(P) = \{n^2 : n \in N\}. \tag{14}$$

Liczbę n^2 otrzymujemy w następujący sposób. Do wielozbioru afc w regionie 3 stosujemy paralelnie $n-1$ razy reguły $a \to ab$ i $f \to ff$ (pierwsze zastosowanie tych reguł da nam wielozbiór $abffc$ a drugie $ab^2 f^4 c$ itd.). W pozostałych regionach nie ma działań ze względu na brak wielozbiorów w tych regionach. Ostatecznie otrzymujemy wielozbiór $ab^{n-1} f^{2^{n-1}} c$ w regionie 3. Teraz stosujemy regułę $a \to b\delta$ i regułę $f \to ff$ (paralelnie do każdego symbolu f). Reguła $a \to b\delta$ usuwa membranę nr 3 i w regionie 2 pojawia się wielozbiór $b^n f^{2^n} c$. W tym regionie stosujemy regułę $b \to d$ i $ff \to f$ (paralelnie dla każdego z symboli). Otrzymujemy wielozbiór $d^n f^{2^{n-1}} c$. Teraz stosujemy reguły $d \to de$ oraz $ff \to f$ $(n-1)$ razy. Otrzymujemy wielozbiór $d^n e^{n(n-1)} f^{2^0} c = d^n e^{n(n-1)} fc$. Dopiero teraz stosujemy regułę $d \to de$ i $cf \to cd\delta$ (nie mogliśmy wcześniej niedeterministycznie zastosować reguły $cf \to cd\delta$, gdyż spowodowałoby to przejście do regionu 1 symbolu f, a tam reguła $f \to f$ nie pozwalałaby zakończyć działania systemu). Następnie usuwamy membranę 2 i następuje przejście do rejonu 1 wielozbioru $d^{n+1} e^{n^2} c$. Tutaj reguła $e \to (e, out)$ powoduje przeniesienie symboli e na zewnątrz systemu i na tym kończy się działanie P. Na zewnątrz mamy wielozbiór e^{n^2}, czyli wynikiem działania jest liczba n^2. Z opisu tego wynika również, że żadna inna liczba niebędąca postaci n^2 nie może być rezultatem działania P.

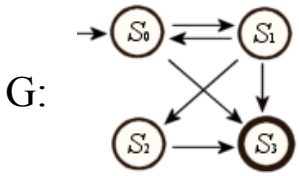
4. Wybrane praktyczne implementacje obliczeń za pomocą DNA

4.1. Obliczenia molekularne Adlemana

W 1994 roku Leonard Adleman [1] jako pierwszy wykorzystał *DNA* do wykonania obliczeń. W swoim artykule przedstawił możliwość rozwiązania problemu drogi Hamiltona w grafie za pomocą odpowiednio zaprojektowanych cząsteczek *DNA*. Zagadnienie drogi Hamiltona było wielokrotnie badane przez informatyków i matematyków. Nie udało się jednak opracować algorytmu rozwiązującego problem drogi Hamiltona w odpowiednio krótkim czasie (wielomianowym).

Ogólna idea metody Adlemana jest następująca. Wejściowe dane (wierzchołki, krawędzie) zakodowane są w pewnych, relatywnie krótkich, łańcuchach *DNA* (w dużej ilości kopii – w teoretycznym modelu odpowiada to wielozbiorowi). W roztworze łańcuchy te łączą się na wiele sposobów. Pewne z nich reprezentują rozwiązanie problemu (o ile takie istnieje). Zadanie praktyczne polega na wykryciu (wyodrębnieniu), za pomocą operacji na *DNA*, takich

właściwych łańcuchów. Obliczenie laboratoryjne zostało zaprojektowane dla następującego grafu G o czterech wierzchołkach (rys. 13).



Rys. 13. Graf problemu obliczonego za pomocą *DNA*
Fig. 13. Graph of a problem computed by *DNA*

W G istnieje tylko jedna droga Hamiltona, która kolejno będzie przechodzić przez stany: S_0, S_1, S_2, S_3. Każdemu wierzchołkowi (rys. 14) oraz krawędziom (rys. 15) przypisano odpowiednie kodowania za pomocą *DNA*.

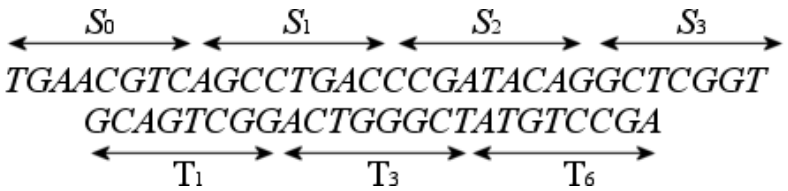
Wierzchołek	Kodowanie	Wierzchołek	Kodowanie
S_0	***TGAACGTC***	S_2	***CCGATACA***
S_1	***AGCCTGAC***	S_3	***GGCTCGTT***

Rys. 14. Kodowanie wierzchołków grafu G
Fig. 14. Coding of vertices in the graph G

Przejście	Sekwencja *DNA*	Przejście	Sekwencja *DNA*
T1: $S_0 \longrightarrow S_1$	***GCAGTCGG***	T4: $S_1 \longrightarrow S_3$	***ACTGCCGA***
T2: $S_0 \longrightarrow S_3$	***GCAGCCGA***	T5: $S_1 \longrightarrow S_0$	***ACTGACTT***
T3: $S_1 \longrightarrow S_2$	***ACTGGGCT***	T6: $S_2 \longrightarrow S_3$	***ATGTCCGA***

Rys. 15. Kodowanie krawędzi w grafie G
Fig. 15. Coding of edges in the graph G

Adleman wykorzystał komplementarność w cząsteczce *DNA*, czyli własność do łączenia się zasad: adeniny z tyminą oraz guaniny z cytozyną. W probówce umieścił nici *DNA* kodujące wszystkie wierzchołki oraz krawędzie grafu G i po kilku minutach wygenerował szukaną drogę w grafie (wśród wielu innych niewłaściwych) reprezentowaną przez łańcuch *DNA*.



Rys. 16. Łańcuch *DNA* będący rozwiązaniem problemu
Fig. 16. *DNA* strand which is a solution of the problem

Ze względu na pracochłonność metod biotechnologicznych odczyt informacji wynikowej zajął mu jednak bardzo dużo czasu. Do wykrycia właściwego wyniku Adleman wykorzystał następujące metody biotechnologiczne: PCR (reakcja łańcuchowa polimeryzacji), elektroforezę (rozdział na żelu) oraz amplifikację (rozdział oparty na powinowactwie).

4.2. Automat Shapiro

W 2001 roku Y. Benenson, R. Adar, T. Paz-Elizur, Z. Livneh, E. Shapiro [3] przedstawili programowalny automat zbudowany z *DNA* (automat Shapiro) oraz enzymu restrykcyjnego *FokI*. Automat ten jest zbliżony do teoretycznego biomolekularnego urządzenia opracowanego w 1973 roku przez Bennetta [6]. Automat Shapiro jest dwustanowym niedeterministycznym automatem skończonym. Umożliwia on rozwiązanie prostych problemów algorytmicznych. Wszystkie elementy tego automatu oraz słowo wejściowe zbudowane są z *DNA*. Mieszamy ze sobą ciągi *DNA* w wielu kopiach (wielozbiór) reprezentujące program oraz ciąg wejściowy automatu. W wyniku naprzemiennej operacji cięcia i łączenia słowa wejściowego zbudowanego z *DNA* otrzymujemy w roztworze sekwencję terminalną *DNA*, która oznacza akceptowanie słowa wejściowego.

Słowo wejściowe, np.: $A \in \{a,b\}^*$ zbudowane jest z *DNA*. Łańcuch ten zaczyna się lepkim końcem, który koduje pierwszy symbol słowa A oraz stan początkowy. Następnie kodowane są kolejne symbole słowa wejściowego. Słowo wejściowe kończy się sekwencją terminalną.

Kolejnym elementem automatu zbudowanego z *DNA* są łańcuchy przejść (ruchy). Dla automatu dwustanowego mamy możliwych osiem przejść, które oznaczono $T1,...,T8$ (rys. 17). Każdy łańcuch przypisany ruchowi rozpoczyna się od sekwencji *GGATG*, która jest rozpoznawana przez enzym *FokI*. Po jej odnalezieniu enzym *FokI* tnie łańcuch *DNA* po 9 molekułach na nici *DNA* od $5'$ do $3'$ oraz po 13 molekułach na nici *DNA* od $3'$ do $5'$ od rozpoznanego miejsca.

Przejście	Sekwencja DNA	Przejście	Sekwencja DNA
T1: $S_0 \xrightarrow{a} S_0$	***GGATGC*** ***CCTACGCCGA***	T5: $S_1 \xrightarrow{a} S_0$	***GGATG*** ***CCTACACCG***
T2: $S_0 \xrightarrow{a} S_1$	***GGATGCC*** ***GCTACGGCCGA***	T6: $S_1 \xrightarrow{a} S_1$	***GGATGC*** ***CCTACGACCG***
T3: $S_0 \xrightarrow{b} S_0$	***GGATGC*** ***CCTACGGTCC***	T7: $S_1 \xrightarrow{b} S_0$	***GGATG*** ***CCTACCGTC***
T4: $S_0 \xrightarrow{b} S_1$	***GGATGCC*** ***CCTACGGGTCC***	T8: $S_1 \xrightarrow{b} S_1$	***GGATGC*** ***CCTACGCGTC***

Rys. 17. Kodowanie przejść
Fig. 17. Coding of transitions

Proces przetwarzania informacji w automacie Shapiro rozpoczyna się od zmieszania ze sobą: łańcuchów *DNA* kodujących słowo wejściowe A, łańcuchów *DNA* reprezentujących wybrane ruchy z $T1,...,T8$, enzymu Ligaza umożliwiającego łączenie się ciągów *DNA* oraz enzymu *FokI* umożliwiającego cięcie łańcuchów *DNA*. Przetwarzanie informacji odbywa się automatycznie przez stopniowe analizowanie słowa wejściowego w cyklu przejść. Każdy cykl składa się z następujących operacji:

- słowo wejściowe jest łączone (za pomocą enzymu ligaza) z odpowiednią molekułą reprezentującą przejście,
- eliminacja kolejnego symbolu ze słowa wejściowego w wyniku działania enzymu *FokI*.

W przypadku gdy wprowadzone słowo jest akceptowane przez automat, stopniowa analiza słowa wejściowego powinna doprowadzić do odpowiedniej sekwencji terminalnej oznaczającej zakończenie działania automatu. Podobnie jak w eksperymencie Adlemana wykrycie właściwego wyniku działania automatu wymagało pracochłonnych metod biotechnologicznych: PCR oraz elektroforezy.

Shapiro pokazał możliwość wykorzystania automatów zbudowanych z *DNA* do innych działań niebędących obliczeniami. Po dalszych badaniach zespół z Instytutu Weizmanna przedstawił w 2004 roku w pracy [5] możliwość wykrywania pewnych sekwencji np. nowotworowych przez automaty zbudowane z *DNA*, a następnie uwalniania leku. Badania Shapiro pokazały, że możliwe jest zmuszenie pewnych zbiorów łańcuchów *DNA* do działania i przetwarzania informacji w sposób automatyczny.

4.3. Automat grający w kółko i krzyżyk

Kolejną ciekawą laboratoryjną implementacją jest molekularny automat (MAYA), umożliwiający za pomocą *DNA* realizowanie algorytmu znanej gry w kółko i krzyżyk (problem tic-tac-toe). W 2003 roku [23] M. Stojanovic oraz D. Stefanovic przedstawili kolejną metodę kontrolowanego przetwarzania informacji za pomocą *DNA*, a dokładniej deoxyrybozymu, który jest cząsteczką *DNA* o katalitycznych właściwościach, takich jak enzymy. Deoxyrybozym umożliwia cięcie (rozerwanie) *DNA*. Z teoretycznego punktu widzenia w eksperymencie tym został użyty automat Mealy'ego, który na dany ruch oponenta podaje ruch komputera. Przy założeniu że pierwszy ruch wykonuje komputer, automat ten zawsze wygrywa lub przy szczególnym układzie ruchów oponenta remisuje.

Automat MAYA składa się z dziewięciu probówek odpowiadających polom na tablicy 3×3.

1	2	3
4	5	6
7	8	9

Rys. 18. Pola tablicy 3×3
Fig. 18. Fields of a table 3×3

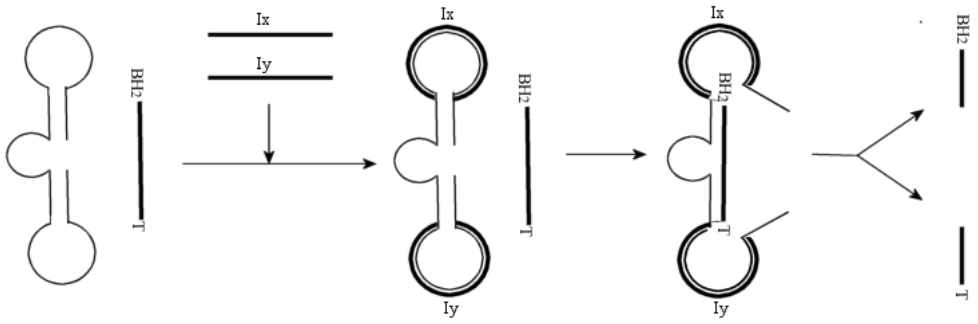
W każdej probówce umieszczane są odpowiednie łańcuchy *DNA*. Ruchy oponenta polegają na dodaniu do wszystkich probówek tego samego łańcucha *DNA*, odpowiadającego numerowi pola, w którym oponent chce postawić symbol kółka O. Odpowiedniość ta jest następująca (rys. 19).

Ruch oponenta	Łańcuch *DNA*
I_1	***TCTGCGTCTATAAAT***
I_2	***ATCGTATGTTGTTCA***
I_3	***GTATAGTCTGTTTGT***
I_4	***GTAAGTGCTCAAATGTC***
I_6	***GTCTAATTCTCACGGTC***
I_7	***TAGTCTGTGTGTTGT***
I_8	***TCTATATGAGCGTAA***
I_9	***TGTCCATCTAAATCC***

Rys. 19. Kodowanie ruchów oponenta
Fig. 19. Coding of opponent moves

Ze względu na specyfikę algorytmu tic-tac-toe brak jest ruchu I_5. Jest to związane z koniecznością wykonania przez automat pierwszego ruchu w strategii typu wygraj (czyli wyboru pola 5). W poszczególnych probówkach zachodzą reakcje na *DNA*, w konsekwencji których jedna z nich zmienia barwę dzięki fluorescencji (jest to odpowiedź *DNA* komputera na ruch oponenta). Następuje kolejny ruch oponenta, zmiana barwy w pewnej probówce itd. Komputer *DNA* zawsze wygrywa lub w jednym tylko przypadku remisuje.

Istota konstrukcji polega na budowie odpowiednich łańcuchów *DNA* umieszczonych w poszczególnych probówkach. Zilustrujemy to na przykładzie bramki logicznej AND, czyli zdania logicznego $x \wedge y$ (gra w kółko i krzyżyk jest pewnym automatem Mealy'ego, który można zapisać za pomocą zdań logicznych). W probówce znajduje się łańcuch *DNA* w formie pętli (spinki) oraz substrat (BH_2 – *GAGAAGG-rA-TATCACT* – T), gdzie na jednym końcu, znajduje się związek fluorescencyjny T (TAMRA), a na drugim związek absorbujący fluorescencję. Rozerwanie substratu powoduje zanik absorpcji i świecenie fluorescencyjne danej probówki. Jeżeli do probówki dodamy ruch oponenta I_x i następnie I_y (są to wejścia dla bramki logicznej $x \wedge y$; obecność I_x oznacza prawdziwość zdania x, jego nieobecność oznacza fałszywość zdania x i podobnie dla I_y), to połączą się one z pętlami, które zostaną otworzone, pozwalającymi substratowi połączyć się z tym łańcuchem (rys. 20). Deoxyrybozym w tej sytuacji tnie substrat, powodując zanik absorpcji i tym samym fluorescencję (czyli wyjściem jest prawda).



Rys. 20. Przetwarzanie informacji dla bramki AND
Fig. 20. Processing of information for gate AND

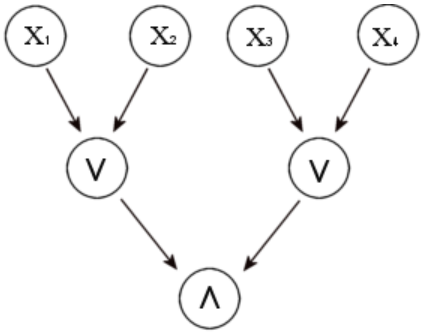
Wykorzystując inne właściwości deoxyrybozymu, można wytworzyć inne bramki logiczne, które wystarczą do zaimplementowania gry w kółko i krzyżyk.

Autorzy zwracają uwagę, że metoda może zostać w przyszłości wykorzystana do szybkiego diagnozowania we krwi wirusów, nowotworów itd. Stojanovic oraz Stefanovic napisali, że w przyszłości powstaną narzędzia, oparte na *DNA* komputerach, umożliwiające wykrywanie i jednoczesne zabijanie komórek nowotworowych.

4.4. Bramki logiczne Ogihary i Raya

W 1997 roku M. Ogihara oraz A. Ray [12] zaimplementowali w laboratorium bramki logiczne (sieci logiczne) zbudowane z *DNA*. Zatem autorzy udowodnili, że *DNA* może zostać użyte do implementacji standardowego modelu przetwarzania używanego w informatyce.

Jednym z teoretycznych modeli obliczeń, równoważnym maszynom Turinga, są sieci logiczne. Z definicji jest to graf skierowany, acykliczny mający 2 rodzaje wierzchołków: wierzchołki wejściowe $\{x_1,...,x_n\}$ reprezentujące zmienne i wierzchołki zwane bramkami logicznymi $\{\vee,\wedge,\neg\}$ reprezentującymi spójniki logiczne, np. sieć logiczna z rys. 21 przedstawia zdanie logiczne $(x_1 \vee x_2) \wedge (x_3 \vee x_4)$.

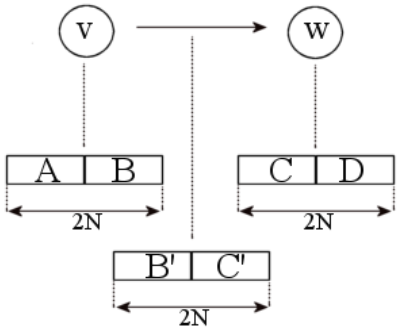


Rys. 21. Przykład sieci logicznej
Fig. 21. An example of Boolean circuit

Jeżeli każdej zmiennej przypiszemy wartość logiczną 0 lub 1, to w wyniku działania sieci logicznej otrzymamy również wartość 0 lub 1 (w ogólniejszym modelu może być więcej wierzchołków wyjściowych niż jedna oraz do bramek logicznych $\vee$ oraz $\wedge$ może dochodzić więcej krawędzi niż dwie itd.). Formalnie sieci logiczne mogą reprezentować każdą funkcję logiczną $F : \{0,1\}^n \rightarrow \{0,1\}$.

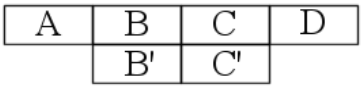
Główna idea implementacji sieci logicznej za pomocą DNA jest następująca. Dla danej sieci logicznej każdy wierzchołek (wejściowy i bramki logiczne) ma przypisany unikalny jednoniciowy łańcuch *DNA* o tej samej długości (parzystej) 2N. Podobnie każdej krawędzi przypisany jest też jednoniciowy łańcuch *DNA* długości 2N, którego pierwsza część jest komplementarna do drugiej części łańcucha odpowiadającego początkowi krawędzi, a druga

część jest komplementarna do pierwszej części łańcucha odpowiadającego końcowi wierzchołka krawędzi.

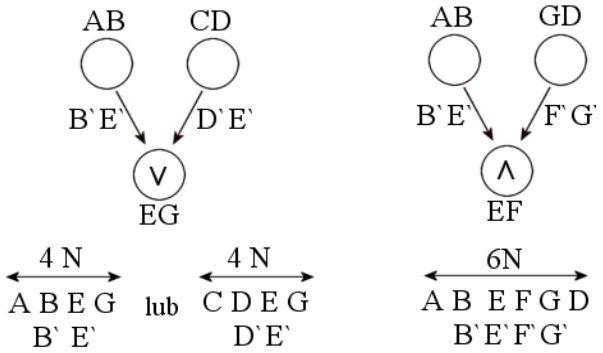


Rys. 22. *DNA* odpowiadające elementom grafu
Fig. 22. *DNA* corresponding to graph elements

Zatem jeżeli w roztworze (probówce) występują łańcuchy odpowiadające wierzchołkom v i w oraz krawędzi łączącej je, to w roztworze powstaną łańcuchy *DNA* długości 4N.



Rys. 23. Łańcuch *DNA* długości 4N
Fig. 23. *DNA* strand of length 4N



Rys. 24. Przykład działania bramek
Fig. 24. An example of gates

Proces przetwarzania informacji przebiega etapami numerowanymi kolejnymi poziomami grafu danej sieci. W pierwszym etapie w probówce umieszczamy nici *DNA* odpowiadające zmiennym, którym przypisujemy wartość 1 (nici, którym przypisujemy wartość 0 nie będą użyte). Na przykład dla sieci z rys. 21, jeżeli chcemy obliczyć wartość zdania dla wartości zmiennych $x_1 \to 0, x_2 \to 1, x_3 \to 1, x_4 \to 1,$ to w probówce umieszczamy nici zmiennych (wierzchołków) x_2, x_3, x_4. Następny krok to dodanie do roztworu nici odpowiadających krawędziom pierwszego poziomu. Dla bramek 1 poziomu, które przyjmują wartość 1, powstaną łańcuchy długości 4N (w przypadku bramki OR) i 6N (w przypadku bramki AND). Na przykład, jeśli długości słów $|A| = |B| = |C| = |D| = |E| = |F| = |G| = N$ i wierzchołkom

oraz krawędziom przypiszemy łańcuchy tak jak na rys. 24, to powstaną następujące łańcuchy długości 4N i 6N.

Za pomocą elektroforezy na żelu poliakrylowym usuwamy z roztworu łańcuchy o mniejszej długości (<4N). Następnie pozostałe tniemy enzymami restrykcyjnymi tak, by pozostały nici wierzchołków pierwszego poziomu (tych, które mają wartość 1). Następnie powtarzamy cykl dla drugiego poziomu itd. Jeżeli w wyniku tych etapów w roztworze znajdziemy łańcuch *DNA* odpowiadający wierzchołkowi wyjściowemu, to oznacza, że wartością logiczną wyjściową jest 1.

4.5. Problem skoczka Faulhammera

W 2000 roku D. Faulhammer, A. Cukras, R. Lipton, L. Landweber [7] przedstawili implementację za pomocą *DNA* i *RNA* znanego NP-problemu zupełnego SAT na przykładzie „problemu skoczka szachowego". Problem ten polega na znalezieniu wszystkich rozmieszczeń skoczków na szachownicy $n \times n$ tak, by wzajemnie nie atakowały się.

Rys. 25. Przykładowe dobre i złe rozmieszczenie skoczków
Fig. 25. Examples of correct and wrong position of knights

Problem ten można zamienić na problem spełnialności pewnego zdania logicznego (z problemu SAT). Na przykład na szachownicy 3×3 (rys. 26), jeżeli poszczególne pola oznaczymy kolejnymi literami alfabetu, to jest to zdanie logiczne

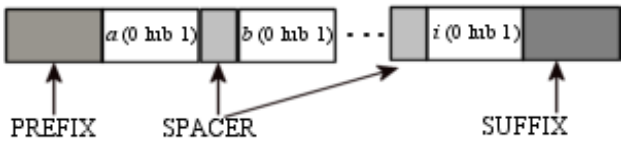
$$((\neg h \wedge \neg f) \vee \neg a) \wedge ((\neg g \wedge \neg i) \vee \neg b) \wedge ((\neg d \wedge \neg h) \vee \neg c) \wedge ((\neg c \wedge \neg i) \vee \neg d) \wedge ((\neg a \wedge \neg g) \vee \neg f) \tag{15}$$

Oznacza to, że dobre rozmieszczenie skoczków na tej szachownicy jest równoważne przypisaniu wartości logicznych zmiennym $a,b,...,i$ (1, gdy na tym polu jest skoczek i 0 w przeciwnym przypadku) tak, by zdanie (15) było spełnione (czyli miało wartość logiczną 1).

a	b	c
d	e	f
g	h	i

Rys. 26. Szachownica 3×3
Fig. 26. The chess-board 3×3

Implementacja problemu polega na wytworzeniu, najpierw w postaci łańcuchów *DNA*, biblioteki wszystkich możliwych rozmieszczeń skoczków na szachownicy, a następnie dokonaniu transkrypcji ich w postaci *RNA*. Każdy taki łańcuch ma postać (rys. 27).



Rys. 27. Łańcuch reprezentujący rozmieszczenie skoczków
Fig. 27. The strand representing a knights position

Każda ze zmiennych może przyjmować wartość 0 (brak skoczka na tym polu) lub 1 (skoczek jest obecny na tym polu). Na rys. 28 podany jest sposób kodowania wartości 0 i 1 za pomocą łańcuchów *DNA*.

POLE	0 – SKOCZEK OBECNY	1 – SKOCZEK NIEOBECNY
a	***CTCTTACTCAATTCT***	***TCCTCACATTACTTA***
b	***CATATCAACATCTTA***	***ACTTCCTTTATATCC***
c	***ATCCTCCACTTCACA***	***TTATAACAAACATCC***
d	***TTAAAATCTTCCCTC***	***ACATAACCCTCTTCA***
e	***CTATTTATCCACACC***	***ACCTTACTTTCCATA***
f	***GCTTCAAACAATTCC***	***GTACATTCTCCCTAC***
g	***AACTCTCAAATTCAA***	***CATAATCTTATATTC***
h	***CTAACCTTTACTTCA***	***ATAATCACATACTTC***
i	***CATTCCTTATCCCAC***	***TCCACCAACTACCTA***

Rys. 28. Kodowanie wartości zmiennych
Fig. 28. Coding of values of variables

Po wytworzeniu wszystkich możliwych rozmieszczeń skoczków na szachownicy następuje drugi etap, polegający na niszczeniu łańcuchów niespełniających zdania logicznego (15). Polega to na zaznaczeniu „złych" łańcuchów *RNA* (odpowiadających złym ustawieniom skoczków) przez dodanie do roztworu odpowiednich komplementarnych łańcuchów *DNA* oraz na ich niszczeniu za pomocą *RNasy H* (rybonukleaza H niszczy wiązania fosfodiestrowe łańcuchów *RNA* połączonych z *DNA*).

Proces ten powtarzamy kolejno dla każdych zdań cząstkowych w (15):

$$\begin{aligned}&((\neg h \wedge \neg f) \vee \neg a),\ ((\neg g \wedge \neg i) \vee \neg b),\ ((\neg d \wedge \neg h) \vee \neg c),\\ &((\neg c \wedge \neg i) \vee \neg d),\ ((\neg a \wedge \neg g) \vee \neg f)\end{aligned} \tag{16}$$

niszcząc te łańcuchy, które nie spełniają tych zdań cząstkowych. Dla danego zdania cząstkowego np. pierwszego w (16)

$$((\neg h \wedge \neg f) \vee \neg a) \tag{17}$$

polega to na:

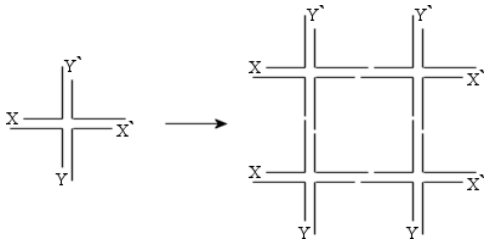
- podzieleniu roztworu na dwie równe części,
- w pierwszej części niszczymy łańcuchy o wartości $a = 1$ (pozostają łańcuchy o wartości $a = 0,$ a więc spełniające (17)),

- w drugiej części roztworu niszczymy łańcuchy o wartości $a = 0$ oraz łańcuchy o wartości $h = 1$ lub $f = 1$ (pozostają łańcuchy o wartościach $a = 1$, $h = 0$ i $f = 0$, a więc spełniające (17)),
- mieszamy ze sobą oba roztwory (w nim dokładnie znajdują się łańcuchy spełniające (17)),
- powtarzamy proces dla każdych zdań cząstkowych,
- po zakończeniu procesu w roztworze otrzymujemy łańcuchy, które spełniają całe zdanie (15), a więc wszystkie rozwiązania problemu.

4.6. Nanotechnologia Seemana

Kolejne ciekawe badania doświadczalne przeprowadził N. Seemann, który otrzymał nagrodę w dziedzinie nanotechnologii za opracowanie różnych struktur geometrycznych za pomocą *DNA*. Z badań autora [20] wynika, że *DNA* ma wiele właściwości umożliwiających potencjalne zastosowanie w nanotechnologii, szczególnie w systemach samowytwarzania struktur, które można traktować jak proces przetwarzania informacji – model samoskładania. Ze względu na swoje rozmiary oraz możliwość komplementarnego łączenia nici *DNA* jest bardzo dobrym medium do tworzenia struktur nanotechnologicznych.

W przetwarzaniu informacji struktury te wykorzystali dopiero P. Rothemund, E. Winfree [17] w swoim modelu opartym na samoskładaniu. Seeman podał jednak jako pierwszy techniczne, inne niż liniowa, możliwości budowy struktur opartych na *DNA*. Podał wiele różnych możliwości budowy struktur opartych na *DNA*. Do przetwarzania informacji nadają się znakomicie struktury zbudowane z *DNA* o czterech ramionach kończących się lepkimi końcami.



Rys. 29. Struktura czteroramienna zbudowana z *DNA* i przykład łączenia
Fig. 29. The four-arms structure built on DNA and an example of binding

Elementy te można wykorzystać do samowytwarzania skomplikowanych struktur, a nawet implementować problemy informatyczne oparte na modelu samowytwarzania. W 1998 roku E. Winfree, F. Liu, L. Wenzler, oraz N. Seeman [26] przedstawili, oparty na matematycznej „teorii części", programowalną strukturę samowytwarzania. Programowanie zostało osiągnięte przez komplementarność łańcuchów *DNA* oraz odpowiednie kodowanie lepkich końców struktur zbudowanych z *DNA*. W doświadczeniu użyte zostały dwa zbiory „części"

A oraz B tworzące pasma tzw. krat. Struktury powstałe w wyniku odpowiedniego programowania *DNA* zostały przedstawione w mikroskopie elektronowym.

5. Porównanie implementacji praktycznych

Różnorodność modeli teoretycznych oraz implementacji praktycznych skłania do uporządkowania wiedzy w dziedzinie „*DNA* Computingu" oraz podania głównych kierunków badań. Informacje dotyczące teoretycznych modeli budowy *DNA* komputerów zamieszczone są w pracach [2, 8, 13, 14, 15, 16, 17], natomiast różne praktyczne implementacje są między innymi w następujących artykułach [1, 3, 4, 5, 7, 12, 20, 21, 22, 23, 26].

Z przedstawionego przeglądu ważniejszych implementacji praktycznych wynika, że główne badania nad *DNA* komputerami koncentrują się na opracowaniu różnych implementacji znanych algorytmów [1, 7, 23] lub modeli przetwarzania informacji używanych w informatyce [3, 4, 11, 12] z zastosowaniem metod inżynierii genetycznej.

Pionierskie badania L. Adlemana pokazały nową drogę badań nad automatycznym wykonywaniem obliczeń za pomocą *DNA*. Od tego czasu powstało wiele prac naukowych pokazujących możliwości kodowania i przetwarzania informacji za pomocą *DNA*. Prace w tym zakresie koncentrowały się w zasadzie na dwóch rodzajach badań. Pierwszym były teoretyczne rozważania nad możliwościami budowy modeli (systemów) zbudowanych z *DNA*. W badaniach tych wykorzystuje się ogólnie znane narzędzia formalne, głównie teorię automatów i języków formalnych [9, 10]. W książce „*DNA Computing*" [14] autorzy opisują wyczerpująco modele obliczeń za pomocą *DNA* i systematyzują wiedzę związaną z tymi modelami. Uwagę koncentrują jednak głównie na modelu opartym „na splataniu".

Drugi kierunek rozwoju *DNA* komputerów koncentruje się na badaniach doświadczalnych, co głównie związane jest z kontrolowanym (programowalnym) przetwarzaniem zbiorów zawierających odpowiednio zaprojektowane cząsteczki *DNA*. W praktycznych implementacjach dostrzegalna jest różnorodność koncepcji budowy danych wejściowych oraz wyjściowych, a także procesu przetwarzania informacji.

Sposoby kodowania słowa wejściowego w implementacjach praktycznych różnią się znacząco od siebie. W pracy L. Adlemana [1] wykorzystane są pojedyncze nici *DNA*, które zawierają informacje o rozwiązywanym problemie. Podobną koncepcję przedstawiają M. Ogihara i A. Ray [12], którzy konstruują jednoniciowe łańcuchy *DNA*, za pomocą których realizują w laboratorium standardowy element przetwarzania informacji – bramki logiczne. W swoich badaniach E. Shapiro i jego zespół [3, 4] używają natomiast jako wejścia dwuniciowego *DNA*, którym koduje dowolne słowo z alfabetu $\Sigma = \{a, b\}$ podlegające przetwarzaniu. Kolejny autor (D. Faulhammer i inni [7]) konstruuje natomiast dwuniciowe biblioteki

DNA, które ulegają niszczeniu w procesie przetwarzania informacji (realizacji algorytmu). Zupełnie inne podejście do konstruowania słowa wejściowego przedstawiają M. Stojanovic i D. Stefanovic [23]. W implementacji tej operuje się na specyficznych cząsteczkach zbudowanych z *DNA*. Charakteryzują się one w odróżnieniu od poprzednich rozwiązań zupełnie inną konstrukcją – mają kształt kolisty. Podsumowując, w przypadku większości eksperymentów [1, 7, 12, 23], aby zainicjować proces przetwarzania informacji umieszczane są w probówce pojedyncze nici *DNA*, natomiast E. Shapiro [3,4] oraz N. Seeman [20], aby rozpocząć proces przetwarzania wprowadzają podwójne nici *DNA*, które posiadają tzw. lepki koniec. W każdym z podejść programowanie odbywa się przez umieszczenie w roztworze, w którym znajdują się dane wejściowe, odpowiedniego jedno- lub dwuniciowego *DNA*.

W procesie przetwarzania informacji w poszczególnych implementacjach praktycznych występują również znaczne różnice. Jednak we wszystkich podejściach cały proces przetwarzania informacji zachodzi dzięki komplementarności nici *DNA*. W podejściu L. Adlemana [1] oraz N. Seemana [20] wykorzystuje się wyłącznie komplementarność. Natomiast w pozostałych rozwiązaniach [3, 4, 7, 12, 23] zastosowano dodatkowo operację cięcia *DNA* realizowaną enzymami restrykcyjnymi.

Zakończenie procesu przetwarzania informacji zostało rozwiązane w różny sposób w poszczególnych implementacjach. U L. Adlemana [1] wyjściem jest cząsteczka *DNA* o liniowej strukturze, określonej długości oraz sekwencji zasad, którą należy wykryć w roztworze. W przypadku N. Seemana [20] proces przetwarzania kończy się odpowiednią strukturą *DNA*, będącą rozwiązaniem problemu, jest ona jednak bardziej skomplikowana (wielowymiarowa) niż w podejściu L. Adlemana. Automat E. Shapiro [3,4] kończy natomiast działanie, gdy w roztworze pojawi się odpowiednia cząsteczka terminalna, świadcząca o akceptacji słowa wejściowego. U E. Shapiro [3,4] oraz D. Faulhammera [7] mamy do czynienia z „niszczeniem" odpowiednio zaprojektowanego wcześniej słowa wejściowego. Natomiast w przypadku L. Adlemana [1] oraz N. Seemana [20] proces polega na „samowytwarzaniu", czyli automatycznym konstruowaniu struktur zbudowanych z *DNA*.

Na obecnym etapie rozwoju metod używanych w biotechnologii występują jednak duże problemy z odczytem rezultatów działań na *DNA*. Do analizy danych wyjściowych stosuje się głównie metodę PCR (reakcja polimeryzacji), za pomocą której powiela się określone sekwencje *DNA* oraz metodę elektroforezy na żelu. W przypadku implementacji przetwarzania informacji opartego na modelu samoskładania powstałe struktury są na tyle duże, że można je zobaczyć w mikroskopie elektronowym. W pozostałych metodach wykrywamy krótkie sekwencje *DNA*, więc jedyną możliwą metodą jest PCR i elektroforeza. W przypadku przetwarzania informacji uzyskanej przez M. Stojanovica i D. Stefanovica [23] zastosowano bardzo ciekawą i wymagającą dużo mniejszego nakładu pracy metodę fluorescencji.

6. Zakończenie

Warto zastanowić się nad mocnymi, a także słabymi stronami komputerów zbudowanych z *DNA*. Bardzo ważnymi zaletami *DNA* komputerów są: możliwość bardzo dobrego upakowania informacji, masowa równoległość działania, a także duża energooszczędność zachodzącego procesu. W objętości kilku kropel możliwe jest umieszczenie milionów molekularnych procesorów działających jednocześnie i wykonujących obliczenia. Niewątpliwie mocną stroną *DNA* komputerów jest również kompatybilność z organizmami żywymi, która daje duże możliwości zastosowania w biotechnologii, medycynie, czy farmacji. Wydaje się, że dzięki obecnie prowadzonym badaniom możliwe będzie w przyszłości tworzenie biochipów jednoznacznie diagnozujących choroby, takie jak nowotwory.

Do słabych stron należy zaliczyć problemy techniczne z odczytem informacji, które związane są z małym wachlarzem metod biotechnologicznych umożliwiających działanie na *DNA*. Brak jest również uniwersalnego modelu teoretycznego, dobranego do obecnych możliwości technicznych działania na *DNA*. Kluczowym problemem w rozwoju tej koncepcji budowy komputerów jest jednak brak pomysłu na ogólnodostępne zastosowanie przetwarzania informacji za pomocą *DNA*.

Mimo małego zaawansowania tej dziedziny wydaje się, że badania w tym kierunku rozwiną się znacznie w najbliższych latach. Program operacyjny Unii Europejskiej „Innowacyjna Gospodarka" na lata 2007-2013 umożliwia finansowanie projektów, zajmujących się nowymi technologiami. Daje to podstawy do tworzenia grup badawczych, a także przedsięwzięć komercyjnych skupionych wokół tematyki *DNA* komputerów. Kluczowe wydaje się jednak znalezienie prostego ogólnodostępnego zastosowania komputerów zbudowanych z *DNA*. Przedstawione badania są we wczesnym etapie rozwoju, w ciągu kilku lat należy się jednak spodziewać rozwiązań komercyjnych implementujących przedstawione pomysły.

LITERATURA

1. Adleman L.: Molecular computation of solutions to combinatorial problems. Science 226, 1994, s. 1021÷1024.
2. Amos M.: Theoretical and Experimental *DNA* Computation. Springer, Berlin, Heidelberg, New York 2005.
3. Benenson Y., Paz-Elizur T., Adar R., Keinan E., Livneh Z., Shapiro E.: Programmable and autonomous computing machine made of biomolecules. Nature 414, 2001, s. 430÷434.

4. Benenson Y., Adar R., Paz-Elizur T., Livneh Z., Shapiro E.: *DNA* molecule provides a computing machine with both data and fuel. PNAS 100, 2003, s. 2191÷2196.
5. Benenson Y., Gil B., Ben-Dor U., Adar R., Shapiro E.: An autonomous molecular computer for logical control of gene expression. Nature 429, 2004,s. 423÷429.
6. Bennett C.: Logical reversibility of computation. IBM J. R.&D. 17, 1973, s. 525÷532,.
7. Faulhammer D., Cukras A., Lipton R., Landweber L.: Molecular compution: RNA solutions to chess problems. PNAS 97, 1999, s. 1385÷1389.
8. Head T.: Formal language theory and *DNA*: an analysis of the generative capacity of specific recombinant behavior. Bulletin of Mathematical Biology 49, 1987, s. 737÷759.
9. Hopcroft J., Ullman J.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley 1979.
10. Krasiński T.: Automaty i języki formalne. Wydawnictwo UŁ, Łódź 2007.
11. Lipton R.: *DNA* solution of hard computational problems. Science 268, 1995, s. 542÷545.
12. Ogihara M., Ray A.: Simulating Boolean circuits on a *DNA* computer. Proceeding of the First Annual International Conference on Computational Molecular Biology, 1997, s. 226÷231.
13. Păun G.: On the power of the splicing operation. Internationals Journal of Computer Mathematics 59, 1995, s. 27÷35.
14. Păun G., Rozenberg G., Salomaa, A.: *DNA* Computing. New Computing Paradigms. Springer. Berlin, Heidelberg, New York 1998.
15. Păun G.: Membrane computing: An introduction. Springer, Berlin 2002.
16. Păun G.: Computing with membrane. Journal of Computer and System Science 61, 2000, s. 108÷143.
17. Rothemund P., Winfree E.: The program-size complexity of self-assembled squares (extended abstract). Proceedings of the Third-Second Annual ACM Symposiom on Theory of Computing, ACM Press, 1999, s. 459÷468.
18. Rothemund P., Papadakis N., Winfree E.: Algorithmic self-assembly of *DNA* Sierpinski triangles. PLoS Biol. 424, 2004.
19. Rothemund P.: A *DNA* and restriction enzyme implementation of Turing machines. American Mathematical Society, 1995, s. 75÷120.
20. Seeman N.: *DNA* Nicks and Nodes and Nanotechnology. Nano Letters, 2001, s. 22÷26.
21. Seeman N.: *DNA* engineering and its application to nanotechnology. Trends Biotechnol. 17, 4, 1999, s. 37÷443.
22. Soreni M., Yogev S., Kossoy E., Shoham Y., Keinan E.: Parallel biomolecular computation on surfaces with advanced finite automata. J. Am. Chem. Soc., 127, 2005, s. 3935÷3943.

23. Stojanovic M., Stefanovic D.: A Deoxyribozime-Based Molecular Automaton. Nature Biotechnology 21, 2003, s. 1069÷1074.
24. Stryer L., Tymoczko J., Berg J.: Biochemia. PWN, Warszawa 2005.
25. Unold O., Troć M., Dobosz T., Trusiewicz A.: Extended molecular computing model. WSEAS Trans. Biol. Biomed. 1, 2004, s. 15÷19.
26. Winfree E., Liu F., Wenzler L., Seeman N.: Design and self-assembly of twodimensional *DNA* crystals. Nature 394, 1998, s. 539÷544.
27. Węgrzyn S., Graja J., Bugajski S., Gibas M., Winiarczyk R., Znamirowski L., Miszczak J., Nowak S.: Nano i kwantowe systemy informatyki. Wyd. Pol. Śl., Gliwice 2003.
28. Węgrzyn S., Znamirowski L.: Zarys nanonauki i informatycznych molekularnych nanotechnologii. Wyd. Pol. Śl., Gliwice 2007.

Recenzent: Dr hab.inż. Lech Znamirowski

Wpłynęło do Redakcji 1 marca 2008 r.

Abstract

The paper is a review on *DNA* computing. In the first part (Section 2) we briefly describe the structure and basic operations on *DNA*. Next (Section 3) we present some theoretical models of *DNA* computation. We describe: filtering model, splicing system, tile assembly model and membrane model. In Section 4 we give short descriptions of chosen practical implementations of *DNA* computation. They are: Adleman experiment, Shapiro automaton, Stojanovic and Stefanovic molecular automaton MAYA, Ogihara and Ray boolean circuit model, Faulhammer chess problem and Seeman nanostructures. In Section 5 we compare practical implementations of *DNA* computation with respect to: ways of encoding, input data, methods of processing and ways of obtaining outputs (results of computations). In conclusion we describe weak and strong sides of *DNA* computation.

Adresy

Tadeusz KRASIŃSKI: Uniwersytet Łódzki, Wydział Matematyki i Informatyki, Banacha 22, Polska, 90-238 Łódź, krasinsk@uni.lodz.pl
Sebastian SAKOWSKI: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, sebastian.sakowski@op.pl