Sprawozdanie
Bioinformatyka 2025

Sekwencjonowanie przez hybrydyzacje
Binary Chip
z błędami negatywnymi

Marcin Byczyński 155888,
Szymon Giernalczyk 155890,
Grupa L8

1. Opis i formalizacja problemu

Sekwencjonowanie przez hybrydyzację to metoda określania sekwencji DNA oparta na zjawisku łączenia się (hybrydyzacji) krótkich sond oligonukleotydowych z komplementarnymi fragmentami badanego DNA. W praktyce stosuje się zestaw sond o znanej sekwencji, które są przytwierdzone do powierzchni (np. mikromacierzy). Próbkę DNA znakuje się i doprowadza do kontaktu z sondami – sondy, które pasują do fragmentów DNA, tworzą stabilne dupleksy. Na podstawie informacji, które sondy związały się z DNA, komputer odtwarza pełną sekwencję badanego fragmentu.

W klasycznej wersji problemu sekwencjonowania dostajemy listę wszystkich oligonukleotydów (k-merów), które występują w badanej próbce. Nie mamy informacji o ich kolejności, ale wiemy, że każdy z nich gdzieś występuje w docelowej sekwencji. Tworzymy graf, gdzie każdy k-mer to wierzchołek. Łączymy dwa k-mery krawędzią, jeśli ostatnie $k-1$ nukleotydów jednego pokrywają się z pierwszymi $k-1$ drugiego (czyli mają tzw. nakładkę). Po skonstruowaniu grafu w ten sposób, celem jest znalezienie takiej ścieżki przez graf, która odwiedza wszystkie k-mery i odtworzy pełną sekwencję.

Wykorzystanie Binary Chip'a różni się nieco od klasycznego podejścia. W podejściu klasycznym zakładamy, że sonda zwraca nam gotowe oligonukleotydy, które następnie korzystając z wybranego algorytmu połączymy w ostateczną sekwencję. Binary Chip zaś, opiera się na dwóch podziałach nukleotydów:

- ze względu na siłę wiązań: Z = { W, S }

  W (weak) = { A, T } & S (strong) = { C, G }

- ze względu na typ zasady azotowej: P = { R, Y }

  R (puryny) = { A, G } & Y (purymidyny) = { C, T }

Bazując na tym podziale, chip posiada 2 sondy, które analizują próbkę według tych dwóch komplementarnych spektr:

- Spektrum Z: sonda $S_Z \rightarrow Z_1 Z_2 \ldots\ldots Z_k Z_{N+1}$
- Spektrum P: sonda $S_P \rightarrow P_1 P_2 \ldots\ldots P_k P_{N+1}$

gdzie N oznacza dowolny nukleotyd. Ten ostatni pełny nukleotyd pełni ważną funkcję, a mianowicie jest on elementem weryfikującym i pozwala na odpowiednie połączenie wyników z obu spektr.

Ideą tego podejścia jest odpowiednie łączenie informacji z dwóch spektr, co pozwoli na zidentyfikowanie znalezionych w próbce nukleotydów:

$$W \cap R = \{A\}$$

$$W \cap Y = \{T\}$$

$$S \cap R = \{G\}$$

$$S \cap Y = \{C\}$$

W naszym problemie występują **błędy negatywne.** Ich obecność niesie za sobą potencjalne problemy:

- sygnał na sondzie jest zbyt słaby, przez co oligonukleotyd może nie zostać zauważony

- nie mamy informacji o liczbie wystąpień danego oligonukleotydu, zatem w sytuacji w której dany oligonukleotyd pojawia się w próbce więcej niż raz, nie wiemy o tym

2. Podejście dokładne

Algorytm dokładny ma na celu znalezienie za każdym razem najlepszego dostępnego rozwiązania. Przeszukuje on wszystkie możliwe kombinacje oligonukleotydów, co gwarantuje, że jeśli istnieje rozwiązanie spełniające wszystkie warunki (długość sekwencji, liczba dopuszczalnych błędów), to je znajdzie. Nasza implementacja algorytmu dokładnego działa w następujący sposób:

I. **Wczytanie danych**:
   Program pobiera dane z pliku, w którym znajdują się dwa zbiory oligonukleotydów:
   A. Z – z informacją o sile wiązań (W/S)
   B. P – z informacją o rodzaju zasady (R/Y)

II. **Konwersja i przygotowanie**:
Program przekształca oligonukleotydy do odpowiednich form WS i RY, a następnie ustawia tzw. fragment startowy, od którego rozpocznie składanie.

III. **Budowanie ścieżki rozwiązania**:
Program łączy kolejne fragmenty na podstawie ich końców i początków – jeśli się „nakładają" (ang. *overlap*), to mogą zostać połączone.

IV. **Sprawdzanie poprawności**:
Dla każdego nowego fragmentu:
   A. Obliczana jest długość obecnej sekwencji.
   B. Sprawdzana jest liczba tzw. błędów negatywnych (czyli dopasowań, które nie są idealne).

V. **Rekursywne przeszukiwanie**:
Program kontynuuje składanie sekwencji rekurencyjnie (czyli w sposób „w głąb", jak drzewo decyzyjne), aż osiągnie pełną długość lub zakończy możliwości.

Podsumowując, nasz algorytm realizuje podejście dokładne do problemu sekwencjonowania przez hybrydyzację – przeszukuje wszystkie możliwe poprawne ścieżki budujące pełną sekwencję DNA, na podstawie danych z dwóch częściowych spektrów (WS i RY). Nie stosuje uproszczeń ani losowych prób – działa deterministycznie, przez co może działać wolniej, ale gwarantuje znalezienie najlepszego rozwiązania, jeśli takie istnieje.

3. Podejście heurystyczne

Wykorzystany przez nas algorytm heurystyczny to przeszukiwanie Tabu (ang. Tabu Search). Wybraliśmy ten algorytm ponieważ umożliwia on elastyczne dostosowanie działania w zależności od priorytetów użytkownika — pozwala samodzielnie decydować, czy większy nacisk położyć na dokładność i jakość rozwiązania, czy na szybkość jego uzyskania. Możemy to dostosowywać zmieniając liczbę iteracji algorytmu tabu - więcej iteracji to większa dokładność, i vice versa.

Opis działania algorytmu:

I. **Wczytanie danych, konwersja i przygotowanie**:
Program pobiera dane z pliku, w którym znajdują się dwa zbiory oligonukleotydów:
   A. Z – z informacją o sile wiązań (W/S)
   B. P – z informacją o rodzaju zasady (R/Y)
Program przekształca oligonukleotydy do odpowiednich form WS i RY, a następnie ustawia tzw. fragment startowy, od którego rozpocznie składanie.

II. **Wygenerowanie rozwiązania startowego algorytmem Greedy:**
Program wybiera kolejne elementy na podstawie lokalnie najlepszej decyzji, czyli minimalnego kosztu w danym kroku. To rozwiązanie startowe jest

następnie wykorzystywane przez algorytm **t**abu search, który próbuje je ulepszyć, przeszukując sąsiedztwo i unikając powrotu do niedawnych stanów.

III. **Iteracje algorytmu Tabu:**
   A. **Generowanie sąsiadów -** odbywa się poprzez usunięcie lub dodanie oligonukleotydu/klastra, bądź też ich przesunięcie w kierunku ściślejszego upakowania lub rozciągnięcia
   B. **Ocena sąsiadów oraz wybór najlepszego rozwiązania -** obliczenie wartości funkcji oceny - globalnej lub zagęszczenia - dla każdego sąsiedniego rozwiązania, aby określić ich jakość. Następnie algorytm wybiera najlepsze rozwiązanie spośród nich, z uwzględnieniem ograniczeń tabu, czyli pomija ruchy zabronione przez listę tabu, aby uniknąć powrotu do wcześniej odwiedzonych stanów.

Powyższy proces oceny oraz wyboru najlepszego sąsiada jest powtarzany przez ustaloną liczbę iteracji. Proces ten służy stopniowemu ulepszaniu rozwiązania w ramach metaheurystyki tabu search.

IV. **Sprawdzenie, czy udało się znaleźć idealne rozwiązanie —** czyli takie, które ma długość dokładnie równą oczekiwanej i liczbę błędów negatywnych (zgodną z wartością referencyjną . Jeśli tak, algorytm zwraca to rozwiązanie jako końcowe.

V. **Zwrócenie najlepszego rozwiązania spośród wszystkich napotkanych w trakcie iteracji -** jeżeli rozwiązanie idealne nie zostało znalezione, wówczas jako najlepsze, wybierane jest takie rozwiązanie, które ma możliwie największą długość zrekonstruowanego DNA, i jak najbardziej zbliżoną (minimalnie odchyloną) liczbę błędów negatywnych względem oczekiwanej.

4. Analiza wyników

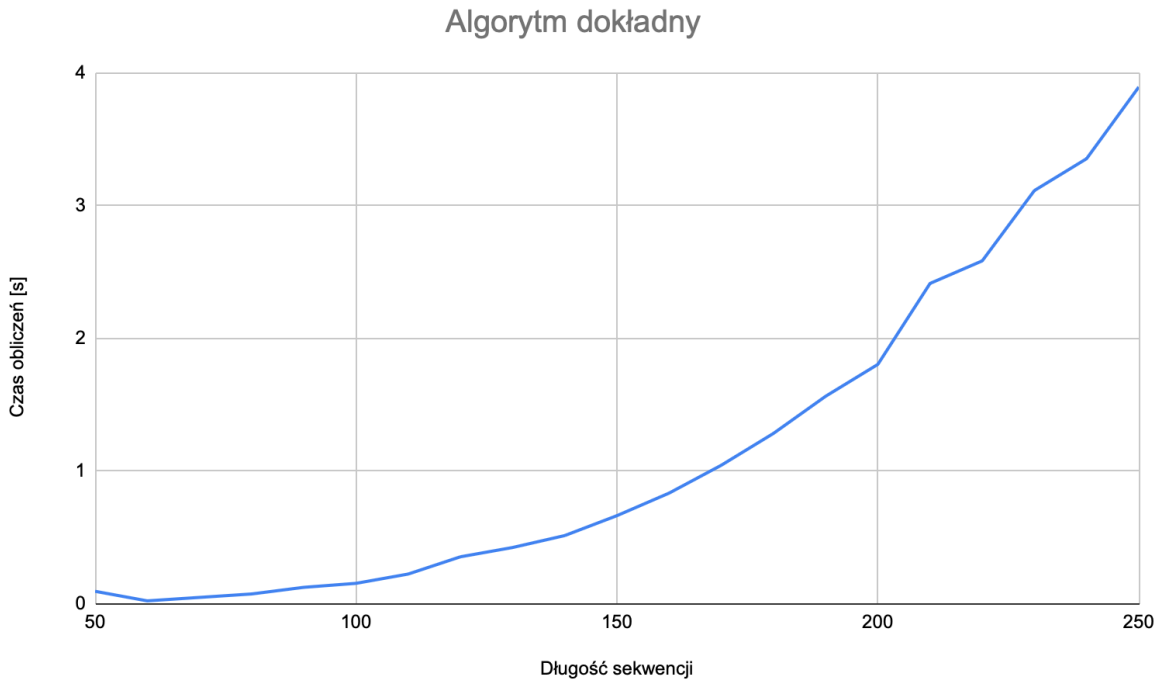
W celu przetestowania obydwu algorytmów pobrano dane testowe z linku www.cs.put.poznan.pl/pwawrzyniak/bio/bio.php. Dla każdego rozmiaru instancji, rozmiar oligonukleotydu k, był zawsze ustawiony na taką samą wartość (k =10). Również współczynnik błędów negatywnych dla każdej instancji był identyczny (sqnep = 10 %).

Algorytm dokładny:

I. Algorytm dokładny wykazuje znaczną niestabilność czasową. Pomimo identycznego rozmiaru danych wejściowych, czas jego działania może się istotnie różnić w zależności od konkretnej instancji problemu. W praktyce obserwowano przypadki, w których rozwiązanie znajdowane było w czasie poniżej jednej sekundy, jak i takie, które wymagały nawet kilkunastu sekund obliczeń. Zjawisko to wynika prawdopodobnie z różnic w strukturze instancji, wpływających na głębokość oraz rozgałęzienia drzewa rekursji.

II. Algorytm dokładny charakteryzuje się pełną deterministycznością — dla danej instancji danych wejściowych zawsze zwraca identyczne rozwiązanie. Właściwość ta zapewnia powtarzalność wyników oraz możliwość ich jednoznacznej weryfikacji i porównania między różnymi uruchomieniami.

III. Ze względu na znaczny rozrzut czasów wykonania, na wykresie uwzględniono jedynie najlepsze (najkrótsze) czasy działania algorytmu dla każdej konfiguracji. Umożliwiło to przejrzyste przedstawienie wyników oraz zachowanie czytelności wizualizacji. Uwzględnienie wszystkich wartości skutkowałoby zniekształceniem skali i utrudniłoby interpretację.

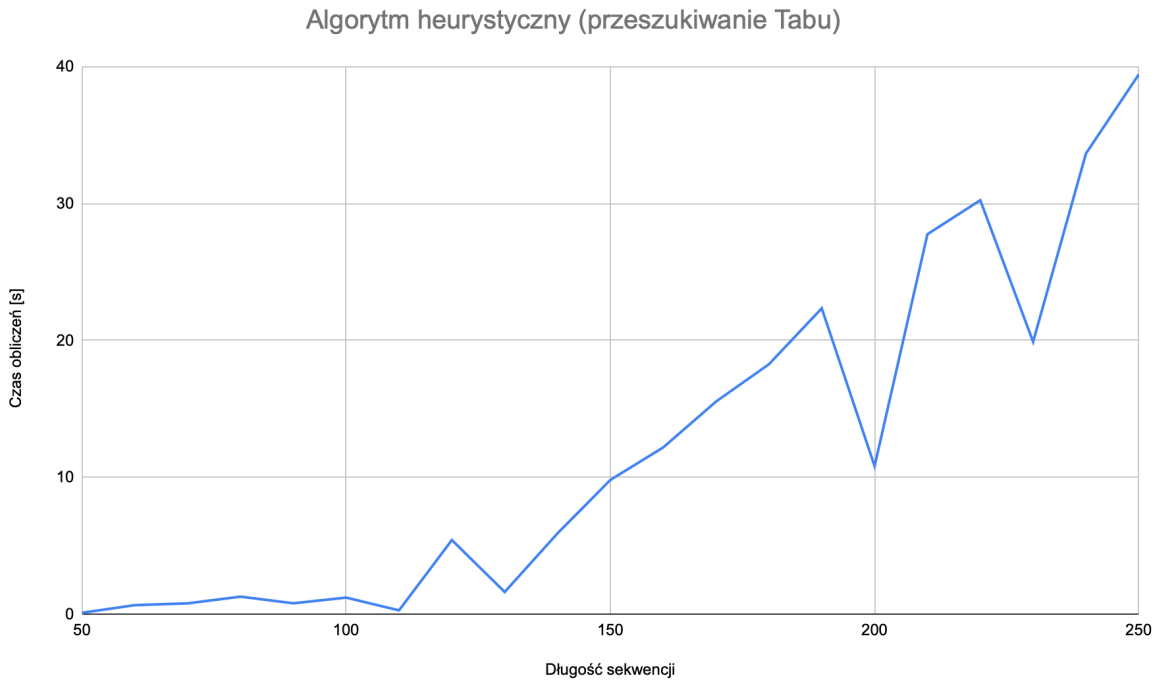


Algorytm heurystyczny (przeszukiwanie Tabu)

I. Wszystkie testy algorytmu przeszukiwania Tabu wykonane były z następującymi parametrami: liczba iteracji = 30, liczba generowanych sąsiadów = 30.

II. Ze względu na odgórnie ograniczoną liczbę iteracji, czas działania algorytmu przeszukiwania Tabu dla danej instancji nigdy nie przekroczy pewnej wartości. Może jedynie ulec skróceniu, jeżeli idealne rozwiązanie zostanie znalezione przed wykonaniem wszystkich iteracji - wówczas program kończy działanie, i zwraca znalezione rozwiązanie. Z tego wynikają nieregularności na wykresie. Co prawda widać pewną tendencję wzrostową na wykresie, jednak często pojawiają się również spadki, czyli znacznie krótsze czasy wyszukiwania rozwiązania.

III. Choć algorytm tabu nie zawsze generował rozwiązanie idealne, uzyskiwane wyniki zazwyczaj charakteryzowały się długością bardzo bliską długości oczekiwanej (najczęściej różnica nie przekraczała 10 nukleotydów). Niemniej jednak, w wielu przypadkach jakość nałożeń oligonukleotydów pozostawiała pewne niedoskonałości — sekwencje końcowe zawierały więcej błędów negatywnych niż wynik optymalny.



Podsumowując, algorytm dokładny, mimo że w wielu przypadkach potrafił znaleźć rozwiązanie szybciej niż algorytm heurystyczny (Tabu Search), cechuje się znaczną niestabilnością czasową. Dla instancji o tych samych parametrach wejściowych, ale różnej zawartości, czas działania algorytmu dokładnego mógł różnić się od ułamka sekundy do kilkunastu sekund. Zjawisko to staje się coraz bardziej widoczne wraz ze wzrostem rozmiaru instancji — im dłuższa sekwencja do odtworzenia, tym większy rozrzut czasów wykonania, co znacząco utrudnia przewidywanie wydajności.

W przeciwieństwie do tego, algorytm heurystyczny działał w sposób stabilny i przewidywalny dzięki ograniczeniu liczby iteracji. Choć nie zawsze dostarczał rozwiązania idealnego, to jakość otrzymanych sekwencji była satysfakcjonująca, a czas działania — w pełni kontrolowany. Z tego względu, mimo pewnych kompromisów na poziomie dokładności, w zastosowaniach praktycznych algorytm heurystyczny okazuje się bardziej użyteczny i niezawodny niż podejście dokładne.