Sekwencjonowanie poprzez hybrydyzację z użyciem chipów alternatywnych z błędami pozytywnymi

Łukasz Bialik 155956
Aleksander Szymaniak 155922

Opis i sformalizowanie problemu

Celem zadania było rozwiązanie problemu sekwencjonowania DNA metodą sekwencjonowania przez hybrydyzację, przy wykorzystaniu tzw. chipów alternatywnych oraz uwzględnieniem obecności błędów pozytywnych.

Problem SBH (sequencing by hybridization) polega na odtworzeniu nieznanej sekwencji DNA na podstawie zbioru krótkich fragmentów (oligonukleotydów), które były w stanie związać się z sekwencją badaną podczas eksperymentu hybrydyzacji. W klasycznej wersji SBH, chip zawiera jedno spektrum, tj. zbiór wszystkich oligonukleotydów długości *k*, które są obecne w sekwencji. W naszym wariancie rozważaliśmy chip alternatywny, w którym generowane są dwa osobne widma (S1, S2), zawierające oligonukleotydy pochodzące z tej samej sekwencji docelowej. Każde z widm jest tworzone w oparciu o inną regułę maskowania, co daje dwa różne, wzajemnie uzupełniające się spojrzenia na strukturę sekwencji..

Dodatkowym utrudnieniem były błędy pozytywne - czyli oligonukleotydy, które nie występują faktycznie w sekwencji DNA, ale znalazły się w widmie wskutek błędu eksperymentalnego (np. niespecyficznego wiązania lub niedoskonałości technologii chipu). Obecność takich błędów oznacza, że proces odtwarzania sekwencji wymaga nie tylko poprawnego złożenia fragmentów, ale także unikania lub detekcji oligonukleotydów zbytecznych.

Dane wejściowe:

- Długość sekwencji *N* do odtworzenia,
- sekwencja początkowa *start_seq* (fragment znanej sekwencji, pierwsze *k* znaków odtwarzanej sekwencji),
- zbiór oligonukleotydów S1,
- zbiór oligonukleotydów S2,

W przeciwieństwie do klasycznej wersji sekwencjonowania przez hybrydyzację (SBH), tutaj mamy do czynienia z tzw. **alternatywnym chipem**, w którym:

- sondy z S1 mają długość `k1` i strukturę typu `N X N X N X N` (czyli naprzemiennie nukleotyd rzeczywisty i pozycja maskowana lub dowolna),
- sondy z S2 mają długość `k2 < k1 + 1`, przy czym zawierają dodatkowy nukleotyd na końcu – `N X N X N N`.

Cel:

Zbudować pełną sekwencję DNA długości *N (w algorytmie heurystycznym - jak najdłuższą)*, dobierając kolejne oligonukleotydy z *widm*, tak aby kolejne oligonukleotydy nakładały się zgodnie z regułami hybrydyzacji.

Minimalizować liczbę wykorzystanych błędnych oligonukleotydów (tzw. **błędów pozytywnych**), tj. przypadków, gdzie oligonukleotyd nie pasuje dokładnie do odpowiedniego fragmentu odtwarzanej sekwencji.

Formalizacja:

Można przedstawić problem jako poszukiwanie ścieżki w grafie nakładania się oligonukleotydów, która rekonstruuje sekwencję o zadanej długości *N*, przy możliwie minimalnej liczbie błędów pozytywnych i bez pomijania fragmentów.

Problem należy do kategorii problemów **NP-trudnych**, ponieważ w ogólności nie da się w czasie wielomianowym zagwarantować znalezienia optymalnego rozwiązania dla dowolnego układu błędów i kombinacji oligonukleotydów.

W celu rozwiązania tego problemu zaimplementowaliśmy:

algorytm dokładny — pełne przeszukiwanie przestrzeni rozwiązań, z deterministycznym wyborem fragmentów i weryfikacją poprawności,

algorytm heurystyczny (ACO) — oparty o metaheurystykę **algorytmu mrówkowego**, symulujący zachowanie populacji sztucznych mrówek, które budują rozwiązania na podstawie reguł probabilistycznych i informacji zawartych w feromonach. Algorytm ten pozwala na efektywne poszukiwanie dobrych rozwiązań w obecności błędów i dużych zbiorów danych.

Opis algorytmu dokładnego

Do rozwiązania postawionego problemu zastosowano algorytm dokładny oparty na metodzie **rekurencyjnego przeszukiwania z powrotem (backtracking)**. Jego celem jest odtworzenie docelowej sekwencji DNA o długości N, rozpoczynając od znanego fragmentu (`start`), analizując poszczególne próbki z dwóch spektrów S1 oraz S2

Istnienie 2 spektrów o różnych formach wymusza specjalny sposób łączenia oligonukleotydów w celu rekonstrukcji oryginalnej sekwencji – rozszerzanie możliwe jest tylko w sytuacji, gdy końcówka aktualnej sekwencji pokrywa się (z tolerancją pozycji „X") z kandydatem z S1, a ostatni znak obu sond (`c1[-1]` i `c2[-1]`) jest identyczny.

Działanie algorytmu:

1. **Start od znanej sekwencji:** Algorytm rozpoczyna z fragmentem początkowym (`start`).

2. **Dobór kandydatów:** Dla każdej iteracji algorytm wyszukuje możliwe sondy z S1 i S2, które mogą zostać dołączone do sekwencji – czyli takie, których początek (bez ostatniego znaku) może pokrywać się z końcówką sekwencji (uwzględniając, że 'X' jest zgodny z dowolnym znakiem).

3. **Szukane są zgodne pary:** Następnie tworzone są pary `(c1, c2)` z S1 i S2, które mają ten sam ostatni znak. Tylko takie pary mogą być użyte do rozszerzenia sekwencji.

4. **Podejście deterministyczne i rekurencyjne:**

    - Jeśli istnieje tylko jedna pasująca para – zostaje ona użyta bez rekurencji, a sekwencja jest wydłużana.

    - W przypadku wielu par – algorytm wchodzi w fazę rekurencyjnego przeszukiwania, próbując każdą możliwą ścieżkę kontynuacji sekwencji.

5. **Odrzucanie błędnych ścieżek:** Jeżeli dla danej ścieżki nie da się znaleźć dalszego poprawnego rozszerzenia (brak odpowiedniej pary), algorytm kończy swoje działanie co pozwala unikać oligonukleotydów pochodzących z błędów pozytywnych.

Specyfika wynikająca z budowy chipu:

Ze względu na odmienną długość i przesunięcie sond S1 i S2, poprawne złożenie wymaga, by końcówka sekwencji była zgodna jednocześnie z początkiem sondy z S1 oraz sondy z S2, a zgodność oznacza zgodność znaków rzeczywistych oraz możliwość dopasowania w obecności pozycji 'X'.

Ten warunek znacząco zawęża liczbę możliwych kontynuacji, a jednocześnie sprawia, że algorytm może skutecznie eliminować błędne sondy nawet przy obecności błędów pozytywnych.

Złożoność i zastosowanie:

- **Złożoność:** Algorytm ma charakter wykładniczy w najgorszym przypadku ze względu na rozgałęziające się przeszukiwanie przestrzeni rozwiązań. Jednak dzięki optymalizacji (np. wcześniejsze odrzucanie kandydatów, jednoznaczne rozszerzenia bez rekurencji), jego działanie w praktyce może być wystarczająco szybkie dla umiarkowanych długości sekwencji.

- **Zastosowanie:** To podejście znajduje zastosowanie w rekonstrukcji sekwencji DNA z danych eksperymentalnych pochodzących z alternatywnych chipów hybrydyzacyjnych, gdzie obecność błędów i specyficzna struktura danych wyklucza stosowanie prostszych metod heurystycznych.

Opis algorytmu wielomianowego (przybliżonego)

Do rozwiązania problemu sekwencjonowania z błędami pozytywnymi zastosowaliśmy również **algorytm heurystyczny** oparty na metaheurystyce **Ant Colony Optimization (ACO)**. Jest to algorytm wielomianowy, przybliżony, inspirowany zachowaniem kolonii mrówek poszukujących najkrótszej ścieżki do źródła pożywienia.

W naszym przypadku mrówki reprezentują symulowane jednostki konstruujące sekwencję DNA, dokładając kolejne oligonukleotydy z widma **combined_spectrum**, czyli połączonych fragmentów z dwóch chipów alternatywnych (S1, S2). Każda mrówka zaczyna od zadanej sekwencji początkowej i stara się przedłużać ją o jeden nukleotyd na raz, wybierając w każdej iteracji taki oligonukleotyd, który pasuje (częściowo) do aktualnego sufiksu sekwencji.

Wybór kolejnego oligonukleotydu odbywa się w oparciu o:

- **feromon**: informacja o skuteczności danego oligo w poprzednich iteracjach,
- **czynnik heurystyczny**: opcjonalna wartość preferująca np. krótsze dopasowania, penalizująca błędy itp.
- **regułę probabilistyczną**: każdy oligo ma przypisaną wagę będącą funkcją feromonu podniesionego do potęgi α i heurystyki do potęgi β.

$$p_j = \frac{(\tau_j^\alpha) \cdot (\eta_j^\beta)}{\sum_{j \in \Lambda} (\tau_j^\alpha) \cdot (\eta_j^\beta)}$$

τ - aktualna ilość feromonu przypisana oligonukleotydowi

η - wartość heurystyki dla oligonukleotydu *j* ,

α— parametr sterujący wpływem feromonu,

β— parametr sterujący wpływem heurystyki,

□— zbiór oligonukleotydów, które można aktualnie dołożyć do sekwencji (pasują sufiksem).

Po każdej iteracji feromony odparowują, a następnie są wzmacniane na podstawie wyników wszystkich mrówek — im dłuższą (i bardziej poprawną) sekwencję zbudowała dana mrówka, tym więcej feromonu pozostawia na wykorzystanych oligonukleotydach.

Cały algorytm działa w *n_iterations* cyklach, a w każdej iteracji startuje *n_ants* mrówek. Najlepsza znaleziona sekwencja (najdłuższa) jest zapamiętywana. Algorytm kończy działanie, gdy osiągnięta zostanie pełna długość docelowej sekwencji lub gdy wyczerpana zostanie liczba iteracji.

Złożoność obliczeniowa:

Złożoność czasowa algorytmu heurystycznego ACO w najgorszym przypadku wynosi:

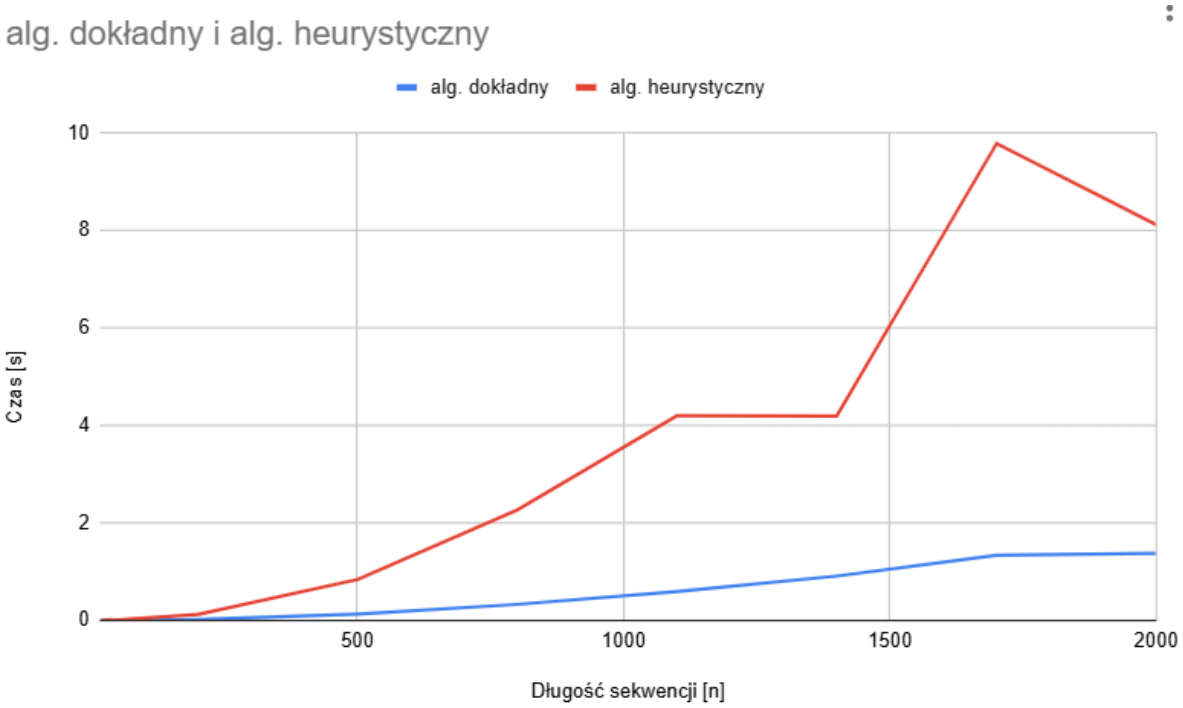
O(n_ants × n_iterations × |spectrum|)

gdzie |spectrum| to liczba oligonukleotydów w połączonym widmie. Ponieważ algorytm nie przeszukuje wszystkich możliwych kombinacji, ale prowadzi eksplorację sterowaną heurystyką i feromonami, jego czas działania jest wielomianowy względem rozmiaru danych, co czyni go efektywnym nawet dla dużych instancji.

Zalety podejścia:

- działa dobrze w obecności błędów pozytywnych (możliwość pominięcia błędnych oligonukleotydów),
- pozwala na szybkie uzyskanie przybliżonego rozwiązania,
- nie wymaga pełnego przeszukiwania przestrzeni rozwiązań.

Analizę uzyskanych wyników



Na wykresie przedstawiono zależność czasu działania (w sekundach) obu zaimplementowanych algorytmów w funkcji długości sekwencji *n*. Porównano algorytm dokładny oraz algorytm heurystyczny (mrówkowy).

Na podstawie wyników można sformułować następujące wnioski:

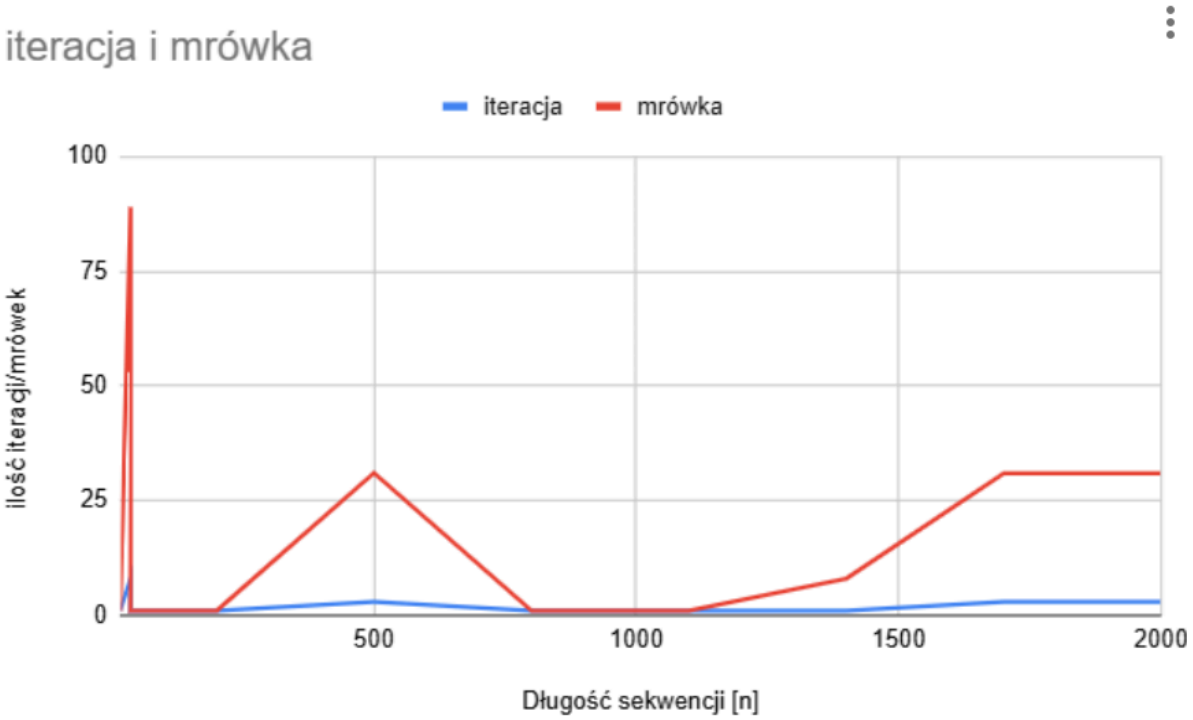
1. **Algorytm dokładny charakteryzuje się znacznie bardziej stabilnym czasem działania**. Czas rośnie stosunkowo powoli wraz ze wzrostem długości sekwencji. Nawet dla największych rozważanych długości (do 2000 nukleotydów), wzrost jest umiarkowany i niemal liniowy. Wynika to z faktu, że w przypadku poprawnych danych i dobrze zaprojektowanego widma, proces przeszukiwania ma często charakter deterministyczny, co ogranicza liczbę wywołań rekurencyjnych.

2. **Algorytm heurystyczny (ACO) wykazuje dużo większe fluktuacje czasowe** i dla większych długości sekwencji jego czas działania znacząco przewyższa czas działania algorytmu dokładnego. Jest to zgodne z charakterem algorytmu ACO - dla długich sekwencji przestrzeń poszukiwań staje się bardzo duża, a znalezienie pełnej sekwencji wymaga wielu iteracji i przebiegów wielu mrówek.

3. Dla krótkich sekwencji (do około 200 nukleotydów) oba algorytmy działają w porównywalnym czasie, jednak od długości około 500 nukleotydów przewaga algorytmu dokładnego w czasie działania staje się wyraźna.

4. Warto zauważyć, że pomimo wielomianowej złożoności nominalnej algorytmu ACO, w praktyce dla tego problemu sekwencjonowania z błędami pozytywnymi jego czas działania okazuje się wyraźnie mniej korzystny niż czas działania zoptymalizowanego algorytmu deterministycznego z backtrackingiem.

5. Z punktu widzenia praktycznego, dla danych o wysokiej jakości (czyli przy niewielkiej liczbie błędów pozytywnych), stosowanie algorytmu dokładnego może być korzystniejsze — zapewnia on deterministyczne i szybkie rozwiązanie. Natomiast algorytm heurystyczny może być bardziej elastyczny przy większych poziomach szumu i bardziej skomplikowanych instancjach, ale kosztem większego czasu obliczeń.



Na drugim wykresie przedstawiono, w jaki sposób zmieniała się liczba iteracji oraz liczba użytych mrówek w algorytmie heurystycznym w zależności od długości sekwencji *n*. Wartości na osi pionowej reprezentują względną liczbę iteracji i mrówek potrzebnych do znalezienia rozwiązania.

Na podstawie wyników można zauważyć następujące zależności:

1. Liczba iteracji algorytmu ACO pozostaje stosunkowo stała i niska w całym zakresie długości sekwencji. Oznacza to, że proces globalny (liczba pełnych cykli uczenia populacji mrówek) jest raczej stabilny, niezależnie od długości docelowej sekwencji.

2. Natomiast liczba użytych mrówek (czyli liczba mrówek, które skutecznie znalazły ścieżki składające się na najlepsze dotychczasowe rozwiązanie) wykazuje silne fluktuacje. W szczególności można zaobserwować bardzo duże wartości dla “trudnych” sekwencji, czyli takich w których błędy pozytywne bardzo utrudniają znalezienie rozwiązania.

3. Zmienność liczby użytych mrówek odzwierciedla fakt, że algorytm ACO jest algorytmem probabilistycznym - trudniejsze dane wejściowe prowadzą do większego rozrzutu wyników pomiędzy kolejnymi mrówkami.

Podsumowując, wykres ten potwierdza charakterystyczne cechy algorytmu ACO: stabilność na poziomie globalnym (iteracje), przy jednocześnie dużej zmienności na poziomie lokalnym (aktywność mrówek), zwłaszcza w przypadku bardziej złożonych lub dłuższych instancji problemu.