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) pokrywa 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. Użyta para jest usunięta z pamięci w celu przyśpieszenia działania algorytmu.

    - 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.

6. **Koniec działania algorytmu:** algorytm kończy swoje działanie po znalezieniu pierwszej sekwencji o oczekiwanej długości. Dzięki deterministycznemu doborowi kandydatów, mamy pewność, że będzie to rozwiązanie poprawne.

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)

Wstęp:

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 do aktualnego sufiksu sekwencji.

Budowa combined_spectrum:

Iterując po wszystkich parach oligonukleotydów z dwóch spektrów, szukamy takich, których początki (dla pierwszego spektrum jest to ciąg bez dwóch ostatnich znaków, a dla drugiego bez ostatniego znaku), są równe.

Wszystkie takie pary scalamy w jeden oligonukleotyd, który wygląda tak jak ten z pierwszego spektrum, ale posiada o jedną zasadę azotową więcej w miejscu ostatniej niewiadomej. To spektrum zazwyczaj (czasami w sytuacji wielu błędów pozytywnych jest inaczej) jest mniej liczne niż sumaryczna liczność spektra pierwszego i drugiego, dzięki czemu ograniczamy przetwarzanie w algorytmie mrówkowym.

To podejście ma też swoje minusy. Jest to kolejna sekwencja operacji o złożoności O(n^2), a także tworzy pary na podstawie błędów pozytywnych i przekazuje je do algorytmu mrówkowego. Eliminacja błędów odbywa się dopiero tam.

Przykład scalania spektrów:

```
A X C X G X G + -> spectrum 1
A X C X G G =   -> spectrum 2
A X C X G G G   -> combined_spectrum
```

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|)

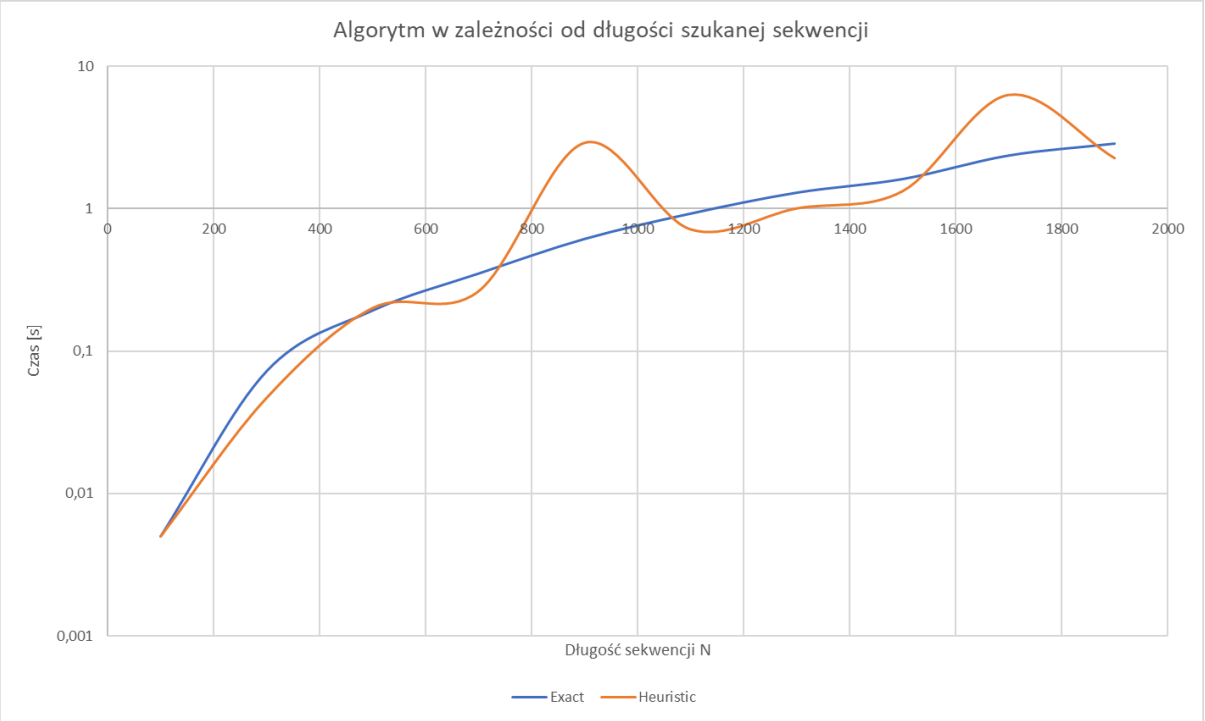
gdzie |spectrum| to liczba oligonukleotydów w połączonym widmie. Ponieważ algorytm nie przeszukuje wszystkich możliwych kombinacji, ale prowadzi eksplorację sterowaną heurystyką i feromonami, jego czas działania jest wielomianowy względem rozmiaru danych, co czyni go efektywnym nawet dla dużych instancji.

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

Analiza zmienności czasu algorytmów dla zmiennej długości sekwencji (n):



Dane wejściowe: N - [100, 1900], SQPE - N*25%, K -10

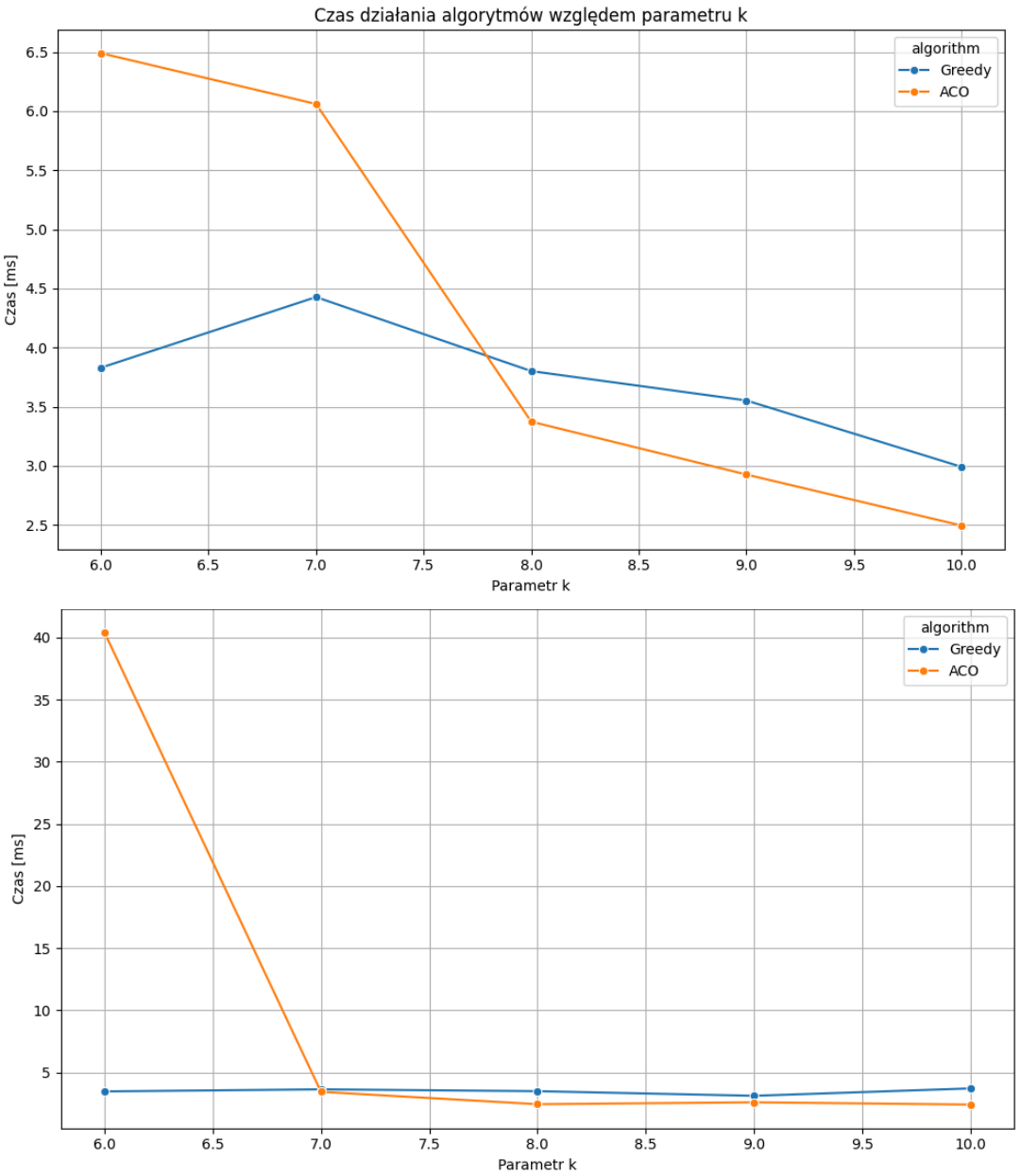
Na wykresie przedstawiono wpływ długości docelowej sekwencji N na czas działania algorytmów Exact oraz Heuristic. W obu przypadkach czas rośnie wraz z wydłużaniem sekwencji, jednak wzrost ten nie jest liniowy.

Algorytm Exact wykazuje stabilny i przewidywalny wzrost czasu – jego charakterystyka jest gładka i monotoniczna.

Heurystyczny algorytm potrafi działać szybciej niż metoda dokładna, jednak jego nieregularność czasowa może czasami prowadzić do gorszej wydajności. Wynika to prawdopodobnie z trudności w znalezieniu optymalnej ścieżki w bardziej złożonej przestrzeni rozwiązań, co skutkuje koniecznością eksploracji większej liczby wariantów.

Podsumowując, heurystyka oferuje lepszą wydajność w sprzyjających warunkach, ale jej skuteczność zależy silnie od przypadku, podczas gdy algorytm Exact zapewnia większą stabilność kosztem ogólnego czasu działania.

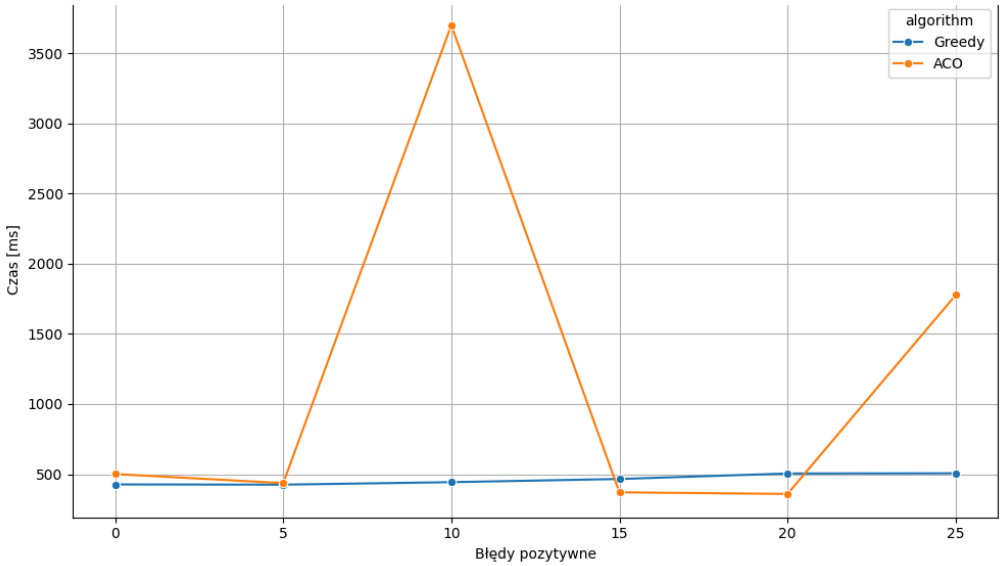
Analiza zmienności czasu algorytmów dla zmiennej długości oligonukleotydu (k):

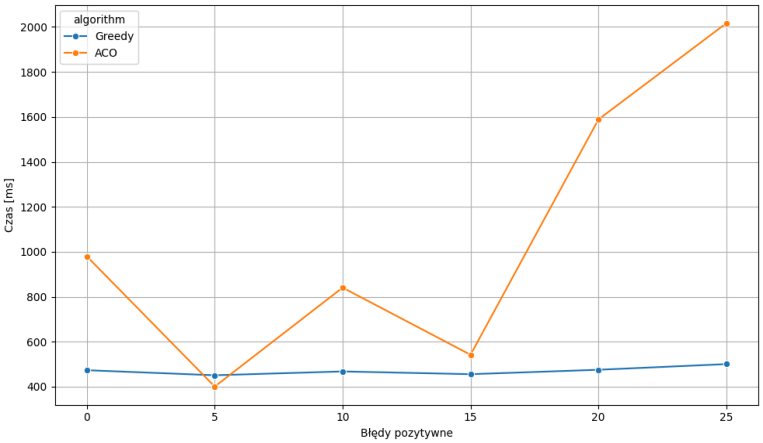


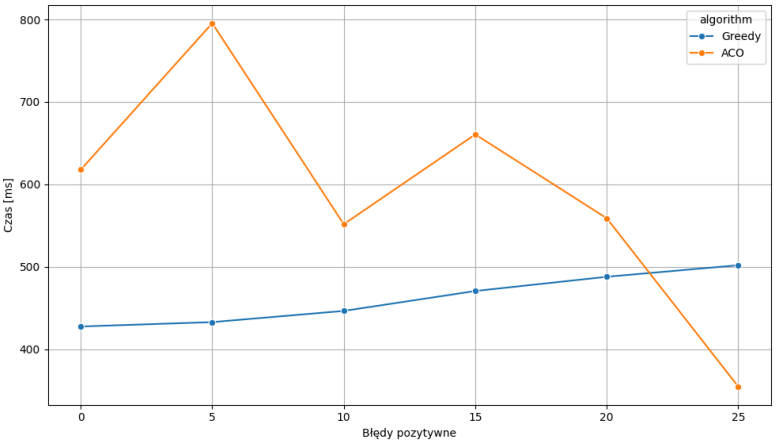
Dane wejściowe: N - 1000, SQPE - 0, K - [6, 10]

Na podstawie obu wykresów można zauważyć, że czas działania algorytmów ACO(heuristic) i Greedy(exact) w ogólności delikatnie maleje wraz ze wzrostem parametru k. Dla małych wartości k (krótkie i mniej precyzyjne wzorce) algorytmy działają wolniej – wynika to z większej liczby możliwych połączeń, nawet bez obecności błędów pozytywnych. Wraz ze wzrostem k, wzorzec staje się bardziej restrykcyjny, co znacząco redukuje przestrzeń przeszukiwań i powoduje spadek czasu działania, szczególnie zauważalny w przypadku ACO Algorytm Greedy działa dość stabilnie, ale zazwyczaj nieco wolniej niż ACO. Heurystyka okazuje się więc szybsza, ale mniej stabilna dla krótkich, mało informacyjnych wzorców.

Analiza zmienności czasu algorytmów dla zmiennej ilości błędów pozytywnych:







Dane wejściowe: N - 1000, SQPE - [0-25]%*N, K = 10

Na wykresach przedstawiono wpływ procentowej zawartości błędów pozytywnych w widmie oligonukleotydów na czas działania algorytmów ACO i Greedy.

Algorytm Greedy charakteryzuje się dużą stabilnością — jego czas działania delikatnie rośnie wraz z większą zawartością błędów. Jest to normalne, bo w takich sytuacjach mamy więcej błędnych ścieżek, a algorytm musi robić więcej “rollback`ów”.

Czas działania algorytmu ACO (heurystycznego) jest znacznie bardziej zmienny i wykazuje wrażliwość na obecność zakłóceń. Z reguły, im większa zawartość błędów tym większy wzrost czasu (względem alg. Greedy).

Mimo to, ACO potrafi być wyraźnie szybszy od Greedy — zwłaszcza przy niskim poziomie błędów, a czasami także przy wysokim, jeśli heurystyka szybko natrafi na dobrą trajektorię. Niemniej jednak ta szybkość okupiona jest mniejszą przewidywalnością.

Podsumowanie wyników

ACO może oferować lepszą wydajność, ale jego niestabilność czyni go bardziej ryzykownym wyborem w obecności błędów pozytywnych, podczas gdy Greedy zapewnia stałą i niezawodną efektywność.