Główne sprawozdanie - problem SBH

Kierunek Bioinformatyka stopień 1, semestr 3	**data** 23 stycznia 2024 r.
Przedmiot Optymalizacja kombinatoryczna - laboratoria	**projekt SBH**
Wykonawcy Hubert Łapka, Emilia Bacik	**grupa** 2

W poniższym sprawozdaniu znajdują się opisy zaimplementowanego generatora instancji, algorytmu losowego, poszukującego jakiegokolwiek rozwiązania problemu i metaheurystyki opartej na algorytmie mrówkowym. W dalszej części został przedstawiony zarys przeprowadzonych testów algorytmu wraz z wynikami oraz ich wizualizacją.

Format sprawozdania

1) **Charakterystyka zaimplementowanych modułów**
    - generator instancji
    - algorytm poszukujący jakiegokolwiek rozwiązania (“losowy”)
    - metaheurystyka - algorytm mrówkowy
    - złożoność obliczeniowa
2) **Testy zaimplementowanego programu**
    - zależność między parametrem algorytmu “losowego” a jakością rozwiązania
    - wpływ błędów pozytywnych na rozwiązanie
    - wpływ błędów negatywnych na rozwiązanie
    - oddziaływanie długości sekwencji a wielkość k-meru
    - zależność między dwoma wybranymi parametrami metaheurystyki a jakością rozwiązania
    - porównanie wyników algorytmu losowego i metaheurystyki
3) **Konkluzje i uwagi końcowe**
    - wnioski końcowe
    - procedura uruchomienia programu
    - załączniki
4) **Źródła i odniesienia**

Wstęp

Rozważnym w ramach tego projektu był klasyczny model problemu sekwencjonowania przez hybrydyzację. Informacje z jakich mogliśmy korzystać w programie były następujące:

- długość sekwencji DNA
- wielkość oligonukleotydów
- informacje jakie błędy są w spektrum
- początkowy oligonukleotyd

Główne sprawozdanie - problem SBH

Spektrum po utworzeniu było poddawane sortowaniu alfabetycznym w celu zasymulowania rzeczywistych warunków przeprowadzanego eksperymentu hybrydyzacyjnego. Jedyną informacją jaka pozostawała po takim zabiegu jest sekwencja pierwszego k-meru, odpowiadająca startowemu wierzchołkowi.

Charakterystyka zaimplementowanych modułów

Zmienne globalne (znane dla algorytmu): długość sekwencji (int n), wielkość k-mera (int k), ilość błędów pozytywnych i negatywnych (int, int)

Moduł 1 - generator instancji
Wpierw zostaje wylosowana **sekwencja** o długości n, która ma imitować oryginalną nić DNA. Przechowywaną w wektorze sekwencję zapisujemy do pliku, aby móc odwołać się do niej w czasie sprawdzania rozwiązania, jakie znajdzie algorytm.
Po wygenerowaniu sekwencji zostaje zapamiętany **pierwszy oligonukleotyd**, czyli ciąg pierwszych k znaków w wektorze. (zgodnie z założeniami problemu)

W celu wygenerowania **spektrum** składającego się ze zbioru oligonukleotydów o długości k, przystępujemy kolejno do :

- utworzenia bezbłędnego spektrum o wielkości n-k+1 podczas, którego sprawdzane jest czy żaden k-mer się nie powtarza (powtórzenie powoduje zwiększenie o 1 wartości zmiennej negative_errors)
- dodanie **błędów negatywnych** odbywa się przez losowy wybór elementu ze spektrum(oprócz pierwszego) i usunięcie jego zawartości oraz przesunięcie pozostałych elementów
- dodanie **błędów pozytywnych** odbywa się przez generowanie oligonukleotydu podobnego do losowego już istniejącego w spektrum i wstawianie go do spektrum tylko jeżeli nie występuje już w nim, różnica w stosunku do oryginalnego oligonukleotydu może z pewnym prawdopodobieństwem pojawić się na każdym nukleotydzie (20%)

Za pomocą funkcji *Quick_sort* i przeciążonych operatorów “<” i “>” (w celu ustalenia porządku między oligonukleotydami) elementy spektrum zostają posortowane alfabetycznie, co powoduje utratę informacji o ich początkowej kolejności.

Moduł pośredni - budowanie grafu
W celu budowy grafu algorytm inicjalizuje macierz sąsiedztwa *graph* na podstawie naszego spektrum (początkowo wypełnioną zerami). Każdej komórce w macierzy zostanie przypisana waga krawędzi między oligonukleotydami lub jej brak (zero). Wagi te są określane przez funkcję *Imposition*, która przypisuje konkretną wartość połączenia, analizując nakładanie się k-merów w trzech rodzajach przesunięcia. Przesunięcia k-merów sprawdzane są trzema warunkami *if..else,* zwracając odpowiednio wartości 1,2 lub 3, jeśli któreś z przesunięć istnieje. Mniejsze nałożenia (4 nukleotydy itp) nie są uwzględniane.

Moduł 2 - algorytm losowy
Zwraca wektor *path* zawierający numery kolejnych oligonukleotydów budujących rozwiązanie.

Główne sprawozdanie - problem SBH

Na początku utworzony zostaje wektor - tablica, zawierający dla każdego oligonukleotydu informację o tym, czy został już odwiedzony. Następnie wywołana zostanie funkcja *Find_next()*, która przyjmując za argument zapisany oligonukleotyd, będący pierwszym oligonukleotydem w sekwencji, zapisuje go w rozwiązaniu, a następnie przeszukuje jego następniki i wybiera taki, dla którego łuk między nimi ma najmniejszą wagę. Jeżeli jest kilka następników z połączeniem o tej samej wadze do pierwszego oligonukleotydu (wierzchołka w grafie graph) następnik do rozwiązania zostaje wybrany losowo. Funkcja jest wykonywana rekurencyjnie z wybranym wierzchołkiem (oligonukleotydem) jako kolejnym argumentem tak długo, aż długość wektora *path* nie będzie równa (lub większa) oczekiwanej długości rozwiązania (ilość wierzchołków w grafie - liczba błędów pozytywnych lub dopóki algorytm nie zakleszczy się (nie znajdzie się w miejscu, w którym nie widzi żadnego dostępnego następnika). Funkcja *Find_next* nigdy nie wybiera wierzchołka, który raz został już dodany do rozwiązania (odwiedzony).
Funkcję *Find_next* wspiera dodatkowo funkcja *Suboptimum_path()*, która uruchomi się gdy algorytm wybierze krawędzie inne niż te o wadze 1 tyle razy pod rząd, ile wskazuje parametr związany z tą funkcją - *parametr_not_chosen*. Funkcja ta pomaga chronić algorytm przed zbyt wczesnym zakleszczeniem. Funkcja *Suboptimum_path* losuje jeden z niewybranych do tej pory wierzchołków, a następnie posiłkując się implementacją algorytmu Dijkstry znajduje wśród tych wierzchołków najkrótszą ścieżkę do niego i dodaje ją do rozwiązania, wywołując następnie znów funkcję *Find_next* z ostatnim oligonukleotydem ze znalezionej najkrótszej ścieżki (czyli wylosowanym na początku) jako argumentem.

Algorytm polega w wielu miejscach na czynnikach losowych, dlatego jest w stanie uzyskać dobre rozwiązanie jedynie dla instancji o niewielkiej zawartości błędów (negatywnych i pozytywnych). Wygenerowane przez niego rozwiązanie ma największy współczynnik prawdopodobnego odpowiadania rzeczywistemu na pierwszych wybranych wierzchołkach (na początku wynikowej sekwencji). Algorytm często zwraca zbyt krótkie rozwiązania z powodu zbyt wczesnego zakleszczenia (zwłaszcza przy wysokich wartościach parametru *parametr_not_chosen*).

Moduł 3 - wybrana metaheurystyka: algorytm mrówkowy
Ogólny opis algorytmu:
Algorytm opiera się na algorytmie losowym i był implementowany na jego podstawie, jednak jest dużo bardziej złożony. Na początku działania algorytmu tworzona jest macierz feromonowa o wielkości odpowiadającej wszystkim wierzchołkom grafu (n*n), wypełniona zerami. Następnie pętla odpowiadająca ilości mrówek (*parametr_number_of_ants*) uruchamia wielokrotnie algorytm losowy zapisując w ten sposób wiele początkowych (zwykle słabych) ścieżek. Uruchomiona zostaje funkcja *Rate_solution*, która z wszystkich zapisanych ścieżek wybiera 10 ‘najlepszych’ i odkłada w macierzy feromonowej na łukach, które zostały użyte w tych ścieżkach, następująco wartości: 1 dla łuków z najlepszego rozwiązania, 0,9 dla drugiego najlepszego, 0,8 dla trzeciego i tak do dziesiątego rozwiązania. Na potrzeby naszego projektu przyjęliśmy następujący schemat oceniania rozwiązań:

- podstawowa ilość punktów to podwojenie ilości odwiedzonych wierzchołków (odpowiada stopniowi pokrycia grafu),
- następnie funkcja dodaje odpowiednio +1 punkt, -2 punkty i -10 punktów za każdy łuk w rozwiązaniu z wagą o wartości 1, 3 lub ponad 3 (choć domyślnie nie wybierane, w

pechowym scenariuszu możliwe do wybrania np podczas poszukiwania sub optymalnej ścieżki). Łuki o wadze 2 nie wpływają na przyznane punkty.

- funkcja porównuje ilość wybranych łuków o wadze 2 (sugeruje błąd negatywny), wadze 3 (sugeruje dwa błędy negatywne) i większej z ilością oczekiwanych błędów negatywnych. Jeśli wartości te są równe, ilość punktów jest przemnażona przez wartość 1,5. Jeśli różnica między oczekiwanymi i znalezionymi błędami jest mniejsza lub równa 10 przyznane punkty zostaną przemnożone przez 0,2; w przeciwnym wypadku podzielone przez 0,3.
- funkcja porównuje ilość nie odwiedzonych wierzchołków w grafie z liczbą oczekiwanych błędów pozytywnych i przemnaża ilość punktów w taki sam sposób jak w punkcie powyżej.

Powyższego schematu w żadnym stopniu nie uważamy za najlepszy możliwy schemat i nie został on przetestowany w sposób pozwalający wysnuć istotne wnioski co do jego wpływu na jakość rozwiązań. Składa się z wielu parametrów, określających przełożenie konkretnych czynników na punktację i każdy z nich musiałby zostać wystrojony, żeby otrzymać najlepsze wyniki algorytmu. Z uwagi na ograniczony czas przygotowania całej metaheurystyki oraz edukacyjny charakter projektu nie zdecydowaliśmy się na takie strojenie, skupiając na innych ważnych z punktu widzenia metaheurystyki parametrach i uznając wyniki funkcji *Rate_solution* za zadowalające na etapie pisania algorytmu i sprawdzania działania funkcji dla różnych instancji.

Po każdym odłożeniu wartości na macierz feromonową znalezione przez mrówki ścieżki zostają zapomniane w celu optymalizacji używanej przez program pamięci.

Po pierwszej iteracji algorytmu, w której mrówki poruszają się całkowicie losowo i odłożeniu na macierzy feromonowej pierwszych wartości rozpoczyna się właściwa część algorytmu mrówkowego, która składa się z powtarzalnych iteracji znajdowania przez mrówki rozwiązań, oceniania ich i odkładania wartości na macierz feromonową, a także z funkcji pomocniczych. Algorytm mrówkowy działa od tego momentu przez czas równy w zaokrągleniu *parametr_of_time*, jednak sprawdzenie warunku upłynięcia czasu występuje jedynie po pełnej iteracji, tak więc wartości te nie będą zwykle identyczne. Pełna iteracja algorytmu mrówkowego zaczyna się puszczeniem mrówek po grafie w poszukiwaniu możliwych rozwiązań. *Parametr_of_nonconformity* (wartość od 1 do 100, w testach przyjęliśmy 80) mówi o tym jaki jest początkowy procent szans, że mrówka w danym wierzchołku będzie chciała iść losowo wybranym łukiem, a nie sugerować się macierzą feromonów. Losowy wybór łuku odbywa się w sposób identyczny jak w algorytmie losowym. Jeśli algorytm wylosuje, że w danym wierzchołku mrówka pójdzie zgodnie z feromonami, specjalna funkcja *Find_number_of_probability* wybiera kolejny łuk z łuków o wartościach feromonów większych od 0 w taki sposób, żeby prawdopodobieństwo wybrania łuku z odłożoną na macierzy większą wartością było odpowiednio większe. Przyjęliśmy rozkład, w którym każda kolejna wartość zmiennej losowej ma przypisaną dwa razy większą wartość prawdopodobieństwa niż poprzednia, bez względu na rzeczywistą różnicę w wartościach na macierzy feromonowej, jednak w dalszym rozwoju algorytmu warto byłoby rozważyć sparametryzowanie tej zależności i przetestowanie różnych rozkładów prawdopodobieństwa. Algorytm mrówkowy również korzysta z funkcji znajdującej sub optymalną ścieżkę (opartej na algorytmie Dijkstry). Po znalezieniu ścieżek przez wszystkie mrówki są one oceniane i najlepsze 10 odkłada znów wartości na macierz feromonową. Po każdej iteracji *parametr_of_nonconformity* zmniejszany jest o 5 punktów procentowych, co skutkuje chętniejszym sugerowaniem się macierzą feromonową przez mrówki.

Główne sprawozdanie - problem SBH

Po każdej iteracji uruchamiają się również dwie funkcje pomocnicze - *Pheromone_pairing* - powodująca zmniejszenie wszystkich wartości w macierzy o 0,1 ich wartości; oraz *Smoothing_values* znajdująca średnią wartości w macierzy i zmniejszająca (zwiększająca) wartości od niej większe (mniejsze) według następujących progów: brak zmiany dla wartości równej średniej, 0,05 aktualnej wartości przy różnicy mniejszej niż 5; 0,1 dla różnicy mniejszej niż 10, 0,2 -20, 0,3 - 30, 0,4 - 40, 0,5 - 50, 0,7 - 80 i 0,9 w pozostałych przypadkach.
Gdy po pełnej iteracji algorytm sprawdzi, że minął czas, który został przeznaczony na jego działanie (*parametr_of_time*) zwróci najlepiej ocenioną ze ścieżek, które znalazł podczas ostatniej iteracji.

Złożoność obliczeniowa
Próba znalezienia dokładnego rozwiązania problemu sekwencjonowania przez hybrydyzację szybko skończyłaby się algorytmem wykładniczym, o ile stworzenie takiego przy instancji z błędami byłoby w ogóle możliwe. Algorytm losowy wybiera kolejny wierzchołek losowo, zamiast sprawdzania wszystkich możliwych opcji właśnie w celu optymalizacji złożoności obliczeniowej - choć algorytm nie zwraca zbyt często dobrych jakościowo rozwiązań, jest algorytmem wielomianowym, zależnym jedynie od ilości wierzchołków w grafie (która zależy bezpośrednio od długości oligonukleotydów i długości oryginalnej sekwencji, ewentualnie również od błędów). To daje podstawę do zbudowania pseudowielomianowej heurystyki, która wciąż wykonując się w dużo szybszym czasie niż algorytm dokładny (Brute Force), zwróci zadowalające rozwiązania, dużo bliższe do oryginalnej sekwencji. Złożoność obliczeniowa algorytmu mrówkowego, który zaimplementowaliśmy w ramach tego projektu zależna jest przede wszystkim od dwóch parametrów tego algorytmu - ilości mrówek (paramet_number_of_ants) i czasu po jakim przestanie się wykonywać główna pętla algorytmu (parametr_of_time); oraz tak jak algorytm losowy (na którym metaheurystyka jest oparta) - od ilości wierzchołków w grafie.

Testy zaimplementowanego programu

W tym akapicie zostały dogłębnie opisane przeprowadzone przez nas testy wpływu własności instancji oraz różnych parametrów algorytmu na zwracane rozwiązanie. Każdy punkt pomiarowy na wykresie odpowiada średniej arytmetycznej przynajmniej kilkunastu pomiarów. Wnioski oraz spostrzeżenia dotyczące danego testu znajdują się pod tabelami bądź wykresami. W celu oceny podobieństwa między sekwencją wynikową, a oryginalną została użyta miara odległości między dwoma łańcuchami znaków, a konkretnie miara Levenshteina.

Główne sprawozdanie - problem SBH

Część 1 - parametry algorytmu a dokładność zwracanego rozwiązania

Zależność między parametrem algorytmu “losowego” a dokładnością

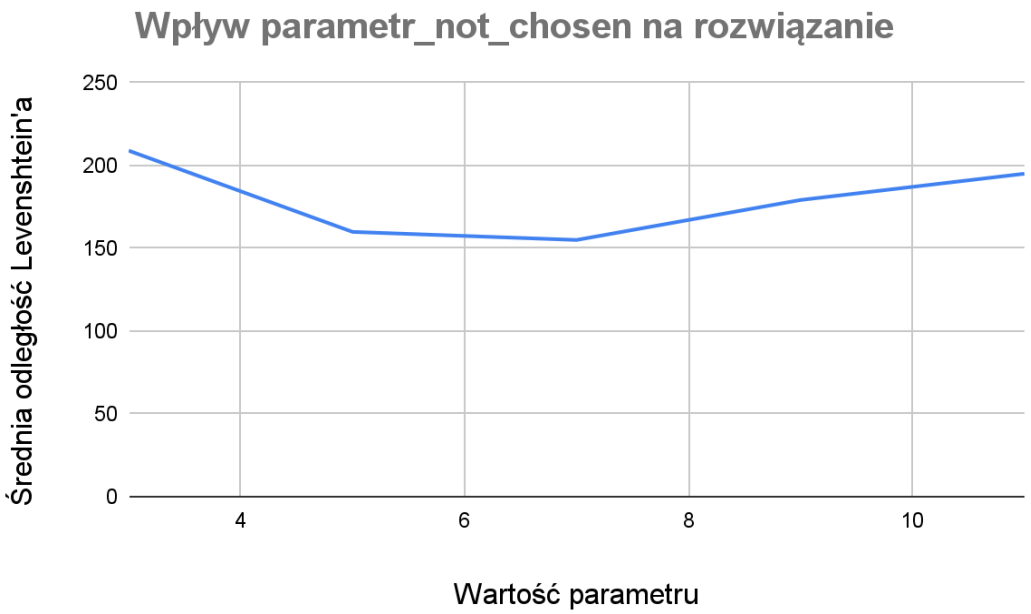
Zbiór możliwych rozwiązań zwracanych przez algorytm jest zależny od parametrów przyjętych w procesie poszukiwania sekwencji wyjściowej. Nasz algorytm posiada zmienną parametr_not_chosen, która odpowiada progowi gorszych przejść w grafie (tzn. różnych od nałożenia o wartości 1) pod rząd, po których nastąpi przeskok zgodnie z zasadą działania algorytmu (opisana w akapicie - Charakterystyka zaimplementowanych modułów) . W tym teście skupiliśmy się nad znalezieniem możliwie najlepszej wartości parametru odpowiadającej najdokładniejszym wynikom, zwracanych dla zróżnicowanych instancji. Do procesu testowania zostało przygotowanych 15 sekwencji o długości n = 500, a na ich podstawie odpowiednio 15 spektrum z odpowiednią obecnością błędów (podane w tabeli) i jednakową wielkością oligonukleotydów k = 8. Badane były ustawienia parametru na kolejno wartość 3, 5, 7, 9, 11.

Wyniki zostały przedstawione w poniższej tabeli:

n = 500, k = 8	Wpływ parametr_not_chosen na rozwiązanie					
Liczba błędów w instancji	numer sekwencji	wartość 3	wartość 5	wartość 7	wartość 9	wartość 11
neg = 6, pos = 5	1	140	113	135	147	162
neg = 8, pos = 5	2	215	159	145	162	150
neg = 1, pos = 10	3	160	102	34	86	137
neg = 2, pos = 10	4	216	209	219	244	194
neg = 17, pos = 3	5	203	206	230	239	241
neg = 15, pos = 3	6	190	113	126	188	126
neg = 10, pos = 10	7	209	110	72	99	183
neg = 20, pos = 20	8	255	105	75	185	221
neg = 13, pos = 0	9	250	123	172	122	237
neg = 36, pos = 0	10	233	223	233	229	243
neg = 0, pos = 15	11	207	217	225	233	231
neg = 0, pos = 35	12	212	140	110	206	204
neg = 14, pos = 10	13	244	246	238	210	227
neg = 10, pos = 14	14	181	153	143	171	190
neg = 15 , pos = 20	15	219	179	168	166	179
średnia:		208,93	159,87	155,00	179,13	195,00
odchylenie standardowe:		31,78	49,73	65,19	49,92	38,66

Główne sprawozdanie - problem SBH

Wizualizacja na wykresie:



Zauważyliśmy bardzo zbliżone średnie jakości rozwiązania zwracane dla wartości 5 i 7. O przypuszczalne optimum lokalne jest więc podejrzana wartość pośrednia. Postanowiliśmy rozszerzyć nasz test o zbadanie zgodności rozwiązań zwracanych dla tych samych instancji przy wartości parametr_not_chosen równej 6.

Fragment uzupełnionej tabeli (dla wartości podejrzanych o optimum):

n = 500, k = 8	Wpływ parametr_not_chosen na rozwiązanie			
Liczba błędów w instancji	numer sekwencji	wartość 5	wartość 6	wartość 7
neg = 6, pos = 5	1	113	130	135
neg = 8, pos = 5	2	159	154	145
neg = 1, pos = 10	3	102	121	34
neg = 2, pos = 10	4	209	226	219
neg = 17, pos = 3	5	206	213	230
neg = 15, pos = 3	6	113	98	126
neg = 10, pos = 10	7	110	98	72
neg = 20, pos = 20	8	105	101	75
neg = 13, pos = 0	9	123	120	172
neg = 36, pos = 0	10	223	226	233
neg = 0, pos = 15	11	217	230	225
neg = 0, pos = 35	12	140	117	110
neg = 14, pos = 10	13	246	221	238

Główne sprawozdanie - problem SBH

neg = 10, pos = 14	14	153	151	143
neg = 15 , pos = 20	15	179	180	168
średnia:		159,87	159,07	155,00
odchylenie standardowe:		49,73	51,88	65,19

Wnioski i spostrzeżenia:
Krzywa powstała na podstawie średniej arytmetycznej otrzymanych dokładności ilustruje powstałe optimum lokalne dla wartości parametru ustawionej na 7. Pomimo iż wyniki eksperymentu wskazują wyższą dokładność rozwiązań przy tej wartości, odchylenie standardowe sugeruje dużą rozbieżność w otrzymanych wynikach. Rozbieżność wynika oczywiście ze zróżnicowanej struktury instancji, jednak dla wartości równej 7 jest ona największa. Analizują poszczególne instancje zauważamy, że zmiana parametru nie wpływała znacząco na zwracane rozwiązania dla niektórych sekwencji.
Między innymi sekwencja nr 10 jest przykładem instancji, dla której wartość ustawionego progu nie ma dużego znaczenia, gdyż zwracane wyniki mają podobną jakość.
Podsumowując wpływ parametru jest znaczący w kontekście dokładności otrzymanych wyników, jednak nie dla każdego przypadku. Sugeruje to iż powinniśmy kierować się charakterystyką danej instancji przy wyborze wartości parametru aby uzyskać możliwie najlepsze wyniki, jednak gdy takich informacji nie posiadamy próg o wartości 7 będzie bezpiecznym wyborem.

Część 2 - własności instancji a dokładność zwracanego rozwiązania

Wpływ błędów pozytywnych na rozwiązanie

Na potrzeby tego testu zostało wygenerowanych 15 sekwencji DNA (oznaczonych kolejno w tabeli) o jednakowej długości n = 500 nukleotydów. Celem było przeprowadzenia badania określającego wpływ obecności danego procenta błędów pozytywnych w spektrum na jakość rozwiązania, zwracanego przez algorytm “losowy”. Badana była obecność 2, 4, 6, 8 i 10% błędów pozytywnych w spektrum. Na podstawie wygenerowanych sekwencji zostały przygotowane odpowiednie spektra o jednakowej wielkości k-merów, równej 8 wraz z odpowiednią zawartością błędów pozytywnych. Wszelkie inne parametry algorytmu pozostawały stałe w procesie testowania, odpowiednio:

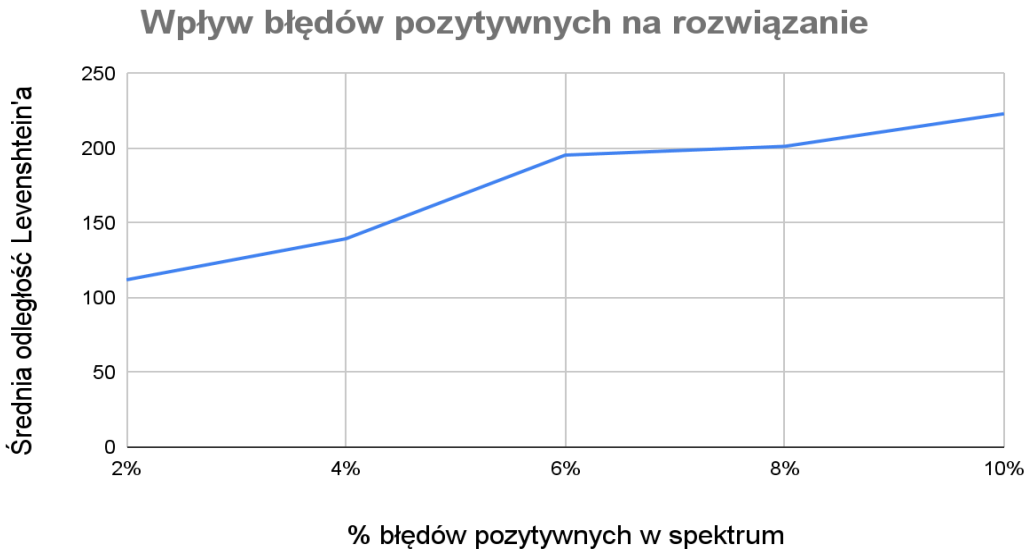
- długość sekwencji n = 500
- wielkość oligonukleotydów k = 8
- parametr algorytmu “losowego” parametr_not_chosen = 7 (wybrana w poprzednim teście)
- ilość błędów negatywnych negative_errors = 0

Główne sprawozdanie - problem SBH

Wyniki zostały przedstawione w poniższej tabeli:

n =500, k=8	Miara Levenshtein'a dla x% błędów pozytywnych				
numer sekwencji	2% spektrum	4% spektrum	6% spektrum	8% spektrum	10% spektrum
1	180	186	223	192	244
2	109	135	102	171	227
3	126	59	188	193	212
4	63	71	223	251	243
5	155	171	202	225	201
6	85	161	190	175	197
7	177	215	209	207	207
8	185	181	242	215	211
9	105	67	198	209	217
10	66	70	119	211	259
11	31	122	184	151	208
12	130	207	227	228	241
13	67	138	180	172	246
14	71	190	234	216	244
15	130	119	211	204	190
średnia:	112,00	139,47	195,47	201,33	223,13
odchylenie standardowe:	48,14	53,55	39,35	26,04	21,51

Wizualizacja na wykresie:



Główne sprawozdanie - problem SBH

Wnioski i spostrzeżenia:
Wyznaczona krzywa rosnąca jasno potwierdza naszą tezę o pogorszeniu rozwiązań wraz ze wzrostem obecności błędów w spektrum. Należałoby jednak przyjrzeć się przedziałom wzrostu miary Levenshtein'a i tego jakie wartości składają się średnią. Większe odchylenie standardowe dla własności 2% i 4% sugeruje nam iż taka rozbieżność mogła być spowodowana poziomem skomplikowania sekwencji (powtarzające się dłuższe ciągi tych samych nukleotydów), co powodowało niewielki zbiór możliwych ścieżek. Pewne odstępstwa od tendencji wzrostowej miary Levenshtein'a (wykazane m.in przez sekwencje nr 9 i 15) obrazują losowość algorytmu, który dla pewnych instancji może ulec zakleszczeniu i zwrócić krótką sekwencję wynikową. Dodawanie kolejnych błędów nie wpłynęło znacznie na pogorszenie rozwiązań dla sekwencji nr 7 może to być spowodowane strukturą ciągu nukleotydów, dla których algorytm przeszukiwał zawsze podobną ścieżkę, a dodawane błędy pozytywne nieznacznie na nią wpływały.

Wpływ błędów negatywnych na rozwiązanie
Do tego testu zostało użytych dokładnie 15 tych samych sekwencji DNA o długości n = 500 (oznaczonych kolejno w tabeli), co w poprzednim teście. Zabieg ten pozwala nam na porównanie ze sobą wpływu obu rodzajów błędów. Podczas tego testu analogicznie jak w powyższym badana była obecność 2, 4, 6, 8 i 10% błędów negatywnych w spektrum. Na podstawie sekwencji zostały przygotowane właściwe spektra o jednakowej wielkości k-merów, równej k = 8 wraz z odpowiednią zawartością błędów negatywnych. Wszelkie inne parametry algorytmu pozostawały stałe w procesie testowania, odpowiednio:

- długość sekwencji n = 500
- wielkość oligonukleotydów k = 8
- parametr algorytmu “losowego” parametr_not_chosen = 7
- ilość błędów pozytywnych positive_errors = 0

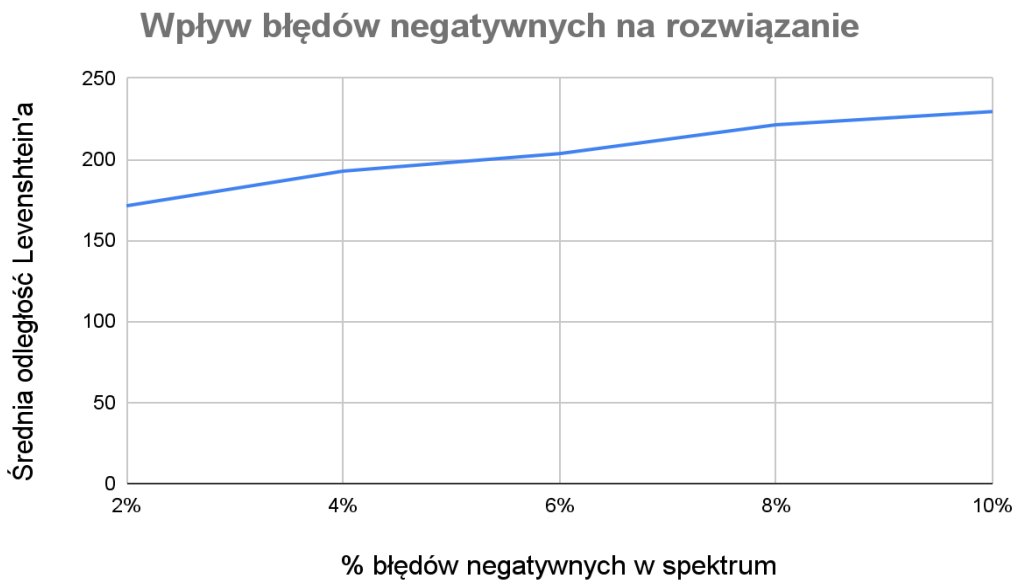
Wyniki zostały przedstawione w poniższej tabeli:

n =500, k=8	Miara Levenshtein'a dla x% błędów negatywnych				
numer sekwencji	2% spektrum	4% spektrum	6% spektrum	8% spektrum	10% spektrum
1	255	228	226	244	243
2	124	103	163	226	199
3	135	200	171	180	185
4	217	245	258	255	250
5	190	201	215	229	222
6	138	144	163	216	225

Główne sprawozdanie - problem SBH

7	203	216	210	235	246
8	202	211	209	223	229
9	124	112	186	175	216
10	125	164	226	229	238
11	200	196	209	217	233
12	187	239	240	255	258
13	192	235	209	213	231
14	123	213	203	218	225
15	155	184	165	205	242
średnia:	171,33	192,73	203,53	221,33	229,47
odchylenie standardowe:	41,86	44,05	28,92	22,93	19,12

Wizualizacja na wykresie:



Wnioski i spostrzeżenia:

Błędy negatywne usuwające z reprezentacji grafowej istotne elementy ścieżki okazały się dla naszego algorytmu bardzo kłopotliwe. Wartości miary Levenshtein'a, czyli tym samym rozbieżności naszego rozwiązania utrzymują się dla wszystkich pomiarów na wysokim poziomie. Odchylenie standardowe ponownie okazało się maleć wraz ze wzrostem utraconych oligonukleotydów, sugerując iż w niektórych spektrach mógł być utracony ważny element ścieżki. Strata coraz większej ilości k-merów wyrównywała strukturę użytych instancji.

Główne sprawozdanie - problem SBH

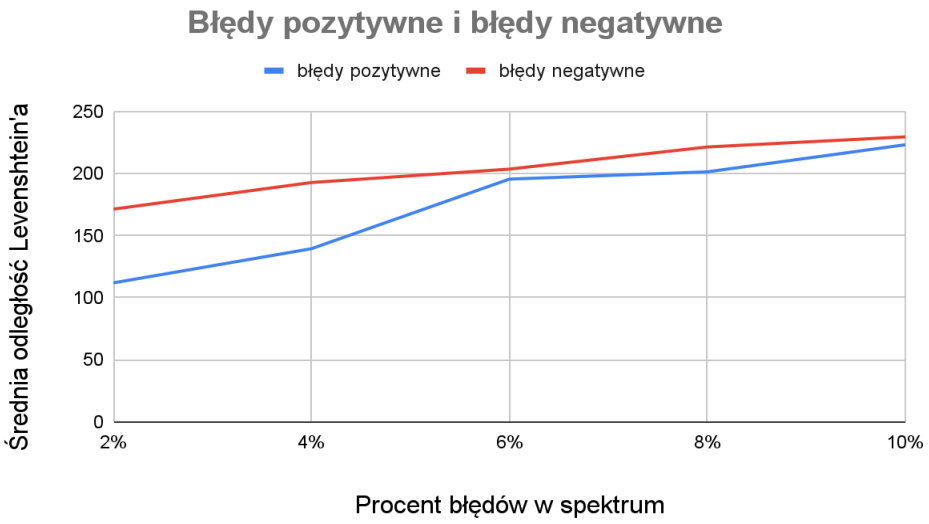
Porównanie wpływy obu rodzajów błędów na jakość rozwiązania

Rozważmy jak mają się do siebie utrata pewnych oligonukleotydów w spektrum a zwiększona liczba możliwych wyborów podczas poszukiwania ścieżki dla naszego algorytmu.

Tabela porównawcza:

n = 500, k = 8	Średnia odległość Levenshtein'a	
procent błędów	błędy pozytywne	błędy negatywne
2%	112	171,33
4%	139,47	192,73
6%	195,47	203,53
8%	201,33	221,33
10%	223,13	229,47

Wizualizacja na wykresie



Wnioski i spostrzeżenia:

Konkluzją płynącą z przeprowadzonych testów jest zasadniczy wpływ błędów hybrydyzacyjnych w instancji na jakość zwracanych rozwiązań. Błędy pozytywne okazały się mniej kłopotliwe niż błędy negatywne. Utrata oligonukleotydów skutkowała znacznym pogorszeniem jakości rozwiązań. Obecność 10% błędów w spektrum w obu przypadkach nieznacznie różniła się średnią odległością Levenshtein'a między sekwencją wynikową a oryginalną.

Oddziaływanie długości sekwencji a wielkość k-meru

Wskazania prowadzącego oraz literatura prowadzą nas do wniosku, że dobrana wielkość oligonukleotydów jest ściśle związana z długością sekwencji. Ma to na celu stworzenie

Główne sprawozdanie - problem SBH

odpowiedniego do testów spektrum, jednak strona biochemiczna przeprowadzanego badania hybrydyzacyjnego jest od tego ściśle uzależniona (wielkość macierzy). Naszym celem w tym teście było określenie długości sekwencji wejściowej, dla której k-mery o wielkości 8 przestają być wydajne.

Do testu zostały przygotowane instancje o długości sekwencji 500, 600 oraz 700 nukleotydów. Przygotowane na ich podstawie spektrum zawierało stosunkowo niewielką zawartość błędów (opisana w tabeli), gdyż głównym czynnikiem stanowiącym trudność w znalezieniu rozwiązania miała być zwiększona długość sekwencji. Procesie testowania został poddany algorytm losowy z parametrami:

- liczba błędów hybrydyzacyjnych (zgodnie z tabelą)
- wielkość oligonukleotydów k = 8
- parametr algorytmu “losowego” parametr_not_chosen = 7

W związku z tym, że badane sekwencje mają różną długość zastosowaliśmy w ich porównaniu jako miarę zgodności zależność opartą na średniej odległości Leneshtein’a daną wzorem:

$$\textit{zgodność sekwencji} = \frac{\textit{długość sekwencji} - \textit{miara Levenshteina}}{\textit{długość sekwencji}} * 100\%$$

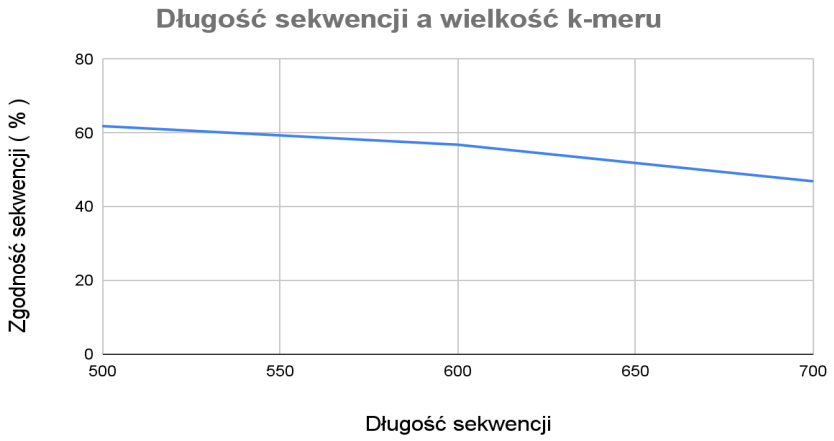
Zwracana procentowa zgodność rozwiązania ma umożliwić wyciągnięcie wniosków, gdyż sama miara odległości między dwoma łańcuchami znaków jest ściśle związana z ich długością.

Poniższa tabela zawiera wyniki testu:

k = 8	Długość sekwencji a wielkość k-meru			
% błędów w spektrum	numer sekwencji	n = 500	n = 600	n = 700
neg = 4%, pos = 3%	1	240	263	401
neg = 2%, pos = 2%	2	217	258	305
neg = 1%, pos = 2%	3	171	290	371
neg = 2%, pos = 1%	4	133	201	368
neg = 2%, pos = 2%	5	168	299	346
neg = 2%, pos = 2%	6	193	283	444
neg = 3%, pos = 2%	7	165	248	358
neg = 3%, pos = 3%	8	246	294	373
neg = 3%, pos = 3%	9	191	253	405
neg = 1%, pos = 3%	10	182	201	345
średnia:		190,60	259,00	371,60
zgodność sekwencji		61,88%	56,83%	46,91%
odchylenie standardowe:		35,20	35,38	38,33

Główne sprawozdanie - problem SBH

Wizualizacja na wykresie:



Wnioski i spostrzeżenia:
Przyjęliśmy sekwencję o długości n = 500 jako wzorcową dla oligonukleotydów o wielkości k = 8. Chęć by w przyszłości poddać sekwencjonowaniu dłuższe fragmenty przy użyciu tej samej mikromacierzy (zawierającej k-mery o długości 8) przez nasz algorytm musimy zaniechać. Wyniki dla sekwencji o długości 600 nie są już zadowalające ale wciąż utrzymują chociaż 50% zgodność, jednak dla długości 700 ta granica akceptowalności została przekroczona. Należy przyjąć iż nasz algorytm przy k-merach rzędu wielkości 8 poradzi sobie z co najwyżej sekwencją o długości 550 - 600 oligonukleotydów.

Część 3 - metaheurystyka (algorytm mrówkowy)

Algorytm mrówkowy zawiera w sobie mnóstwo parametrów. Postanowiliśmy wystroić i przeprowadzić testy różnych wartości i ich wpływu na rozwiązanie dla dwóch z nich: liczby mrówek (parametr_number_of_ants) oraz czasu działania głównej części metaheurystyki (parametr_of_time).

Wpływ parametru parametr_of_time (czas działania głównej części programu) na jakość rozwiązania
Przetestowaliśmy 6 różnych wartości (10, 15, 20, 25, 30, 35) w sekundach parametru na 15 różnych instancjach o różnych ilościach błędów, różnej długości całej sekwencji i oligonukleotydów, na których pracuje program. W testach przyjęliśmy wartość parametru parametr_number_of_ants równą 100, parametr_not_chosen=7 oraz parametr_of_nonconformity=80.
Poniższa tabela zawiera wyniki testu:

Główne sprawozdanie - problem SBH

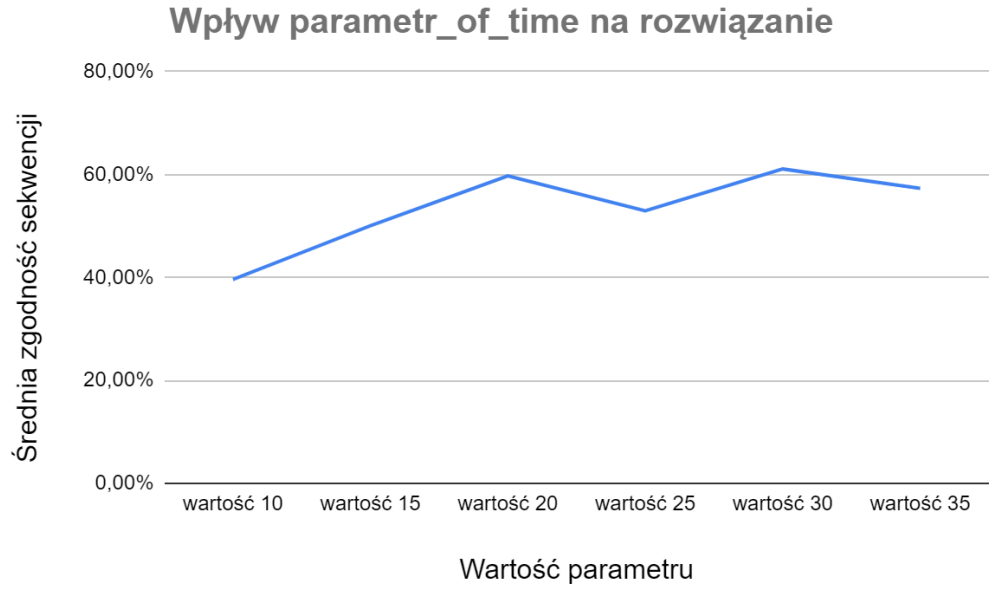
Wpływ czasu działania algorytmu mrówkowego przy liczbie mrówek równej 100								
długość sekwencji i k-meru	błędy	numer sekwencji	wartość 10	wartość 15	wartość 20	wartość 25	wartość 30	wartość 35
n=300, k=7	pos=10, neg=5	1	116	155	144	135	100	100
n=300, k=8	pos=10, neg=8	2	238	168	179	183	195	238
n=300, k=10	pos=20, neg=15	3	190	0	190	190	0	0
n=300, k=10	pos=50, neg=30	4	241	241	0	241	0	0
n=500, k=8	pos=15, neg=10	5	276	151	151	276	151	276
n=500, k=10	pos=0, neg=20	6	380	380	380	380	380	380
n=500, k=10	pos=80, neg=50	7	305	154	154	305	304	154
n=500, k=8	pos=20, neg=3	8	389	389	85	49	242	389
n=500, k=10	pos=70, neg=0	9	337	270	268	270	366	366
n=500, k=9	pos=20, neg=15	10	436	436	436	0	0	0
n=500, k=10	pos=30, neg=20	11	0	0	0	0	0	0
n=550, k=10	pos=120, neg=60	12	414	414	69	414	414	414
n=400, k=10	pos=0, neg=60	13	100	100	100	100	100	100
n=400, k=7	pos=20, neg=10	14	193	169	192	149	177	192
n=400, k=9	pos=100, neg=60	15	284	250	209	250	209	264
średnia:			259,93	218,47	170,47	196,13	175,87	191,53
odchylenie standardowe:			124,98	139,41	122,00	126,45	144,58	154,43
średnia procentów zgodności:			39,61%	50,07%	59,76%	52,95%	61,09%	57,34%

Z powodu różnic w długościach sekwencji postanowiliśmy wprowadzić dodatkową miarę - zgodność sekwencji, zdefiniowaną przez nas już wcześniej jako

Główne sprawozdanie - problem SBH

$\frac{długość\ \ sekwencji - miara\ Levenshteina}{długość\ sekwencji} * 100\%$. Miara ta miała pomóc nam lepiej porównać sekwencje, ponieważ miara Levenshteina jest uzależniona od długości sekwencji (daje większy wynik dla dłuższej sekwencji, choć procentowo sekwencja wyjściowa może być tak samo podobna do oryginalnej).

Wizualizacja na wykresie:



Wnioski i spostrzeżenia:
Wykres nie jest liniowy, co sugeruje, że istotność statystyczna przeprowadzonego przez nas testu nie jest zbyt duża. Najlepsze wyniki algorytm przyjmuje dla wartości 30, a następnie wartości 20, dziwi więc minimum lokalne przy wartości 25. Może to być spowodowane wartościami instancji 4, 5, 7, 12 i 15, które z powodu dużych ilości błędów mogły spowodować losowość zwracanych rozwiązań. Postanowiliśmy sprawdzić średnie bez tych konkretnych instancji.

średnia procentów zgodności:	43,70%	53,01%	52,22%	62,86%	63,48%	58,73%

Minimum lokalne zniknęło. Wartość parametru 30 daje najlepsze wyniki, jednak w celu potwierdzenia tej tezy trzeba by przeprowadzić więcej testów.

Wpływ parametru parametr_number_of_ants (liczba mrówek) na jakość rozwiązania
Sprawdziliśmy 5 różnych wartości (50, 100, 150, 200 i 250) parametru na 15 różnych instancjach o różnych ilościach błędów, różnej długości całej sekwencji i oligonukleotydów, na których pracuje program. W testach przyjęliśmy wartość parametru parametr_of_time równą 30, parametr_not_chosen=7 oraz parametr_of_nonconformity=80. W tym teście również posłużyliśmy się miarą zgodności.

Główne sprawozdanie - problem SBH

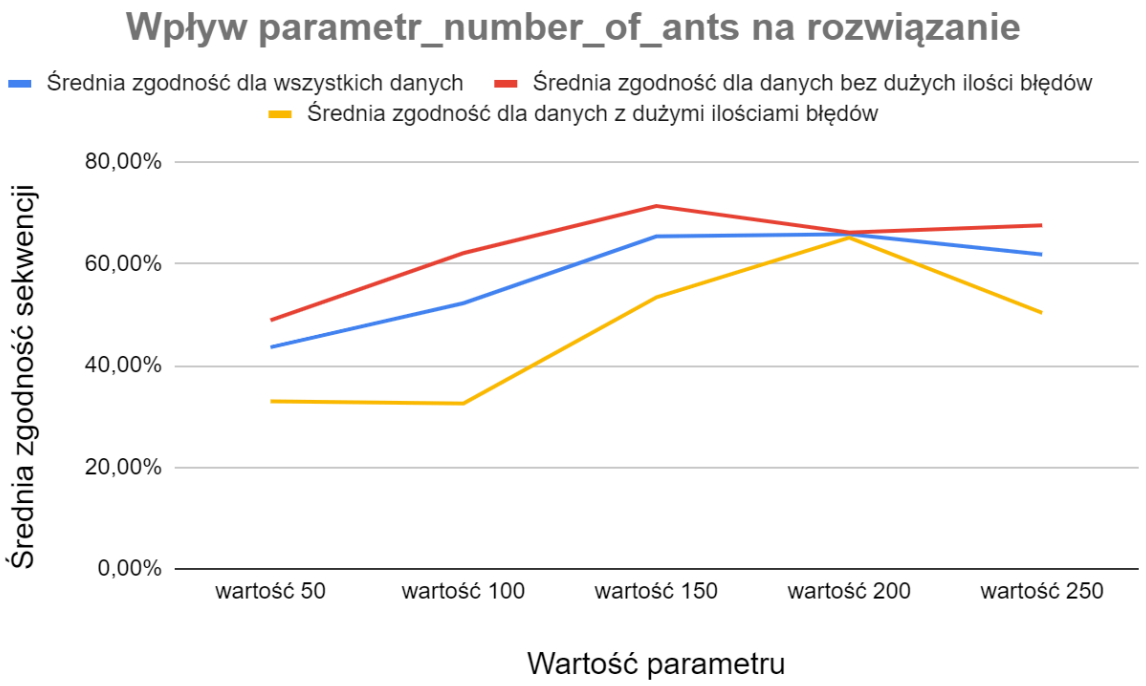
Poniższa tabela zawiera wyniki testu:

Wpływ liczby mrówek na rozwiązanie przy parametr_of_time równym 30 sekund							
długość sekwencji i k-meru	błędy	numer sekwencji	wartość 50	wartość 100	wartość 150	wartość 200	wartość 250
n=300, k=7	pos=5, neg=9	1	247	195	132	132	190
n=300, k=7	pos=10, neg=15	2	107	249	107	160	142
n=300, k=8	pos=10, neg=10	3	0	0	34	34	34
n=300, k=9	pos=10 neg=10	4	0	124	0	124	124
n=300, k=10	pos=25, neg=20	5	233	233	158	51	51
n=500, k=8	pos=5, neg=10	6	422	129	190	156	129
n=500, k=9	pos=5, neg=5	7	364	11	11	11	0
n=500, k=9	pos=20, neg=19	8	261	439	0	61	216
n=500, k=9	pos=80, neg=40	9	254	254	295	40	335
n=500, k=10	pos=5, neg=14	10	0	0	0	0	0
n=500, k=10	pos=120, neg=66	11	437	269	271	350	269
n=500, k=9	pos=20, neg=10	12	250	350	350	350	250
n=700, k=10	pos=120, neg=65	13	470	470	470	470	470
n=700, k=10	pos=80, neg=40	14	641	641	372	372	372
n=500, k=10	pos=40, neg=20	15	471	0	160	160	160
średnia:			277,13	224,27	170,00	164,73	182,80
odchylenie standardowe:			193,65	191,96	152,49	149,94	138,73
średnia procentów zgodności:			43,59%	52,26%	65,38%	65,81%	61,84%
średnia procentów zgodności pomijając trudniejsze przypadki:			48,90%	62,11%	71,37%	66,15%	67,57%
średnia procentów zgodności dla trudniejszych przypadków:			32,96%	32,56%	53,40%	65,13%	50,37%

Główne sprawozdanie - problem SBH

Kolorem pomarańczowym zaznaczyliśmy przypadki, zawierające w sobie znacząco większą ilość błędów. Postanowiliśmy porównać dane w trzech grupach: instancje o dużej ilości błędów, instancje o mniejszej ilości błędów oraz wszystkie instancje na jakich testowaliśmy algorytm.

Wizualizacja na wykresie:



Wnioski i spostrzeżenia:

Ciekawym spostrzeżeniem są różne najlepsze wartości parametrów dla różnych ilości błędów w spektrum. Jeżeli instancja nie ma ich zbyt wielu, zalecaną przez nas wartością do przyjęcia jako wartość parametru parametr_number_of_ants byłoby 150; jednak jeżeli błędów w spektrum jest więcej lub ich ilość nie jest znana najbezpieczniejszą wartością do przyjęcia zdaje się 200. Przy analizowaniu danych należy jednak zwrócić uwagę na dość dużą rozbieżność w wynikach dla różnych instancji, a nawet nieliniowe wyniki (raz lepsze, raz gorsze i znowu lepsze) w obrębie jednej i tej samej. Domyślamy się, że powodem jest oparcie metaheurystyki na losowości i nie wystrojenie pozostałych parametrów (bez czego puszczona dwa razy metaheurystyka z tymi samymi wartościami wszystkich parametrów mogłaby dać dwa różne rozwiązania). Mimo to, widzimy, że algorytm mrówkowy jest w stanie otrzymać miarę Levenshteina równą 0 (zgodność sekwencji = 100%) dla prostszych instancji. Warto byłoby przeprowadzić dodatkowe, większe testy po wystrojeniu innych parametrów w celu upewnienia się, że ich wartości nie wpłynęły na najlepszą wartość badanego w tym teście parametru.

Główne sprawozdanie - problem SBH

Część 4 - porównanie wyników algorytmu losowego i metaheurystyki

Mimo, że metaheurystyka nie została przez nas całkowicie wystrojona, co oznacza, że nie zwraca jeszcze najlepszych rozwiązań, jakie mogłaby zwracać, postanowiliśmy porównać jej wyniki z wynikami algorytmu losowego w celu ocenienia potencjału metaheurystyki już na tym etapie. Tak jak w poprzednich testach, przetestowaliśmy oba algorytmy na 15 zróżnicowanych instancjach i oparliśmy się na średnich miarach Levenshteina oraz naszej mierze zgodności. Przyjęliśmy wartości: parametr_of_time=30; parametr_not_chosen=7; parametr_number_of_ants=150 i parametr_of_nonconformity=80. Z uwagi na czynnik losowy zwracanego rozwiązania, postanowiliśmy każdy algorytm dla danej instancji uruchomić 3 razy i wybrać najlepszy z uzyskanych wyników.

Poniższa tabela zawiera wyniki testu:

Porównanie wyników zwracanych przez oba algorytmy				
długość sekwencji i k-meru	błędy	numer sekwencji	Algorytm losowy	Metaheurystyka
n=300, k=9	pos=20, neg=15	1	0	0
n=300, k=7	pos=20, neg=15	2	111	88
n=300, k=8	pos=40, neg=25	3	85	0
n=300, k=9	pos=50 neg=25	4	113	166
n=300, k=10	pos=25, neg=20	5	75	0
n=500, k=8	pos=10, neg=9	6	425	425
n=500, k=9	pos=5, neg=5	7	85	0
n=500, k=9	pos=20, neg=21	8	279	265
n=500, k=9	pos=80, neg=40	9	405	405
n=500, k=10	pos=5, neg=14	10	242	242
n=500, k=10	pos=120, neg=66	11	471	471
n=500, k=10	pos=0, neg=80	12	386	386
n=500, k=10	pos=80, neg=0	13	355	257
n=700, k=10	pos=80, neg=41	14	231	188
n=700, k=10	pos=60, neg=30	15	393	0
średnia:			243,73	192,87
odchylenie standardowe:			155,79	173,82
średnia procentów zgodności:			50,22%	59,89%

Główne sprawozdanie - problem SBH

Wnioski i spostrzeżenia:
Dla 15 różnych instancji tylko w jednej algorytm losowy dał rozwiązanie lepsze niż metaheurystyka (prawdopodobnie z powodu czynnika losowego). Oba algorytmy dawały podobne rozwiązania głównie przy instancjach o dużych ilościach błędów. W większości przypadków metaheurystyka dawała lepszy rezultat, zwłaszcza duża różnica w jakości zwracanych rozwiązań występowała przy długich sekwencjach (700 nukleotydów). Widać jasno też, że algorytm mrówkowy dużo częściej potrafił bezbłędnie odtworzyć sekwencję wejściową.

Konkluzje i uwagi końcowe

Podsumowując przedstawione w tym dokumencie wyniki pracy, chcielibyśmy przytoczyć nasze opinie oraz przemyślenia w kontekście funkcjonalności algorytmu i postępów pracy poczynionych przez cały semestr.

Wnioski końcowe

Analiza stosowalności programu wykazuje iż w obecnej formie algorytm “losowy” zwraca odległe jakościowo rozwiązania. Jest to wynikiem zakleszczania się algorytmu dla niektórych przypadków, co skutkuje krótką sekwencja wynikową. Jednak zaimplementowana metaheurystyka oparta na algorytmie mrówkowym już teraz przy prostszych instancjach potrafi bezbłędnie odnaleźć sekwencję oryginalną i wykazuje duży potencjał w kontekście dalszego udoskonalania m.in. w kontekście dostrajania parametrów. Praca nad tym projektem pozwoliła nam poznać zagadnienia z zakresu implementacji algorytmów przybliżonych, teorii złożoności oraz zalet i ograniczeń metod sekwencjonowania (w szczególności sekwencjonowania przez hybrydyzację). Przedstawione w dokumencie opisy i wyniki są efektem ciężkiej pracy przez cały semestr, staraliśmy się przedłożyć nasz tok rozumowania w jak najbardziej przejrzystej formie.

Procedura uruchomienia programu

Wszelkie pliki konieczne do prawidłowego działania programu zostały załączone do bieżącego sprawozdania, w razie problemów można pobrać je z repozytorium GitHub, do którego link został podany w następnym akapicie. Na całokształt programu składa się kilka plików wykonywalny oraz nagłówkowych, których połączenie w jeden spójny projekt jest niezbędne. W ramach pracy korzystaliśmy z takich środowisk programistycznych jak

Główne sprawozdanie - problem SBH

Code::Blocks oraz Visual Studio. Poniżej znajduję się lista wskazówek dotyczacych korzystania z programu:

- w kodzie umieszczone są komentarze mające na celu szybkie wyszukiwanie odpowiednich fragmentów
- do sprawdzania działania metaheurystyki dla sekwencji dłuższych niż 550 nukleotydów polecamy środowisko Code::Blocks lub inne (nie Visual Studio)
- podawanie parametrów do programu
    - w pliku *main* (liczba błędów pozytywnych i negatywnych, długość sekwencji, wielkość oligonukleotydu, nazwy plików odczytu i zapisu)
    - w pliku *Instance_generator* (nazwy plików odczytu i zapisu zgodnie z konkretnym celem)
    - w pliku *Weak_algorithm* (wartość progu parametr_not_chosen)
    - w pliku *Strong_algorithm* (liczba mrówek, wartość prawdopodobieństwa losowej trasy oraz czas)

Załączniki

Dbając o kompletność sprawozdania końcowego dołączamy niezbędne dane oraz elementy stanowiące o poprawności przedstawionego toku rozważań.

Lista załączników:

- folder *instancje* (instancje związane z przeprowadzonymi testami) - zawierający pliki tekstowe odpowiednio zawierające sekwencje, spektrum oraz wynik; struktura podkatalogów została zaprojektowana tak, aby w prosty sposób odnaleźć dane dla konkretnego testu, przykład:
  plik o nazwie: *sekwencja1.txt* w podkatalogu Błędy pozytywne zawiera sekwencje użytą w testowaniu wpływu błędów pozytywnych i odpowiada indeksowi 1 w tabeli
- kody do zaimplementowanego programu w C++ - pliki z rozszerzeniami odpowiednio cpp i h są elementami całego projektu
- link do repozytorium w serwisie GitHub, które również zawiera pełen zestaw plików potrzebnych do uruchomienia programu, link:

https://github.com/EmiliaBacik/Optymalizacja-kombinatoryczna/tree/5718acee5848e38af3e1b2c8fa854e0d2a5935c0/Optymalizacja%20kombinatoryczna?fbclid=IwAR0wmqE2z5Py3xsHHumu4atXuyB0PfrcuXpjUT_EQbNNeitJb2_1q51RMyU

Główne sprawozdanie - problem SBH

Źródła i odniesienia

W ramach pracy nad projektem korzystaliśmy z rozmaitych źródeł naukowych oraz materiałów przygotowanych dla nas przez prowadzącego. Poniżej znajduję się kompletna lista źródeł wiedzy:

- materiały na stronie prowadzącego: https://www.cs.put.poznan.pl/mradom/
- *“Artificial ants as a computational intelligence technique”* M Dorigo, M Birattari, T Stutzle - IEEE computational intelligence magazine, 2006
- J-H. Zhang, L-Y. Wu, and X-S. Zhang. Reconstruction of DNA sequencing by hybridization. Bioinformatics, 19:14–21, 2003.
- implementacja wyznaczania odległości Levenshtein’a w C++: http://www.algorytm.org/przetwarzanie-tekstu/odleglosc-levenshteina-odleglosc-edycyjna.html
- narzędzie online do wyznaczania miary Levenshtein’a: https://planetcalc.com/1721/#google_vignette