Optymalizacja Kombinatoryczna – Bioinformatyka

Adam Łangowski 147896

1. Wybrany problem:

Dane jest klasyczne spektrum DNA jako zbiór $\mathbf{S}^{class}$, długość oryginalnego DNA *n*, długość użytych oligonukleotydów *l=k* (dla *k*>=7) oraz procentowa liczba błędów pozytywnych i negatywnych (potencjalnie obu typów).

2. Rozwiązanie:

Sekwencja DNA zrekonstruowana z oligonukleotydów.
Metoda: **algorytm genetyczny**.

Algorytm posiada następujące dane wejściowe:

I. **Długość DNA** – *given_length_n*
II. **Długość oligonukleotydów** – *given_k*
III. Informacje jakie błędy są w spektrum:
- **Błędy negatywne** – brak niektórych fragmentów naszego badanego DNA
- **Błędy pozytywne** – dodatkowe, fałszywe elementy w spektrum
- Ogólna **procentowa ilość błędów** – poza insercjami i delecjami obejmuje jeszcze substytucję (zamianę nukleotydu na inny)
(*negative_errors_percentage; positive_errors_percentage; errors_percentage)*

IV. **Ilość permutacji** – *permutations_count*
V. **Maksymalną liczbę generacji** – *max_generations_count*
VI. **Rozmiar początkowej populacji** – *population_count*

3. Generator instancji

Zaimplementowano generator instancji dla wybranego problemu z opisanymi powyżej parametrami wejściowymi (I-III). Zawiera on funkcje:

- tworzenia oligonukleotydów o długości k
- dzielenia sekwencji na nakładające się fragmenty o długości k
- dodawania mutacji do sekwencji.

W efekcie finalnym **generator zwraca spektrum** reprezentujące spektrum danej sekwencji DNA. Spektrum DNA określamy jako zbiór oligonukleotydów z komórek mikrochipu, które przyczepiły się do DNA w trakcie sekwencjonowanie przez hybrydyzację, z których każdy z nich jest o długości k.

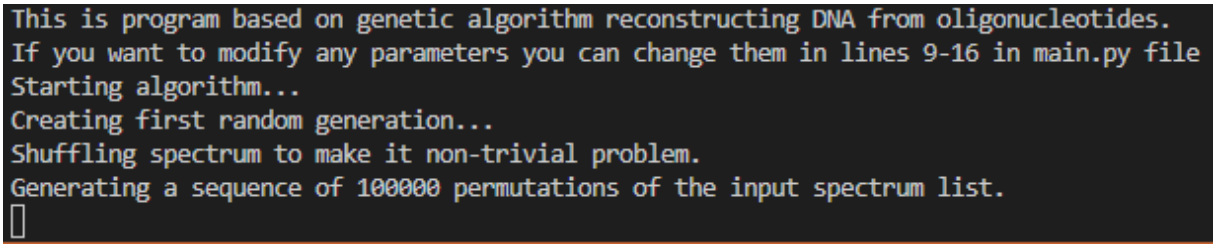
4. Opis algorytmu genetycznego:

Algorytm genetyczny składa się z kilku różnych parametrów, które decydują o jego działaniu. W naszym przypadku obejmują one:

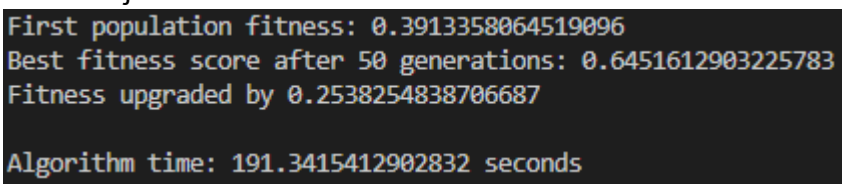
- Zbiór danych wejściowych – **spectrum skonstruowane przez generator na podstawie zadanych parametrów.**

- Funkcję celu - algorytm genetyczny wymaga określenia funkcji celu, która będzie oceniać jakość poszczególnych rozwiązań i decydować o tym, które z nich są lepsze. W naszym przypadku funkcja celu określa jak dobrze sekwencja pasuje do danego spektrum. **Ocena osobników, czyli możliwych rozwiązań, przeprowadzana jest za pomocą funkcji przystosowania (*mean_fitness*), będącej miarą ich jakości. Chcemy dążyć do maksymalizacji tej funkcji.**
- Populacja początkowa - algorytm genetyczny wymaga określenia początkowej populacji chromosomów, które będą używane do rozpoczęcia procesu ewolucji. **Pierwsza populacja, od której zaczyna się działanie algorytmu genetycznego, formowana jest w sposób losowy i ma ustaloną wielkość.**
- Metoda selekcji – **selekcja turniejowa (torunament_selection).**
  Jej działanie opiera się kolejno na: dzieleniu losowo osobników na pewną liczbę grup; następnie w każdej grupie osobniki rywalizują ze sobą i wygrywa ten, który posiadał najlepszą funkcję przystosowania. Dla każdej epoki wyliczana zostaje wartość parametru *mean_fitness,* która jest prezentowana na wyjściu, w czasie rzeczywistym działania algorytmu.
- Operatory genetyczne - algorytm genetyczny wymaga określenia operatorów genetycznych, takich jak **krzyżowanie i mutacja**, które będą stosowane w celu wygenerowania nowych rozwiązań.
  Zaimplementowano krzyżowanie jednopunktowe (przecięcie chromosomów rodziców względem wybranego punktu) oraz wielopunktowe (przecięcie chromosomu w kilku losowych miejscach).
  Możliwe mutacje również tutaj obejmują insercję, delecję oraz substytucję.
- Warunki stopu – wymagane jest określenia warunków stopu, które będą decydować o tym, kiedy proces ewolucji zostanie zakończony. W naszym przypadku będzie to **ograniczona z góry ilość epok**. Eksperymentowano także z ustawieniem zatrzymania algorytmu po braku polepszania kolejnych pokoleń. Jednak dla większości ustawień parametrów algorytm wciąż bardzo nieznacznie polepszał swoje potomne rozwiązania.

Początek działania programu:



Informacje końcowe:

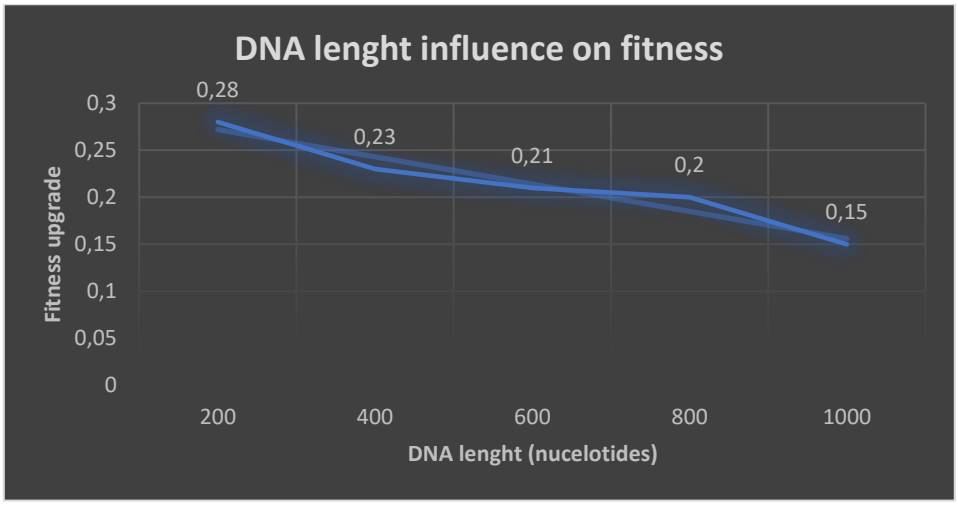


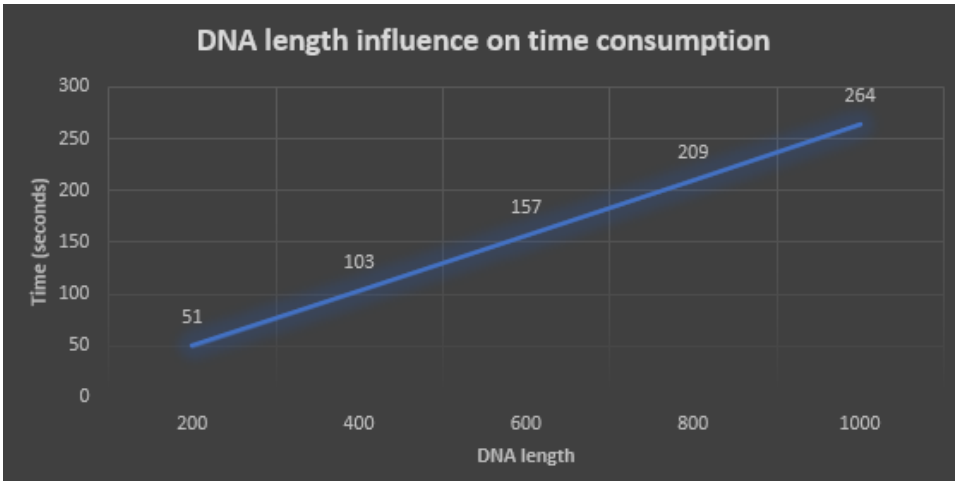
5. Zmienne parametry, tunning:

- ✓ analiza w kolejnym punkcie dotyczyć będzie modyfikacji parametrów wejściowych dla generatora, a także tych dotyczących funkcjonowania algorytmu genetycznego
- ✓ **ustawiono wagi informujące o częstości poszczególnych rodzajów zjawiska krzyżowania (crossing over), odwzorowujące w przybliżeniu częstość mutacji w rzeczywistości:**
  jednopunktowe = 0.6; wielopunktowe = 0.3, brak zjawiska = 0.1
- ✓ **ustawiono maksymalną liczbę kolejnych generacji na 50 dla większości testów – po osiągnięciu tej epoki rozwiązań algorytm znacząco wolniej poprawia swoje wyniki**
- ✓ **spectrum po stworzeniu zostaje oczywiście wymieszane, inaczej algorytm zdecydowanie za łatwo znajduje rozwiązanie bliskie ideału, a już pierwszą utworzoną populację charakteryzuje bardzo wysoki (>90%) fitness.**

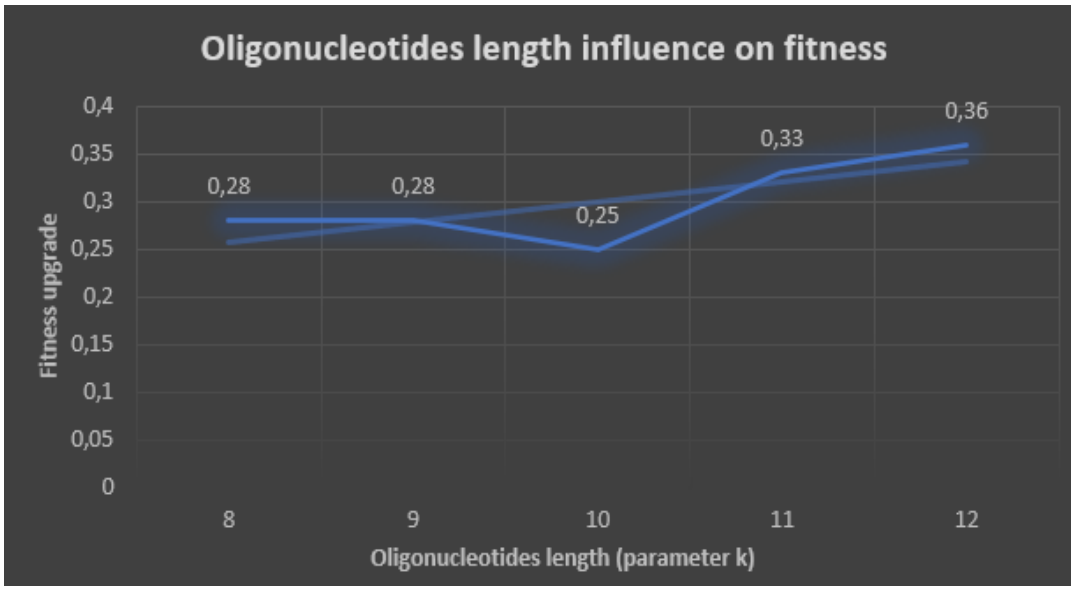
6. Testy i analiza wyników

Testy wykonano przy użyciu środowiska PyCharm oraz Pythona w wersji 3.9.14
Parametry: Procesor AMD Ryzen 5 3600 6-Core Processor, 3593 MHz; 16GB RAM.

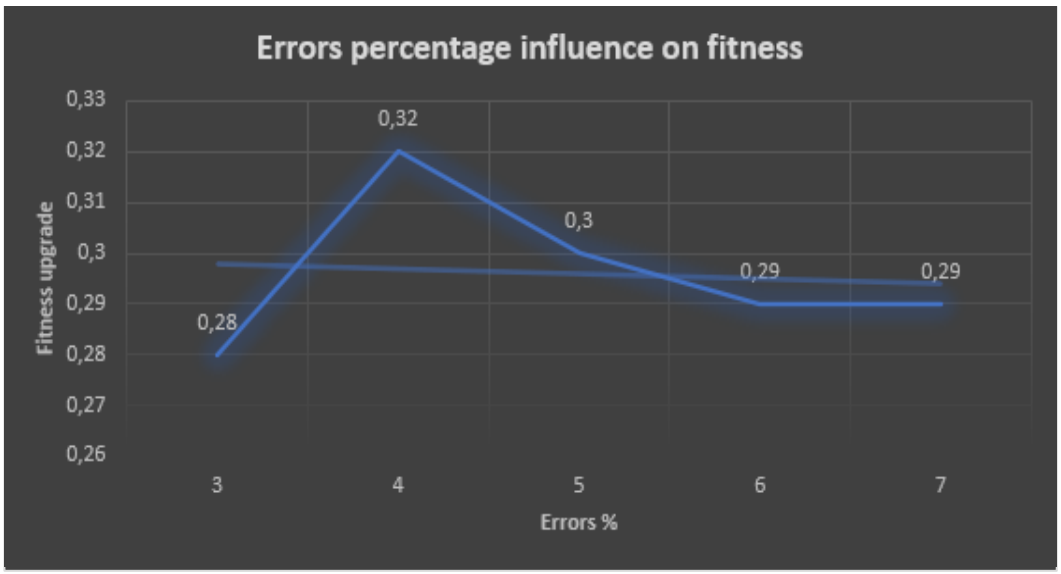




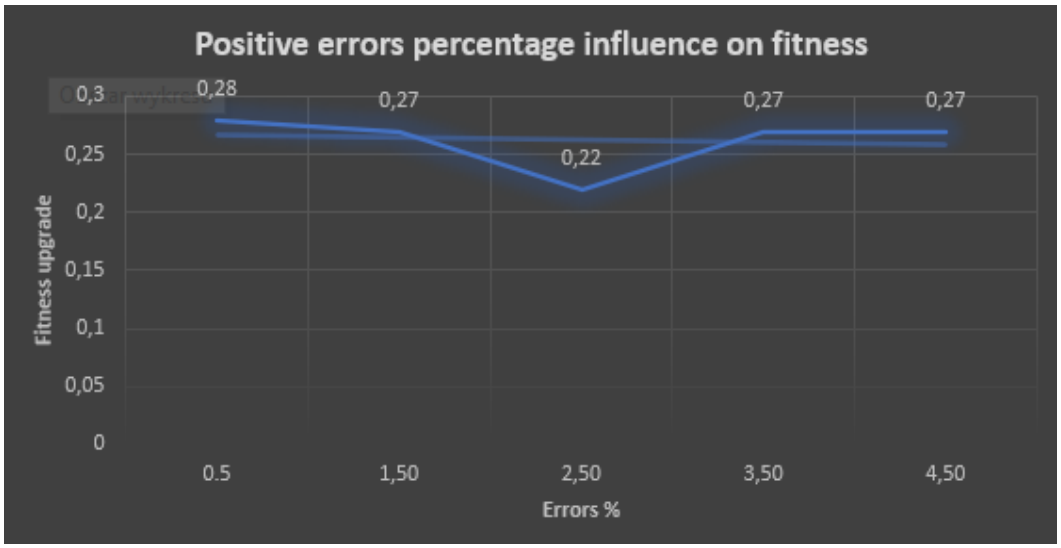
- Zwiększając wielkość populacji powodujemy zmniejszenie efektywności algorytmu (zaczyna spadać *fitness_upgrade)* oraz znacznie zwiększamy czas działania programu (dla dwukrotnej wielkości populacji ponad dwukrotnie rośnie czas).

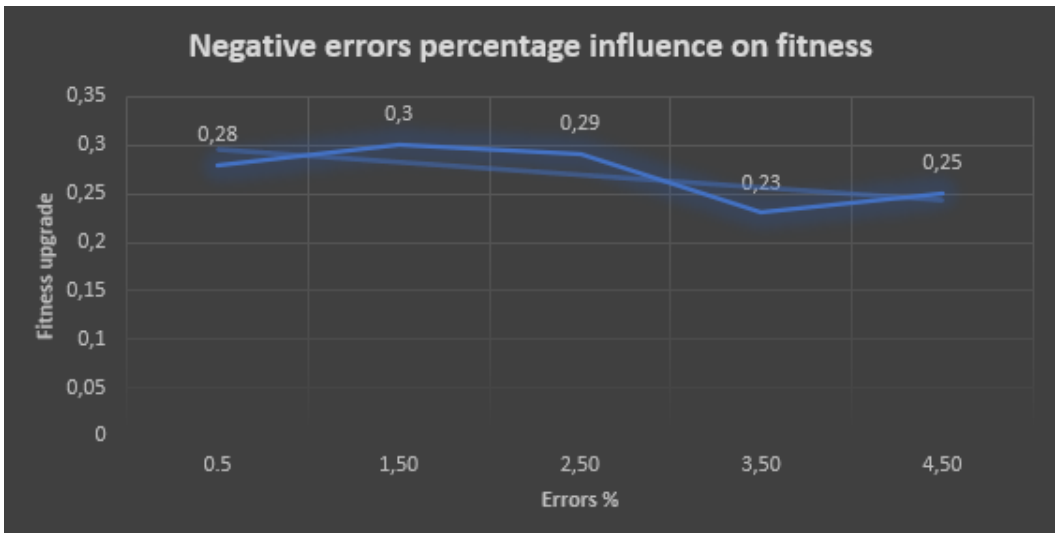


- Zwiększenie długości oligonukleotydów implikuje lepszy fitness_upgrade, zwiększając również niestety czas działania. Jest to zgodne z oczekiwaniami, ponieważ dłuższe fragmenty łatwiej zostają odnalezione. Dla k = 12 i pozostałych parametrów domyślnie ustawionych osiągnięto najlepszy wynik fitnessu dla 50. Populacji równy 0.67.

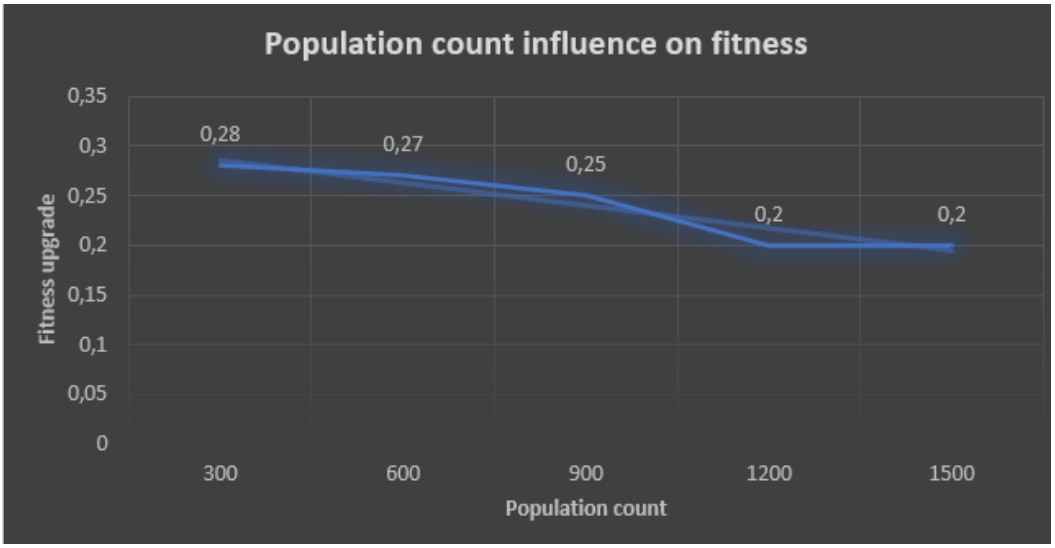


- Rosnąca ilość błędów polegających na substytucji nukleotydów nieznacznie może pogarszać jakość znajdywanych rozwiązań.





- Błędy negatywne mają nieznacznie większe przełożenie na wyniki niż błędy pozytywne – brakujące elementy spektrum są zatem bardziej istotne niż pojedyncze dodatkowe nukleotydy.



- Rosnąca liczba populacji początkowej nie powoduje wcale polepszenia fitnessu finalnego rozwiązania, zwiększa natomiast znacznie czas operacji (do 115 sekund przy wielkości populacji 1500). Na tym etapie zbadana została bliżej początkowa wartość populacji. Najlepszy wskaźnik *fitness_upgrade* przy jednoczesnym zmniejszaniu czasu algorytmu otrzymujemy w okolicach 100-120 osobników. Zmniejszając dalej ten parametr zaczynamy otrzymywać dużo gorsze rezultaty.

7. Podsumowanie

Algorytmy metaheutystyczne, w tym algorytm genetyczny, od klasycznych metod optymalizacyjnych odróżnia kilka istotnych cech: nie przetwarzają bezpośrednio parametrów zadania, lecz ich zakodowaną postać; nie wychodzą z pojedynczego punktu, lecz wykorzystują pewną ich populację; nie korzystają z pochodnych ani innych informacji pomocniczych, lecz z funkcji celu/przystosowania; nie stosują reguł deterministycznych, lecz probabilistyczne. Powyższe cechy dają im odporność oraz przewagę nad innymi metodami poszukiwania.
Metoda selekcji turniejowej dobrze sprawdza się, gdy mamy do czynienia z liczną populacją, gdyż pozwala pominąć etap sortowania osobników od najlepszego do najgorszego (w obszarze funkcji selekcji).
Z perspektywy złożoności obliczeniowej, ten algorytm genetyczny z selekcją turniejową charakteryzuje złożoność większa od: **O(g(nm + nm + n))** – gdzie ‘g’ to liczba generacji, ‘n’ to wielkość populacji oraz ‘m’ jako wielkość pojedynczych rozwiązań. Taka złożoność charakteryzuje algorytm z krzyżowaniem punktowym bez błędów, a w tej implementacji występuję również wielopunktowy crossing-over, który prawdopodobnie zwiększa zauważalnie złożoność obliczeniową oraz spektrum z błędami zarówno pozytywnymi jak i negatywnymi.