**Programowanie Efektywnych Algorytmów  
Zadanie projektowe 3**

„Implementacja i analiza efektywności algorytmu genetycznego dla problemu Komiwojażera”

Kacper Chrostowski 259120

**Problem Komiwojażera**

Problem optymalizacyjny polegający na znalezieniu w n-wierzchołkowym pełnym grafie takiego cyklu Hamiltona(doliczając do tego powrót do wierzchołka startowego), w którym suma długości ścieżek między wierzchołkami jest jak najmniejsza. Algorytm genetyczny jest heurystyką symulującą faktyczne sposoby ewolucji w środowiskach naturalnych. Kolejno przeprowadzana jest krzyżowanie, mutacja i selekcja nowej populacji.

W omawianym rozwiązaniu zastosowano krzyżowanie OrderCrossover (OX) oraz dwa rodzaje mutacji: Scramble oraz Invert.

**Przebieg algorytmu:**

1)Wygeneruj startową populację składającą się z losowych permutacji rozwiązania tsp

2)Wylosuj dwa chromosomy należące do populacji i porównaj ich długość, zapamiętaj ten w którym trasa jest krótsza jako pierwszego rodzica

3)Powtórz krok 2 i zapamiętaj wynik jako drugiego rodzica

4)Korzystając z krzyżowania wygeneruj parę potomstwa dla wybranych rodziców i zapisz je w wektorze

5)Wróć do punktu 2 i powtarzaj tak długo aż ilość wygenerowanego potomstwa będzie wynosiła rozmiar\_startowej\_populacji \* współczynnik\_krzyżowania.

6)Dla każdego potomka sprawdź prawdopodobieństwo wystąpienia mutacji (określane przez współczynnik mutacji). Jeżeli prawdopodobieństwo na to wskaże, wykonaj odpowiednią mutację na potomku.

7)Połącz ze sobą struktury przechowujących rodziców i dzieci, posortuj populację rosnąco, a następnie usuń nadmiar chromosomów w taki sposób aby struktura miała rozmiar wskazany przez rozmiar startowej populacji

8)Wróć do punktu 2 i powtarzaj dopóki nie minie wskazany czas

9)Najlepsze znalezione rozwiązanie znajduje się w populacji na pierwszym miejscu.

**Opis klas:**

*Menu*: Jest odpowiedzialna za nawigację po programie. Przyjmuje instrukcje od użytkownika, a następnie uruchamia odpowiednie metody.

*Timer*: Klasa odpowiedzialna za pomiar czasu. Zawiera metody uruchamiające pomiar czasu oraz pobierającą aktualny czas.

*Graph*: Klasa, po której dziedziczy klasa Genetic. Służy ona do obsługi danych, na których operuje algorytm (wczytanie i wyświetlanie danych) oraz zawiera metodę obliczającą długość podanej trasy.

*Genetic:* Klasa, która przeprowadza algorytm genetyczny. Zawiera główną metodę , w której wykonuje się główne zadanie oraz metody będące niezbędne do przeprowadzenia algorytmu czyli operator krzyżowania OrderCrossover oraz dwa rodzaje mutacji Scramble oraz Inverse.

**Plan eksperymentu**

Dla każdego algorytmu przeprowadzono 10 testów. Każdy test trwał 120 sekund oraz co 10 sekund pobierano najlepsze znalezione rozwiązanie. W ten sposób będzie można przeanalizować jak z upływem czasu poprawia się rozwiązanie.

Dla każdego zestawu danych testy powtórzono dla dwóch rodzajów mutacji, oraz dla populacji początkowych liczących 1000, 10000 i 100000 osobników.

Testy przeprowadzone zostały na grafach zapisanych w plikach *br17*, *ft53* oraz *ftv170*. Najlepsze otrzymane wyniki będą porównywane z najlepszymi rozwiązaniami tych problemów rozwiązanych przy pomocy algorytmu TabuSearch z poprzedniego zadania.

**Wyniki eksperymentu**

**Wnioski**

Algorytm działa poprawnie. Dla małych i średnich problemów wraz ze wzrostem rozmiaru populacji można zaobserwować poprawę wyniku dla każdego problemu. Natomiast dla problemów dużych większa populacja daje gorsze wyniki. Dla dużego problemu najlepiej sprawdzała się populacja o rozmiarze 10000. Algorytm dobrze radzi sobie z małymi i średnimi problemami. W najlepszym przypadku znajdowane były rozwiązania o kilkuprocentowym błędzie. Dla dużych instancji natomiast znajdowano bardzo niedobre rozwiązania. W przypadku każdego testu prócz tego z 170 miastami, Algorytm Genetyczny miał lepsze najoptymalniejsze rozwiązanie niż Tabu Search. Większa populacja startowa przeciwdziałała wpadaniu w lokalne ekstrema. Lepszym schematem mutacji okazał się TUTAJ CO JEST LEPSZE.

Za gorsze wyniki dla dużych instancji można obarczyć optymalizację kodu. Najprawdopodobniej największy wpływ miało zastosowanie gotowego algorytmu sortowania, który do poprawnego działania używał metody obliczającej długość trasy zadanego wektora, w którym przechowywano ścieżkę. Lepsze wyniki można by także otrzymać dzięki lepszemu doborowi parametrów. Świadczy o tym fakt, że mniejsza populacja startowa w przypadku testu z 170 miastami dała lepszy wynik.

**Źródła:**

<http://staff.iiar.pwr.wroc.pl/antoni.sterna/>  
<https://www.ii.uni.wroc.pl/~prz/2011lato/ah/opracowania/met_podz_ogr.opr.pdf>