**METAHEURYSTYKI W ROZWIĄZYWANIU PROBLEMÓW**

SPRAWOZDZNIE 1

TEMAT:

ROZWIĄZANIE PROBLEMU KOMIWOJAŻERA Z UŻYCIEM ALGORYTMU GENETYCZNEGO

Wojciech Jankowiak

221013

**WSTĘP**

Problem komiwojażera, zwany też problemem wędrującego handlarza (Traveling Salesman Problem – TSP), został po raz pierwszy sformułowany w połowie XIX wieku, jednak formalnie został spisany dopiero w 1930 roku przez Karla Mengera – Austriackiego matematyka.

Problem dotyczy znalezienia jak najkrótszego cyklu Hamiltonowskiego w grafie pełnym ważonym. Jest on przykładem problemu NP.-trudnego. Przestrzeń rozwiązań staje się ogromna już dla małej liczby wierzchołków zadanego grafu , co w połączeniu z brakiem algorytmu (innego niż brute-force) czyni komiwojażera strasznie pracochłonnym problemem, którego zwykłymi sposobami nie da się rozwiązać w zadowalającym czasie (przynajmniej od pewnej liczby wierzchołków).

Mimo braku algorytmu zwracającego idealne rozwiązanie dla danego problemu, opracowanych zostało wiele metod, które w zadowalającym czasie zwracają rozwiązanie. Nie zawsze jest ono najlepsze, nie ma nawet gwarancji znalezienia jakiegokolwiek, jednak wyniki są na tyle dobre, że akceptowalne. Jednym z tych podejść jest algorytm genetyczny.

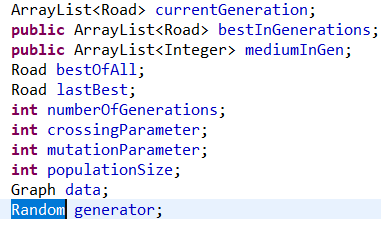
„Algorytm” genetyczny, zwodniczo nazwany, algorytmem nie jest. Jest to heurystyka, czyli ciąg działań, który zwraca rozwiązanie problemu w zadowalającym czasie, jednak nie zawsze takie samo, niekoniecznie najlepsze, a czasem w ogóle. Działanie jego polega na symulacji naturalnej ewolucji, na zasadzie „przetrwają najsilniejsi”. Na początku losowane są osobniki do pierwszej generacji – ich dobór jest przypadkowy, byle były żywe, czyli trzymały się ustalonych zasad zadanego problemu. Po wylosowaniu, osobniki są oceniane, pod względem wcześniej zdefiniowanej funkcji oceny(tu najkrótsza całkowita droga), a następnie wybierane do następnej generacji te, które są najlepiej przystosowane. Samo to jednak powodowałoby szybki zastój algorytmu w maksimum lokalnym, i dominacje osobnika o jednym genotypie. Dlatego wprowadza się do algorytmu operatory krzyżowania i mutacji, które wprowadzają do populacji większą różnorodność. Krzyżowanie polega na „wymianie” genów osobników-rodziców w celu stworzenia jednego lub więcej osobników-dzieci. Mutacja natomiast jest zamianą jednego genu w określonym osobniku. Ważne w działaniu tego algorytmu jest aby wielkość populacji cały czas była taka sama. Dokładniejsze wyjaśnienie znajduje się w kolejnym rozdziale sprawozdania.

**ALGORYTM GENETYCZNY**

W tym rozdziale znajdzie się szczegółowe objaśnienie działania i implementacji algorytmu genetycznego.

W programie został zdefiniowany graf(nazywany dalej mapą), który reprezentuje na zasadzie listy sąsiedztwa miasta oraz odległości między nimi. Dane wczytywane są z pliku dzięki osobnej klasie czytającej plik i parsującej zawarte w nim dane.

Klasa algorytmu genetycznego(class GeneticAlgorithm) zawiera następujące pola:



Gdzie:

currentGeneration służy do zapamiętania osobników aktualnej generacji

bestInGeneration oraz mediumInGen zapamiętują odpowiednio najlepsze osobniki z każdej generacji oraz średnią ocenę generacji.

bestOfAll – najlepsze rozwiązanie, które udało się znaleźć

lastBest – najlepsze rozwiązanie z ostatniej generacji

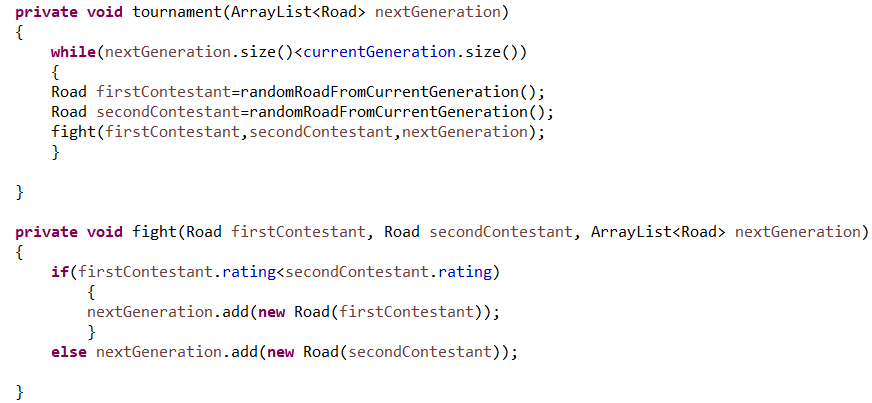
numberOfGenerations, crossingParameter, mutationParameter, populationSize – parametry sterujące działaniem algorytmu

data – graf z danymi z pliku

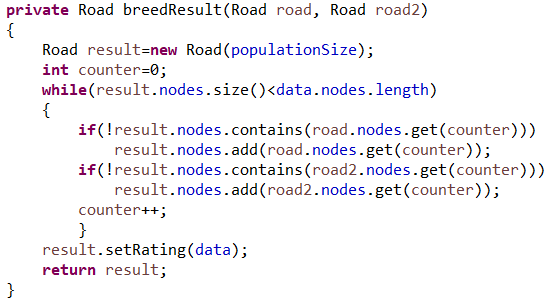
generator – generator liczb losowych.

Po zainicjalizowaniu wszystkich potrzebnych pól, można wywołać publiczną metodę passGenerations, która *numberOfGeneration* razy wykonuje kluczowe czynności (wywołuje metodę passOneGeneration).

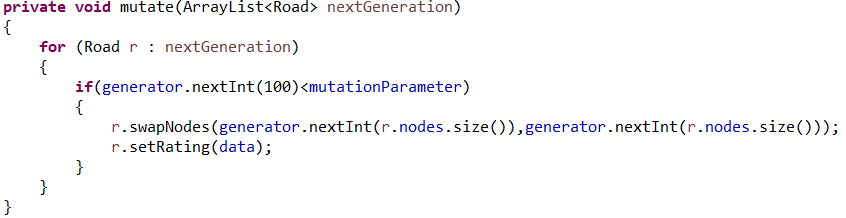
Najpierw uruchamiany jest algorytm symulujący selekcję naturalną. Zaimplementowano go na zasadzie turnieju dla dwóch osobników, gdzie szansę na wejście do następnej generacji dostaje lepiej przystosowany osobnik.



Gdy turniej się dokończy, następuje krzyżowanie osobników nowo wybranych. Algorytm krzyżowania jest prosty – na kolejną pozycję osobnika-dziecka wpisywana jest i-ta pozycja jednego z rodziców, na następną i-ta pozycja drugiego z rodziców, przy czym sprawdzana jest unikalność każdego wpisywanego elementu i wpisywane tylko te, które nie zawierają się w osobniku-dziecku. Dzięki takiemu podejściu, zwłaszcza w późniejszej fazie działania algorytmu umożliwia się znalezienie dużo krótszej odległości miedzy miastami, które w pierwotnych osobnikach były daleko od siebie. To, czy do krzyżowania w ogóle dojdzie jest oczywiście sterowane odpowiednim parametrem.

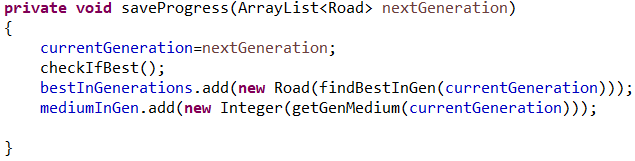


Mutacja, która następuje po krzyżowaniu dodaje do generacji nowego osobnika, który różni się od swego pierwowzoru dwoma genami (zamiana pozycji jednego z drugim).



W tym momencie nowa generacja zawiera o wiele za dużo osobników niż powinna. Dlatego do momentu osiągnięcia odpowiedniej ilości osobników w generacji usuwane są losowe osobniki, co tłumaczy się wystąpieniem „nieszczęśliwych wypadków”.

Teraz algorytm jest gotowy do przejścia do następnej generacji, więc obecna generacja zastąpiona zostaje nową, a wyniki zapisywane do odpowiednich pól klasy.



Po przebiegu wszystkich generacji test dla tych samych parametrów jest powtarzany jeszcze 9 razy, po czym uśrednione wyniki najlepszych osobników oraz średnich wyników generacji zapisywane są do pliku, w celu umożliwienia dalszej analizy.

**WYNIKI TESTÓW**

Przeprowadzono testy, dla wielu różnych parametrów dla pliku, kroE100.tsp , wyniki których przedstawiono i opisano na wykresach poniżej.

Objaśnienie:

Na większości wykresów znajdują się wyniki kilku testów dla podobnych danych. Pierwsze trzy wykresy przedstawiają najlepszych osobników z odpowiedniej generacji, kolejne przedstawiają wyniki średnie. Z prawej strony wykresu, przy kolorze linii opisane są w kolejności: ilość generacji, współczynnik krzyżowania (w%), współczynnik generacji (w %), wielkość populacji. Analiza znajduje się pod wykresami.

Analiza:

Na pierwszy rzut oka zauważyć można niewiele różniące się kształty wykresów najlepszych osobników i wykresów średnich z generacji. Zauważyć również można moment (w ilościach generacji), gdzie wykres wypłaszcza się, a co za tym idzie algorytm zwalnia. Jest to około 2000 do 2500 generacji. Jeśli chodzi o współczynniki mutacji i krzyżowania, to bardzo dobrze działa krzyżowanie od 30% do 70%, natomiast współczynnik mutacji powyżej 30% zaczyna bardzo psuć wyniki. Jeśli chodzi o ilość osobników w populacji, najlepiej wypadł test z tysiącem osobników, łącząc w sobie wysoką sprawność algorytmu, z względnie małym czasem potrzebnym na wykonanie testu. Jednak co okazało się mało intuicyjne, dla wykonanego testu z populacją 20, nie zatrzymał się algorytm wcześnie w maksimum lokalnym, a co więcej dążył do zrównania z resztą testów w następnych kilku tysiącach populacji( czas potrzebny do wykonania tego samego testu dla populacji 5000 a 20 znacznie się różni).Co więcej, dla 20 osobników w generacji, już przy 5000 generacji udało się osiągnąć wyniki ~45000, czyli takie jakie osiągano przy 2000 generacji na populacjach liczących po kilka tysięcy! Najlepsze wyniki osiągnięto dla testu z danymi 2000 generacji, współczynnik krzyżowania 70(wysoko!), mutacji 30(powyżej 30 zaczęło się psuć), oraz ilości osobników = 10000. Warto jednak zaznaczyć, że test ten otrzymał wynik ~36000, przy czym ten sam test na 2000 generacji znalazł wynik ~39000, co przy startowej wartości ~140000 jest małym zyskiem, w stosunku do czasu potrzebnego.

**WNIOSKI**

Algorytmy genetyczne w bardzo krótkim czasie pozwoliły na znalezienie zadowalających rozwiązań w bardzo krótkim czasie. Widoczną zaletą używania tego sposobu rozwiązywania jest możliwość dostrajania parametrów pod względem wymagań – czasowych, jakościowych. Pewne zachowania okazały się zaskakujące (dobre wyniki dla bardzo małej populacji dwudziestu osobników), niektóre nie zaskoczyły w ogóle( złe wyniki dla testów z wysokim poziomem mutacji).

Bardzo miłym zaskoczeniem były dobre wyniki przy wysokim współczynniku krzyżowania, gdyż operator ten zaimplementowany został „autorsko”, i eksperymentalnie, z założenia miał działać tylko dla małych wartości parametru krzyżowania, jednak jak pokazały testy, najlepsze wyniki algorytm osiągał dla krzyżowania =70% (co udowadnia, że krzyżowanie nie psuje populacji, lecz znacząco ją wzmacnia).