Autor: Dawid Smalcuga
Temat: implementacja algorytmu TCX dla problemu MSTP przy użyciu GA (https://www.sciencedirect.com/science/article/pii/S0377221713000908).

Szablon programu:

- Klasa City – każde miasto ma współrzedne i numer (w celu odróżnienia, traktowany jako nazwa miasta).
- Klasa Tour – przedstawia każdą podróż. Jeśli nie jest dzieckiem, to pobiera miasta i losuje liczbę miast odwiedzaną przez każdego sprzedawcę.
  Zawiera funkcje zmieniające kolejność miast i liczby odwiedzanych miast przez sprzedawców (SWAPS_NUMBER razy), żeby każda nowa podróż miała jak największą szansę na unikalną kolejność elementów w wektorach cities oraz salesmen.
- Klasa Population – każda populacja składa się z POPULATION_SIZE podróży. Jeśli jest to nowa populacja (bool newPop) to doda ona stworzone podróże do wektora tours. Dodatkowo, zawiera zmienne pmutation i nmutation, wykorzystywane potem do mutacji.
- Klasa GA – implementacja algorytmu genetycznego. Funkcje:
  - Select - wybiera rodziców z populacji, opiera się na implementacji ruletkowej (Goldberg, 1989).
  - Flip – ustala na podstawie pmutation czy nastąpi mutacja.
  - Mutation – wraz ze wzrostem liczby wystąpień mutacji w przeszłości, zmniejsza się prawdopodobieństwo na wystąpienie kolejnej (Goldberg, 1989). Jeśli algorytm uzna, że mutacja powinna nastąpić, zamienia losowo miasta w konkretnej podróży.
  - Tcx – dokładne działanie opisane w załączonym artykule.
  - Evolution – po wybraniu rodziców (select) i stworzeniu dzieci (tcx), poddaje je mutacji (mutation), a następnie losowo wybiera w populacji podróże, które zostaną usunięte, a na ich miejscu będą dzieci. Reszta populacji pozostaje taka sama – tzw. Stady-State GA (DeJong, 1975).
  - Optimization – wywołuje evolution ITERATIONS razy.

Dodatkowe założenia:

- Distance w klasie Tour to suma dróg odbytych przez wszystkich sprzedawców.
- Wartość pmutation (procemtowo prawdopodobieństwo wystąpienia mutacji) jest ustalana w następujący sposób:
  (nmutation -> pmutation)
  0 -> 20
  1, 2 - > 17
  3, 4, 5 - > 13
  6, ..., 10 -> 10
  11, ... -> 5
  Przy czym nmutation to liczba mutacji w stu ostatnich iteracjach.
- Standard_crossover – losową ilość miast z jednego rodzica przepisuje w takiej samej kolejności do dziecka, następnie miasta, które nie zostały przepisane, są brane z drugiego rodzica (na wzór krzyżowania jednopunktowego). Losowo wybiera rodzica, od którego odziedziczy drugą część chromosomu.

Wykorzystane biblioteki: std, fstream, vector, algorithm, math.h, random.

W tabelach prezentowany jest najkrótszy dystans całkowity podróży, tj. suma długości dystansów przebytych przez wszystkich sprzedawców najlepszego osobnika. Z lewej strony początkowego dystansu umieszczone zostały wyniki dla TCX, z prawej dla krzyżowania jednopunktowego.

1. **Zmiana rozmiaru populacji**
   Zakładamy, że im większy rozmiar populacji, tym lepszy wynik uzyskamy.

Miasta = 50
Sprzedawcy = 5

Pop/Iter	100	500	1000	**Początkowy dystans**	100	500	1000
10	20148	18390,3	19123,2	**22615**	20417	17825,7	16844
50	18235,1	14410,4	12911,6	**22252,2**	19291,4	16905,9	15752,9
100	18054	16137,3	14039,1	**21972,4**	20257	18134,5	16157,9
250	20001,5	15284,8	12074	**21398,7**	19677,2	17876,1	17799,6
500	17410,4	15864,1	15864,1	**21398,7**	18900	16905,1	15858,1

Miasta = 50
Sprzedawcy = 15

Pop/Iter	100	500	1000	**Początkowy dystans**	100	500	1000
50	13433,2	10510	10865,6	**16344,8**	13734,4	10484,6	9259,08
100	14791,5	12202,2	10832,1	**14791,5**	13737,6	9045,15	8555,46
250	14449,4	13874,4	10389,5	**14791,5**	14084,5	10491,1	9368,95
500	14791,5	14791,5	12592,6	**14791,5**	12486,2	9651,57	8552,16

TCX: Przy wzroście rozmiaru populacji, jest coraz mniejsza szansa na zabicie najlepszych osobników. Jak widać dla populacji równej 10, wyniki przy wzroście liczby iteracji utrzymują się na podobnym poziomie. Dzieje się tak z powodu dosyć dużego prawdopodobieństwa na zabicie najlepszych osobników (w każdej iteracji zabijane są 2 osobniki, więc w tym przypadku aż 20% populacji).
Dla coraz większych populacji, znacząco szybciej dążymy do optymalnego rozwiązania. Dzieje się tak, bo jest bardzo dużo możliwości tworzenia nowych osobników, a więc i większa szansa na stworzenie lepszych osobników. Jednak zawsze istnieje możliwość zabicia dobrych osobników oraz nie stworzenia lepszych, przez co zwiększenie rozmiaru populacji nie zawsze jest odpowiednim krokiem.

Porównanie: Lepsze wyniki uzyskuje algorytm standardowy. Dla tak dużej populacji, istnieje spora szansa, że mu się „poszczęści" i dojdzie do lepszego rozwiązania niż TCX.

2. Zmiana liczby miast

Zakładamy, że liczba miast nie ma wpływu na działanie algorytmu.

Populacja = 1000
Sprzedawcy = 25

Miasta/Iter	100	500	1000	**Początkowy dystans**	100	500	1000
100	36821,8	27914,1	25259,7	**36821,8**	33309,5	27027,2	24980
150	55953,4	46936,1	45977,2	**57648,4**	53361,9	42841,1	36632,9
250	107830	87003,6	79811,1	**113884**	110527	100981	92242,9
500	222781	208715	186679	**239207**	233943	215668	206579

Populacja = 100
Sprzedawcy = 5

Miasta/Iter	100	500	1000	**Początkowy dystans**	100	500	1000
10	1778,92	1778,92	1521,89	**2373,59**	1966,01	1682,39	1682,39
25	7073,6	6115,59	6737,65	**9877,94**	9837,52	6762,69	6263,47
50	18054	16137,3	14039,1	**21972,4**	20257	18134,5	16157,9
100	40147,4	32158,1	32509,6	**45488,2**	42834,3	39419,5	34421,2

TCX: W każdym wierszu widać, że dla większej liczby iteracji, algorytm TCX działa w ten sam sposób – znajduje krótszą drogę. Liczba miast nie wpływa na działanie algorytmu.

Porównanie: Dla większej liczby miast algorytm TCX zdecydowanie przewyższa algorytm standardowy – losowość wyboru miast w algorytmie standardowym spowalnia proces.

3. Zmiana liczby sprzedawców

Zakładamy, że im bardziej liczba sprzedawców jest zbliżona do liczby miast, tym lepszy wynik uzyskamy.

Populacja = 250
Miasta = 150

Sprzedawcy/Iter	100	500	1000	**Początkowy dystans**	100	500	1000
15	61047,5	50267,9	49155	**64347,3**	57935,8	49066	42801,4
20	54546,8	47076,3	41688,9	**62120,7**	57239,9	50996,6	41821,2
25	55746,1	43620,1	39429,1	**59229,2**	55676,2	46467,9	38449,6
30	47729,1	40652,4	37913,2	**60338,4**	55639,9	42400,5	38156

Populacja = 100
Miasta = 50

Sprzedawcy/Iter	100	500	1000	**Początkowy dystans**	100	500	1000
5	18054	16137,3	14039,1	**21972,4**	20257	18134,5	16157,9
10	15910,3	14108,9	11914,6	**20722,2**	16763,8	15407	12905
15	14791,5	12202,2	10832,1	**14791,5**	13736.6	9045,15	8555,46
20	12799,9	9320,58	7795,86	**16028,1**	12333,4	10565,7	10033

TCX: Można zauważyć, że im większa liczba sprzedawców, tym lepsze wyniki uzyskujemy. Dzieje się tak, bo każdy ze sprzedawców ma coraz mniej miast do odwiedzenia i dzięki temu łatwiej jest znaleźć miasta znajdujące się bliżej siebie.

Porównanie: Brak wpływu liczby sprzedawców na algorytm standardowy – wyniki nie zachowują się monotonicznie, w przeciwieństwie do TCX.

<u>Wnioski:</u>
Zaimplementowany algorytm nie jest algorytmem elitarnym, co można zauważyć przy badaniu jego zachowania dla małego rozmiaru populacji – najlepsze osobniki często zostają zabite.
Przede wszystkim, TCX pozwala zachować różnorodność w drugiej części chromosomu (liczby miast przypisane dla każdego sprzedawcy). Ponadto, zachowuje użyteczne fragmenty pierwszej części chromosomu (kolejność odwiedzanych miast).
Z przeprowadzonych testów wynika, że algorytm TCX jest lepszym wyborem niż algorytm standardowy. Pomimo skomplikowanej implementacji, działa znacznie lepiej. Musimy jednak brać pod uwagę wpływ losowości na algorytm standardowy, dzięki której czasami może mu się udać dotrzeć do optymalnego rozwiązania zdecydowanie szybciej, niż TCX.