Projektowanie Efektywnych Algorytmów

Projekt 3

Implementacja i analiza efektywności algorytmu genetycznego (ewolucyjnego) dla wybranego problemu optymalizacji

Marta Głowacka

234999

1. Wstęp teoretyczny

1.1. Opis elementów

W projekcie zastosowano heurystyczne podejście do problemu TSP (inaczej zwany problemem komiwojażera – jest to zagadnienie minimalizacyjne, NP-trudne, w którym należy znaleźć cykl hamiltonowski o najmniejszej długości; długość między wierzchołkami symbolizują krawędzie o określonych wagach; dokładne omówienie samego TSP jest w sprawozdaniu z Projektu 1).

Heurystyczne podejście oznacza, że niekoniecznie znajdziemy rozwiązanie minimalne, ale w rozsądnym czasie (zadanym przez użytkownika) znajdziemy takie, które leży blisko minimum i również nas satysfakcjonuje.

Projekt polegał na wykorzystaniu programowania genetycznego opartego na procesie ewolucji biologicznej. Definiujemy środowisko, w którym żyje populacja osobników. Każdy osobnik ma określony genotyp (w problemie TSP odpowiada mu permutacja wierzchołków wchodząca w skład trasy, czyli przykładowe rozwiązanie). Funkcja przystosowania, którą w problemach minimalizacji sprowadza się do funkcji maksymalizacji, ocenia jakość rozwiązania. Na jej podstawie – w moim programie – wybierani są najlepsi przedstawiciele populacji, którzy będą wchodzić w skład późniejszego pokolenia.

Osobniki krzyżują się ze sobą, wytwarzając nowe pokolenie. Istnieją różne metody krzyżowania m.in. PMX, OX, PX, SXX. W programie zastosowano krzyżowanie OX.

Cechą algorytmu genetycznego jest wprowadzenie czynnika losowego, jakim są mutacje, czyli losowe zmiany w genotypie osobników.

W czasie powinno dać się zaobserwować powstającą coraz lepszą populację – lepiej przystosowaną do warunków (czyli o krótszych trasach).

1.2. Opis algorytmu

W projekcie przyjęto następujące założenia.

Populacja będzie składała się na początku z k (losowo utworzonych) osobników (dwuwymiarowa tablica int – wiersze to osobniki, w kolumnach są chromosomy).

W pętli przeprowadzane są etapy tworzenia nowej generacji. Liczba iteracji oznacza liczbę generacji pokoleń – zadana jest przez użytkownika.

Z losowo wybranych rodziców powstają dzieci – prawdopodobieństwo, czy nastąpi krzyżowanie jest określone. Zadane jest również prawdopodobieństwo mutacji dziecka. Z

dwóch rodziców powstaje zawsze dwoje dzieci. W rezultacie nowe pokolenie ma 2*k osobników. Wybieramy z nich k najlepszych członków, którzy będą tworzyć już nową populację - selekcja przeprowadzona jest poprzez posortowanie osobników wg funkcji przystosowania i wybranie z nich pierwszych k osobników. Najlepszy członek populacji jest brany do sprawdzenia, czy aby nie jest dotychczasowym najlepszym rozwiązaniem. Jeśli jest, zapisujemy uzyskany koszt drogi i osobnika (permutację), która jemu odpowiada.

2. Opis najważniejszych klas w projekcie

W projekcie zostały wdrożona jedna główna klasa – j TSP_G odpowiadająca za zainicjalizowanie zmiennych potrzebnych do wykonania algorytmu genetycznego oraz jego wykonanie.

Klasa posiada metody prywatne i publiczne, **prywatne** to:

- create_basic_population (generująca losowo początkową populację).
- shuffle (współpracująca z powyższą metodą, miesza chromosomy w genotypie – miasta w permutacji)
- calculate_fitness (obliczenie przystosowania dla osobnika – sprowadzenie problemu do maksymalizacji)
- try_mutate (próba mutacji dziecka)
- transposition_mutation (mutacja transpozycja chromosomów - losowych)
- inversion_mutation (mutacja inwersja chromosomów w podciągu chromosomów)
- calculate_cost (licząca koszt aktualnej drogi).

```
//Metody
void create_basic_population();
void shuffle(std::vector<int> &population_member, int num);
void create_new_generation();
float calculate_fitness(std::vector<int> population_member);

//Mutacja
void try_mutate(std::vector<int> &population_member);
void transposition_mutation(std::vector<int> &permutation);
void inversion_mutation(std::vector<int> &permutation);


int calculate_cost(std::vector<int>& permutation);
```

Publiczne metody:

- solve (rozwiązująca algorytm),
- get_data(inicjalizuje algorytm genetyczny)

```
void get_data(int iterations,     float mutation_probability,int
mutation_manner, int crossover_manner, float crossover_probability);

void solve();
```

Podobnie jak w poprzednim projekcie stworzona została klasa Graph trzymająca informacje o grafie rozumianego jako macierz sąsiedztwa.

Do pomiaru czasu wykorzystano bibliotekę <chrono> oraz metody ze strony: http://antoni.sterna.staff.iiar.pwr.wroc.pl/pea/PEA_time.pdf.

3. Wykresy zależności błędu względnego od czasu i wnioski

Testy zostały przeprowadzone na tych samych instancjach TSP jak w projekcie nr 2: (br17, ftv55, ftv170).

Br17

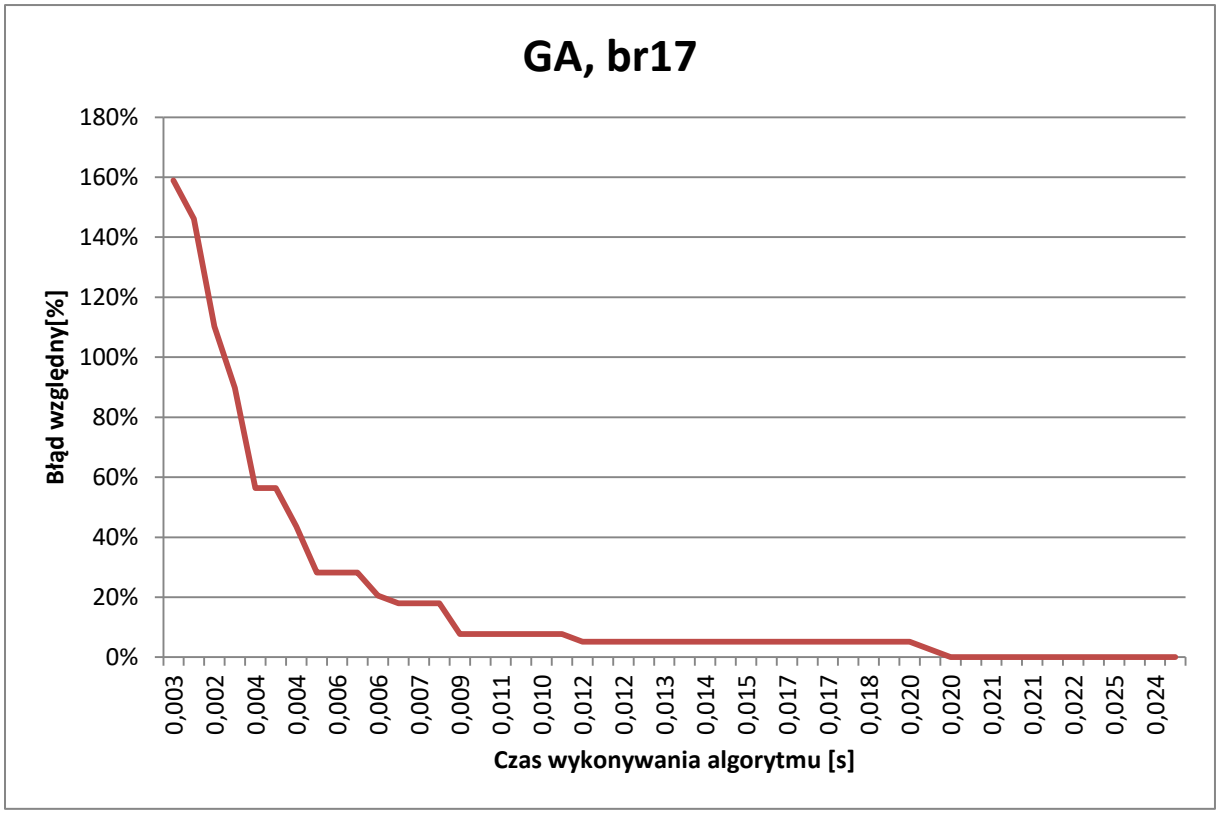
Czas zwiększano, zaczynając od takiego, w którym algorytm nie wykona zadanej liczby iteracji. Wykonano 50 prób – w każdej nieco zwiększano czas, ale tak, aby algorytm nie poradził sobie z zadaniem szybciej niż zadany czas maksymalny. Polepszenie wyniku wiązało się z dłuższym czasem wykonywania algorytmu, a więc wykonania większej liczby iteracji-generacji (im więcej generacji, tym większa szansa na to, że będzie ona ogólnie coraz lepsza, a więc najlepszy osobnik również będzie lepszy).

Stałe parametry jakie dobrano do tego testu to:

```
100, // population size
MAX_INT, // iterations
```

```
0.1, // mutation probability
1,//mutation manner 1 or 0,
1, // crossover manner 1 or 0 (ox only available)
0.9,// crossover  probability
0 //max time
```

Zadając liczbę iteracji jako MAX_INT mamy pewność, że test uzależniony jest przede wszystkim od czasu, a bezpośrednio od innych parametrów.



Test wykonano w 50 próbach.

Algorytm dosyć szybko dochodzi do rozwiązania optymalnego. Gdybyśmy zwiększyli liczbę iteracji, stałoby się to prawdopodobnie jeszcze wcześniej.

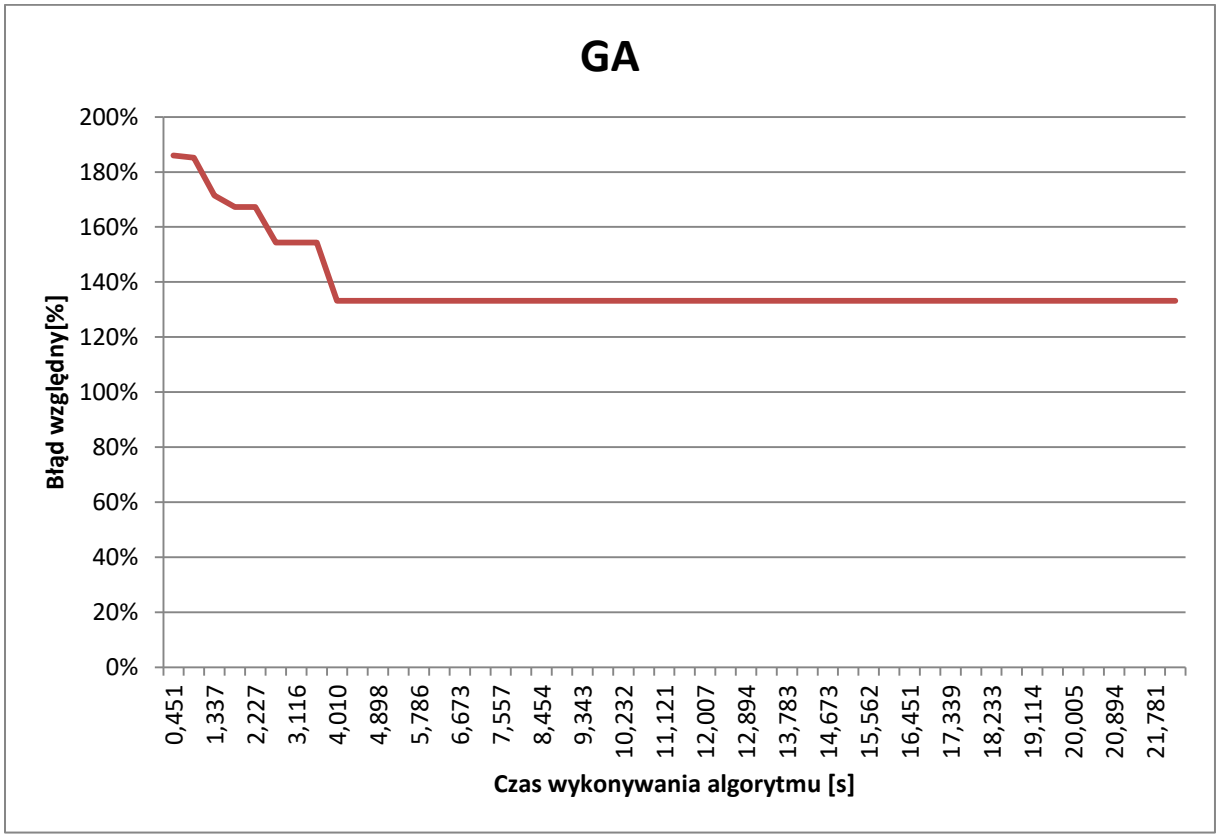
Ftv55

Wykres błędu względnego, choć zbliża się do zera, nie osiąga wartości poniżej 133% procent. Być może ma na to wpływ rozmiar populacji, którą dobrano na początku – 100 osobników było dobrą wartością dla br17, dla instancji ftv55 może okazać się za małą. Może mieć na to wpływ to, że w późniejszym etapie testu zmiany o 0,5 s są za krótkie w porównaniu z początkową fazą i uzyskujemy pod rząd te same wyniki.

Do testów dobrano takie wartości (wszystkie stałe prócz czasu, który jest zwiększany o ok. 0,5 s).

```
100, // population size
INT_MAX, // iterations
0.1, // mutation probability
```

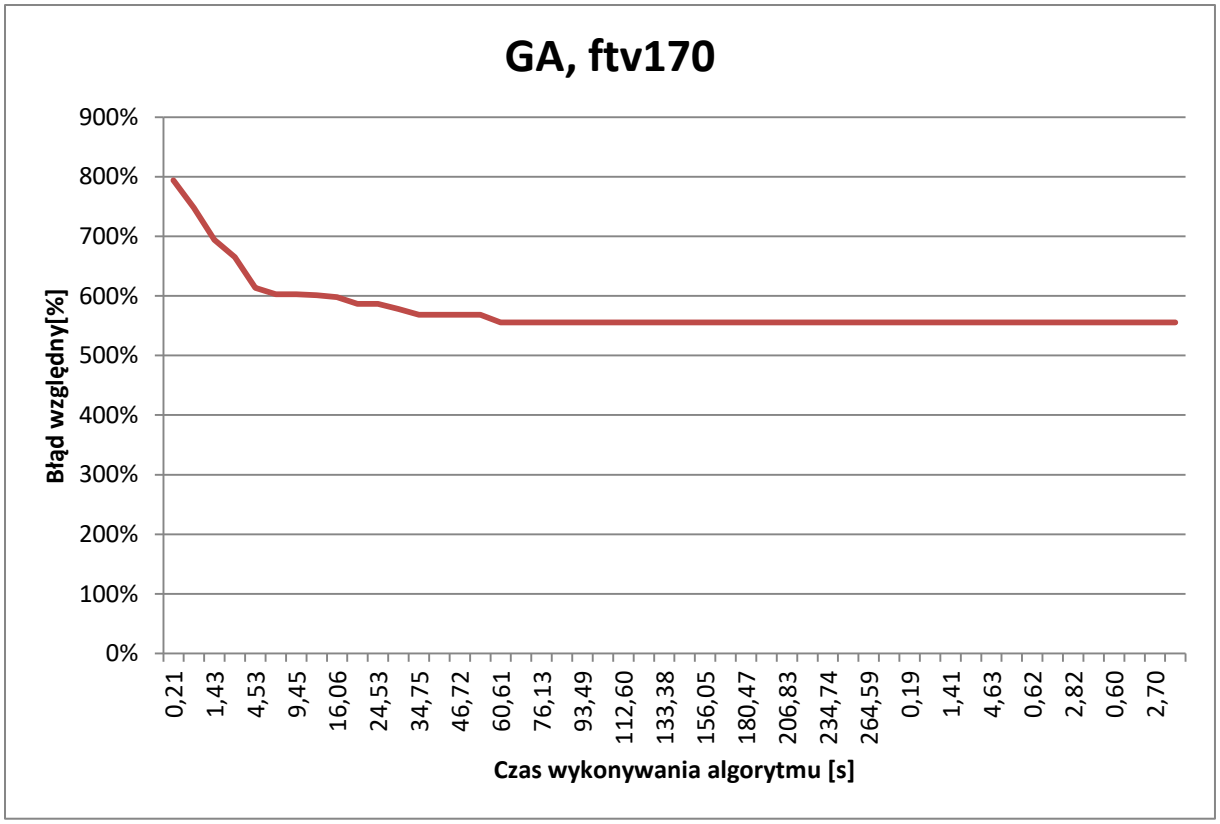
```
1,//mutation manner 1 or 0,
1, // crossover manner 1 or 0 (ox only available)
0.9,// crossover  probability
0}; //max time
```



Ftv170

W ostatnim teście na największej instancji okazało się, że funkcja jest od pewnego momentu const i polepszyć ją może tylko czas bardzo duży. Czas nie był zwiększany liniowo, ale dodawano w każdej iteracji kolejno 1,2,3,4… sekundy. Mimo tego funkcja była const – widocznie trudno algorytmowi było znaleźć w zadanym czasie lepsze rozwiązanie od wstępnie wyszukanego.

Mimo upływu czasu i zwiększających się iteracji, najlepszym osobnikiem pozostawał poprzez generacje wciąż ten sam osobnik. Rozwiązaniem tego problemu mogłoby być wprowadzenie następującej funkcjonalności: krzyżuje się większość osobników w populacji, a resztę zastępujemy losowymi osobnikami. Zwiększałoby to, co prawda losowość, ale pozwalałoby zaczerpnąć nowe dane – bo mutacja w tym przypadku mogła mieć za słaby wpływ - o ile inwersja pojedynczego miasta w br17 może zmienić znacznie rozwiązanie, w ftv170 mutacja ta ma bardzo nikły wpływ.



Najmniejszy błąd jaki udało się uzyskać w tym teście dla ftv170 to 250% (uzasadnienie we wnioskach). Na wykresie najmniejszy widoczny błąd dla tej instancji testu to 556%.

Do testów dobrano takie wartości:

```
100, // population size
INT_MAX, // iterations
0.5, // mutation probability
0,//mutation manner 1 or 0,
1, // crossover manner 1 or 0 (ox only available)
0.9,// crossover  probability
0.001}; //max time
```

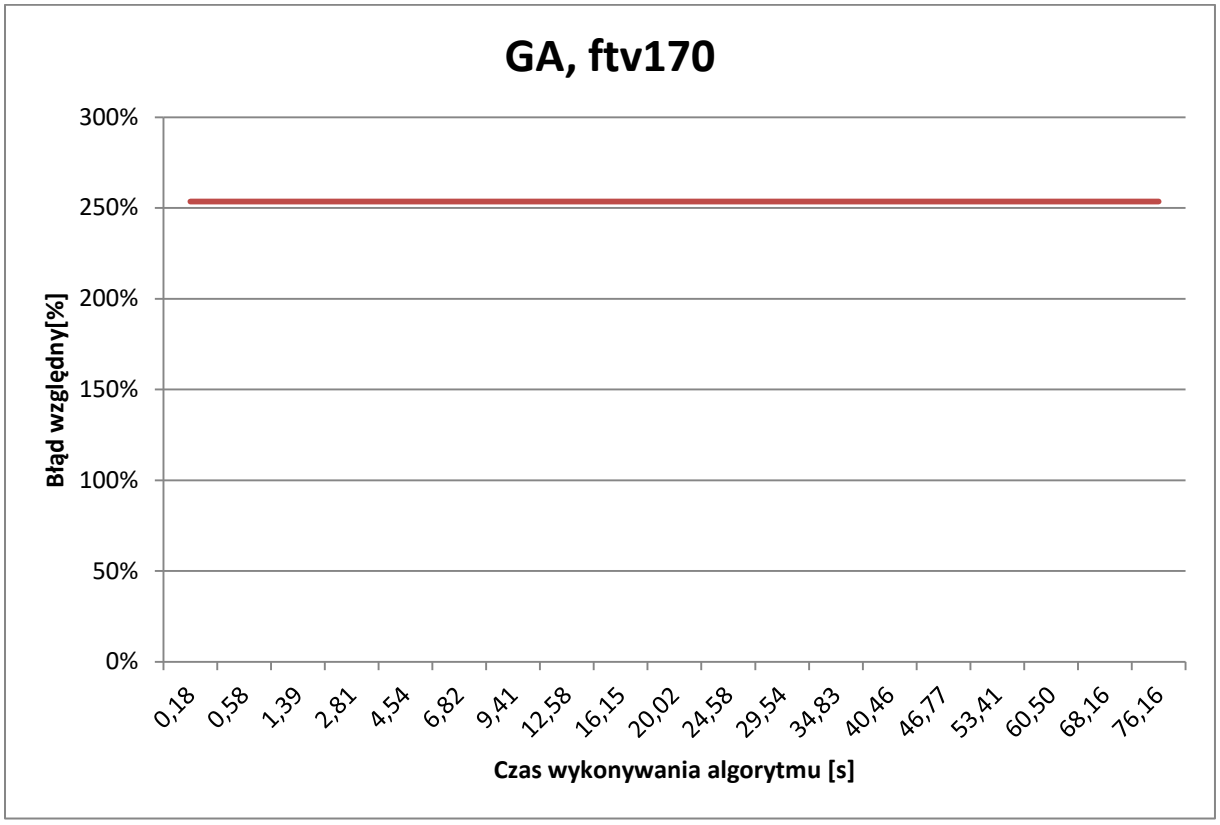
Wnioski

Algorytm dobrze działał dla mniejszych instancji – kilkunastu i kilkudziesięciu wierzchołków. Dla większych instancji dobranie parametrów, które w racjonalnym czasie pokazałyby polepszenie wyników, okazało się być trudne.

Na początku rozpoczęcia każdego algorytmu generowano początkową populację poprzez mieszanie n-krotne permutacji, która jest złożona z kolejnych miast tzn. 0 1,2,3,4… Dla instancji br17 10-krotne przemieszanie wprowadzało dużą różnorodność początkową. Dla ftv50 dziesięciokrotne przemieszanie nie spowodowało aż takiej różnorodności, co może mieć wpływ na constans wykresu od pewnego argumentu (można było wykonać więcej przemieszań). Natomiast dla ftv170 10-krotne przemieszanie początkowe sprawiło, że osobnicy prawie wcale nie byli zróżnicowani – stąd wykres przyjmował wartości constans na całym swoim przedziale. Zwiększenie początkowego przemieszania

do 100 razy (100 krotne losowe zmienienie miejscami dwóch wartości) zmieniło monotoniczność wykresu i można było zaobserwować polepszanie się populacji w czasie. Jednak minimalny błąd jaki osiągnięto wynosił ponad 500%. Przy przemieszaniu 10-krotnym wynosił nieco ponad 250%.

Również wcześniejszy projekt potwierdził przypuszczenie, że dla wybranych przeze mnie instancji korzystne jest rozpoczęcie poszukiwań od permutacji, która ma postać kolejnych miast 0,1,2,3… Czy to będzie osobnik, który znajdzie się w populacji, czy od niego zaczniemy przeszukiwać przestrzeń w metodzie Tabu Search, to kolejne ustawienie miast daje najlepsze minimalne rezultaty.

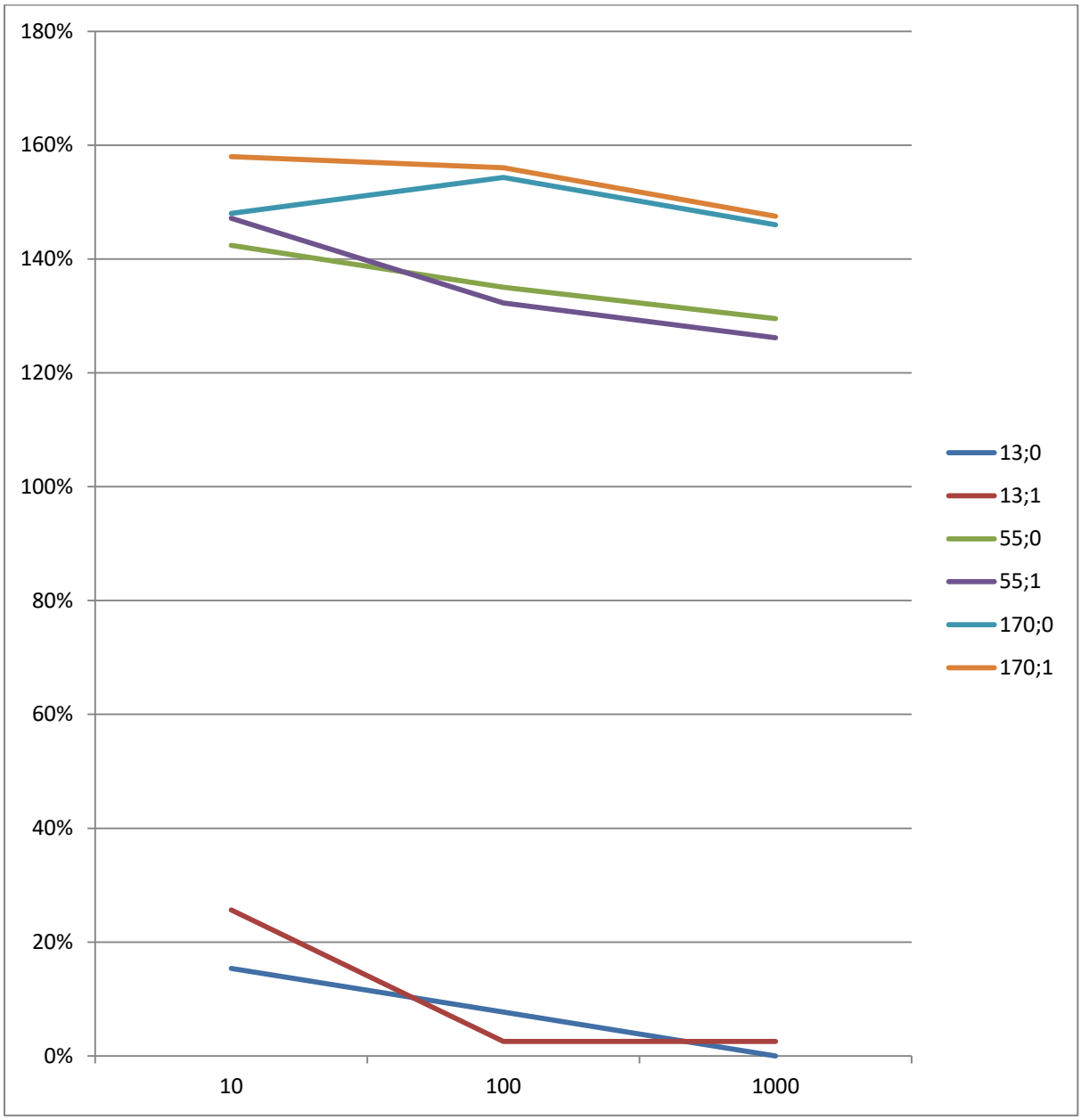


4. Wykresy zależności populacji na wynik i wnioski

Test przeprowadzono dla 10, 100 oraz 1000 osobników. Mimo, że nie sposób było wyeliminować losowe zmiany takie jak mutacje, krzyżowanie i one miały również wpływ na wyniki testu (wielkość populacji nie była jedynym czynnikiem), to wykresy okazały się dość miarodajne. Prawie w każdym przypadku prócz jednego (170;0) wykres nie był rosnący, co ogólnie działa na korzyść populacji o dużym rozmiarze.

Faktycznie, im więcej mamy osobników, tym większe szanse, że któryś z nich będzie w wyniku mutacji lub krzyżowania najlepiej dostosowany do środowiska – czyli w przypadku problemu TSP – będzie posiadał najkrótszą trasę.

Wyraźnie widać, że instancja ftv170 uplasowała się na samej górze, a instancja br13 na samym dole wykresu. Wynika to z tego, że przestrzeń potencjalnych rozwiązań br13 jest względnie mała w stosunku do liczby osobników (prawie dziesięciokrotnie większa) zaproponowanych na osi poziomej. Jako że wszystkie funkcje miały znaleźć się na jednym wykresie, to pójście na taki kompromis był konieczny.

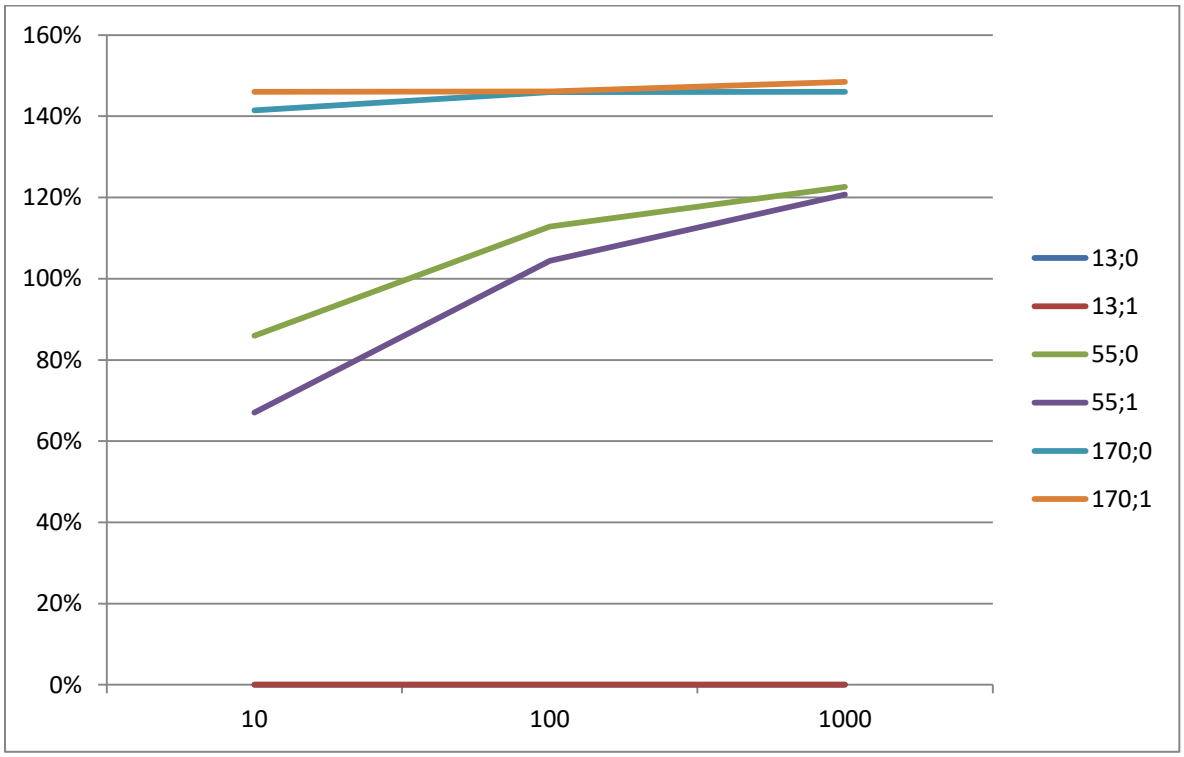


Legenda opisana jest następująco:

- 13 oznacza br13, 55 oznacza ftv55, a 170 ftv170
- 0 po średniku oznacza mutację – transpozycję, a 1 – inwersję

Tabele do wykresu znajdują się na końcu sprawozdania. Wykres dotyczy drugiej tabeli, drugiego rzędu.

5. Wykresy wpływu zależności współczynnika krzyżowania na wynik (przy ustalonym współczynniku mutacji 0,01)



13 br		13 br	
transpozycja (0)		inwersja (1)	
wsp. krzyż.	średni wynik	wsp. krzyż.	średni wynik
0,5	39	0,5	39
0,7	39	0,7	39
0,9	39	0,9	39
ftv55		ftv55	
transpozycja (0)		inwersja (1)	
wsp. krzyż.	średni wynik	wsp. krzyż.	średni wynik
0,5	2990	0,5	2686
0,7	3422	0,7	3287
0,9	3579	0,9	3549
ftv170		ftv170	
transpozycja (0)		inwersja (1)	
wsp. krzyż.	średni wynik	wsp. krzyż.	średni wynik
0,5	6651	0,5	6777
0,7	6776	0,7	6780
0,9	6777	0,9	6846

13 br		13 br	
transpozycja (0)		inwersja (1)	
wsp. krzyż.	średni wynik	wsp. krzyż.	średni wynik
0,5	0%	0,5	0%
0,7	0%	0,7	0%
0,9	0%	0,9	0%
ftv55		ftv55	
transpozycja (0)		inwersja (1)	
wsp. krzyż.	średni wynik	wsp. krzyż.	średni wynik
0,5	86%	0,5	67%
0,7	113%	0,7	104%
0,9	123%	0,9	121%
ftv170		ftv170	
transpozycja (0)		inwersja (1)	
wsp. krzyż.	średni wynik	wsp. krzyż.	średni wynik
0,5	141%	0,5	146%
0,7	146%	0,7	146%
0,9	146%	0,9	148%

Co, nieco zaskakujące, im wyższy był parametr krzyżowania, tym uzyskiwany wynik był gorszy. Testy robiono na 3 próbach (pojedynczy test trwał długo, bo populacja miała 1000 osobników), z których brano medianę do tabelki.

Wyniki między różnymi współczynnikami krzyżowaniami różniły się nieznacznie (np. dla ftv170 kolejno wyszło 6777, 6780, 6846).

Dla instancji br13 zmienianie współczynnika krzyżowania nie miało wpływu, bo populacja 1000 osobników jest aż nadto wystarczająca, by prawidłowo za każdym razem określić wynik.

Dla instancji ftv170 nie widać było zbytniej różnicy, a pojedyncze procenty, jakimi różnią się wyniki można uznać za mało znaczące.

Najbardziej intrygujące jest zachowanie w przypadku ftv55 – widocznie taka liczba osobników (1000) ze współczynnikiem krzyżowania (odpowiednio 0,5 ; 0,7 ; 0,9) powoduje korzystniejsze zachowanie się funkcji, jeśli częściej kopiowani będą rodzice niżby się krzyżowali. Pamiętajmy, że rodzice do krzyżowania wybierani są losowo. Częste skrzyżowania może zanadto wprowadzić czynnik losowy i w efekcie nie uzyskamy dążenia do prawidłowego wyniku, ale chaotyczny rozrzut. To tylko hipoteza, bo podłoże takiego zachowania może być inne.

Porównanie z Tabu Search (najlepszy wynik): dla instancji br13 oba algorytmy poradziły sobie dobrze i błąd względny wynosił 0%. Dla instancji ftv55 Tabu Search miał najlepszy błąd względny ok. 40-50% przy ftv55 (133% dla GA) i ok. 70-90% przy instancji ftv170. (ok. 250% dla GA). Algorytm Tabu Serach radził sobie nieco lepiej.

Bibliografia

http://www.zio.iiar.pwr.wroc.pl/pea/w9_ga_tsp.pdf

https://pl.wikipedia.org/wiki/Algorytm_genetyczny

http://riad.pk.edu.pl/~szymonl/nauka/KKMU2006.pdf

http://aragorn.pb.bialystok.pl/~wkwedlo/EA13.pdf

http://mat.uab.cat/~alseda/MasterOpt/GeneticOperations.pdf

https://www.youtube.com/watch?v=3GAfjE_ChRI

http://www.imio.polsl.pl/Dopobrania/Cw%20MH%2007%20(TSP).pdf

http://aragorn.pb.bialystok.pl/~wkwedlo/EA5.pdf

http://www.zio.iiar.pwr.wroc.pl/pea/w9_ga_tsp.pdf

instancja tsp	13 br		13 br		ftv55		ftv56		ftv170		ftv170	
metoda mutacji	transpozycja (0)		inwersja (1)		transpozycja (0)		inwersja (1)		transpozycja (0)		inwersja (1)	
dane	koszt	populacja	koszt	populacja	koszt	populacja	koszt	populacja	koszt	populacja	koszt	populacja
	48	10	49	10	4014	10	3974	10	6854	10	7146	10
	40	100	40	100	3875	100	3793	100	6853	100	6875	100
muatcja 0.4	39	1000	39	1000	3755	1000	3782	1000	6838	1000	6854	1000
	45	10	49	10	3898	10	3974	10	6833	10	7107	10
	42	100	40	100	3779	100	3735	100	7007	100	7053	100
mutacja 0.1	39	1000	40	1000	3691	1000	3637	1000	6777	1000	6819	1000

instancja tsp	13 br		13 br		ftv55		ftv56		ftv170		ftv170	
metoda mutacji	transpozycja (0)		inwersja (1)		transpozycja (0)		inwersja (1)		transpozycja (0)		inwersja (1)	
dane	koszt	populacja	koszt	populacja	koszt	populacja	koszt	populacja	koszt	populacja	koszt	populacja
	123%	10	126%	10	250%	10	247%	10	249%	10	259%	10
	103%	100	103%	100	241%	100	236%	100	249%	100	250%	100
muatcja 0.4	100%	1000	100%	1000	234%	1000	235%	1000	248%	1000	249%	1000
	115%	10	126%	10	242%	10	247%	10	248%	10	258%	10
	108%	100	103%	100	235%	100	232%	100	254%	100	256%	100
mutacja 0.1	100%	1000	103%	1000	230%	1000	226%	1000	246%	1000	248%	1000