Sztuczna Inteligencja i inżynieria wiedzy

Sprawozdanie z zadania pierwszego laboratorium

„Implementacja i badanie Algorytmów Ewolucyjnych”

Prowadzący laboratorium:
dr. inż. Paweł Myszkowski

Przygotował:
Aleksander Poławski

1. Wstęp

Celem zadania pierwszego była implementacja metaheurystyki nazywanej algorytmem ewolucyjnym oraz przeprowadzenie badań jej skuteczności. Algorytm należało przystosować do rozwiązywania problemów „Travelling Salesman Problem" (problemów komiwojażera).

Problem komiwojażera jest NP-trudnym zadaniem optymalizacyjnym, którego celem jest wyznaczenie najkrótszej trasy pomiędzy danymi punktami tak, aby w wyznaczonej ścieżce każdy punkt był zawarty dokładnie jeden raz. Punkty często nazywane są miastami, korzystając z analogii do komiwojażera, który musi w jak najszybszym czasie odwiedzić wszystkie z nich.

W celu zrealizowania wytycznych zaimplementowano:

- „loader" ładujący przykładowe dane problemów komiwojażera z przygotowanych wcześniej plików

- algorytm ewolucyjny, z możliwością zmiany parametrów takich jak: liczba iteracji, wielkość populacji

- definicję osobnika reprezentującego rozwiązanie problemu oraz jego funkcję oceny

- dwie operatory krzyżowania osobników: „OX" oraz „PMX", z możliwością regulacji prawdopodobieństwa wystąpienia

- dwie operatory mutacji osobnika: „swap" oraz „inverse", z możliwością regulacji prawdopodobieństwa wystąpienia

- turniej, jako metodę selekcji, z możliwością ustawienia jego rozmiaru

Ponadto w celu uzyskania odniesienia do wyników algorytmu zaimplementowano również inne algorytmy przystosowane do rozwiązania tego samego problemu:

- algorytm zachłanny
- algorytm znajdujący N rozwiązań losowych

W celu zautomatyzowania badań zaimplementowano algorytm zapisujący wyniki poszczególnych eksperymentów do pliku w formacie csv.

Program zawierający wszystkie wymienione składowe napisano w języku C++, aby uzyskać jak najlepszą wydajność.

2. Badanie skuteczności algorytmu ewolucyjnego w odniesieniu do skuteczności algorytmu zachłannego i algorytmu losowego

Badania przeprowadzono na pięciu instancjach problemu:
- berlin52 – instancja łatwa, zawierająca współrzędne 52 miast
- kroA100 – instancja średnia, zawierająca współrzędne 100 miast
- kroA150 – instancja średnia, zawierająca współrzędne 150 miast
- kroA200 – instancja średnia, zawierająca współrzędne 200 miast
- fl417 – instancja trudna, zawierająca współrzędne 417 miast

Wyniki algorytmu losowego oparte są na 10 000 włączeniach tego algorytmu dla każdego problemu.

Algorytm zachłanny jest dla problemu komiwojażera deterministyczny w przypadku kiedy zaczynamy od konkretnego miasta. Z tego powodu dla każdego problemu włączono go dokładnie tyle razy z ilu miast składa się problem, za każdym razem zaczynając ścieżkę od innego miasta (przegląd wszystkich możliwych rozwiązań każdego problemu algorytmem zachłannym).

Po wstępnym strojeniu, algorytm zachłanny ustawiono dla każdego problemu w ten sam sposób, z wyjątkiem liczby iteracji (każdy kolejny problem wymaga więcej czasu, aby uzyskać satysfakcjonujące wyniki). Skorzystano z poniższych ustawień algorytmu:

- wielkość populacji: 100
- metoda selekcji: turniej
- wielkość turnieju: 5% populacji (w tym wypadku 5)
- operator krzyżowania: PMX
- prawdopodobieństwo krzyżowania na poziomie osobnika: 50%
- operator mutacji: inwersja
- prawdopodobieństwo mutacji na poziomie osobnika: 15%

Dla każdego problemu wyniki oparto na dziesięciu włączeniach algorytmu genetycznego.

Wyniki eksperymentu przedstawiono w tabelach na następnej stronie sprawozdania.

Wyniki eksperymentu w postaci tabel:

Instancja	**Wyn. opt.**	**Alg. losowy [10000x]**				**Alg. zachłanny [N]**			
		best	**worst**	**avg**	**std**	**best**	**worst**	**avg**	**std**
berlin52	7542	23403.8	34829.7	29932.7	1607.4	8182.2	10299.4	9373.4	476.2
kroA100	21282	140511.0	199440.9	171087.2	8248.8	24698.5	28694.7	27045.2	816.2
kroA150	26524	215822.8	292799.9	257721.3	10098.4	31482.0	35935.3	33640.0	865.3
kroA200	29368	293587.4	383850.3	340234.7	12031.2	34547.7	42045.1	37468.9	1408.9
fl417	11861	445920.7	547453.2	495910.2	13538.5	13802.4	16722.7	15724.3	494.0

Instancja	**Wyn. opt.**	**Alg. ewolucyjny [10x]**				
		best	**worst**	**avg**	**std**	**iteracje**
berlin52	7542	7933.23	8816.48	8297.76	279.54	500
kroA100	21282	23222.49	25329.41	24111.15	688.77	1500
kroA150	26524	30955.32	33861.91	32123.32	865.05	2000
kroA200	29368	33926.58	36353.27	34839.27	785.11	4000
fl417	11861	13620.5	16040.4	14491.6	692.5	15000

Wnioski:

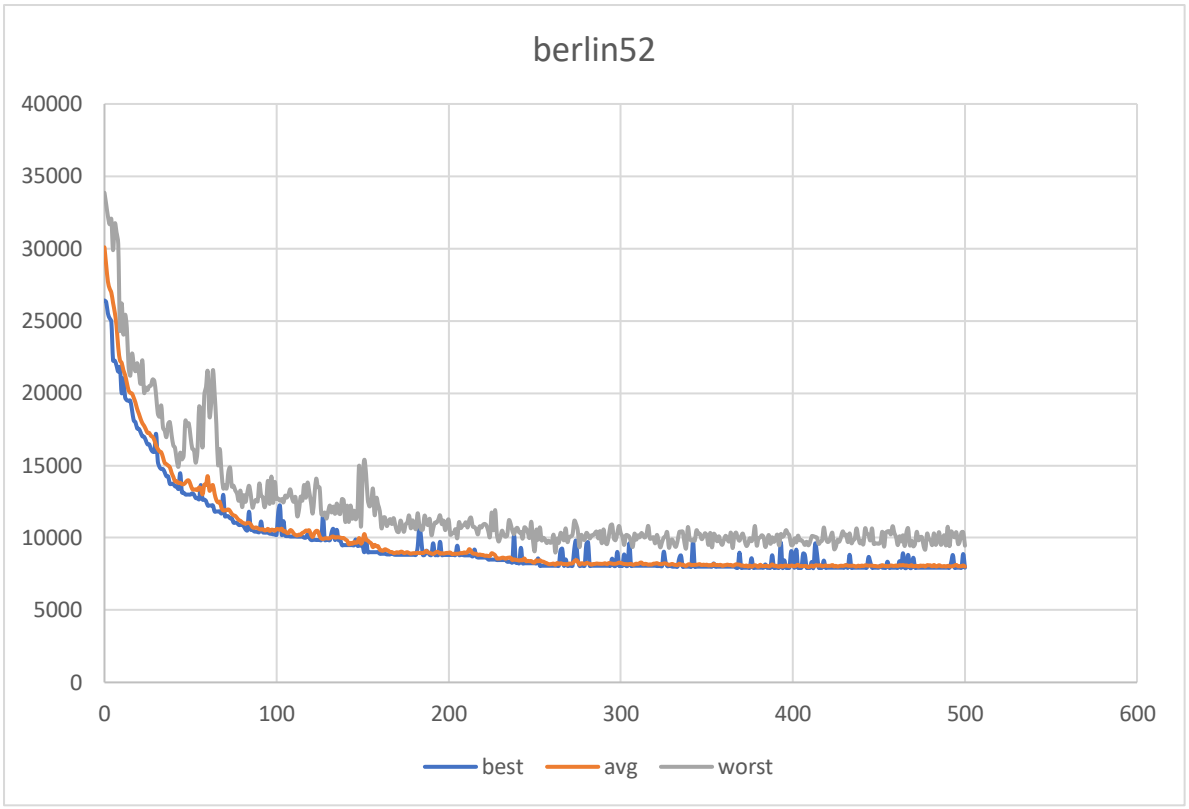
- zgodnie z przewidywaniami algorytm losowy radzi sobie z rozwiązaniem problemu najgorzej, a jego wyniki są najmniej deterministyczne (duże odchylenie standardowe). Znacznie bardziej skuteczny jest algorytm zachłanny, który oferuje rozwiązania o wynikach znacznie bliższych do optymalnych.

- zgodnie z oczekiwaniami dobrze skonfigurowany algorytm genetyczny oferuje najlepsze rozwiązania problemu komiwojażera, przewyższając skutecznością pozostałe algorytmy

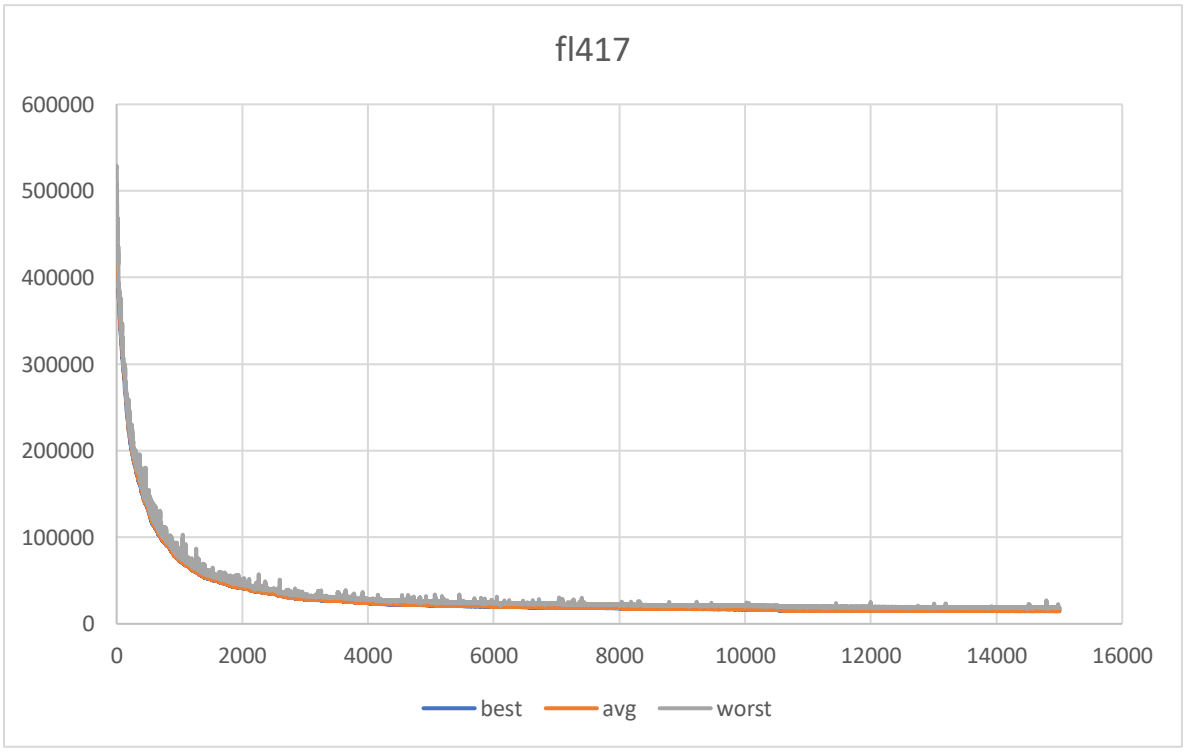
- poszczególne uruchomienia algorytmu genetycznego dla danego problemu dają również (w porównaniu do pozostałych algorytmów) najbardziej zbliżone wyniki (najniższe odchylenie standardowe)

- warto zauważyć, że wyniki otrzymywane algorytmem genetycznym mogłyby być jeszcze bliższe optymalnym po zwiększeniu parametru liczby iteracji (większy czas działania programu), natomiast wyniki algorytmu zachłannego są wynikami ostatecznymi

Wykres przedstawiający działanie algorytmu genetycznego dla jednego z uruchomień dla problemu łatwego „Berlin52”:



Wykres przedstawiający działanie algorytmu genetycznego dla jednego z uruchomień dla problemu trudnego „fl417”:

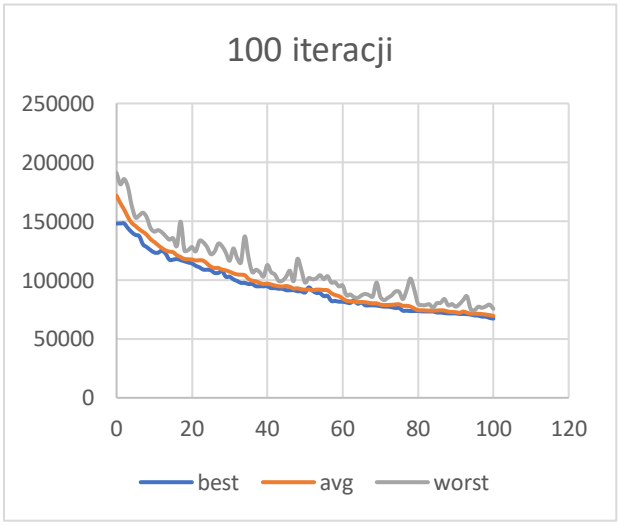


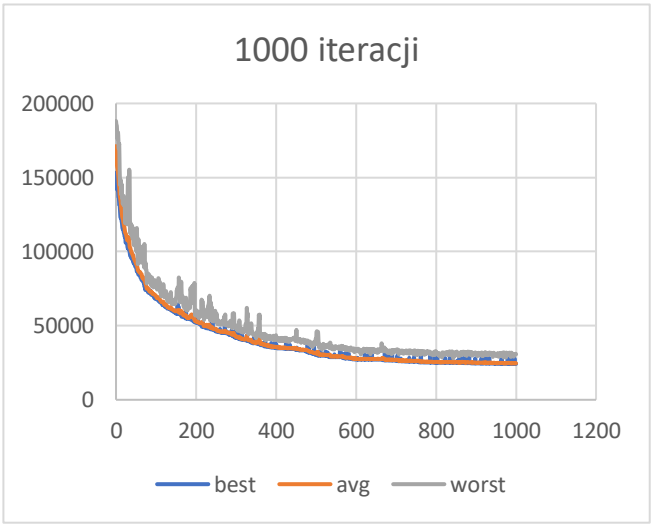
3. **Badanie skuteczności algorytmu ewolucyjnego przy różnych ustawieniach jego parametrów**

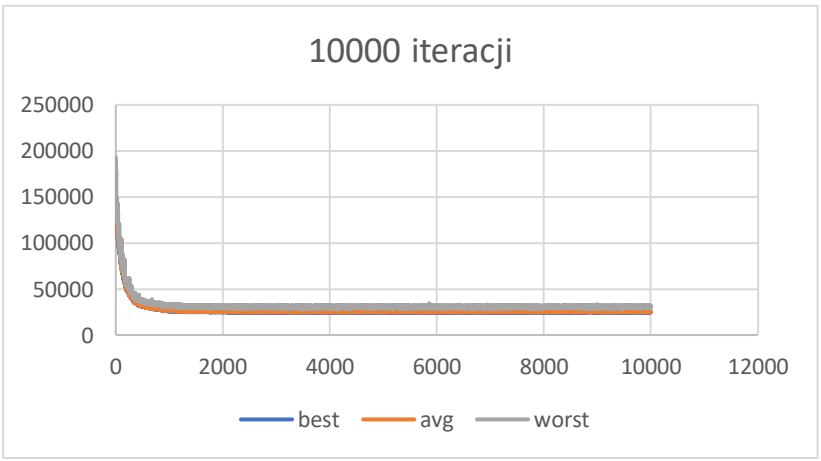
Wszystkie badania przeprowadzono rozwiązując problem o nazwie: kroA100. W przypadku kiedy dane parametry nie był obiektem badań przyjmowały stałą wartość:

- liczba iteracji: 1000
- wielkość populacji: 100
- metoda selekcji: turniej
- wielkość turnieju: 5% populacji (w tym wypadku 5)
- operator krzyżowania: PMX
- prawdopodobieństwo krzyżowania na poziomie osobnika: 50%
- operator mutacji: inwersja
- prawdopodobieństwo mutacji na poziomie osobnika: 15%

a) Liczba iteracji :

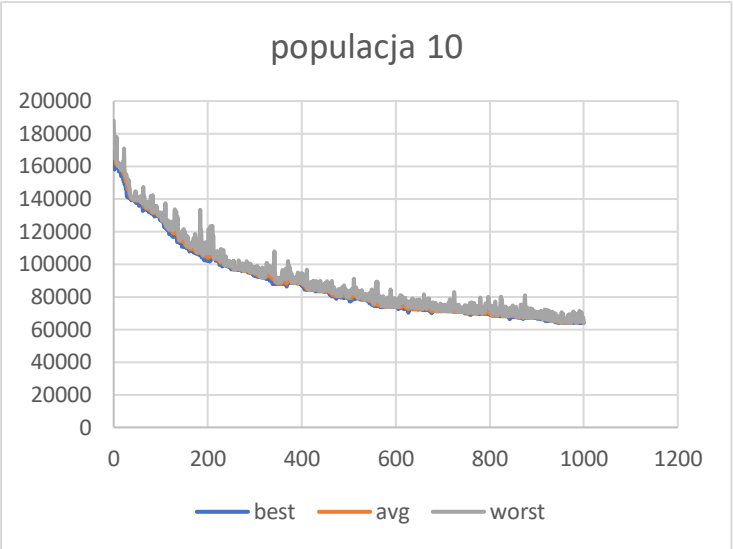


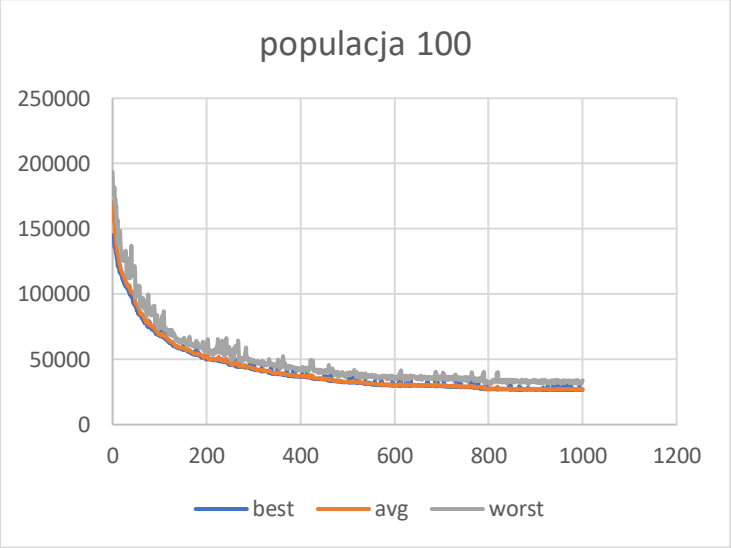


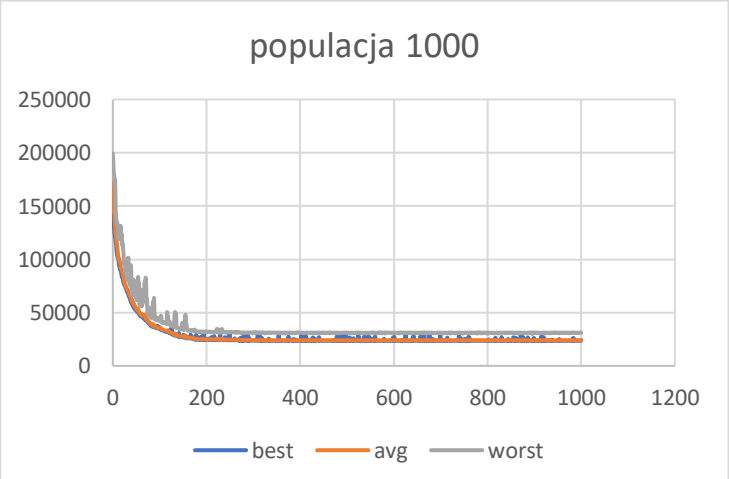


- zbyt mała liczba iteracji powoduje, że algorytm nie zdąży dotrzeć do najlepszych rozwiązań
- zbyt duża liczba iteracji powoduje, że algorytm nie znajduje już lepszych rozwiązań, wydłużając niepotrzebnie czas jego działania

b) **Wielkość populacji:**

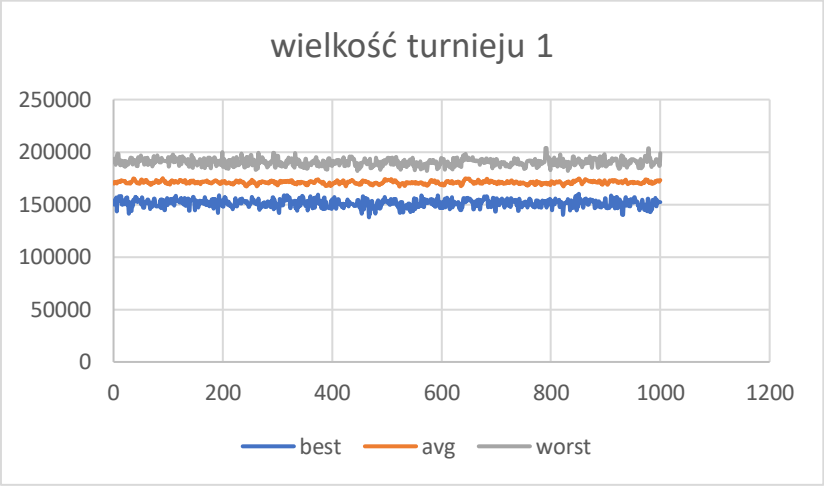


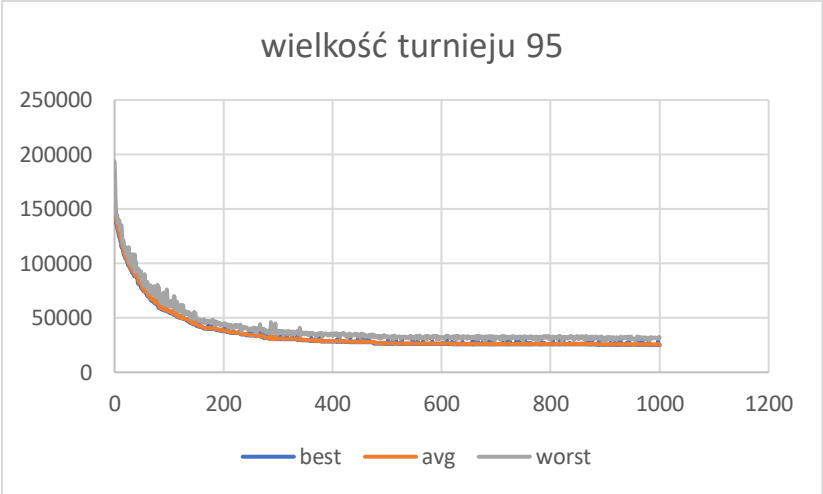


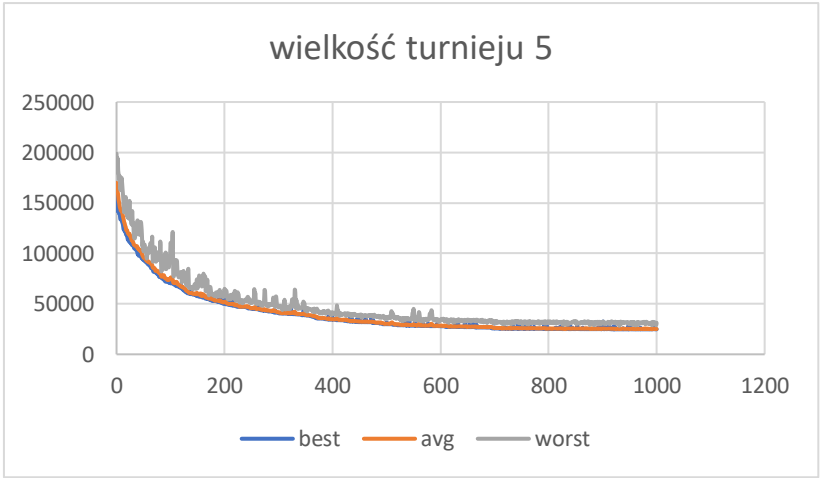


- zbyt mała liczba populacji powoduje, że algorytm wolniej osiąga pożądane wyniki
- zbyt duża liczba populacji wydłuża jednak znacznie czas działania algorytmu

c) **Wielkość turnieju:**

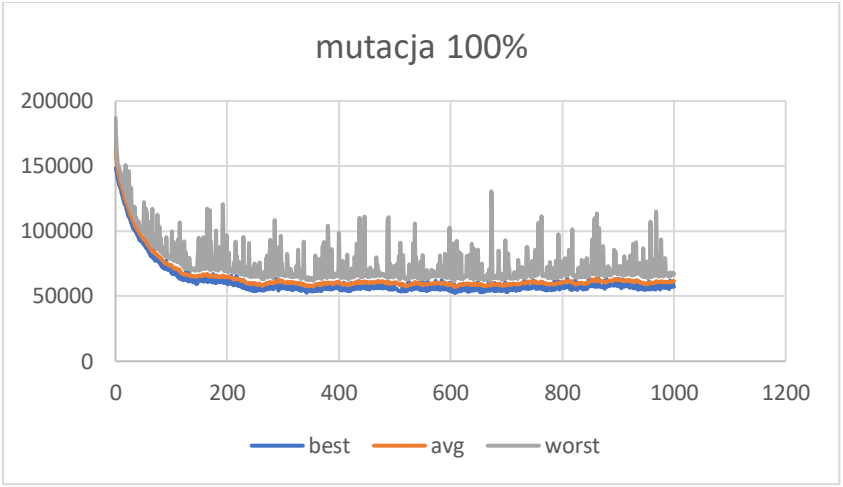


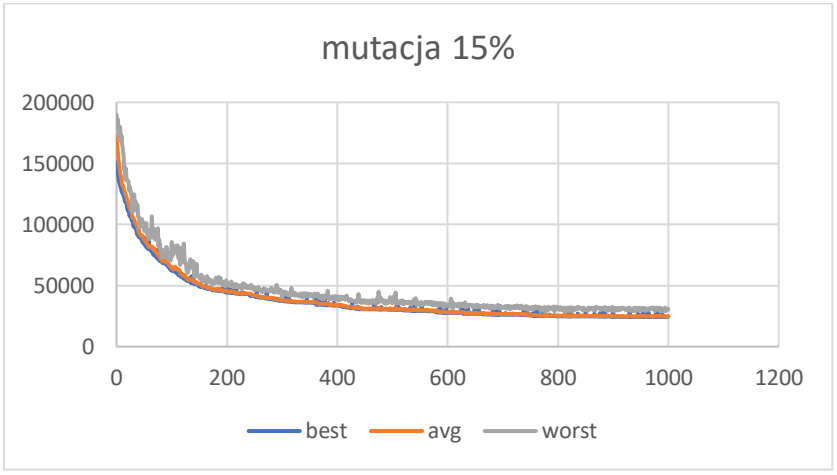




- zbyt mała wielkość turnieju sprawi, że rodzice będą wybierani losowo (do kolejnej populacji przechodzić będą nieprzystosowane osobniki).
- zbyt duża wielkość turnieju sprawi, że do następnej populacji będą wybierane tylko najlepsze osobniki, co zgubi różnorodność i w wyniku doprowadzi do zbyt szybkiego zbiegania algorytmu do lokalnego optimum

d) **Prawdopodobieństwo mutacji:**



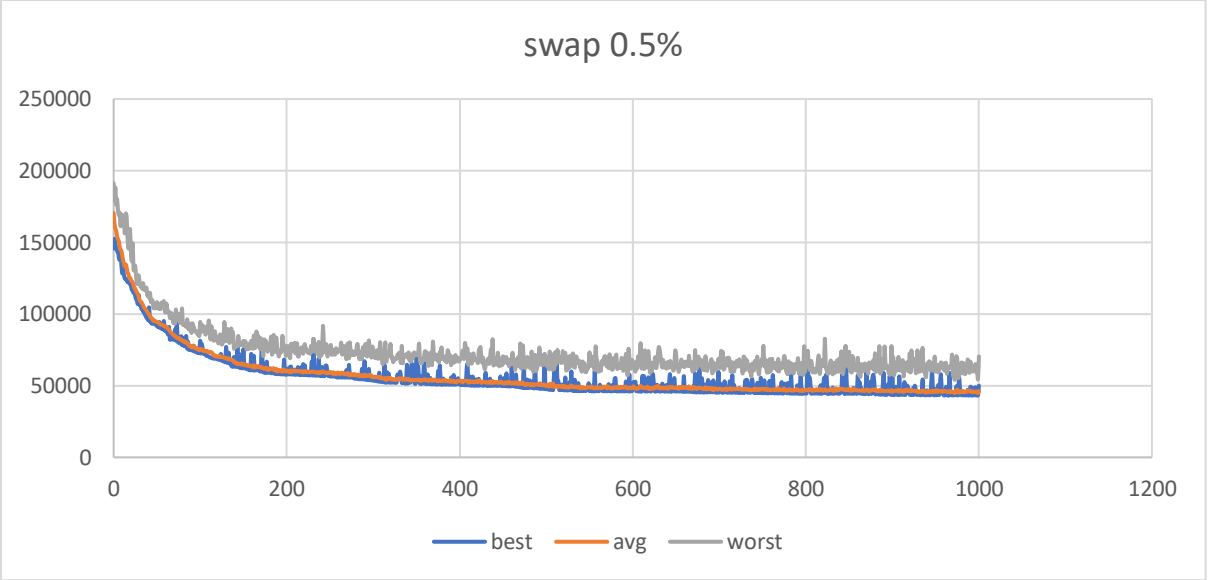


- zbyt mała mutacja sprawi, że algorytm zbiegnie do lokalnego optimum i nie będzie w stanie znaleźć nowych, lepszych rozwiązań

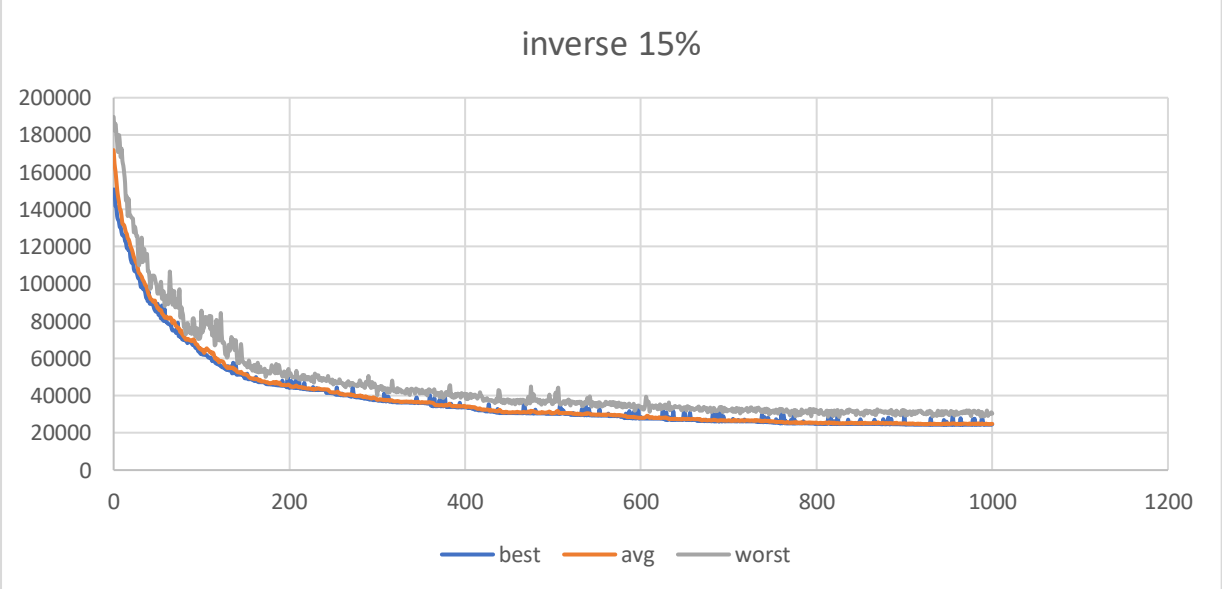
- zbyt duża mutacja sprawi, że algorytm zacznie gubić najlepiej przystosowane osobniki (do następnych populacji przechodzić będą losowe osobniki)

4. **Badanie skuteczności algorytmu ewolucyjnego przy zmianie operatora mutacji**

Badanie algorytmu dla operatora mutacji „swap” o prawdopodobieństwie zajścia 0.5% na poziomie genu (wartość prawdopodobieństwa dobrana na podstawie wcześniejszego strojenia algorytmu)::



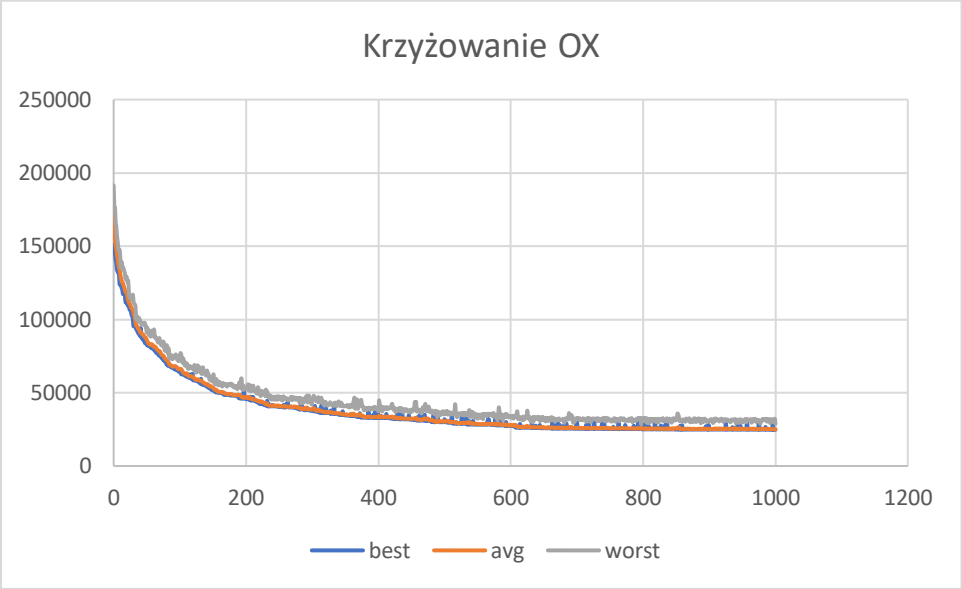
Badanie algorytmu dla operatora mutacji „inverse” o prawdopodobieństwie zajścia 15% na poziomie osobnika (wartość prawdopodobieństwa dobrana na podstawie wcześniejszego strojenia algorytmu):



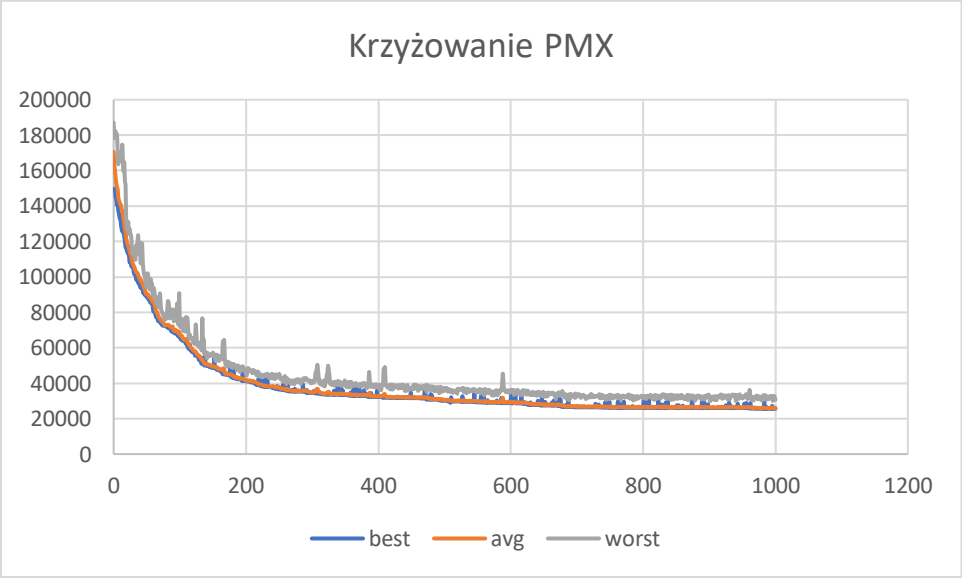
Z badań wynika, że dla opisywanego problemu, znacznie lepiej sprawuje się operator inwersji. Korzystając z operatora „swap”, algorytm wprowadza zbyt dużą losowość i w wyniku traci możliwość znajdywania lepszych rozwiązań.

5. **Badanie skuteczności algorytmu ewolucyjnego przy zmianie operatora krzyżowania**

Badanie algorytmu dla operatora krzyżowania „OX” o prawdopodobieństwie zajścia 50%:



Badanie algorytmu dla operatora krzyżowania „PMX” o prawdopodobieństwie zajścia 50%:



Nie stwierdzono większych różnic w działaniu algorytmu przy zmianie operatora krzyżowania. Obydwa operatory sprawują się dobrze przy rozwiązywaniu problemów TSP.

6. **Prezentacja przykładowego pliku wynikowego zapisywanego po wykonaniu programu**

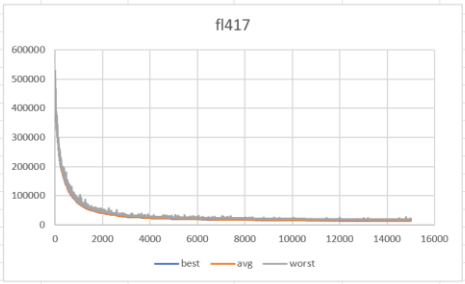
Plik zawiera dane problemu, ustawienia algorytmów oraz wyniki i dane statystyczne dla wszystkich algorytmów użytych przy rozwiązywaniu problemu:

NAME	LEVEL	DIMENSION	OPTIMAL	GREEDY BEST	GREEDY MEAN	GREEDY WORST	GREEDY STANDARD	RANDOM BEST	RANDOM MEAN	RANDOM WORST	RANDOM STANI	RANDOM POPULATI
fl417	Hard	417	11861	13802.38	15724.26	16722.67	494.07	445920.74	495910.24	547453.21	13538.52	10000

GENETIC							
ITERATIOI	POPULATION SI	SELECTION TYPE	TOURNAM	CROSSING TYPE	CROSSING PRO	MUTATION TYPE	MUTATION PROB ‰
15000	100	tournament	5	PMX	500	inverse	150

LAST BEST	LAST AVG	LAST WORST
14691.99	15052.85	18879.76

BEST BEST	MEAN BEST	WORST BEST	STANDARD
13620.56	14491.66	16040.45	692.52



BESTS
13620.56
14797.05
13973.8
14823.16
14388.33
14029.05
13640
14912.21
16040.45
14691.99

iteration	best	avg	worst
0	448291.43	492776.57	526101.7
1	452904.94	481001.48	528823.5
2	446818.65	469909.07	504318.9
3	445420.55	465395.03	507749.9
4	444628.88	458221.79	505705.7
5	441647.38	455144.13	493610.6
6	440798.01	452056.14	500501.6
7	434979.65	447862.35	486126.4
8	425685.87	443409.78	485342.9
9	423171.02	438840.87	458261.1
10	423146.2	433910.76	458521
11	422703.97	430325.92	448447.6
12	409761.38	425300.86	442878.5
13	409761.38	423249.53	469206.3

Jedynym elementem pliku, który nie tworzy się automatycznie jest wykres.

7. **Podsumowanie**

Algorytm genetyczny pozwala znaleźć rozwiązania o wysokiej jakości problemów takich jak TSP. Aby działał prawidłowo wymaga on jednak strojenia, które jest procesem złożonym i czasochłonnym.

Dodatkowo, czas trwania algorytmu jest znacznie dłuższy od czasu trwania innych, konwencjonalnych algorytmów. Algorytm ten jest więc przydatny, kiedy nie jesteśmy ograniczeni czasem.

W wielu dziedzinach życia rozwiązania skuteczniejsze nawet nieznacznie od rozwiązań znanych są jednak w stanie przełożyć się w większej skali na ogromne skoki wydajnościowe danego procesu.