Sprawozdanie

Algorytmy Metaheurysyczne - Labolarorium

Radosław Wojtczak, Witold Karaś

1 Wstęp:

1.1 Algorytm Genetyczny

Algorytmy genetyczne są heurystykami, które działają w oparciu o przeszukiwanie przestrzeni rozwiązań w celu wyszukania najlepszego rozwiązania względem zadanego krytermium. Nazwa ów algorytmu pochodzi od sposobu działania, który przypomina znane w przyrodzie zjawisko ewolucji biologicznej.
Algorytm genetyczny rozpoczyna pracę poprzez wybranie pewnej grupy osobników, którą będziemy nazywać **populacją**. Każda osobnik należący do populacji posiada przypisane pewne informacje, które stanowią jego **genotyp**, na podstawie którego tworzony jest **fenotyp**. Genotyp opisuje proponowane rozwiązanie problemu, a fenotyp przedstawia nam, jak dobre jest to rozwiązanie (tu: funkcja celu). Genotyp składa się z **chromosomów**, natomiast chromosomy składają się z **genów**. W naszym przypadku pojedynczym genem będzie jedno miasto. Po wygenerowaniu populacji początkowej algorytm dokonuje szeregu operacji, których zadaniem jest przystowanie osobników do danego środowiska. W naszym przypadku algorytm będzie dążył do redukcji funkcji celu.
Przed rozpoczeciem działania algorytm pobiera od użytkownika szereg informacji, które w znaczy sposób mogą wpłynąć na wydajność jak i ostateczny wynik wyprodukowany przez algorytm. Parametry zależne od użytkownika to:

- Maksymalna liczba iteracji algorytmu (jest to również warunek końcowy działania programu) reprezentowany przez nieujemną liczbę całkowitą
- Współczynnik mutacji- liczba wymierną z zakresu $[0, 1]$, która przedstawia z jaką szansą dany osobnik może ulec mutacji
- Współczynnik selekcji- W zależności od trybu działania programu parametr przyjmuje jedną z dwóch form:

  - Nieujemną liczbę całkowitą w przypadku skorzystania z turnieju. Wtedy ów liczba reprezentuje z ilu uczestników turnieju wybieramy rodziców do dalszego krzyżowania
  - Liczbę wymierną z zakresu $[0, 1]$, która przedstawia jaki procent populacji ulegnie krzyżowaniu. Wybierając liczbę 0.7 przypisujemy operacji krzyżowania 70% najlepszych osobników, 30% najgorszych nie ulegnie ewolucji i zostanie zastąpiona (odcięcie)
  - Liczbę wymierną z zakresu $[0, 1]$, która wykorzysytywana jest w metodzie selekcji typu ruletka, która polega na znomrmalizowaniu współczynnika dopasowania tak, że suma wszystkich wynosi 1 odwrotnie propocjonalnie do długości ścieżki.

- Rozmiar populacji określający ilu osobników wchodzi w skład populacji
- Maksymalna liczba iteracji bez poprawy, której przekroczenie uruchamia specjalną procedurę mającą na celu rozwiązanie problemu stagnacji oraz potencjalnej zbieżności osobników
- Maksymalny wiek- jeśli osobnik osiągnie maksymalny wiek zostanie poddany specjalnej procedurze modyfikującej oraz jego wiek zostanie zresetowany. Ma to na celu zapobieganie stagnacji oraz zbyt szybkiej zbieżności osobników do najstarszego, najlepiej przystosowanego

W trakcie swojego działania algorytm wykonuje następujące kroki:

1. Pierwszym krokiem jest wygenerowanie pierwszej populacji przy użyciu metody **generate population()**
2. Dochodzi do operacji selekcji osobników, która realizowana jest przez metodę **selection()**. W zależności od wyboru użytkownika, selekcja jest przeprowadzana w formie turnieju lub ograniecczenia zbioru populacji do wskazanego procenta, najsłabi osobnicy poza wskazanym procentem są nadpisywani przez najlepszych (tak zwane **Odcięcie**).
3. Przy użyciu metody **crossover()** dochodzi do operacji krzyżowania, która wykorzystując dwóch osobników z obecnej populacji (tak zwanych **rodziców**) generuje nowych dwóch osobników (**dzieci**). Generowanie dzieci zachodzi poprzez wymianę genów między rodzicami z zachowaniem własności cyklu Hamiltona. Z reguły generacja drugiego dziecka przebiega identycznie do wygenerowania drugiego, z jedyną różnicą zamiany kolejności rodziców w argumentach wywowałnia funkcji. Wykorzystane metody krzyżowania to:
   (a) Partially Mapped Crossover (PMX)
   (b) Cycle Crossover (CX)

4. Metoda **mutation()** odpowiada za dokonywanie mutacji osobników populacji ze wskazanym przez użytkownika prawdopodobieństwem. W celu dokonywania mutacji zaimplementowano dwie metody

   (a) Znana metoda z algorytmu two opt - invert, która dokonuje inwersji kawałka drogi między dwoma losowo wybranymi miastami

   (b) RGIBNNM - metoda, która poza invertem wyszukuje najbliższego sąsiada losowego miasta i dokonuje z nim operacji zamiany miejscami. Ów metoda dodatkowo jest wykorzystywana w celu uniknięcia procesu stagnacji

5. Dla każdego osobnika z populacji zostaje dodany wiek. Osobnicy przekraczający maksymalny wiek zostają poddani procesowi "odmłodzenia" przy pomocy metody **unstack()**

6. Po wygenerowaniu nowej populacji znajdujemy najbardziej przystosowanego do warunków zadania osobnika. Jeśli jest on lepszy, niż obecnie znaleziony osobnik to dokonujemy nadpisania i przechodzimy do następnego etapu

7. Opcjonalny etap, który polega na wykonaniu operacji **unstack()** w sytuacji, gdy algorytm wykryje wystąpenie stagnacji

UWAGA: Kroki 2-7 wykonywane są w pętli, której warunkiem końcowym jest liczba iteracji podana przez użytkownika jako jeden z argumentów uruchomienia programu.

Złożoność obliczeniowa:

***UWAGA**: Literą **n** oznaczono rozmiar permutacji, **l**- rozmiar populacji, a **k**- liczbę iteracji algorytmu*

Główna pętla programu znajduje się w metodzie o nazwie **populate()**. Analiza złożoności obliczeniowej zostanie wykonana poprzez analizę poszczególnych metod wykonywanych w trakcie działania głównej pętli. Przyjmujemy, że liczba wykonań pętli **while()** jest wartością stałą.

1. **GeneratePopulation()**- W tej metodzie dochodzi do wygenerowania początkowej populacji. Ze względu na fakt, iż uzyskane wyniki zależą od początkowej populacji, w realizacji programu zdecydowaliśmy się na dwie metody generacji. Drugą z nich jest metoda o nazwie **ImproveAtsp()**. Metoda **Generate Population** generuje liczbę osobników równą rozmiarowi populacji. Rozmiar populacji jest stałą, podaną przez użytkownika. W pętli głównej metody dochodzi do generacji osobników dwoma sposobami. Jeden z nich wykorzystuje wbudowaną funkcję **shuffle()**, drugą natomiast jest wykorzystanie wcześniej zaimplementowanego algorytmu **K-random()** ze stałą liczbą powtórzeń równą *10*. Złożoność obliczeniowa ów metod generacyjnych wynosi odpowiednio: $O(n)$ oraz $O(n)$. Zauważamy więc, iż wygenerowanie

pojedynczego osobnika zależy liniowo od liczby miast w danej instancji. Wygenerowanie wszystkich osobników pierwszej populacji zajmuję więć $O(l * n) = O(n)$,

2. Metoda **ImproveAtsp()** dodatkowo generuje 5% osobników przy wykorzystaniu metody najbliższego sąsiada, której implementacja odbyła się w ramach poprzednich list zadań. Ponadto w celu wymieszania osobników wykorzystujemy metodę **unstack()**, której złożoność wynosi $O(n * logn)$ Wiemy więc, iż złożoność obliczeniowa metody najbliższego sąsiada wynosi $O(n^2 * logn)$, zatem z sumy $O((19 * l/20) * n) + O((l/20) * n^2 * logn) + O((l/20) * n * logn)$ otrzymujemy złożoność obliczeniową na poziomie $O(n^2 * logn)$.

3. **FindBest()** - Metoda znajdująca najlepszego osobnika po funkcji celu wykorzystująca wbudowaną w pythonie funkcję **min()**. Złożoność obliczeniowa jest stała względem wielkości instancji.

4. **Selection()** - Metoda dokonująca selekcji osobników. W zależności od użytej metody otrzymujemy odpowiednie złożoności obliczeniowe:

   - Wykorzystując odcięcie korzystamy z sortowania wcześniej obliczonych funkcji celu, z tego powodu złożoność obliczeniowa tego sposobu wynosi $O(l)$, czyli z perspektywy długości instancji jest równa $O(1)$.
   - Wykorzystując turniej losujemy z populacji daną część osobników, z której następienie wybieramy dwóch najlepszych. Jak powyżej, ze względu na wykonanie sortowania otrzymujemy złożonosć równą $O(k * l) = O(1)$, gdyż ów operację powtarzamy, aż osiągniemy wskazany przez użytkownika rozmiar populacji.

5. **Crossover()** - Metoda dokonująca krzyżowania osobników. Program realizuje krzyżowania przy pomocy dwóch metod:

   - PMX - W tej metodzie po wybraniu odpowiedniego punktu przecięcia dokonywane jest odpowiednie przestawianie genów rodziców w celu uzyskania dwójki potomstwa. Zakładając, że w najgorszym przypadku punkt przecięcia to ostatni gen zauważamy, iż złożoność obliczeniowa tej metody wynosi $O(n)$, gdyż wszystkie operacje w pętli for wykonują się w czasie stałym
   - CX - W tej metodzie wyszukujemy cykle między rodzicami, po czym generujemy potomstwo poprzez odpowiednie przepisanie wybranych cyklów. W najgorszym przypadku jeden z cykli może składać się z wszystkich genów- wtedy złożoność ów metody również wynosi $O(n)$.

Reasumując, metoda krzyżowań działa ze złożonością $O((l/2) * n) = O(l * n)$ ze względu na to, iż krzyżowaniu ulega każdy osobnik z poprzedniej generacji.

6. **Mutation()** - Metoda dokonująca mutacji. Mutacja pojedynczego osobnika w najgorszym przypadku może się odbywać w czasie liniowym względem liczby miast w instancji (jeśli do inverta zostanie wylosowane pierwsze i ostatnie miasto permutacji). Dodatkowo częstotliwość mutacji jest zależna od współczynnika wprowadzonego przez użytkownika. Złożoność obliczeniowa tej metody wynosi $O(l * n) = O(n)$, gdyż w najgorszym przypadku dla każdego osobnika z populacji możemy dokonać mutacji w pojedynczym przebiegu metody.

7. **AddAge()** - Metoda dodająca wiek do każdego członka populacji, która wykonuje się w czasie stałym względem rozmiaru pojedyncznej instancji, poza przypadkiem, gdy osobnik osiągnie maksymalny wiek. Wtedy uruchamiana jest procedura **unstack()**, której złożoność obliczeniowa wynosi $O(n * logn)$

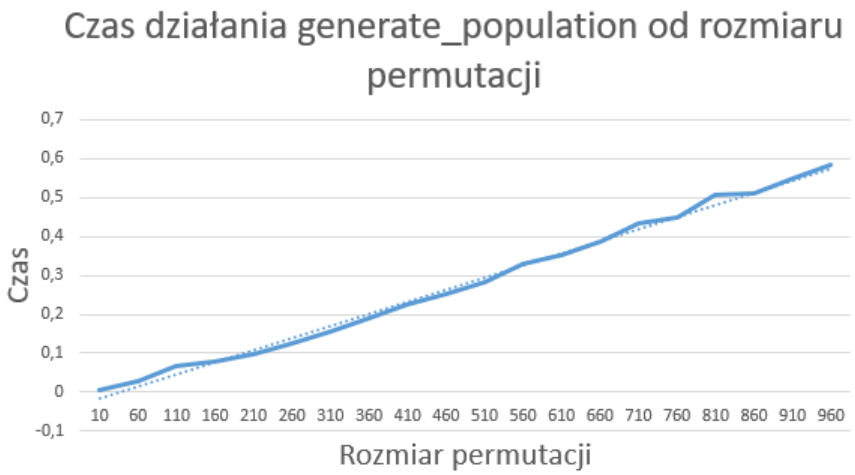
8. **Unstack()** - Metoda wykorzystywana do uniknięcia stagnacji oraz przedwczesnych zbieżności. Ze względu na użycie w niej sortowania otrzymujemy złożność na poziomie $O(l * n * logn) = O(n * logn)$.

Reasumując: Metody oznaczone na liście numerami 3-8 wykonują się w pętli **while()** dokładnie k razy. Policzmy więc złożoność obliczeniową pojedynczego wykonania głównej pętli metody **populate()**.
$O(1)+O(1)+O(l*n)+O(l*n)+O(1)+O(l*n*logn) = O(l*n*logn)$. Dodając liczbę wykonań pętli otrzymujemy złożoność na poziomie $O(k*l*n*logn)$. Biorąc pod uwagę, iż k oraz l to stałe otrzymujemy złożoność na poziomie $O(n * logn)$
UWAGA: Powyższa złożoność zakładałą wykorzystanie metody generacji o nazwie **GeneratePopulation()** oraz metody selekcji typu odcięcie. Wykorzystując metodę generacji **ImproveAtsp()**, która jak sama nazwa wskazuje jest wkorzystywana przy pracy z instancjami asymetrycznymi, złożoność obliczeniowa zmienia się na $O(k * l * n^2 * logn) = O(n^2 * logn)$.
Wykresy: Testy zostały wykonane przy użyciu następujących, stałych parametrów:

- Liczba iteracji: 100
- Współczynnik mutacji: 0.2
- Współczynnik selekcji: 0.7
- Rozmiar populacji: 200
- Maksymalna stagnacja: 10

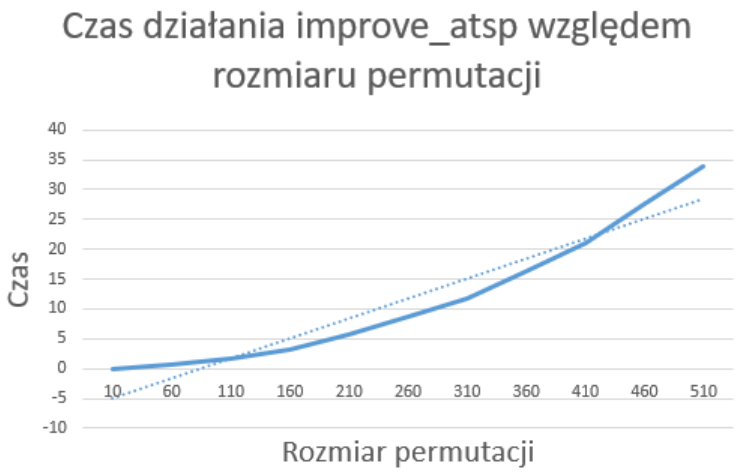
- Maksymalny wiek: 35

Ponadto badane rozmiary pojedynczych permutacji należą do zbioru $n \in \{10, 60, 110, ..., 960\}$ Poniższe wykresy przedstawiają faktyczną złożoność obliczeniową każdej z wyżej wymienionych metod:



Rysunek 1: Zależność czasu działania metody GeneratePopulation od rozmiaru permutacji

Przewidywana złożoność: $O(n)$
Złożoność wynikająca z testów: $O(n)$

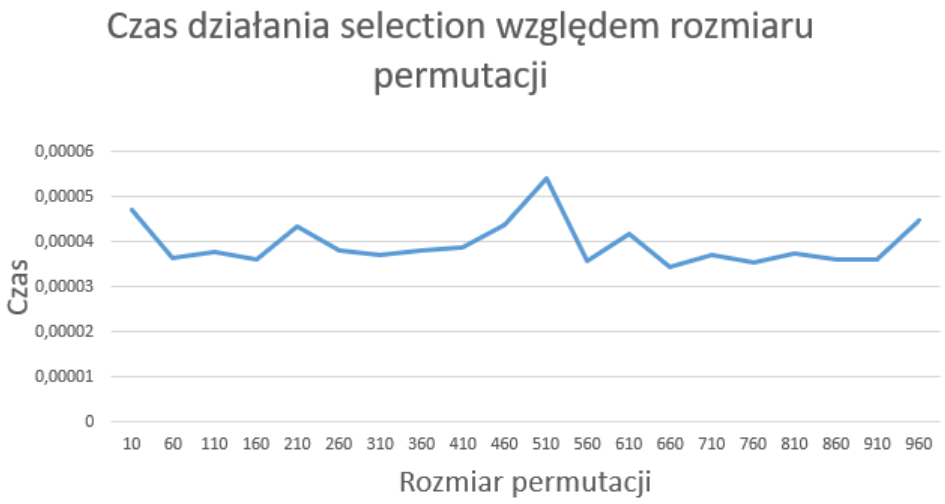


Rysunek 2: Zależność czasu działania metody ImproveAtsp od rozmiaru permutacji

Przewidywana złożoność: $O(n^2 * logn)$
Złożoność wynikająca z testów: Przypominająca $O(n^2)$
Należy zauważyć, iż przeprowadzana analiza jest dla przypadku worst-case, stąd mogą wystąpić rozbieżności między faktyczną, a przewidywaną złożonością.

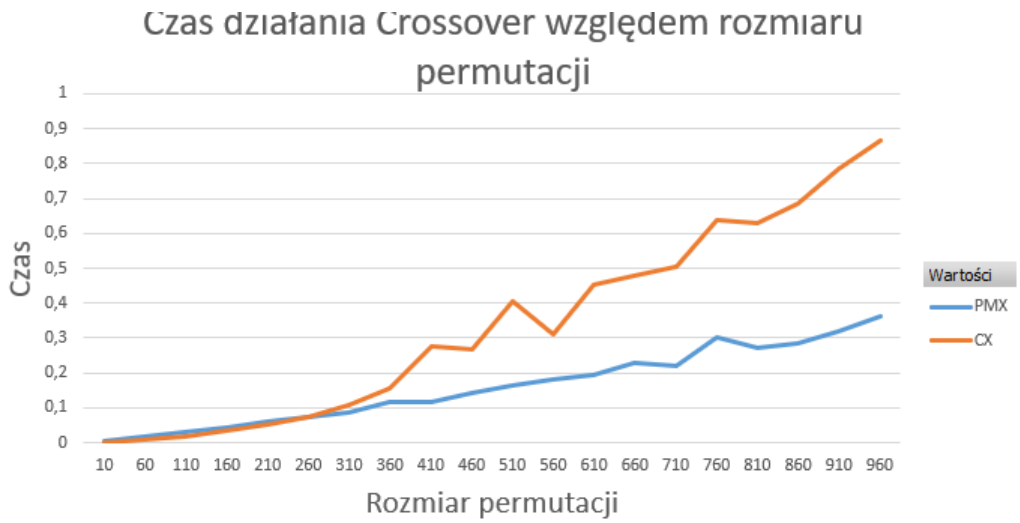
Rysunek 3: Zależność czasu działania metody FindBest od rozmiaru permutacji

Przewidywana złożoność: $O(1)$
Złożoność wynikająca z testów: $O(1)$



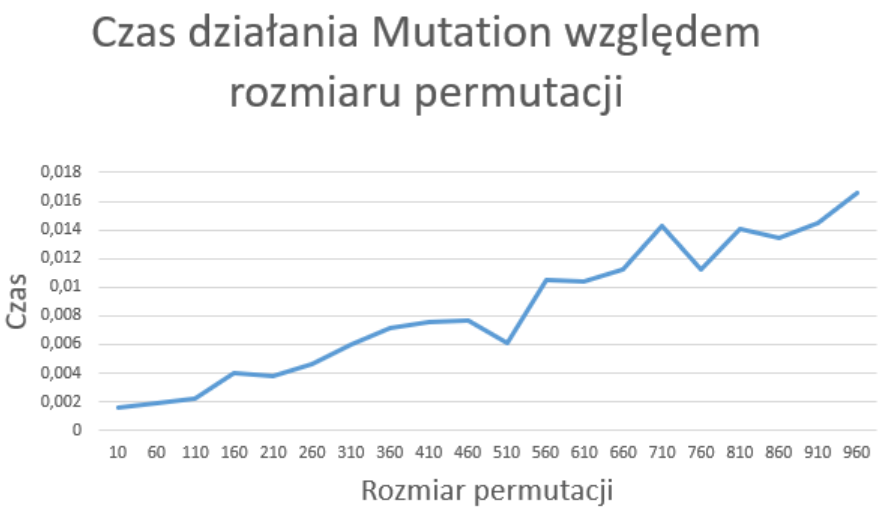
Rysunek 4: Zależność czasu działania metody Selection od rozmiaru permutacji

Przewidywana złożoność: $O(1)$
Złożoność wynikająca z testów: Przypominająca $O(1)$



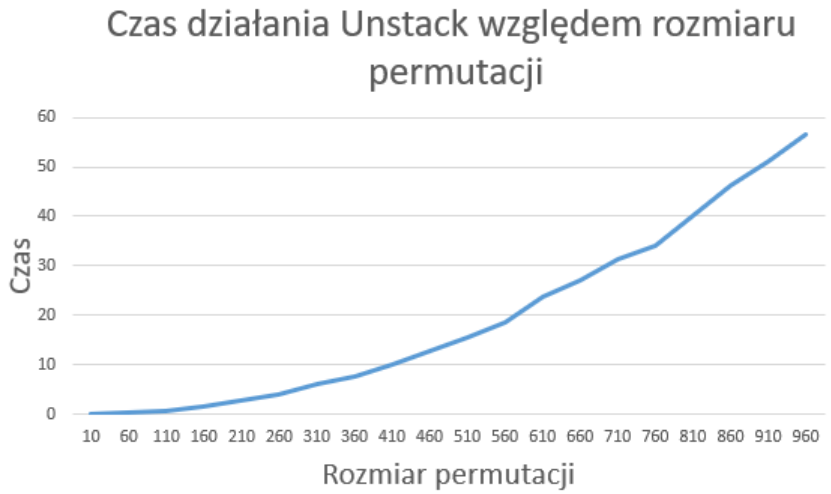
Rysunek 5: Zależność czasu działania metody Crossover od rozmiaru permutacji

Przewidywana złożoność: $O(n)$
Złożoność wynikająca z testów: $O(n)$



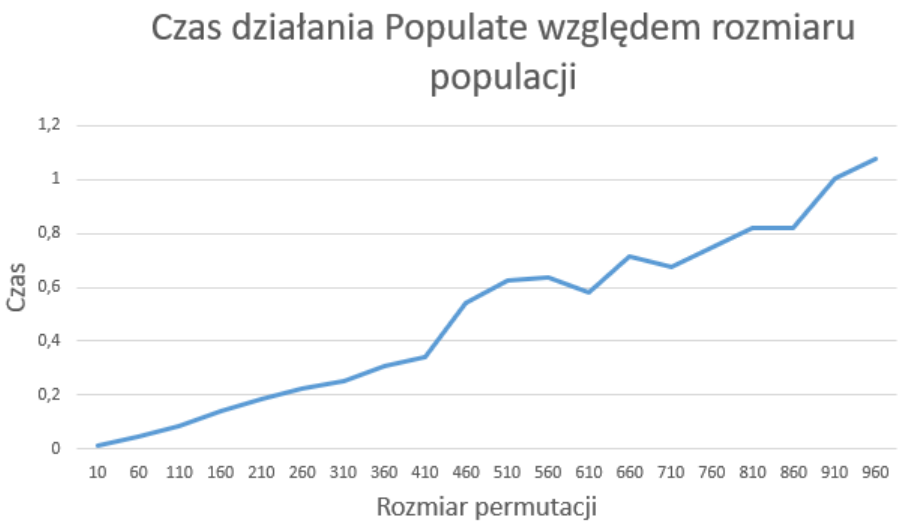
Rysunek 6: Zależność czasu działania metody Mutation od rozmiaru permutacji

Przewidywana złożoność: $O(n)$
Złożoność wynikająca z testów:$O(n)$



Rysunek 7: Zależność czasu działania metody Unstack od rozmiaru permutacji

Przewidywana złożoność: $O(n * logn)$
Złożoność wynikająca z testów: $O(n^2)$



Rysunek 8: Zależność czasu działania metody Populate od rozmiaru permutacji

Przewidywana złożoność: $O(n * logn)$
Złożoność wynikająca z testów: Przypominająca $O(n)$

Wniosek: Zauważamy rozbieżność między teoretycznym wynikiem otrzymanym w wyniku analizy złożoności z praktycznym wynikiem otrzymanym w ramach testów. Wynika to z faktu, iż teoretyczny model zakłada podejście "worst-case", czyli podejście najgorszego przypadku, w którym metoda **Unstack()** wykonuje się w każdej iteracji pętli, jednakże w praktyce ów procedura wykonuje się stosunkowo rzadko. Ze względu na ten fakt zauważamy, iż w większości iteracji możemy pominąć wywołanie tej funkcji, co prowadzi do otrzymania liniowej złożoności obliczeniowej, którą faktycznie obserwujemy w przeprowadzonych testach.

2 Przypadek Testowy 1 - Algorytm genetyczny - zależność PRD od liczby iteracji

2.1 Cel:

W tej części zostaną ze sobą porównane PRD rozwiązania algorytmu genetycznego w zależności od liczby iteracji.

2.2 Założenia:

Do badania tego przypadku została wykorzystana instancje

- berlin52.tsp
- eil76.tsp
- eil51.tsp
- gr17.tsp
- gr21.tsp
- gr24.tsp
- gr48.tsp
- hk48.tsp
- kroA100.tsp
- kroA150.tsp
- kroB100.tsp
- kroB150.tsp
- swiss42.tsp
- ftv33.atsp
- ftv35.atsp
- ftv38.atsp
- ftv44.atsp
- ftv47.atsp
- br17.atsp

Dodatkowo współczynnik mutacji został ustalony na 0.2, współczynnik selekcji na 0.7. Ponad to wielkość populacji została ustalona na 100. Badane iteracje były z zakresu 100, 200, 300, ... 1000.

2.3 Wyniki:

Poniższa tabela przedstawia wyniki testów - średnie PRD (w procentach), odchylenie standardowe oraz błęd standardowy.

Iterations	PRD	SD	SE
100	136.24	143.32	33.78
200	93.97	106.53	25.11
300	68.04	78.38	18.48
400	55.14	66.65	15.71
500	47.74	56.61	13.34
600	40.10	44.34	10.45
700	35.53	37.29	8.79
800	30.60	31.65	7.46
900	29.82	29.68	7.00
1000	31.33	32.55	7.67

Tabela 1: PRD - uśrednione wyniki, SD - odchylenie standardowe, SE - Błąd standardowy

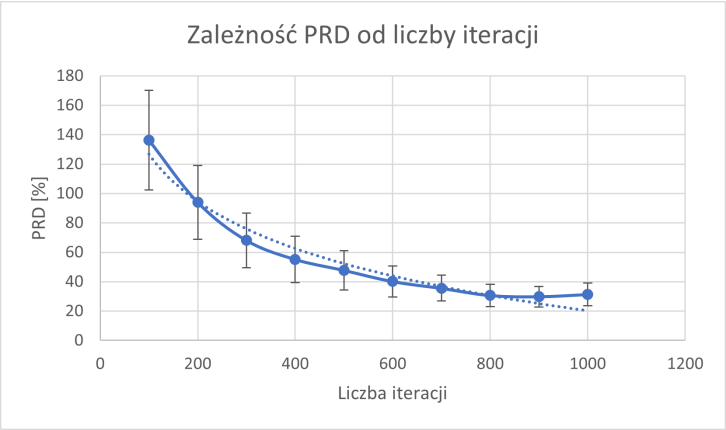
Odchylenie standardowe oraz błąd standardowy zostały obliczone według wzorów:
Odchylenie standardowe:

$$\sigma = \sqrt{\frac{\sum_{n=1}^{18}(\bar{x} - x_n)^2}{18}}$$

Błąd standardowe:

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{18}}$$

2.4 Wykresy:



Rysunek 9: Zależność PRD od liczby iteracji dla różnych wielkości populacji

Na wykresie przedstawione są średnie wartości PRD dla badanych danych. Uśrednione wartości PRD maleją logarytmicznie, wraz ze wzrostem liczby iteracji.

2.5 Wnioski:

Zgodnie z oczyekiwaniami, zwiękaszanie liczby iteracji zwiększa jakość rozwiązań. wartości maleją logarytmicznie od liczby iteracji, zgodnie z zaznaczoną linią trendu. Ponadto wraz ze wzrostem liczby iteracji zmiejsza się odchylenie standardowe oraz błąd standardowy wyników.

3 Przypadek Testowy 2 - Algorytm genetyczny - zależność PRD od wskaźnika mutacji

3.1 Cel:

W tej części zostaną ze sobą porównane PRD rozwiązania algorytmu genetycznego w zależności od wskaźnika mutacji.

3.2 Założenia:

Do badania tego przypadku została wykorzystana instancje

- berlin52.tsp
- eil76.tsp
- eil51.tsp
- gr17.tsp
- gr21.tsp
- gr24.tsp
- gr48.tsp
- hk48.tsp
- kroA100.tsp
- kroA150.tsp
- kroB100.tsp
- kroB150.tsp
- swiss42.tsp
- ftv33.atsp
- ftv35.atsp
- ftv38.atsp
- ftv44.atsp
- ftv47.atsp
- br17.atsp

Dodatkowo współczynnik selekcji został ustalony na 0.7, wielkość populacji została ustalona na 100, liczba iteracji to 200. Badane współczynniki mutacji $m \in 0.05, 0.10, 0.15...0.95$. Metoda selekcji to obcięcie.

3.3 Wyniki:

Poniższa tabela przedstawia wyniki testów, odchylenie standardowe oraz błęd standardowy.

MR	PRD	SD	SE
0.05	119.80	125.64	29.61
0.10	105.52	118.54	27.94
0.15	96.14	109.88	25.90
0.20	88.65	104.24	24.57
0.25	81.35	101.21	23.85
0.30	88.38	102.82	24.24
0.35	83.54	97.42	22.96
0.40	81.49	96.37	22.71
0.45	82.40	100.09	23.59
0.50	88.34	106.14	25.02
0.55	93.55	107.88	25.43
0.60	101.05	113.10	26.66
0.65	100.91	112.06	26.41
0.70	115.50	109.33	25.77
0.75	120.50	110.17	25.97
0.80	129.11	118.01	27.81
0.85	131.69	113.98	26.87
0.90	137.50	115.65	27.26
0.95	142.55	118.34	27.89

Tabela 2: PRD - wyrażone w procętach, SD - Odchylenie standardowe, SE - Błąd standardowy

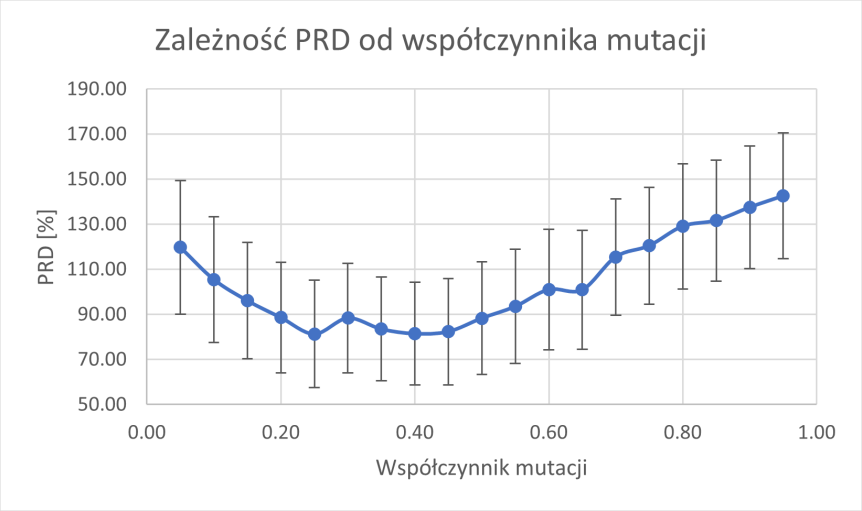
Odchylenie standardowe oraz błąd standardowy zostały obliczone według wzorów:
Odchylenie standardowe:

$$\sigma = \sqrt{\frac{\sum_{n=1}^{18}(\bar{x} - x_n)^2}{18}}$$

Błąd standardowe:

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{18}}$$

3.4 Wykresy:



Rysunek 10: zależność PRD od współczynnika mutacji

Na wykresach przedstawione są średnie wartości PRD dla badanych danych.

3.5 Wnioski:

Dla zebranych wyników, dla współczynnika mutacji w zakresie od 0.00 do 0.25 oraz 0.30 do 0.40 PRD maleje, od 0.4 do 1.00 rośnie. Przy szansie na mutację 0.25 algorytm osiągnął minimum w testach. Po przekroczeniu 0.5 algorytm daje coraz gorsze wyniki co ma związek z częstymi mutacjami - bardziej zaczyna przypominać k-random.

4 Przypadek Testowy 3 - Algorytm genetyczny - zależność PRD od użytej metody selekcji

4.1 Cel:

W tej części zostaną ze sobą porównane PRD rozwiązania algorytmu genetycznego w zależności użytej metody selekcji.

4.2 Założenia:

Do badania tego przypadku została wykorzystana instancje

- berlin52.tsp
- eil76.tsp
- eil51.tsp
- gr17.tsp
- gr21.tsp
- gr24.tsp
- gr48.tsp
- hk48.tsp
- kroA100.tsp
- kroA150.tsp
- kroB100.tsp
- kroB150.tsp
- swiss42.tsp
- ftv33.atsp
- ftv35.atsp
- ftv38.atsp
- ftv44.atsp
- ftv47.atsp
- br17.atsp

Dodatkowo współczynnik mutacji został ustalony na 0.2, liczba iteracji została ustalona na 200, wielkość populacji została ustalona na 100. współczynnik selekcji był w zakresie $SR \in 0.5, 0.55, 0.6...0.9$.

4.3 Metody selekcji:

4.3.1 Ruletka:

Metoda selekcji przedstawicieli populacji polega na znormalizowaniu współczynnika dopasowania tak że suma wszystkich wynosi 1 odwrotnie proporcjonalnie do długości ścieżki. Następnie losowana jest liczba z przedziału $[0, 1)$. Do nowej populacji dopisywana jest jedna ścieżka aż nowa populacja będzie tego samego rozmiaru co poprzednia.

4.3.2 Odcięcie:

Metoda selekcji przedstawicieli populacji polega na posortowaniu populacji względem długości rozwiązania i odcięcie sparametryzowanej najgorszej części (np. 10%). aby uzupełnić populację najlepsze odobniki są doklejane po raz kolejny.

4.4 Wyniki:

Poniższa tabela przedstawia wyniki testów, odchylenie standardowe oraz błęd standardowy. Odchylenie standardowe oraz błąd standardowy zostały

SR	O	R	T	SD_O	SD_R	SD_T	SE_O	SD_R	SD_T
0.5	90.31	274.21	76.47	104.79	207.25	84.46	24.70	48.85	19.91
0.55	85.55	270.70	67.55	100.87	208.68	81.11	23.77	49.19	19.12
0.6	86.40	271.83	74.78	100.70	210.56	83.83	23.74	49.63	19.76
0.65	87.69	269.76	71.43	103.10	211.25	83.36	24.30	49.79	19.65
0.7	91.79	272.51	75.35	106.24	205.00	84.03	25.04	48.32	19.81
0.75	91.51	273.32	74.74	108.68	216.22	85.11	25.62	50.96	20.06
0.8	93.19	264.47	71.91	105.50	204.03	87.26	24.87	48.09	20.57
0.85	91.89	272.87	79.23	108.05	207.47	86.79	25.47	48.90	20.46
0.9	98.22	262.21	72.88	117.56	185.75	83.56	27.71	43.78	19.69
0.95	113.79	263.77	71.11	120.98	199.82	82.98	28.52	47.10	19.56

Tabela 3: SR - współczynnik selecji, O - PRD dla selekcji przez odcięcie, R - PRD dla selekcji przez ruletkę, T - PRD dla selekcji przez turniej, SD_O - odchylenie standardowe dla selekcji przez odcięcie, SD_R - odchylenie standardowe dla selekcji przez ruletkę, SD_T - odchylenie standardowe dla selekcji przez turniej, SE_O - Błąd standardowy dla selekcji przez odcięcie, SE_R - Błąd standardowy dla selekcji przez ruletkę, SE_T - Błąd standardowy dla selekcji przez turniej

obliczone według wzorów:

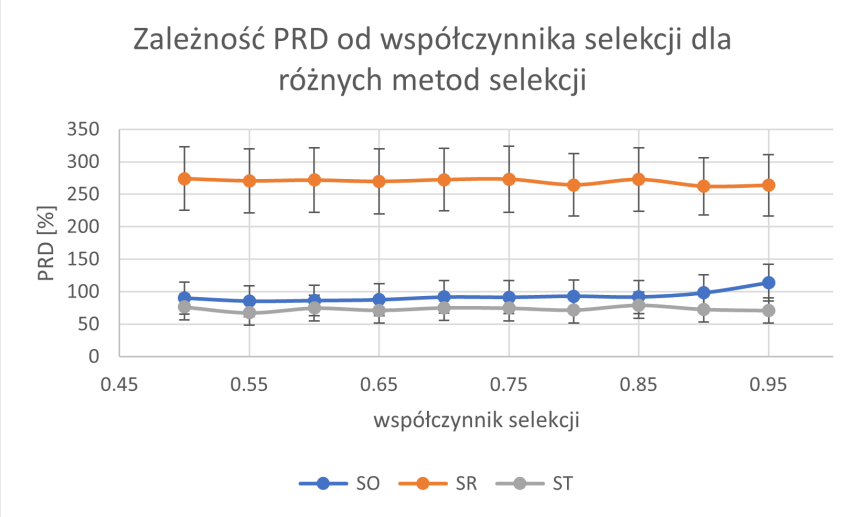
Odchylenie standardowe:

$$\sigma = \sqrt{\frac{\sum_{n=1}^{18}(\bar{x} - x_n)^2}{18}}$$

Błąd standardowe:

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{18}}$$

4.5 Wykresy:



Rysunek 11: zależność PRD od współczynnika selekcji

Na wykresach przedstawione są średnie wartości PRD dla badanych danych.

4.6 Wnioski:

Selekcja przez odcięcie oraz turniejowa sprawdza się zdecydowanie lepiej niż selekcja metodą ruletki. W metodzie ruletki najsłabsze osobniki mają szansę przejść do krzyżowania gdzie w metodzie odcięcia oraz w turnieju są zawsze usuwane i krzyżują się tylko najlepsze cechy.

5 Przypadek Testowy 4 - Algorytm genetyczny - zależność funkcji celu od rozmiar populacji

5.1 Cel:

Celem tego testu jest sprawdzenie w jakim stopniu rozmiar populacji wpływa na wartość funkcji celu. W celu wykonania testu wygenerowano instancje losowe typu **EUC2D** o rozmiarach permutacji należących do zbioru $n \in \{30, 60, 90, 120, 150\}$. Wszystkie inne argumenty zostały zainicjowane stałymi wartościami optymalnymi dla każdej z instancji. Rozmiary populacji należą do zbioru $\{50, 100, 150, 200, 250\}$

5.2 Wyniki:

Wyniki testu przedstawione zostały w poniższej tabeli :

Rozmiar	Funkcja celu
50	1031
100	999
150	910
200	837
250	768

Tabela 4: Wyniki otrzymane dla instancji składającej się z 30 miast

Rozmiar	Funkcja celu
50	2533
100	2089
150	1936
200	1898
250	1868

Tabela 5: Wyniki otrzymane dla instancji składającej się z 60 miast

Rozmiar	Funkcja celu
50	3766
100	3502
150	3429
200	3280
250	3203

Tabela 6: Wyniki otrzymane dla instancji składającej się z 90 miast

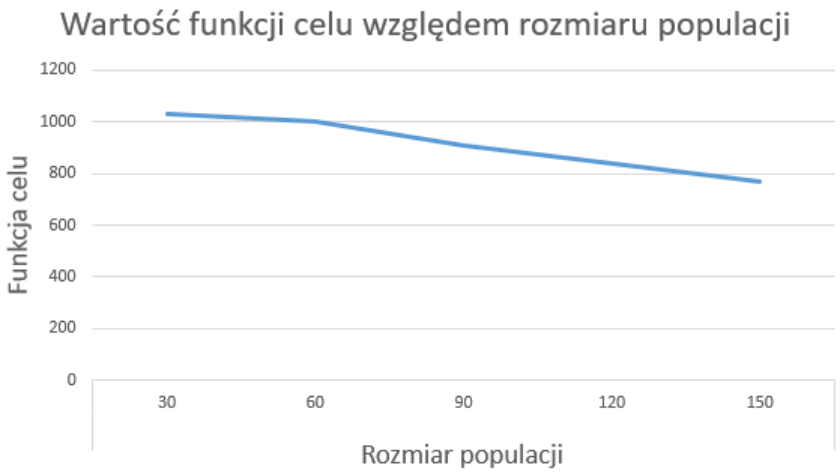
Rozmiar	Funkcja celu
50	5358
100	4952
150	4803
200	4596
250	4468

Tabela 7: Wyniki otrzymane dla instancji składającej się z 120 miast

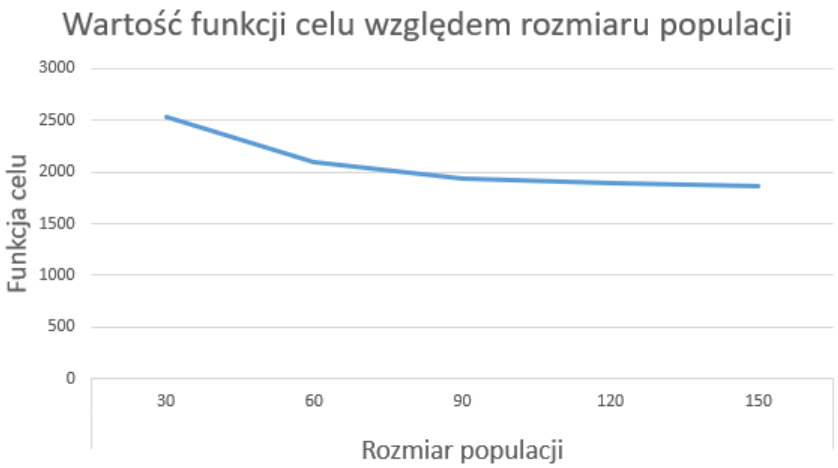
Rozmiar	Funkcja celu
50	7154
100	6504
150	6320
200	6021
250	5933

Tabela 8: Wyniki otrzymane dla instancji składającej się z 150 miast

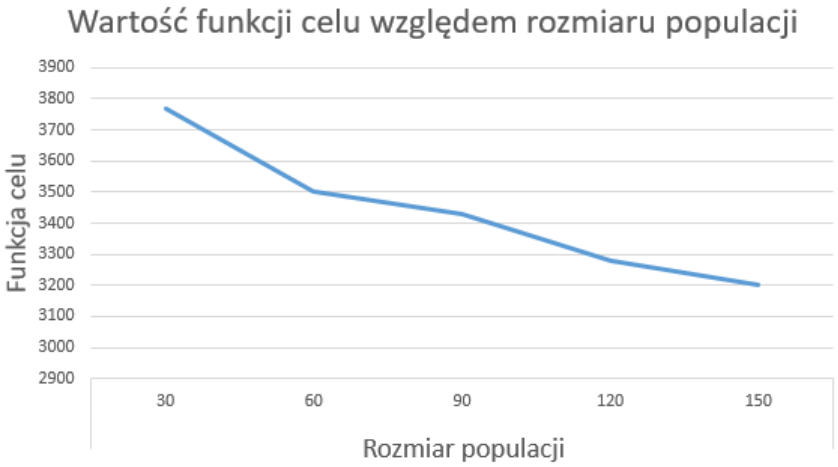
5.3 Wykresy:



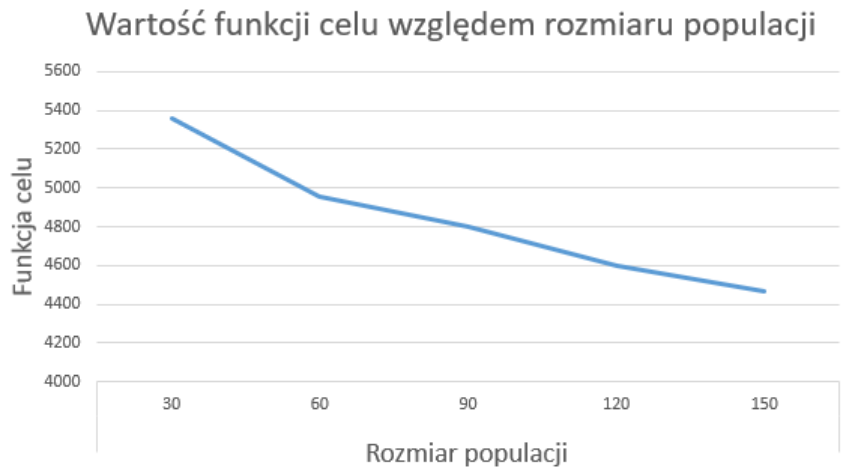
Rysunek 12: Wykres zależności funkcji celu od rozmiaru populacji dla permutacji liczącej 50 miast



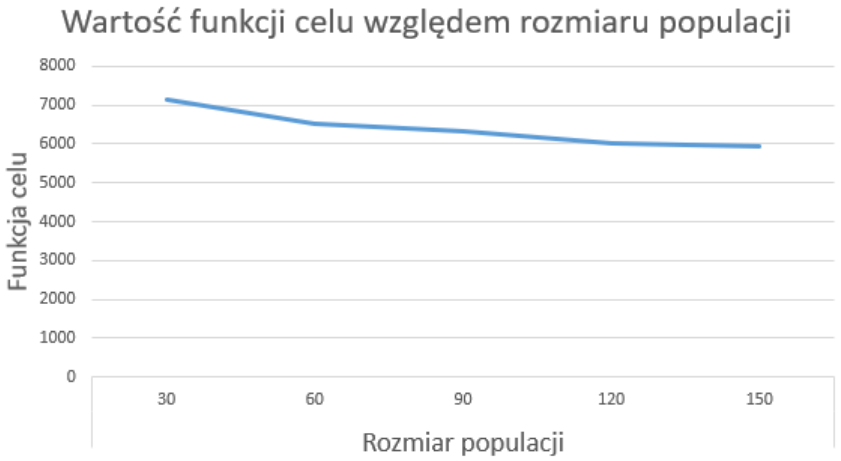
Rysunek 13: Wykres zależności funkcji celu od rozmiaru populacji dla permutacji liczącej 100 miast



Rysunek 14: Wykres zależności funkcji celu od rozmiaru populacji dla permutacji liczącej 150 miast



Rysunek 15: Wykres zależności funkcji celu od rozmiaru populacji dla permutacji liczącej 200 miast



Rysunek 16: Wykres zależności funkcji celu od rozmiaru populacji dla permutacji liczącej 250 miast

5.4 Wnioski:

Zauważamy, iż zwiększenie rozmiaru populacji przy stałym utrzymaniu tych samych wariantów wpływa na uzyskanie lepszych wyników.

6 Przypadek Testowy 5 - Algorytm genetyczny - zależność PRD od wybranej metody krzyżowania

6.1 Cel:

Celem tego testu będzie sprawdzenie, w jakim stopniu otrzymywane wyniki zależą od wybranej metody krzyżowania. W naszej implementacji algorytmu genetycznego znalazły się dwie metody krzyżowania:

- Partially Mapped Crossover (PMX)
- Cycle Crossover (CX)

Testy wykonano na wybranych instancjach z biblioteki TSPLIB:

1. gr48.tsp
2. gr21.tsp
3. berlin52.tsp
4. gr120.tsp
5. br17.atsp
6. ftv47.atsp
7. ftv70.atsp
8. kro124p.atsp

Wszystkie parametry wywołania pozostały niezmienne przez cały okres wykonywania testów.

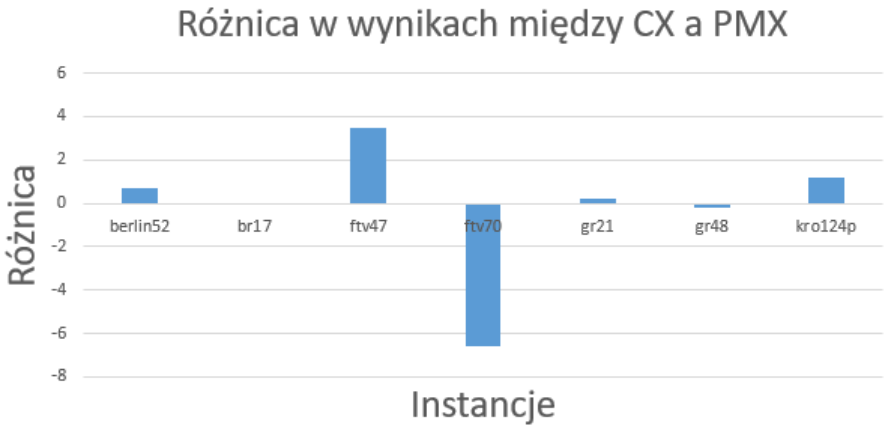
6.2 Wyniki:

UWAGA: W poniższej tabeli kolumna *Różnica* oznacza różnicę między otrzymanymi średnimi wartościami PRD dla testowanych metod krzyżowania. Dodatnia wartość różnicy świadczy o tym, iż w danym teście lepsze wyniki średnio produkowały wywołania wykorzystujące metodę krzyżowania CX, natomiast ujemna różnica świadczy o lepszych wynikach metody PMX.
Poniższa tabela przedstawia uśrednione wyniki testów:

Instancja	PMX	CX	Różnica
gr48.tsp	3.324 %	3.506 %	-0,182 %
gr21.tsp	3.746 %	3.502 %	0,244 %
berlin52.tsp	8,096 %	7,362 %	0,734 %
br17.atsp	0 %	0 %	0 %
ftv47.atsp	39,3 %	35,8 %	3,5 %
ftv70.atsp	25,898 %	32,482 %	-6,584 %
kro124p.atsp	27,184 %	26,008 %	1,176 %

Tabela 9: Tabela zależności otrzymanych wyników PRD od wybranej metody krzyżowania

6.3 Wykresy:



Rysunek 17: Otrzymane różnice dla wskazanych metod krzyżowań

6.4 Wnioski:

Z testów wynika, iż nie można jednoznacznie określić, która z metod krzyżowania osobników jest uniwersalnie lepsza, zauważamy, iż odpowiednie metody radzą sobie lepiej od pozostałych dla odpowiednich przykładów.

7 Przypadek Testowy 6 - Test statystyczny - Porównanie działania GA z 2OPT

7.1 Cel:

W teście statystycznym porównanym działanie algorytmu genetycznego z algorytmem TABU, którego implementacją zajęliśmy się w ramach listy 2. Hipotezą startową jest stwierdzenie, że algorytm genetyczny zwraca lepszą średnią medianę wyników niż algorytm TABU. Do analizy uzyskanych wyników zostanie wykorzystany test Wilcoxon'a dla par obserwacji (one-sided).

7.2 Założenia:

Do wykonania tego testu wykorzystano instancje znajdujące się w bibliotece TSPLIB. Początkowe permutacje są generowane w sposób losowy.

7.3 Wykorzystane instancje:

W tym teście zostały wykorzystane następujące instancje z biblioteki TSPLIB:

1. berlin52.tsp
2. bier127.tsp
3. ch150.tsp
4. eil76.tsp
5. gr120.tsp
6. gr48.tsp
7. hk48.tsp
8. kroA150.tsp
9. kroB100.tsp
10. kroC100.tsp
11. kroD100.tsp
12. pr107.tsp
13. pr124.tsp
14. pr144.tsp
15. pr76.tsp
16. st70.tsp
17. u159.tsp

7.4 Test:

Hipoteza zerowa: GA >= TABU
Hipoteza alternatywna: GA < TABU

Pair	GA	TABU	Abs.Diff	Rank	Sign
6	11941	11976	35	1	-1
15	678	718	40	2	-1
5	5214	5283	69	3	-1
4	7703	7895	192	4	-1
1	8098	8295	197	5	-1
8	23508	23302	206	6	+1
3	7964	7332	632	7	+1
13	64418	63677	741	8	+1
9	21360	22141	781	9	-1
10	23717	25243	1526	10	-1
11	46617	48680	2063	11	-1
12	69279	71421	2142	12	-1
2	132632	130217	2415	13	+1
7	31533	28460	3073	14	+1
16	53914	49123	4791	15	+1
14	109056	118174	9118	16	-1

Tabela 10: Tabela rang dla testu Wilcoxona dla wartości funkcji celu

$$W_- = 1 + 2 + 3 + 4 + 5 + 9 + 10 + 11 + 12 + 16 = 73$$
$$W_+ = 6 + 7 + 8 + 13 + 14 + 15$$
$$min(W_+, W_-) = 63 = T$$

Poniżej powtórzony test dla wartości PRD:
UWAGA: W kolumnach *GA* oraz *TABU* wartości podawane są w **PROCENTACH!**. Ze względów estetycznych w samych tabelach oznaczenia procenta (%) zostały pominięte.

Pair	GA	TABU	Abs.Diff	Rank	Sign
6	4.2	4.5	0.3	1	-1
8	6.1	5.2	0.9	2	+1
13	10	8.8	1.2	3	+1
5	3.3	4.6	1.3	4	-1
2	12	10	2	5	+1
1	7	10	3	6.5	-1
4	11	14	3	6.5	-1
12	17.3	21	3.7	8	-1
9	2.9	6.7	3.8	9	-1
11	5.2	9/9	4.7	10	-1
15	0.1	6.3	6.2	11	-1
10	11.3	18.6	7.3	12	-1
14	0.8	9.3	8.5	13	-1
3	22	12	10	14	+1
16	28.1	16.7	11.4	15	+1
7	18.8	7.3	11.5	16	+1

Tabela 11: Tabela rang dla testu Wilcoxona dla wartości PRD

$$W_- = 1 + 4 + 6.5 + 6.5 + 8 + 9 + 10 + 11 + 12 + 13 = 81$$
$$W_+ = 2 + 3 + 5 + 14 + 15 + 16 = 55$$
$$min(W_+, W_-) = 55 = T$$

W obu przypadkach wnioski będą identyczne.

7.5 Wnioski:

Wartość krytyczna $\alpha = 0.05$, dla tego typu statystyk $T_{crit} = 35$ (dana z tabeli dla hipotez "o jednym ogonie"). Nie możemy odrzucić hipotezy zerowej, gdyż nie ma wystarczających dowodów aby spekulować o tym, iż różnica median w tym przypadku jest mniejsza niż 0 ($T > 35$).

8 Przypadek Testowy 7 - Test statystyczny - Porównanie działania GA z 2OPT

8.1 Cele:

W teście statystycznym porównanym działanie algorytmu genetycznego z algorytmem 2OPT, którego implementacją zajęliśmy się w ramach listy 2. Hipotezą startową jest stwierdzenie, że algorytm genetyczny zwraca lepszą średnią medianę wyników niż algorytm 2OPT. Do analizy uzyskanych wyników zostanie wykorzystany test Wilcoxon'a dla par obserwacji (one-sided).

8.2 Założenia:

Do wykonania tego testu wykorzystano instancje znajdujące się w bibliotece TSPLIB. Początkowe permutacje są generowane w sposób losowy.

8.3 Wykorzystane instancje:

Lista wykorzystanych instancji jest identyczna do tej, która znajduje się dla 6 przypadku testowego.

8.4 Test:

Hipoteza zerowa: GA >= 2OPT
Hipoteza alternatywna: GA < 2OPT

Pair	GA	2OPT	Abs.Diff	Rank	Sign
6	11941	11930	11	1	+1
15	678	696	18	2	-1
5	5214	5304	90	3	-1
8	23508	23325	183	4	+1
4	7703	7423	280	5	+1
1	8098	7811	287	8	+1
13	64418	64025	393	7	+1
11	46617	47035	418	8	-1
3	7964	7098	866	9	+1
9	21360	22364	1004	10	-1
10	23717	21866	1851	11	+1
7	31533	28218	3315	12	+1
14	109056	112479	3423	13	=1
2	132632	127530	5102	14	+1
12	69279	64071	5208	15	+1
16	53914	46294	7620	16	+1

Tabela 12: Tabela rang dla testu Wilcoxona dla wartości funkcji celu

$$W_- = 2 + 3 + 8 + 10 + 13 = 36$$
$$W_+ = 1 + 4 + 5 + 6 + 7 + 9 + 11 + 12 + 14 + 15 + 16 = 100$$
$$min(W_+, W_-) = 36 = T$$

Poniżej powtórzony test dla wartości PRD:
UWAGA: W kolumnach *GA* oraz *2OPT* wartości podawane są w **PROCENTACH!**. Ze względów estetycznych w samych tabelach oznaczenia procenta (%) zostały pominięte.

Pair	GA	2OPT	Abs.Diff	Rank	Sign
6	4.2	4	0.2	1	+1
13	10	9.3	0.7	2	+1
8	6.1	5.3	0.8	3	+1
11	5.2	6.1	0.9	4	=1
5	3.3	5	1.7	5	-1
15	0.1	3	2.9	6	-1
1	7	4	3	7	+1
14	0.8	4	3.2	8	-1
2	12	8	4	9.5	+1
4	11	7	4	9.5	+1
9	2.9	7.8	4.9	11	-1
10	11.3	2.7	8.6	12	+1
12	17.3	8.5	8.8	13	+1
7	18.8	6.3	12.5	14	+1
3	22	9	13	15	+1
16	28.1	1	27.1	16	+1

Tabela 13: Tabela rang dla testu Wilcoxona dla wartości PRD

$$W_- = 1 + 2 + 3 + 7 + 9.5 + 9.5 + 12 + 13 + 14 + 15 + 16 = 102$$
$$W_+ = 4 + 5 + 6 + 8 + 11 = 34$$
$$min(W_+, W_-) = 34 = T$$

Zauważamy, że w tym przypadku otrzymujemy całkowicie różne wnioski

8.5 Wnioski:

Wartość krytyczna $\alpha = 0.05$, dla tego typu statystyk $T_{crit} = 35$ (dana z tabeli dla hipotez "o jednym ogonie").
Dla testu, który składał się z wartości równych funkcji celu otrzymaliśmy $T = 36$. Nie możemy odrzucić hipotezy zerowej, gdyż nie ma wystarczających dowodów aby spekulować o tym, iż różnica median w tym przypadku jest mniejsza niż 0 ($T > 35$). Natomiast dla testu, który składał się z wartości równych otrzymanemu PRD otrzymaliśmy $T = 34$, dla którego już hipoteza zerowa zostaje odrzucona.
Zauważamy, iż użyte dane mają znaczenie na otrzymane wyniki. Ze względów normalizacyjnych poprawniejsze wyniki produkuje porównywanie otrzymanych PRD.