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: 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. **Generate Population** - W tej metodzie dochodzi do wygenerowania początkowej populacji. Ze względu na fakt, iż uzyskae wyniki zależą od początkowej populacji w realizacji programu zdecydowaliśmy się na dwie metody generowania populacji. Drugą z nich jest **Improve Atsp**, której omówienie znajduje się poniżej. W ów metodzie generujemy liczbę osobników równą rozmiarowi populacji. Rozmiar populacji jest liczbą stałą podaną przez użytkownika. W pętli głównej dochodzi do wygenerowania osobników dwoma sposobami. Jednym z nich jest wykonanie funkcji **shuffle()**, drugą jest wykorzystanie wcześniej zaimplementowanego algorytmu **K random()** ze stałą równą *10*. Złożoności obliczeniowa ów metod generujących to odpowiednio: $O(n)$ oraz $O(n)$. Reasumując dochodzimy do wniosku, iż złożoność obliczeniowa rozpatrywanej metody wynosi $O(k * n)$, gdzie k oznacza rozmiar populacji.

2. **Improve atsp** - W tej metodzie dodatkowo generujemy 5 osobników korzystając z metody najbliższego sąsiada, której implementacją oraz analizą zajmowaliśmy się w ramach pierwszej listy zadań, stąd wiemy iż jej złożoność obliczeniowa jest równa $O(n^2 * logn)$. Dodatkowo do lekkiego wymieszania osobników wykorzystujemy metodę **unstack()**, której złożoność wynosi $O(k * n)$, gdzie k to rozmiar populacji (co zostanie wykazane w późniejszej częście sprawozdania).

3. **Find Best** - Metoda znajdująca najlepszego osobnika po funkcji celu wykorzystująca wbudowaną w pythonie funkcję **min()**. Złożoność obliczeniowa- $O(n)$.

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

    - Wykorzystując odcięcie jedyną problematyczną z punktu widzenia złożoności operacją jest operacja sortowania, która przy wykorzystaniu wbudowanej metody **sorted()** wykonuje się w czasie $O(n * logn)$,
    - Wykorzystując turniej losujemy z populacji daną część osobników, z której następienie wybieramy dwóch najlepszych. Jak powyżej, ze względu na wykonanie sortowania otrzymujemy złożoność równą $O(k * n * logn)$, gdzie k jest rozmiarem populacji a n liczbą wybieranych osobników

5. **Crossover** - Metoda dokonująca krzyżowań, która wykonuje się w czasie liniowym względem liczności populacji

6. **Mutation** - Metoda dokonująca mutacji, która wykonuje się w czasie liniowym względem liczności populacji

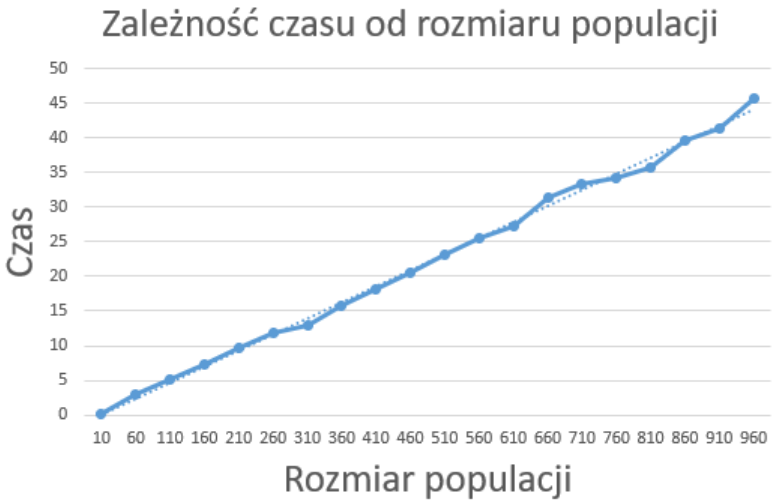
7. **Add Age** - Metoda dodająca wiek do każdego członka populacji wykonująca się w czasie liniowym względem liczby populacji

8. **Unstack** - Metoda wykorzystywana do uniknięcia stagnacji oraz przedwczesnych zbieżności, która wykonuje się w czasie liniowym względem liczby populacji

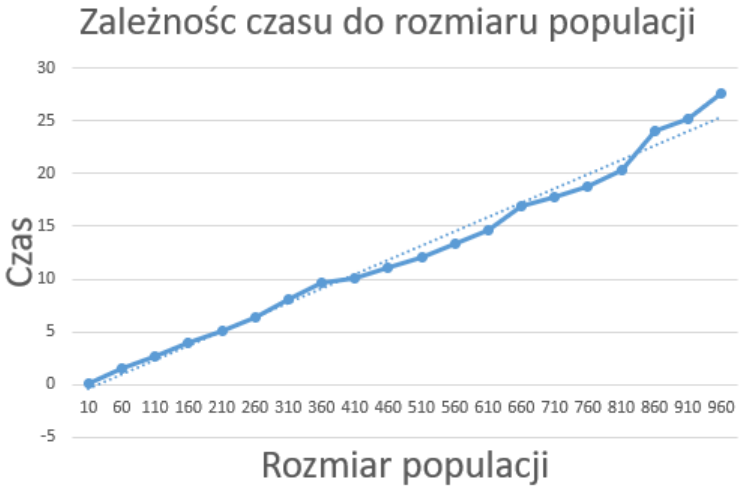
Wykresy: Na poniższych wykresach eksperymentalnie wykazaliśmy przedstawioną powyżej złożoność obliczeniową. Do testów wykorzystaliśmy instancje **berlin52** oraz **gr24**, które znajdują się w bibliotece TSPLIB. Dodatkowo badanymi parametrami były liczba iteracji oraz rozmiar populacji.

Rysunek 1: Zależność czasu od liczby iteracji dla instancji berlin52

Rysunek 2: Zależność czasu od liczby iteracji dla instancji gr24



Rysunek 3: Zależność czasu od rozmiaru populacji dla instancji berlin52



Rysunek 4: Zależność czasu od rozmiaru populacji dla instancji gr24

Wniosek: Złożoność obliczeniowa metody **populate()** wynosi $O(k*n*logn)$, gdzie k oznacza liczbę iteracji, a n- rozmiar populacji.
Przyglądając się otrzymanym wykresom zauważamy, iż przypominają one bardziej linię prostą aniżeli $n*logn$. Zapewne wynika to z faktu, iż rozmiar populacji jest zdecydowanie mniejszy niż liczba wykonywanych iteracji.
Złożoność pamięciowa:

2 Przypadek Testowy 1 - Algorytm genetyczny - zależność PRD od ilości 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 instancja **berlin52.tsp**. Dodatkowo współczynnik mutacji został ustalony na 0.15, współczynnik selekcji na 0.95. Ponad to wielkość populacji została ustalona na 10, 20, 30, 40. Badane iteracje były z zakresu 1000, 2000, 3000, ... 10000. Dla każdej wielkości populacji testy zostały przeprowadzone 10 razy.

2.3 Wyniki:

Poniższe tabele przedstawiają wyniki testów, odchylenie standardowe oraz błęd standardowy.

	10	20	30	40
1000	80.47	61.66	43.67	57.52
2000	62.40	48.94	39.31	45.41
3000	47.33	45.31	38.33	38.38
4000	36.40	39.02	32.11	32.29
5000	49.98	34.98	30.57	25.19
6000	34.92	33.88	32.87	30.54
7000	37.85	34.57	29.66	25.77
8000	39.34	34.57	25.27	27.23
9000	37.54	26.08	25.11	22.17
10000	38.83	28.65	24.14	22.52

Tabela 1: Wyniki to wartości PRD (wyrażone w procętach), uśrednione.

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

$$\sigma = \sqrt{\frac{\sum_{n=1}^{10}(\bar{x} - x_n)^2}{10}}$$

Błąd standardowe:

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{10}}$$

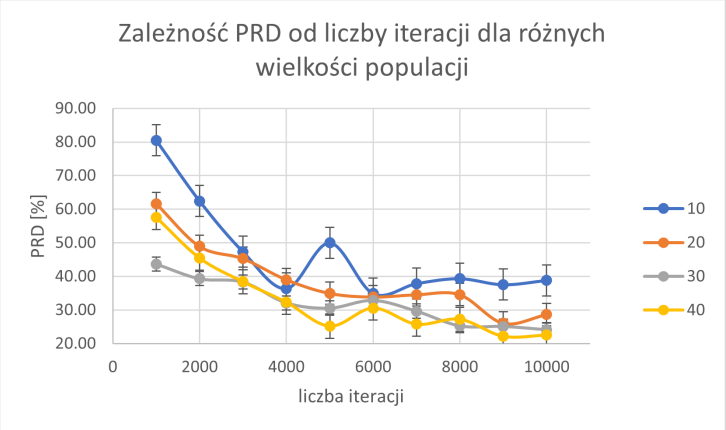
SD	10	20	30	40
1000	13.96	11.90	9.46	18.87
2000	18.91	11.07	8.42	16.18
3000	12.87	12.22	10.20	12.82
4000	15.65	16.56	9.80	10.56
5000	18.25	10.17	14.08	5.57
6000	11.27	10.79	7.59	8.99
7000	11.27	15.08	10.46	8.64
8000	13.86	9.02	9.84	9.10
9000	15.78	8.49	7.34	8.81
10000	17.20	9.70	8.36	7.42

Tabela 2: SD - Odchylenie standardowe

SE	10	20	30	40
1000	4.41	3.76	2.99	5.97
2000	5.98	3.50	2.66	5.12
3000	4.07	3.86	3.23	4.06
4000	4.95	5.24	3.10	3.34
5000	5.77	3.22	4.45	1.76
6000	3.56	3.41	2.40	2.84
7000	3.56	4.77	3.31	2.73
8000	4.38	2.85	3.11	2.88
9000	4.99	2.68	2.32	2.79
10000	5.44	3.07	2.64	2.35

Tabela 3: SE - Błąd standardowy

2.4 Wykresy:



Rysunek 5: 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, zwiększanie ilości iteracji zwiększa jakość rozwiązań jednak w okolicach 5000 iteracji zaczyna poprawa zaczyna być coraz mniejsza - zależniść nie jest liniowa, raczej logarytmiczna. Dodatkowo z wykresu można odczytać że zwiększanie rozmiaru populacji nie koniecznie zwiększa jakość rozwiązania dla jednakowej ilości iteracji.

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 instancja **berlin52.tsp**. Dodatkowo współczynnik selekcji został ustalony na 0.95. Ponad to wielkość populacji została ustalona na 10, liczba iteracji to 10000. Badane współczynniki mutacji $m \in 0.05, 0.15, 0.25...0.95$. Dla każdej wielkości współczynnika mutacji test został przeprowadzony trzykrotnie.

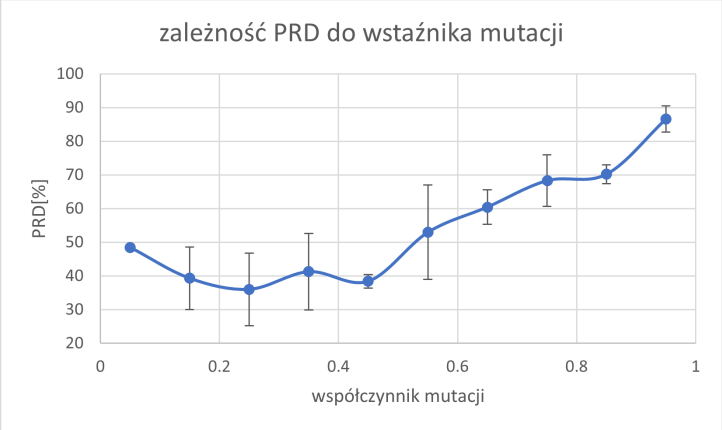
3.3 Wyniki:

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

m	PRD	SD	SE
0.05	48.47	1.54	0.89
0.15	39.30	16.07	9.28
0.25	36.01	18.63	10.76
0.35	41.24	19.61	11.32
0.45	38.39	3.52	2.03
0.55	52.92	24.27	14.01
0.65	60.43	8.97	5.18
0.75	68.26	13.21	7.62
0.85	70.17	4.90	2.83
0.95	86.56	6.82	3.94

Tabela 4: PRD - wyrażone w procętach [%], SD - Odchylenie standardowe, SE - Błąd standardowy

3.4 Wykresy:



Rysunek 6: zależność PRD od współczynnika mutacji

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

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

$$\sigma = \sqrt{\frac{\sum_{n=1}^{10}(\bar{x} - x_n)^2}{10}}$$

Błąd standardowe:

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{10}}$$

3.5 Wnioski:

Zmiana współczynnik mutacji przynosi niejednoznaczne rezultaty, przez wbudowaną w mutację losowość. Można jednak zauważyć pewne zależności. Dla współczynnika mutacji 0.05 odchylenie standardowe wyników jest najmniejsze co świadczy stałości rozwiązania przy niskim poziomie mutacji. Przy szansie na mutację 0.25 algorytm osiągnął minimum w testach, jednak odchylenie standardowe było duże. W punkcie 0.45, na wykresie pojawia się minimum lokalne, w tym punkcie odchylenie standardowe znów jest niskie. 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 instancja **berlin52.tsp**. Dodatkowo współczynnik selekcji został ustalony na 0.95. Ponad to wielkość populacji została ustalona na 10, liczba iteracji to 10000. Zmienne współczynniki mutacji $m \in 0.05, 0.15, 0.25...0.95$ oraz 2 metody selekcji. Dla każdej wielkości współczynnika mutacji test został przeprowadzony trzykrotnie.

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 obliczone według wzorów:
Odchylenie standardowe:

$$\sigma = \sqrt{\frac{\sum_{n=1}^{3}(\bar{x} - x_n)^2}{3}}$$

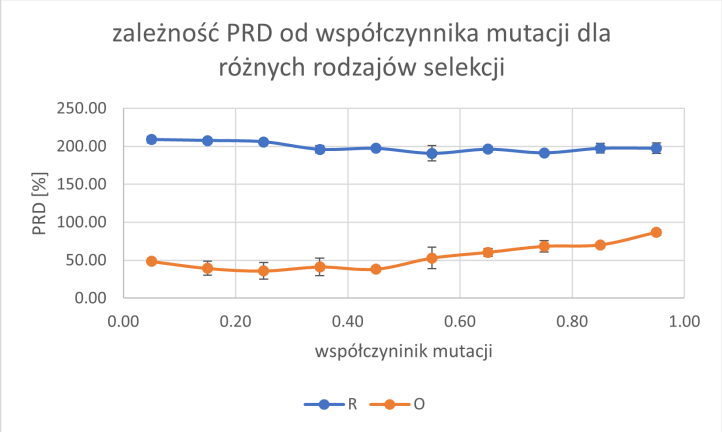
Błąd standardowe:

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{3}}$$

m	R	SD_R	SE_R	O	SD_O	SE_O
0.05	208.90	7.58	4.38	48.47	1.54	0.89
0.15	207.63	6.68	3.86	39.30	16.07	9.28
0.25	205.91	2.91	1.68	36.01	18.63	10.76
0.35	196.12	8.89	5.13	41.24	19.61	11.32
0.45	197.33	4.13	2.38	38.39	3.52	2.03
0.55	190.82	17.28	9.98	52.92	24.27	14.01
0.65	196.19	6.80	3.92	60.43	8.97	5.18
0.75	191.50	4.34	2.50	68.26	13.21	7.62
0.85	197.37	11.00	6.35	70.17	4.90	2.83
0.95	197.44	11.80	6.81	86.56	6.82	3.94

Tabela 5: m - współczynnik mutacji, R - PRD dla selekcji przez ruletkę, SD_R - odchylenie standardowe dla selekcji przez ruletkę, SE_R - Błąd standardowy dla selekcji przez ruletkę, O - PRD dla selekcji przez odcięcie, SD_O - odchylenie standardowe dla selekcji przez odcięcie, SE_O - Błąd standardowy dla selekcji przez odcięcie

4.5 Wykresy:



Rysunek 7: zależność PRD od współczynnika mutacji

Na wykresach przedstawione są średnie wartości PRD dla badanych danych.

4.6 Wnioski:

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

5 Przypadek Testowy 4 - Algorytm genetyczny - zależność PRD od rozmiar populacji

5.1 Cel:

Celem tego testu jest sprawdzenie w jakim stopniu rozmiar populacji wpływa na uzyskane PRD. Do testu testu zostaną wykorzystane następujące instancje z **TSPLIB**

1. berlin52
2. gr120
3. ftv70

Wszystkie inne argumenty zostały zainicjowane stałymi wartościami optymalnymi dla każdej z instancji. Rozmiary populacji należą do zbioru $[10, 50, 100, 150, 200, 250]$

5.2 Wyniki:

Wyniki testu przedstawione zostały w poniższej tabeli :

Rozmiar	PRD
50	19.84
100	14.16
150	6.92
200	6.36
250	6.57

Tabela 6: Wyniki otrzymane dla instancji berlin52

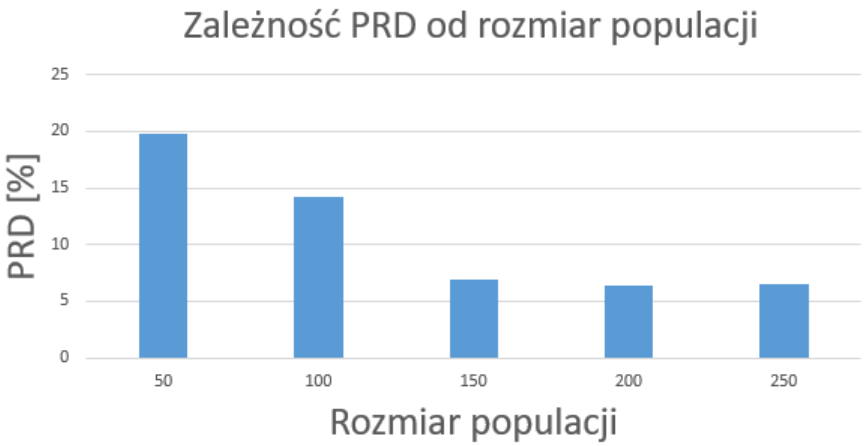
Rozmiar	PRD
50	158.72
100	140.95
150	101.75
200	95.82
250	93.08

Tabela 7: Wyniki otrzymane dla instancji gr120

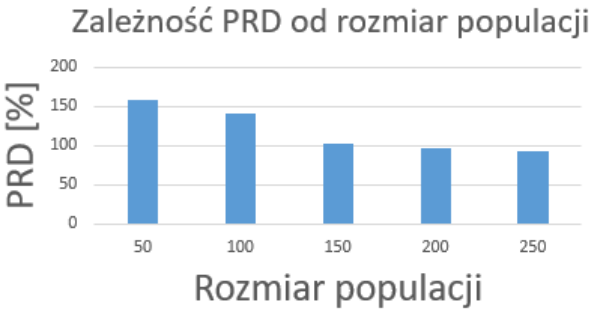
Rozmiar	PRD
50	155.58
100	156.61
150	144.25
200	146.0
250	138.35

Tabela 8: Wyniki otrzymane dla instancji ftv70

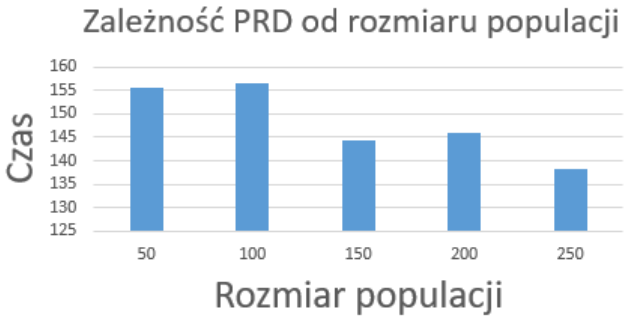
5.3 Wykresy:



Rysunek 8: Zależność otrzymanego PRD od rozmiaru populacji dla instancji berlin52



Rysunek 9: Zależność otrzymanego PRD od rozmiaru populacji dla instancji gr120



Rysunek 10: Zależność otrzymanego PRD od rozmiaru populacji dla instancji ftv70

5.4 Wnioski:

Zauważamy, iż zwiększenie rozmiaru populacji przy stałym utrzymaniu tych samych wariantów wpływa na uzyskanie lepszych wyników, z tą różnicą, że dla instancji asymetrycznych wpływ rozmiaru populacji jest zdecydowanie mniejszy niż dla instancji symetrycznych.

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.

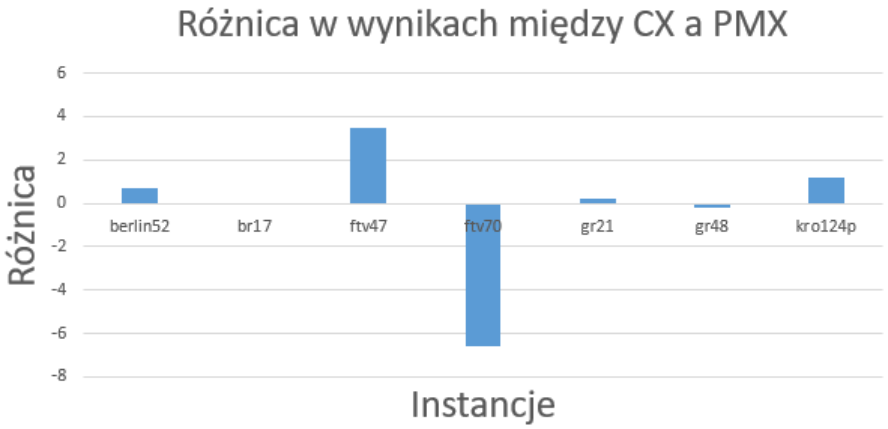
6.2 Wyniki:

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 11: 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

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

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 Wnioski:

8.5 Test:

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

8.6 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 ($T > 35$).

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 11: Tabela rang dla testu Wilcoxona

$$W_- = 2 + 3 + 8 + 10 + 13 = 36$$

$$W_+ = 1 + 4 + 5 + 6 + 7 + 9 + 11 + 12 + 14 + 15 + 16 = 100$$

$$min(W_+, W_-) = 36 = T$$