Sprawozdanie

Algorytmy Metaheurysyczne - Labolarorium

Radosław Wojtczak, Witold Karaś

1 Wstęp:

1.1 Algorytm Tabu Search

Algorytm Tabu Search jest metaheurystyką opierającą się na przeszukiwaniu przestrzeni stworzonej ze wszystkich możliwych rozwiązań, za pomocą sekwencji ruchów. Istotą ów algorytmu jest istnienie ruchów niedozwolonych (czyli ruchów tabu). Dzięki swojej strukturze algorytm unika ciągłego rozpatrywania jednego optimum lokalnego.
Nasza implementacja przewiduje dwa sposoby determinowania sąsiedztwa: poprzez ruchy **swap** oraz **invert**. Ruchy **swap** polegają na zamienianiu pozycjami dwóch miast z danej permutacji, natomiast ruchy typu **invert** polegają na odwróceniu kolejności w jakiej występują wszystkie miasta w danej permutacji między dwoma losowo wybranymi indeksami.

Złożoność obliczeniowa:
Główna pętla algorytmu odbywa się aż do momentu osiągnięcia kryterium zatrzymującego pracę. W naszej implementacji jest to liczba iteracji, która jest podawana przez użytkownika w momencie uruchomienia programu. Uznajemy więc, iż liczba iteracji jest liczbą stałą. Następnie odbywa się generowanie sąsiedztwa dla obecnej permutacji. Generowanie sąsiedztwa odbywa się w pętli for, której liczba iteracji jest podawana przez użytkownika w momencie uruchomienia programu.Na potrzeby ów analizy ustalmy, iż użytkownik generuje liczbę sąsiadów równą liczbie miast dla danej instancji (jest to zalecane rozwiązanie dla większych instancji). Po dokonaniu powyższego założenia zauważamy, iż pojedyncza realizacja funkcji generującej sąsiadów wykonuje się w czasie liniowym (podobnie jak w poprzednim zadaniu, wbudowana funkcja *reverse* w języku python wykonuje się w **czasie liniowym**, poza ów funkcją wykonywane są jedynie takie operacje jak losowanie, o stałym czasie realizacji, czy kopiowanie tablicy, które również jest realizowane w czasie liniowym). Ostatecznie wnioskujemy, iż złożoność obliczeniowa funkcji *get neighbors* wynosi $O(n^2)$. Następnie dochodzi do porównania wyników otrzymanych przez funkcję *fitness* (która jest odwrotnością funkcji kosztu) dla każdego z wygenerowanych sąsiadów. Biorąc pod uwagę, iż liczba sąsiadów jest liniowo zależna od liczby miast oraz funkcja *cost* wykonuje się w

czasie liniowym, dokonanie sprawdzenia, czy któryś z sąsiadów zwraca lepszy wynik niż aktualna permutacja również odbywa się w złożoności $O(n^2)$. Wszystkie następne operacje wykonywane w głonej pętli to operacje przypisania bądź porównania, które odbywają się w czasie stałym. Dodatkowo funkcja *shuffle* również odbywa się w czasie liniowym.
Wniosek: Funkcja *tabu* realizowana jest w czasie $O(n^2)$.
Złożoność pamięciowa:
Algorytm w swoim procesie często manewruje permutacjami miast, stąd wiemy, iż złożoność pamięciowa jest liniowo zależna od liczby miast. Jedynym wyjątkiem jest lista sąsiadów (gdzie w zależności od wprowadzonych przez użytkownika danych trzymamy odpowiednią liczbę permutacji) oraz lista tabu, która również zależy od wprowadzonych danych. Oznaczając jako k większą z wartości sąsiedztwa oraz pojemności listy tabu stwierdzamy, iż złożoność pamięciowa funkcji *tabu* wynosi $O(k * n)$

2 Przypadek Testowy 1 - Tabu Search - warianty tsp vs atsp

2.1 Cel:

W tej części zostaną ze sobą porównane PRD rozwiązania algorytmu Tabu Search dla przypadków symetrycznych oraz asymetrycznych.

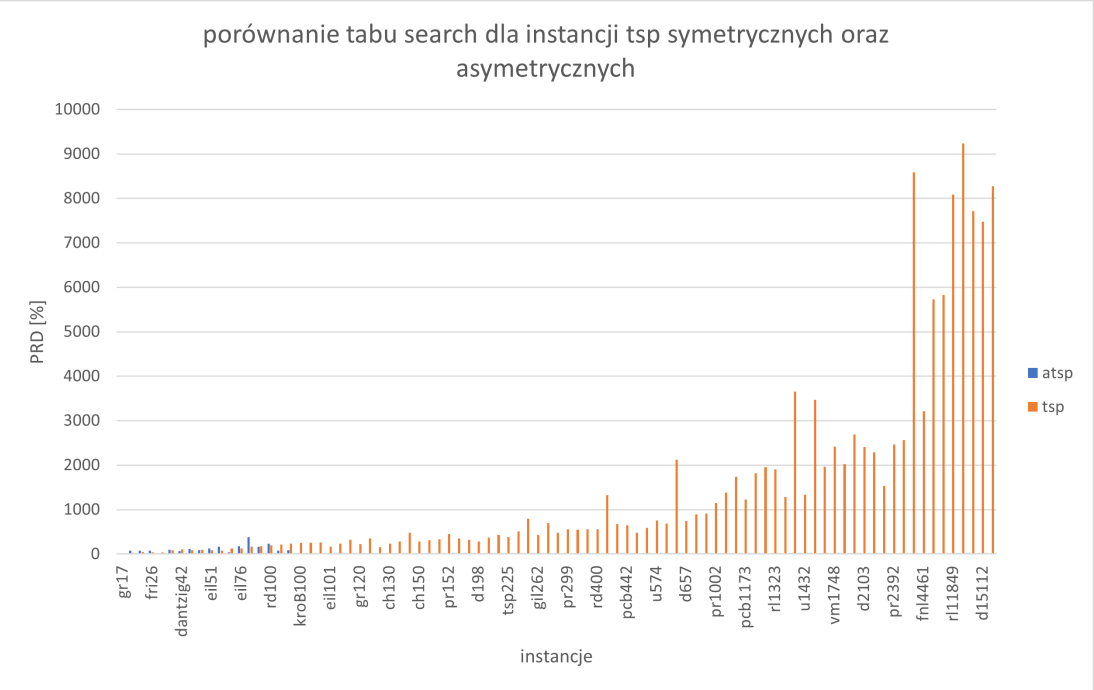
2.2 Założenia:

Do badania tego przypadku zostały wykorzystane grafy z list atsp oraz tsp. Dodatkowo liczba iteracji jest stała, równa 1000, maksymalna długość listy tabu równa 7, maksymalna liczba iteracji bez poprawy równa 10 oraz liczba sąsiadów równa 10% wszystkich węzłów w grafie (zaokrąglona w dół).

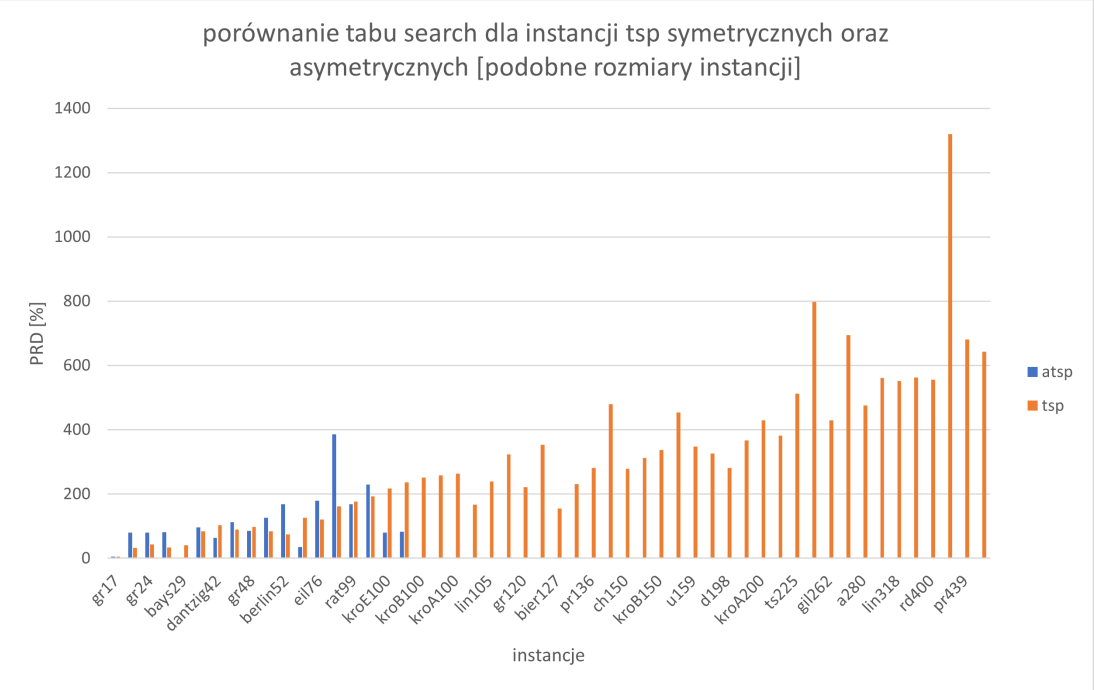
2.3 Wyniki:

Ze względu na czytelność wszystkie wyniki są załączone w pliku (out_test_1_tsp.csv) oraz (out_test_1_atsp.csv).

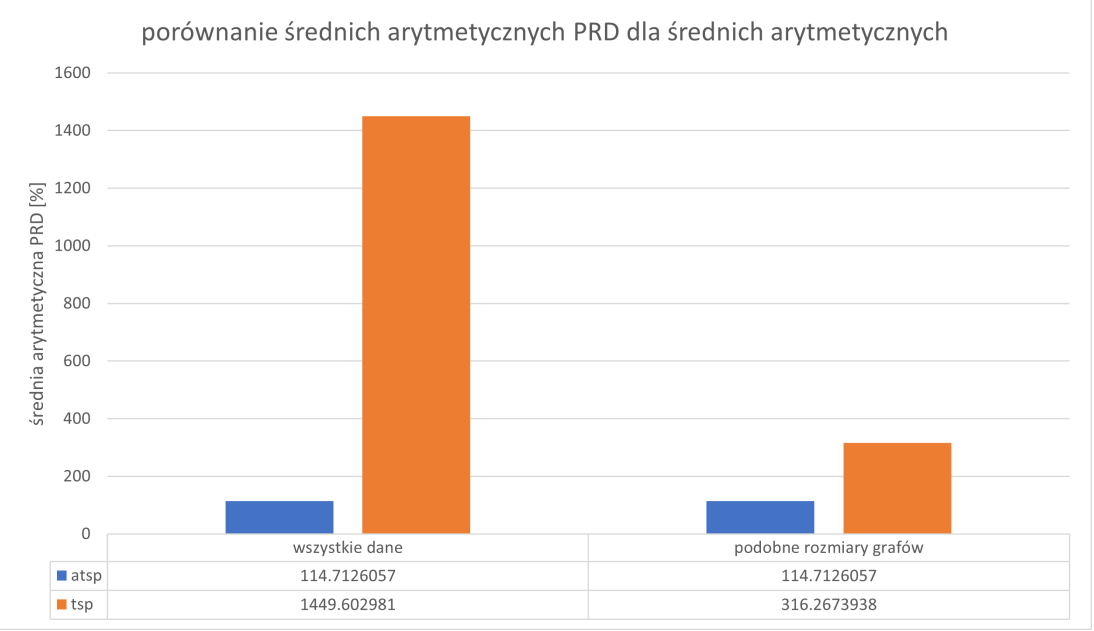
2.4 Wykresy:



Rysunek 1: Porównanie PRD dla instancji symetrycznych oraz asymetrycznych



Rysunek 2: Porównanie PRD dla instancji symetrycznych oraz asymetrycznych podobnych rozmiarów



Rysunek 3: Średnie PRD dla instancji symetrycznych oraz asymetrycznych

2.5 Wnioski:

Zauważmy, że dla danych asymetrycznych oraz zadanych parametrów algorytm średnio lepiej poradził sobie z instancjami asymetrycznymi. Widoczne jest również, że wraz ze wzrostem wielkości grafu oraz stałą liczbą iteracji algorytm radzi sobie coraz gorzej.

3 Przypadek Testowy 2 - Tabu Search - zależność PRD od liczby iteracji

3.1 Cel:

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

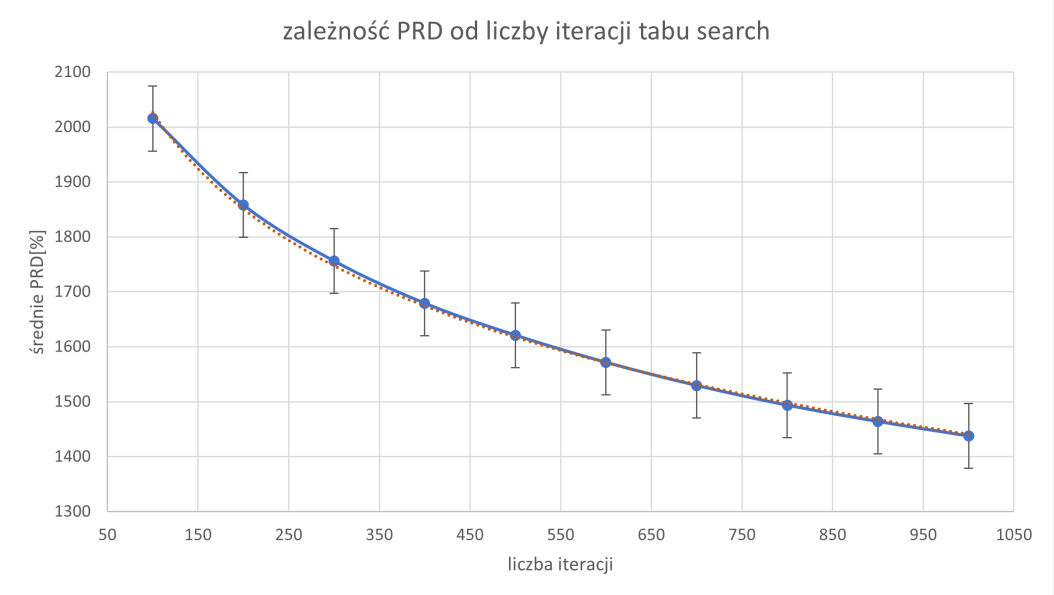
3.2 Założenia:

Do badania tego przypadku zostały wykorzystane instancje grafów z paczki tsp. Dodatkowo maksymalna długość listy tabu równa 50, maksymalna liczba iteracji bez poprawy równa 200 oraz liczba sąsiadów równa 10% wszystkich węzłów w grafie (zaokrąglona w dół). Ponad to dla każdej instancji ilość iteracji została ograniczona w zakresie od 100 do 1000 z iteracją co 100;

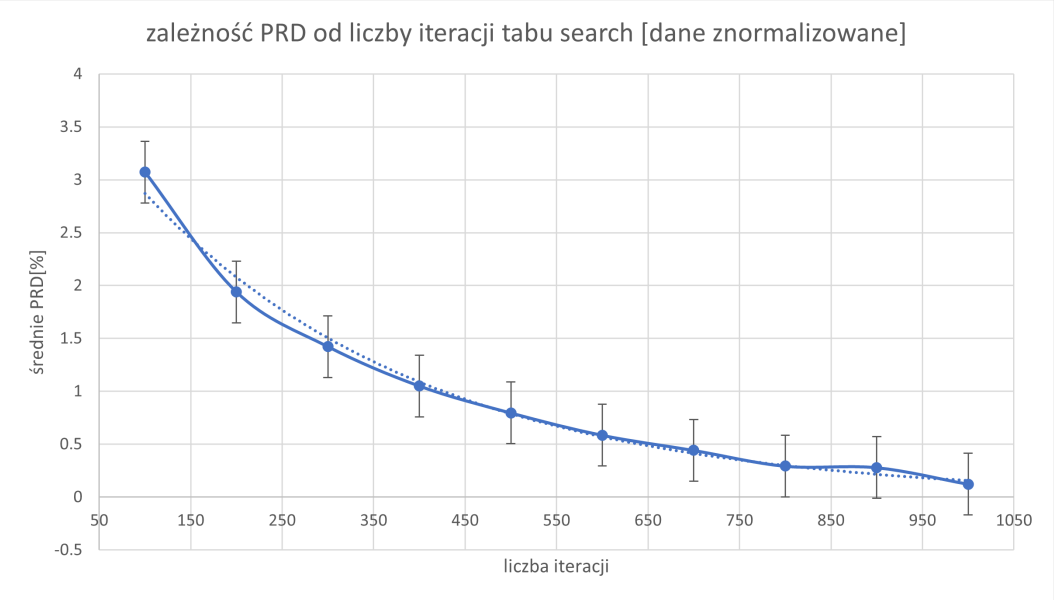
3.3 Wyniki:

Ze względu na czytelność wszystkie wyniki są załączone w pliku **out_test_2.csv**.

3.4 Wykresy:



Rysunek 4: zależność PRD od liczby iteracji



Rysunek 5: zależność PRD od liczby iteracji - dane znormalizowane

Na wykresach przedstawione są średnie wartości PRD dla badanych danych. Uśrednione wartości PRD maleją logarytmicznie, zgodnie z zaznaczoną linią trendu, 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:

Zauważmy, że dla danych testowych poprawa jakości rozwiązania względem zwiększania liczby iteracji była logarytmiczna. Co więcej dla mniejszych instancji rozwiązania były zdecydowanie lepsze.

4 Przypadek Testowy 3 - Tabu Search - Wpływ rozmiaru listy tabu na otrzymane wyniki

4.1 Cel:

Celem tego testu jest sprawdzenie w jaki sposób rozmiar listy tabu oddziałuje na otrzymany wynik. Do wykonania tego testu wykorzystano instancje znajdujące się w bibliotece *TSPLIB*. Dla każdego testu wykonano 1000 iteracji z maksymalną stagnacją równą 20 iteracji bez poprawy. Dodatkowo liczba sąsiadów wynosi tyle samo, ile wystepuje miast w danej instancji.
Sprawdzane rozmiary listy tabu należą do zbioru $k \in \{\sqrt{n}-5, \sqrt{n}-2, \sqrt{n}, \sqrt{n}+2, \sqrt{n}+5\}$.
Wartości pierwiastka są zaokrąglane **W DÓŁ** do najbliższej liczby całkowitej.

4.2 Wykorzystane instancje:

W tym teście zostały wykorzystane następujące instancje z biblioteki *TSPLIB*:

1. berlin52.tsp
2. bier127.tsp
3. eil76.tsp
4. gr120.tsp
5. gr48.tsp
6. hk48.tsp
7. pr107.tsp
8. pr76.tsp
9. st70.tsp
10. u159.tsp

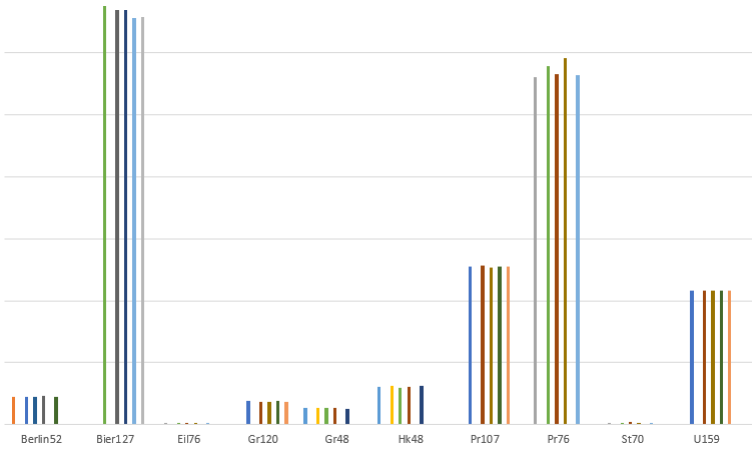
4.3 Wyniki:

Otrzymane wyniki przedstawia poniższa tabela:
Uwaga: Ze względow objętościowych w sprawozdaniu zostały zawarte jedyne informacje na temat pojedynczego testu

Instancja	Koszt	Rozmiar Tabu
berlin52	8865	2
berlin52	8905	5
berlin52	8979	7
berlin52	9314	9
berlin52	9026	12
bier127	135148	6
bier127	133727	9
bier127	133711	11
bier127	131323	13
bier127	131445	16
eil76	588	3
eil76	571	6
eil76	579	8
eil76	572	10
eil76	593	13
gr120.tsp	7765	5
gr120.tsp	7383	8
gr120.tsp	7461	10
gr120.tsp	7509	12
gr120.tsp	7370	15
gr48	5269	1
gr48	5303	4
gr48	5278	6
gr48	5336	8
gr48	5231	11
hk48	12223	1
hk48	12356	4
hk48	11951	6
hk48	12197	8
hk48	12614	11
pr107	50950	5
pr107	51262	8
pr107	50772	10
pr107	50859	12
pr107	51033	15
pr76	112189	3
pr76	115826	6
pr76	113237	8
pr76	118227	10
pr76	112758	13
st70	702	3
st70	697	6
st70	711	8
st70	703	10
st70	7039	13
u159	43396	5
u159	43338	8
u159	43299	10
u159	43151	12
u159	43171	15

Tabela 1: Tabela przedstawiająca otrzymane wyniki

4.4 Wykresy:



Rysunek 6: zależność funkcji kosztu od rozmiaru tablicy tabu dla podanych instancji

4.5 Wnioski:

W wykonanych testach zauważyliśmy brak większych zalezności między długością listy tabu a otrzymanymi wynikami. Brak klarownych rezultatów może być związany z niewielkimi rozmiarami, dla których były wykonywane testy, bądź z niewystarczającą liczbą iteracji algorytmu tabu.

5 Przypadek Testowy 4 - Tabu Search - Wpływ liczby rozpatrywanych sąsiadów na otrzymane wyniki

5.1 Cel:

Celem tego testu jest sprawdzenie w jaki sposób liczba rozpatrywanych sąsiadów w każdej iteracji algorytmu wpływa na wynik.
W tym teście sąsiedzi wyznaczani są przy pomocy ruchów typu *invert*.
Do wykonania tego testu wykorzystano wygenerowane grafy typu *FULL MATRIX* o liczbie miast ze zbioru $n \in \{100, 150, ..., 600\}$. Dla każdego testu wykonano 1000 iteracji z maksymalną stagnacją równą 20 iteracji bez poprawy. Dodatkowo rozmiar listy tabu jest stały równy 15 elementów. Sprawdzane rozmiary rozpatrywanych sąsiadów należą do zbioru $k \in \{100\%, 50\%, 25\%, 10\%,\}$.
Rozmiar podawany jest w procentach względem rozmiaru danej instancji $(100\% = n, 50\% = \frac{n}{2}, ...)$

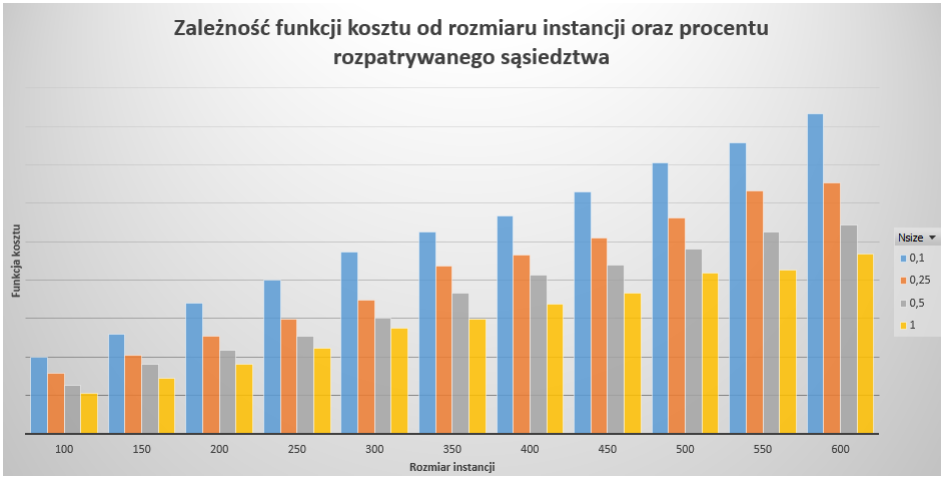
5.2 Wyniki:

Otrzymane wyniki przedstawia poniższa tabela:
Uwaga: Ze względow objętościowych w sprawozdaniu zostały zawarte jedyne informacje na temat pojedynczego testu

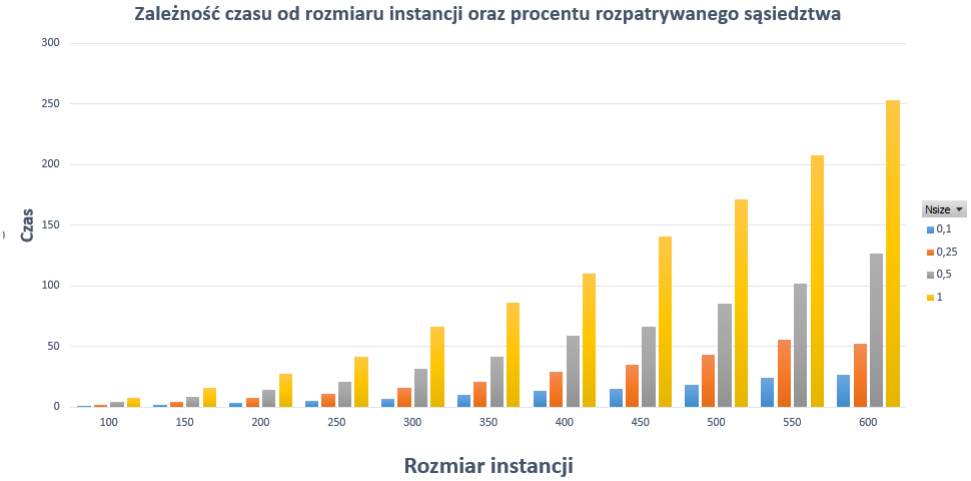
n	cost	time	Nsize
100	2005	1,35	10%
100	1575	2,39	25%
100	1267	4,08	50%
100	1068	7,53	100%
150	2601	2,29	10%
150	2053	4,35	25%
150	1807	8,22	50%
150	1438	15,84	100%
200	3393	3,75	10%
200	2538	7,8	25%
200	2188	14,07	50%
200	1818	27,89	100%
250	3999	4,92	10%
250	2979	10,97	25%
250	2538	21,19	50%
250	2220	41,99	100%
300	4737	6,85	10%
300	3486	16,04	25%
300	3015	31,39	50%
300	2762	66,19	100%
350	5258	10,6	10%
350	4366	21,29	25%
350	3664	41,41	50%
350	2983	85,81	100%
400	5666	13,41	10%
400	4640	29,54	25%
400	4142	59,31	50%
400	3386	110,54	100%
450	6303	14,92	10%
450	5084	35,32	25%
450	4398	66,15	50%
450	3652	140,77	100%
500	7060	18,77	10%
500	5616	43,4	25%
500	4799	85,06	50%
500	4192	170,88	100%
550	7559	24	10%
550	6318	55,35	25%
550	5240	101,99	50%
550	4262	207,54	100%
600	8330	26,54	10%
600	6520	52,15	25%
600	5441	126,8	50%
600	4677	252,83	100%

Tabela 2: Tabela przedstawiająca otrzymane wyniki, Nsize- rozmiar sąsiedztwa, czas podawany w sekundach

5.3 Wykresy:



Rysunek 7: zależność funkcji kosztu od rozmiaru instancji oraz procentu rozpatrywanego sąsiedztwa



Rysunek 8: zależność czasu od rozmiaru instancji oraz procentu rozpatrywanego sąsiedztwa

5.4 Wnioski:

Zauważamy, iż zwiększenie liczby potencjalnych kandydatów w każdej iteracji znacząco wpływa na otrzymane wyniki testowanego algorytmu, jak i na czas, w jakim ów algorytm wykonuje swoją pracę. Testy pokazały, iż optymalnym ustawieniem jest 50%, w którym otrzymujemy satysfakcjonujące rozwiązanie w rozsądnym czasie.

Warto zauważyć, iż na otrzymane wyniki czasowe mają wpływ takie czynniki jak język programowania, w którym algorytm został zaimplementowany oraz zużycie procesora komputera, na którym wykonywano powyższe testy.

6 Przypadek Testowy 5 - Tabu Search - Test statystyczny

6.1 Cel:

W teście statystycznym porównamy działanie algorytmu Tabu Search oraz wcześniej zaimplementowanego algorytmu 2-opt. Hipotezą startową jest stwierdzenie, iż Tabu Search średnio zwraca lepsze wyniki niż algorytm 2-opt. Do analizy uzyskanych wyników zostanie wykorzystany test Wilcoxon'a dla par obserwacji.

6.2 Założenia:

Do wykonania tego testu wykorzystano instancje znajdujące się w bibliotece *TSPLIB*. Dla Tabu Search, jak i dla 2-opt, początkowe permutacje zostały wygenerowane losowo.

6.3 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

6.4 Test statystyczny Wilcoxona:

Hipoteza zerowa: $TABU >= OPT$, gdzie OPT oznacza algorytm 2-opt, a Tabu- algorytm Tabu Search.
Hipoteza alternatywna: $TABU < OPT$

Pair	TABU	OPT	Abs.Diff	Rank	Sign
16	693	698	5	1	-1
4	563	575	12	2	-1
5	7440	7463	23	3	-1
11	22185	22292	107	4	-1
10	22110	21760	350	5	+1
1	8201	7829	372	6	+1
6	5613	5208	405	7	+1
3	7505	7042	463	8	+1
7	12423	11941	482	9	+1
8	30436	29681	755	10	+1
9	24676	23556	1110	11	+1
12	46033	48564	2531	12	-1
15	115958	118846	2888	13	-1
17	48240	44887	3353	14	+1
14	63132	59638	3494	15	+1
2	130696	125647	5049	16	+1
13	69337	64270	5067	17	+1

Tabela 3: Tabela rang dla testu Wilcoxona

$W_- = 35$.
$W_+ = 118$.
Wartość mniejsza to W_-, więc $T = 35$.

6.5 Wnioski:

Wartość krytyczna $\alpha = 0.05$. Dla tego typu statystyki $T_{crit} = 41$ (dana z tabeli dla hipotez "o jednym ogonie"). Hipoteza zerowa jest odrzucona, gdy $T \leqslant 41$. U nas $T = 35$, jako minimum z W_- i W_+, więc hipoteza zerowa zostaje odrzucona.

7 Przypadek Testowy 6 - Tabu Search - zależność PRD od maksymalnej liczby iteracji bez poprawy

7.1 Cel:

W tej części zostaną ze sobą porównane PRD rozwiązania algorytmu Tabu Search w zależności od maksymalnej liczby iteracji bez poprawy.

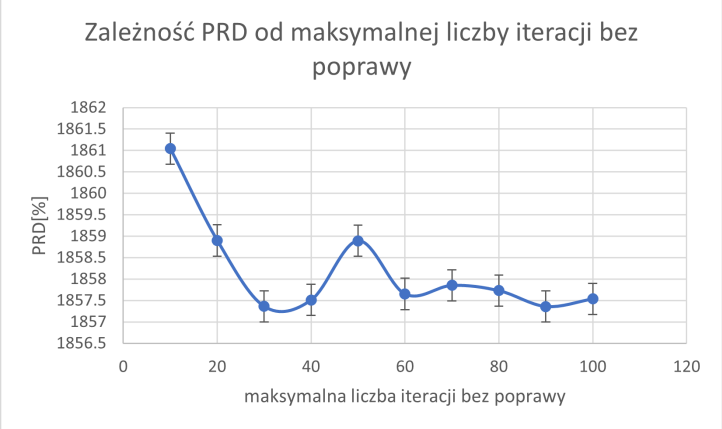
7.2 Założenia:

Do badania tego przypadku zostały wykorzystane instancje grafów z paczki tsp. Dodatkowo maksymalna długość listy tabu równa 50, maksymalna liczba iteracji równa 200 oraz liczba sąsiadów równa 10% wszystkich węzłów w grafie (zaokrąglona w dół). Ponad to dla każdej instancji ilość iteracji została ograniczona w zakresie od 10 do 100 z iteracją co 10;

7.3 Wyniki:

Ze względu na czytelność wszystkie wyniki są załączone w pliku **out_test_3.csv**.

7.4 Wykresy:



Rysunek 9: zależność PRD od maksymalnej liczby iteracji bez poprawy

Na wykresach przedstawione są średnie wartości PRD dla badanych danych.

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:

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

7.5 Wnioski:

Zauważmy, że dla danych testowych poprawa jakości rozwiązania względem zwiększania maksymalnej liczby iteracji nie jest wprost rosnąca/malejąca. Jednak można zauważyć że większa większa liczba iteracji bez poprawy w zakresie $\{10, 40\}$ wpływa na poprawe jakości rozwiązania. Przyglądając się poszczególnym przypadkom można zauważyć takie w których zmiana maksymalnej liczby iteracji bez poprawy nie zmienia jakości rozwiązania, jak i takie przypadki gdzie jakość rozwiązania stabilizuje się.