Sprawozdanie

Algorytmy Metaheurysyczne - Labolarorium

Radosław Wojtczak, Witold Karaś

1 Wstęp:

1.1 Algorytm k-random

Algorytm k-random polega na wygenerowaniu losowego dopuszczalnego rozwiązania (co w naszym przypadku, gdzie wszystkie grafy są gęste, oznacza wybranie losowej permutacji wierzchołków) i wyznaczenie dla niego funkcji celu. Ów schemat powtarzamy k razy, z czego wybieramy najmniejszą drogę co do wartości.

Złożoność obliczeniowa: Procedura *k-random* wykonuje n razy funkcję *cost*, która działa w czasie liniowym. Dodatkowo do generowania nowych rozwiązań korzystamy z wbudowanej biblioteki pythona o nazwie *random*, która zawiera funkcję *shuffle*, której złożoność obliczeniowa wynosi O(n), gdyż korzysta z **"Fisher–Yates-shuffle"**. Ostatecznie więc, złożoność obliczeniowa tej metody wynosi $O(k * n)$.

Złożoność pamięciowa: Procedura *k-random* generuje n-elementową permutację wierzchołków w formie listy. Następnie w specjalnej zmiennej przechowywana jest lista o najmniejszej funkcji celu, stąd ostatecznie nasza funkcje składa się z dwóch list o długości n. Wniosek: O(n)

1.2 Algorytm najbliższego sąsiada: omówienie

Algorytm najbliższego sąsiada jest algorytmem z grupy algorytmów zachłannych- wybiera lokalnie najlepsze rozwiązanie, w ten sposób budując pełne rozwiązanie.

Złożoność obliczeniowa: Procedura *nearestneighbor* wykonuje się aż do momentu, gdy cała lista *path* zostanie zapełniona. W każdym kroku dodajemy do niej jeden wierzchołek, więc zewnętrzny while wykonuje się dokładnie n razy. Następnie wewnątrz while'a przechodzimy liniowo po strukturze w poszukiwaniu sąsiada, do której jeszcze nie dotarliśmy(złożoność liniowa), jednakże dodatkowo dokonuje sortowania wierzchołków w celu ułatwienia iterowania po sąsiadach. Do sortowania korzystamy z wbudowanej funkcji *sorted*, która wykonuje się w czasie $O(n * logn)$. Ostatecznie otrzymujemy złożonośc obliczeniową worst-case na poziomie $O(n^2 * logn)$.

Złożoność pamięciowa: Wewnątrz algorytmu dokonujemy tworzenia dodatkowe słownika, stąd wnioskujemy, iz złożnośc pamięciowa tego algorytmu jest równa O(n).

1.3 Algorytm rozszerzonego najbliższego sąsiada: omówienie

Algorytm rozszerzonego najbliższego sąsiada polega na oduzależnieniu działania algorytmu od wyboru wierzchołka początkowego poprzez wykonanie podstawowego sąsiada dla wszystkich wierzchołków startowych **Złożność:** Algorytm wykorzystuje wszystkie funkcje algorytmu najbliższego sąsiada z tą różnicą, iż iteruje po wszystkich wierzchołkach, których jest dokładnie n, dlatego złożoność obliczeniowa tego algorytmu jest równa $O(n^3 * logn)$. Złożoność pamięciowa nie ulega zmianom.

1.4 Algorytm 2-opt

Algorytm 2-opt jest algorytmem, który korzysta z własności przestrzeni euklidesowych zwanej nierównością trójkatą. Zauważa, iż jeśli drogi są ze sobą "splecione" to ich rozwinięcie swodouje zmniejszenie funkcji celu. Na tej podstawie posiadając permutację wejściową dokonujemy inwersji pewnej części tablicy w oczekiwaniu, iż doszło do takowych rozplątań. **Złożoność obliczeniowa:** Ze względu na występującą pętle while, która wykonuje się do momentu, aż nie natrafimy na ulepszenie aktualnej permutacji bardzo trudno określić złożoność obliczeniową tego algorytmu. Na podstawie testów dla przestrzeni Euklidesowy wnioskujemy, iż ów algorytm może pracować nawet w złożoności $O(n^3)$. Sytuacja komplikuje się, gdy przechodzimy w przestrzenie, które nie spełniają nierówności trójkątą. Intuicja może nam podpowiadać, iż w tej sytuacji możemy osiągnąć aż wykładniczą złożoność w najgorszym przypadku, jednakże żadne z naszych testów nie potwierdziły tej tezy. **Złożoność pamięciowa:** Ze względu na to, iż w procedurze tworzymy jedną listę n elementową przechowującą aktualną permutację, i jedną listę n elementową przechowującą najlepszą permutację, złożoność pamięciowa sprowadza się do O(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.

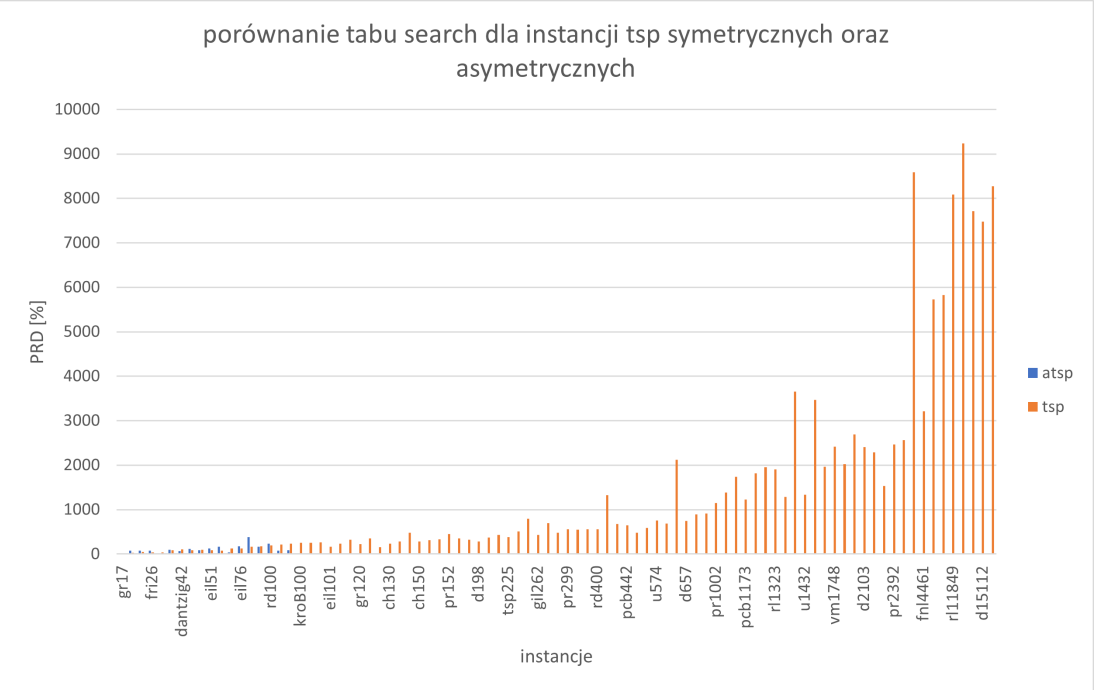
2.2 Założenia:

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).

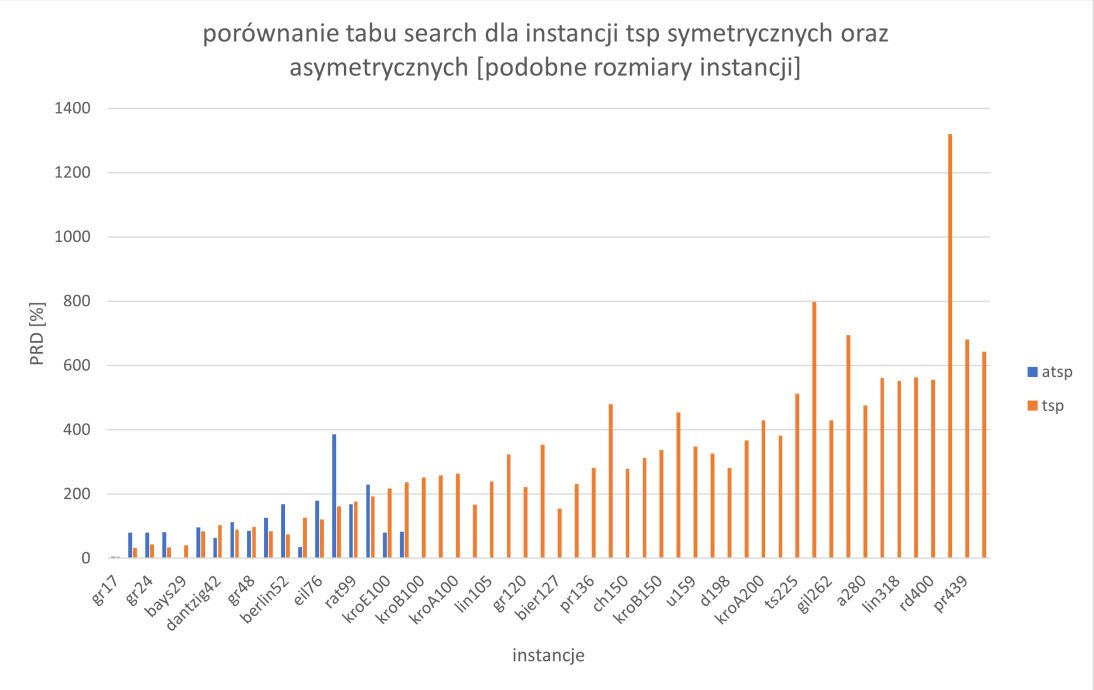
2.3 Wyniki:

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

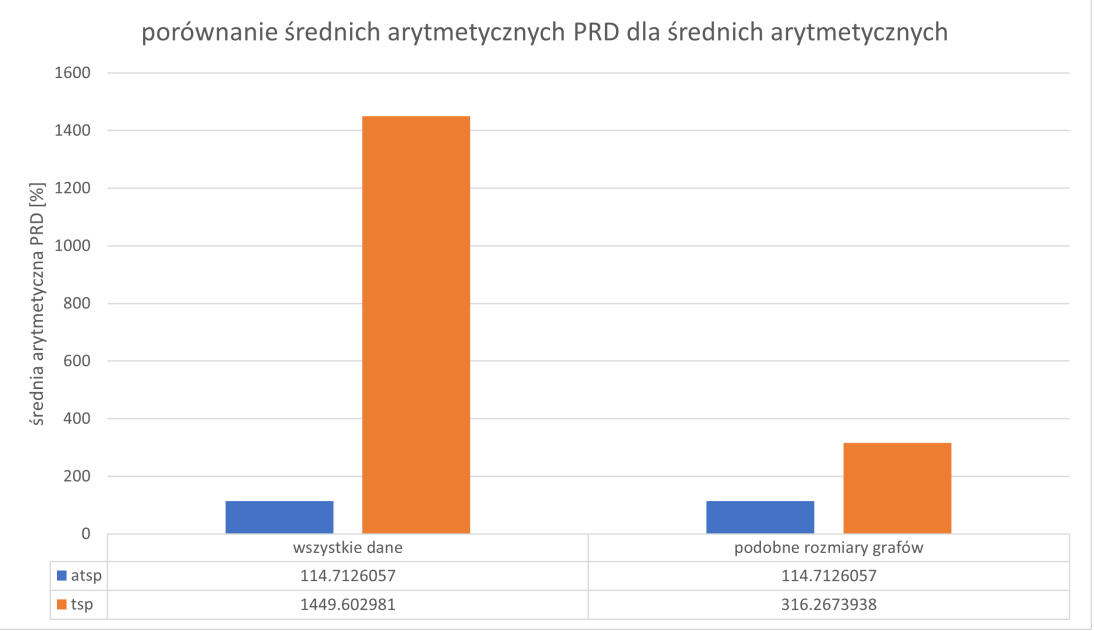
2.4 Wykresy:



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.

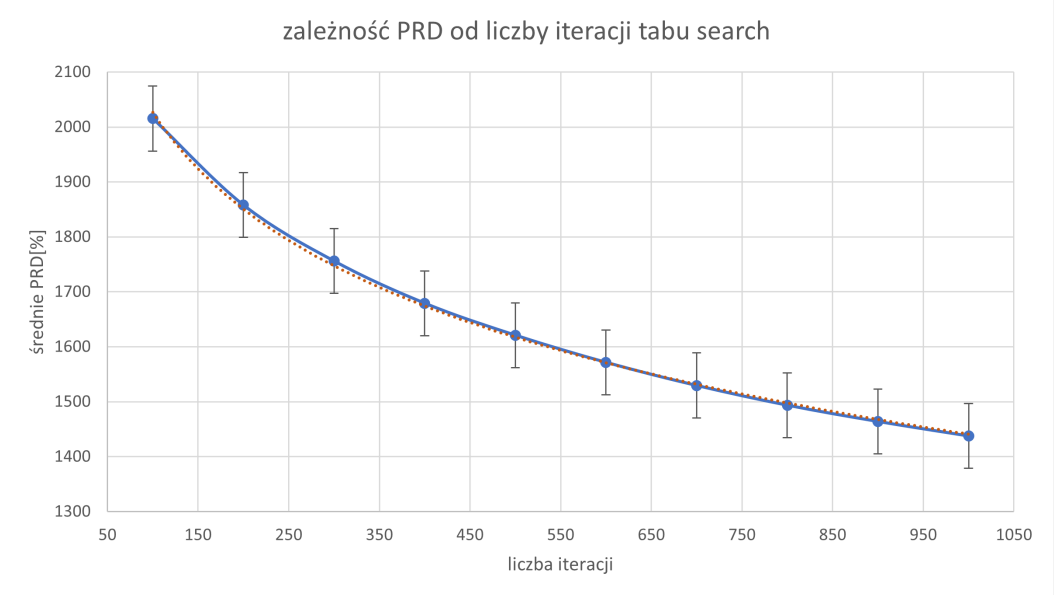
3.2 Założenia:

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;

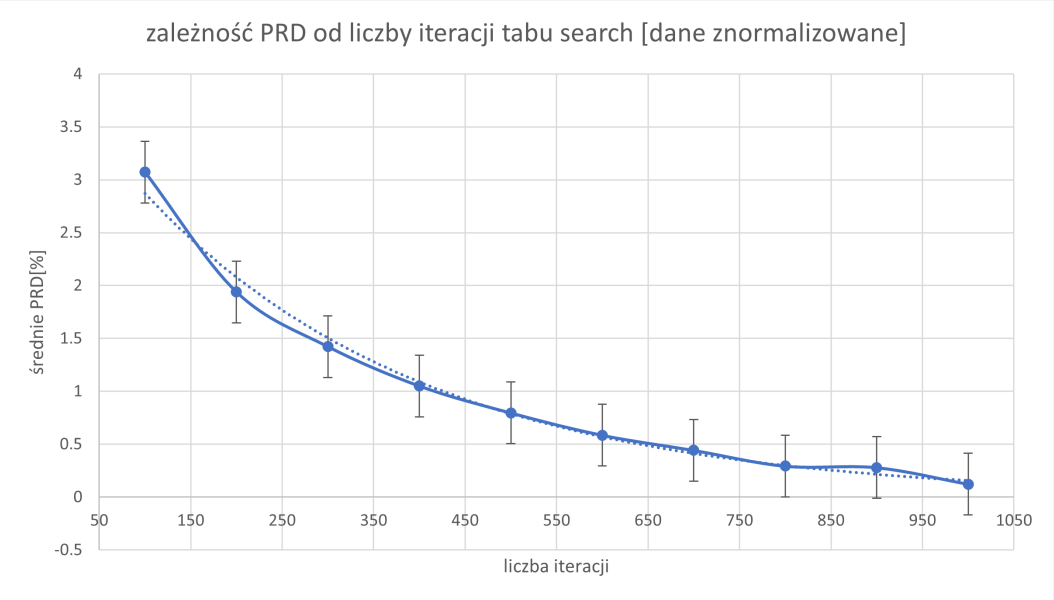
3.3 Wyniki:

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

4.1 Cel:

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

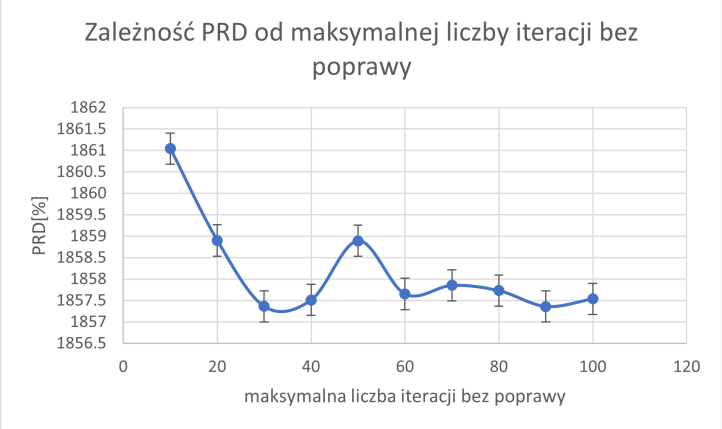
4.2 Założenia:

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;

4.3 Wyniki:

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

4.4 Wykresy:



Rysunek 6: 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:

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

4.5 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.