Projektowanie efektywnych algorytmów

Projekt

21.12.2020

248821 Michał Rajkowski

(5)Tabu Search

Spis treści

1. Sformułowanie zadania

Zadanie polega na opracowaniu, implementacji i zbadaniu efektywności algorytmu „poszukiwania z zakazami", rozwiązującego problem komiwojażera w wersji optymalizacyjnej.

Problem komiwojażera (ang. Travelling salesman problem) jest to zagadnienie optymalizacyjne polegające na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym.

Zagadnienie to można podzielić na dwa podtypy ze względu na wagi ścieżek łączących wierzchołki grafu. W symetrycznym problemie komiwojażera (STSP) dla dowolnych dwóch wierzchołków A B krawędź łącząca wierzchołek A z B ma taką samą wagę jak ta łącząca wierzchołek B z A. Natomiast w asymetrycznym problemie komiwojażera (ATSP) wagi te mogą się różnić.

Główną trudnością problemu jest duża liczba danych do analizy. W przypadku problemu symetrycznego dla n miast liczba kombinacji wynosi $\frac{(n-1)!}{2}$.

Dla problemy asymetrycznego jest to aż n!.

2. Metoda

2.1 Opis metody

Tabu search jest metaheurystyczną metodą rozwiązywania problemów optymalizacji kombinatorycznej. Jest to algorytm deterministyczny, co oznacza iż, każde jego uruchomienie doprowadzi do tego samego wyniku.

Tabu search swoje działanie opiera na wyborze potencjalnego rozwiązania, a następnie sprawdzeniu jego sąsiadów, z nadzieją na znalezienie lepszego wyniku. Przeszukiwania lokalne posiadają jednak problem gdy znajdą się w lokalnym minimum, nie ma wtedy już sąsiadów danego stanu o lepszej wartości.

Z tego powodu metodę rozbudowano o dodatkowe rozwiązania pozwalające opuszczać lokalne minima. Akceptowalne są ruchy pogorszające w momencie gdy żaden ruch nie prowadzi do lepszego rozwiązania. Istnieje także lista zakazów (tabu), której celem jest zniechęcenie algorytmu do odwiedzania stanów w których już się znajdował.

2.2 Elementy Tabu Search mające wpływ na jakość i czas rozwiązania

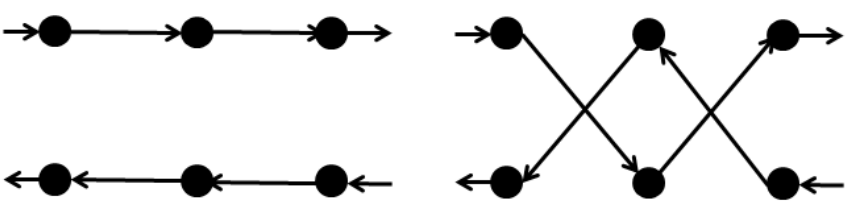
Generowanie rozwiązania początkowego

Algorytm mógłby rozpocząć się od dowolnego stanu początkowego, bowiem i tak doszedłby do rozwiązania bliskiemu optimum przy odpowiednio długim czasie pracy. Wybranie właściwej wartości rozwiązania początkowego pozwala jednak zaoszczędzić znacznie na czasie poszukiwania, co oznacza, że przy stałej liczbie iteracji zwróci wyniki bliższe rozwiązania optymalnego.

Nieoptymalnym, ale zwracającym dobre wyniki (oraz bardzo prostym w implementacji) rozwiązaniem jest zachłanne znalezienie rozwiązania początkowego. Wierzchołek startowy łączymy z najbliższym, niepołączonym jeszcze wierzchołkiem, następnie operacje tą powtarzamy dla każdego nowopołączonego wierzchołka, aż do utworzenia pełnej ścieżki.

Sąsiedztwo

Metody generowania rozwiązań sąsiednich opierają się na usuwaniu ze ścieżki krawędzi i zastępowaniu ich innymi. Wybraną przeze mnie metodą jest Vertex Exchange, polega ona na zamianie pozycji dwóch wierzchołków rozwiązania. (rys 1). Prowadzi to do utworzenia nowego cyklu o koszcie bliskim aktualnemu.



Rys1: Ilustracja działania metody Vertex Exchange

Rozmiar listy tabu

Rozmiar listy tabu dobierany jest eksperymentalnie. Określa rozmiar listy stanów do których algorytm nie może się cofać w kolejnych ruchach. Im większa lista tabu tym większa dywersyfikacja algorytmu. Jej wartość przeważnie sięga od 3n/2 do n/32 (gdzie n to liczba wierzchołków tsp). Należy jednak pamiętać, iż wraz ze wzrostem rozmiaru tabu list proporcjonalnie rośnie czas wykonywania się algorytmu.

Kryterium aspiracji

Czasami lista tabu okazuje się być zbyt silna/pamiętliwa. Zabrania ona wykonywania pewnych kroków, mimo iż nie powoduje to powstaniu cykli. Istnieją więc sytuacje, w których opłacalne byłoby złamanie jej zakazu, nazywamy je kryterium aspiracji.

Najbardziej intuicyjnym kryterium aspiracji jest dozwolenie złamania zakazu tabu w przypadku znalezienia rozwiązania lepszego od najlepszego znalezionego rozwiązania.

Długość kadencji

Określa jak długo elementy rezydują w liście tabu.

Wybrana została przeze mnie najprostsza opcja, elementy przechowywane są w liście tabu przez ilość iteracji równą długości listy, następnie zostaną nadpisane przez nowo umieszczane elementy.

Strategia dywersyfikacji

Strategia dywersyfikacji pozwala opuścić minima lokalne w których program został uwięziony. Sprawia ona, że chce on także badać nowe stany, mimo iż są one gorsze od aktualnie znalezionych.

Dywersyfikacja w przygotowanym przeze mnie programie w dużej mierze oparta jest o listę tabu. Nie pozwala ona wracać na zbadane niedawno ścieżki. W algorytmie dozwolone jest także odwiedzanie stanów o wartościach gorszych od aktualnego.

Kryterium zakończenia

Kryterium terminacji decyduje kiedy algorytm kończy swoje działanie. TSP bowiem z założeń nigdy nie kończy swojego działania samoczynnie. Algorytm pamięta jedynie najlepszy wynik, ale nie zapamiętuje wszystkich miejsc w których znalazł się w przeszłości. Z tego powodu nawet w momencie znalezienia wyniku optymalnego będzie kontynuował swoje działanie.

Najczęściej terminacja następuje po upływie określonej liczby cykli pętli głównej. Wartość ta dobierana jest w sposób empiryczny.

Obsługa zdarzeń krytycznych

Obsługa zdarzeń krytycznych pozwala na opuszczenie algorytmowi obszarów stanów w których utknął. Następuje to gdy przez k kolejnych iteracji nie zostało znalezione lepsze rozwiązanie bądź rozwiązania kolejnych iteracji były bliskie rozwiązania startowego. Uruchamiana jest wtedy procedura generowania nowego rozwiązania startowego, może być ono losowe lub oparte na zgromadzonych już informacjach. W przypadku zaimplementowanego przeze mnie algorytmu nie występuje resetowanie jego działania.

2.3 Pseudokod

TSP Tabu:

wybierz lub wylosuj punkt startowy $x_0 \in X$
$x_{opt} \leftarrow x0$
tabu list $\leftarrow \emptyset$
repeat
    $x_0 \leftarrow$ AspirationPlus(x_0)
    if $f(x_0) > f(x_{opt})$ **then**
        $x_{opt} \leftarrow x_0$
    zweryfikuj tabu list
    $\forall$element $\in$ tabu list **do**
        - - $kadencja_i$
        if $kadencja_i = 0$ **then**
            usuń element($atrybut_i, kadencja_i$) z tabu list
    if CriticalEvents() **then**
    $x_0 \leftarrow$ Restart() (**Dywersyfikacja**)
    if $f(x_0) > f(x_{opt})$ **then**
        $x_{opt} \leftarrow x_0$
until warunek zakończenia

2.4 Teoretyczna złożoność obliczeniowa

Zastanówmy się jaka jest teoretyczna złożoność obliczeniowa algorytmu na podstawie jego pseudokodu.

Algorytm wykonuje się w pętli głównej. Pętla ta kończy swoje działania w momencie terminacji. Ten określany jest przez predefiniowaną liczbę cykli k.

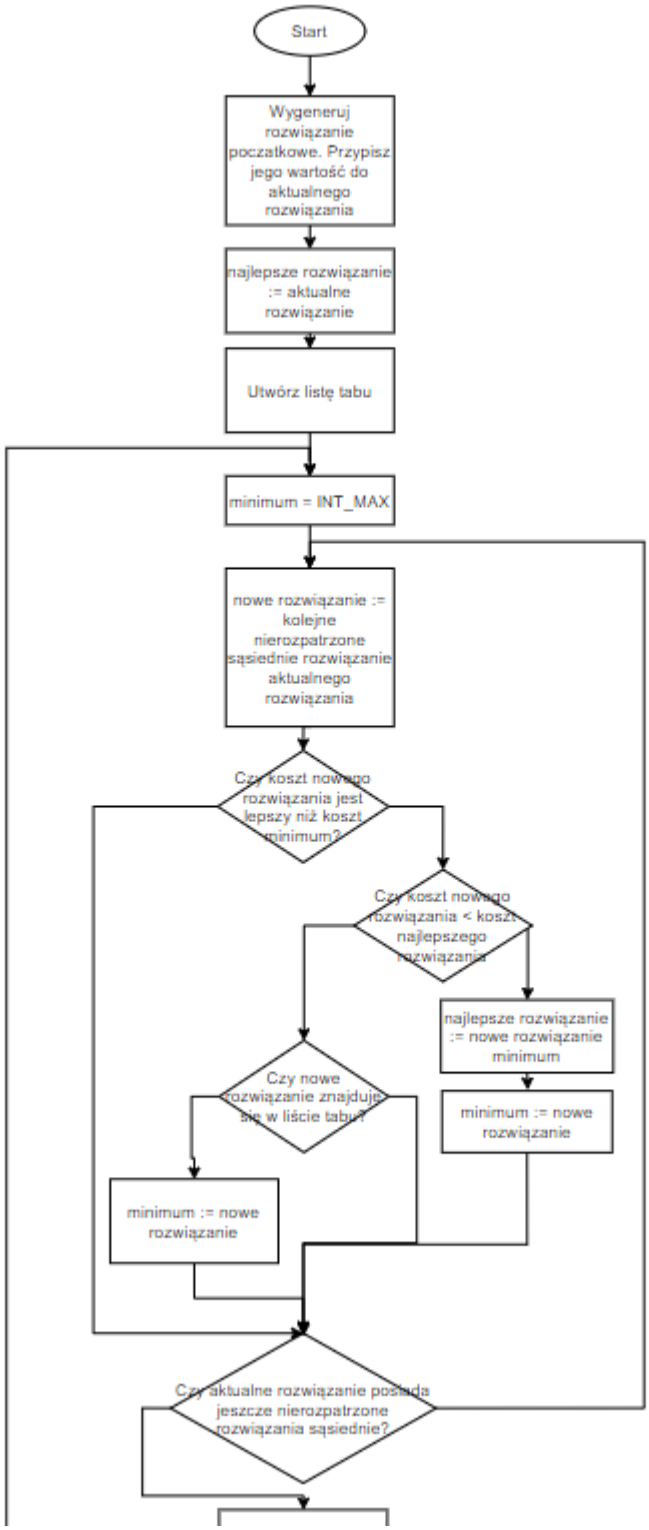
W pętli głównej algorytm dla aktualnie przechowywanej ścieżki sprawdza wszystkie jej sąsiednie rozwiązania. Tworzone są one poprzez zamianę miejscami 2 wierzchołków ścieżki. Dla ścieżki długości n możemy wybrać jej 2 wierzchołki na $\binom{n}{2}$ sposoby, zatem jest to n*(n-1)/2 operacji.

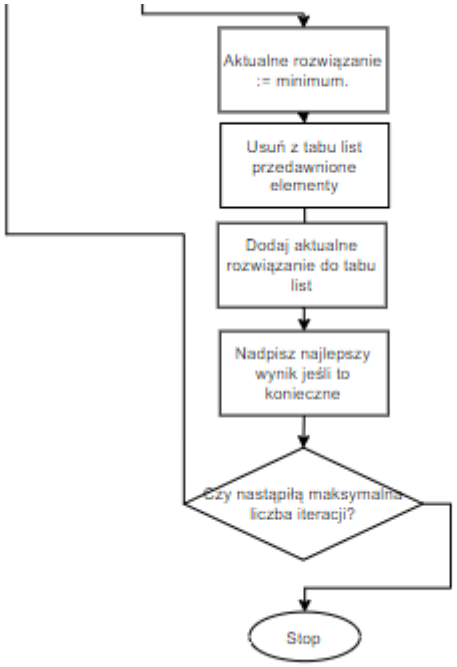
Dla każdego otrzymanego wyniku musimy przejrzeć listę tabu, której długość jest zależna od liczby wierzchołków TSP n.

Pozostałe operacje to operacje stałe. Zatem na tej podstawie możemy oszacować, że złożoność czasowa obliczeniowa algorytmu przystaje do $O(n^3)$. (lub do $O(k * n^3)$ jeżeli chcemy uwzględnić ilośc cykli do terminacji).

Złożoność pamięciowa algorytmu to w dużej mierze struktóra przechowująca graf. Ma ona postać macierzy n*n. Sam algorytm posiada jedynie zapamiętane kilka ścieżek oraz liste tabu więc złożoność pamięciowa samego algorytmu bez przechowywania danych problemu to O(n), wraz z macierzą to O(n^2).

3. Algorytm


Start
Wygeneruj rozwiązanie początkowe. Przypisz jego wartość do aktualnego rozwiązania
najlepsze rozwiązanie := aktualne rozwiązanie
Utwórz listę tabu
minimum = INT_MAX
nowe rozwiązanie := kolejne nierozpatrzone sąsiednie rozwiązanie aktualnego rozwiązania
Czy koszt nowego rozwiązania jest lepszy niż koszt minimum?
Czy koszt nowego rozwiązania < koszt najlepszego rozwiązania
najlepsze rozwiązanie := nowe rozwiązanie minimum
minimum := nowe rozwiązanie
Czy nowe rozwiązanie znajduje się w liście tabu?
minimum := nowe rozwiązanie
Czy aktualne rozwiązanie posiada jeszcze nierozpatrzone rozwiązania sąsiednie?



Rys 2: schemat blokowy algorytmu

Niestety schemat okazał się zbyt duży aby zmieścił się na 1 stronie. Został on ucięty w najbardziej nieinwazyjny sposób jak było to możliwe.

4. Dane testowe

Do sprawdzenia poprawności działania algorytmu oraz do badań wybrano następujący zestaw instancji:

- tsp_10.txt
- tsp_13.txt
- tsp_17.txt
- gr24.txt
- gr48.txt
- kroA100.txt
- rbg323.txt

tsp_10.txt - tsp_17.txt to pliki testowe pochodzący ze strony doktora Mierzwy:

http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pea-stud/tsp/

Pliki gr24.txt ,gr48.txt oraz kroA100.txt zostały utworzone na bazie plików z:

http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html

rbg323.txt:

http://elib.zib.de/pub/mp-testdata/tsp/tsplib/atsp/index.html

i zostały okrojone o niepotrzebne nagłówki pliku aby łatwiej było obsługiwać je w programie, stąd rozszerzenie txt.

5. Procedura badawcza

Jako, iż program posiadał kilka parametrów, pomiary zostały wykonane wielokrotnie przy ich zmianie.

Jako pierwszy obliczono wpływ długości tablicy Tabu na dokładność i czas trwania pomiarów w zależności od liczby wierzchołków.

Następnie dla ustalonej wartości tabu zbadano dokładność pomiarów oraz czas ich trwania w zależności od liczby wierzchołków oraz maksymalnej liczby iteracji tabu searcha.

Dla najlepszych parametrów obliczono błędy względne w zależności od liczby wierzchołków tsp oraz czasy wykonywania obliczeń.

Finalnie zmierzono i porównano wyniki dla najlepszych parametrów gdy dane inicjacyjne są wybierane w sposób zachłanny i gdy są wybierane w sposób losowy (za losowy przyjmujemy przypisanie im 1-2-3-4-…-N, gdyż wynik będzie tożsamy. Nie znamy bowiem wag krawędzi zatem takie ułożenie jest w rzeczywistości losowym startem, losowość tkwi w wagach).

Dane wejściowe przedstawione były w następującej formie: (nazwa pliku, typ pliku, liczba wykonań, optymalne rozwiązanie (typ pliku to specjalne słowo ułatwiające odczytywanie plików z różnym typem zapisu danych)):

tsp_10.txt full 10 212

tsp_13.txt full 10 269

tsp_17.txt full 10 39

gr24.txt half 10 1272

gr48.txt half 10 5046

kroA100.txt points 10 21282

rbg323.txt full 3 1326

final_test.csv

Dane wyjściowe każdych pomiarów zapisywano do osobnych plików. Następnie ubogacano je o wykresy. W następujących plikach można zobaczyć dane obliczeń:

Tabu_czas.xlsx – wyniki wpływu rozmiaru tabu na czas obliczeń

Tabu_percent.xlsx – wyniki wpływu rozmiaru tabu na błąd względny obliczeń

Terminacja_czasy.xlsx – wyniki wpływu liczby iteracji do terminacji na czas obliczeń

Terminacja_percent.xlsx – wyniki wpływu liczby iteracji do terminacji na błąd względny obliczeń

Optymalne_parametry_czasy.xlsx – czas obliczeń tsp instancji przy optymalnych parametrach

Optymalne_parametry_percent.xlsx – błąd względny obliczeń tsp instancji przy optymalnych parametrach

5.1 Metoda pomiaru czasu

Pomiary czasu uzyskane zostały przy pomocy funkcji clock pochodzącej z biblioteki time.h.

Funkcja clock zwraca czas zużyty przez procesor w „clock ticks”. Następnie otrzymane tiki zostają przekonwertowane na milisekundy przy pomocy makra CLOCKS_PER_SEC. Za ostateczne obliczanie czasu odpowiedzialna jest następująca linia kodu:

```
cpu_time = ((double) (end - start)) / (CLOCKS_PER_SEC/1000);
```

5.2 Sprzęt

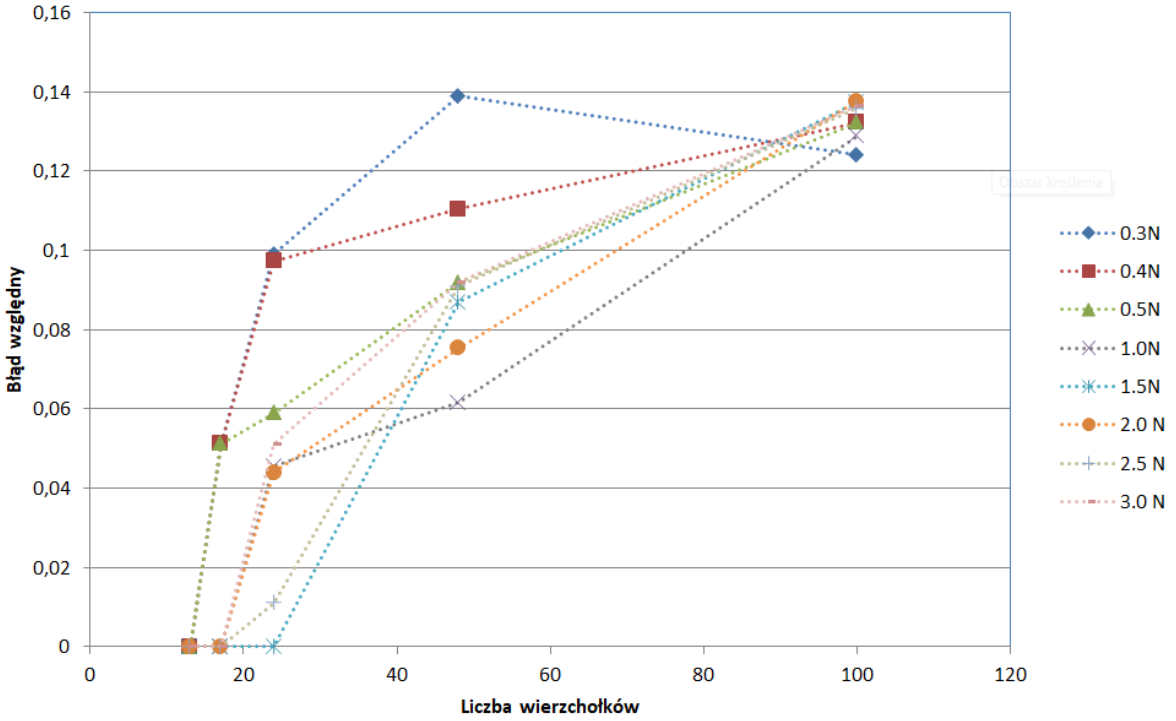
Procesor : Intel Xeon x5450, 4 rdzenie, 3.00 GHz

RAM: 8.00 GB

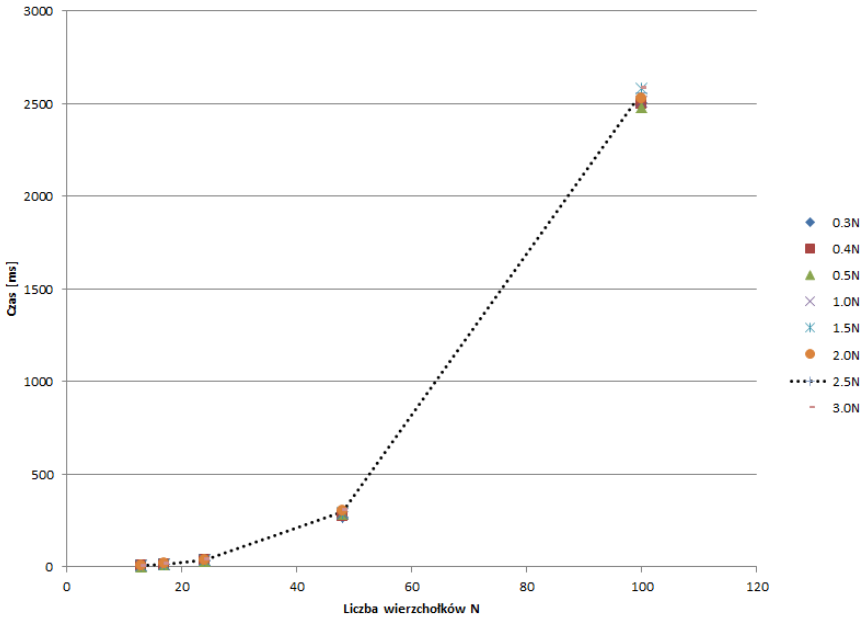
System: 64-bitowy system operacyjny, procesor x64

6. Wyniki

6.1 Wpływ długości Tabu na wyniki obliczeń

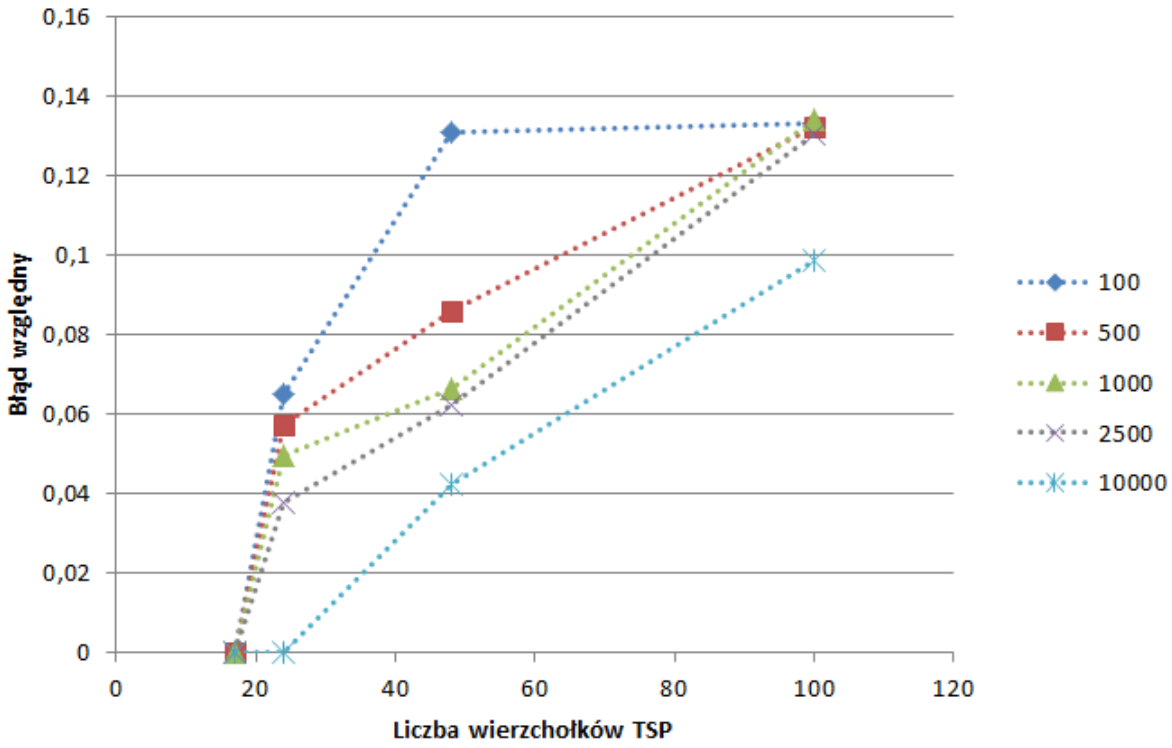


Rys 3. Wykres zależności błędu względnego od liczby wierzchołków dla danych długości listy Tabu

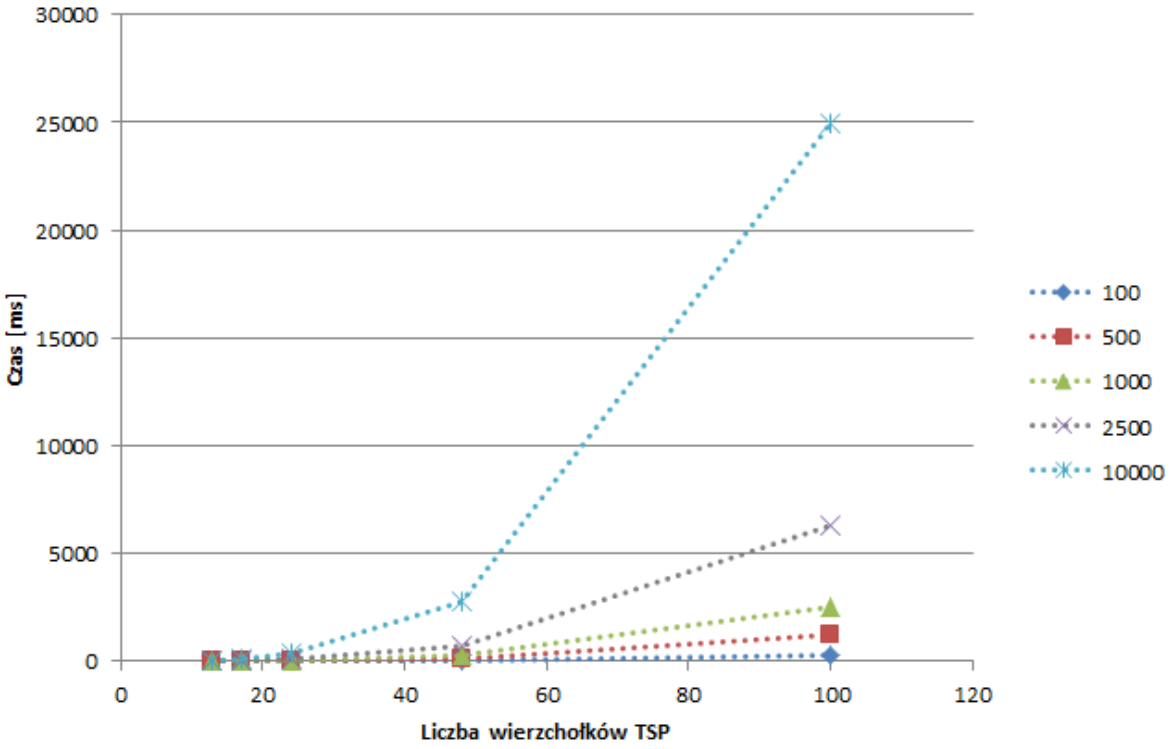


Rys 4. Wykres zależności czasu obliczeń od liczby wierzchołków dla danych długości listy Tabu

6.2 Wpływ maksymalnej liczby iteracji na wyniki obliczeń

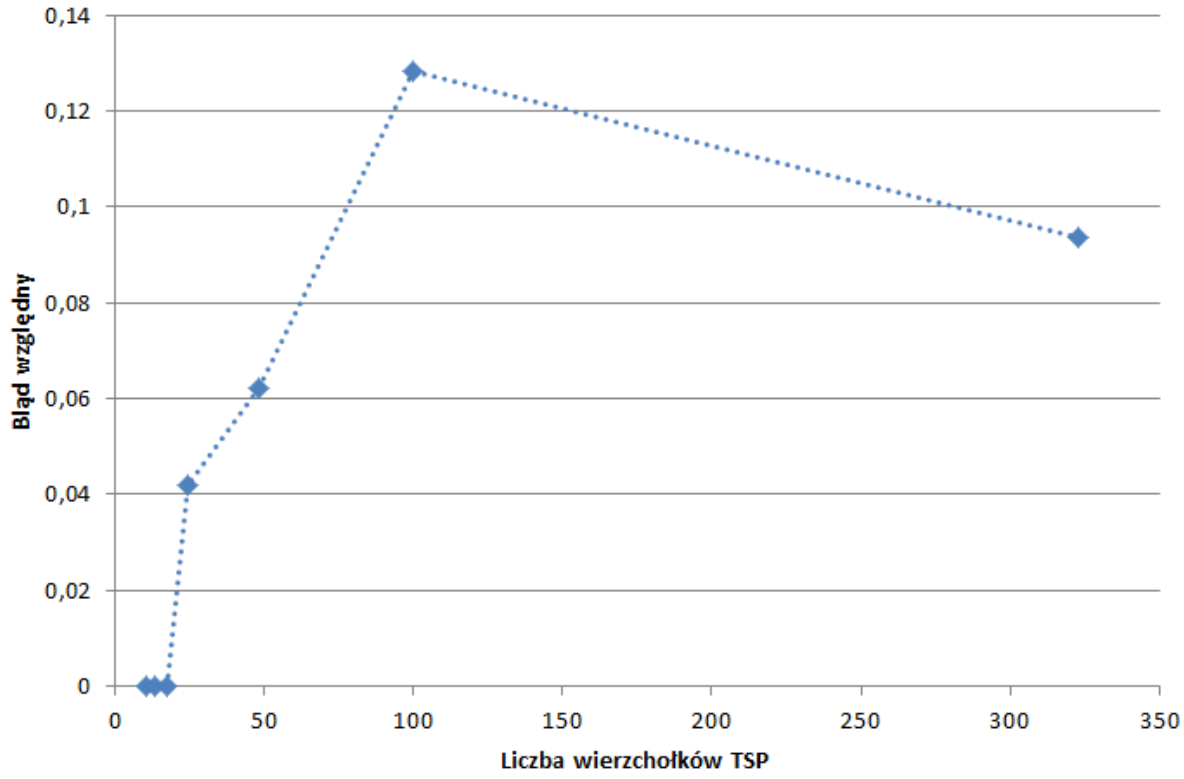


Rys 5. Wykres zależności błędu względnego od liczby wierzchołków dla danej liczby iteracji tabu Search

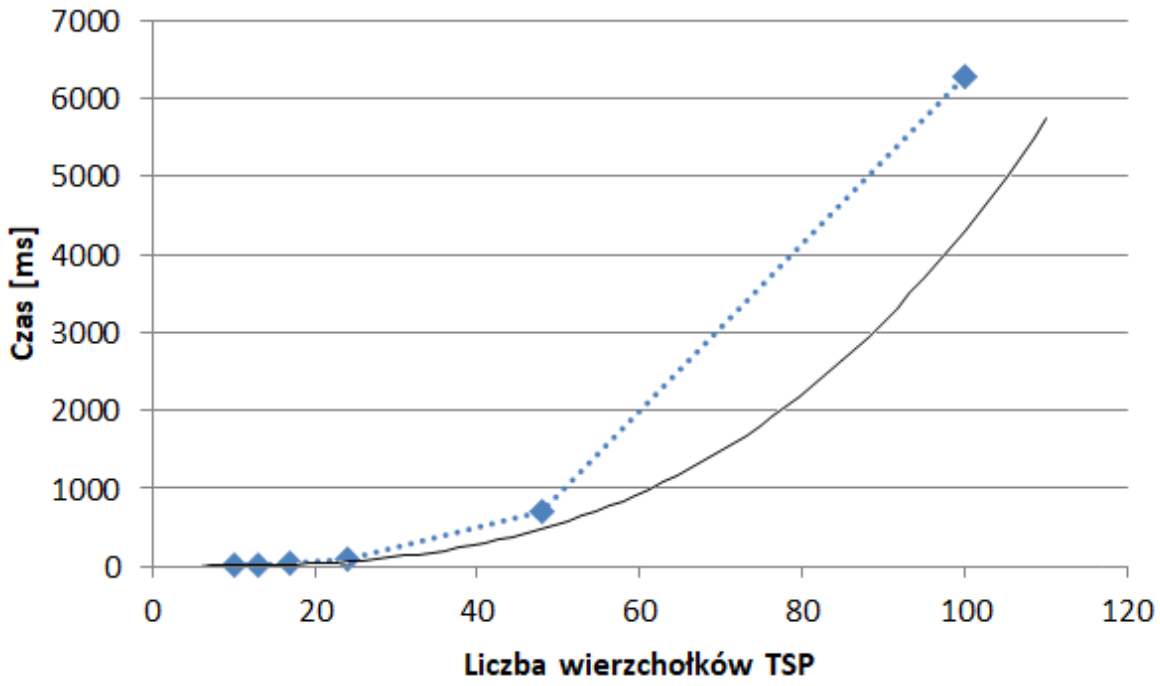


Rys 6. Wykres zależności czasu obliczeń od liczby wierzchołków dla danej liczby iteracji tabu Search

6.3 Wyniki dla najlepszych parametrów



Rys 7. Wykres zależności błędu względnego od liczby wierzchołków dla wybranych najlepszych parametrów



Rys 8. Wykres zależności czasu obliczeń od liczby wierzchołków dla wybranych najlepszych parametrów

Instancja	Rozmiar instancji	Wynik optymalna	Wynik otrzymany	Błąd względny [%]
tsp_10.txt	10	212	212	0,00
tsp_13.txt	13	269	269	0,00
tso_17.txt	17	39	39	0,00
gr24.txt	24	1272	1330	4,56
gr48.txt	48	5046	5359	6,20
kroA100.txt	100	21282	24097	13,22
rbg323.txt	323	1326	1450	9,35

Tab 1. Podsumowanie obliczeń dla ustalonych najlepszych parametrów

Ustalone parametry to:

rozmiaru listy ruchów zakazanych (tabu list): 1.0n

maksymalna liczba iteracji: 2500

długości kadencji: tabuSize

kryterium aspiracji: znalezienie wyniku lepszego niż najlepsze dotychczasowe rozwiązanie

sposobu wyboru rozwiązania początkowego: greedy search

sposób wyboru rozwiązań sąsiednich: Vertex Exchange

6.4 Wpływ wyboru rozwiązania początkowego na wyniki obliczeń

tsp_10.txt	10	212	212	0,00
tsp_13.txt	13	269	269	0,00
tso_17.txt	17	39	39	0,00
gr24.txt	24	1272	1317	3,54
gr48.txt	48	5046	5585	10,68
kroA100.txt	100	21282	29114	36,80
rbg323.txt	323	1326	1565	18,02

Tab 2. Podsumowanie obliczeń dla najlepszych parametrów przy losowym wyborze startowej instancji

7. Obserwacje i wnioski

Empirycznie sprawdzono jak długość listy Tabu wpływa na dokładność wyników (Rys 3.). Widzimy że krzywe te nie posiadają stałego trendu. Z początku dysproporcja między krzywymi jest duża. Oznacza to, że dla instancji o małej liczbie wierzchołków tabu search działa znacznie lepiej wraz z wydłużeniem tablicy tabu. Różnica ta szybko zaciera się i przestaje mieć znaczenie dla instancji o liczbie wierzchołków 100+. Na podstawie obserwacji ustalono, że najlepsze wyniki są zwracane dla listy tabu długości n.

Wydawałoby się, iż wydłużanie listy tabu odbije się na czasie działania programu. Bowiem w naszym O(n^3) lista tabu odpowiedzialna była za jedno z tych 3 n'ów w iloczynie. Zatem moglibyśmy przewidywać, że wraz ze wzrostem długości listy tabu proporcjonalnie wzrośnie czas wykonywania obliczeń. Jednak widzimy, iż tak się nie dzieje (Rys 4.). Trudno mi określić dokładnie jaki jest tego powód. Przypuszczam, że dzieje się to dlatego, iż nie dla każdego sąsiada przeszukujemy listę tabu. Listę tabu przeszukujemy w momencie znalezienia sąsiada o najlepszym koszcie. Widocznie znalezienie to jest wykonywane na tyle rzadko że nie wpływa znacząco na działanie algorytmu.

Widzimy, iż wraz ze wzrostem liczby iteracji do terminacji, wzrasta dokładność naszych obliczeń. (Rys 4.). Dzieje się to jednak kosztem znacznego pogorszenia czasu wykonywanych obliczeń. (Rys 5.) Widzimy, iż zlekceważony przez nas parametr k przy obliczaniu teoretycznej złożoności obliczeniowej ma duży wpływ na tą złożoność. Nie możemy traktować go jako stałej. Proporcjonalnie do ilości iteracji wzrasta czas wykonywania programu.

Na podstawie rysunku 4 oraz rysunku 5 ustalono, iż najlepszą liczbą iteracji będą 2.5 tysiąca. Przy liczbie tej widzimy znaczną poprawę dokładności obliczeń programu. Przy tym czas obliczeń nie rośnie jeszcze do zbyt dużych wartości i sięga około 6 sekund dla 100 wierzchołków. Czas ten jest do zaakceptowania, jednak 10 000 mogłoby być już problematyczne zważając na ilość powtórnych obliczeń na tych samych instancjach.

Na podstawie Rys 7, możemy dostrzec że błąd względny nie zależy bezpośrednio od wielkości instacnji. Istnieje między nimi pewna korelacja, jest to oczywiste, bowiem im większa instancja tym więcej potencjalnych minimów lokalnych oraz nasza lista tabu pokrywa mniejszy obszar. (bo lista tabu wydłuża się w skali n a lista sąsiadów jako n^2). Wykres zależy jednak także od ułożenia krawędzi w grafie, ilości minimów lokalnych, jego chaotyczności. Zatem nie da się przyrównać tego wykresu do żadnej krzywej z definicji.

Inaczej sprawa będzie się miała w przypadku wykresu zależności czasu obliczeń od wielkości instancji (Rys 8). Widzimy, że podobny jest on do krzywej n^3 umieszczonej na rysunku. Zatem wykres zachowuje się zgodnie z przewidywaniami (O(n^3)).

Finalnie możemy przyjrzeć się jeszcze jak duże znaczenie ma rozpoczęcie działania programu nie w losowym miejscu, a w heurystycznie obliczonym stanie. Błędy względne obliczeń są około 2 razy większe w przypadku nie zastosowania greedy searcha (Tabela 1., Tabela 2.).

Podsumowując, tabu search jest najbardziej wydajnym algorytmem pod względem czasu i pamięci z obecnie poznanych. Mówienie o czasie jednak jest trochę złudne, bo algorytm tak naprawdę nigdy nie kończy swojego działania. Mimo to, jest to bardzo dobra metoda do znajdowania wyników TSP obarczonych o błędy.