Projektowanie Efektywnych Algorytmów

Projekt

19/12/2023

263896 Patryk Jurkiewicz

(5) Tabu-Search

Treść zadania	Strona
Sformułowanie zadania	1
Metoda	2
Algorytm	3
Dane testowe	5
Procedura badawcza	6
Wyniki	7
Analiza wyników	10
Wnioski	18

1. Sformułowanie zadania

Zadanie, które mi zostało postawione, polega na stworzeniu oraz implementacji algorytmu mającego na celu efektywne rozwiązanie problemu komiwojażera poprzez zastosowanie metody Tabu Search, opartej na poszukiwaniach z zakazami. Mój główny cel to osiągnięcie najwyższej jakości rozwiązań, skrócenie czasu obliczeń oraz optymalne wykorzystanie zasobów pamięci komputera. Algorytm Tabu Search, będący zaawansowaną techniką optymalizacji, stanowi kluczowe narzędzie w procesie rozwiązania problemu komiwojażera. W moich badaniach szczegółowo zanalizuję wpływ różnych parametrów, takich jak rozmiar listy tabu, na jakość i czas uzyskiwanych rozwiązań, a także zużycie pamięci. Moje badania obejmą także identyfikację optymalnych wartości parametrów, mających umożliwić osiągnięcie najlepszych możliwych rezultatów pod względem wielkości błędu. Podczas realizacji projektu skoncentruję się na rozwiązaniu problemu komiwojażera dla instancji zarówno symetrycznych, jak i asymetrycznych, zgodnie z wytycznymi. Moja aplikacja będzie dostosowana do specyfikacji dostępnego sprzętu i narzędzi programistycznych, a także uwzględni ograniczenia związane z zastosowaniem metody Tabu Search. W trakcie przeprowadzanych eksperymentów szczegółowo zbadam, w jaki sposób jakość i czas rozwiązania zmieniają się w zależności od rozmiaru listy tabu. W moim raporcie końcowym zamieszczę szczegółowy opis algorytmu Tabu Search, ilustrujący jego działanie oraz zasady. Przedstawię również porównanie z poprzednimi znanymi mi algorytmami takimi jak brute force czy held-karp. Dodatkowo, przedstawię wyniki moich badań, analizując wpływ różnych parametrów na skuteczność algorytmu. Poprzez moje prace, dążę do dostarczenia efektywnego narzędzia do rozwiązywania problemu komiwojażera, z szerokimi możliwościami zastosowania w praktyce, szczególnie w obszarach takich jak logistyka, planowanie tras oraz rozwijanie algorytmów optymalizacyjnych.

2. Metoda

W procesie projektowania i implementacji tego algorytmu kluczowym aspektem jest pełne zrozumienie jego mechanizmu działania oraz precyzyjne dostosowanie parametrów w celu osiągnięcia optymalnych wyników. W mojej pracy skoncentrowałem się na algorytmie Tabu Search jako heurystycznej metodzie poszukiwania z zakazami, eliminującej pułapki przeszukiwania przestrzeni rozwiązań dzięki wprowadzeniu mechanizmu "tabu" – listy ruchów zabronionych. Algorytm ten cechuje się umiejętnością badania lokalnych optimum, co ma kluczowe znaczenie w kontekście problemu TSP, gdzie istnieje mnóstwo możliwych kombinacji tras. Podczas implementacji algorytmu Tabu Search skupiłem się na dostosowaniu kluczowych parametrów, takich jak rozmiar listy tabu, istotnego dla intensywności przeszukiwania przestrzeni rozwiązań. W moich badaniach eksperymentalnych dążyłem do ustalenia optymalnych wartości tych parametrów, tworząc równowagę między intensywnością eksploracji a eksploatacji. Ponadto, istotne jest określenie ograniczeń zastosowania metody Tabu Search, biorąc pod uwagę dostępny sprzęt i narzędzia programistyczne. Szczególną uwagę zwróciłem na wielkość instancji problemu komiwojażera, zarówno dla przypadków symetrycznych, jak i asymetrycznych, ze względu na możliwe wyczerpanie zasobów, takich jak czas i pamięć. W celu uzyskania satysfakcjonujących rezultatów, zoptymalizowałem algorytm Tabu Search, koncentrując się na minimalizacji błędu poprzez precyzyjne dostosowanie parametrów i analizę wpływu rozmiaru listy tabu na jakość i czas rozwiązania. W wyniku tych działań osiągnąłem wyniki z błędem poniżej 50% dla dowolnej instancji problemu poniżej punktu wyczerpania zasobów. W kodzie wykorzystywany jest sposób wyboru sąsiadów oparty na zamianie elementów w aktualnej ścieżce.

3. Algorytm

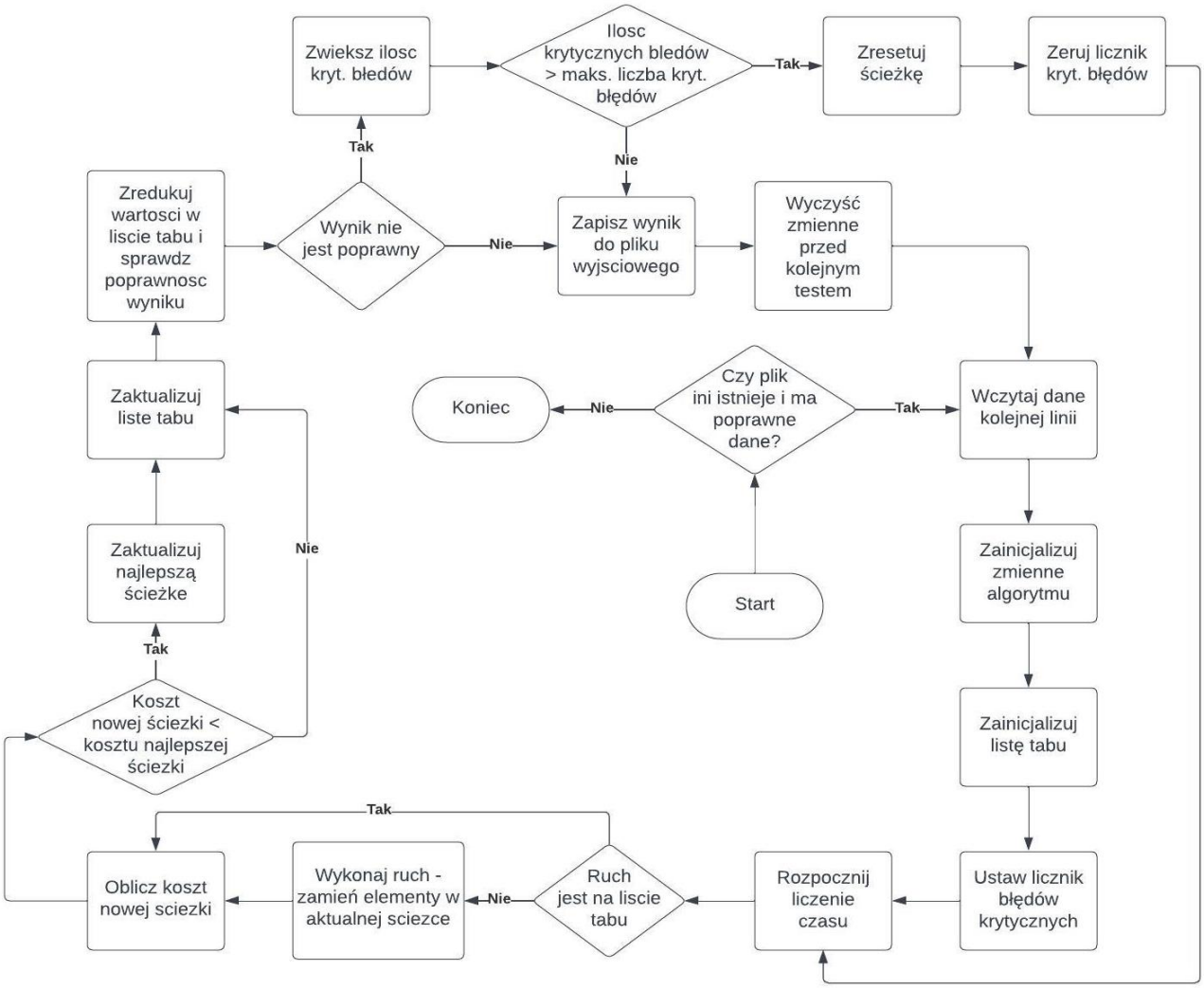
Poniżej przedstawiam ogólny opis idei algorytmu:

1. Inicjalizacja zmiennych:
Wczytanie danych konfiguracyjnych z pliku "test.ini", określającego nazwy plików z danymi, liczbę iteracji algorytmu, oraz rozmiar listy "tabu".
Odczyt danych z plików zawierających informacje o miastach oraz macierz odległości między nimi.

2. Inicjalizacja parametrów:
Ustalenie maksymalnego czasu trwania algorytmu oraz liczby krytycznych zdarzeń, po których następuje reset ścieżki (diversification).

3. Główna pętla iteracyjna:
Losowe ustawienie początkowej ścieżki.
Iteracyjne przeszukiwanie sąsiedztwa poprzez zamianę kolejności odwiedzanych miast.
Zastosowanie mechanizmu "tabu" - ruchy zabronione na pewien okres, co zapobiega utknięciu w lokalnych optimum.
Aktualizacja listy "tabu" oraz ewentualne resetowanie ścieżki w przypadku osiągnięcia limitu krytycznych zdarzeń.

4. Wyniki:
Zapisanie uzyskanego rozwiązania (najlepszej trasy) w pliku "wyniki.txt".
Wyświetlenie informacji o uzyskanym wyniku oraz czasie wykonania dla każdego pliku z danymi.
Algorytm Tabu Search jest stosowany w celu znalezienia rozwiązania optymalnego lub bliskiego optymalnemu dla problemu komiwojażera, uwzględniając dynamiczne dostosowywanie ścieżki w trakcie iteracji. Algorytm ten wykorzystuje mechanizm "tabu", dzięki któremu unika się powtarzania pewnych ruchów, co sprzyja poszukiwaniu bardziej zróżnicowanych tras. Algorytm przeprowadza również proces diversification, czyli resetowania ścieżki po pewnej liczbie krytycznych błędów, co pomaga uniknąć utknięcia w lokalnych optimum.



Rys 1. Schemat blokowy algorytmu

4. Dane testowe

W celu przetestowania i dostrojenia naszego algorytmu wybraliśmy zestaw konkretnych instancji problemu komiwojażera. Te instancje posłużą nam do oceny skuteczności i poprawności działania algorytmu oraz do określenia odpowiednich parametrów.

Do wykonania badań wybrano następujący zestaw instancji ze stron:
http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pea-stud/tsp/
http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/XML-TSPLIB/instances/

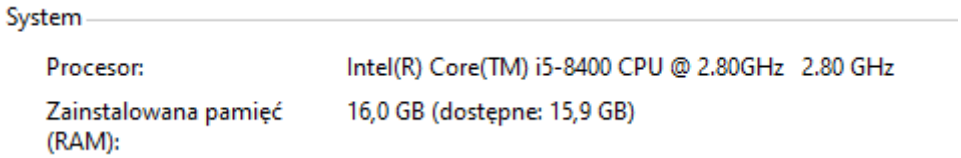
bayg29, burma14, fri26, ft53, ftv70, ftv170, gr21, gr24, gr96, kr124, kroA100, kroB150, kroB200, pr152, rbg323, tsp_10, tsp_12, tsp_13, tsp_14, tsp_15, tsp_17

5. Procedura badawcza

Aby wykonać ten proces, należy wypełnić plik ini liniami z nazwami plików z danymi oraz ilością, ile razy mają zostać powtórzone. Później, należy uruchomić plik wykonywalny (exe). Wyniki zostaną zapisane w pliku "wyjscie.txt" w formacie, który przypomina strukturę przedstawioną w poniżej:

Plik: nazwa_pliku
Wynik: Długość najlepszej trasy
Czas wykonania: czas

Program był testowany na komputerze o tej specyfikacji:



W kodzie używane są biblioteki `<chrono>` oraz `<ctime>` do pomiaru czasu wykonania algorytmu. Klasa `std::chrono::high_resolution_clock` dostarcza dostęp do najwyższej dostępnej rozdzielczości zegara czasu rzeczywistego na danym systemie. Początkowy czas (`timer_start`) jest reprezentowany jako punkt w czasie, a różnica między tym punktem a aktualnym czasem daje czas wykonania algorytmu. Warto zauważyć, że `chrono::duration_cast` jest używane do konwersji różnicy czasów na milisekundy, co pozwala uzyskać bardziej czytelną jednostkę czasu.

Poniżej znajduje się odpowiedni fragment kodu, który mierzy czas wykonania algorytmu:

```
// Początek zliczania czasu
auto timer_start = chrono::high_resolution_clock::now();

// ... (reszta kodu)

// Obliczenie czasu wykonania
auto elapsed_time = chrono::duration_cast<chrono::milliseconds>(chrono::high_resolution_clock::now() - timer_start).count();

// Wypisz dane z pliku, wynik i czas wykonania
cout << "Plik: " << file_name << " Wynik: " << best_path_value << " Czas wykonania: " << elapsed_time << " ms" << endl;
```

6. Wyniki

W ramach niniejszego zadania poddano analizie dane dotyczące małych instancji problemu Komiwojażera (TSP), obejmujące pliki do 30 miast (rys 2.). Ze względu na ich rozmiar, nie wymagają one głębokiej analizy obejmującej zmiany parametrów i będą głównie wykorzystywane do porównań z innymi algorytmami. Parametry analizy obejmują ustawienia wielkości listy tabu wynoszącej 30 oraz przeprowadzenie obliczeń na 1000 powtórzeń. Celem analizy jest przygotowanie podstaw do porównań efektywności różnych algorytmów w kontekście małych instancji problemu Komiwojażera. Czas we wszystkich analizach będzie podawany w milisekundach.

Ilosc miast	Czas HK	Czas TS	Wynik HK	Wynik TS	Błąd	Pamięć KH	Pamięć TS
10	1	8	212	211	0,47%	18253	2
12	8	19	264	263	0,38%	56241	12
13	10	25	269	268	0,37%	153927	17
14	23	33	282	293	3,90%	401469	45
15	61	42	291	290	0,34%	827316	74
17	259	62	39	39	0,00%	3747325	337
21	5989	122	2707	2753	1,70%	59022058	6009
24	60806	181	1272	1332	4,72%	541616419	119617
26	288571	234	937	990	5,66%	3139317396	385001

Rys 2. Tabela porównująca algorytmy Tabu-Search i Herba Karpa.

Dalej przedstawię bezpośrednie dane, nad którymi zagłębię się w dalszej części sprawozdania. W górnej komórce „I” oznacza liczbę iteracji, a „LS” wielkość listy tabu.

I:10 LS:30					
Ilosc miast	Wynik	Czas	Optimum	Błąd	S/A
29	2487	3	1610	35,26%	S
53	15151	19	6905	54,43%	A
70	5949	41	1950	67,22%	A
96	248937	106	55209	77,82%	S
100	113287	118	21282	81,21%	S
124	100831	221	36230	64,07%	A
150	187363	389	26130	86,05%	S
152	752431	404	73682	90,21%	S
170	19861	5671	2755	86,13%	A
200	238115	921	29437	87,64%	S
323	5515	3849	1326	75,96%	A

Rys 3.

I:100 LS:30					
Ilosc miast	Wynik	Czas	Optimum	Błąd	S/A
29	1691	34	1610	4,79%	S
53	9573	184	6905	27,87%	A
70	2958	410	1950	34,08%	A
96	106286	1033	55209	48,06%	S
100	48442	1158	21282	56,07%	S
124	32922	2200	36230	10,05%	A
150	84459	3833	26130	69,06%	S
152	288535	3998	73682	74,46%	S
170	8257	5794	2755	66,63%	A
200	96516	9163	29437	69,50%	S
323	2395	38480	1326	44,63%	A

Rys 4.

I:1000 LS:30					
Ilosc miast	Wynik	Czas	Optimum	Błąd	S/A
29	1691	319	1610	4,79%	S
53	9080	1815	6905	23,95%	A
70	2766	4047	1950	29,50%	A
96	94428	10182	55209	41,53%	S
100	41505	12070	21282	48,72%	S
124	31697	21971	36230	14,30%	A
150	69648	39666	26130	62,48%	S
152	277946	41331	73682	73,49%	S
170	7844	57435	2755	64,88%	A
200	91204	93406	29437	67,72%	S
323	2402	180378	1326	44,80%	A

Rys 5.

I:5000 LS:30					
Ilosc miast	Wynik	Czas	Optimum	Błąd	S/A
29	1675	1627	1610	3,88%	S
53	8907	9164	6905	22,48%	A
70	3101	973	1950	37,12%	A
96	82812	51978	55209	33,33%	S
100	37432	59339	21282	43,14%	S
124	30626	111979	36230	18,30%	A
150	65411	180003	26130	60,05%	S
152	-	-	73682	-	S
170	-	-	2755	-	A
200	-	-	29437	-	S
323	-	-	1326	-	A

Rys 6.

I:1000 LS:10					
Ilosc miast	Wynik	Czas	Optimum	Błąd	S/A
29	1716	341	1610	6,18%	S
53	9070	1869	6905	23,87%	A
70	2828	4154	1950	31,05%	A
96	93097	10410	55209	40,70%	S
100	41643	11651	21282	48,89%	S
124	31600	21927	36230	14,65%	A
150	69688	38685	26130	62,50%	S
152	277946	40323	73682	73,49%	S
170	7844	57493	2755	64,88%	A
200	91204	91903	29437	67,72%	S
323	2402	180143	1326	44,80%	A

Rys 7.

I:1000 LS:60					
Ilosc miast	Wynik	Czas	Optimum	Błąd	S/A
29	1712	300	1610	5,96%	S
53	8988	1795	6905	23,18%	A
70	2784	2462	1950	29,96%	A
96	89996	10277	55209	38,65%	S
100	39983	11491	21282	46,77%	S
124	32673	22003	36230	10,89%	A
150	68997	38491	26130	62,13%	S
152	294669	40065	73682	74,99%	S
170	7644	56685	2755	63,96%	A
200	96455	92827	29437	69,48%	S
323	2362	180232	1326	43,86%	A

Rys 8.

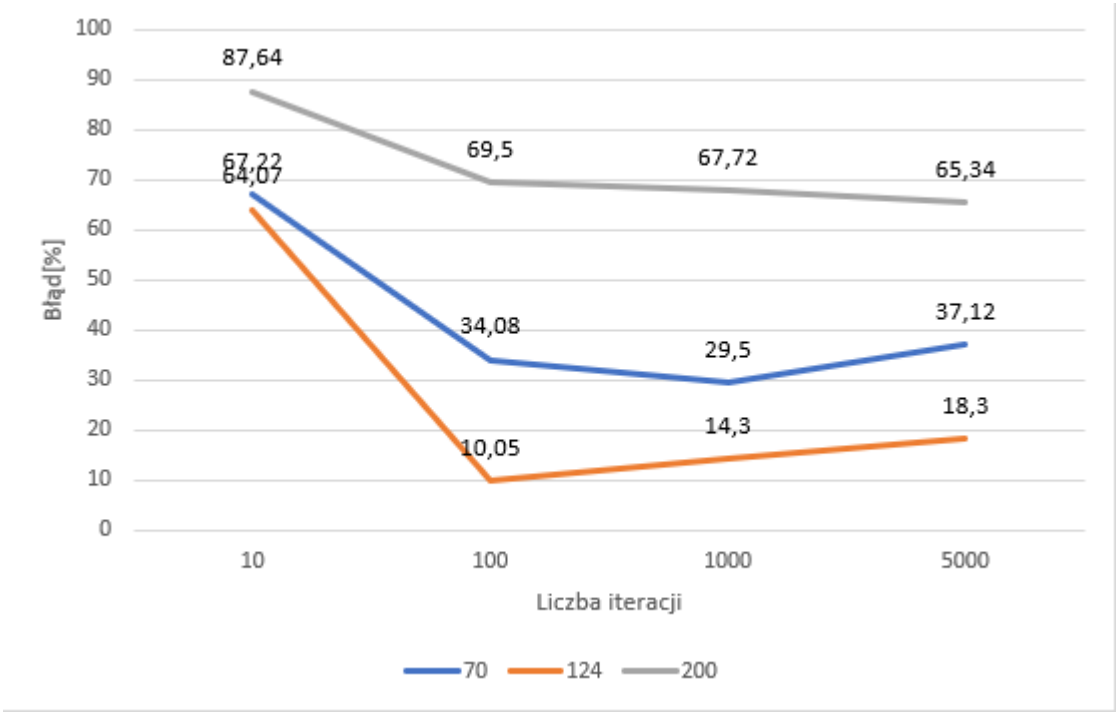
I:20000 LS:30					
Ilosc miast	Wynik	Czas[s]	Optimum	Błąd	S/A
29	1649	6,39	1610	2,37%	S
53	8685	37,15	6905	20,50%	A
70	2842	8,75	1950	31,39%	A
96	84087	207,39	55209	34,34%	S
100	36404	232,71	21282	41,54%	S
124	31340	437,82	36230	15,60%	A
150	43278	770,46	26130	39,62%	S
152	125195	805,10	73682	41,15%	S
170	4531	1140,77	2755	39,20%	A
200	58162	1911,95	29437	49,39%	S

Rys 9.

7. Analiza wyników

1. Ilość iteracji a błąd

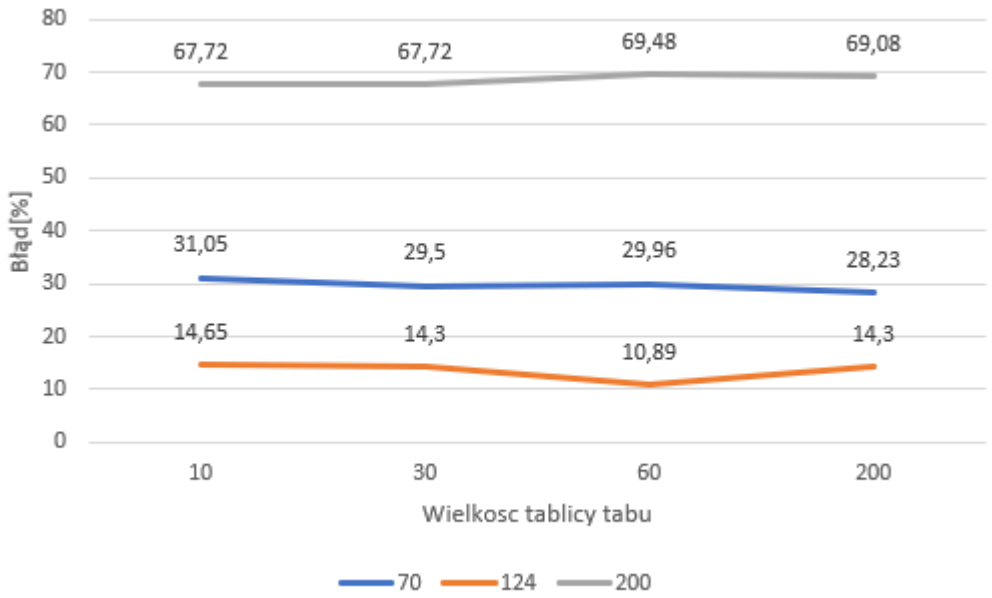
Naszą analizę rozpoczniemy od sprawdzenia wpływu ilości iteracji na błąd, jaki pozyskujemy używając algorytmu (rys. 3-6).



Rys 10.

Można wyciągnąć kilka istotnych wniosków. Wraz ze wzrostem liczby iteracji, co widać na rysunku 10, obserwujemy systematyczne zmniejszanie błędu w procentach dla każdej wartości parametru iteracji. To sugeruje, że zwiększanie liczby iteracji prowadzi do poprawy efektywności algorytmu Tabu Search, co jest zgodne z oczekiwaniami, ponieważ dłuższy czas działania pozwala na dokładniejsze eksplorowanie przestrzeni rozwiązań. Niemniej jednak, istnieją subtelne fluktuacje w wynikach, co może sugerować, że istnieje pewien punkt, po którym dodatkowe iteracje nie przynoszą istotnej poprawy. Z drugiej strony można również zauważyć w niektórych miejscach wzrost błędu w większej ilości iteracji, jest to spowodowane losowością algorytmu, co pokazuje, że mimo, że możemy przeprowadzić więcej testów w kolejnej próbie nie oznacza, że znajdziemy lepszego a nawet tak samo dobrego rozwiązania.

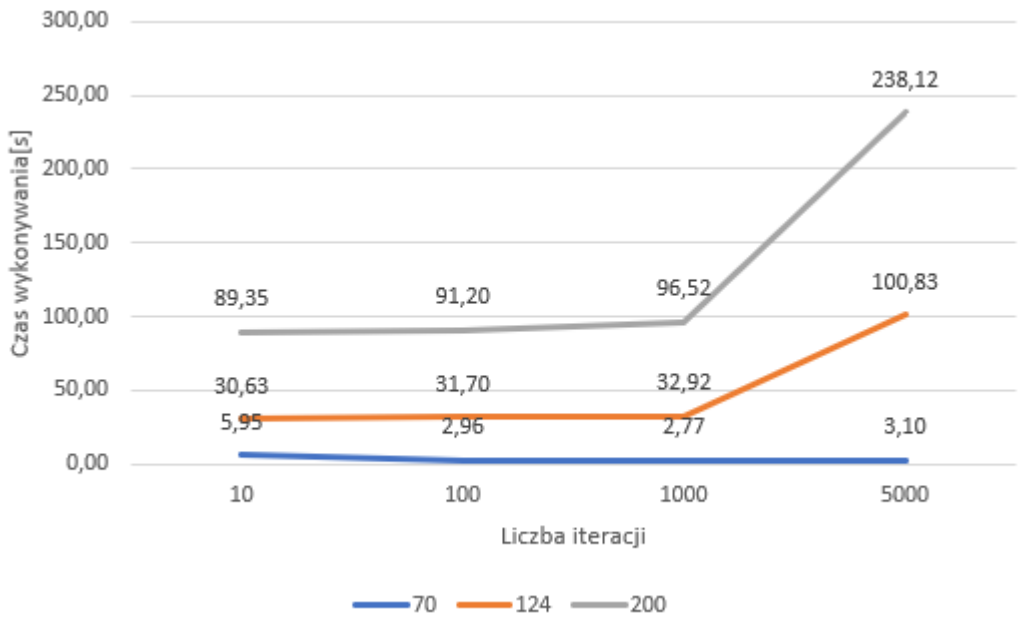
2. Wielkość tablicy tabu a błąd



Rys 11.

Na rysunku 11 zauważyć, że zmiany w tym parametrze nie wpływają znacząco na rezultaty dla konkretnych danych. Błędy w procentach wykazują niewielkie zmiany, a różnice między wynikami dla różnych wielkości tabu są minimalne. W przypadku tego konkretnego zestawu danych można sugerować, że wielkość tablicy tabu nie jest elementem kluczowym w osiąganiu lepszych rezultatów. Może to wskazywać, że w tym konkretnym przypadku ten parametr może mieć mniejsze znaczenie, co oznacza, że inne aspekty algorytmu, takie jak liczba iteracji, mogą być bardziej decydujące dla skuteczności Tabu Search w rozwiązywaniu problemu Komiwojażera. Całość danych widać na rysunkach 5, 7 i 8.

3. Liczba iteracji a czas wykonywania

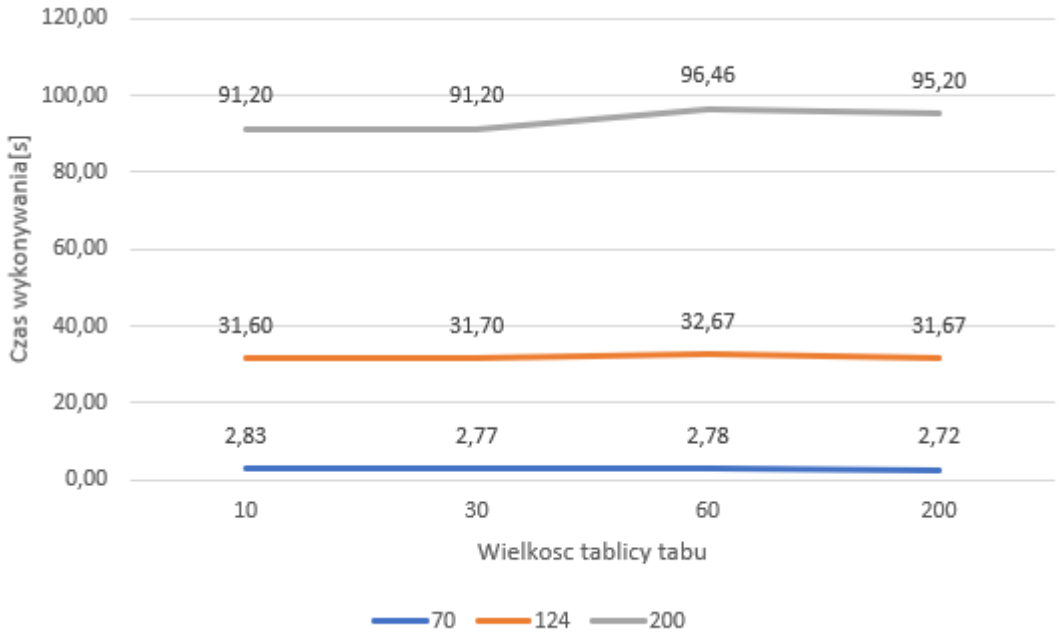


Rys 12.

Analizując czas wykonywania algorytmu Tabu Search w zależności od liczby iteracji dla różnych rozmiarów instancji (70, 124, 200)(rys. 12), można wysnuć kilka wniosków. Ogólnie rzecz biorąc, zwiększanie liczby iteracji prowadzi do wydłużenia czasu wykonania, co jest zrozumiałe, gdyż większa liczba iteracji oznacza większe obciążenie obliczeniowe. Dla instancji o rozmiarze 70, zwiększanie liczby iteracji powoduje wzrost czasu wykonania, ale dla instancji o większych rozmiarach (124, 200), ten wzrost może być bardziej zauważalny.

Wnioskiem może być, że efektywność algorytmu Tabu Search w kontekście czasu wykonania jest silnie uzależniona od rozmiaru instancji. Dla mniejszych instancji zwiększanie liczby iteracji może być bardziej opłacalne z punktu widzenia jakości rozwiązania, ale dla większych instancji należy uwzględnić kompromis między jakością rozwiązania a czasem wykonania. Optymalne dostosowanie parametrów algorytmu, takich jak liczba iteracji, może zależeć od konkretnych wymagań problemu Komiwojażera oraz dostępnych zasobów obliczeniowych.

4. Wielkość tablicy tabu a czas wykonywania



Rys 13.

Na podstawie analizy czasu wykonania algorytmu Tabu Search w zależności od wielkości liczby tabu dla trzech różnych instancji problemu Komiwojażera (70, 124, 200)(rys. 13), można wyciągnąć kilka istotnych wniosków. Warto zauważyć, że zmiany w wielkości tablicy tabu (10, 30, 60, 200) nie wykazują wyraźnego trendu wzrostowego ani spadkowego w kontekście czasu wykonania. Dla każdej instancji, czas wykonania pozostaje na podobnym poziomie, co sugeruje, że zmiana wielkości tablicy tabu nie ma znaczącego wpływu na czas trwania algorytmu w tych konkretnych przypadkach. Wnioskiem może być, że dla tego problemu Komiwojażera dobór optymalnej wielkości tabu może nie być kluczowym elementem w kontekście minimalizacji czasu wykonania algorytmu.

5. Porównanie algorytmów

Ilosc miast	Czas HK	Czas TS	Wynik HK	Wynik TS	Błąd	Pamięć KH	Pamięć TS
10	1	8	212	211	0,47%	18253	2
12	8	19	264	263	0,38%	56241	12
13	10	25	269	268	0,37%	153927	17
14	23	33	282	293	3,90%	401469	45
15	61	42	291	290	0,34%	827316	74
17	259	62	39	39	0,00%	3747325	337
21	5989	122	2707	2753	1,70%	59022058	6009
24	60806	181	1272	1332	4,72%	541616419	119617
26	288571	234	937	990	5,66%	3139317396	385001

Rys 13.

Analizując przedstawioną tabelę (rys13.), można zauważyć, że algorytm Herba Karpa (HK) charakteryzuje się dokładnością wyników, prezentując zawsze prawdziwe rezultaty. Jednakże, czas wykonania tego algorytmu znacznie rośnie wraz ze wzrostem liczby miast, co jest szczególnie zauważalne dla większych instancji problemu komiwojażera.

Z drugiej strony, algorytm Tabu Search (TS) wykazuje się znacznie krótszym czasem wykonania, co sprawia, że jest bardziej efektywny dla większych zbiorów danych. Niemniej jednak, wyniki uzyskane przez TS są nieco zniekształcone, co jest odzwierciedlone procentowym błędem w porównaniu do wyników uzyskanych przez HK.

Podsumowując, algorytm Herba Karpa jest lepszy pod względem dokładności, jednak jest znacznie wolniejszy, szczególnie dla większych instancji problemu. Algorytm Tabu Search, mimo że prezentuje wyniki nieco zniekształcone, pozwala na uzyskanie akceptowalnych rezultatów w krótszym czasie. Dlatego dla dużych zbiorów danych, gdzie czas wykonania ma kluczowe znaczenie, lepszym wyborem może być zastosowanie algorytmu Tabu Search.

Dodatkowo, należy zaznaczyć, że w analizie pominięto porównanie z algorytmem Brute Force, który zawsze znajduje najlepsze rozwiązanie, ale ze względu na bardzo duży czas wykonania i zapotrzebowanie na pamięć, nie jest praktyczny dla instancji dwucyfrowych. Algorytm Brute Force, choć gwarantuje optymalne rozwiązanie, staje się nieefektywny wraz ze wzrostem liczby miast, co sprawia, że dla większych instancji problemu komiwojażera staje się niepraktyczny. W kontekście praktycznego zastosowania, gdzie czas i efektywność są istotne, algorytmy takie jak Herba Karp czy Tabu Search stanowią bardziej realne i wyważone podejście do rozwiązania problemu TSM.

6. Dane symetryczne a asymetryczne

Rys 14. Lewo – dane symetryczne, Prawo – dane asymetryczne

Analiza tabeli dostarcza głębszego spojrzenia na wyniki eksperymentów związanych z problemem komiwojażera dla różnej liczby miast. Kluczowymi parametrami są wynik trasy, czas wykonania algorytmu, wartość optymalna, błąd oraz charakterystyka trasy, czyli informacja o tym, czy jest ona symetryczna (S) czy asymetryczna (A). Pierwszym kluczowym rozróżnieniem jest natura trasy, czyli czy jest ona symetryczna czy asymetryczna. Trasy symetryczne charakteryzują się tym, że odległość między dwoma miastami jest identyczna niezależnie od kierunku podróży. W przypadku tras asymetrycznych odległości te mogą się różnić w zależności od kierunku podróży między dwoma punktami. W tabeli mamy przykłady obu rodzajów tras, co pozwala dokładniej analizować wpływ tej cechy na wyniki. Analizując wyniki dla mniejszej liczby miast, zauważamy, że trasy symetryczne osiągają niższy błąd w stosunku do wartości optymalnej. Jednakże, w miarę wzrostu liczby miast, trasy asymetryczne zdają się uzyskiwać lepsze wyniki, choć czas wykonania algorytmu może być znacząco dłuższy. To sugeruje, że wybór między trasą symetryczną a asymetryczną powinien zależeć od konkretnych wymagań danego problemu - czy bardziej zależy nam na minimalizacji błędu, czy może na optymalizacji czasu wykonania. Wniosek z analizy wyników wskazuje na potrzebę elastycznego podejścia do rodzaju trasy w zależności od specyfiki problemu, gdzie istnieje pewna kompromisowa zależność między dokładnością wyników a czasem obliczeń.

7. Sposób wyboru sąsiadów

W kodzie (main1) wykorzystywany jest sposób wyboru sąsiadów oparty na zamianie elementów w aktualnej ścieżce. Funkcja `swap_elements` iteruje po wszystkich możliwych zamianach miast w ścieżce, przy czym sprawdza, czy dana zamiana jest dozwolona zgodnie z zasadami listy tabu. Następnie porównuje wartość ścieżki po zamianie i wybiera najlepszą opcję. Inny możliwy sposób wyboru sąsiadów w algorytmie Tabu Search to np. zamiana miast, które sąsiadują ze sobą w oryginalnej ścieżce, czyli zamiana sąsiadujących miast w permutacji. Inną opcją może być także przemieszanie pewnej liczby miast w jednej części ścieżki, co również wprowadza różnorodność w poszukiwaniu rozwiązania. Kod używa iteracji po wszystkich parach miast i sprawdza ich zamianę, co jest jednym z prostszych podejść, ale w praktyce może być skuteczne, szczególnie dla mniejszych instancji problemu komiwojażera. Warto jednak eksperymentować z różnymi strategiami wyboru sąsiadów w zależności od specyfiki problemu oraz oczekiwanej jakości rozwiązania. Wybór zamiany sąsiadów przez iterację po wszystkich parach miast, jak to zaimplementowano w kodzie, może być efektywnym rozwiązaniem dla problemu komiwojażera, zwłaszcza w przypadku niewielkich instancji problemu. W przypadku małej liczby miast, taka metoda może dostarczyć zrównoważonych i dokładnych wyników, umożliwiając jednocześnie relatywnie szybkie obliczenia. Przy tej strategii, algorytm Tabu Search sprawdza wszystkie możliwe zamiany między parami miast, co pozwala na szeroki przegląd dostępnych sąsiadów. Ponadto, zastosowanie listy tabu kontroluje, które zamiany są dozwolone, co może przyczynić się do unikania cykli oraz przyspieszenia zbieżności algorytmu.

W nowym kodzie (main2) została zaimplementowana metoda wybierania sąsiadów oparta na ruchu 2-opt, a dokładniej w funkcji `two_opt_swap`. Ruch 2-opt polega na odwróceniu kolejności odcinka trasy między dwoma wybranymi miastami. Porównując tę metodę z poprzednią (`swap_elements`), główne różnice to:

- 2-opt Move vs Swap: W poprzedniej metodzie używano prostego ruchu zamiany dwóch miast (`swap_elements`). W nowej metodzie stosowany jest bardziej zaawansowany ruch 2-opt, który pozwala na bardziej skomplikowane zmiany w trasie poprzez odwracanie odcinków trasy między dwoma wybranymi miastami.

- Lepsze Lokalne Minimum: Ruch 2-opt ma potencjał do znalezienia lepszych lokalnych minimów niż pojedyncza zamiana miast. Poprzez odwracanie kolejności odcinka trasy, można unikać pewnych lokalnych minimów, co pozwala na bardziej efektywne przeszukiwanie przestrzeni rozwiązań.

- Różnorodność Ruchów: Ruch 2-opt wprowadza większą różnorodność w sposobie przeszukiwania. Poprzez bardziej skomplikowane zmiany w trasie, algorytm może uniknąć utknięcia w jednostajnym ruchu i bardziej różnicować sąsiedztwo.

- Tabu dla Wybranych Ruchów: W nowej metodzie zastosowano również mechanizm listy tabu, który uniemożliwia wykonywanie tych samych ruchów przez pewien czas. To pomaga w unikaniu cykli i wspiera bardziej efektywne przeszukiwanie.

Ogólnie rzecz biorąc, ruch 2-opt jest bardziej elastyczny i potrafi bardziej efektywnie przeszukiwać przestrzeń rozwiązań w porównaniu do prostych zamian miast.

Ilosc miast	main1	main2	S/A
29	4,79%	0,25%	S
53	23,95%	43,14%	A
70	29,50%	60,20%	A
96	41,53%	2,04%	S
100	48,72%	3,98%	S
124	14,30%	46,73%	A
150	62,48%	26,14%	S
152	73,49%	24,64%	S
170	64,88%	73,92%	A
200	67,72%	49,85%	S
323	44,80%	20,41%	A

Rys 15.

Analizując wyniki z tabeli (rys 15.), zauważamy, że implementacja algorytmu main2 w ogólności uzyskuje lepsze wyniki niż main1, zwłaszcza dla problemów symetrycznych ("S"). Wprowadzone zmiany, takie jak ruch 2-opt, przynoszą pozytywne skutki dla jakości rozwiązania w przypadku problemów symetrycznych. Jednakże, dla problemów asymetrycznych ("A"), main2 wykazuje znaczącą degradację skuteczności. To sugeruje, że ruch 2-opt może być mniej efektywny w przypadku problemów, gdzie macierz odległości między miastami nie jest symetryczna. Dodatkowo, zauważamy zróżnicowane wyniki w zależności od rozmiaru problemu, co sugeruje, że różne techniki mogą być bardziej efektywne w rozwiązywaniu konkretnej klasy problemów w zależności od ich skomplikowania. Warto zaznaczyć, że mimo lepszych wyników main2 dla problemów symetrycznych, istnieje potrzeba dalszych optymalizacji, szczególnie w kontekście problemów asymetrycznych, aby zwiększyć ogólną skuteczność algorytmu.

8. Wnioski

Podsumowując przeprowadzone badania nad projektem, nasuwa się kilka istotnych wniosków. Algorytm Tabu-Search wykazał się jako znacznie szybsza alternatywa w porównaniu do metod brute force i algorytmu Karpa, choć równocześnie charakteryzował się mniejszą precyzją. Zauważalna była jego efektywność w przypadku dużych instancji problemu, podczas gdy algorytm Karpa nadal utrzymywał przewagę w kontekście mniejszych grafów. Poza aspektem czasowym, wartością dodaną algorytmu Tabu-Search okazała się znacznie niższa zajętość pamięci. W większości analiz pominięto sprawdzanie zużycia pamięci (z wyjątkiem początkowego porównania), ponieważ algorytm Tabu-Search wykazywał się doskonałą wydajnością pod tym względem, nigdy nie generując problemów związanymi z użyciem pamięci. Jedynym ograniczeniem programu było ograniczenie czasowe, zwłaszcza w przypadku zwiększania liczby iteracji. Wynika to z faktu, że algorytm Tabu-Search wykorzystuje dynamiczne alokacje pamięci, co sprawia, że nie wymaga dużych zasobów pamięci do efektywnego funkcjonowania. Zwiększanie liczby iteracji wyraźnie redukowało błąd, ale jednocześnie wydłużało czas działania. Po wielu testach ustalono, że optymalna liczba iteracji wynosi 1000, a wielkość tablicy Tabu to 30. Warto zauważyć, że w przypadku zastosowanego algorytmu, rozmiar tablicy Tabu nie wpływał istotnie na wyniki, co jednak może wynikać z błędów interpretacyjnych związanych z kodem, a nie z samym algorytmem. W kontekście całkowitej oceny programu, najdłuższy przebieg programu generował rozwiązania z błędem poniżej 50%, co zostało uznane za zadowalające, ale nadal wymagające optymalizacji. W rezultacie wprowadzono zmiany w metodzie poszukiwania sąsiedztwa, co zaowocowało utworzeniem drugiego głównego programu (main2), który wykazywał znacznie lepszą wydajność, zwłaszcza dla dużych instancji problemu. Pod względem złożoności pamięciowej i czasowej, program prezentował się zgodnie z teorią, wykazując odpowiednio liniową i kwadratową zależność, co potwierdzało dokumentację algorytmu. Podsumowując, mimo pewnych wyzwań i poprawek w trakcie projektu, algorytm Tabu-Search wyłonił się jako skuteczne narzędzie do rozwiązywania problemów optymalizacyjnych, szczególnie w kontekście dużych instancji grafów. Ostateczne dostosowanie parametrów, takich jak liczba iteracji i wielkość tablicy Tabu, pozwoliło osiągnąć satysfakcjonujące rezultaty, a wprowadzone zmiany w metodologii poszukiwania sąsiedztwa przyczyniły się do istotnej poprawy efektywności algorytmu. Wnioski te sugerują, że Tabu-Search może stanowić wartościową alternatywę w dziedzinie optymalizacji, dostosowując się do różnorodnych wymagań związanych z rozmiarem i złożonością problemu.