

Projektowanie Efektywnych Algorytmów

Projekt

23/ 01 / 2024

263895 Jacek Bogdański

7 Algorytm mrówkowy

Spis treści	
1 Sformułowanie zadania	2
2 Metoda	3
3 Algorytm	5
4 Dane testowe	8
5 Procedura badawcza	9
6 Wyniki	12
7 Analiza wyników i wnioski	18

1. Sformułowanie zadania

Zadanie polega na opracowaniu, implementacji i zbadaniu efektywności algorytmu rozwiązującego problem komiwojażera w oparciu o algorytm mrówkowy.

Problem komiwojażera jest jednym z najbardziej znanych zagadnień w kombinatoryce. Polega na znalezieniu najkrótszej drogi, którą musi pokonać komiwojażer, aby odwiedzić wszystkie miasta dokładnie jeden raz i wrócić do miasta początkowego. TSP jest NP-trudny, co oznacza, że nie istnieje algorytm pozwalający na znalezienie optymalnego rozwiązania w rozsądnym czasie.

Algorytm mrówkowy, oparty na obserwacjach zachowania mrówek podczas poszukiwania najkrótszej trasy do źródła pożywienia, jest sposobem na rozwiązanie różnych problemów, w tym problemu komiwojażera (TSP). Bazuje on na zastosowaniu sztucznych mrówek, które imitują naturalne zachowanie tych istot, w tym pozostawianie feromonów na ścieżkach. Ślad pozostawionych przez mrówki feromonów wyznacza najlepsze rozwiązanie.

Algorytm zostanie zaimplementowany w języku C++, a następnie zbadany pod kątem jego efektywności i zdolności znajdowania optymalnych rozwiązań oraz wpływu parametrów ALPHA i BETA na zachowanie algorytmu (jakość uzyskiwanych rozwiązań, czas uzyskiwania rozwiązań i zużycie pamięci). Do badań zastosowane zostaną 2 schematy rozkładu feromonu – DAS i CAS.

W tym zadaniu zostanie zbadany wpływ różnych strategii i parametrów algorytmu mrówkowego na jego skuteczność w rozwiązywaniu problemu TSP. Skuteczność ta będzie oceniana pod kątem:

- Efektywność algorytmu w znajdowaniu najkrótszej ścieżki.
- Czasu Znajdowania Rozwiązań: Szybkość, z jaką algorytm znajduje optymalne rozwiązania.
- Zużycia zasobów wymaganych do wykonania algorytmu.

Wartości parametrów $ALPHA = 1$ i $BETA = 2$ w sposób istotny powinny wpływać na zachowanie algorytmu. Wprowadzenie zmian w wartościach tych parametrów, zarówno w górę, jak i w dół, powinno skutkować zmianą średniego błędu rozwiązania.

Wielkość błędu, w zależności od wielkości instancji, nie powinna przekraczać:

- dla $n < 25$, 00 %
- dla $24 < n < 74$, 50 %
- dla $75 < n < 449$, 100 %
- dla $450 < n < 2500$, 150 %

2. Metoda

Metoda mrówkowa jest heurystycznym algorytmem optymalizacyjnym inspirowanym zachowaniem kolonii mrówek w poszukiwaniu najkrótszej ścieżki z mrowiska do źródła pożywienia. Stosuje się ją między innymi do rozwiązywania problemu komiwojażera (TSP).

W 1989 roku Deneubourg odkrył, że mrówki potrafią efektywnie odnajdywać najkrótszą drogę do pożywienia, wykorzystując zjawisko tzw. inteligencji stadnej. Mrówki komunikują się ze sobą poprzez pozostawianie feromonów, tworząc ścieżki, a proces ten nosi nazwę stygmergii.

Algorytm mrówkowy, zastosowany do rozwiązania problemu komiwojażera (TSP), działa poprzez iteracyjne poszukiwanie cyklu Hamiltona przez sztuczne mrówki, które podejmują decyzje o wyborze następnego wierzchołka na podstawie pewnego prawdopodobieństwa, opartego o ilość feromonów i długość ścieżki. Mrówka przechodząc po trasie, zostawia ona lokalnie feromony, a po zakończeniu każdej iteracji (przejścia wszystkich mrówek) feromony aktualizowane są globalnie. Istnieją trzy typy pozostawiania feromonów: ilościowy, gęstościowy i cykliczny. W przypadku algorytmu cyklicznego, feromony są uaktualniane po wykonaniu iteracji, a każda mrówka posiada jednakową ilość feromonu, dodawaną do każdej krawędzi trasy po podziale przez jej długość. Szybkość parowania feromonów jest ustalana w parametrze Rho .

Metaheurystyka

Populacja niezależnych „agentów” konstruuje rozwiązanie (każdy agent swoje), w kolejnych losowych krokach. Rozwiązania podlegają (po utworzeniu) ocenie. Krawędzie ścieżki stanowiącej rozwiązanie oznaczane są feromonem (zwiększeniem wagi, istotności), proporcjonalnie do jakości uzyskanego rozwiązania. Kolejne pokolenia agentów tworzą kolejne rozwiązania uwzględniając ilość feromonu na krawędziach.

Heurystyka wyboru lokalnego

Poziom feromonu, rozkładanego na całej drodze, określa globalną jakość tej drogi. Niemniej jednak, może on powodować zakazywanie eksploracji przestrzeni rozwiązań. Aby uniknąć utkania w ekstremach lokalnych, wybór krawędzi zależny jest również od lokalnego kryterium.

Charakterystyka mrówek wirtualnych

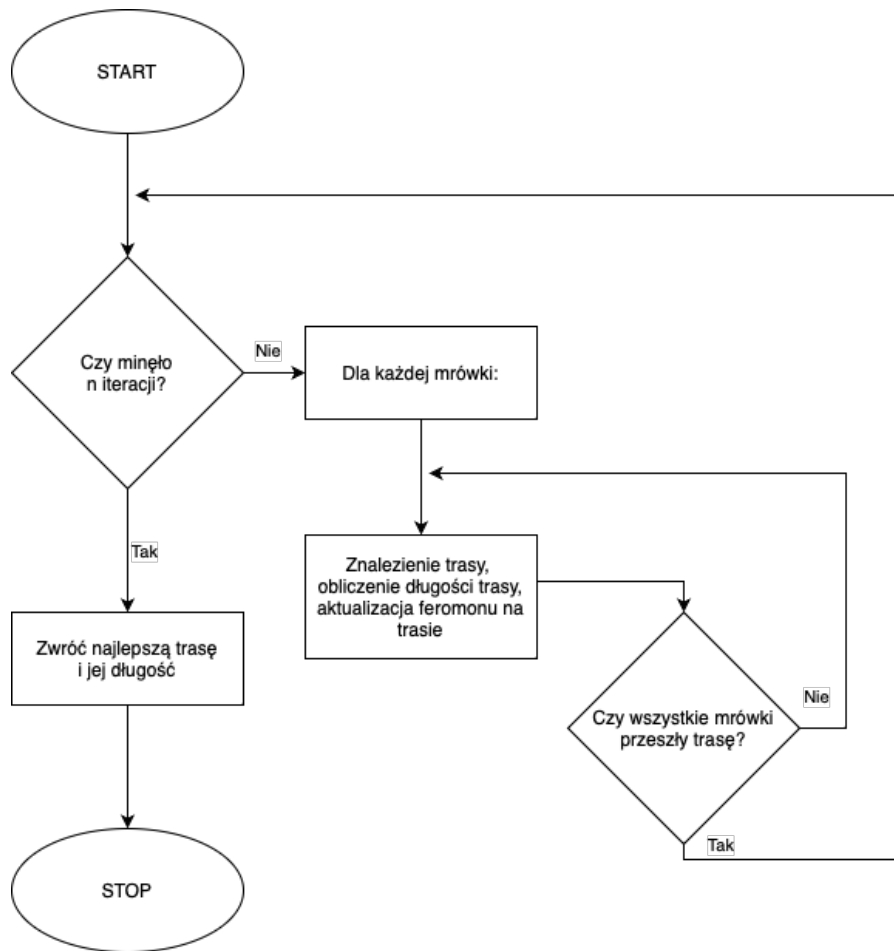
Poruszają się w dyskretnych chwilach czasu, w dyskretnej przestrzeni zdefiniowanych grafów. Posiadają określony algorytm optymalizacji. Mogą aktualizować ślad feromonowy w trakcie i po znalezieniu rozwiązania. Mogą określać, rodzajem bądź ilością feromonu, jakość trasy, mogą rozpoznawać różnicę w jakości dróg w danym miejscu – węzle grafu oraz mogą podejmować decyzje o wyborze trasy o identycznym poziomie feromonu na podstawie określonego kryterium. Mogą również posiadać ograniczoną pamięć odwiedzonych węzłów.

Należy zwrócić uwagę na to, że mrówki sztuczne są modelem wyposażonym w pewne cechy w zależności od rozwiązywanego problemu. Mają one tyle wspólnego z naturalnymi, co sztuczny neuron z komórką nerwową.

Ogólny opis metody:

- Inicjalizacja feromonów (nadanie początkowych wartości feromonów na wszystkich krawędziach grafu)
- Umieszczenie mrówek w losowych miastach
- Przejście mrówek:
 - Wybór następnego miasta na podstawie prawdopodobieństwa, zależnego od ilości feromonów na krawędziach i odległości między miastami
 - Lokalna aktualizacja feromonów po przejściu mrówki
- Po zakończeniu przejścia wszystkich mrówek, globalna aktualizacja feromonów
- Ślady feromonów subtelnie parują, symulując naturalne procesy parowania
- Kryterium stopu: Liczba iteracji lub warunek zbieżności.

3. Algorytm



Rysunek 1: Schemat blokowy algorytmu

Algorytm mrówkowy do rozwiązania problemu komiwojażera (TSP) działa następująco:

- Uruchomienie określonej liczby iteracji - ustalonej z góry lub przerwanej po osiągnięciu warunku celu
- Mrówki mogą być rozmieszczone w losowym mieście.
- Na początku, ilość feromonu na każdej krawędzi zależy od szacowanej długości trasy i liczby mrówek.
- W każdej iteracji poszukiwanie przez grupę mrówek cyklu Hamiltona
- Po każdej iteracji ilość feromonów na ścieżkach jest aktualizowana
- Wybór następnego wierzchołka: Mrówka podczas trasy decyduje o wyborze następnego wierzchołka na podstawie pewnego prawdopodobieństwa.
- Po wykonanej iteracji feromony odparowują zgodnie z parametrem ρ

Algorytm został zaimplementowany w języku C++.

Prawdopodobieństwo wyboru wierzchołka określane jest wg wzoru:

$$p(c_{ij}|s^p) = \frac{\tau_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum_{c_{il} \in N(s^p)} \tau_{il}^\alpha \cdot \eta_{il}^\beta}, \quad \forall c_{ij} \in N(s^p),$$

gdzie:

$C = \{c_{ij}\}, i = 1, \dots, n, j = 1, \dots, |D_i|$ - zbiór komponentów rozwiązania,

s^p - rozwiązanie częściowe uzupełniane o dopuszczalne komponenty z sąsiedztwa $N(s^p) \subset C$,

τ_{ij} i η_{ij} - wartości określające ilość feromonu i lokalnej funkcji wyboru dla komponentu (krawędzi) c_{ij} ,

α i β - współczynniki (dodatnie) określające wpływ (ważność) feromonu i heurystyki

Rysunek 2: Wzór na prawdopodobieństwo wyboru wierzchołka

CAS

W algorytmie cyklicznym stała ilość feromonu $Q3$ dzielona jest przez długość drogi Lk przebytej przez k -tą mrówkę i rozkładana na wszystkich krawędziach tej drogi.

DAS

W algorytmie gęstościowym, po każdym przejściu mrówki po krawędzi i, j , na każdą jednostkę jej długości rozkładana jest stała ilość feromonu $Q1$

Złożoność obliczeniowa

Złożoność obliczeniowa algorytmu CAS wynosi: $\mathcal{O}(LC(n^2)m)$, gdzie:

LC - liczba cykli,

n - rozmiar instancji (liczba wierzchołków grafu),

m - liczba mrówek.

Dorigo z zespołem znaleźli liniową zależność pomiędzy rozmiarem instancji (liczbą wierzchołków/miast) a liczbą mrówek, stąd złożoność algorytmu wynosi: $\mathcal{O}(LC(n^3))$.

Przyjęte parametry w AS wg Dorigo:

- α (ALPHA) = 1,
- β (BETA) = 2,
- ρ (RHO) = 0,5,
- $m = n$, gdzie m to liczba mrówek, n liczba miast,
- $\tau_0 = m / Cnn$, gdzie Cnn jest szacowaną długością trasy.

Skład programu:

- Inicjalizacja Struktur i Zmiennych:
 - Zdefiniowanie stałych ALPHA, BETA, RHO, numIterations.
 - Inicjalizacja zmiennych takich jak configFileName, sourceDirectory, outputFileName, statFileName, sourceFileName, tspType, asType, repeatCount, expectedLength.
- Konstruktor i Destruktor:
 - Konstruktor przyjmuje nazwę pliku konfiguracyjnego.
 - Destruktor jest odpowiedzialny za zwolnienie zasobów.
- Uruchomienie programu:
 - Funkcja handleConfigFile jest główną funkcją uruchamiającą program.
- Obsługa Plików:
 - Funkcje initFiles inicjalizują strumienie plikowe.
 - countConfigLines sprawdza liczbę linii w pliku konfiguracyjnym.
 - setOutputFileName ustawia nazwę pliku wyjściowego.
- Obsługa Konfiguracji:
 - handleConfigLine obsługuje każdą linię pliku konfiguracyjnego.
 - readSourceFile wczytuje dane z pliku.
- Algorytm Mrówkowy (Ant Algorithm):
 - antAlgorithm to główna funkcja wykonująca algorytm mrówkowy.
 - antSteps opisuje ruchy mrówek.
 - updatePheromones aktualizuje feromony globalnie po przejściu mrówek.
 - runAnts to pętla powtórzeń iteracji.
 - getRandomInt generuje losowe liczby całkowite.
 - estimateTourLength oblicza szacunkową długość trasy algorytmem najbliższego sąsiada.
- Pomocnicze Funkcje:
 - shiftTour przesuwa trasę tak, aby zaczynała się od miasta o numerze 0.
 - calculateDistance oblicza odległość między dwoma miastami.
 - getRandomInt generuje liczbę losową
- Cały kod znajduje się w przestrzeni nazw PEA.
- Kod jest zabezpieczony przed powtórным dodaniem nagłówka za pomocą dyrektywy #ifndef, #define, #endif.

4. Dane testowe

Do sprawdzenia poprawności działania algorytmu zostały wykorzystane instancje pochodzące z:

<http://comopt.ifl.uni-heidelberg.de/software/TSPLIB95/tsp/>

TSP (

burma14.tsp,
gr17.tsp,
gr21.tsp,
gr24.tsp,
bays29.tsp,
ch150.tsp,
pcb442.tsp

)

oraz <http://comopt.ifl.uni-heidelberg.de/software/TSPLIB95/atsp/>

ATSP (

ftv33.atsp,
ftv44.atsp,
ft53.atsp,
ftv70.atsp,
ftv170.atsp,

).)

Do wykonania badań wykorzystano ten sam zestaw instancji.

5. Procedura badawcza

Celem badania jest określenie czasu potrzebnego na znalezienie rozwiązania problemu oraz określenie błędu otrzymanego rozwiązania względem rozwiązania optymalnego. Badanie polega na uruchomieniu programu z instrukcjami zawartymi w pliku config.ini – pliku inicjującym znajdującym się w tym samym katalogu, co plik .exe.

Źródło danych testowych: <http://comopt.ifl.uni-heidelberg.de/software/TSPLIB95/tsp/> oraz <http://comopt.ifl.uni-heidelberg.de/software/TSPLIB95/atsp/>

Program automatycznie wykrywa i obsługuje pliki .tsp i .attp w najpopularniejszych formatach:

- FULL_MATRIX – pełna macierz odległości – dla ATSP
- LOWER_DIAG_ROW – kolejne liczby stanowiące trójkąt pod dolną przekątną macierzy odległości – tylko dla TSP
- COORD_DISPLAY – lista miast wraz z ich koordynatami (X, Y) na mapie

Sterowanie trybem CAS/DAS odbywa się poprzez umieszczenie w pliku inicjującym config.ini frazy MODE_CAS lub MODE_DAS.

Wyniki są zapisywane do 2 plików, których nazwy również są definiowane w pliku inicjującym – 2 następujące po sobie w jednej linii nazwy plików. Do pierwszego pliku trafiają wyniki poszczególnych powtórzeń algorytmu, a do drugiego uśrednione wyniki ze wszystkich powtórzeń.

Format linii z plikiem źródłowym:

(nazwa_pliku liczba_powtorzen najkrotsza_znana_dlugosc najkrotsza_znana_sciezka)

Przykładowy plik config.INI:

```
gr24.tsp 10 1272 [16 11 3 7 6 24 ...  
bays29.tsp 10 2020 [1 28 6 12 9 ...  
ch150.tsp 10 6528 [1 98 103 82 ...  
gr202.tsp 5 40160 [1 3 16 14 13 ...  
pcb442.tsp 2 50778 [1 2 3 4 5 6 ...
```

```
MODE_CAS  
output.csv stats.csv
```

Tabela 1: Plik .ini programu

Do pliku wyjściowego output.csv zapisywany jest, dla każdego wykonania algorytmu, czas wykonania, błąd znalezionej rozwiązania względem znanego rozwiązania optymalnego podanego w pliku .ini, długość znalezionej ścieżki oraz kolejne wierzchołki znalezionej ścieżki. Format pliku wyjściowego to CSV.

Plik output.csv:

```
Time[ms]; Error; Distance; Path  
pr1002.tsp; 259045; [1 2 5 3 4 6 7 8 9 10 11 12 63 ...]  
3749371.7250; 33.5031%; 345833; [0 73
```

Tabela 2: Pierwszy plik wyjściowy programu

Drugi plik wyjściowy zawiera uśrednione wartości czasu i błędów z powtórzeń wykonania algorytmu dla poszczególnych plików. Plik ten zostanie wykorzystany do analizy, więc zawiera również kolumny sygnalizujące rodzaj (ATSP / TSP) oraz tryb (CAS / DAS).

Plik stats.csv:

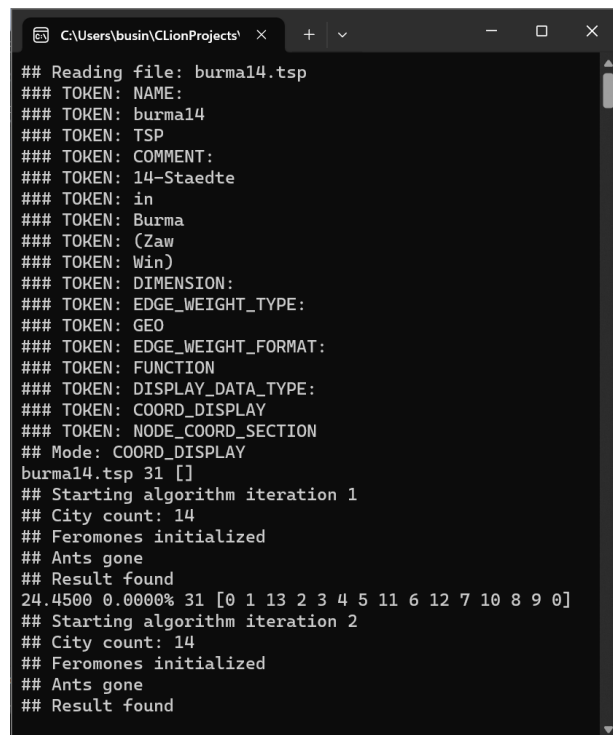
```
Filename;Type;AS Type;Expected Distance;Best Distance;Avg Error[%];Avg Time[ms];Best Path  
ftv33.atsp; ATSP; CAS; 1286; 1316; 8.11042; 154.795; [0 13 12 14 15 16 1 25 24 23 27 ...  
ftv44.atsp; ATSP; CAS; 1613; 1711; 11.9467; 346.147; [0 1 2 5 20 7 9 8 40 34 36 37 38 ...  
ft53.atsp; ATSP; CAS; 6905; 8209; 22.5851; 573.968; [0 36 35 40 38 45 43 42 47 46 ...
```

Tabela 3: Drugi plik wyjściowy programu

Program drukuje w terminalu konsoli informacje dotyczące jego działania, aby można było monitorować stan wykonywanych operacji. Przy odczycie pliku źródłowego, następuje jego skan linia po linii i odczyt m.in. formatu i liczby miast. Ustalenie formatu jest potwierdzane komunikatem:

Mode: COORD_DISPLAY.

W czasie wykonywania algorytmu program sygnalizuje w konsoli kluczowe dane z jego działania – numer porządkowy powtórzenia algorytmu, liczbę miejscowości, zakończenie inicjalizacji feromonów, zakończenie ruchów mrówek, znalezienie rozwiązania.



```

## Reading file: burma14.tsp
### TOKEN: NAME:
### TOKEN: burma14
### TOKEN: TSP
### TOKEN: COMMENT:
### TOKEN: 14-Staedte
### TOKEN: in
### TOKEN: Burma
### TOKEN: (Zaw
### TOKEN: Win)
### TOKEN: DIMENSION:
### TOKEN: EDGE_WEIGHT_TYPE:
### TOKEN: GEO
### TOKEN: EDGE_WEIGHT_FORMAT:
### TOKEN: FUNCTION
### TOKEN: DISPLAY_DATA_TYPE:
### TOKEN: COORD_DISPLAY
### TOKEN: NODE_COORD_SECTION
## Mode: COORD_DISPLAY
burma14.tsp 31 []
## Starting algorithm iteration 1
## City count: 14
## Feromones initialized
## Ants gone
## Result found
24.4500 0.0000% 31 [0 1 13 2 3 4 5 11 6 12 7 10 8 9 0]
## Starting algorithm iteration 2
## City count: 14
## Feromones initialized
## Ants gone
## Result found

```

Rysunek 3: Wyjście konsoli programu

Wówczas do pliku i do wyjścia konsoli trafia linia informująca o wyniku algorytmu i algorytm jest wykonywany ponownie. Taki proces jest powtarzany aż do przejścia przez wszystkie zadane pliki źródłowe.

Badania zostały przeprowadzone na urządzeniu:

- Procesor Intel® Core™ i7-8565U CPU @ 1.80GHz – 1.99GHz x64
- Pamięć RAM DDR3 16.0 GB
- System Windows 11 Home 22H2 64-bit

Pomiar czasu został wykonany przy pomocy biblioteki <chrono>. Czas jest mierzony od momentu rozpoczęcia wykonywania algorytmu dla danej instancji do końca jego wykonywania.

```

auto start_time = std::chrono::high_resolution_clock::now();

// Wywołanie algorytmu mrówkowego

auto end_time = std::chrono::high_resolution_clock::now();
auto duration = std::chrono::duration_cast<std::chrono::microseconds>(end_time - start_time);
long double milliseconds = duration.count() / 1000.0;

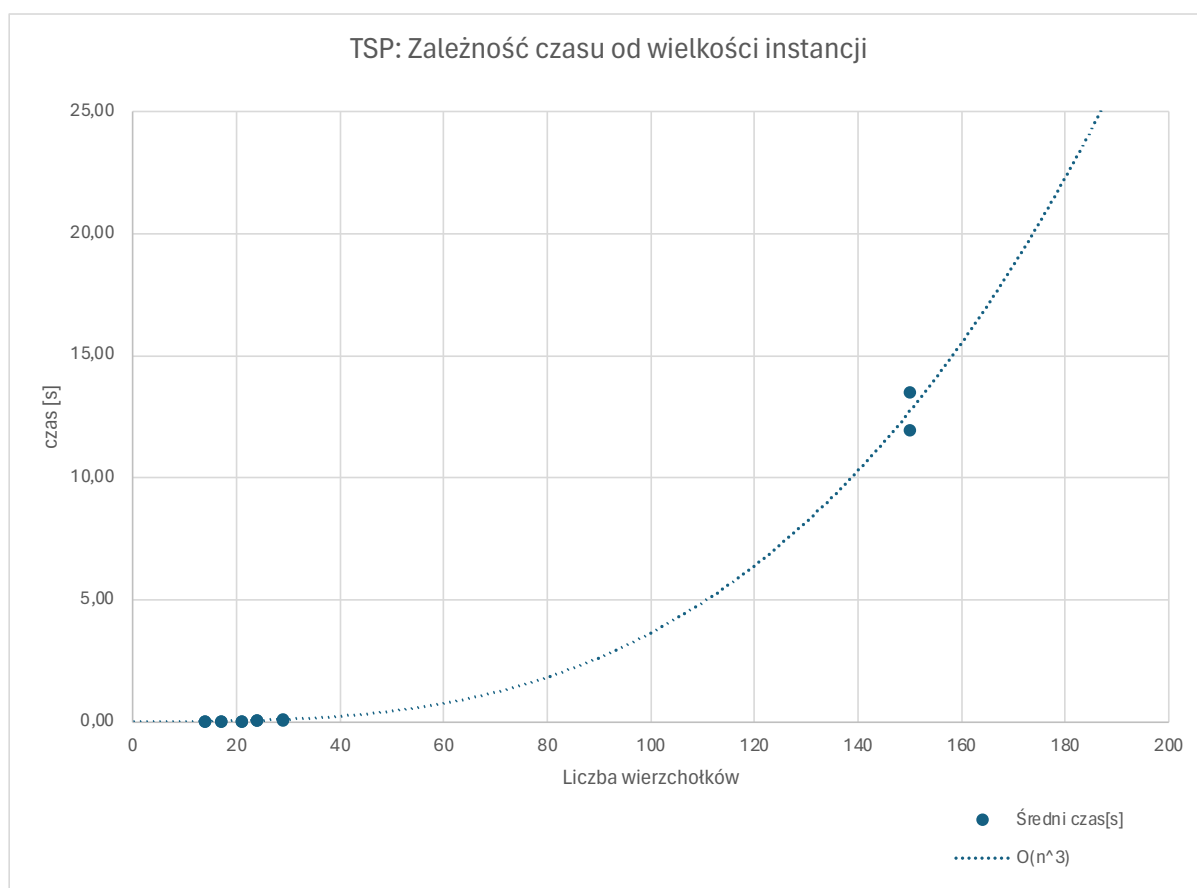
```

Tabela 3: Kod wykonujący pomiar czasu

6. Wyniki

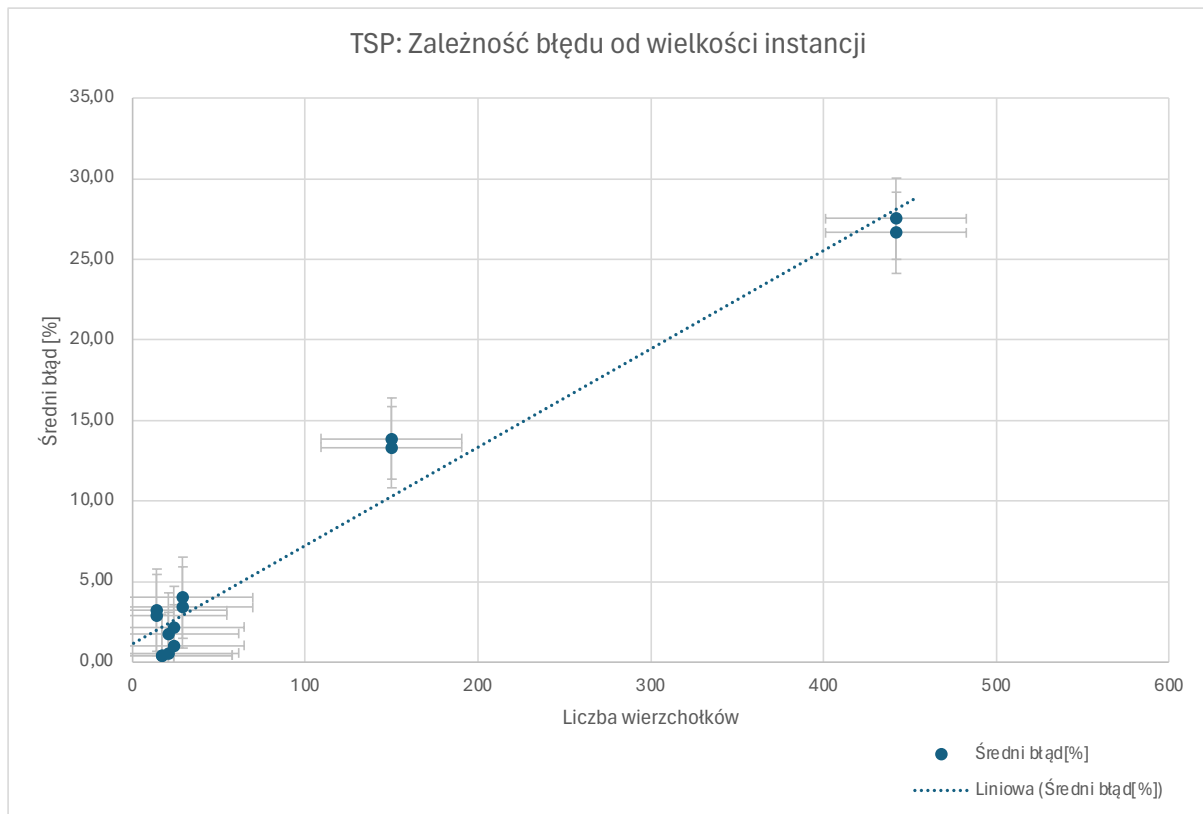
W trakcie eksperymentów przeprowadzonych na różnych instancjach symetrycznego i asymetrycznego problemu komiwożacza algorytm był poddany pomiarowi czasu wykonywania i jakości zwracanego rozwiązania. Największą zbadaną instancją TSP było pcb442.tsp, natomiast dla ATSP wykorzystano ftv170.atsp. W przypadku większych instancji, program wykonywał się przez ponad godzinę, co zmusiło do przerywania procesu.

6.1 TSP



Rysunek 4: Wykres czasu od wielkości instancji TSP

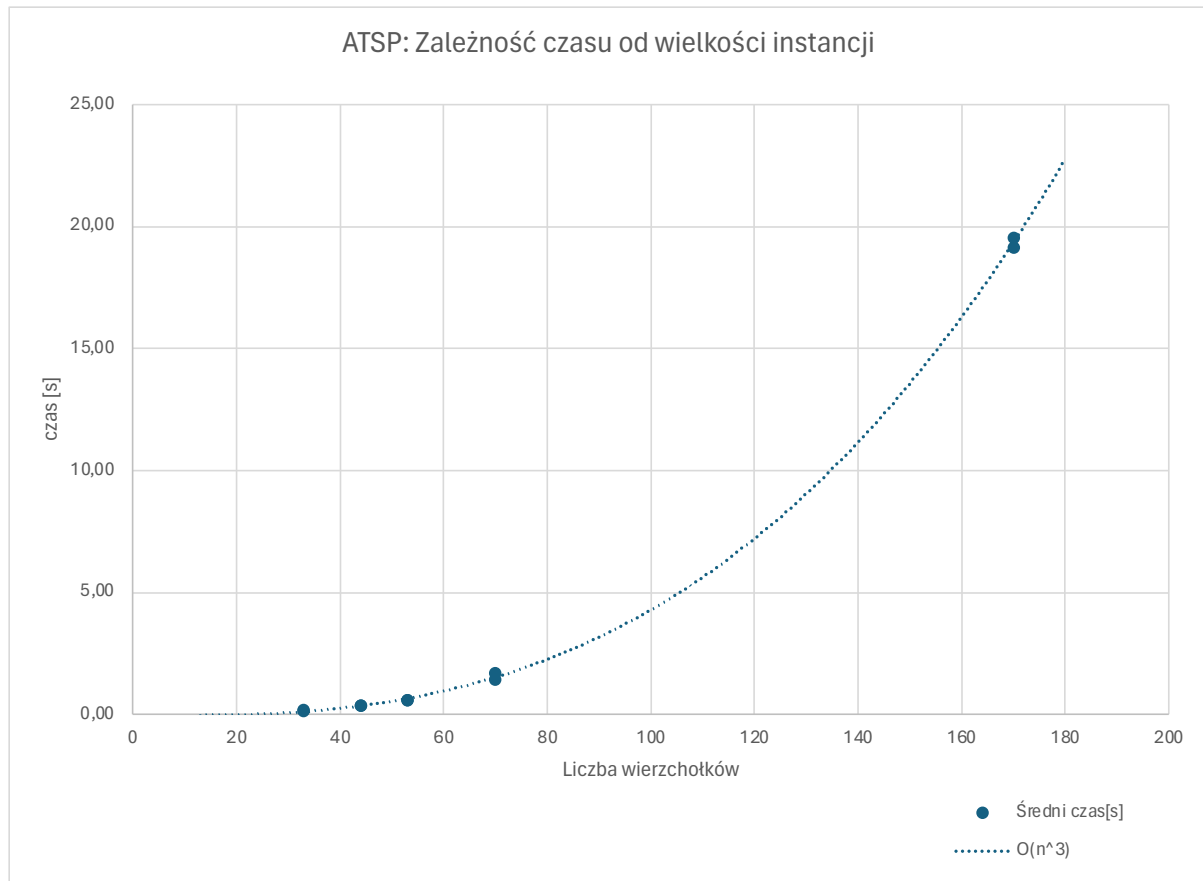
Na wykresie znajdującym się na Rysunku 4. widać, że algorytm dla problemu TSP osiąga złożoność obliczeniową bliską $O(n^3)$. Jest to najbardziej prawdopodobna złożoność tego algorytmu, kropkowana krzywa wielomianowa pasuje do danych uzyskanych eksperymentalnie.



Rysunek 5: Wykres błędów od wielkości instancji TSP

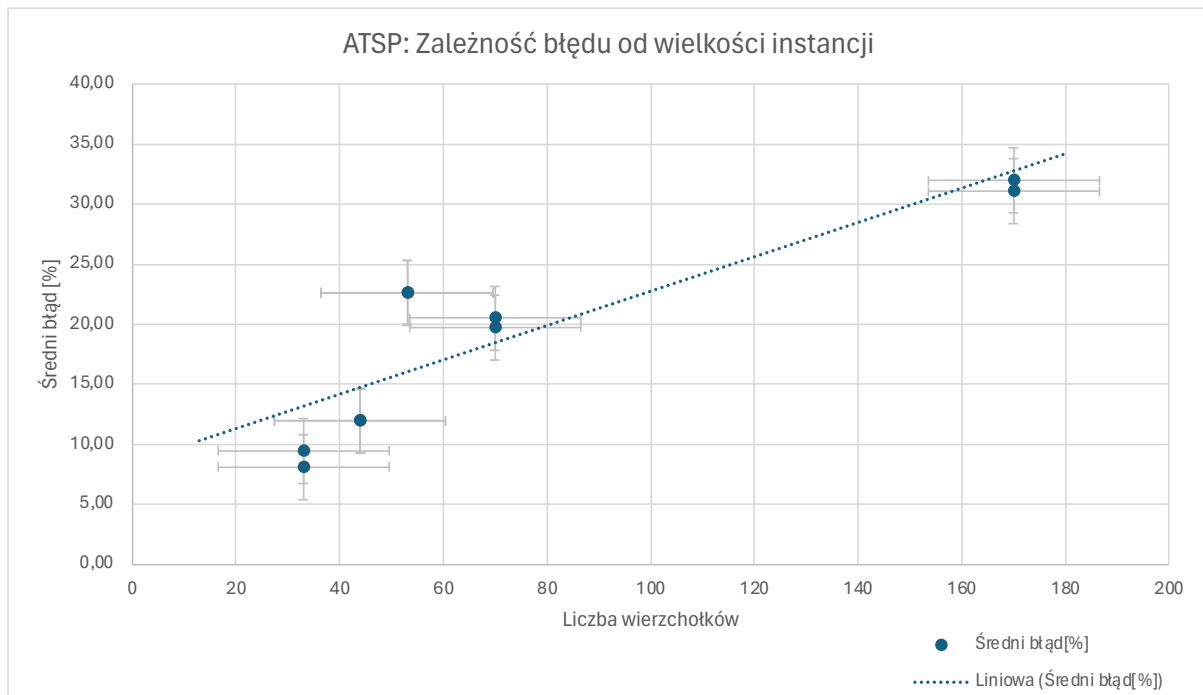
Na Rysunku 5. znajduje się wykres błędów znajdowanego rozwiązania względem znanego najlepszego rozwiązania dla problemu TSP. Jak widać, dla instancji do 150 wierzchołków algorytm osiąga średni błąd poniżej 15%, co jest bardzo dobrym wynikiem. Wartość ta oraz wartość ~28% błęd przy 450 wierzchołkach mieści się w założonych przedziałach. Funkcja średniego błęd może wskazywać na funkcję liniową, ale bardziej przypomina funkcję logarytmiczną – czerwona linia. Aby dokładniej zbadać zachowanie błęd względny, należałoby powtórzyć badanie dla większej liczby instancji (>50).

6.2 ATSP



Rysunek 6: Wykres czasu od wielkości instancji ATSP

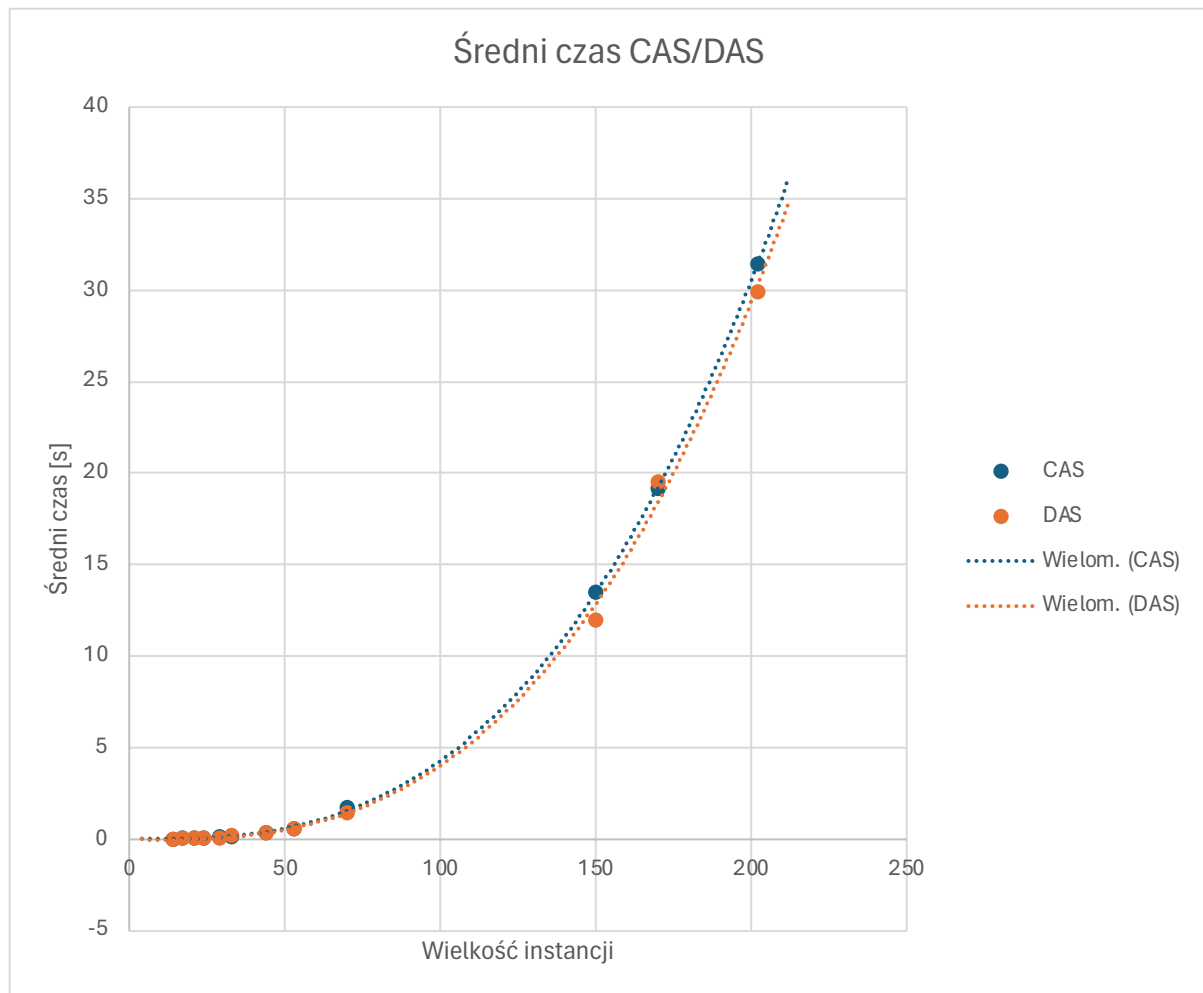
Na wykresie znajdującym się na Rysunku 6. widać, że algorytm dla problemu ATSP również osiąga złożoność obliczeniową bliską $O(n^3)$. Tak jak w przypadku TSP, jest to najbardziej prawdopodobna złożoność tego algorytmu, kropkowana krzywa wielomianowa pasuje do danych uzyskanych w badaniu.



Rysunek 7: Wykres błędów od wielkości instancji ATSP

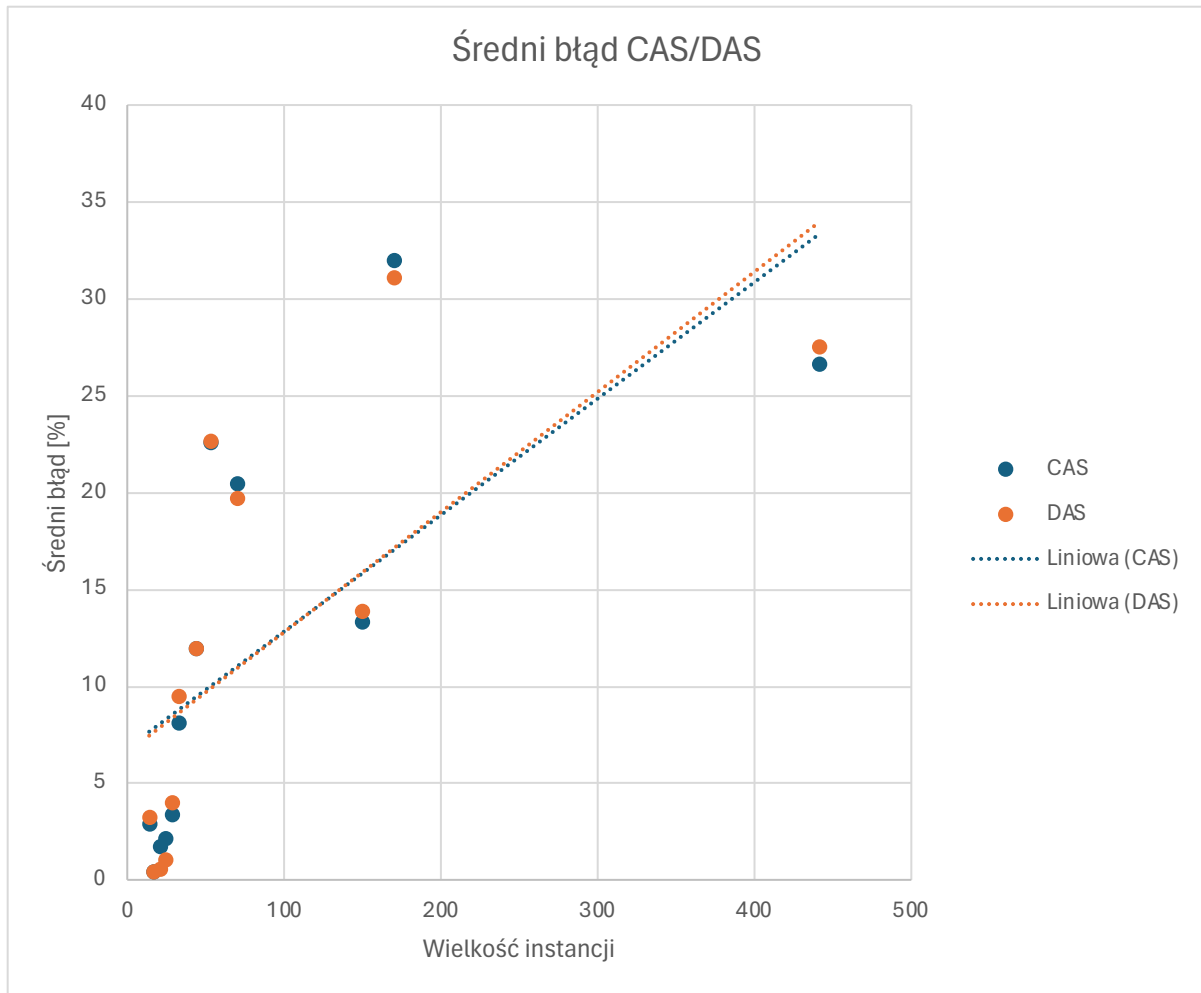
Na Rysunku 7. znajduje się wykres błędów znajdowanego rozwiązania względem znanego najlepszego rozwiązania dla problemu ATSP. Jak widać, dla instancji do 150 wierzchołków algorytm osiąga średni błąd poniżej 35%, co jest wynikiem gorszym od 15% dla TSP, ale mieszczącym się w limicie 100% ustalonym dla badania dla tej liczby wierzchołków. Funkcja średniego błędów bardziej wskazuje na funkcję liniową, ale nadal przypomina funkcję logarytmiczną – czerwona linia. Aby dokładniej zbadać zachowanie błędów względnego, również należałoby powtórzyć badanie dla większej liczby instancji (>50).

6.3 CAS/DAS



Rysunek 8: Wykres czasu od wielkości instancji CAS/DAS

Zestawiając ze sobą algorytm cykliczny (CAS) z algorytmem gęstościowym (DAS) na Rysunku 8, zauważamy, że w kontekście czasu obliczeń lepiej wypada CAS – jest o 6,6% szybszy od DAS dla 200 instancji. Oba algorytmy najprawdopodobniej osiągają złożoność obliczeniową $O(n^3)$ – linie wielomianowe n^3 są dopasowane bardzo dobrze do obu algorytmów.



Rysunek 9: Wykres błędu od wielkości instancji CAS/DAS

Na Rysunku 9. znajduje się zestawienie błędów znajdowanego rozwiązania względem znanego najlepszego rozwiązania dla algorytmów CAS i DAS. Jak widać, średni błąd mieści się w limicie 100% ustalonym dla badania dla tej liczby wierzchołków. Funkcja średniego błędu może wskazywać na funkcję liniową, ale przypomina w tym przypadku bardziej funkcję logarymiczną – czerwona linia. Aby dokładniej zbadać zachowanie błędu względnego, również należałoby powtórzyć badanie dla większej liczby instancji (>50).

Wielkość błędu we wszystkich badanych instancjach mieści się w zadanych granicach.

7. Analiza wyników i wnioski

Wpływ parametrów ALPHA i BETA na działanie algorytmu

Wartościami optymalnymi dla parametrów ALPHA i BETA okazały się wartości podane na wykładzie: $\text{ALPHA} = 1$ i $\text{BETA} = 2$. Zmiana w dół lub w górę jednego z nich lub obu powodowała w testach zwiększanie średniego błędu nawet dla najmniejszych instancji ($n=14$). Oba parametry są bardzo ważne, w pewien sposób przeciwważają się – ALPHA odpowiada za wpływ feromonów na mrówkę, a BETA na preferencję krótszych tras. Oba te parametry biorą udział w procesie decyzji, do jakiego wierzchołka przejdzie mrówka w następnym kroku.

Wpływ ALPHA na eksplorację i intensyfikację

- Wartość ALPHA wpływa na stopień, w jakim mrówki są kierowane przez feromony.
- Wysoka wartość ALPHA skupi mrówki na intensyfikacji, czyli skoncentruje się na silnych ścieżkach feromonowych.
- Niska wartość ALPHA skupi mrówki na eksploracji, czyli bardziej skłoni je do poszukiwania nowych ścieżek.

Wpływ BETA na heurystykę

- Wartość BETA reguluje wpływ heurystyki na wybór ścieżki przez mrówki.
- Wysoka wartość BETA skupi mrówki na heurystyce - będą preferować krótsze ścieżki.
- Niska wartość BETA skupi mrówki na feromonach, ignorując w dużej mierze heurystykę.

Optymalizacja

- Zwiększenie ALPHA może skłonić algorytm do bardziej lokalnej optymalizacji, skupiając się na odkrytych wcześniej ścieżkach
- Zwiększenie BETA może zwiększyć skłonność algorytmu do globalnej optymalizacji, kierując się bardziej heurystyką.

Parametry należy dobierać indywidualnie pod rozwiązywany problem, $\text{ALPHA} = 1$ i $\text{BETA} = 2$ sprawdziły się w tym przypadku.

Porównując Algorytm Mrówkowy do innych algorytmów zaimplementowanych podczas tego projektu, otrzymujemy następujące podsumowanie:

Brute Force (BF) jest algorytmem enumeracyjnym, który sprawdza wszystkie możliwe rozwiązania danego problemu. Jest to algorytm o złożoności eksponencjalnej, co oznacza, że czas potrzebny na znalezienie rozwiązania rośnie wykładniczo wraz ze wzrostem liczby zmiennych w problemie. Program był w stanie obliczyć jakieś rozwiązanie maksymalnie dla 14 wierzchołków. Dalsze zwiększanie rozmiaru instancji prowadzi do czasu obliczeń rzędu dziesiątek, setek i tysięcy godzin. **Złożoność obliczeniowa: najgorsza - $O(n!)$**

Branch&Bound (B&B) jest algorytmem przeszukiwania drzewa, który wyklucza z dalszych rozważań gałęzie, które nie mogą zawierać rozwiązania optymalnego. Jest to algorytm o złożoności wykładniczej, ale może być znacznie wydajniejszy niż Brute Force dzięki zastosowaniu funkcji ograniczenia dolnego. Program był w stanie obliczyć rozwiązanie dla maksymalnie 15 wierzchołków, z powodu całkowitego zużycia zasobów pamięci. **Złożoność obliczeniowa: wykładnicza $O(2^n)$**

Symulowane wyżarzanie (SA) jest algorytmem metaheurystycznym, który wykorzystuje analogię do procesu wyżarzania metalu. Jest to algorytm o złożoności kwadratowej, ale może znaleźć dobre rozwiązania do problemów, które są trudne do rozwiązania za pomocą algorytmów deterministycznych. Przewagą algorytmów heurystycznych nad dokładnymi jest zdolność do znajdowania rozwiązania w instancjach o rozmiarach liczonych w setkach lub nawet tysiącach w rozsądnym czasie. W moim przypadku, algorytm SA obliczył rozwiązanie dla instancji o maksymalnym rozmiarze równym 229. **Złożoność obliczeniowa: kwadratowa $O(n^2)$**

Algorytm mrówkowy (ACO) jest algorytmem metaheurystycznym, który wykorzystuje analogię do zachowania mrówek. Jest to algorytm o złożoności zależnej od parametrów, może znaleźć dobre rozwiązania do problemów, które są trudne do rozwiązania za pomocą algorytmów deterministycznych. Program w czasie krótszym niż godzina jest w stanie obliczyć zbliżoną do optymalnej trasę dla instancji do 500 miast. **Złożoność obliczeniowa: $O(LC \cdot n^3)$** .

Spośród wyżej wymienionych algorytmów, dla małych instancji do 13 miast najlepiej sprawdzi się algorytm Brute Force. Dla instancji od 14 do 15, na lepszych urządzeniach do 20, sprawdzi się Branch&Bound. Te algorytmy gwarantują znalezienie optymalnego rozwiązania. Dla instancji powyżej 20 miast zmuszeni jesteśmy do wyboru algorytmów heurystycznych, które znajdują optymalne rozwiązanie z pewnym przybliżeniem. Przestrzeń rozwiązań TSP, problemu NP-zupełnego, jest tak duża, że nie znamy algorytmu deterministycznego znajdującego rozwiązanie w rozsądnym czasie. Po testach metod Symulowanego Wyżarzania i Algorytmu Mrówkowego, lepiej w kwestii błędów względem najlepszego znanego rozwiązania optymalnego wypada Algorytm Mrówkowy (błędy poniżej 100%, gdzie w SA błędy sięgały 800%).