# Problem Komiwojażera

## Spis Treści

Spis treści

[Problem Komiwojażera 1](#_Toc212371474)

[Spis Treści 1](#_Toc212371475)

[Wstęp 2](#_Toc212371476)

[Testowane Algorytmy i Dane 2](#_Toc212371477)

[Kluczowe Elementy Implementacji 2](#_Toc212371478)

[Sekwencje Ruchu (Operatorzy Sąsiedztwa) 3](#_Toc212371479)

[Parametryzacja i Cel Projektu 3](#_Toc212371480)

[Podsumowanie i Cel Końcowy 3](#_Toc212371481)

[Rozwiązania Solver (Excel) 3](#_Toc212371482)

[Algorytm Najbliższego Sąsiada (Nearest Neighbour - NN) 4](#_Toc212371483)

[Zasada Działania 4](#_Toc212371484)

[Parametryzacja 4](#_Toc212371485)

[Testowanie 4](#_Toc212371486)

[Wnioski 5](#_Toc212371487)

[Algorytm Wspinaczki Wzgórz (Hill Climbing - HC) 5](#_Toc212371488)

[Zasada Działania 5](#_Toc212371489)

[Parametryzacja i Protokół Testowania 5](#_Toc212371490)

[Protokół Testowania 6](#_Toc212371491)

[Testowanie 6](#_Toc212371492)

[Wnioski 8](#_Toc212371493)

[Algorytm Symulacji Wyżarzania (Simulated Annealing - SA) 9](#_Toc212371494)

[Zasada Działania 9](#_Toc212371495)

[Parametryzacja i Protokół Testowania 9](#_Toc212371496)

[Protokół Testowania 10](#_Toc212371497)

[Testowanie 10](#_Toc212371498)

[Wnioski 12](#_Toc212371499)

[Algorytm Przeszukiwania Tabu (Tabu Search - TS) 13](#_Toc212371500)

[Zasada Działania 13](#_Toc212371501)

[Parametryzacja i Protokół Testowania 14](#_Toc212371502)

[Testowanie 14](#_Toc212371503)

[Wnioski 16](#_Toc212371504)

[Algorytm Mrówkowy (Ant Colony Optimization - ACO) 16](#_Toc212371505)

[Zasada Działania 16](#_Toc212371506)

[Parametryzacja 18](#_Toc212371507)

[Testowanie 19](#_Toc212371508)

[Wnioski 19](#_Toc212371509)

## Wstęp

Niniejsze sprawozdanie prezentuje analizę i porównanie różnych algorytmów heurystycznych oraz metaheurystycznych zaprojektowanych do rozwiązania **Problemu Komiwojażera (TSP - Traveling Salesperson Problem)**. TSP, będący klasycznym przykładem problemu optymalizacji kombinatorycznej, polega na znalezieniu najkrótszej możliwej trasy, która łączy zbiór miast, przy czym każde miasto jest odwiedzane dokładnie raz, a trasa kończy się w mieście startowym.

### Testowane Algorytmy i Dane

W ramach projektu zaimplementowano i przetestowano szereg algorytmów optymalizacyjnych. Ich skuteczność i wydajność zostały ocenione na podstawie trzech dostarczonych nam zbiorów danych TSP: **TSP\_127**, **TSP\_48** oraz **TSP\_76**. Różnica w liczbie węzłów w tych zbiorach danych pozwala na wszechstronną ocenę skalowalności i efektywności zastosowanych metod.

### Kluczowe Elementy Implementacji

Wszystkie zaimplementowane algorytmy zostały wzbogacone o mechanizm **multistart (wielokrotnego startu)**. Multistart jest strategią mającą na celu zwiększenie szans na znalezienie globalnie optymalnego rozwiązania, zwłaszcza w przypadku problemów z wieloma lokalnymi minimami. Polega ona na **wielokrotnym uruchamianiu algorytmu z różnych, losowo lub heurystycznie wybranych, początkowych rozwiązań (tras)**. Najlepsze rozwiązanie spośród wszystkich uzyskanych w poszczególnych przebiegach jest traktowane jako ostateczny wynik. Dzięki temu mechanizmowi, algorytm ma możliwość eksploracji szerszego obszaru przestrzeni rozwiązań.

### Sekwencje Ruchu (Operatorzy Sąsiedztwa)

W celu eksploracji przestrzeni rozwiązań i poszukiwania lepszych tras, algorytmy lokalnego przeszukiwania wykorzystują różne **sekwencje ruchu**, czyli operatory sąsiedztwa, definiujące możliwe przejścia z obecnego rozwiązania do rozwiązań sąsiednich:

* **Swap (zamiana):** Polega na **zamianie miejscami dwóch losowo wybranych miast** w bieżącej trasie. Jest to najprostszy operator sąsiedztwa.
* **Insert (wstawianie):** Polega na **usunięciu jednego miasta z obecnej pozycji i wstawieniu go w inną, losowo wybraną pozycję** w trasie.
* **Reverse (odwrócenie), często znane jako 2-opt:** Polega na **wybraniu dwóch krawędzi w trasie, usunięciu ich, a następnie odwróceniu sekwencji miast między tymi punktami, aby utworzyć dwie nowe krawędzie**. Ta operacja jest często bardzo efektywna w znajdowaniu lepszych lokalnie rozwiązań, ponieważ efektywnie minimalizuje przecinające się krawędzie. W naszej pracy zastosowaliśmy różne podejścia do Reverse nie tylko klasyczne ale również różniące się (w zależności od algorytmu)

### Parametryzacja i Cel Projektu

Algorytmy optymalizacyjne są **sparametryzowane**, co oznacza, że ich działanie jest regulowane przez zestaw specyficznych wartości (np. liczba iteracji, temperatura początkowa w Simulated Annealing, wielkość populacji w Algorytmach Ewolucyjnych itp.). W ramach projektu przetestowane zostały różne **sekwencje parametrów** i ich kombinacje, których szczegółowy opis i analiza wpływu na wyniki zostaną przedstawione w dalszych sekcjach sprawozdania.

### Podsumowanie i Cel Końcowy

Głównym celem niniejszego projektu jest **wskazanie, który z zaimplementowanych algorytmów (wraz z jego optymalnymi parametrami)** zapewnia najlepsze rozwiązania dla każdego z testowanych zbiorów danych (**TSP\_127, TSP\_48, TSP\_76**). Końcowa analiza ma na celu identyfikację optymalnego algorytmu i zestawu parametrów dla każdego przypadku, co pozwoli na wyciągnięcie wniosków dotyczących efektywności różnych strategii heurystycznych w kontekście Problemu Komiwojażera.

## Rozwiązania Solver (Excel)

## Algorytm Najbliższego Sąsiada (Nearest Neighbour - NN)

Algorytm Najbliższego Sąsiada (NN) jest najprostszą **heurystyką konstrukcyjną** wykorzystywaną do szybkiego generowania początkowego rozwiązania dla TSP.

### Zasada Działania

Algorytm NN buduje trasę na zasadzie **zachłannego wyboru**:

1. Zaczyna od wybranego miasta startowego.
2. Zawsze wybiera jako następne to miasto, które jest **najbliżej** obecnego, spośród miast jeszcze nieodwiedzonych.
3. Proces jest kontynuowany do momentu odwiedzenia wszystkich miast, a trasa zamyka się powrotem do miasta startowego.

Ze względu na swoją naturę, NN nie gwarantuje globalnego optimum, ale jest szybkim sposobem na uzyskanie rozwiązania bazowego.

### Parametryzacja

W przypadku algorytmu Najbliższego Sąsiada, **jedynym parametrem** wpływającym na wynik jest wybór **miasta startowego**.

Aby zminimalizować wpływ tego wyboru na jakość trasy, testy zostały przeprowadzone dla **każdego możliwego miasta startowego** w danym zbiorze danych. Ostatecznym wynikiem algorytmu NN jest **najkrótsza trasa** uzyskana ze wszystkich tych symulacji. Technika ta jest specyficzną formą **multistartu** zastosowaną do algorytmu konstrukcyjnego.

### Testowanie

**Analiza Wyników dla 48 Miast (NN\_Results\_48\_Cities.csv)**

Dla zbioru 48 miast, algorytm przetestował wszystkie 48 możliwych punktów startowych. Najlepszy znaleziony wynik to:

* **Koszt trasy (FinalRouteCost):** 12012
* **Średni czas wykonania (AvgExecutionTimeMs):** 0.95 ms

**Analiza Wyników dla 76 Miast (NN\_Results\_76\_Cities.csv)**

W przypadku 76 miast, procedura została powtórzona, testując 76 wariantów startowych. Najlepszy uzyskany wynik:

* **Koszt trasy (FinalRouteCost):** 130921.00
* **Średni czas wykonania (AvgExecutionTimeMs):** 2.28 ms

**Analiza Wyników dla 127 Miast (NN\_Results\_127\_Cities.csv)**

Dla największego zbioru danych (127 miast), algorytm wybrał najlepszą trasę spośród 127 symulacji:

* **Koszt trasy (FinalRouteCost):** 133970.65
* **Średni czas wykonania (AvgExecutionTimeMs):** 10.42 ms

### Wnioski

Algorytm Najbliższego Sąsiada (z mechanizmem wyboru najlepszego miasta startowego) jest ekstremalnie szybką metodą konstrukcyjną. Czas działania jest bardzo niski – nawet dla 127 miast średni czas potrzebny na wygenerowanie i porównanie 127 tras to zaledwie 10.42 milisekundy. Algorytm zwraca pojedyncze, deterministyczne rozwiązanie (najlepsze spośród wszystkich możliwych startów) i służy jako szybka metoda do generowania "wystarczająco dobrych" tras bazowych.

### Algorytm Wspinaczki Wzgórz (Hill Climbing - HC)

Algorytm Wspinaczki Wzgórz (HC) jest podstawową **heurystyką lokalnego przeszukiwania (local search)**, której celem jest znalezienie najlepszego rozwiązania w bezpośrednim sąsiedztwie obecnego.

### Zasada Działania

Algorytm zaczyna od rozwiązania początkowego i w każdej iteracji przechodzi do **lepszego rozwiązania sąsiedniego**. Generowanie rozwiązań sąsiednich odbywa się za pomocą operatorów ruchu: **SWAP, INSERT lub REVERSE (2-opt)** (szczegółowo opisanych we Wstępie). Algorytm zatrzymuje się, gdy nie jest w stanie znaleźć lepszego rozwiązania w swoim sąsiedztwie (utknięcie w **lokalnym minimum**).

### Parametryzacja i Protokół Testowania

Aby zwiększyć szansę na ucieczkę z lokalnych minimów i wszechstronnie ocenić algorytm, zastosowano mechanizm **multistart** (jak wyjaśniono we Wstępie), a jego działanie regulują następujące parametry, testowane w różnych konfiguracjach:

| **Parametr** | **Zbiór testowanych wartości** | **Opis** |
| --- | --- | --- |
| **maxIterations** (Maksymalna Liczba Iteracji) | 1000, 5000, 10000, 20000 | Kryterium zatrzymania oparte na liczbie kroków. |
| **maxStagnation** (Maksymalna Stagnacja) | 250, 500, 1000, 2000 | Kryterium zatrzymania oparte na liczbie iteracji bez poprawy. |
| **methods** (Operator Ruchu) | "SWAP", "INSERT", "REVERSE" | Definiuje, który z operatorów sąsiedztwa jest używany w danej serii testów. |

Protokół Testowania

Każda unikalna kombinacja tych parametrów jest testowana w **10 niezależnych uruchomieniach (multistart = 10)**. Ostatecznym wynikiem dla danej konfiguracji jest **najlepsza (najkrótsza) trasa** uzyskana ze wszystkich 10 startów.

### Testowanie

Dla problemu 48 miast, algorytm Wspinaczki Wzgórz bardzo szybko znajduje dobrej jakości rozwiązania.

* **Wpływ Metody:** Operator REVERSE (2-opt) jest zdecydowanie najskuteczniejszy. Uzyskuje on zarówno najlepszy średni koszt (11644) jak i najlepszy pojedynczy wynik (**10784**). Metoda INSERT (średni koszt 12718) jest zauważalnie gorsza, a SWAP (średni koszt 14930) radzi sobie najsłabiej.
* **Wpływ Iteracji (MaxIterations):** Największy skok jakościowy następuje przy przejściu z 1000 na 5000 iteracji (średni koszt spada z 15519 do 12348). Dalsze zwiększanie liczby iteracji (do 10k czy 20k) przynosi już tylko marginalną poprawę. Sugeruje to, że dla tak małego problemu algorytm szybko zbliża się do dobrego minimum lokalnego.
* **Wpływ Stagnacji (MaxStagnation):** Zwiększanie maksymalnej stagnacji konsekwentnie poprawia średni wynik. Pozwala to algorytmowi dłużej przeszukiwać sąsiedztwo w poszukiwaniu wyjścia z minimum lokalnego.
* **Czas Wykonania:** Czasy są bardzo niskie, rzędu kilku milisekund. Najszybszy jest operator REVERSE (średnio 4.15 ms), a najwolniejszy INSERT (7.35 ms).

**Najlepsza znaleziona konfiguracja (48 Miast):**

* **Metoda:** REVERSE
* **MaxIterations:** 5000
* **MaxStagnation:** 500
* **Koszt:** 10784

**Analiza Wyników dla 76 Miast (HC\_Results\_76\_Cities.csv)**

Przy 76 miastach problem staje się znacznie bardziej złożony, co widać po kosztach i wpływie parametrów.

* **Wpływ Metody:** Podobnie jak wcześniej, REVERSE (średni koszt 140172) jest bezkonkurencyjny. Przewaga nad INSERT (167414) i SWAP (191312) jest bardzo wyraźna.
* **Wpływ Iteracji (MaxIterations):** W przeciwieństwie do problemu 48 miast, tutaj *każde* zwiększenie liczby iteracji przynosi znaczącą poprawę. Średni koszt spada systematycznie od 223779 (dla 1000 iteracji) do 143909 (dla 20000 iteracji). Oznacza to, że algorytm potrzebuje znacznie więcej czasu na eksplorację większej przestrzeni rozwiązań.
* **Wpływ Stagnacji (MaxStagnation):** Trend nie jest tu tak jednoznaczny jak w poprzednim przypadku (optimum dla średniej to 1000), jednak najlepszy absolutny wynik został osiągnięty przy maksymalnej stagnacji.
* **Czas Wykonania:** Czasy rosną, ale pozostają bardzo akceptowalne (średnio 10-14 ms).

**Najlepsza znaleziona konfiguracja (76 Miast):**

* **Metoda:** REVERSE
* **MaxIterations:** 20000
* **MaxStagnation:** 2000
* **Koszt:** 109572.017

**Analiza Wyników dla 127 Miast (HC\_Results\_127\_Cities.csv)**

Największy zbiór danych potwierdza trendy obserwowane przy 76 miastach.

* **Wpływ Metody:** Dominacja operatora REVERSE (średni koszt 189596) jest jeszcze wyraźniejsza. INSERT (225036) i SWAP (243622) pozostają daleko w tyle.
* **Wpływ Iteracji (MaxIterations):** Obserwujemy ten sam trend co dla 76 miast – im więcej iteracji, tym lepszy średni wynik. Koszt spada liniowo wraz ze wzrostem liczby iteracji (od 312338 dla 1000 do 170824 dla 20000). Wskazuje to, że dla złożonych problemów algorytm HC wymaga długiego czasu działania, aby zejść do głębokich minimów lokalnych.
* **Wpływ Stagnacji (MaxStagnation):** Tutaj zależność jest ponownie klarowna – wyższa wartość MaxStagnation systematycznie poprawia średni koszt, pozwalając algorytmowi na dokładniejszą eksplorację.
* **Czas Wykonania:** Czas wykonania rośnie proporcjonalnie do złożoności i liczby iteracji, osiągając średnio 40.5 ms dla 20000 iteracji.

**Najlepsza znaleziona konfiguracja (127 Miast):**

* **Metoda:** REVERSE
* **MaxIterations:** 20000
* **MaxStagnation:** 2000
* **Koszt:** 131208.755

### Wnioski

Na podstawie analizy wszystkich trzech zbiorów danych można wyciągnąć następujące wnioski:

1. **Najlepszy Operator:** Operator REVERSE (2-opt) jest zdecydowanie najskuteczniejszym mechanizmem generowania sąsiedztwa. Jego zdolność do "odplątywania" skrzyżowanych ścieżek daje mu ogromną przewagę nad prostymi operatorami SWAP i INSERT.
2. **Skalowalność Iteracji:** W przypadku prostych problemów (48 miast) algorytm szybko znajduje dobre rozwiązanie i dalsze iteracje przynoszą niewielką korzyść. Jednak dla problemów złożonych (76 i 127 miast) **liczba iteracji jest kluczowym parametrem** i powinna być jak największa, aby umożliwić algorytmowi zejście do głębokich minimów lokalnych.
3. **Rola Stagnacji:** Wyższy próg stagnacji jest generalnie korzystny, ponieważ prosty algorytm wspinaczkowy bardzo łatwo wpada w lokalne optima. Danie mu więcej "szans" na znalezienie poprawy (nawet jeśli zajmuje to wiele iteracji bez postępu) poprawia końcowy wynik, szczególnie w złożonych problemach.

## Algorytm Symulacji Wyżarzania (Simulated Annealing - SA)

Algorytm Symulacji Wyżarzania (SA) to **metaheurystyka**, która rozszerza przeszukiwanie lokalne, wprowadzając element losowości pozwalający na **akceptowanie gorszych ruchów** i tym samym na **ucieczkę z lokalnych minimów**.

### Zasada Działania

W przeciwieństwie do HC, SA pozwala na gorsze ruchy z prawdopodobieństwem zależnym od aktualnej **temperatury (T)** – im wyższa temperatura, tym większa szansa na akceptację gorszej trasy (eksploracja). W miarę postępu algorytmu temperatura jest stopniowo obniżana (**chłodzenie**), co sprawia, że algorytm staje się coraz bardziej zachłanny (eksploatacja). Ruchy są generowane za pomocą operatorów sąsiedztwa (**SWAP, INSERT, REVERSE**).

### Parametryzacja i Protokół Testowania

Wydajność SA jest silnie zależna od odpowiedniego doboru schematu chłodzenia, kontrolowanego przez następujące parametry, testowane w różnych konfiguracjach:

| **Parametr** | **Zbiór testowanych wartości** | **Opis** |
| --- | --- | --- |
| **Initial Temperature (T0)** | 1000.0, 500.0, 200.0 | Kontroluje początkowy stopień eksploracji (prawdopodobieństwo akceptacji gorszych ruchów). |
| **Cooling Rate (alpha)** | 0.99, 0.999, 0.95 | Wpływa na tempo obniżania temperatury. Wartość bliższa 1 oznacza wolniejsze chłodzenie i dokładniejsze przeszukiwanie. |
| **Solutions Per Temperature (L)** | 10, 50, 100 | Liczba prób zmian (ruchów) wykonywanych przy stałej temperaturze. |
| **Methods** (Operator Ruchu) | "SWAP", "INSERT", "REVERSE” | Definiuje, który operator sąsiedztwa jest wykorzystywany. |

### Protokół Testowania

Podobnie jak w przypadku HC, **każda unikalna kombinacja** parametrów SA jest testowana w **10 niezależnych uruchomieniach (multistart = 10)**, startując z różnych rozwiązań początkowych. Najkrótsza trasa z tych 10 prób jest traktowana jako ostateczny wynik danej konfiguracji.

### Testowanie

Dla najmniejszego zbioru danych, algorytm SA jest w stanie znaleźć rozwiązania bardzo bliskie tym z algorytmu HC, a nawet nieznacznie lepsze.

* **Wpływ Metody:** Podobnie jak w HC, operator REVERSE (średni koszt 12070) jest najskuteczniejszy. Co ciekawe, INSERT (średni koszt 12512) radzi sobie tutaj lepiej niż w HC w porównaniu do SWAP (średni koszt 13286).
* **Wpływ Parametrów Chłodzenia:**
  + **Temperatura (T0):** Niższa temperatura początkowa (np. T0=200) dała najlepsze *średnie* wyniki. Sugeruje to, że dla prostego problemu zbyt duża początkowa eksploracja (wysokie T0) nie jest konieczna i może prowadzić do gorszych obszarów przestrzeni rozwiązań.
  + **Współczynnik (Alpha):** Najlepsze wyniki dało Alpha=0.99. Bardzo wolne chłodzenie (0.999) okazało się najgorsze, prawdopodobnie dlatego, że algorytm nie zdążył się "schłodzić" (zejść do eksploatacji) w ramach narzuconego limitu iteracji.
  + **Rozwiązania na Temperaturę (L):** Im więcej rozwiązań na temperaturę (L=100), tym lepszy średni wynik.
* **Czas Wykonania:** Czasy są wyższe niż w HC (średnio 16-20 ms), co wynika ze znacznie większej liczby obliczeń (pętle wewnętrzne).

**Najlepsza znaleziona konfiguracja (48 Miast):**

* **Metoda:** REVERSE
* **T0:** 1000.0
* **Alpha:** 0.99
* **L:** 100
* **Koszt:** 10628 (To wynik lepszy niż najlepszy znaleziony przez HC).

**Analiza Wyników dla 76 Miast (SA\_Results\_76\_Cities.csv)**

Przy średnim problemie SA pokazuje swoją siłę, znajdując rozwiązanie lepsze niż najlepsze rozwiązanie HC.

* **Wpływ Metody:** Dominacja REVERSE (średni koszt 114581) jest absolutna. Jest to jedyny operator, który konsekwentnie dostarcza konkurencyjne wyniki.
* **Wpływ Parametrów Chłodzenia:**
  + **Temperatura (T0):** Wpływ temperatury początkowej na *średni* wynik jest minimalny; wszystkie wartości T0 dały bardzo zbliżone średnie koszty.
  + **Współczynnik (Alpha):** Wolniejsze chłodzenie jest wyraźnie lepsze. Alpha=0.999 (średni koszt 124415) i 0.99 127697) znacząco wyprzedzają szybkie chłodzenie 0.95 144076).
  + **Rozwiązania na Temperaturę (L):** Zależność jest bardzo silna – im wyższe L, tym lepszy wynik. L=100 (średni koszt 122246) jest znacznie lepsze niż L=10 147013).
* **Czas Wykonania:** Średni czas wzrasta do 21-26 ms.

**Najlepsza znaleziona konfiguracja (76 Miast):**

* **Metoda:** REVERSE
* **T0:** 1000.0
* **Alpha:** 0.99
* **L:** 100
* **Koszt:** 108280.457 (To wynik lepszy niż najlepszy znaleziony przez HC 109572).

**Analiza Wyników dla 127 Miast (SA\_Results\_127\_Cities.csv)**

Dla najbardziej złożonego problemu, SA również deklasuje HC, znajdując znacznie lepsze rozwiązanie.

* **Wpływ Metody:** Ponownie, REVERSE (średni koszt 143061) jest zdecydowanie najlepszym wyborem.
* **Wpływ Parametrów Chłodzenia:**
  + **Temperatura (T0):** Podobnie jak w problemie 48 miast, niższe T0 dało nieznacznie lepsze *średnie* wyniki, choć najlepszy absolutny wynik znaleziono przy T0=200.
  + **Współczynnik (Alpha):** Wolne chłodzenie (0.99 i 0.999) jest kluczowe i daje drastycznie lepsze wyniki niż szybkie chłodzenie (0.95).
  + **Rozwiązania na Temperaturę (L):** Jest to jeden z najważniejszych parametrów. L=100 (średni koszt 145633) jest o ponad 25% lepsze niż L=10 (średni koszt 199668).
* **Czas Wykonania:** Średni czas wykonania to 35-40 ms.

**Najlepsza znaleziona konfiguracja (127 Miast):**

* **Metoda:** REVERSE
* **T0:** 200.0
* **Alpha:** 0.999
* **L:** 100
* **Koszt:** 121456.030 (To wynik znacznie lepszy niż najlepszy znaleziony przez HC ~ 131208

### Wnioski

Symulowane Wyżarzanie, dzięki mechanizmowi akceptacji gorszych rozwiązań, było w stanie znaleźć znacznie lepsze wyniki (niższe koszty) niż algorytm Wspinaczki Wzgórz dla wszystkich trzech problemów, co dowodzi jego wyższej skuteczności w uciekaniu z minimów lokalnych.

1. **Najlepszy Operator:** REVERSE (2-opt) pozostaje operatorem o najwyższej skuteczności, niezależnie od zastosowanej metaheurystyki (HC czy SA).
2. **Długość Łańcucha (L):** Parametr SolutionsPerTemperature (L) ma krytyczny wpływ na jakość. Wyższa wartość (L=100) pozwala na lepsze ustabilizowanie się algorytmu na danym poziomie temperatury i jest niezbędna do znalezienia dobrych rozwiązań.
3. **Schemat Chłodzenia:** Dla złożonych problemów (76 i 127 miast) kluczowe jest **wolne chłodzenie** (wysokie Alpha, np. 0.99 lub 0.999). Zbyt szybkie chłodzenie (0.95) "zamraża" algorytm w słabym minimum lokalnym.
4. **Temperatura Początkowa (T0):** Wpływ T0 jest najmniej jednoznaczny. Chociaż niskie T0 dawało dobre średnie wyniki, najlepsze absolutne trasy dla 48 i 76 miast znaleziono przy T0=1000. Wydaje się, że wysokie L i Alpha są ważniejsze niż dokładna wartość T0.

## Algorytm Przeszukiwania Tabu (Tabu Search - TS)

Algorytm **Przeszukiwania Tabu (TS)** jest metaheurystyką, która, podobnie jak Symulowane Wyżarzanie, rozszerza proste przeszukiwanie lokalne (**Hill Climbing**) o mechanizmy pozwalające na ucieczkę z minimów lokalnych, ale robi to w sposób deterministyczny.

### Zasada Działania

Algorytm TS działa poprzez **systematyczne eksplorowanie sąsiedztwa** bieżącego rozwiązania, z tą różnicą, że utrzymuje **listę tabu** ruchów (lub atrybutów ruchów), które zostały wykonane niedawno i których **nie wolno** powtórzyć przez określony czas (długość listy tabu).

1. **Lista Tabu:** Przechowuje informacje o ostatnich ruchach. W kontekście TSP i operacji sąsiedztwa (np. **Swap**), na liście tabu rejestrowany jest atrybut ruchu, np. zamiana miast A i B. Przez czas bycia na liście, powrót do poprzedniego stanu (np. ponowna zamiana A i B) jest zabroniony. Ma to na celu unikanie cykli w przestrzeni rozwiązań.
2. **Akceptacja Gorszych Rozwiązań:** W każdej iteracji TS wybiera **najlepsze** rozwiązanie sąsiednie, nawet jeśli jest ono gorsze od obecnego (w przeciwieństwie do HC). Wyjątkiem są ruchy znajdujące się na liście tabu.
3. **Kryterium Aspiracji:** Ruch, który jest tabu, może zostać wykonany, jeśli prowadzi do rozwiązania lepszego niż **najlepsze dotychczas znalezione** (**global best**). Ten warunek, zwany kryterium aspiracji, pozwala algorytmowi na podjęcie "zakazanego" ruchu, gdy jest to strategicznie opłacalne.
4. **Operator Sąsiedztwa:** Generowanie sąsiadów odbywa się za pomocą operatorów ruchu (**SWAP, INSERT, REVERSE**).

Dzięki połączeniu listy tabu i kryterium aspiracji, TS jest w stanie prowadzić efektywne przeszukiwanie w szerokim obszarze przestrzeni rozwiązań, minimalizując ryzyko utknięcia w płytkim minimum lokalnym.

### Parametryzacja i Protokół Testowania

Wydajność algorytmu Tabu Search jest ściśle związana z doborem kluczowych parametrów, które kontrolują równowagę między eksploracją a eksploatacją przestrzeni rozwiązań. W ramach projektu przetestowane zostaną następujące parametry:

| **Parametr** | **Zbiór testowanych wartości** | **Opis** |
| --- | --- | --- |
| **tabuListLengths** (Długość Listy Tabu) | 10, 20, 50 | Liczba iteracji, przez które ruch (lub jego atrybut) pozostaje "tabu". Kontroluje pamięć algorytmu. |
| **maxIterationsWithoutImprovement** (Maksymalna Stagnacja) | 500, 1000, 2000 | Kryterium zatrzymania oparte na liczbie iteracji bez poprawy najlepszego **globalnego** rozwiązania. |
| **maxIterations** (Maksymalna Liczba Iteracji) | 10000 (Wartość stała) | Kryterium zatrzymania oparte na całkowitej liczbie kroków. |
| **moveMethodMap** (Operator Ruchu) | "SWAP", "INSERT", "REVERSE" | Definiuje, który z operatorów sąsiedztwa jest używany w danej serii testów. |

### Testowanie

Analiza została przeprowadzona na podstawie dostarczonych plików TS\_Results\_48\_Cities.csv oraz TS\_Results\_76\_Cities.csv. Niestety, ze względu na brakujące dane, **analiza dla 127 miast nie mogła zostać przeprowadzona**.

**Analiza Wyników dla 48 Miast (TS\_Results\_48\_76.csv)**

Dla problemu 48 miast algorytm Tabu Search był w stanie znaleźć rozwiązanie optymalne, wyrównując najlepszy wynik uzyskany przez Symulowane Wyżarzanie.

* **Wpływ Metody (Operatora Ruchu):** Ponownie, operator **REVERSE** (średni koszt ok. 11 200) jest zdecydowanie najskuteczniejszy. Operator **INSERT** (średni koszt ok. 12 300) radzi sobie zauważalnie lepiej niż **SWAP** (średni koszt ok. 13 600), który pozostaje najsłabszym wyborem.
* **Wpływ Długości Listy Tabu (tabuListLength):** Parametr ten ma niewielki, ale zauważalny wpływ. Najlepsze wyniki (zarówno średnie, jak i absolutne) uzyskano dla tabuListLength = 20. Wartości 10 i 50 dały nieznacznie gorsze rezultaty.
* **Wpływ Stagnacji (maxIterationsWithoutImprovement):** Zwiększenie maksymalnej stagnacji z 500 do 1000 przynosi znaczącą poprawę średniego wyniku dla operatora REVERSE. Dalsze zwiększanie do 2000 daje już tylko marginalną korzyść. Najlepszy absolutny wynik został znaleziony przy stagnacji ustawionej na 1000.
* **Czas Wykonania:** Czasy wykonania są znacząco wyższe niż w przypadku HC i SA, wahając się od ok. 300 ms do ponad 1100 ms. Co ciekawe, REVERSE był średnio najszybszym operatorem, podczas gdy INSERT najwolniejszym.

**Najlepsza znaleziona konfiguracja (48 Miast):**

* **Metoda:** REVERSE
* **tabuListLength:** 20
* **maxIterationsWithoutImprovement:** 1000
* **Koszt:** 10628
  + *Wynik ten jest identyczny z najlepszym wynikiem SA (10628) i lepszy niż HC (10784).*

**Analiza Wyników dla 76 Miast (TS\_Results\_48\_76.csv)**

W przypadku średniego problemu, Tabu Search uzyskał wynik identyczny z najlepszym wynikiem algorytmu Hill Climbing, jednak nieznacznie gorszy niż Symulowane Wyżarzanie.

* **Wpływ Metody (Operatora Ruchu):** Dominacja operatora **REVERSE** (średni koszt ok. 115 000) jest absolutna. Operatory INSERT (śr. ok. 160 000) i SWAP (śr. ok. 186 000) są niekonkurencyjne.
* **Wpływ Długości Listy Tabu (tabuListLength):** Wpływ tego parametru na ostateczny wynik jest minimalny. Wszystkie trzy testowane wartości (10, 20, 50) dały bardzo zbliżone średnie koszty dla operatora REVERSE.
* **Wpływ Stagnacji (maxIterationsWithoutImprovement):** Jest to **kluczowy parametr** dla tej wielkości problemu. Obserwujemy silny, jednoznaczny trend: **im wyższa stagnacja, tym lepszy wynik**. Średni koszt systematycznie spadał wraz ze wzrostem maksymalnej liczby iteracji bez poprawy. Najlepsze absolutne rozwiązanie znaleziono przy najwyższej testowanej wartości stagnacji (2000).
* **Czas Wykonania:** Czasy znacząco rosną, co jest oczekiwane, i są silnie skorelowane z parametrem stagnacji, osiągając średnio ok. 1.45 sekundy dla maxIterationsWithoutImprovement = 2000.

**Najlepsza znaleziona konfiguracja (76 Miast):**

* **Metoda:** REVERSE
* **tabuListLength:** 20
* **maxIterationsWithoutImprovement:** 2000
* **Koszt:** 109572
  + *Wynik ten jest identyczny z najlepszym wynikiem HC (109572), ale gorszy niż SA (108280).*

**Analiza Wyników dla 127 Miast (TS\_Results\_127\_Cities.csv)**

### Wnioski

## Algorytm Mrówkowy (Ant Colony Optimization - ACO)

Algorytm Mrówkowy (ACO) jest zaawansowaną **metaheurystyką populacyjną**, inspirowaną rzeczywistym zachowaniem mrówek poszukujących najkrótszej drogi między mrowiskiem a źródłem pożywienia. W odróżnieniu od algorytmów opartych na lokalnym przeszukiwaniu (jak HC, SA czy TS), ACO należy do rodziny algorytmów **konstrukcyjnych**. Oznacza to, że w każdej iteracji rozwiązania (trasy) są budowane od podstaw przez "agentów" (mrówki), a nie modyfikowane.

### Zasada Działania

Algorytm symuluje działanie kolonii mrówek, które komunikują się ze sobą w sposób pośredni (tzw. **stygmergia**) poprzez zostawianie śladu feromonowego na przebytych ścieżkach (krawędziach między miastami).

Działanie algorytmu opiera się na dwóch głównych fazach, powtarzanych w pętlach:

1. **Konstrukcja Tras przez Mrówki:**
   * W każdej iteracji, określona liczba mrówek (numAnts) jest umieszczana w losowo wybranych miastach startowych.
   * Każda mrówka niezależnie buduje pełną trasę (cykl Hamiltona), odwiedzając każde miasto dokładnie raz.
   * Wybór następnego, nieodwiedzonego miasta przez mrówkę jest **probabilistyczny**. Prawdopodobieństwo przejścia z miasta *i* do miasta *j* zależy od dwóch czynników:
     + **Ilości feromonu (tau):** Siły śladu feromonowego na krawędzi *(i, j)*. Im silniejszy ślad (pozostawiony przez poprzednie mrówki), tym większe prawdopodobieństwo wyboru.
     + **Widoczności (eta):** Heurystycznej atrakcyjności krawędzi. W problemie TSP jest to zazwyczaj odwrotność odległości (1/dij). Mrówki preferują krótsze, "lepiej widoczne" krawędzie.
2. **Aktualizacja Feromonu:**
   * Po tym, jak wszystkie mrówki zbudują swoje trasy, następuje globalna aktualizacja mapy feromonów.
   * **Parowanie (Evaporation):** Ilość feromonu na *wszystkich* krawędziach jest redukowana o pewien stały współczynnik (rho). Mechanizm ten pozwala algorytmowi "zapominać" o złych lub nieużywanych ścieżkach, zapobiegając zbyt szybkiej zbieżności do słabego optimum lokalnego.
   * **Wzmocnienie (Reinforcement):** Mrówki, które w danej iteracji zbudowały *najkrótsze* trasy, nanoszą dodatkowy feromon na krawędzie, z których korzystały. Siła wzmocnienia jest zazwyczaj odwrotnie proporcjonalna do długości znalezionej trasy (im krótsza trasa, tym silniejszy ślad).

Proces ten jest powtarzany przez określoną liczbę iteracji (maxIterations). Z biegiem czasu ślady feromonowe na krawędziach należących do najlepszych globalnie rozwiązań stają się najsilniejsze, kierując kolejne "pokolenia" mrówek w stronę optymalnych tras.

**Ważna uwaga:** W odróżnieniu od algorytmów Hill Climbing, Symulowanego Wyżarzania i Przeszukiwania Tabu, w tej implementacji algorytm mrówkowy **nie wykorzystuje operatorów sąsiedztwa** (sekwencji ruchu) takich jak SWAP, INSERT czy REVERSE. Cały proces optymalizacji opiera się wyłącznie na mechanizmie konstrukcji tras i aktualizacji feromonów.

### Parametryzacja

Wydajność algorytmu ACO jest krytycznie zależna od zestrojenia parametrów, które balansują między eksploracją (poszukiwaniem nowych tras) a eksploatacją (korzystaniem z już znalezionych, dobrych ścieżek).

Poniższa tabela przedstawia parametry zdefiniowane dla algorytmu:

| **Parametr** | **Zbiór testowanych wartości** | **Opis** |
| --- | --- | --- |
| numAnts (Liczba mrówek) | { 10, 20, 50 } | Liczba agentów (mrówek) budujących niezależne trasy w każdej iteracji. |
| alphas (Waga feromonu, alpha | { 1.0, 2.0, 5.0 } | Kontroluje wpływ śladu feromonowego ($\tau$) na decyzję mrówki. Wyższa wartość $\alpha$ oznacza, że mrówki silniej polegają na "doświadczeniu" poprzednich mrówek. |
| betas (Waga heurystyki, beta | { 2.0, 5.0, 7.0 } | Kontroluje wpływ heurystyki (atrakcyjności krawędzi, $\eta$). Wyższa wartość $\beta$ oznacza bardziej "zachłanne" działanie (silniejszą preferencję dla najbliższych miast). |
| rhos (Współczynnik parowania, rho | { 0.1, 0.5, 0.8 } | Określa tempo "zapominania" feromonu (parowania). Wartość bliska 0 oznacza bardzo wolne parowanie (silna pamięć), a bliska 1 bardzo szybkie (słaba pamięć). |

Dodatkowo, zdefiniowano parametry kontrolujące przebieg całego eksperymentu:

* **maxIterations (Liczba iteracji):** Ustawiona na stałą wartość **700**. Definiuje, ile cykli (budowa tras -> aktualizacja feromonu) wykona algorytm w ramach jednego uruchomienia.
* **MultiStartCount (Liczba uruchomień):** Ustawiona na **10**. Podobnie jak w poprzednich algorytmach, cała procedura ACO jest uruchamiana wielokrotnie z różnymi ziarnami losowości, a jako wynik końcowy wybierane jest najlepsze globalnie znalezione rozwiązanie.

### Testowanie

Dobrze, oto skondensowana analiza wyników algorytmu mrówkowego (ACO), skupiająca się na najlepszych kombinacjach parametrów i kluczowych wnioskach w zależności od rozmiaru problemu.

**Wnioski z Eksperymentów Algorytmu Mrówkowego (ACO)**

Wydajność ACO krytycznie zależy od **zrównoważenia eksploracji (poszukiwania nowych tras)** i **eksploatacji (wykorzystywania znalezionych dobrych tras)**. Optymalne parametry zmieniają się w zależności od trudności instancji (liczby miast).

**1. Wpływ Liczby Mrówek (NumAnts)**

* **Mała Instancja (48):** Wystarczająca jest **mała populacja (NumAnts=10)**, co optymalizuje czas wykonania przy jednoczesnym osiągnięciu najlepszego wyniku.
* **Duże Instancje (76 i 127):** **Duża populacja (NumAnts=50)** była kluczowa do osiągnięcia najlepszych wyników. W większej przestrzeni rozwiązań, większa liczba agentów zwiększa **siłę eksploracji**, poprawiając jakość końcowego rozwiązania, choć kosztem dłuższego czasu.

**2. Wpływ Równowagi (alpha i beta)**

* **Waga Heurystyki (beta):** Najlepsze wyniki osiągano konsekwentnie przy **niskiej wartości beta (głównie 2.0)**. Oznacza to, że zbyt duża "zachłanność" (silne preferowanie najbliższego miasta) nie jest efektywna; mrówki muszą brać pod uwagę nie tylko odległość, ale i **feromon**.
* **Równowaga (alpha i beta):** Dla większych problemów (NumAnt=50), optymalna była konfiguracja **alpha=2.0 i beta=2.0**, co wskazuje na potrzebę **równoważenia** podążania za doświadczeniem zbiorowym (feromon) i lokalną informacją (odległość).

**3. Krytyczny Wpływ Parowania (rho)**

* Współczynnik parowania (rho) jest **najbardziej krytycznym** i **zmiennym** parametrem zależnym od rozmiaru problemu. Określa on "pamięć" algorytmu.
* **Szybkie Parowanie (rho=0.8) dla Mniejszych Problemów (48, 76):**
  + W mniejszych przestrzeniach szybkie parowanie (**słaba pamięć**) jest korzystne, ponieważ pozwala algorytmowi **szybko "zapominać" złe ścieżki** i adaptować się do nowych, lepszych rozwiązań znalezionych przez obecne pokolenia mrówek (promowanie **eksploracji**).
* **Wolne Parowanie (rho=0.1) dla Największego Problemu (127):**
  + W ogromnej przestrzeni 127 miast, niska wartość rho (**silna, długa pamięć**) była konieczna. Zapewnia to **długotrwałe wzmacnianie dobrych, historycznych tras**, co kieruje przyszłe pokolenia mrówek do optymalnych regionów poszukiwań (promowanie **eksploatacji**).

### Wnioski

**Wniosek ogólny:** Optymalna strategia ACO polega na: **Dużej Populacji** (dla dużych problemów), **Równowadze** między feromonem a heurystyką, a co najważniejsze, na **Dostosowaniu Pamięci (rho)** – **słaba pamięć** (wysokie rho) dla łatwiejszych problemów i **silna pamięć** (niskie rho) dla najtrudniejszych instancji.

Najlepsze wyniki:

| **Instancja** | **Najlepszy Koszt** | **NumAnts** | **α (Feromon)** | **β (Heurystyka)** | **ρ (Parowanie)** |
| --- | --- | --- | --- | --- | --- |
| **48 Miast** | **10636** | 10 | 1.0 | 2.0 | **0.8** |
| **76 Miast** | **112224.46** | 50 | 2.0 | 2.0 | **0.8** |
| **127 Miast** | **121263.32** | 50 | 2.0 | 2.0 | **0.1** |