Metaheurystyki — zadanie 4

Algorytm mrówkowy (ACO)

GRUPA 3 — piątek 10:15

Bartosz Kołaciński
251554

Nikodem Nowak
251598

18 grudnia 2025

Użyte technologie	Python 3.13
Użyte biblioteki	random, math, time, numpy, pandas, matplotlib.pyplot

Spis treści

1 Opis zasad działania algorytmu

1.1 Opis problemu

Problem polega na znalezieniu najkrótszej trasy odwiedzającej wszystkie atrakcje w wesołym miasteczku. Jest to wariant problemu komiwojażera (ang. *Traveling Salesman Problem*, TSP). Dany jest zbiór punktów (atrakcji) o określonych współrzędnych na płaszczyźnie. Celem jest znalezienie takiej kolejności odwiedzenia wszystkich punktów, aby łączna przebyta odległość była jak najmniejsza.

1.2 Opis algorytmu mrówkowego (ACO)

Algorytm mrówkowy (Ant Colony Optimization) to metaheurystyka inspirowana zachowaniem mrówek szukających pożywienia. Mrówki w naturze komunikują się za pomocą feromonów — substancji chemicznych pozostawianych na trasie. Ścieżki z większą ilością feromonu są chętniej wybierane przez kolejne mrówki.

1.2.1 Pseudokod algorytmu

Algorithm 1 Algorytm mrówkowy (ACO)

Inicjalizacja:
Zainicjuj macierz feromonów $\tau_{ij} \leftarrow 1.0$ dla wszystkich krawędzi (i, j)
Oblicz macierz odległości d_{ij} dla wszystkich par wierzchołków
for $t = 1$ **to** T (liczba iteracji) **do**
    Konstrukcja tras:
    for każda mrówka $k = 1, \ldots, m$ **do**
        Wybierz losowy wierzchołek startowy
        while nie odwiedzono wszystkich wierzchołków **do**
            Wybierz następny wierzchołek j z prawdopodobieństwem:
            $p_{ij} = \frac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in \text{dozwolone}} [\tau_{il}]^{\alpha} \cdot [\eta_{il}]^{\beta}}$
            gdzie $\eta_{ij} = 1/d_{ij}$ (heurystyka odwrotności odległości)
        end while
        Oblicz długość trasy mrówki k
    end for
    Aktualizacja feromonów:
    Wyparowanie: $\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij}$ dla wszystkich (i, j)
    for każda mrówka k **do**
        Depozyt: $\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{k}$ na trasie mrówki
        gdzie $\Delta\tau_{ij}^{k} = 1/L_k$ (L_k — długość trasy mrówki k)
    end for
    Zaktualizuj najlepsze znalezione rozwiązanie
end for
return najlepsza znaleziona trasa

1.2.2 Parametry algorytmu

- m — liczba mrówek w kolonii
- α — wpływ feromonów na wybór ścieżki
- β — wpływ heurystyki (odwrotności odległości) na wybór ścieżki
- ρ — współczynnik wyparowywania feromonów ($0 < \rho < 1$)
- T — liczba iteracji algorytmu
- p_{random} — prawdopodobieństwo losowego wyboru (eksploracja)

1.3 Opis implementacji rozwiązania

1.3.1 Reprezentacja grafu i obliczanie odległości

Graf jest reprezentowany jako macierz odległości euklidesowych między wszystkimi parami atrakcji. Dane wejściowe są wczytywane z pliku tekstowego w formacie: `nrAtrakcji współrzędnaX współrzędnaY`.

```
def read_data(filepath: str, separator: str = ' ') -> tuple:
    from math import dist
    import pandas as pd

    data_csv = list(pd.read_csv(
        filepath, sep=separator, skipinitialspace=True, header=None
    ).to_records(index=False))
    data_csv = [(int(row[0]), int(row[1]), int(row[2])) for row in data_csv]

    data_length = len(data_csv)
    distances_matrix = [[0.0] * data_length for _ in range(data_length)]
    for i in range(data_length):
        for j in range(data_length):
            if i != j:
                p1 = (data_csv[i][1], data_csv[i][2])
                p2 = (data_csv[j][1], data_csv[j][2])
                distances_matrix[i][j] = dist(p1, p2)

    index_to_id_map = {i: rec[0] for i, rec in enumerate(data_csv)}
    return data_csv, distances_matrix, index_to_id_map

```

Kod 1: Wczytywanie danych i obliczanie macierzy odległości

1.3.2 Reprezentacja trasy i kosztu

Trasa jest reprezentowana jako lista indeksów odwiedzonych wierzchołków. Koszt (długość trasy) jest obliczany jako suma odległości między kolejnymi wierzchołkami.

```
def get_path_length(self) -> float:
    length = 0.0
    for i in range(len(self.visited) - 1):
        l1 = self.visited[i]
        l2 = self.visited[i + 1]
        length += self.distances_matrix[l1][l2]
    return length

```

Kod 2: Obliczanie długości trasy

1.3.3 Inicjalizacja feromonów

Macierz feromonów jest inicjalizowana wartością 1.0 dla wszystkich krawędzi.

```
self.pheromones = [
    [1.0] * self.num_attractions for _ in range(self.num_attractions)
]

```

Kod 3: Inicjalizacja feromonów

1.3.4 Mechanizm konstrukcji trasy (reguła wyboru)

Mrówka wybiera następny wierzchołek na podstawie prawdopodobieństwa zależnego od feromonów i heurystyki. Z małym prawdopodobieństwem p_{random} może wybrać losowy wierzchołek (eksploracja).

```
def select_next_node(self, p_random: float) -> None:
    current_node = self.visited[-1]
    allowed_nodes = list(set(range(self.num_attractions)) - set(self.visited))

    if not allowed_nodes:
        return

    # Losowy wybor (eksploracja)
    if random.random() < p_random:
        next_node = random.choice(allowed_nodes)
        self.visited.append(next_node)
        return

    # Wybor na podstawie feromonow i heurystyki
    possibilities = self._calculate_possibilities(current_node, allowed_nodes)
    next_node = random.choices(allowed_nodes, weights=possibilities)[0]
    self.visited.append(next_node)

def _calculate_possibilities(self, current_node, allowed_nodes):
    scores = []
    denominator = 0.0

    for node in allowed_nodes:
        tau = Ant.pheromones[current_node][node]
        distance = Ant.distances_matrix[current_node][node]
        if distance == 0:
            distance = 1e-10  # Obs uga zerowej odleg o ci
        eta = 1.0 / distance

        score = (tau ** Ant.alpha) * (eta ** Ant.beta)
        scores.append(score)
        denominator += score

    if denominator == 0.0:
        return [1.0 / len(allowed_nodes)] * len(allowed_nodes)

    return [score / denominator for score in scores]

```

Kod 4: Wybór następnego wierzchołka przez mrówkę

1.3.5 Aktualizacja feromonów i wyparowywanie

Po każdej iteracji feromony wyparowują (mnożenie przez $(1 - \rho)$), a następnie mrówki deponują feromony proporcjonalnie do jakości swojej trasy.

```
def _update_pheromones(self, ants):
    evaporation_factor = 1.0 - self.rho

    # Wyparowywanie feromon w
    for i in range(self.num_attractions):
        for j in range(self.num_attractions):
            self.pheromones[i][j] *= evaporation_factor

    # Depozyt feromon w
    for ant in ants:
        path = ant.get_visited()
        path_length = ant.get_path_length()

        if path_length > 0:
            deposit = 1.0 / path_length
            for i in range(len(path) - 1):
                from_node = path[i]
                to_node = path[i + 1]
                self.pheromones[from_node][to_node] += deposit
                self.pheromones[to_node][from_node] += deposit

```

Kod 5: Aktualizacja feromonów

1.4 Instrukcja uruchomienia programu

Program uruchamia się poprzez wywołanie pliku `main.py`:

```
python main.py
```

Program jest interaktywny i prosi użytkownika o:

1. Wybór pliku z danymi (A-n32-k5.txt lub A-n80-k10.txt).
2. Podanie parametrów algorytmu:
   - Liczba mrówek (m)
   - Liczba iteracji (T)
   - Wpływ feromonów (α)
   - Wpływ heurystyki (β)
   - Współczynnik wyparowywania (ρ)
   - Prawdopodobieństwo losowego wyboru (p_{random})

Po zakończeniu działania wyświetlane są: najlepsza znaleziona trasa (kolejność atrakcji) oraz jej długość.

W celu wygenerowania wszystkich eksperymentów opisanych w dalszej części sprawozdania, należy uruchomić skrypt:

```
python all_experiments.py
```

Wykresy generuje się komendą:

```
python plots.py
```

2 Eksperymenty i wyniki

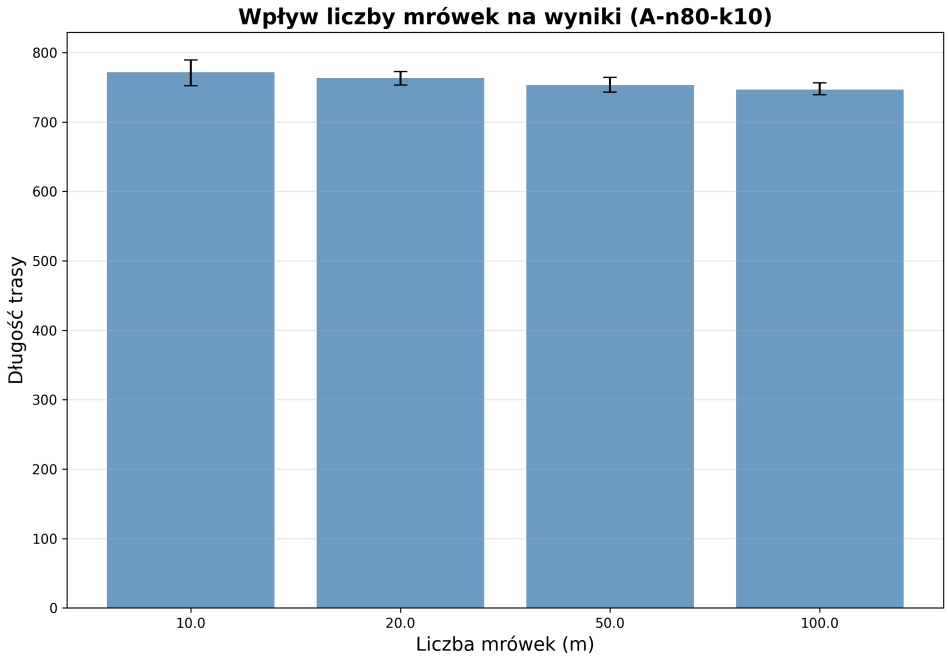
Przeprowadzono serię eksperymentów dla dwóch zbiorów danych:

- **A-n32-k5** — 32 atrakcje
- **A-n80-k10** — 80 atrakcji

Dla każdej konfiguracji algorytm był uruchamiany 5 razy w celu uzyskania statystyk. Poniżej przedstawione wnioski są z eksperymentów dla pliku **A-n80-k10**.

2.1 Wpływ liczby mrówek (m)

Badano wartości: $m \in \{10, 20, 50, 100\}$.



Rysunek 1: Wpływ liczby mrówek na jakość rozwiązania.

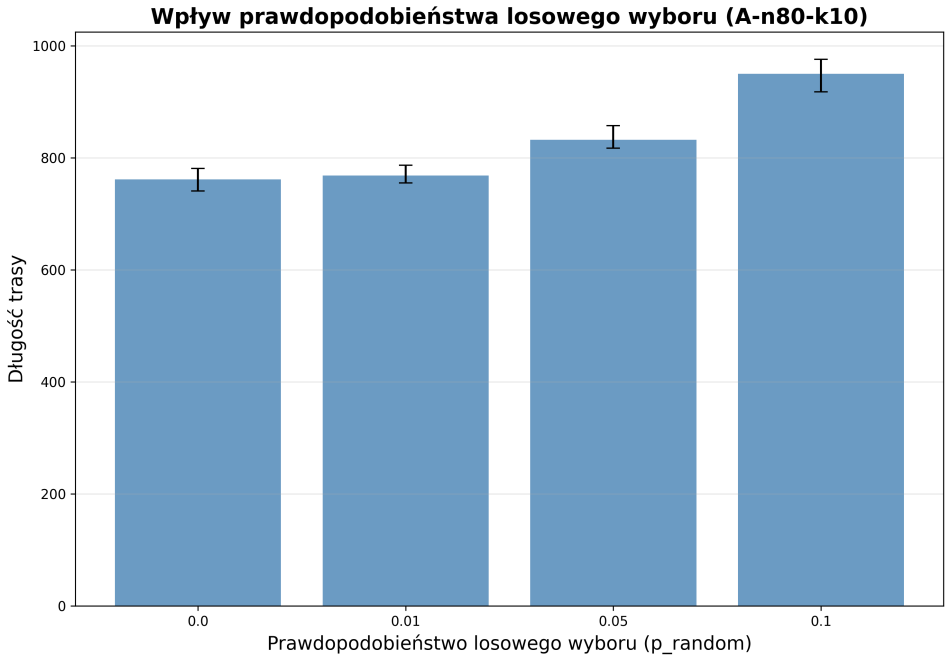
Większa liczba mrówek zwiększa eksplorację przestrzeni rozwiązań, co zazwyczaj prowadzi do lepszych wyników. Jednak wiąże się to z dłuższym czasem obliczeń.

Obserwacje:

- Zwiększenie liczby mrówek z 10 do 50 znacząco poprawia jakość rozwiązań.
- Dalsze zwiększanie (do 100) daje marginalne korzyści przy znacznym wzroście czasu obliczeń.
- Optymalna wartość to $m \approx 20 - 50$ dla badanych instancji.

2.2 Wpływ prawdopodobieństwa losowego wyboru (p_{random})

Badano wartości: $p_{random} \in \{0.0, 0.01, 0.05, 0.1\}$.



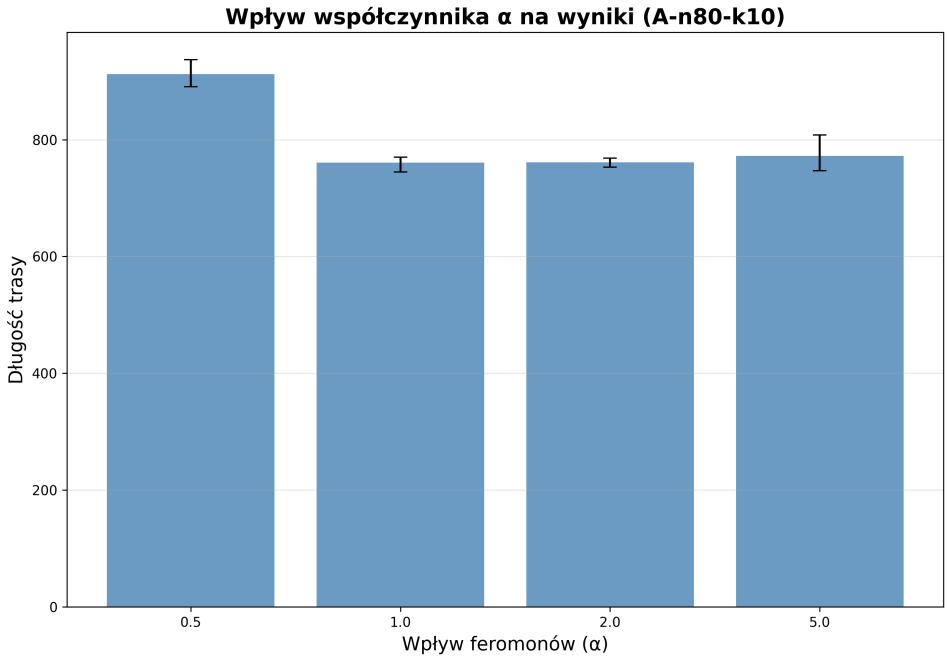
Rysunek 2: Wpływ prawdopodobieństwa losowego wyboru na jakość rozwiązania.

Parametr ten kontroluje eksplorację vs eksploatację.
Obserwacje:

- Niewielkie wartości ($p_{random} = 0.01$) mogą pomóc uniknąć lokalnych minimów.
- Zbyt duże wartości ($p_{random} > 0.1$) wprowadzają zbyt dużo losowości i pogarszają wyniki.
- Najlepsze wyniki uzyskano dla $p_{random} = 0$ lub $p_{random} = 0.01$.

2.3 Wpływ współczynnika α (wpływ feromonów)

Badano wartości: $\alpha \in \{0.5, 1.0, 2.0, 5.0\}$.



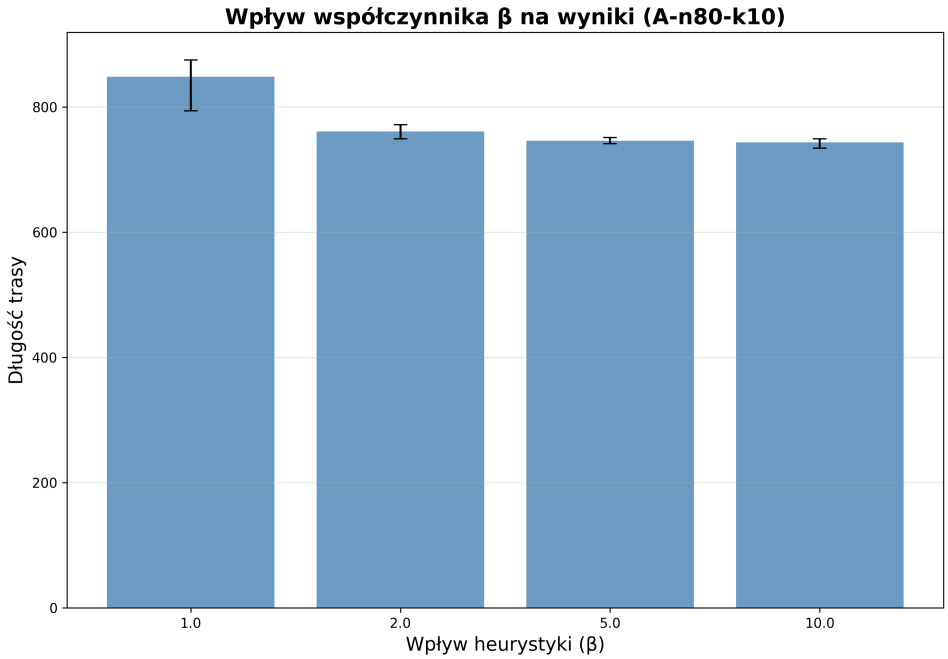
Rysunek 3: Wpływ współczynnika α na jakość rozwiązania.

Obserwacje:

- $\alpha = 1.0$ daje dobre wyniki jako punkt wyjścia.
- Zbyt wysokie wartości ($\alpha = 5.0$) prowadzą do zbyt szybkiej zbieżności i utknięcia w lokalnych minimach.
- Zbyt niskie wartości osłabiają wpływ uczenia się z poprzednich iteracji.

2.4 Wpływ współczynnika β (wpływ heurystyki)

Badano wartości: $\beta \in \{1.0, 2.0, 5.0, 10.0\}$.



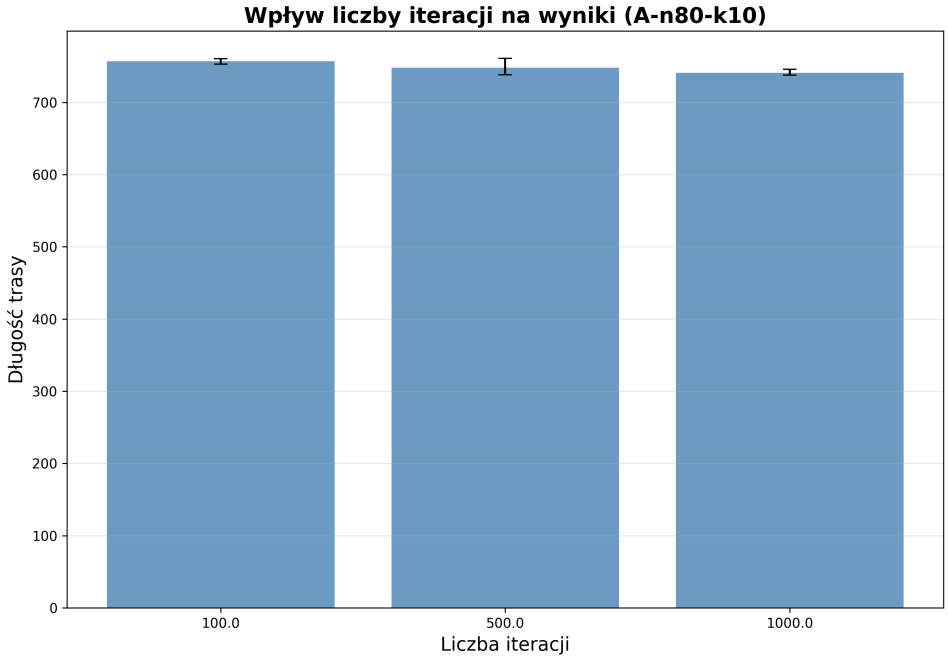
Rysunek 4: Wpływ współczynnika β na jakość rozwiązania.

Obserwacje:

- $\beta = 2.0$ jest dobrą wartością bazową.
- Wyższe wartości ($\beta = 5.0$) faworyzują wybór bliskich wierzchołków (zachłanność).
- Dla mniejszych instancji wyższe β może być korzystne.

2.5 Wpływ liczby iteracji (T)

Badano wartości: $T \in \{100, 500, 1000\}$.



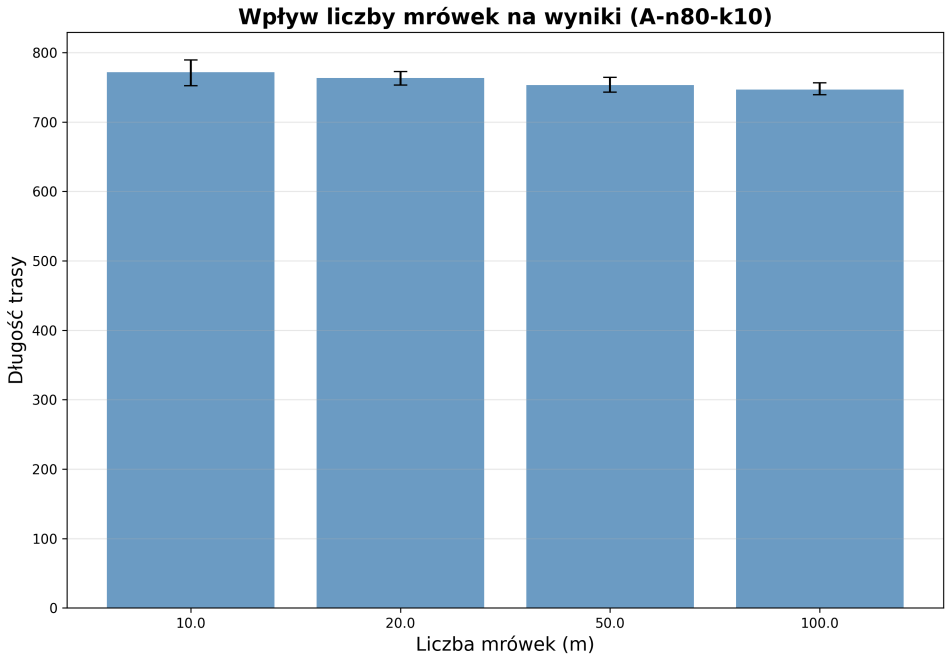
Rysunek 5: Wpływ liczby iteracji na jakość rozwiązania.

Obserwacje:

- Więcej iteracji zazwyczaj prowadzi do lepszych wyników.
- Dla mniejszej instancji (32 atrakcje) wystarczy $T = 100 - 500$.
- Dla większej instancji (80 atrakcji) korzystne jest $T \geq 500$.

2.6 Wpływ współczynnika wyparowywania (ρ)

Badano wartości: $\rho \in \{0.1, 0.3, 0.5, 0.8\}$.



Rysunek 6: Wpływ współczynnika wyparowywania ρ na jakość rozwiązania.

Obserwacje:

- $\rho = 0.5$ stanowi dobry kompromis.
- Zbyt niskie ρ (wolne wyparowywanie) prowadzi do zbyt silnego wpływu starych ścieżek.
- Zbyt wysokie ρ (szybkie wyparowywanie) może utracić dobre rozwiązania.

3 Wnioski

Na podstawie przeprowadzonych eksperymentów sformułowano następujące wnioski:

- **Skuteczność algorytmu**: Algorytm mrówkowy skutecznie znajduje dobre rozwiązania problemu komiwojażera dla obu badanych instancji.
- **Wpływ liczby mrówek**: Większa kolonia zwiększa jakość rozwiązań kosztem czasu obliczeń. Zalecana wartość: $m = 20 - 50$.
- **Losowy wybór**: Niewielkie prawdopodobieństwo losowego wyboru ($p_{random} = 0.01$) może pomóc w eksploracji, ale zbyt duże wartości pogarszają wyniki.
- **Balans α i β**: Typowe wartości $\alpha = 1.0$ i $\beta = 2.0$ dają dobre wyniki. Można dostosować w zależności od charakterystyki problemu.
- **Liczba iteracji**: Więcej iteracji poprawia wyniki, ale z malejącym efektem. Dla badanych instancji $T = 100 - 500$ jest wystarczające.
- **Wyparowywanie**: Wartość $\rho = 0.5$ stanowi dobry kompromis między eksploracją a eksploatacją.

3.1 Rekomendacje parametrów

Na podstawie eksperymentów rekomendujemy następujące zakresy parametrów:

Parametr	**Zalecana wartość**
Liczba mrówek (m)	20–50
Wpływ feromonów (α)	1.0
Wpływ heurystyki (β)	2.0–5.0
Wyparowywanie (ρ)	0.3–0.5
Iteracje (T)	100–500
Losowy wybór (p_{random})	0.0–0.01

3.2 Ograniczenia eksperymentu

- Ograniczony czas obliczeń — nie testowano pełnego gridu parametrów.
- Każda konfiguracja była uruchomiona 5 razy — większa liczba powtórzeń dałaby dokładniejsze statystyki.
- Testowano tylko dwa rozmiary instancji (32 i 80 atrakcji).
- Nie testowano innych wariantów ACO (np. MMAS, ACS).