

Metaheurystyki — zadanie 4

Algorytm mrówkowy (ACO)

GRUPA 3 — piątek 10:15

Bartosz Kołaciński
251554

Nikodem Nowak
251598

19 grudnia 2025

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

Spis treści

1	Opis zasad działania algorytmu	3
1.1	Opis problemu	3
1.2	Opis algorytmu mrówkowego (ACO)	3
1.2.1	Pseudokod algorytmu	3
1.2.2	Parametry algorytmu	3
1.3	Opis implementacji rozwiązania	4
1.3.1	Reprezentacja grafu i obliczanie odległości	4
1.3.2	Reprezentacja trasy i kosztu	4
1.3.3	Inicjalizacja feromonów	4
1.3.4	Mechanizm konstrukcji trasy (reguła wyboru)	4
1.3.5	Aktualizacja feromonów i wyparowywanie	5
1.4	Instrukcja uruchomienia programu	6
2	Eksperymenty i wyniki	7
2.1	Najlepsze znalezione rozwiązania	7
2.2	Wpływ liczby mrówek (m)	9
2.3	Wpływ prawdopodobieństwa losowego wyboru (p_{random})	10
2.4	Wpływ współczynnika α (wpływ feromonów)	11
2.5	Wpływ współczynnika β (wpływ heurystyki)	12
2.6	Wpływ liczby iteracji (T)	13
2.7	Wpływ współczynnika wyparowywania (ρ)	14
3	Wnioski	15
3.1	Rekomendowane parametry	15
3.2	Ograniczenia eksperymentu	15

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)

```
1: Inicjalizacja:
2: Zainicjuj macierz feromonów  $\tau_{ij} \leftarrow 1.0$  dla wszystkich krawędzi  $(i, j)$ 
3: Oblicz macierz odległości  $d_{ij}$  dla wszystkich par wierzchołków
4: for  $t = 1$  to  $T$  (liczba iteracji) do
5:   Konstrukcja tras:
6:   for każda mrówka  $k = 1, \dots, m$  do
7:     Wybierz losowy wierzchołek startowy
8:     while nie odwiedzone wszystkich wierzchołków do
9:       Wybierz następny wierzchołek  $j$  z prawdopodobieństwem:
10:      
$$p_{ij} = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \text{dozwolone}} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta}$$

11:      gdzie  $\eta_{ij} = 1/d_{ij}$  (heurystyka odwrotności odległości)
12:    end while
13:    Oblicz długość trasy mrówki  $k$ 
14:  end for
15:  Aktualizacja feromonów:
16:  Wyparowanie:  $\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij}$  dla wszystkich  $(i, j)$ 
17:  for każda mrówka  $k$  do
18:    Depozyt:  $\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^k$  na trasie mrówki
19:    gdzie  $\Delta\tau_{ij}^k = 1/L_k$  ( $L_k$  — długość trasy mrówki  $k$ )
20:  end for
21:  Zaktualizuj najlepsze znalezione rozwiązanie
22: end for
23: 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.

```
1 def read_data(filepath: str, separator: str = ' ') -> tuple:
2     from math import dist
3     import pandas as pd
4
5     data_csv = list(pd.read_csv(
6         filepath, sep=separator, skipinitialspace=True, header=None
7     ).to_records(index=False))
8     data_csv = [(int(row[0]), int(row[1]), int(row[2])) for row in data_csv]
9
10    data_length = len(data_csv)
11    distances_matrix = [[0.0] * data_length for _ in range(data_length)]
12    for i in range(data_length):
13        for j in range(data_length):
14            if i != j:
15                p1 = (data_csv[i][1], data_csv[i][2])
16                p2 = (data_csv[j][1], data_csv[j][2])
17                distances_matrix[i][j] = dist(p1, p2)
18
19    index_to_id_map = {i: rec[0] for i, rec in enumerate(data_csv)}
20    return data_csv, distances_matrix, index_to_id_map
21
```

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.

```
1 def get_path_length(self) -> float:
2     length = 0.0
3     for i in range(len(self.visited) - 1):
4         l1 = self.visited[i]
5         l2 = self.visited[i + 1]
6         length += self.distances_matrix[l1][l2]
7     return length
8
```

Kod 2: Obliczanie długości trasy

1.3.3 Inicjalizacja feromonów

Macierz feromonów jest inicjalizowana wartością 1.0 dla wszystkich krawędzi.

```
1 self.pheromones = [
2     [1.0] * self.num_attractions for _ in range(self.num_attractions)
3 ]
4
```

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).

```

1 def select_next_node(self, p_random: float) -> None:
2     current_node = self.visited[-1]
3     allowed_nodes = list(set(range(self.num_attractions)) - set(self.visited))
4
5     if not allowed_nodes:
6         return
7
8     # Losowy wybor (eksploracja)
9     if random.random() < p_random:
10         next_node = random.choice(allowed_nodes)
11         self.visited.append(next_node)
12         return
13
14     # Wybor na podstawie feromonow i heurystyki
15     possibilities = self._calculate_possibilities(current_node, allowed_nodes)
16     next_node = random.choices(allowed_nodes, weights=possibilities)[0]
17     self.visited.append(next_node)
18
19 def _calculate_possibilities(self, current_node, allowed_nodes):
20     scores = []
21     denominator = 0.0
22
23     for node in allowed_nodes:
24         tau = Ant.pheromones[current_node][node]
25         distance = Ant.distances_matrix[current_node][node]
26         if distance == 0:
27             distance = 1e-10 # Obsluga zerowej odleglosci
28         eta = 1.0 / distance
29
30         score = (tau ** Ant.alpha) * (eta ** Ant.beta)
31         scores.append(score)
32         denominator += score
33
34     if denominator == 0.0:
35         return [1.0 / len(allowed_nodes)] * len(allowed_nodes)
36
37     return [score / denominator for score in scores]
38

```

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.

```

1 def _update_pheromones(self, ants):
2     evaporation_factor = 1.0 - self.rho
3
4     # Wyparowywanie feromonow
5     for i in range(self.num_attractions):
6         for j in range(self.num_attractions):
7             self.pheromones[i][j] *= evaporation_factor
8
9     # Depozyt feromonow
10    for ant in ants:
11        path = ant.get_visited()
12        path_length = ant.get_path_length()
13
14        if path_length > 0:
15            deposit = 1.0 / path_length
16            for i in range(len(path) - 1):
17                from_node = path[i]
18                to_node = path[i + 1]
19                self.pheromones[from_node][to_node] += deposit
20                self.pheromones[to_node][from_node] += deposit
21

```

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

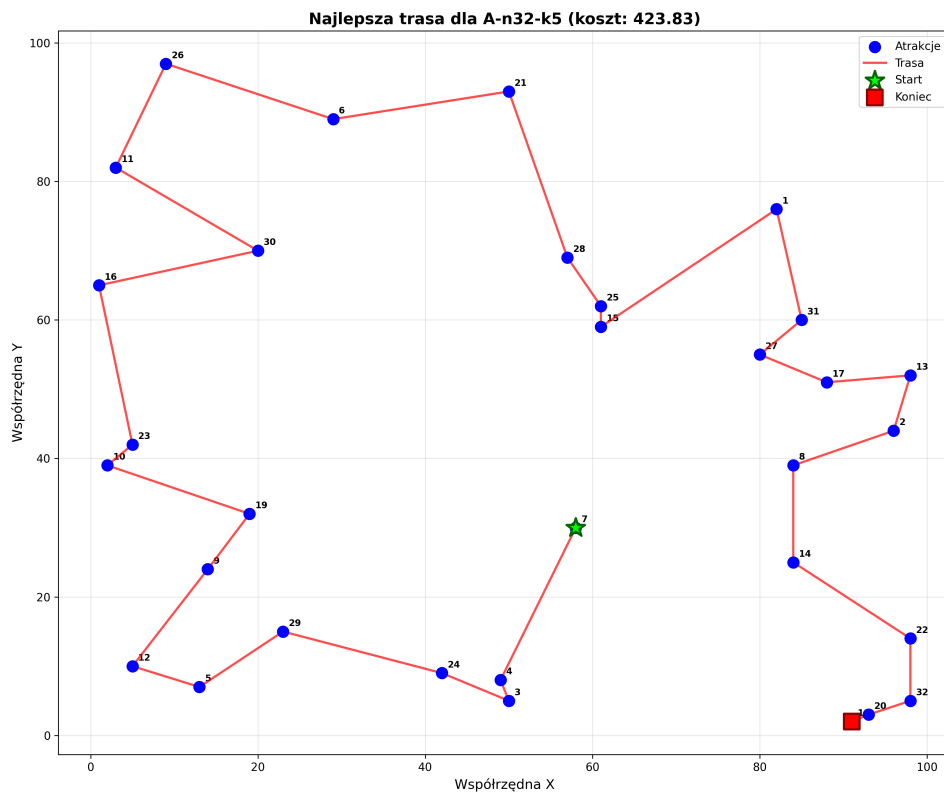
Przeprowadzono serię eksperymentów dla dwóch zbiorów danych:

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

Dla każdej konfiguracji algorytm był uruchamiany 5 razy w celu uzyskania statystyk. Wykresy z analizą wpływu parametrów przedstawione w dalszej części dotyczą instancji **A-n80-k10**, jako że jest ona bardziej wymagająca obliczeniowo i lepiej pokazuje różnice między konfiguracjami.

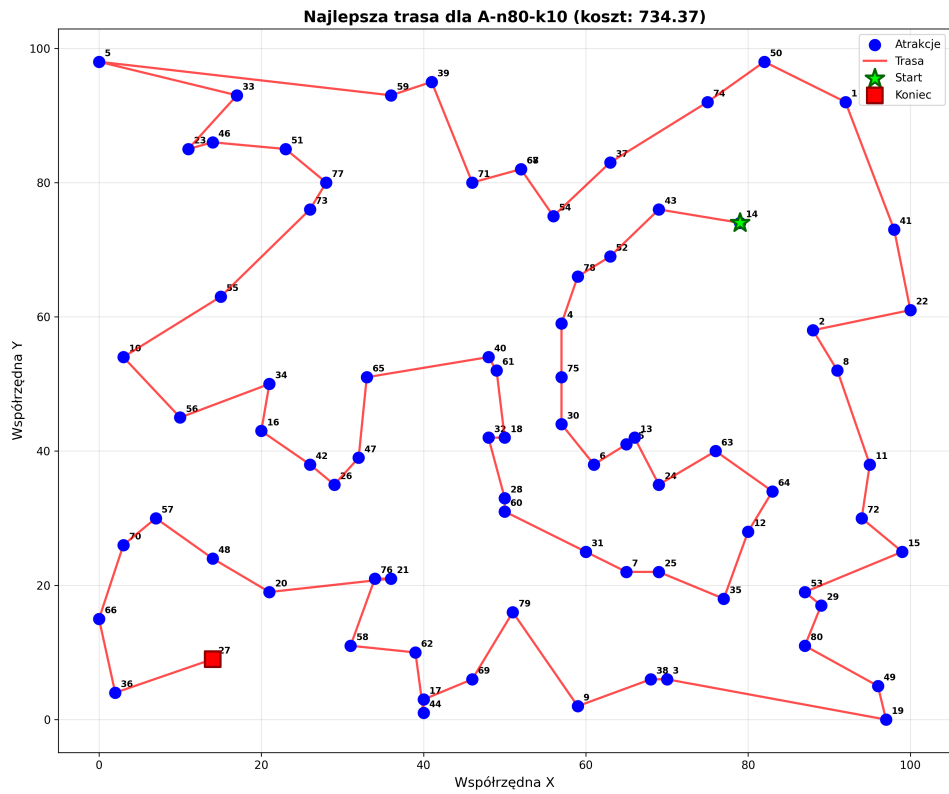
2.1 Najlepsze znalezione rozwiązania

Poniżej przedstawiono wizualizacje najlepszych tras znalezionych przez algorytm dla obu instancji testowych.



Rysunek 1: Najlepsza znaleziona trasa dla instancji A-n32-k5. Zielona gwiazdka oznacza punkt startowy, a czerwony kwadrat — punkt końcowy trasy. Uzyskany koszt to 423.83 jednostki.

Dla mniejszej instancji (32 atrakcje) uzyskana trasa nie zawiera przecięć, a punkty są odwiedzane w logicznej kolejności

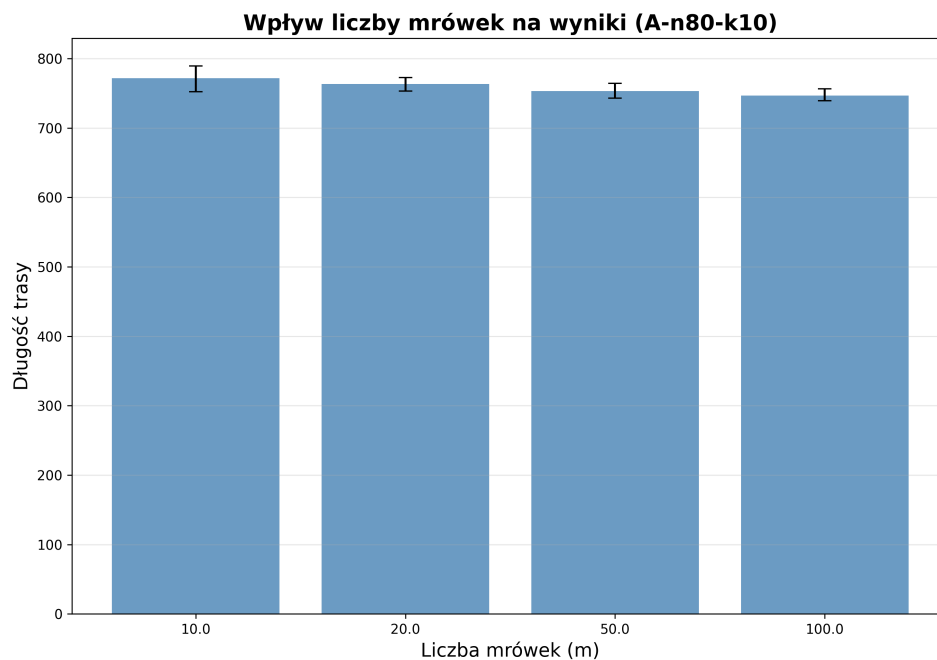


Rysunek 2: Najlepsza znaleziona trasa dla instancji A-n80-k10. Przy 80 atrakcjach problem jest znacznie trudniejszy, ale algorytm mrówkowy nadal znajduje rozsądne rozwiązanie o koszcie 734.37.

Dla większej instancji (80 atrakcji) algorytm również znajduje rozwiązanie bez znaczących przecięć.

2.2 Wpływ liczby mrówek (m)

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



Rysunek 3: Wpływ liczby mrówek na średnią długość trasy. Słupki błędów pokazują odchylenie standardowe z 5 uruchomień. Im niższy słupek, tym lepszy wynik.

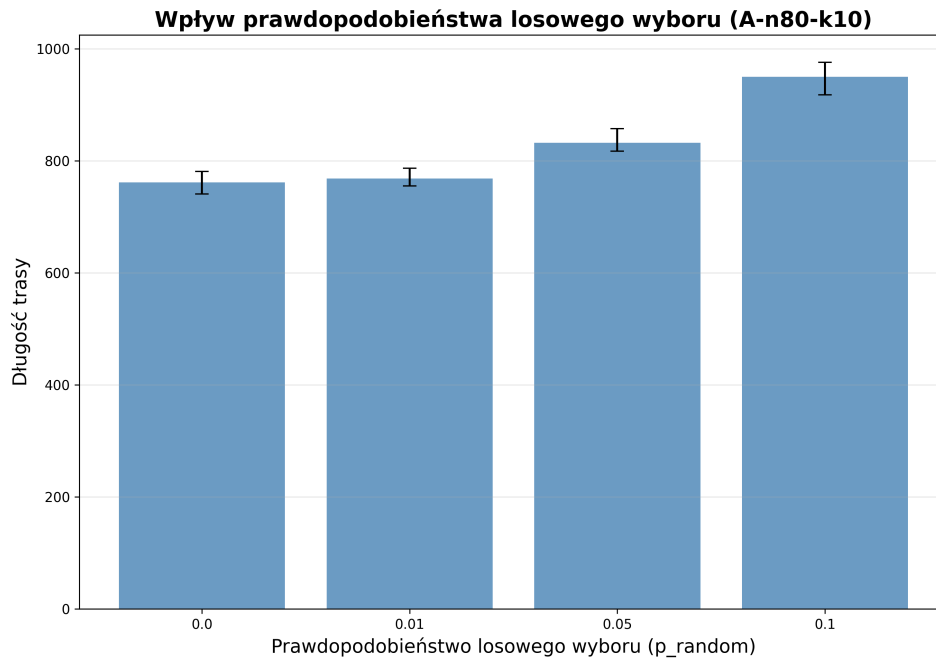
Różnice między poszczególnymi wartościami są umiarkowane — wszystkie wyniki mieszczą się w przedziale 750–770.

Obserwacje:

- Przy $m = 10$ uzyskano najgorszy średni wynik (~ 770) oraz największe odchylenie standardowe.
- Zwiększenie do $m = 50$ – 100 daje poprawę o około 15–20 jednostek i zmniejsza wariancję wyników.
- Wartość $m = 20$ – 50 stanowi rozsądny kompromis między jakością rozwiązania a czasem obliczeń.

2.3 Wpływ prawdopodobieństwa losowego wyboru (p_{random})

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



Rysunek 4: Wpływ prawdopodobieństwa losowego wyboru na średnią długość trasy. Wyraźnie widać, że zbyt duża losowość psuje wyniki.

Ten parametr wykazuje najsilniejszy wpływ na jakość rozwiązania spośród wszystkich badanych.

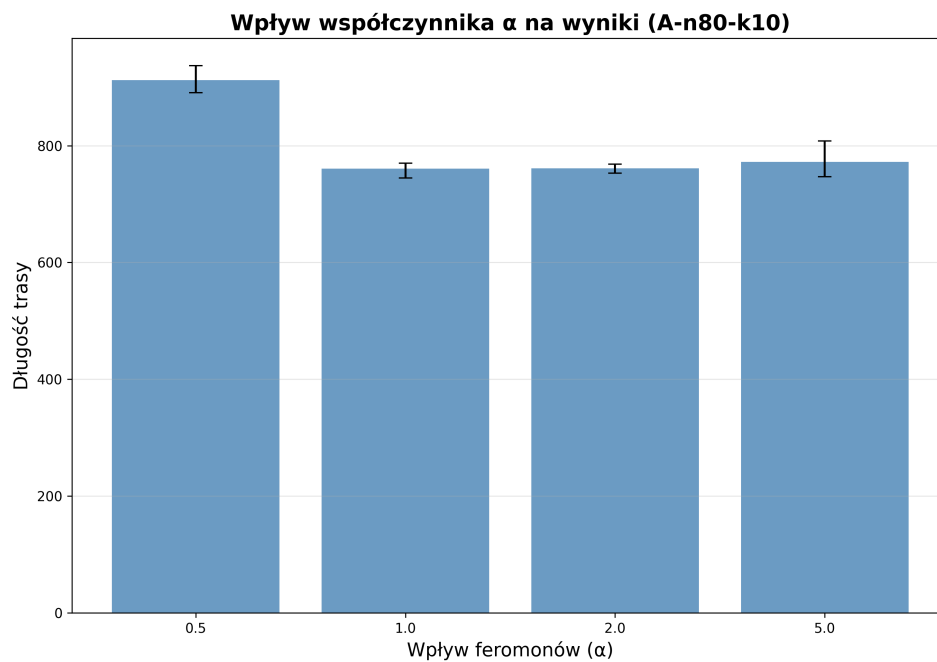
Obserwacje:

- Wartości $p_{random} = 0.0$ i $p_{random} = 0.01$ dają praktycznie identyczne wyniki (~ 760).
- Przy $p_{random} = 0.05$ średni koszt rośnie do około 830.
- Przy $p_{random} = 0.1$ wynik pogarsza się do ~ 945 — prawie 200 jednostek więcej niż bez losowości.

Zbyt duże prawdopodobieństwo losowego wyboru zaburza proces uczenia oparty na feromonach. Zalecana wartość: $p_{random} \leq 0.01$.

2.4 Wpływ współczynnika α (wpływ feromonów)

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



Rysunek 5: Wpływ współczynnika α na średnią długość trasy. Zbyt niska wartość α oznacza ignorowanie feromonów.

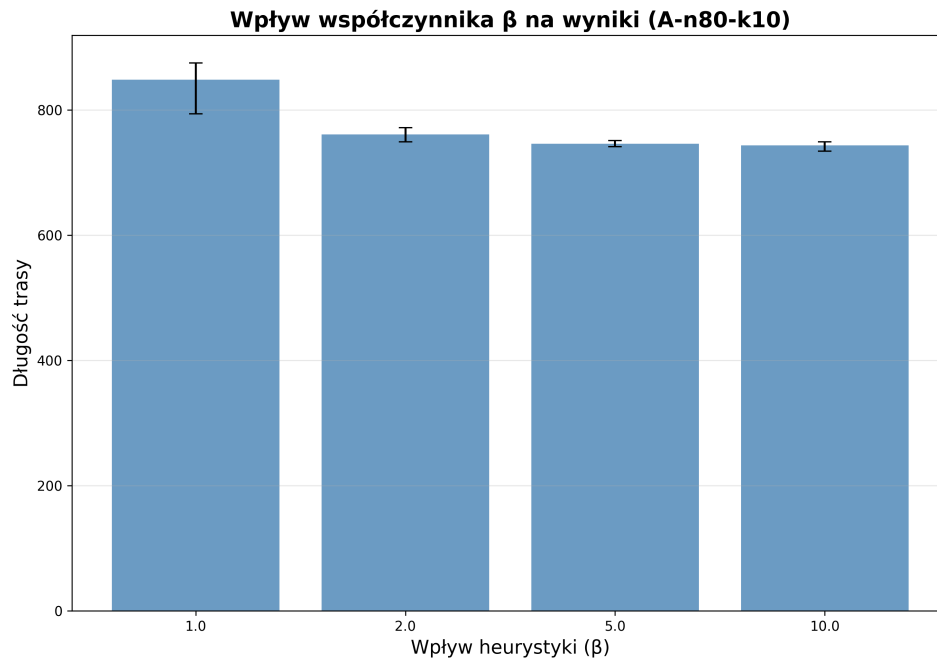
Obserwacje:

- $\alpha = 0.5$ daje najgorszy wynik (~ 900) — zbyt słaby wpływ feromonów ogranicza uczenie kolektywne.
- $\alpha = 1.0$ i $\alpha = 2.0$ dają podobne, dobre wyniki (~ 760).
- $\alpha = 5.0$ jest nieznacznie gorsze (~ 770) z większą wariancją — zbyt silny wpływ feromonów powoduje przedwczesną zbieżność.

Optymalne wartości to $\alpha = 1.0$ – 2.0 , zapewniające równowagę między eksploracją a eksploatacją.

2.5 Wpływ współczynnika β (wpływ heurystyki)

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



Rysunek 6: Wpływ współczynnika β na średnią długość trasy. Wyższe β oznacza większą "zachłanność" — mrówki preferują bliskie wierzchołki.

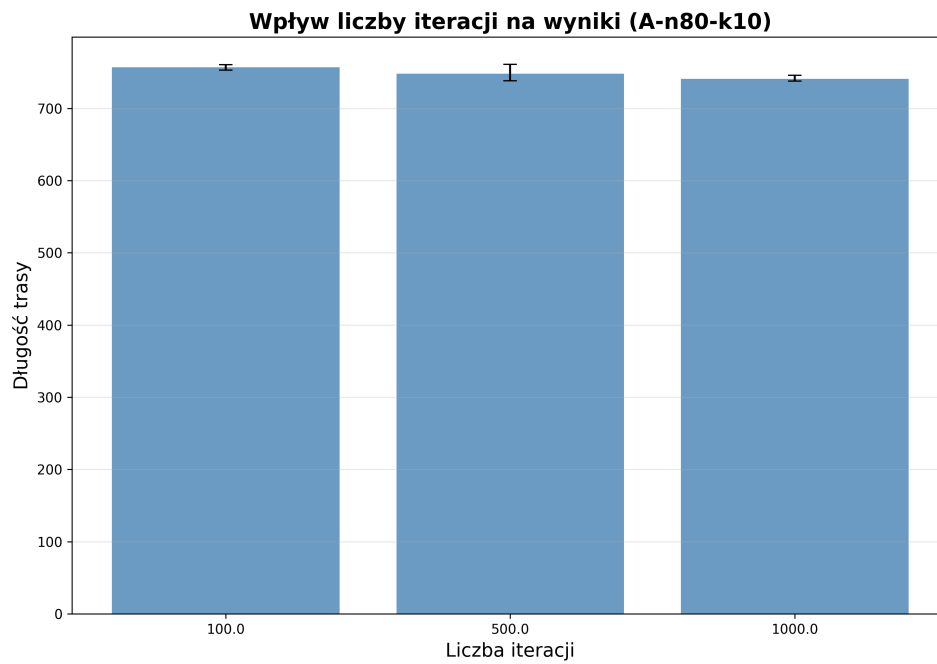
Obserwacje:

- $\beta = 1.0$ daje najgorszy wynik (~ 850) — zbyt słaba preferencja bliskich wierzchołków.
- $\beta = 2.0$ znacząco poprawia wynik (~ 760).
- $\beta = 5.0$ i $\beta = 10.0$ dają najlepsze wyniki (~ 745 – 750).

Wysokie wartości β nie powodują utknięcia w lokalnych minimach. Silna heurystyka odległościowa jest korzystna dla problemu TSP.

2.6 Wpływ liczby iteracji (T)

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



Rysunek 7: Wpływ liczby iteracji na średnią długość trasy. Więcej iteracji = więcej czasu na szukanie, ale zysk jest niewielki.

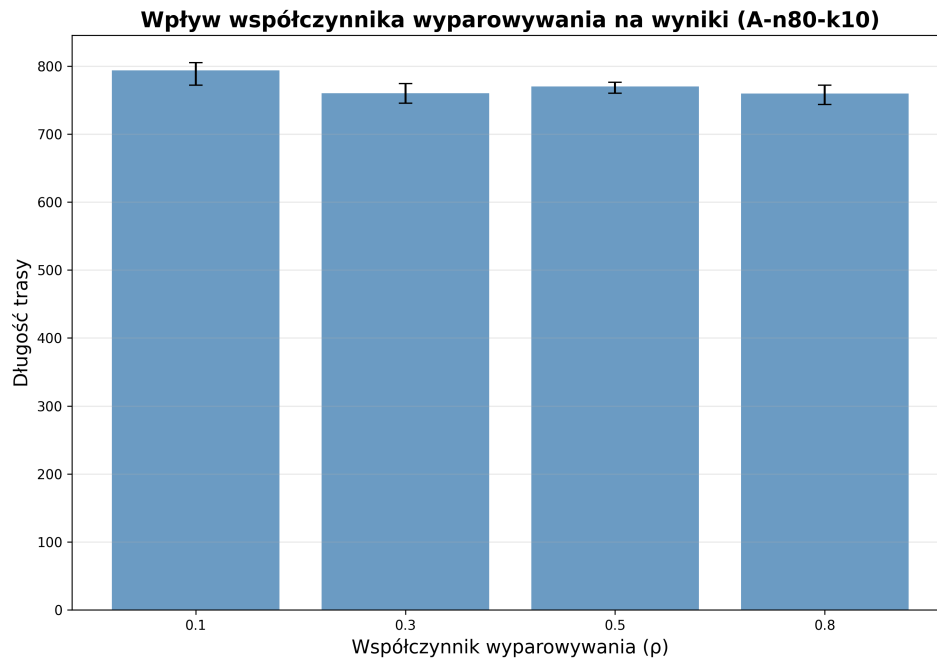
Obserwacje:

- Różnice są niewielkie — wszystkie wyniki mieszczą się w przedziale 740–755.
- Przy $T = 100$ średni koszt wynosi około 755, przy $T = 1000$ — około 740.
- Dziesięciokrotny wzrost liczby iteracji daje poprawę o jedynie 15 jednostek.

Algorytm szybko osiąga stabilizację. Dla badanych instancji $T = 100$ –500 jest wystarczające.

2.7 Wpływ współczynnika wyparowywania (ρ)

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



Rysunek 8: Wpływ współczynnika wyparowywania ρ na średnią długość trasy. Parametr kontroluje, jak szybko "zapominane" są stare ścieżki.

Co widzimy:

- $\rho = 0.1$ (wolne wyparowywanie) daje najgorszy wynik (~ 795) — stare feromony zbyt długo wpływają na decyzje.
- $\rho = 0.3$, $\rho = 0.5$ i $\rho = 0.8$ dają bardzo podobne wyniki (~ 760 – 765).
- W praktyce nie ma dużej różnicy między wartościami 0.3–0.8.

Kluczowe jest unikanie zbyt niskiej wartości ρ , która powoduje nadmierne utrzymywanie starych ścieżek feromonowych.

3 Wnioski

Na podstawie przeprowadzonych eksperymentów sformułowano następujące wnioski:

- **Skuteczność algorytmu:** Dla obu instancji uzyskano trasy o kosztach 423.83 (32 atrakcje) i 734.37 (80 atrakcji).
- **Liczba mrówek:** Zwiększenie z 10 do 100 mrówek daje poprawę o około 20 jednostek. Zalecana wartość: $m = 20\text{--}50$.
- **Prawdopodobieństwo losowego wyboru:** Wartości $p_{\text{random}} > 0.01$ znacząco pogarszają wyniki (nawet o 200 jednostek dla $p_{\text{random}} = 0.1$). Zalecane: $p_{\text{random}} \leq 0.01$.
- **Wpływ heurystyki (β):** Wyższe wartości ($\beta = 5.0\text{--}10.0$) dają lepsze wyniki niż standardowe $\beta = 2.0$. Silniejsza preferencja bliskich wierzchołków poprawia jakość tras.
- **Wpływ feromonów (α):** Optymalne wartości to $\alpha = 1.0\text{--}2.0$. Zbyt niskie ($\alpha = 0.5$) osłabia uczenie, zbyt wysokie ($\alpha = 5.0$) przyspiesza przedwczesną zbieżność.
- **Liczba iteracji:** Różnica między 100 a 1000 iteracji wynosi około 15 jednostek — algorytm szybko osiąga stabilizację.
- **Współczynnik wyparowywania:** Wartość $\rho = 0.1$ daje gorsze wyniki, natomiast $\rho \in \{0.3, 0.5, 0.8\}$ działają porównywalnie.

3.1 Rekomendowane parametry

Na podstawie naszych eksperymentów sugerujemy:

Parametr	Zalecana wartość
Liczba mrówek (m)	20–50
Wpływ feromonów (α)	1.0–2.0
Wpływ heurystyki (β)	5.0–10.0
Wyparowywanie (ρ)	0.3–0.5
Iteracje (T)	100–500
Losowy wybór (p_{random})	0.0–0.01

Warto zauważyć, że zalecane wartości β są wyższe niż typowe wartości literaturowe ($\beta = 2\text{--}5$), co sugeruje, że dla problemu TSP silniejsza heurystyka odległościowa jest korzystna.

3.2 Ograniczenia eksperymentu

- Każda konfiguracja była uruchomiona 5 razy — większa liczba powtórzeń zwiększyłaby wiarygodność statystyczną.
- Testowano tylko dwie instancje (32 i 80 atrakcji) — generalizacja wniosków na większe problemy wymaga dodatkowych badań.