

# Metaheurystyki i ich zastosowania 2023/24

## Zadanie 4 – algorytm mrówkowy

### Autorzy:

Michał Ferdzyn 242383

Artur Grzybek 242399

### 1. Zasada działania programu

Program oparty jest na algorytmie mrówkowym, heurystycznej metaheurystyce inspirowanej zachowaniami mrówek poszukujących pożywienia. Algorytm ten został zastosowany do rozwiązania problemu komiwojażera, czyli znalezienia najkrótszej trasy.

### Działanie programu:

#### 1. Określenie parametrów algorytmu:

- rozmiar populacji mrówek,
- czynnik losowy (współczynnik decydujący o tym czy mrówki wybierają trasy losowo czy na podstawie informacji o feromonach),
- parametry Alfa i Beta (wpływają na to, jak bardzo mrówki są skłonne wybierać trasy oparte na feromonach lub odległościach),
- czynnik parowania feromonów (oznacza szybsze „zapominanie” starych informacji i bardziej dynamiczną aktualizację feromonów)
- liczbę iteracji

2. **Inicjalizacja feromonów:** Początkowo wszystkie krawędzie grafu (trasa między atrakcjami) są inicjalizowane równymi wartościami feromonów.

3. **Wędrówki mrówek:** Każda mrówka rozpoczyna trasę z losowego punktu. W każdym kroku, mrówka decyduje o kolejnym atrakcji, biorąc pod uwagę poziom feromonów i odległość od obecnie odwiedzanej atrakcji. Istnieje również losowy czynnik, który pozwala na zastosowanie przypadkowego wyboru atrakcji.

4. **Aktualizacja feromonów:** Po przejściu wszystkich mrówek, feromony są aktualizowane. Krótsze trasy otrzymują większą ilość feromonów, symulując "zapamiętywanie" skróconych tras.

5. **Powtarzanie kroków:** Kroki 2 i 3 są powtarzane przez określoną liczbę iteracji.

## 2. Wybrane miejsca implementacji

**Klasa Plansza.py** - klasa Plansza jest odpowiedzialna za przechowywanie informacji o miejscach, feromonach i odległościach między nimi, a także za aktualizację poziomu feromonów po przejściu mrówek.

```
class Plansza:
    def __init__(self, sciezka_pliku):
        plik = open(sciezka_pliku, "r")
        punkty = [[int(i) for i in linia.split(" ")] for linia in plik]
        plik.close()
        self.tytul = sciezka_pliku
        self.miejscia = [punkt[0] - 1 for punkt in punkty]
        self.pozycje = [(punkt[1], punkt[2]) for punkt in punkty]
        self.feromony = [[1 for _ in punkty] for _ in punkty]
        self.odleglosci = [[oblicz_odleglosc(i, j) for j in punkty] for i in punkty]
```

**Konstruktor \_\_init\_\_(self, sciezka\_pliku):**

- Inicjalizuje obiekt klasy Plansza.
- Otwiera plik o ścieżce *sciezka\_pliku* i wczytuje z niego punkty, z których utworzona zostanie plansza.
- Atrybut *tytul* przechowuje tytuł planszy (ścieżkę pliku). Atrybut *miejscia* przechowuje identyfikatory miejsc na planszy (indeksy punktów -1).
- Atrybut *pozycje* przechowuje współrzędne punktów na planszy.
- Atrybut *feromony* to macierz feromonów między poszczególnymi punktami, zainicjowana jedynkami.
- Atrybut *odleglosci* to macierz odległości między punktami na planszy.

```
def aktualizuj_feromony(self, czynnik, mrowki):
    self._paruj_feromony(czynnik)
    for mrowka in mrowki:
        self.intensyfikuj_feromony(mrowka)
```

**Metoda aktualizuj\_feromony(self, czynnik, mrowki):**

- Aktualizuje poziom feromonów na planszy po przejściu mrówek.
- Wywołuje prywatną metodę *\_paruj\_feromony(czynnik)* do parowania feromonów.
- Wywołuje prywatną metodę *\_intensyfikuj\_feromony(mrowka)* do intensyfikacji feromonów w śladzie mrówki.

```
def _paruj_feromony(self, czynnik):
    for i in range(len(self.feromony)):
        for j in range(len(self.feromony[0])):
            self.feromony[i][j] -= self.feromony[i][j] * czynnik
```

**Prywatna metoda \_paruj\_feromony(self, czynnik):**

- Paruje feromony, czyli zmniejsza ich wartość o pewien czynnik, symulując proces "zapominania".

```
def _intensyfikuj_feromony(self, mrowka):
    for i in range(len(mrowka.odwiedzone_miejsca) - 1):
        self.feromony[mrowka.odwiedzone_miejsca[i]][mrowka.odwiedzone_miejsca[i + 1]] += (1 / mrowka.odleglosc_przebyta(self))
```

### Prywatna metoda `_intensyfikuj_feromony(self, mrowka)`:

- Intensyfikuje feromony, czyli zwiększa wartość feromonów na trasie przebytej przez daną mrówkę.
- Im krótsza trasa, tym większa intensyfikacja feromonów.

```
def oblicz_odleglosc(p1, p2):
    return np.linalg.norm(np.array(p2) - np.array(p1))
```

### Funkcja `oblicz_odleglosc(p1, p2)`:

- Ta funkcja wykorzystuje bibliotekę NumPy do obliczania odległości między dwoma punktami p1 i p2 w przestrzeni dwuwymiarowej.
- Funkcja korzysta z normy euklidesowej, obliczając długość wektora różnicy między dwoma punktami.

## Klasa Mrowka.py

```
class Mrowka:
    def __init__(self, punkt_startowy):
        self.odwiedzone_miejsca = [punkt_startowy]
```

### Konstruktor `__init__(self, punkt_startowy)`:

- Inicjalizuje obiekt klasy Mrowka z punktu startowego.
- Atrybut `odwiedzone_miejsca` przechowuje listę miejsc odwiedzonych przez mrówkę.

```
def odleglosc_przebyta(self, plansza):
    suma_odleglosci = 0
    for i in range(len(self.odwiedzone_miejsca) - 1):
        suma_odleglosci +=
            plansza.odleglosci[self.odwiedzone_miejsca[i]][self.odwiedzone_miejsca[i + 1]]
    return suma_odleglosci
```

### Metoda `odleglosc_przebyta(self, plansza)`:

- Oblicza odległość przebytą przez mrówkę na podstawie planszy.
- Sumuje odległości między kolejnymi odwiedzonymi miejscami.

```
def nastepny_krok(self, plansza, alfa, beta, czynnik_losowy):
    nieodwiedzone_miejsca = [a for a in plansza.miejsca if a not in
self.odwiedzone_miejsca]
    obecne_miejsce = self.odwiedzone_miejsca[-1]

    if random.random() <= czynnik_losowy:
        self.odwiedzone_miejsca.append(losowa_selekcja(nieodwiedzone_miejsca))
    else:
        wagi = [oblicz_wage(plansza.feromony[obecne_miejsce][nastepne_miejsce],
                           plansza.odleglosci[obecne_miejsce][nastepne_miejsce]),
```

```
alfa, beta)
        for nastepne_miejsce in nieodwiedzone_miejsca]
        prawdopodobienstwa = [waga / sum(wagi) for waga in wagi]
        self.odwiedzone_miejsca.append(selekcia_ruletkowa(nieodwiedzone_miejsca,
prawdopodobienstwa))
```

#### Metoda *nastepny\_krok(self, plansza, alfa, beta, czynnik\_losowy)*:

- Wybiera kolejne miejsce do odwiedzenia na podstawie reguł algorytmu mrówkowego.
- Uwzględnia zarówno losowy wybór (eksploracja), jak i wybór na podstawie wag (eksploatacja).
- Dodaje wybrane miejsce do listy odwiedzonych miejsc.
- W przypadku losowego wyboru:
  1. Sprawdza, czy losowa liczba z przedziału [0, 1) jest mniejsza niż *czynnik\_losowy*.
  2. Jeśli tak, dodaje losowe nieodwiedzone miejsce do listy odwiedzonych.
- W przypadku wyboru na podstawie wag:
  1. Oblicza wagę dla dostępnych nieodwiedzonych miejsc.
  2. Wybiera miejsce na podstawie ruletki, gdzie szanse wyboru są proporcjonalne do wag.

```
def selekcia_ruletkowa(miejsca, wagi):
    return choice(miejsca, None, True, wagi)
```

#### Funkcja *selekcia\_ruletkowa(miejsca, wagi)*:

- Ta funkcja implementuje selekcję ruletkową, czyli wybór elementu z zestawu na podstawie wag.
- Wykorzystuje funkcję choice z biblioteki NumPy do losowego wyboru elementu z listy miejsca z zadanymi wagami wagi.

```
def losowa_selekcia(miejsca):
    return miejsca[random.randint(0, len(miejsca) - 1)]
```

#### Funkcja *losowa\_selekcia(miejsca)*:

- Ta funkcja wykonuje losowy wybór elementu z listy miejsca za pomocą funkcji random.randint.

```
def oblicz_wage(feromony, odleglosc, alfa, beta):
    return (feromony ** alfa) * ((1 / odleglosc) ** beta)
```

#### Funkcja *oblicz\_wage(feromony, odleglosc, alfa, beta)*:

- Oblicza wagę trasy na podstawie feromonów, odległości oraz parametrów alfa i beta.
- Wzór wagowy oparty jest na feromonach i odległości między punktami.

## Klasa main.py

```
def algorytm_mrowkowy(sciezka_pliku, kolor, pozycja, wyniki):
    plansza = Plansza(sciezka_pliku)
    najlepsze_mrowki = []

    for _ in tqdm(range(LICZBA_ITERACJI), colour=kolor, position=pozycja,
    leave=False):
        ostatnie_miejsce = len(plansza.miejsca) - 1
        mrowki = [Mrowka(randint(0, ostatnie_miejsce)) for _ in
range(ROZMIAR_POPULACJI)]
        for _ in range(ostatnie_miejsce):
            for mrowka in mrowki:
                mrowka.nastepny_krok(plansza, ALFA, BETA, CZYNNIK_LOSOWY)

        plansza.aktualizuj_feromony(CZYNNIK_PAROWANIA_FEROMONOW, mrowki)
        najlepsze_mrowki.append(min(mrowki, key=lambda x:
x.odleglosc_przebyta(plansza)))
    wyniki[pozycja] = [plansza, najlepsze_mrowki]
```

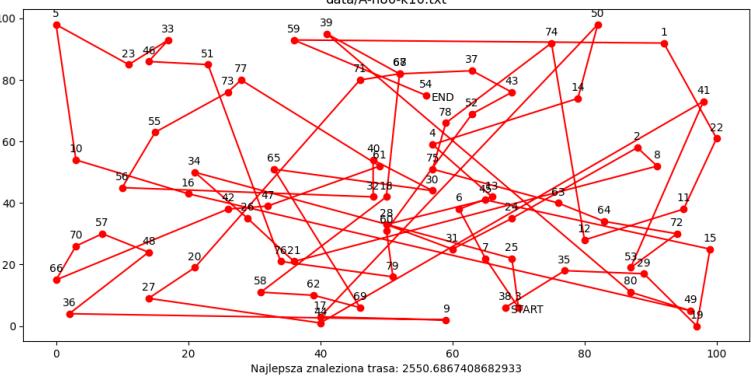
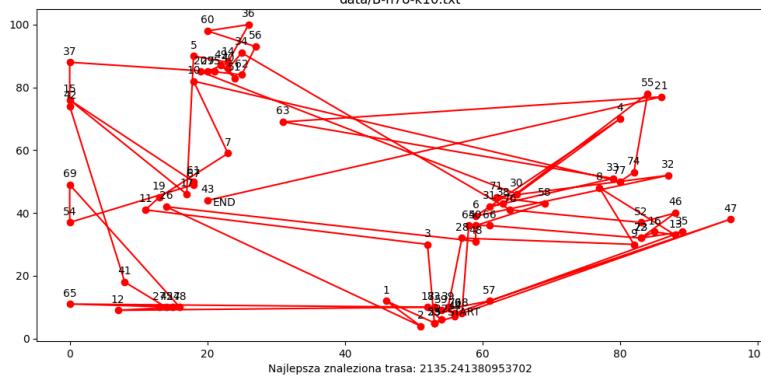
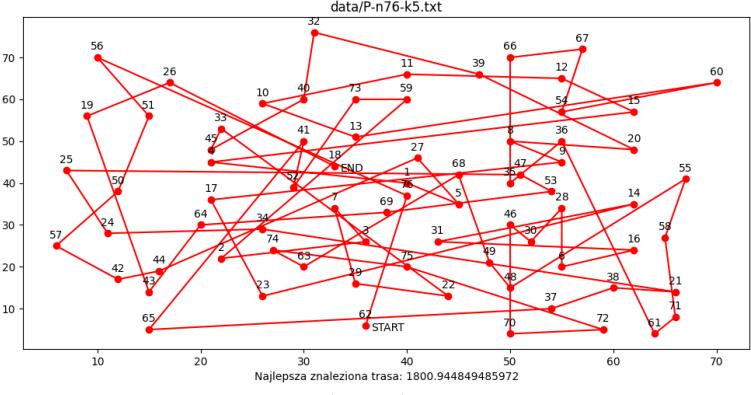
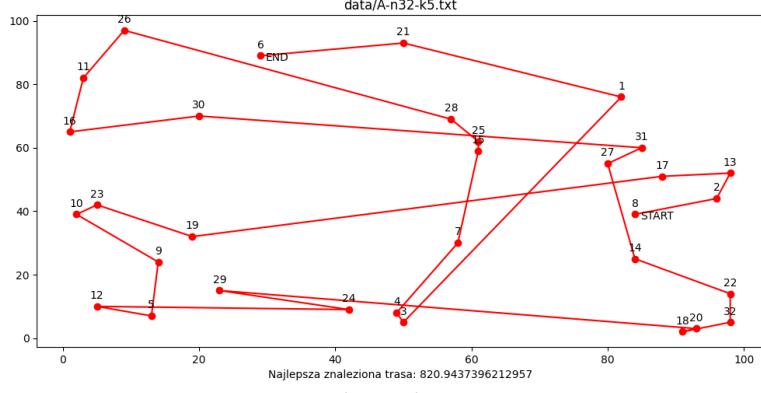
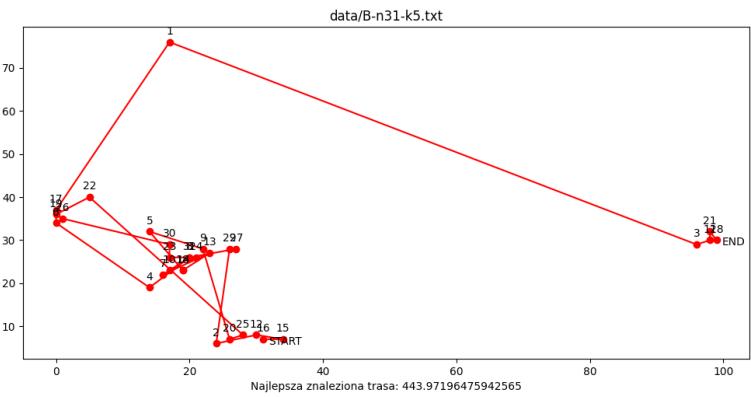
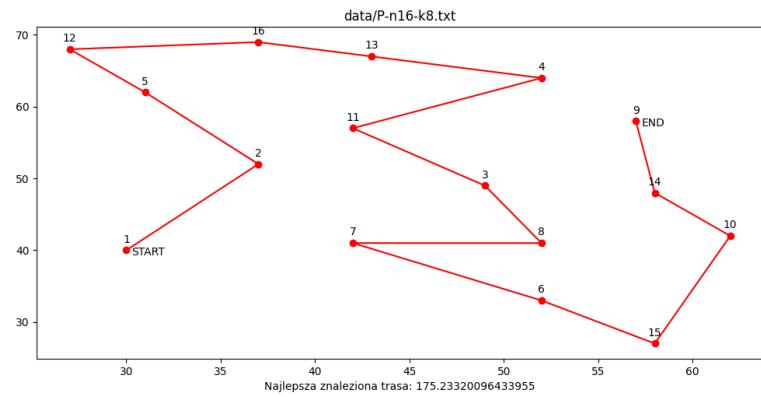
### Funkcja *algorytm\_mrowkowy(sciezka\_pliku, kolor, pozycja, wyniki)*:

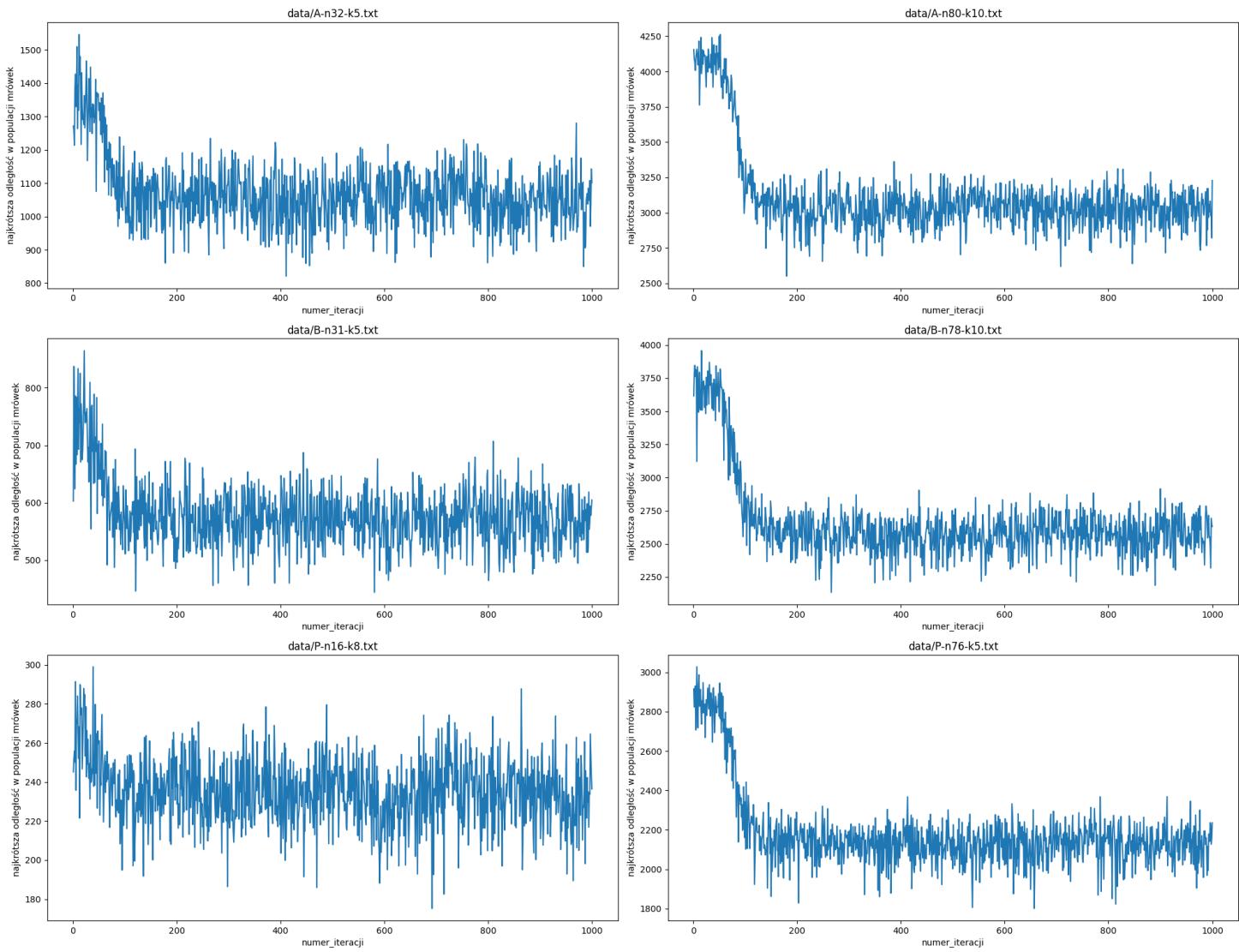
- Ta funkcja wykonuje algorytm mrówkowy dla konkretnej planszy.
- Tworzy populację mrówek, gdzie każda mrówka rozpoczyna trasę od losowego miejsca.
- W każdej iteracji, każda mrówka wykonuje kroki na podstawie reguł algorytmu mrówkowego.
- Po zakończeniu iteracji, aktualizuje feromony na planszy i zapisuje najlepszą mrówkę.

### 3. Wyniki i analiza

Określenie parametrów:

- Liczebność mrówek: 10
- Prawdopodobieństwo wyboru atrakcji: 0.3
- Alfa = 1
- Beta = 1
- Liczba iteracji: 1000
- Współczynnik parowania feromonów: 0.1



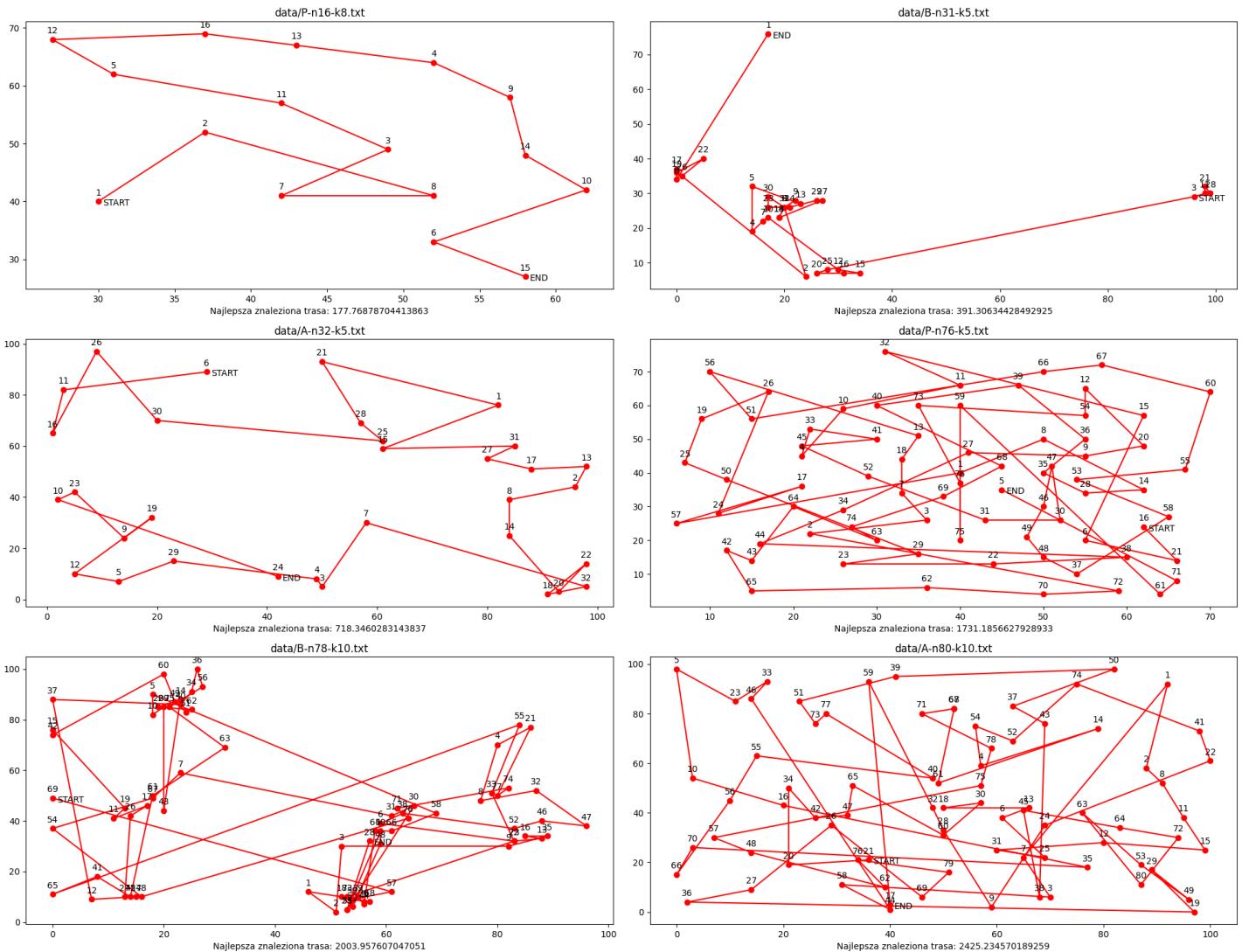


Powyższe wykresy pokazują, w jaki sposób zmieniała się najkrótsza wyznaczona trasa w poszczególnych iteracjach. W każdym przypadku nagle zmiana najlepszego rozwiązania występowała w około 150 iteracji. Nasz wniosek jest, więc taki, że do wyznaczenia najkrótszej trasy nie jest potrzebne ustawianie liczby iteracji na 1000 (wydłuża to znaczco czas działania programu).

## Badanie wpływu populacji mrówek

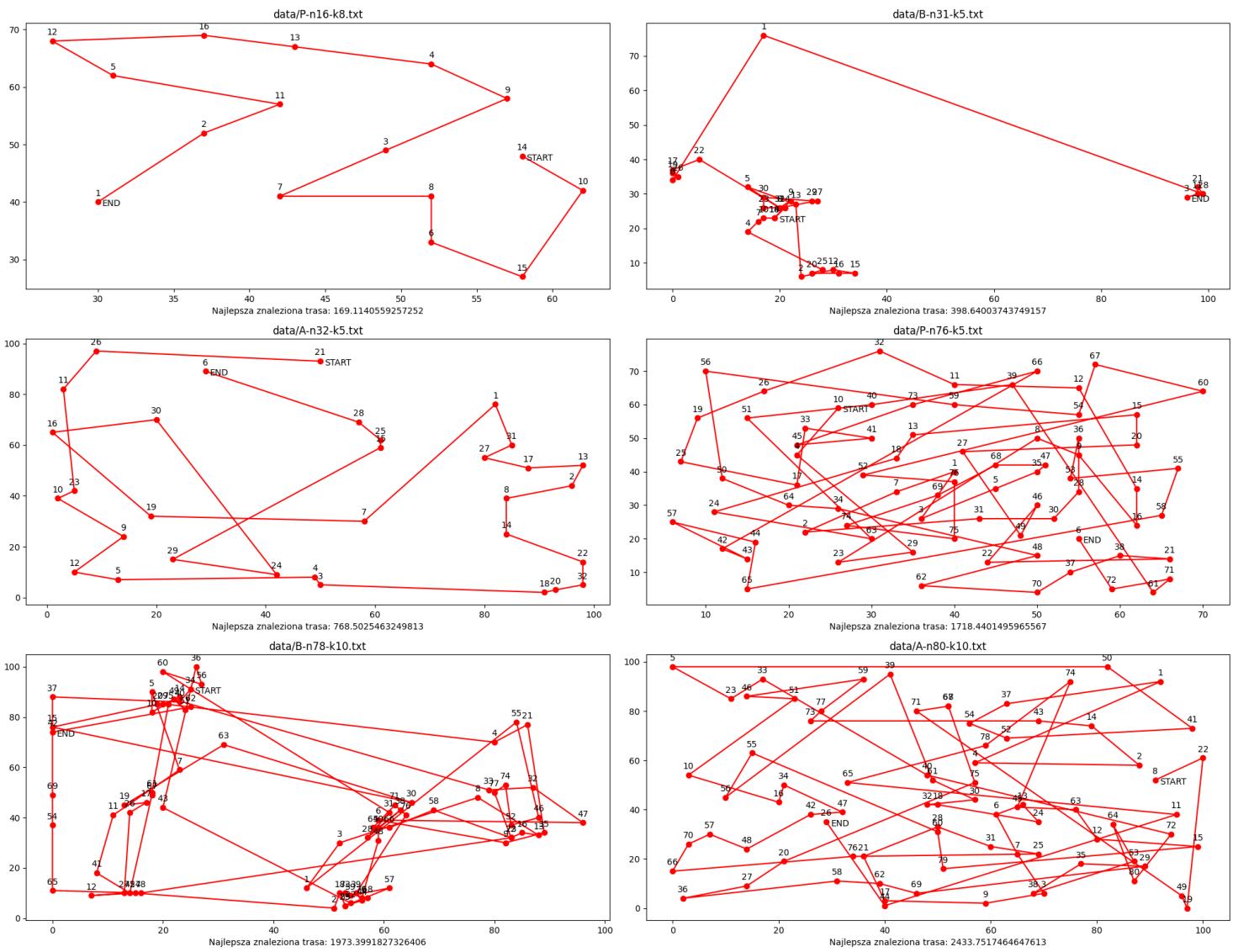
Określenie parametrów:

- Liczebność mrówek: 30
- Prawdopodobieństwo wyboru atrakcji: 0.3
- Alfa = 1
- Beta = 1
- Liczba iteracji: 1000
- Współczynnik parowania feromonów: 0.1



### Określenie parametrów:

- Liczebność mrówek: 50
- Prawdopodobieństwo wyboru atrakcji: 0.3
- Alfa = 1
- Beta = 1
- Liczba iteracji: 1000
- Współczynnik parowania feromonów: 0.1

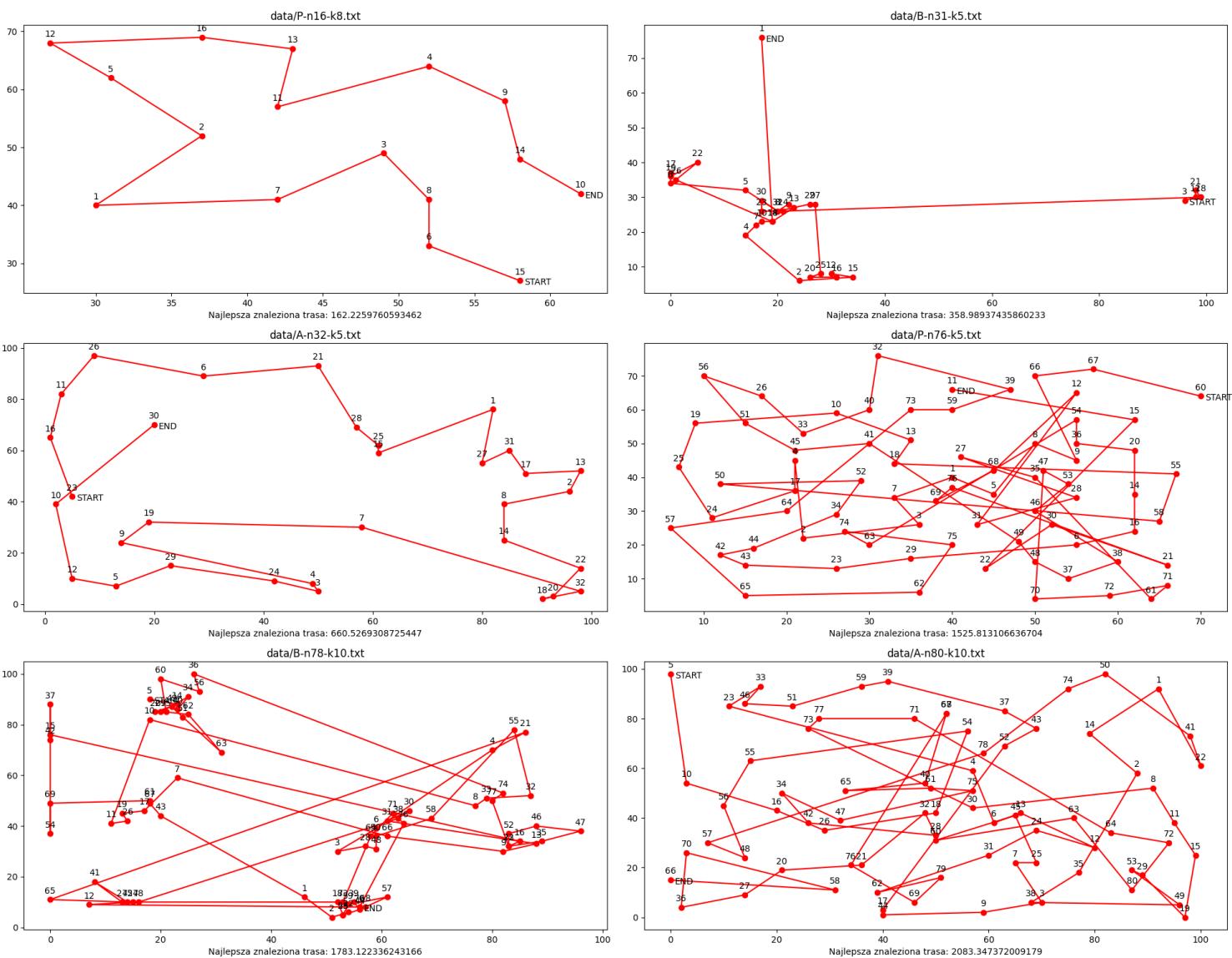


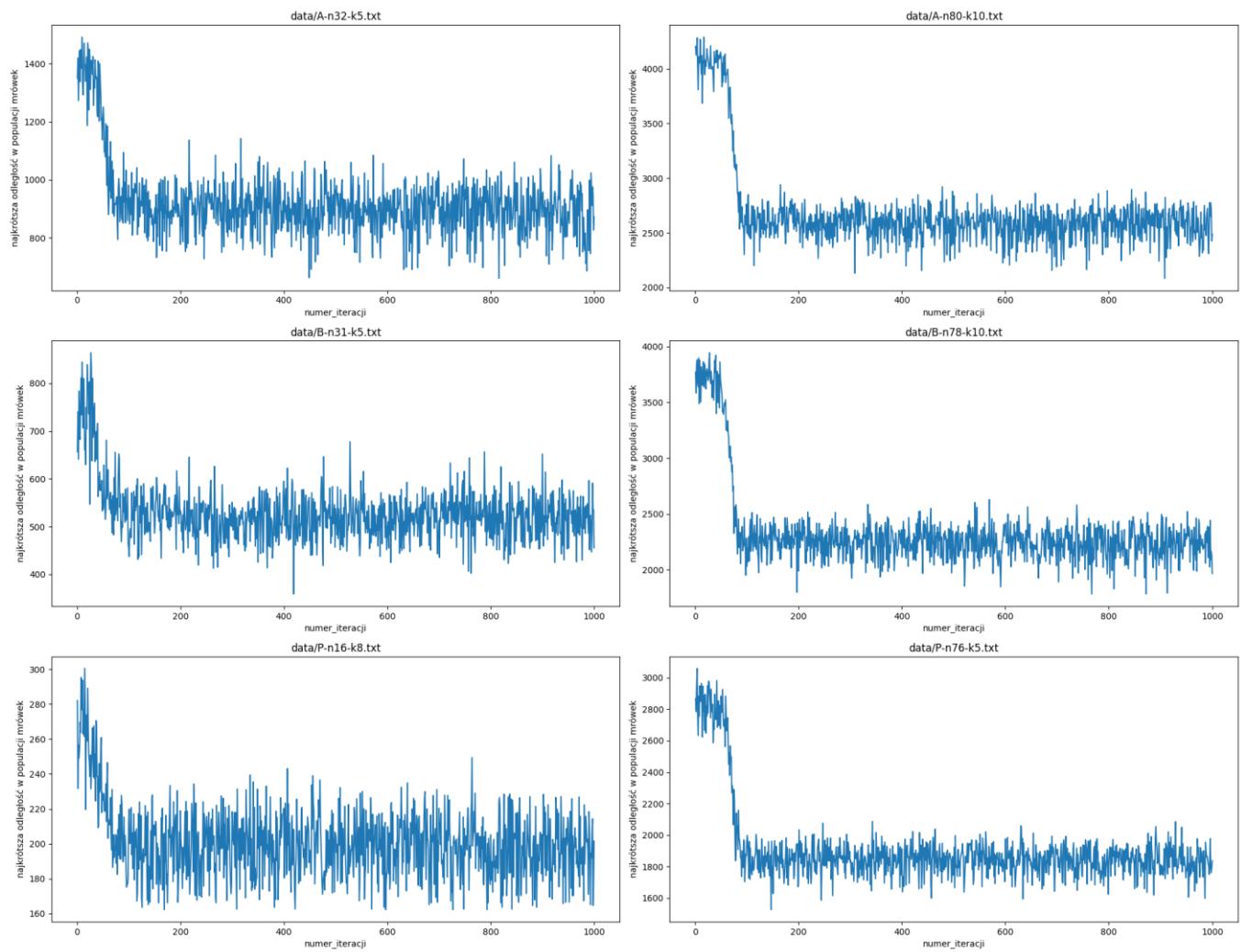
Zauważamy, że wraz ze wzrostem populacji mrówek poprawia się nasze rozwiązywanie w każdym z badanych mapach z atrakcjami. Stwierdzamy jednak, że wzrost jakości rozwiązania nie jest satysfakcyjujący w porównaniu do czasu działania programu (zwiększenie liczby znacząco zwiększa obliczenia).

## Badanie wpływu Alfa

Określenie parametrów:

- Liczebność mrówek: 10
- Prawdopodobieństwo wyboru atrakcji: 0.3
- Alfa = 2
- Beta = 1
- Liczba iteracji: 1000
- Współczynnik parowania feromonów: 0.1



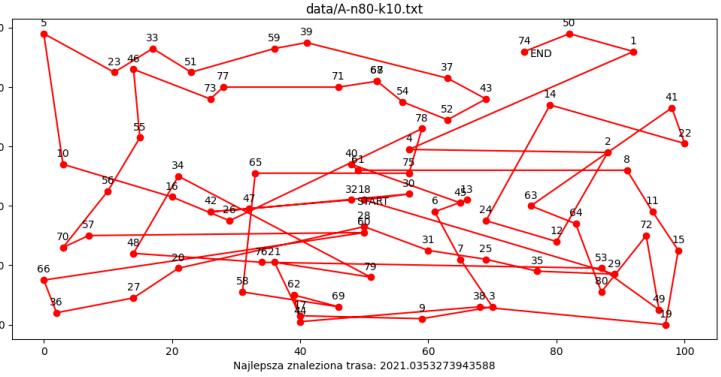
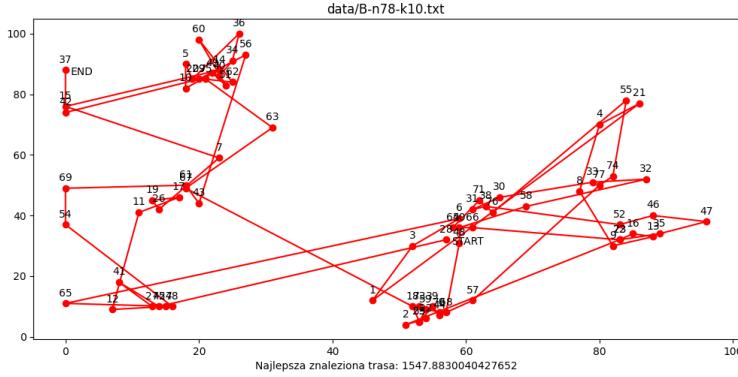
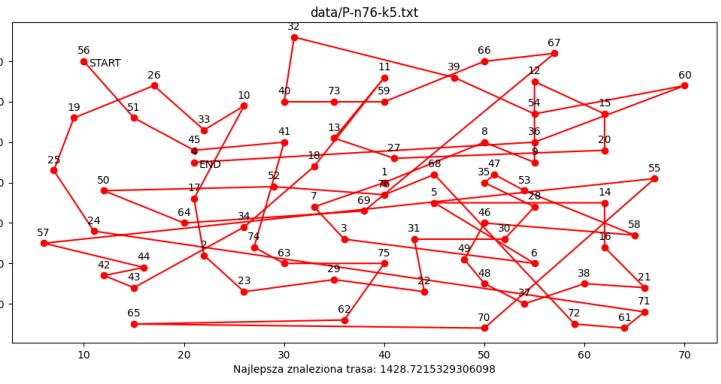
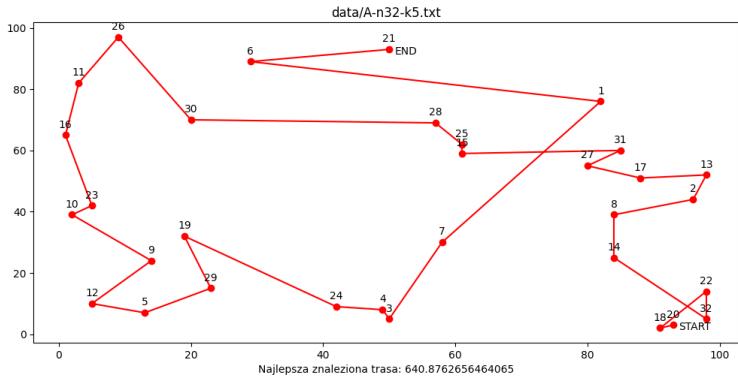
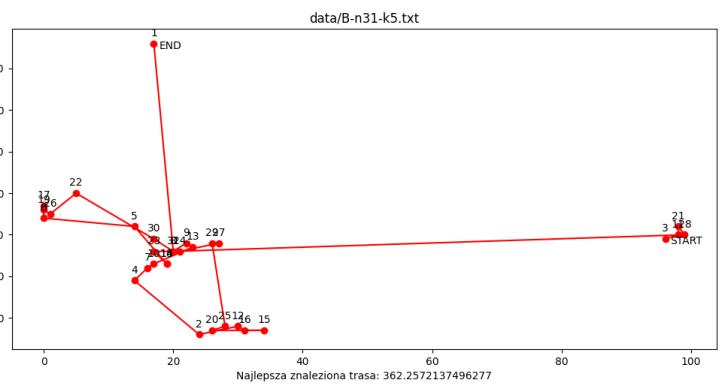
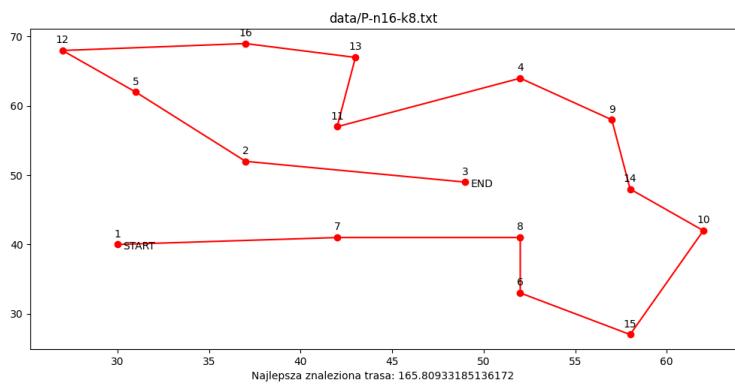


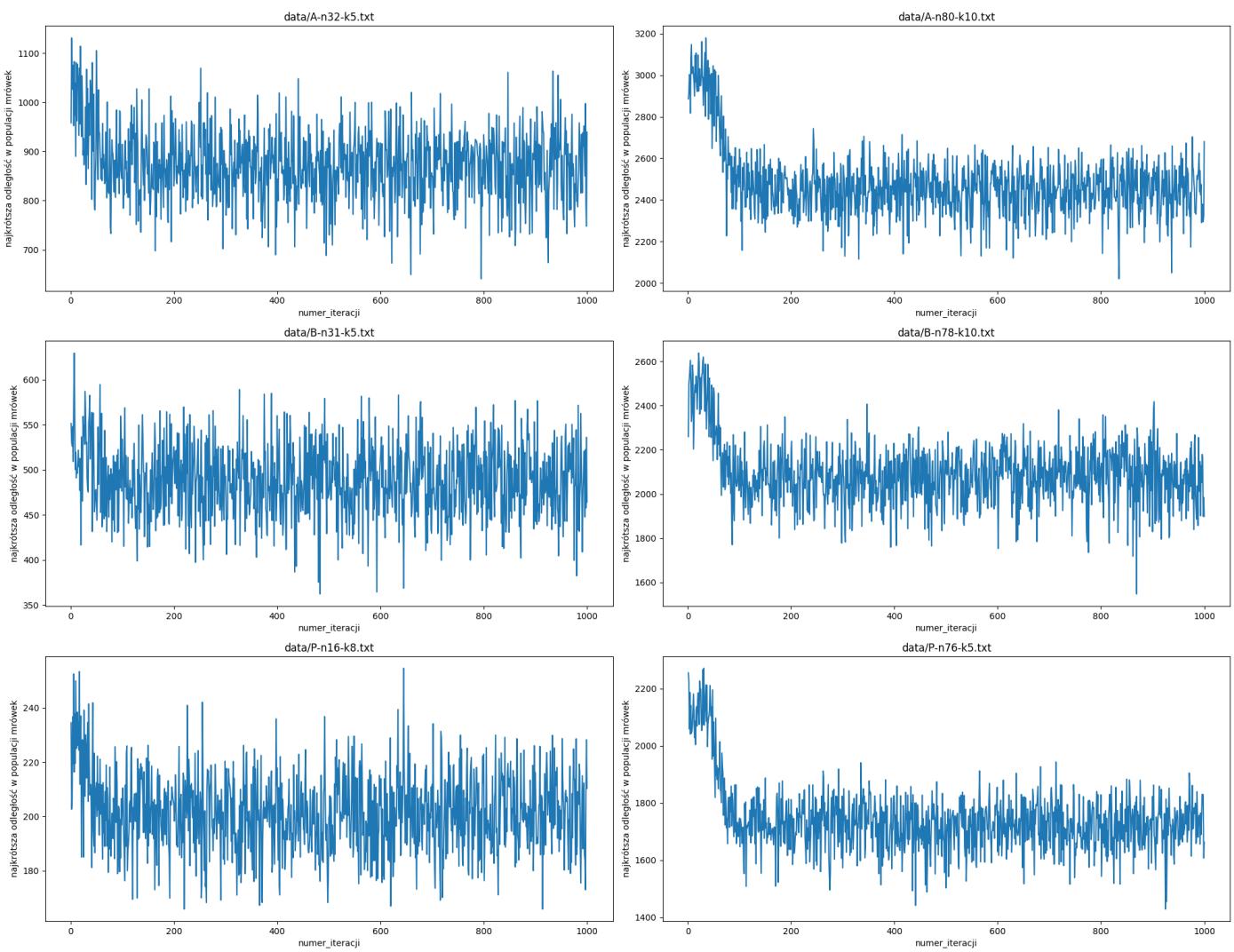
Zauważamy, że w przypadku zastosowania  $\text{Alfa} = 2$  (w stosunku do  $\text{Alfa} = 1$ ) najlepsze rozwiązanie poprawiło się w każdym z badanych przypadków. Dodatkowo zauważamy, że wartość zbliżoną do ostatecznego rozwiązania otrzymaliśmy po mniejszej liczbie iteracji (występuje lepsza szybkość osiągania najlepszych wyników).

## **Badanie wpływu Beta**

## Określenie parametrów:

- Liczebność mrówek: 10
  - Prawdopodobieństwo wyboru atrakcji: 0.3
  - Alfa = 1
  - Beta = 3
  - Liczba iteracji: 1000
  - Współczynnik parowania feromonów: 0.1



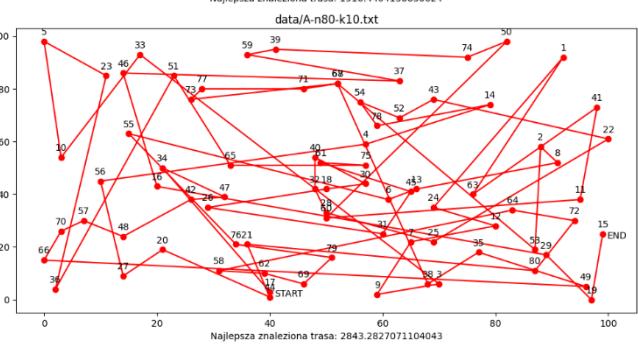
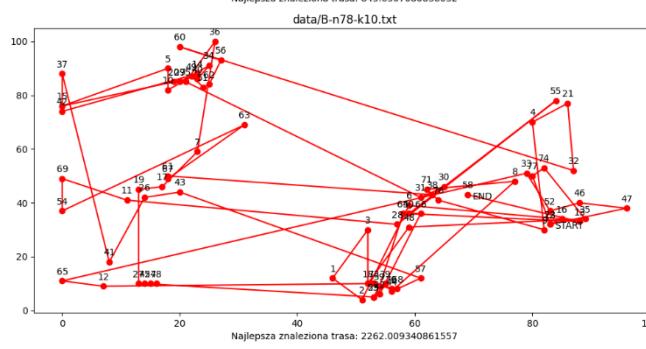
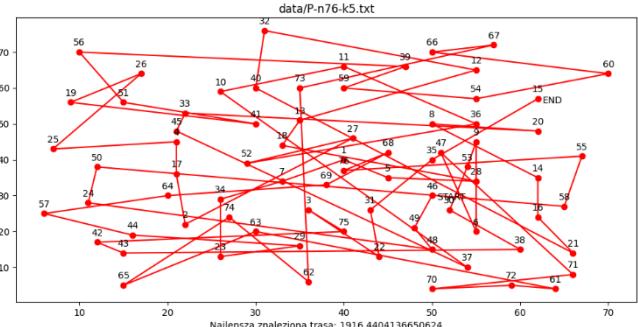
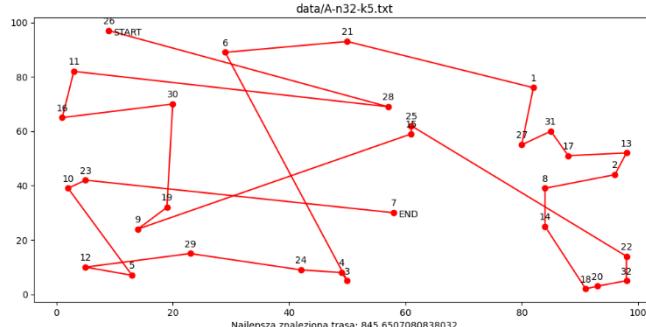
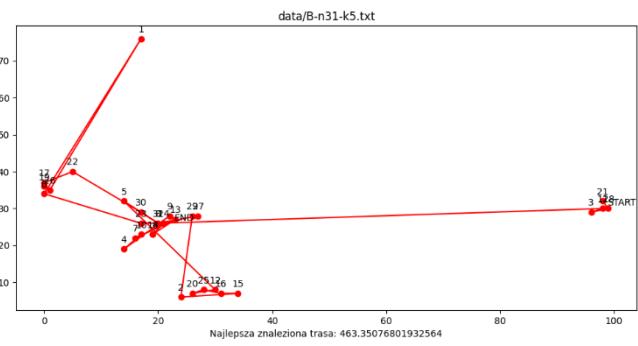
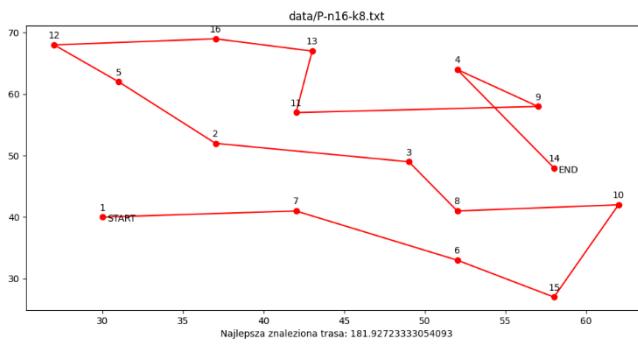


Zauważamy, że dla Beta = 3 (porównując do Beta = 1) najlepsze rozwiązanie uległo poprawie. Również jak w przypadku zwiększenia współczynnika Alfa, najkrótsza trasa była zbliżona do ostatecznego wyniku po mniejszej ilości iteracji (szybsza prędkość osiągnięcia wyników najlepszych).

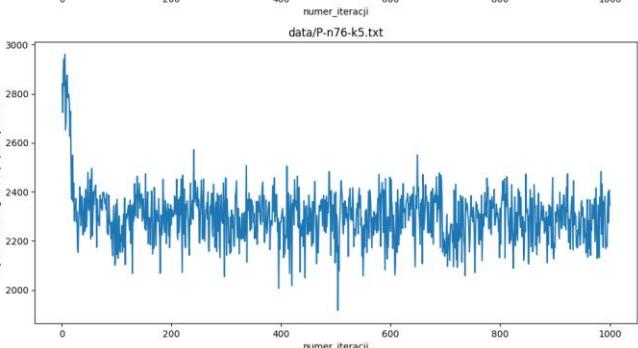
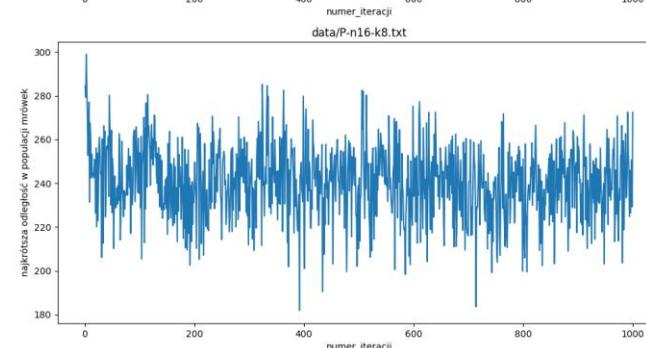
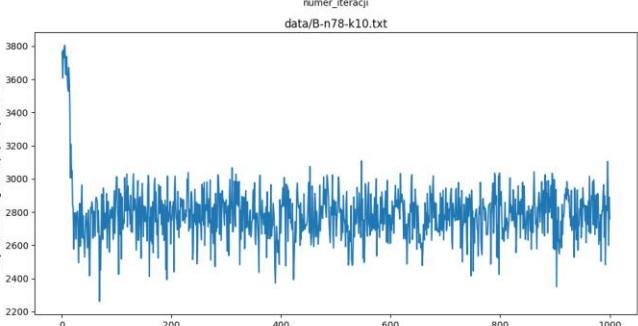
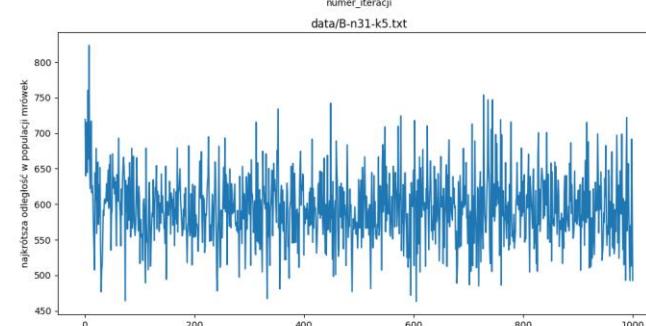
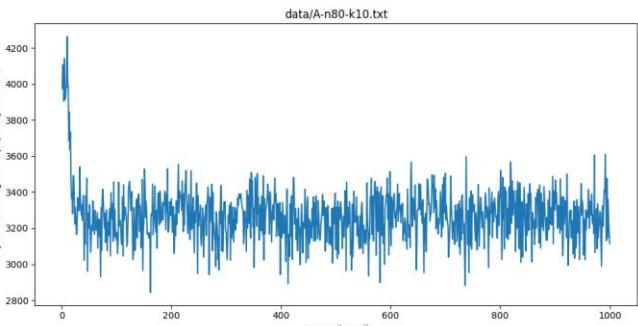
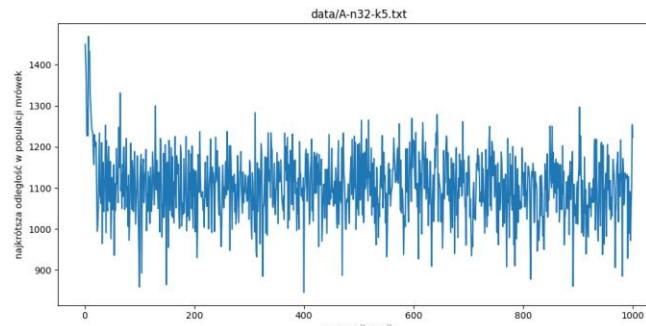
## Badanie wpływu współczynnika parowania feromonów

Określenie parametrów:

- Liczebność mrówek: 10
- Prawdopodobieństwo wyboru atrakcji: 0.3
- Alfa = 1
- Beta = 1
- Liczba iteracji: 1000
- Współczynnik parowania feromonów: 0.5



W W



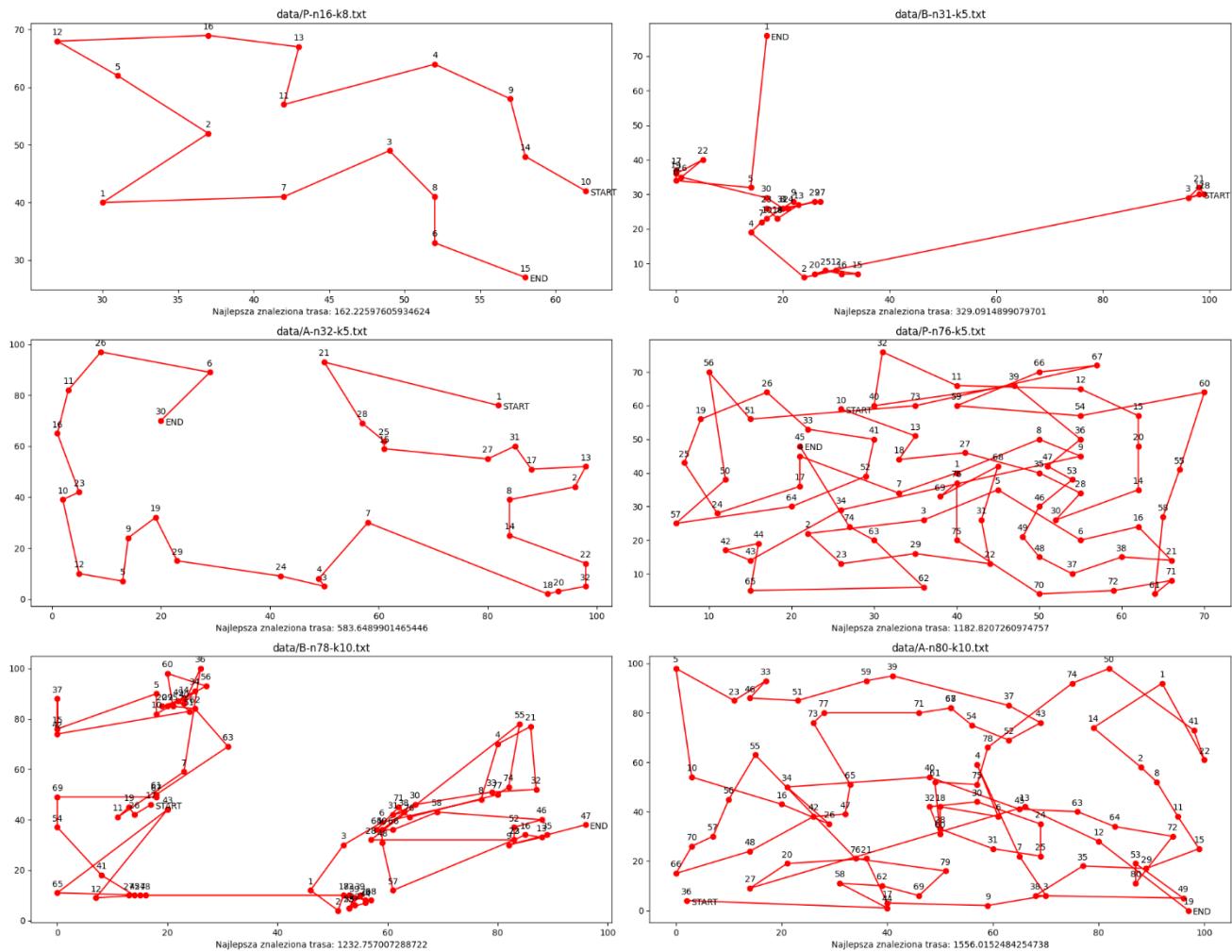
W przypadku zwiększenia współczynnika parowania feromonów nie zauważymy poprawy najlepszego wyniku końcowego. Widoczne jest jednak ponownie zwiększenie szybkości osiągnięcia najkrótszej trasy. Dla współczynnika parowania = 0.5 rozwiązanie zbliżone do optymalnego występowało już po około 20 iteracji.

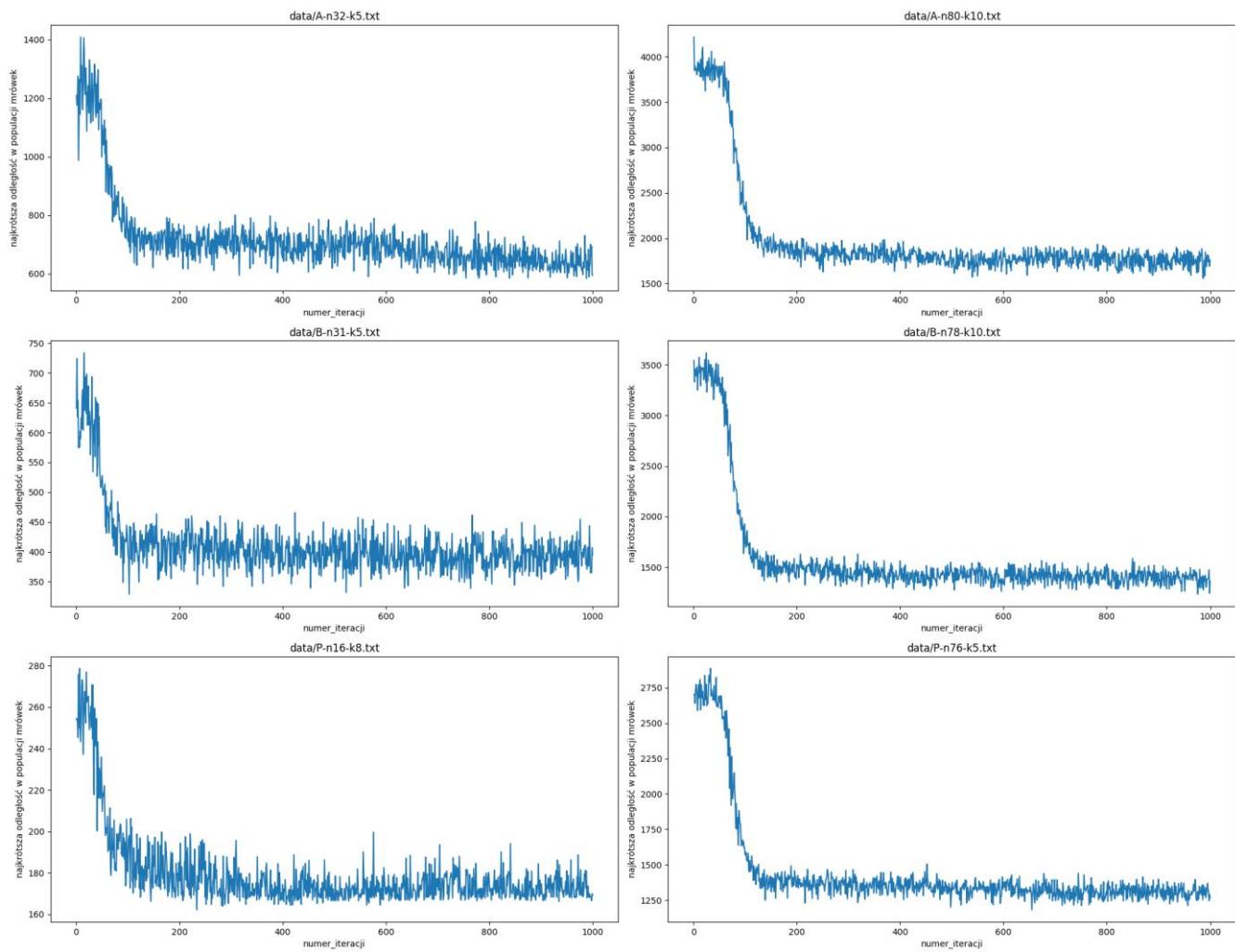
## Badanie wpływu współczynnika losowości

Stwierdziliśmy, że pomimo w poleceniu zadania nie jest wymagane przeprowadzanie tego badania, zdecydowaliśmy się na eksperyment związanego z tym współczynnikiem.

Określenie parametrów:

- Liczebność mrówek: 10
- Prawdopodobieństwo wyboru atrakcji: 0.01
- Alfa = 1
- Beta = 1
- Liczba iteracji: 1000
- Współczynnik parowania feromonów: 0.1





W przypadku zmiany prawdopodobieństwa wybrania przez mrówkę losowej atrakcji z 0.3 na 0.01 zauważamy bardzo znaczącą poprawę najlepszego wyniku (w niektórych przypadkach niemal dwukrotna poprawa wyniku). Do chodzimy, więc do wniosku, że jest to bardzo ważny współczynnik dla działania całego algorytmu. Prawdopodobieństwo wyboru losowej atrakcji powinno być zdecydowanie niższe, ponieważ przy takim samym czasie działania programu otrzymujemy znacząco lepsze rezultaty.