Raport

Przemysław Lis 229940

Wojciech Majchrzak 229947

1. Zasada działania programu

Program tworzy obiekt który pobiera stałe wartości takie jak liczebność mrówek, prawdopodobieństwo wyboru przez mrówkę losowej atrakcji, liczbę iteracji, współczynnik wyparowania oraz dane zaczytane z pliku txt. Następnie poprzez kolejne funkcję algorytm dokonuje jak najbardziej korzystnej trasy. W wyniku działania programu otrzymujemy graf reprezentujący trasę zaplanowanych kolejno dostaw, całkowitą długość trasy wyliczoną na podstawie sumy odległości pomiędzy konkretnymi punktami oraz listę kolejno odwiedzonych atrakcji.

2. Wybrane miejsca implementacji rozwiązania

Nasz program składa się z 3 głównych klas:

- **Algorithm** – klasa bazująca na algorytmie mrówkowym i wykorzystująca jej metody w celu doboru optymalnej drogi

```
def configure_ants(self):
    ants_number = round(self.multiplier)
    self.ant_colony = []
    for _ in range(0, ants_number):
        self.ant_colony.append(Ant(self.attractions_number,
self.attraction_distance, self.all_attraction))

def update_pheromone(self):
    for x in range(0, self.attractions_number):
        for y in range(0, self.attractions_number):
            self.pheromone_traces[x][y] = self.pheromone_traces[x][y] *
self.evaporation
            for ant in self.ant_colony:
                self.pheromone_traces[x][y] += 1 / ant.get_total_distance()

def get_best_ant(self):
    for ant in self.ant_colony:
        total_distance = ant.get_total_distance()
        if total_distance < self.best_ant.get_total_distance():
            self.best_ant = ant
```

Metody sterujące populacją mrówek, generowanie populacji, aktualizacja feromonów, wybór najlepszych mrówek z populacji

```
def solve(self):
    for i in range(0, self.iteration_number):
        self.configure_ants()
        for r in range(0, self.attractions_number - 1):
            for ant in self.ant_colony:
                if ant.is_max_capacity():

ant.visited_attractions.append(ant.all_attractions[0].index)
                    ant.capacity = 0
                # select attraction
                method = random.uniform(0, 1)
                if method <= self.probability_random_attraction:
                    ant.visit_random_attraction()
                else:
                    try:
                        index_to_visit =
ant.roulette_selection(self.pheromone_traces, self.alfa, self.beta)[0][0]
                    except IndexError:
                        self.make_chart()
                        return

                    while index_to_visit in ant.visited_attractions:
                        index_to_visit =
ant.roulette_selection(self.pheromone_traces, self.alfa, self.beta)[0][0]
                    ant.visit_attraction(index_to_visit)
                ant.current_distance = ant.get_total_distance()
                if r == 0:
                    self.best_ant = self.ant_colony[0]
        self.update_pheromone()
        self.get_best_ant()
    self.make_chart()
```

Metoda rozwiązująca problem rzeczywisty na podstawie algorytmu mrówkowego i metody selekcji jaką jest ruletka

```
def make_chart(self):
    # prepare data
    x_val = []
    y_val = []
    for attraction in self.best_ant.visited_attractions:
        x_val.append(self.all_attraction[attraction].x)
        y_val.append(self.all_attraction[attraction].y)

    plt.plot(x_val, y_val, '-o')
    plt.show()
```

Metoda rysująca na ekran wygenerowaną drogę

- **Ant** – klasa reprezentująca obiekt mrówki, oddzielona od Algorithm aby można było sterować całą populacją

```
def is_max_capacity(self) -> bool:
    if self.capacity > self.max_capacity:
        return True
    return False
```

Funkcja sprawdzająca czy została przekroczona dopuszczalna wartość pojemności pojazdu

```
def get_total_distance(self):
    total_distance = 0
    for a in range(1, len(self.visited_attractions)):
        total_distance += math.sqrt(pow(
            self.all_attractions[self.visited_attractions[a - 1]].x -
self.all_attractions[
                self.visited_attractions[a]].x, 2) + pow(
            self.all_attractions[self.visited_attractions[a - 1]].y -
self.all_attractions[
                self.visited_attractions[a]].y, 2))

    return total_distance
```

Funkcja ta służy do obliczania łącznej odległości pokonanaje przez daną mrówkę

```
def visit_attractions_probabilistically(self, pheromone_traces, alfa,
beta):
    current_attraction = self.visited_attractions[-1]
    self.__update_available_attractions()

    probability_sum = 0
    using_probability = []
    self.using_index = []
    self.using_probability = []
    for attraction in self.available_attractions:
        self.using_index.append(attraction.index)
        pheromone =
pow(pheromone_traces[current_attraction][attraction.index], alfa)
        heuristic = pow(1 /
self.attraction_distance[current_attraction][attraction.index], beta)
        probability = pheromone * heuristic
        using_probability.append(probability)
        probability_sum += probability
    self.using_probability = [probability / probability_sum for probability
in
                              using_probability]
```

W tej funkcji dokonywany jest wybór odwiedzenia kolejnej atrakcji. Mrówki wybierają lokalizację na podstawie dwóch czynników: intensywności feromonów na wszystkich ścieżkach oraz obliczonej heurystycznie wartości dla wszystkich dostępnych ścieżek. Bazuje ona na odległości ścieżek pomiędzy atrakcjami. Warto dodać że mrówki nie odwiedzają atrakcji które już zostały przez nie odwiedzone.

- **read_data** – odczyt danych z pliku w klasie

```
class Reader:
    @staticmethod
    def __edit(file_name: str):
        data = ''
        first_space = True
        with open(file_name, 'r') as file:
            for line in file:
                for char in line:
                    if char != ' ':
                        data += char
                        first_space = True
                    else:
                        if first_space:
                            data += ' '
                            first_space = False
        with open(file_name, 'w') as file:
            file.write(data)

    @staticmethod
    def read_data(file_name):
        Reader.__edit(file_name)
        data = pd.read_csv(file_name, sep=" ")
        try:
            data.drop('Unnamed: 0', axis=1, inplace=True)
        except KeyError:
            pass

        return data
```

Funkcje odpowiadające za odczyt i i aktualizowanie danych z pliku txt

3. Wyniki

Eksperymentalne wartości algorytmu mrówkowego:

```
ants = [10, 30, 50]
random_attraction = 0.3
alpha = [1, 2]
beta = [1, 3]
iteration = 10
evaporation = [0.1, 0.5]
max_capacity = 200
```

Na których otrzymaliśmy następujące wyniki

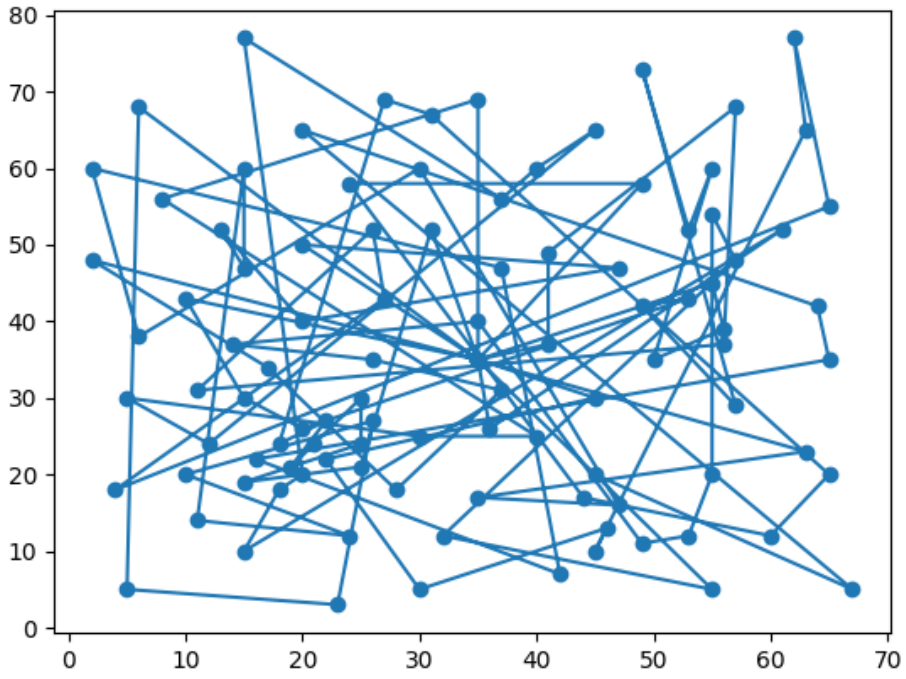
Liczba mrówek	Współczynnik atrakcji	Alpha	Beta	Liczba iteracji	Współczynnik parowania	Maks pojemność
10	0.3	1	1	10	0.1	200

Problem: R101.100

Liczba mrówek	Współczynnik atrakcji	Alpha	Beta	Liczba iteracji	Współczynnik parowania	Maks pojemność
10	0.3	1	1	10	0.1	200

Całkowity dystans: 2588.212330303898

Lista kolejno odwiedzonych punktów: [0, 3, 68, 12, 9, 33, 66, 54, 76, 77, 99, 5, 8, 0, 58, 78, 14, 100, 37, 64, 70, 30, 20, 61, 17, 13, 40, 47, 32, 0, 7, 50, 18, 48, 82, 19, 44, 42, 16, 94, 43, 38, 49, 0, 51, 62, 73, 39, 25, 90, 63, 93, 96, 98, 59, 6, 95, 91, 97, 31, 0, 28, 1, 71, 80, 84, 88, 52, 86, 34, 65, 35, 22, 74, 15, 92, 26, 0, 46, 60, 87, 53, 89, 83, 27, 75, 56, 4, 81, 79, 57, 23, 0, 55, 2, 72, 10, 45, 36, 69, 41, 85, 24, 29, 11, 67, 21]

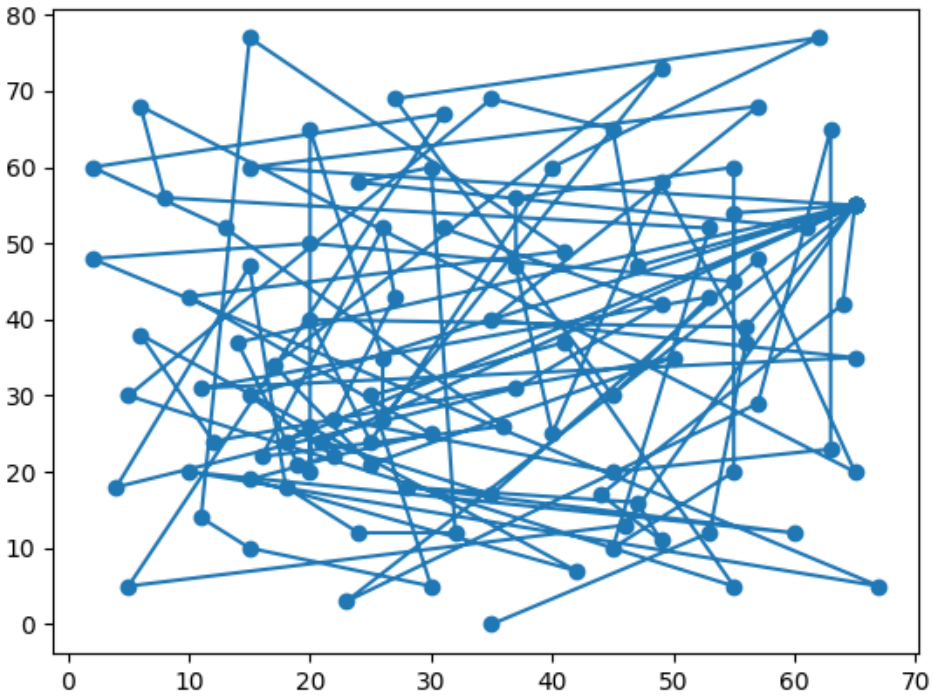


Wartość porównywana:
Całkowity dystans: 1637.7

Liczba mrówek	Współczynnik atrakcji	Alpha	Beta	Liczba iteracji	Współczynnik parowania	Maks pojemność
10	0.3	1	1	10	0.5	200

Całkowity dystans: 3161.8468422731735

Lista kolejno odwiedzonych punktów: [0, 56, 80, 51, 40, 69, 70, 9, 4, 22, 33, 47, 49, 25, 79, 2, 59, 34, 93, 99, 37, 11, 89, 94, 66, 60, 97, 12, 61, 45, 41, 91, 92, 5, 34, 83, 98, 13, 8, 1, 64, 44, 14, 15, 6, 28, 23, 17, 32, 20, 50, 34, 19, 71, 27, 24, 84, 77, 26, 63, 65, 30, 95, 18, 68, 34, 43, 54, 35, 55, 21, 96, 52, 88, 85, 58, 48, 36, 90, 38, 74, 34, 81, 3, 7, 46, 67, 16, 39, 87, 72, 75, 73, 29, 34, 78, 62, 10, 57, 42, 100, 82, 86, 53, 76, 31]

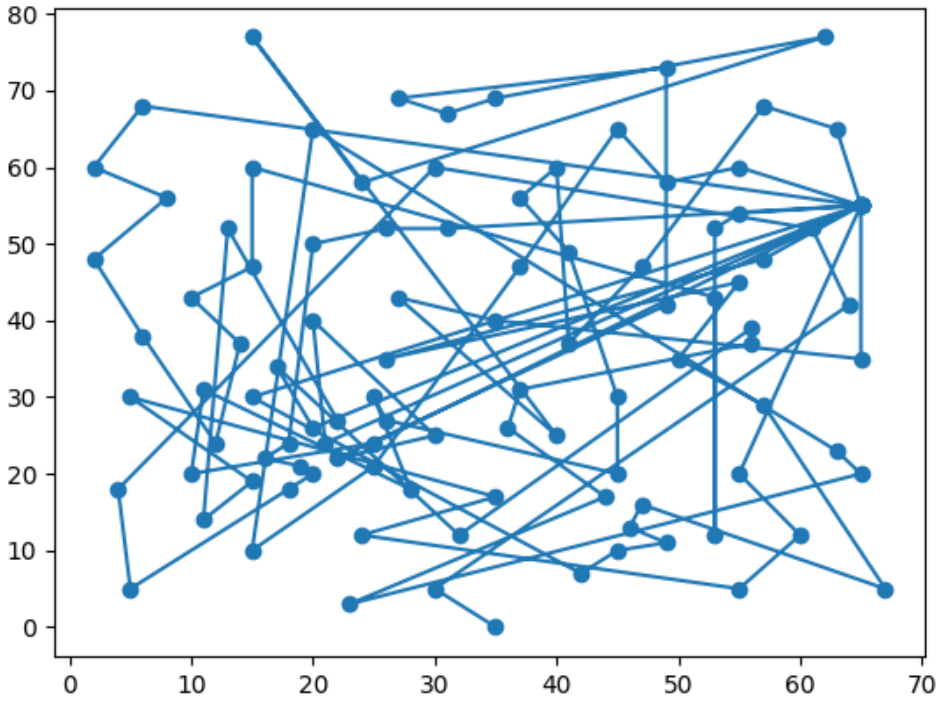


Wartość porównywana:
Całkowity dystans: 1637.7

Liczba mrówek	Współczynnik atrakcji	Alpha	Beta	Liczba iteracji	Współczynnik parowania	Maks pojemność
50	0.3	2	3	10	0.5	200

Całkowity dystans: 2124.446483654519

Lista kolejno odwiedzonych punktów: [0, 15, 29, 78, 10, 86, 38, 100, 37, 98, 85, 93, 7, 88, 31, 34, 35, 71, 50, 28, 30, 70, 1, 26, 21, 94, 6, 87, 84, 16, 95, 34, 81, 33, 56, 77, 19, 82, 8, 83, 61, 45, 46, 47, 36, 49, 34, 9, 51, 20, 69, 97, 14, 11, 55, 25, 43, 73, 58, 53, 80, 68, 57, 60, 99, 34, 79, 89, 3, 12, 54, 67, 72, 74, 75, 22, 41, 5, 34, 24, 27, 52, 40, 64, 62, 65, 32, 90, 63, 66, 76, 92, 13, 18, 59, 34, 4, 39, 23, 42, 2, 17, 91, 44, 48, 96]



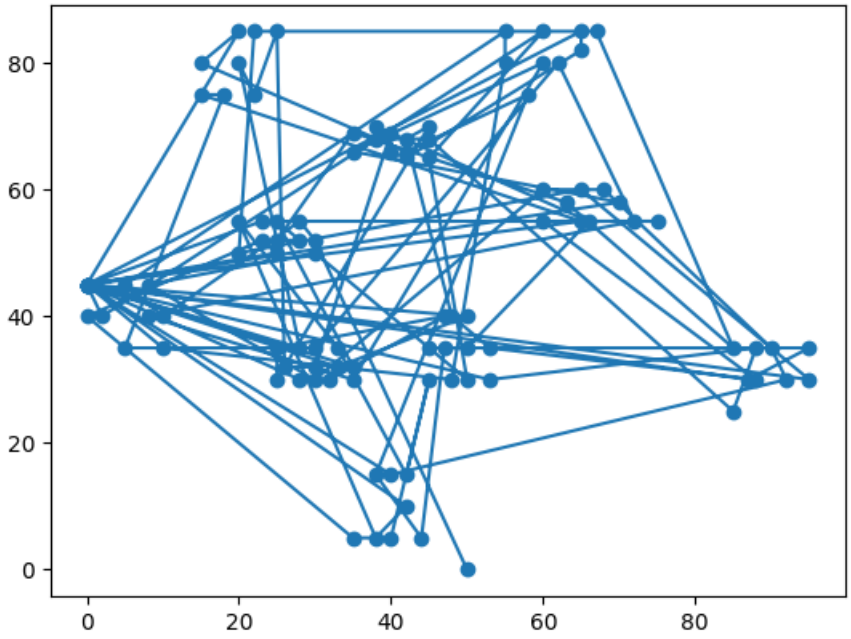
Wartość porównywana:
Całkowity dystans: 1637.7

Problem: C101.100

Liczba mrówek	Współczynnik atrakcji	Alpha	Beta	Liczba iteracji	Współczynnik parowania	Maks pojemność
10	0.3	1	1	10	0.1	200

Całkowity dystans: 3325.3602754635203

Lista kolejno odwiedzonych punktów: [0, 21, 23, 32, 36, 100, 99, 64, 31, 39, 63, 42, 39, 28, 82, 90, 77, 74, 65, 4, 2, 51, 52, 67, 44, 48, 39, 72, 78, 80, 1, 8, 9, 86, 62, 66, 53, 59, 98, 3, 19, 16, 39, 57, 73, 76, 85, 91, 45, 34, 20, 61, 75, 97, 37, 39, 49, 15, 13, 12, 50, 46, 6, 7, 5, 94, 93, 39, 24, 27, 38, 60, 56, 68, 55, 43, 30, 58, 54, 39, 41, 26, 22, 25, 11, 89, 81, 92, 14, 29, 88, 70, 39, 87, 18, 17, 35, 47, 95, 83, 33, 40, 96, 10, 84, 39, 79, 71, 69]

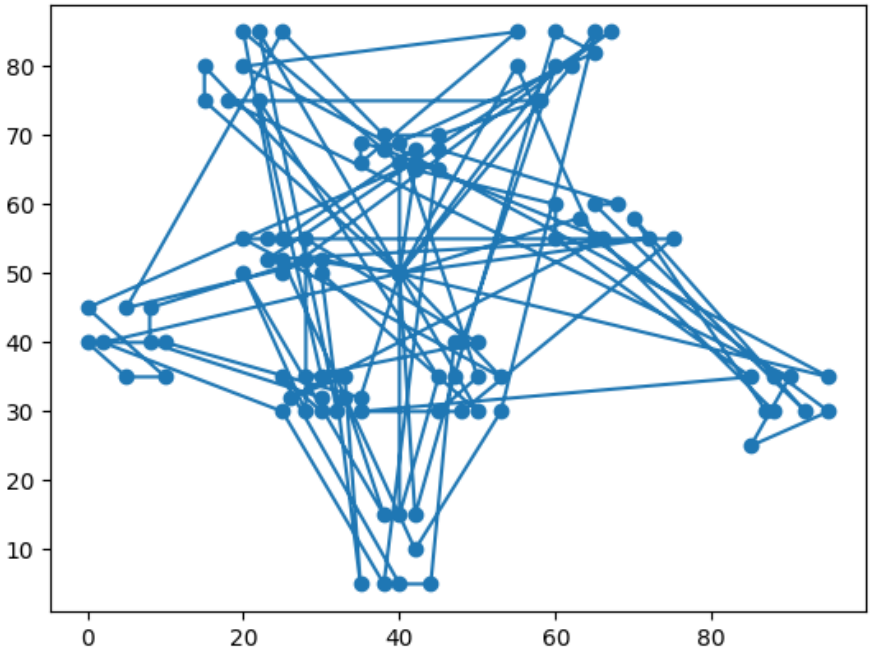


Wartość porównywana:
Całkowity dystans: 827.3

Liczba mrówek	Współczynnik atrakcji	Alpha	Beta	Liczba iteracji	Współczynnik parowania	Maks pojemność
10	0.3	1	1	10	0.5	200

Całkowity dystans: 2895.86160234953

Lista kolejno odwiedzonych punktów: [0, 95, 93, 72, 54, 25, 24, 92, 27, 83, 0, 5, 91, 90, 81, 40, 64, 82, 28, 30, 74, 19, 18, 0, 14, 44, 42, 59, 29, 48, 23, 63, 49, 47, 13, 69, 57, 0, 71, 75, 15, 100, 10, 11, 8, 86, 50, 43, 60, 20, 61, 0, 22, 34, 33, 46, 45, 52, 56, 53, 67, 65, 96, 0, 41, 32, 38, 35, 31, 39, 3, 4, 26, 16, 7, 0, 37, 51, 58, 1, 85, 88, 73, 78, 84, 79, 77, 89, 0, 99, 87, 70, 80, 76, 17, 98, 2, 9, 12, 36, 21, 0, 55, 66, 97, 94, 6, 62, 68]

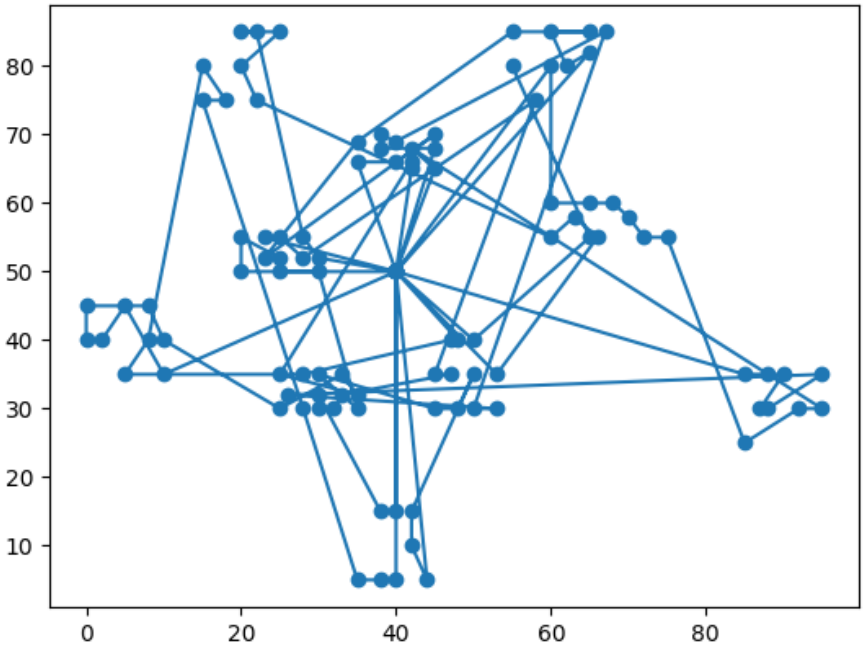


Wartość porównywana:
Całkowity dystans: 827.3

Liczba mrówek	Współczynnik atrakcji	Alpha	Beta	Liczba iteracji	Współczynnik parowania	Maks pojemność
50	0.3	2	3	10	0.1	200

Całkowity dystans: 1624.1781053163127

Lista kolejno odwiedzonych punktów: [0, 28, 26, 22, 98, 69, 66, 42, 44, 45, 46, 59, 57, 0, 1, 8, 9, 6, 92, 61, 72, 50, 71, 77, 79, 76, 78, 81, 0, 21, 23, 14, 16, 12, 15, 13, 90, 89, 86, 74, 0, 65, 67, 49, 35, 33, 19, 17, 18, 48, 60, 58, 56, 0, 75, 4, 70, 73, 80, 82, 83, 84, 85, 88, 91, 96, 0, 5, 3, 52, 41, 43, 51, 32, 34, 39, 38, 37, 36, 31, 0, 94, 95, 97, 93, 100, 11, 27, 2, 7, 10, 0, 24, 25, 30, 29, 20, 40, 47, 68, 64, 62, 55, 54, 53, 0, 63, 87, 99]



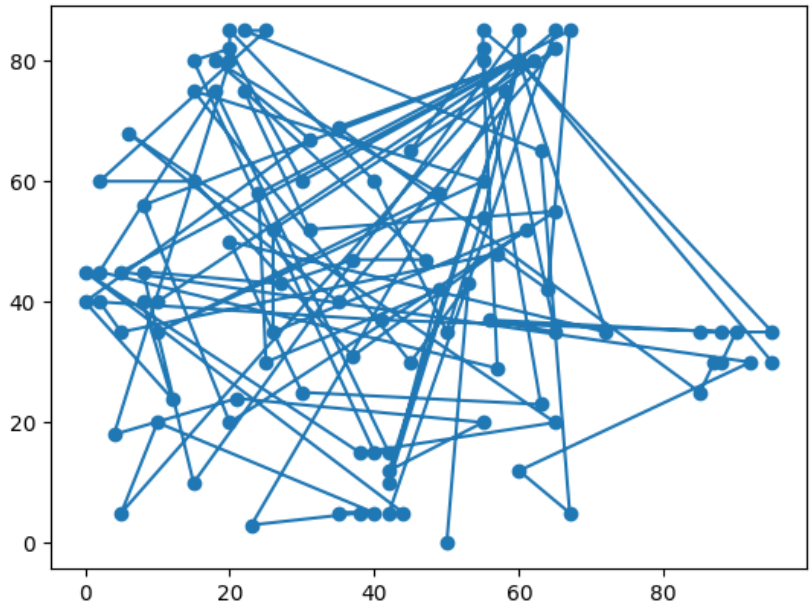
Wartość porównywana:
Całkowity dystans: 827.3

Problem: RC101.100

Liczba mrówek	Współczynnik atrakcji	Alpha	Beta	Liczba iteracji	Współczynnik parowania	Maks pojemność
10	0.3	1	1	10	0.1	200

Całkowity dystans: 3697.010003795571

Lista kolejno odwiedzonych punktów: [0, 92, 64, 7, 54, 9, 81, 46, 100, 16, 87, 11, 10, 39, 40, 58, 12, 82, 90, 80, 2, 69, 71, 74, 78, 39, 36, 48, 18, 17, 24, 63, 79, 65, 37, 61, 44, 43, 39, 70, 33, 32, 30, 29, 66, 15, 13, 94, 67, 35, 88, 52, 91, 19, 39, 50, 53, 57, 85, 42, 84, 47, 6, 20, 22, 8, 45, 55, 39, 38, 98, 99, 93, 83, 60, 73, 1, 3, 72, 89, 76, 28, 14, 39, 75, 59, 21, 25, 23, 77, 41, 62, 95, 31, 34, 27, 39, 49, 51, 86, 97, 4, 5, 68, 56, 96, 39, 26]

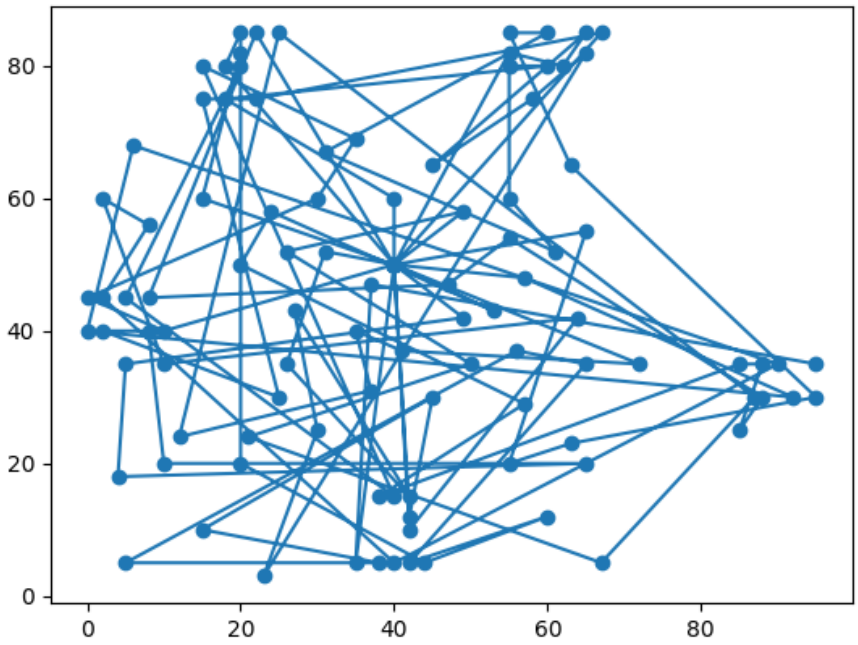


Wartość porównywana:
Całkowity dystans: 1619.8

Liczba mrówek	Współczynnik atrakcji	Alpha	Beta	Liczba iteracji	Współczynnik parowania	Maks pojemność
10	0.3	1	1	10	0.5	200

Całkowity dystans: 3308.618527789184

Lista kolejno odwiedzonych punktów: [0, 92, 88, 53, 84, 24, 34, 31, 33, 30, 96, 80, 12, 3, 0, 44, 38, 36, 61, 41, 37, 77, 57, 82, 20, 99, 69, 0, 81, 98, 56, 86, 89, 32, 1, 87, 83, 65, 66, 50, 0, 10, 17, 55, 70, 8, 22, 14, 4, 45, 60, 0, 94, 28, 16, 79, 29, 21, 47, 78, 73, 9, 67, 19, 0, 68, 6, 39, 42, 54, 93, 100, 40, 43, 72, 26, 85, 51, 71, 0, 46, 5, 74, 18, 76, 48, 62, 95, 75, 23, 58, 64, 49, 0, 91, 13, 97, 63, 59, 11, 15, 52, 7, 2, 35, 0, 25, 90, 27]

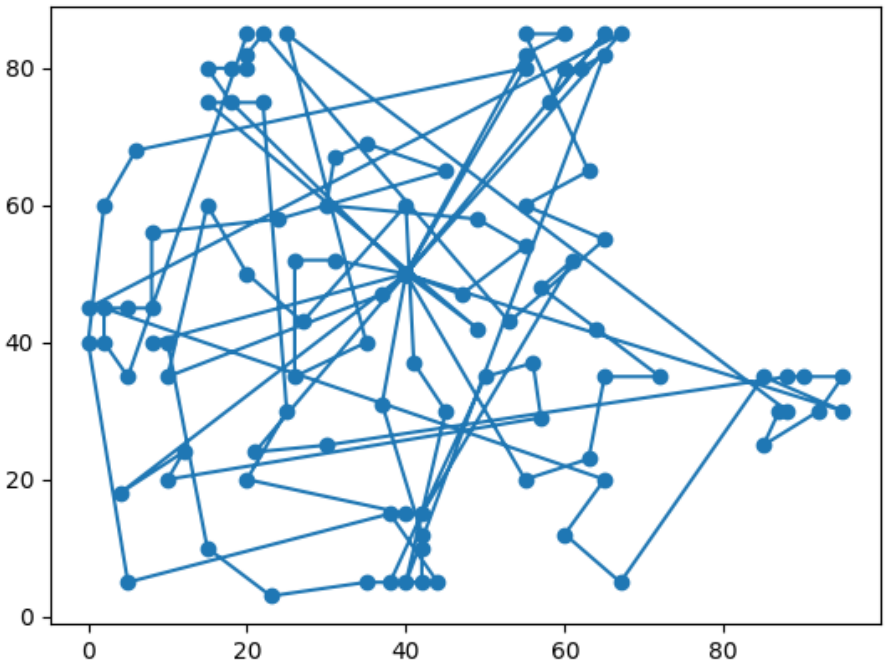


Wartość porównywana:
Całkowity dystans: 1619.8

Liczba mrówek	Współczynnik atrakcji	Alpha	Beta	Liczba iteracji	Współczynnik parowania	Maks pojemność
50	0.3	2	3	10	0.5	200

Całkowity dystans: 2086.1384112082656

Lista kolejno odwiedzonych punktów: [0, 69, 98, 99, 65, 1, 30, 32, 33, 28, 27, 29, 31, 57, 86, 0, 44, 40, 43, 72, 54, 71, 92, 3, 45, 4, 46, 8, 6, 0, 90, 9, 60, 53, 82, 68, 66, 64, 21, 37, 38, 39, 41, 36, 0, 91, 7, 2, 52, 74, 22, 20, 93, 94, 67, 50, 62, 85, 51, 0, 80, 96, 81, 55, 100, 70, 61, 88, 78, 12, 14, 17, 35, 0, 83, 49, 19, 48, 23, 56, 95, 84, 59, 87, 97, 0, 42, 79, 73, 16, 75, 24, 18, 25, 77, 58, 10, 11, 0, 26, 34, 89, 76, 63, 47, 15, 13, 5]



Wartość porównywana:
Całkowity dystans: 1619.8

4. Wnioski

- Nasze rozwiązania są mniej optymalne od sugerowanych na stronie, może być to spowodowane wyborem danej metody selekcji lub wykorzystania algorytmu mrówkowego
- Po zastosowaniu dodatkowej heurystyki moglibyśmy zwiększyć efektywność pokonywanej drogi
- Zwiększenie liczby mrówek, współczynników alpha, beta oraz parowania wydłuża pracę algorytmu natomiast otrzymane wyniki są bardziej optymalne i zbliżone do wartości porównywalnych
- Inkrementacja współczynnika parowania nie miała tak znaczącego wpływu na lepszy wynik niż zwiększenie wartości takich jak startowa liczba mrówek, czy współczynniki alpha i beta