**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, wagę feromonów oraz heurystyki, liczbę iteracji, współczynnik wyparowania oraz współrzędne atrakcji zaczytane z pliku txt. Następnie poprzez kolejne funkcję algorytm dokonuje jak najbardziej korzystnej trasy. W wyniku działania programu otrzymujemy graf reprezentujący nam pokonaną trasę, całkowitą długość trasy wyliczoną na podstawie sumy odległości pomiędzy konkretnymi punktami oraz listę kolejno odwiedzonych atrakcji.

1. **Wybrane miejsca implementacji zadania**

Nasz cały program składa się z 3 następujących klas:

* **Attraction**
  + Klasa odpowiedzialna za wczytywanie współrzędnych kolejnych atrakcji

class Attraction:  
 def \_\_init\_\_(self, index: int, x: int, y: int):  
 self.index = index - 1  
 self.x = x  
 self.y = y

* **Ant**
  + Klasa reprezentująca obiekt mrówki aby łatwo można było sterować i manipulować całą populacją
* 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.

* **Algorithm**
  + Klasa reprezentująca wszystkie metody odpowiedzialne na logikę aplikacji i sposób jej działania

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))

Metoda ta odpowiedzialna jest za generowanie całej populacji mrówek na podstawie podanego mnożnika liczby mrówek

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()

Funkcja modyfikująca ślady feromonowe w celu wpłynięcia na aktywność eksploracji mrówek. Jeśli więcej mrówek przeszło daną ścieżką to znajdzie się na niej więcej feromonów.

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

Funkcja get\_best\_ant zwraca mrówkę cechującą się pokonaniem trasy w jak najkrótszy sposób z całej 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:  
 # select attraction  
 method = random.uniform(0, 1)  
 if method <= self.probability\_random\_attraction:  
 ant.visit\_random\_attraction()  
 else:  
 index\_to\_visit = ant.roulette\_selection(self.pheromone\_traces, self.alfa, self.beta)[0][0]  
 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()

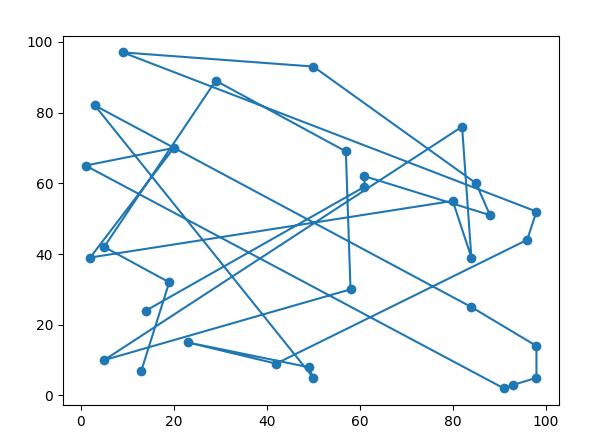
Funkcja sterująca całą kolonią mrówek rozwiązaniem problemu jak i odpowiedzialna za zakończenie pracy algorytmu i zwrócenie wyniku z grafem.

1. **Wyniki**

Wyniki otrzymane na podstawie przyjętych następujących danych:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Liczebność mrówek** | **Prawdopodobieństwo wyboru losowej atrakcji** | **Waga feromonów** | **Waga heurystyki** | **Liczba iteracji** | **Współczynnik wyparowania** |
| 10 | 0.3 | 1 | 1 | 1000 | 0.1 |

Plik: A-n32-k5.txt

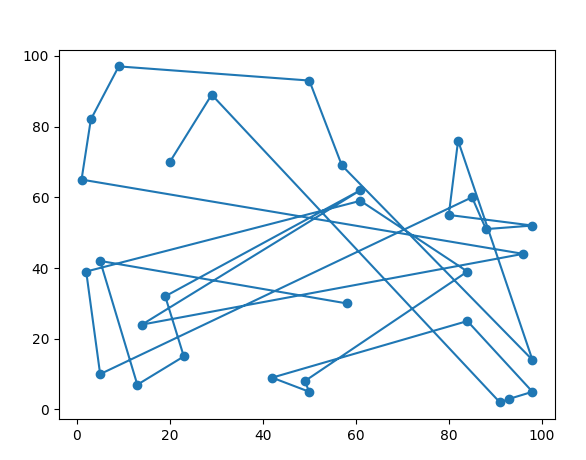


**Całkowity dystans:** 1261.2582394122157

**Lista kolejno odwiedzonych atrakcji:** [8, 14, 24, 16, 30, 20, 25, 12, 1, 23, 28, 3, 2, 10, 13, 21, 31, 19, 17, 15, 29, 9, 26, 7, 0, 11, 6, 27, 5, 22, 18, 4]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Liczebność mrówek** | **Prawdopodobieństwo wyboru losowej atrakcji** | **Waga feromonów** | **Waga heurystyki** | **Liczba iteracji** | **Współczynnik wyparowania** |
| 30 | 0.3 | 1 | 1 | 1000 | 0.1 |

Plik: A-n32-k5.txt

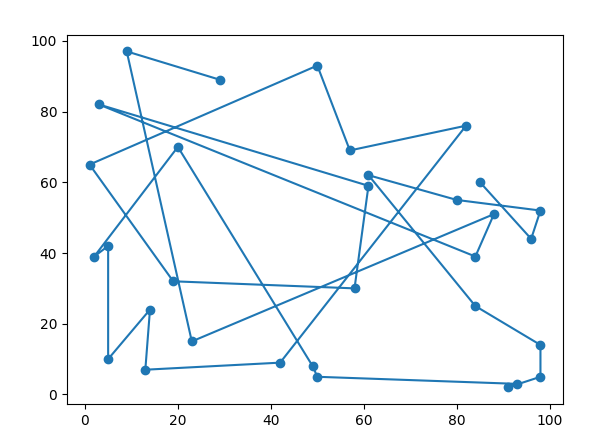


**Całkowity dystans:** 1185.2119399832254

**Lista kolejno odwiedzonych atrakcji:** [29, 5, 17, 19, 31, 13, 23, 2, 3, 7, 14, 9, 11, 30, 16, 12, 26, 0, 21, 27, 20, 25, 10, 15, 1, 8, 24, 18, 28, 4, 22, 6]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Liczebność mrówek** | **Prawdopodobieństwo wyboru losowej atrakcji** | **Waga feromonów** | **Waga heurystyki** | **Liczba iteracji** | **Współczynnik wyparowania** |
| 50 | 0.3 | 1 | 1 | 1000 | 0.1 |

Plik: A-n32-k5.txt

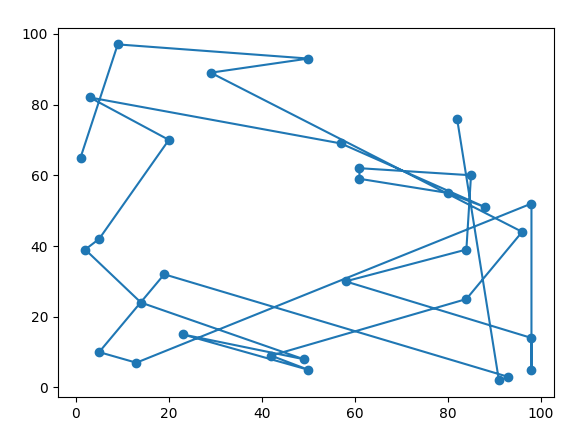


**Całkowity dystans:** 1032.8737288425018

**Lista kolejno odwiedzonych atrakcji:** [30, 1, 12, 26, 24, 13, 21, 31, 17, 19, 2, 3, 29, 9, 22, 11, 8, 4, 23, 0, 27, 20, 15, 18, 6, 14, 10, 7, 16, 28, 25, 5]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Liczebność mrówek** | **Prawdopodobieństwo wyboru losowej atrakcji** | **Waga feromonów** | **Waga heurystyki** | **Liczba iteracji** | **Współczynnik wyparowania** |
| 10 | 0.3 | 2 | 1 | 1000 | 0.1 |

Plik: A-n32-k5.txt

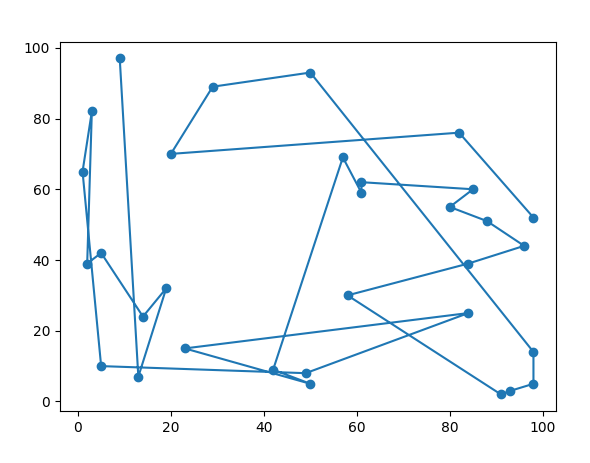


**Całkowity dystans:** 1004.3656944616843

**Lista kolejno odwiedzonych atrakcji:** [0, 17, 19, 18, 11, 4, 12, 31, 21, 6, 7, 30, 24, 14, 26, 16, 27, 10, 29, 22, 9, 8, 3, 28, 2, 23, 13, 1, 5, 20, 25, 15]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Liczebność mrówek** | **Prawdopodobieństwo wyboru losowej atrakcji** | **Waga feromonów** | **Waga heurystyki** | **Liczba iteracji** | **Współczynnik wyparowania** |
| 10 | 0.3 | 1 | 3 | 1000 | 0.1 |

Plik: A-n32-k5.txt

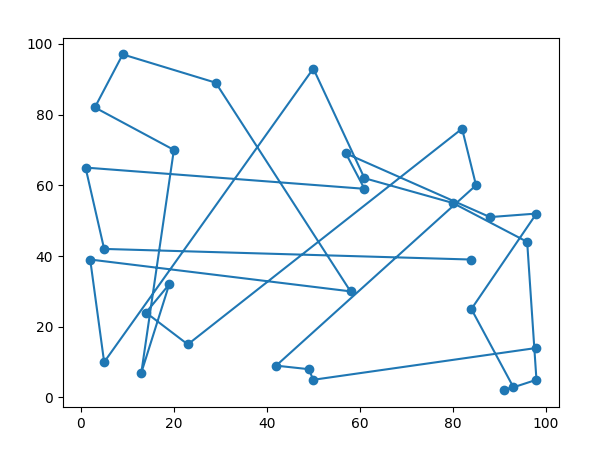


**Całkowity dystans:** 900.113788839789

**Lista kolejno odwiedzonych atrakcji:** [12, 0, 29, 5, 20, 21, 31, 19, 17, 6, 7, 1, 16, 26, 30, 24, 14, 27, 23, 2, 28, 13, 3, 11, 15, 10, 9, 22, 8, 18, 4, 25]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Liczebność mrówek** | **Prawdopodobieństwo wyboru losowej atrakcji** | **Waga feromonów** | **Waga heurystyki** | **Liczba iteracji** | **Współczynnik wyparowania** |
| 10 | 0.3 | 1 | 1 | 1000 | 0.5 |

Plik: A-n32-k5.txt

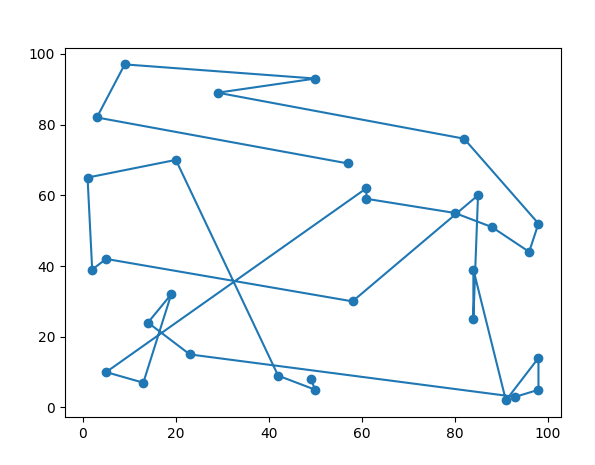


**Całkowity dystans:** 1037.7290120541784

**Lista kolejno odwiedzonych atrakcji:** [21, 2, 3, 23, 30, 0, 28, 8, 18, 4, 29, 10, 25, 5, 6, 9, 11, 20, 24, 26, 1, 31, 17, 19, 13, 12, 16, 27, 14, 15, 22, 7]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Liczebność mrówek** | **Prawdopodobieństwo wyboru losowej atrakcji** | **Waga feromonów** | **Waga heurystyki** | **Liczba iteracji** | **Współczynnik wyparowania** |
| 50 | 0.3 | 2 | 3 | 1000 | 0.5 |

Plik: A-n32-k5.txt



**Całkowity dystans:** 808.2917991924602

**Lista kolejno odwiedzonych atrakcji:** [3, 2, 23, 29, 15, 9, 22, 6, 30, 13, 7, 17, 21, 31, 19, 28, 8, 18, 4, 11, 24, 14, 26, 16, 1, 12, 0, 5, 20, 25, 10, 27]

**Wnioski:**

* Każdy z przetestowanych parametrów ma wpływ na wydajność algorytmu w poszukiwaniu jak najkrótszej drogi
* Waga heurystyki ma większe znaczenie od wagi feromonów w odnajdywaniu bardziej optymalnej drogi
* Zwiększenie populacji mrówek nie jest na tyle wydajne co zwiększenie wag czy współczynnika wyparowywania w odnajdywaniu krótszej drogi, a dodatkowo wydłuża prace algorytmu
* Zwiększenie populacji jak i reszty parametrów maksymalizuje szanse na odnalezienie najbardziej optymalnej i najkrótszej drogi