

# WOJSKOWA AKADEMIA TECHNICZNA



## **Sprawozdanie z przedmiotu: Sztuczna Inteligencja**

Prowadzący: mgr inż. Przemysław Czuba

Autor: Mateusz Jasiński WCY20IY2S1

## 1. W kodzie są przynajmniej dwa poważne błędy (kod nie zgadza się z teorią). Znajdź je i napraw.

- Brak argumentu „total\_iterations” w metodzie solve:

```
def solve(self, total_iterations, evaporation_rate):
```

- Zła wartość zwracana przez metodę visit\_random\_node:

```
def visit_random_node(self):  
    all_nodes = set(range(0, NODE_COUNT))  
    possible_nodes = all_nodes - set(self.visited_nodes)  
    return random.choice(tuple(possible_nodes))
```

Funkcja teraz zwraca losową wartość z set possible\_nodes, która jest indeksem punktu, który zostanie odwiedzony przez mrówkę jako następny.

- Niepoprawna aktualizacja wartości feromonów w tablicy, przez metodę update\_pheromones:

```
def update_pheromones(self, evaporation_rate):  
    for x in range(0, NODE_COUNT):  
        for y in range(0, NODE_COUNT):  
            if x != y:  
                self.pheromone_trails[x][y] = self.pheromone_trails[x][y] *  
evaporation_rate  
                if self.pheromone_trails[x][y] == 0:  
                    self.pheromone_trails[x][y] = sys.float_info.min  
                for ant in self.ant_colony:  
                    for a in range(0, len(ant.visited_nodes)):  
                        if ant.visited_nodes[a - 1] == x and  
ant.visited_nodes[a] == y:  
                            # lub: ant.visited_nodes[a] == x and  
ant.visited_nodes[a - 1] == y  
                            self.pheromone_trails[x][y] += 1 /  
ant.get_distance_travelled()  
                            self.pheromone_trails[y][x] += 1 /  
ant.get_distance_travelled()
```

Funkcja teraz zwiększa wartość feromonu w tablicy wyłącznie dla dróg przez które przechodziła dana mrówka.

2. Spróbuj zbliżyć się do rozwiązania optymalnego dla 48 wierzchołków (33523). Przedstaw dla jakich hiperparametrów algorytm zwraca rozwiązanie optymalne.

```
↑ 9991 Best distance: 34867.0
↓ 9992 Best distance: 34867.0
⌵ 9993 Best distance: 34867.0
⌴ 9994 Best distance: 34867.0
⌵ 9995 Best distance: 34867.0
⌴ 9996 Best distance: 34867.0
⌵ 9997 Best distance: 34867.0
9998 Best distance: 34867.0
9999 Best distance: 34867.0

Process finished with exit code 0
```

Dany wynik został osiągnięty z parametrami:

- ALPHA = 1
- BETA = 3
- RANDOM\_NODE\_FACTOR = 0.03
- NUMBER\_OF\_ANTS\_FACTOR = 0.5
- TOTAL\_ITERATIONS = 10000
- EVAPORATION\_RATE = 0.4

### 3. Jak należałoby zmodyfikować algorytm, aby wyznaczał najkrótszą ścieżkę z punktu A do B? Opisz lub pokaż pomysł implementacji.

Do klasy „Ant” dodaję zmienną „finished = 0”, służy ona do kontrolowania czy została znaleziona droga do określonego punktu oraz dodaje argument do konstruktora, który zawiera indeks punktu A.

```
def __init__(self, start):
    self.visited_nodes = []
    self.visited_nodes.append(start)
    self.finished = 0
```

W metodzie „visit\_node” dodaję sprawdzanie, czy wylosowany punkt jest naszym celem (punktem B).

```
def visit_node(self, pheromone_trails, goal):
    if random.random() < RANDOM_NODE_FACTOR:
        visited_node = self.visit_random_node()
        self.visited_nodes.append(visited_node)
        if visited_node == goal:
            self.finished = 1
    else:
        visited_node =
self.roulette_wheel_selection(self.visit_probabilistic_node(pheromone_trails))
        self.visited_nodes.append(visited_node)
        if visited_node == goal:
            self.finished = 1
```

Z metody „get\_distance\_travelled” usuwam linijkę, która uwzględnia odległość między pierwszym oraz ostatnim punktem (gdyż tu go nie potrzebujemy).

```
def get_distance_travelled(self):
    total_distance = 0
    for a in range(1, len(self.visited_nodes)):
        total_distance +=
node_distances[self.visited_nodes[a]][self.visited_nodes[a-1]]
    return total_distance
```

W metodzie „setup\_ants” dodaję argument „start” zawierający indeks punktu A oraz przekazuje go do konstruktora klasy „Ant”.

```
def setup_ants(self, number_of_ants_factor, start):
    number_of_ants = round(NODE_COUNT * number_of_ants_factor)
    self.ant_colony.clear()
    for i in range(0, number_of_ants):
        self.ant_colony.append(Ant(start))
```

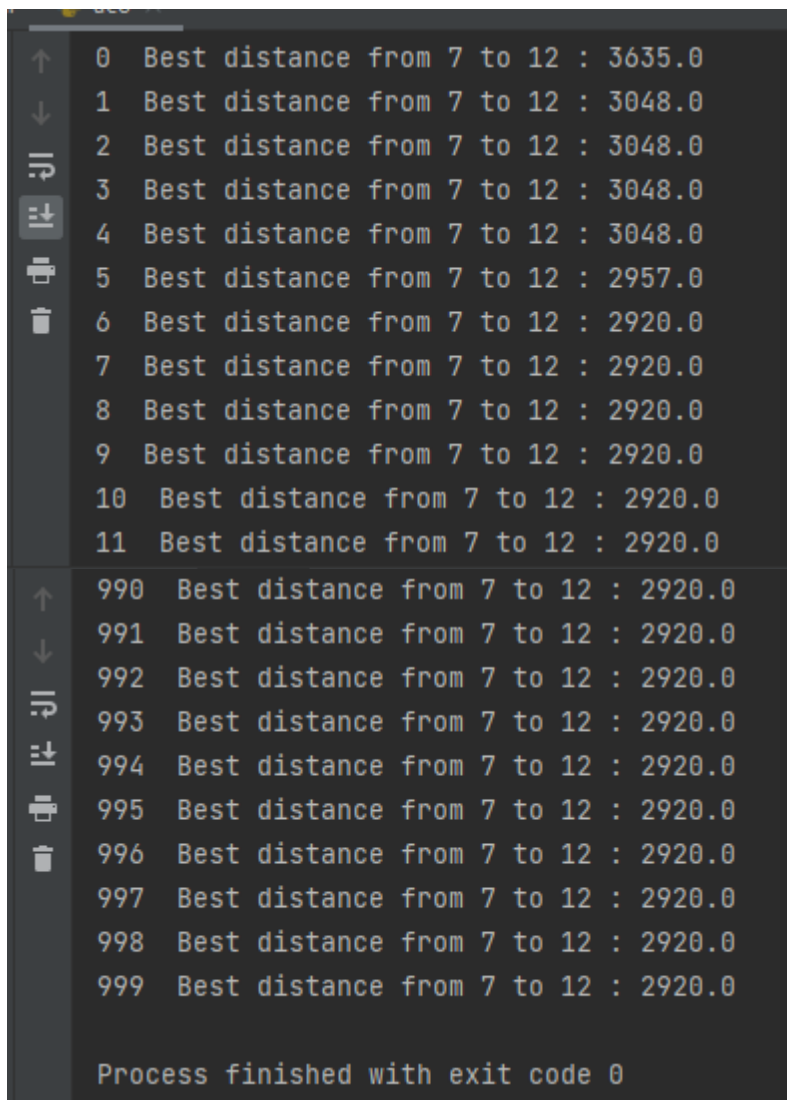
W metodzie „move\_ants” dodaję warunek, który sprawdza, czy mrówka osiągnęła swój cel (punkt B).

```
def move_ants(self, ant_population, goal):
    for ant in ant_population:
        if ant.finished == 0:
            ant.visit_node(self.pheromone_trails, goal)
```

Dodaje dwie dodatkowe zmienne przekazywane do metody „solve”, tj. „POINT\_A” oraz „POINT\_B”, przechowujące indeksy punktu A i punktu B.

```
# Set the percentage of ants based on the total number of nodes
NUMBER_OF_ANTS_FACTOR = 0.5
# Set the number of tours ants must complete
TOTAL_ITERATIONS = 1000
# Set the rate of pheromone evaporation (0.0 - 1.0)
EVAPORATION_RATE = 0.4
POINT_A = 7
POINT_B = 12
aco = ACO(NUMBER_OF_ANTS_FACTOR)
aco.solve(TOTAL_ITERATIONS, EVAPORATION_RATE, POINT_A, POINT_B)
```

Otrzymany wynik wygląda następująco:



```
0 Best distance from 7 to 12 : 3635.0
1 Best distance from 7 to 12 : 3048.0
2 Best distance from 7 to 12 : 3048.0
3 Best distance from 7 to 12 : 3048.0
4 Best distance from 7 to 12 : 3048.0
5 Best distance from 7 to 12 : 2957.0
6 Best distance from 7 to 12 : 2920.0
7 Best distance from 7 to 12 : 2920.0
8 Best distance from 7 to 12 : 2920.0
9 Best distance from 7 to 12 : 2920.0
10 Best distance from 7 to 12 : 2920.0
11 Best distance from 7 to 12 : 2920.0

990 Best distance from 7 to 12 : 2920.0
991 Best distance from 7 to 12 : 2920.0
992 Best distance from 7 to 12 : 2920.0
993 Best distance from 7 to 12 : 2920.0
994 Best distance from 7 to 12 : 2920.0
995 Best distance from 7 to 12 : 2920.0
996 Best distance from 7 to 12 : 2920.0
997 Best distance from 7 to 12 : 2920.0
998 Best distance from 7 to 12 : 2920.0
999 Best distance from 7 to 12 : 2920.0

Process finished with exit code 0
```