Ant Colony Optimization Algorithm

Overview

This repository contains an implementation of the Ant Colony Optimization (ACO) algorithm in Python. The ACO algorithm is a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs. This implementation is tailored for solving the Traveling Salesman Problem (TSP).

Features

- Initialization of pheromones and distances
- Generation of paths based on pheromone and heuristic information
- Updating pheromones based on the quality of paths
- Adjustable parameters for the number of ants, number of best paths to consider, number of iterations, decay rate, and the influence of pheromone versus distance (alpha and beta)

Requirements

- Python 3.x
- NumPy

Usage

Here is an example of how to use the Ant Colony Optimization algorithm:

```
import numpy as np
from ant_colony import AntColony

distances = np.array([[np.inf, 2, 2, 5, 7],
                      [2, np.inf, 4, 8, 2],
                      [2, 4, np.inf, 1, 3],
                      [5, 8, 1, np.inf, 2],
                      [7, 2, 3, 2, np.inf]])

ant_colony = AntColony(distances, n_ants=3, n_best=1, n_iterations=100, decay=0.95, alpha=1, 
shortest_path = ant_colony.run()
print("Shortest path: {}".format(shortest_path))
```

Parameters

- `distances`: A 2D numpy array where the element at [i][j] represents the distance between node i and node j.
- `n_ants`: The number of ants to use in the simulation.
- `n_best`: The number of best ants who deposit pheromone.
- `n_iterations`: The number of iterations to run the simulation.
- `decay`: The rate at which pheromone decays.
- `alpha`: The importance of pheromone (default is 1).
- `beta`: The importance of distance (default is 1).

Classes and Methods

`class AntColony`

`__init__(self, distances, n_ants, n_best, n_iterations, decay, alpha=1, beta=1)`

Initializes the ant colony with the given parameters.

`run(self)`

Executes the ACO algorithm and returns the shortest path found.

`spread_pheromone(self, all_paths, n_best, shortest_path)`

Spreads pheromone on the paths taken by the ants.

`gen_path_dist(self, path)`

Generates the total distance for a given path.

`gen_all_paths(self)`

Generates all paths for all ants.

`gen_path(self, start)`

Generates a path for a single ant starting from the given start node.

`pick_move(self, pheromone, dist, visited)`

Picks the next move for an ant based on the pheromone levels and distances.

Example

The example provided demonstrates how to create an `AntColony` instance with a given distance matrix and run the optimization to find the shortest path.

```
import numpy as np

distances = np.array([[np.inf, 2, 2, 5, 7],
                      [2, np.inf, 4, 8, 2],
                      [2, 4, np.inf, 1, 3],
                      [5, 8, 1, np.inf, 2],
                      [7, 2, 3, 2, np.inf]])

ant_colony = AntColony(distances, n_ants=3, n_best=1, n_iterations=100, decay=0.95, alpha=1, 
shortest_path = ant_colony.run()
print("Shortest path: {}".format(shortest_path))
```

License

The MIT License (MIT)

Acknowledgements

This implementation is based on the Ant Colony Optimization algorithm for solving the Traveling Salesman Problem. More information on the ACO algorithm can be found here.

Feel free to contribute to this project by submitting issues or pull requests.

This README file provides an overview of the project, installation instructions, usage examples, and a brief explanation of the classes and methods. Adjustments can be made based on specific project requirements or preferences.