Instituto Superior Técnico

Object Oriented Programming

MEEC

2018/2019 - 4º Ano, 2º Semestre

Travelling salesmen problem by ant colony optimization

Group 15:	*Number*
Filipa Rente	81324
Gonçalo Valentim	84061
Joana do Carmo	84078

Teacher: Alexandra Sofia Martins de Carvalho

Class: Monday 15:30-17:00

May 9, 2019

Contents

1 Introduction

The objective of the project is to solve a TSP (travelling salesman problem) using an ACO (ant colony optimisation) approach, taking advantage of Object Oriented Programming (Java). The ants will travel the graph randomly and upon finding the path they'll lay down pheromones at the edges that are present in the cycle and re-start the traversing. The objective of this is to find the best *Hamiltonian* cycle (the cycle with the lowest cost). At certain instants of the simulation it will be printed to the terminal the observation's including the *Hamiltonian* cycle and some other requested parameters.

2 Data Structures

The data structures used in our program are the following:

- Graph - Is represented by an array of nodes because the number of nodes is an input parameter and the array has static size and allows for O(1) data access. To the shortest path a list is used because the path is stored in the ants as a list and so when a path is found the list can be copied;
- Node - Uses a List of edges because the number of edges is different for each node and since one edge links two different nodes, the list permits us to have node with different number of edges and to add an edge to two nodes when it is created;
- Colony - An array of ants was used to represent the colony because the number of ants was a static input parameter.
- Pec - To implement the PEC we used a priority queue, as it allows to keep the events ordered. This events are removed from PEC according to the result of the EventComparator, which compares the events in ascending order of their times;
- Ant - Since nodes are constantly being added and removed from the path we found it appropriate to use a List. To keep score of the visited nodes we used an array since the total number of nodes is known and we can access every position with O(1).

3 Object Oriented Solution

3.1 Extensibility

To guarantee that the program is extensible for further developments, the use of interfaces and abstract generic classes is crucial. For that purpose, the Colony is built as a generic abstract class in order to provide other developers the option to extend our solution to similar problems where the colony may have elements different than Ants. Similarly, classes such as the events, simulation and PEC are designed as interfaces so that they are as general and abstract as possible. For instance, the interface for the simulation allows to implement a different run() method which grants the user the ability to explore distinct project goals, other than to find the shortest hamiltonian cycle in a graph.

On the other hand, to prevent the provided implementation from being changed, most fields have private visibility and the methods of some abstract classes such as the graph and colony have protected visibility, which allows their use in subclasses that a client may create but prevent their change in the superclass. Some classes have package visibility so that they are only accessible inside the same package. The HamiltonianSimulation class and the its run() method have public visibility because they represent a service to the client, as for all subclasses that implement the interfaces.

The following interfaces are defined:

- Event interface - the implementation of the method simulateEvent() should be provided;
- Simulation interface - the run() method must be provided;
- PEC interface - the implementation of a PEC must offer public methods to insert new events in the PEC and to remove the next event.

These interfaces keep the implementation open. For example, the client has the possibility of implementing new events - for example a deterministic move - and add them to our solution, as a subclass of the abstract class DeterministicEvent, as long as the simulateEvent() method is provided.

3.2 Polymorphism

Related to polymorphism we have made the classes Event and Simulation abstract. The first one is extended by two other abstract classes, which are StochasticEvent and DeterministicEvent. The events stored in PEC are: Move, Evaporation and ShowResults, where both Move and Evaporation extend StochasticEvent and ShowResults extends DeterministicEvent. The methods overriden in the subclasses are simulateEvent() for the Events and run() for the Simulation.

4 Performance

The performance of the algorithm was tested using graph of different sizes and with different number of ants. The results of those tests can be seen on table 1.

Table 1: Test results for the algorithm.

Nodes	Edges	Colony size	Final Instant	Solution found	Elapsed time
5	4	200	300	YES	0.569 s
10	9	200	300	YES	0.443 s
20	19	200	300	YES	0.510 s
50	20	200	300	YES	0.534 s
50	49	200	300	YES	0.800 s
100	20	200	3000	YES	2.152 s
100	20	200	300	NO	0.683 s
100	20	20	3000	NO	0.607 s
100	99	200	300	YES	1.749 s
100	99	200	3000	YES	5.275 s
100	99	20	3000	YES	1.235 s
1000	500	300	3000	YES	480.783 s
1000	800	200	5000	YES	643.921 s

5 Conclusion

Overall we think the implementation of our project is well done, as it can always find an Hamiltonian cycle if given enough time and the time it takes to run even the big tests is not that long. We can say that the run time always grows with the colony size, the final instant and the number of edges (especially with the number of edges).

To make the performance better we could have tried to use a adjacency matrix instead of a graph and we could have made all the code extensible.