Instituto Superior Técnico

Objected Oriented Programming

Prof. Alexandra Sofia Martins de Carvalho

Discrete Stochastic Simulation

Group 12

Name:	*Number:*
Constança Barroso	81161
Miguel Leão	76934
Pedro Vieira	76738

May 9, 2019

1 Introduction

The goal of this project is to develop a simulation that implements an Ant Colony Optimization (ACO) algorithm. In this simulation, ants randomly traverse a graph and, upon finding a solution to the Traveling Salesman Problem (TSP), lay down pheromone trails.

The evolution of the simulation is governed by certain events. The time of this events occur depends on random variables with exponential probability distributions. The mean value of these distributions is calculated with parameters provided in the input file.

The events are the following:

1. Moves
2. Evaporations

In this project we have considered another event, the Prints, which is the event that presents current results at certain times.
The main goal of the simulation is to find the shortest Hamiltonian cycle in a weighted graph.

2 Main Data Structures

2.1 Array Lists

Arrays were used to represent the paths of the ants (set of nodes being visited).
The decision of using this type of data structures was based on the need of accessing this values a considerable amount of times by its indexes and the well known dimensions of the needed arrays. So there was no need to use Linked Lists (dynamic memory).

2.2 Priority Queue

Priority queues were used as sets of events to form the PEC. The handling of events is quite similar to the handling of a stack/priority queue with the difference that the comparative parameter is the event time, events that happen sooner are always on top of the queue and will be popped in order whenever needed.

2.3 Bidimensional Array

Bidimensional arrays were used to represent the graph as an adjacency matrix. We found this to be the best solution for the same reason as Array Lists were used, we prioritized time of search over, in this case, memory.
It was also used in the path, to keep tracking of the already visited nodes in each visit. Because of this it is possible to do back tracking.

3 OOP Principals

3.1 Open-closed Principal

A very good example of the open-closed principal is in the Event class. All subclasses of the event can override an abstract method of the class, so it is opened for extension and closed for modification, as we don't need to change it in order to extend it.

3.2 Polymorphism

There is plenty of polymorphism in this implementation. For example, the PEC stores instances of the class Event (which is abstract), and we put instances of its subclasses in the PEC. Furthermore, the class Graph can be extended to two types os graphs.

3.3 Abstraction

We use abstraction when we create abstract methods in the superclasses so it can be overridden. Another example is the use of the classes from java.util, as in the use of the priority queue.

3.4 Coupling

We tried not to have a strong coupling in our project. For example, the graph elements has no direct knowledge of the path element. From the perspective of the PEC/event system, the event can be any data structure with any coordinate system.

4 Final Remarks

The application performs as expected. Some conducted tests found that the program took a reasonable amount of time to run without errors or any setbacks. To improve our implementation of the ACO algorithm we suggest a change in the the adjacency matrix and array list of the class path. We could use a structure with dynamic memory to search for the path and the nodes already visited (back tracking).