TSPRanker

Dumitrescu George-Alex 2A5

Alexandru Ioan Cuza University of Iași

1 Introduction

This document describes the final project at Advanced Programming (Java). The project that I chose to make revolves around methods of solving the Travelling Saleseman Problem (constructive, evolutive and exact methods). I chose this project because I saw a opportunity to implement everything that I have learned such as creating a GUI, interacting with databases, REST with Spring, socket programming, multi-threading while also implementing some interesting algorithms, some of them studied last semester at Genetic algorithms.

2 Implemented algorithms

2.1 Introduction

I implemented 2 kinds of algorithms: Constructive and Evolutive. To find the exact value I used the **JGraphT** library that implements the Held-Karp algorithm that finds the exact valute in $O(2^n \cdot n^2)$ where n is the number of vertices. The client can compare this algorithms between them while also being able to chose a test instance and trying to solve better than the "Server".

2.2 Constructive

I implemented three Tour Construction Algorithms: **Nearest Neighbour, Double Tree, and Christofides**. Their performance varies for smaller instances but for the bigger instances Christofides (3/2) followed by Double tree (2) and lastly Nearest Neighbour. All the implementations are standard, implemented fully by me except the algorithm for finding the minimum cost perfect matching used for the Christofides algorithm that was to difficult, so I used the JGraphT library.

2.3 Evolutive

I implemented three Evolutive Algorithms: **Genetic Algorithm, Simulated Annealing, Ant Colony Optimization**. All of them have fixed parameters on the server. All of them for the smaller instances find the optimum but for the bigger instances only the Genetic Algorithm performs well. The rest are to slow.

For the Genetic algorithm I have a initial population of 200 and I run it for 2000 generations. The mutation is based on 2-opt, thus being a preatty agressive mutation, that works well on bigger instances and the crossover algorithm is order crossover. Also I use elitism.

For the Simulated Annealing Algorithm I used two ways of finding the neighbours (transport and reversal) and for each time that I lower the temperature I check 100 neighbours thus doing a slower and more ”explorative” search.

For the Ant Colony Optimization I implemented the basic algorithm without using elitism, min-max or other optimizations. It works great on small instances but on bigger ones because it is implemented without optimization the path finding of an ant takes a lot of time, and the algorithm becomes very slow.

3 Application functionalities

3.1 Server

The server has a database of problem instances and solutions. For each problem instance it uses my implementation of the algorithms to compute solutions. All the solutions for one instance are ranked and can be compared between them or with solutions added by the client. The server also announces the client when a new problem instance is available to be solved.

3.2 Client

The client is able to add problem instances or solutions to the server. For each problem instance he can download it, and post a solution to it and view how it compares to the server solutions and other users solutions. He can also visualize the distribution of the points on the plane.

3.3 Test Generator

A small test generator that I made so that I can test my application and also convert the offictial TSP instances from TSPlib in the JSON format.

4 Technologies used

4.1 OOP model

The server is made out of five main packages: dao, model, service, TSPController and util. In the DAO package is all the database interaction code with two DAO with JDBC for each table: ”problem_instance” and ”solutions”. The model package has a few classes that specify how data is represented. The service package holds all the code related to the algorithms implemented. All of the TSP solvers implement the TSPSolver interface that is runned by a TSPSolverRunner. The TSPController package contains the RESTController and the util package has a few helpful classes such as one to implement a DSU, etc.

The client has two pakages: model and client. As in the server, the model package defines how the data is represented and in the client package I have the main JavaFX application.

4.2 Graphical interface

On the client I have a tree page interface. The first page has a list with available instances that are on the server and buttons that a client can use to: select a instance, add a instance, and refresh.

After selecting a instance, the client is moved to the second page where he can view a ranking of all the available solutions on the server. Here he also has the posibility to go to page 3 where he can view a drawing of the instance.

The first two pages have a Text box where the client can see what are the responses of the server and when a new instance is available on the server the client is announced with a pop-up (alert).

4.3 Database communication

The Server uses JDBC to communicate to the two tables. One has the solutions and one the instances. Because multiple threads could make write and read operations I use a Hikari connection pool. Most of the files are JSONs and even the problem instance is inserted in a table as a JSON. I use postgresql.

4.4 Algorithmic part

The seven algorithms implemented for TSP.

4.5 REST services

The server works as a REST API made in Spring. A client has the posibility to POST/GET/DELETE problem instances and to POST/GET problem solutions.

4.6 Socket programming

The project uses socket programming to announce the client in real time when a new instance is available to be solved.

4.7 Multi-threading

I use multi-threading when solving a genetic algorithm on the server. I start a thread that starts another seven threads that run a TSP algorithm each. Moreover I use multi-threading in the context of sockets. The server has a writer thread and the client a listener thread.

4.8 JUnit testing

The project has JUnit testing for most of the classes in the util package and the service package.

4.9 Conclusions

In conclusion, I tried to use all the learned technologies while also implementing a functional and interesting project.

References

1. Peter J.B. Hancock *Selection Methods for Evolutionary Algorithms*
2. https://pdfs.semanticscholar.org/5a25/a4d30528160eef96adbce1d7b03507ebd3d7.pdf
3. Ulrich Bodenhofer https://www.flll.jku.at/div/teaching/Ga/GA-Notes.pdf
4. Agoston E Eibe, Robert Hinterdi, Zbigniew Michalewicz *Parameter Control in Evolutionary Algorithms*
5. https://jgrapht.org/javadoc-SNAPSHOT/index.html
6. https://profs.info.uaic.ro/ acf/java/
7. https://www.jetbrains.com/idea/guide/tutorials/
8. https://profs.info.uaic.ro/ acf/java/proiecte.html