Scuola Universitaria Professionale
della Svizzera Italiana

Dipartimento
Tecnologie
Innovative

Traveling Salesman Problem
16th Coppa Algoritmi

Student: Niko Storni

Date: 6 maggio 2016

```
final long seed = System.currentTimeMillis();
final Random random = new Random(seed);
```

Problem

The problem we're confronted with consists on finding the shortest Hamiltonian tour in a weighted graph. TSP stands for Traveling Salesman Problem and describes a set of given cities and the relative distance between each of them. The challenge is to find the shortest path possible that covers all the cities and returns to the origin city. It is an NP-Hard problem that can be solved partially or even optimally in many scenarios. During this competition We will be competing against each-other for the lowest possible error rate in less than 3 minutes for the following 10 selected problems:

- *eil76*;
- *kroA100*;
- *ch130*;
- *d198*;
- *lin318*;
- *pr439*;
- *pcb442*;
- *rat783*;
- *u1060*;
- *fl1577*;

Implemented Solutions

Constructive Algorithm

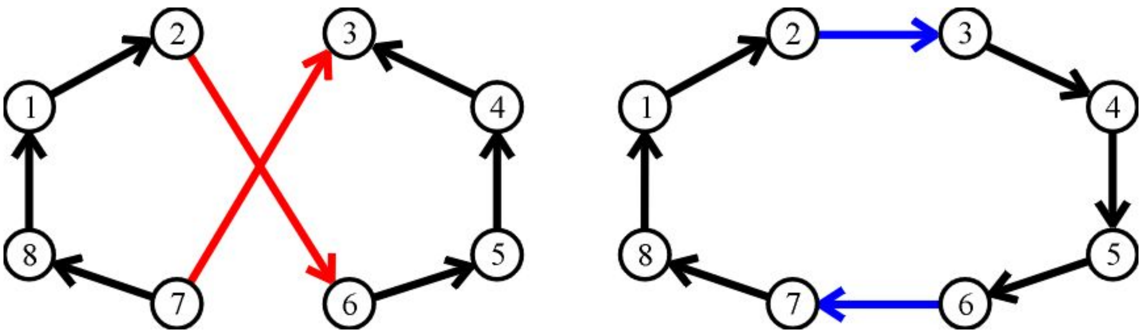
To begin with, after parsing the problems, I implemented the Nearest-Neighbor algorithm. This algorithm, given an initial city, finds (in a greedy way) the closest city to connect to as next destination. It does that up until all edges are taken thus constructing a complete and valid tour.

This implementation is certainly not good enough to even get close to an acceptable solution as it provides results with very high error ratio. I had to implement more algorithms in order to bring down the tour length.

In a later moment, to verify the correctness of my implementation of the candidates lists, I implemented this algorithm using the lists as primary lookup table for the construction of the tour. I didn't bother verifying the difference in efficiency but if there is even one, it's probably so small that it wouldn't make sense accounting it.

Local optimization algorithms

For the local optimization I implemented a variant of the 2-opt algorithm.
This algorithm tries to remove all crossed arches by swapping subsets of the tour.



http://www.devx.com/dotnet/Article/33574/0/page/3

During the whole project I changed my 2-opt implementation in attempts of reducing the time spent looking for better routes.

My attempts include implementations of first improvement 2-opt that break free of the loop as soon as an improving possible swap is found, the implementation of candidate lists with and without fallback on failure to find a better route, to a swap-as-soon-as-possible solution where a swap is performed as soon as a candidate tour is found.

The latter resulted to be the best choice when dealing with big problems such as the fl1577 where a single cycle of 2-opt would take up to 4 seconds compared to up to 1 second when implementing this variant.

This local search alone manages to drop the error ratio to around 4

Meta Heuristic Algorithms

In order to bring the error ration further down it's necessary to implement a meta heuristic algorithm.

Since most of my classmates opted for the simulated annealing, I decided to go for a more complex yet more promising search algorithm: The Ant Colony System.

This algorithm, as found in several papers online and as explained during classes, exploits the way ants find optimal routes to food.

The way the algorithm works is described as follows:
A given number of ants makes up a virtual colony. Each ant is placed over a random city of the tour (previously computed using a constructive algorithm (in my case the Nearest Neighbor), then each ant is allowed to take as many steps as it's required for them to walk over all the cities.

The way ants move to a next city can be broken down in two different situations:

- Greedy next
- Exploration

The first looks at the cities that are yet not visited and picks the one that looks more appealing. The appealing factor is given by a mix of the pheromone level between the two cities and the inverse of the distance to be walked.

The second looks at all the cities that aren't yet visited, discards the most appealing one, sorts all the cities in order from the least appealing and the most appealing city and pseudo-randomly picks the next city to explore. Not every city has the same probability to be chosen as next city, in fact, the most appealing city will have much more chances of being picked than the least appealing one.

After each step, the algorithm simulates evaporation by removing a little bit of pheromone from the arch that was just used.

At the end of the tour, each ant produced a valid tour which has to be optimized with a local search. I used my 2-opt implementation which did the job pretty well.

After each tour has been optimized, the best tour is selected. If such tour has a better solution than the global best tour, then it becomes the global best tour.

In the end, the globally best tour is allowed to let down a little bit of pheromone along its tour. Such action biases the other ants during the next tour so to prefer the best route rather than other trivial directions.

In order to speed up the ACO I researched several papers online and found some good suggestions as well as bad suggestions. I ended up implementing the following features:

Candidates lists: a list of about 8

Lone fire ant: when profiling the ACO, I noticed that the first iteration would potentially take up to 10 seconds for big TSPs as the tour would be completely messed up. The 2-opt algorithm would waste time optimizing n (n being the number of ants) different solutions when in fact it was completely useless. To avoid this huge loss of time I allowed a single ant only to be optimized after the first ant iteration.

Genetic Ants: I stumbled upon a paper [1] that discussed about the possibility of improving the algorithm by giving ants the capability to remember parts of the best tour. I was curious about this implementation so I coded it. After each iteration, before placing down an ant in a random city, a dice is throw (virtually) and if the odds are with the ant, it will suddenly *remember* part of the tour that was found by the best ant globally. Such memorize cities are marked as visited and added in its local solution.

How to execute the program

Platform

The computer used to program this project is a Lenovo Thinkpad T440p, however the PC that executes the runs in order to comply with the CUP the algorithm will run on the desktop computer that was assigned to each of us during this year. The specifications of the system are:

Tabella 1: Used platform

Operative System	Ubuntu 14.04 LTS
Kernel	3.19.0-58-generic x86_64
CPU	Intel Core 2 Quad Q9400 @ 2.66GHz x4
RAM	4GB

Compilation and execution

To compile the program: import it to eclipse as Maven project, select run as, hit "Maven Install". The jar will be generated. Place the jar file somewhere along with the directory containing the TSP problems (it should be named *problems*). scripts are provided to run the executable, therefore running a a problem takes as much as typing:
./problemName.sh i.e.

```
$ ./d198.sh
```

to run the jar directly you can type:

```
$ java -jar jarName.jar problemName alpha beta rho greedyness
    memory colonySize timems ev_seed
```

Here is an example of a possible output:

```
$ java -jar algo.cup.niko-0.0.1-SNAPSHOT.jar fl1577 2 0.1 0.1
    0.995 0 15 179500 1462116966413
Seed for fl1577: 1462116966413 with a performance of: 0.54384464%
     --- population: 15 Greedyness: 0.995 Memoryness: 0.0 Beta:
    0.1 Rho: 0.1 Alpha: 2.0
```

Results

The following table 2 shows the results that were found using my implementation of the ACO Algorithm. More detailed results can be found in the directory *results* attached with this documentation.

Tabella 2: Best solutions found

TSP	Optimum	Result	Error	Seed
eil76	538	538	0%	42
kroA100	21282	21282	0%	1234567890
ch130	6110	6110	0%	69
d198	15780	15780	0%	1337
lin318	42029	42029	0%	16071993
pcb442	50778	50785	0.013785498%	1462343763254
pr439	107217	107217	0%	8008135
rat783	8806	8817	0.124914825%	1462344733968
u1060	224094	224981	0.39581606%	1462378695007
fl1577	22249	22326	0.34608296%	1462116966413
average			0.0880599343%	

Conclusions

This project was extremely challenging and fun to implement. I spent a lot of my time trying to figure out better ways of implementing the algorithms.

I decided to implement the Ants Colony Optimization algorithm because most of my classmates opted for an easier one. I like to be challenged and do things differently. I first wanted to implement the genetic algorithm however it looked like a little bit too complicated and not as efficient as the one I later implemented, so i went for this one instead.

I tried adding as much uniqueness to the project as I could, and I believe that I succeeded. Unfortunately I will not win the cup as there are other classmates that managed to outperform me with different implementations, kudos to them, however I am very satisfied with my results as comparing back to previous winners, I definitely would have won most of them (not accounting computational power differences though).

I must thank Mr. Luca Gambardella for his insights and suggestions as well as for teaching us this extremely interesting algorithm. Also a thanks goes out to my classmates for the great challenge, and to Mr. Marco Cinus for assisting us during practical hours.

Bibliografia

[1] Bifan Li, Lipo Wang, Wu Song *Ant Colony Optimization for the Traveling Salesman Problem Based on Ants with Memory*, http://www.ntu.edu.sg/home/elpwang/PDF_web/08_ICNC_LiBifan.pdf