Final project
Travelling Salesman Problem

Introduction

The Travelling Salesman Problem (TSP) asks the following question : "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city ?" It is an $NP-$hard problem in combinatorial optimization, important in operations research and theoretical computer science.

The problem was first formulated in 1930 and is one of the most intensively studied problems in optimization. It is used as a benchmark for many optimization methods. Even though the problem is computationally difficult, many heuristics and exact algorithms are known, so that some instances with tens of thousands of cities can be solved completely and even problems with millions of cities can be approximated within a small fraction of 1%.

Formally, let $G = (V, E, w)$ be a complete undirected graph such that $|V| = n$ and $w : E \to \mathbb{R}_{\geq 0}$ the weight function. That is, G is the graph K_n with weights on its edges. In this model, each vertex represents a city and each weighted edge joining two vertices represents the distance between the corresponding cities.

Therefore, given a weighted undirected complete graph, the TSP consists in finding a Hamiltonian cycle (a cycle that goes through all vertices exactly one) that minimises the sum of its weights.

Exercises

1. Describe (other) real-life situations that can be modelled as the TSP.
2. Develop and implement an exact algorithm to solve the TSP.
3. Develop and implement a constructive heuristic to solve the TSP.
4. Develop and implement a local search heuristic to solve the TSP.
5. Develop and implement an algorithm that uses the GRASP meta-heuristic [1] for the TSP. In each iteration of GRASP use as a first phase some modification of the heuristic given in 3, and as a second phase the heuristic given in 4. **SET** all the parameters involved (size of the restricted list of candidates (RLC), stop criteria, etc.) through experiments. **JUSTIFY** your elections and the choice of the test instances considered to decide your parameters.
6. For each of the methods 2 to 5 :
   - Explain in detail your algorithm. **DO NOT** give the code, use pseudo-code instead.
   - Calculate its time complexity using the pseudo-code given before.
   - Try to describe instances of the TSP for which the method does not provide an optimal solution. How bad the obtained solution can be with respect to the optimal solution ?
   - Run experiments that allow to observe the performance of the algorithm. Compare the execution time with its theoretical time complexity. Also, compare the quality of the obtained solutions and the execution time, with respect to the input size (and other parameters if appropriate). Within the cases of test should also be included, as pathological cases, those described in the previous item. In case the algorithm has some configurable parameters that determine its behaviour (the meta-heuristic for example, although it remains open to others as well), you should run experiments varying the values of the parameters and choose, if possible, the settings that provide best results for the instances used. Present the results obtained using appropriate graphics.

7. Once you have chosen the best configuration values for each heuristic, run experiments on a **new set of instances** to observe the performance of the methods and compare again the quality of the solutions obtained and the execution time. Present all the results obtained by using appropriate graphics and present conclusions about your work.

General guidelines

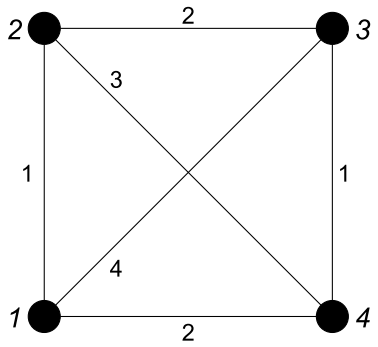
1. A report tackling all points described above should be written (in english, of course) and uploaded in pdf format. You should follow the guidelines described in the document "Normes de mise en page des dossiers", Version 3.2, by Matéo Sorin. Use of `LaTeX` [2] is highly recommended.
2. You can use any programming language you prefer to implement your algorithms (BUT use the same for all of them). When you upload you project, you should include within the source code, a file called "`readme.txt`" that explains how to compile/execute the source code.
3. Your algorithms should read a input file (check the example below) that contains $n + 1$ lines as follows :
   - The first line have the number n of vertices.
   - The next n lines contain the rows of the adjacency matrix (with the weights separated by one space) of the complete graph. Note that the matrix is symmetric.
4. Your algorithms should write a file containing the solution (check the example below) as follows :
   - The name of the file should be "`instance_method.out`" where "`instance`" is the name of the input file (without the extension if any) and "`method`" should be one among "`exact`", "`constructive`", "`local_search`" and "`grasp`".
   - The file should contain two lines. The first one contains the number of the vertices separated by one space, representing in which order the cities should be visited. The second line should contain the weight of the cycle.
5. You should handout as well the test instances you have used to set the configurable parameters and the final set of instances used to compare performances (as they must be referenced in the report).
6. **VERY IMPORTANT :** While you're running measured experiments,
   - **DO NOT** do anything else with your computer **AND** close all unnecessary programs (like chrome, firefox, etc) that might be running. Not doing so, might affect the performance of your algorithms and give inaccurate results. **DO NOT** plug/unplug your computer neither.
   - **USE** the same computer (and under the same conditions specified above) to compare time execution of algorithms. It makes no sense (therefore it will give you wrong results), if you compare time execution, for example, of algorithm A against algorithm B, in two different computers with different processors, memory size, etc.

Example

Consider the file "`test.in`" :

```
4
0 1 3 2
1 0 2 4
3 2 0 1
2 4 1 0
```

This file corresponds to the following graph :



Therefore if we run the exact algorithm in that instance, the file "`test_exact.out`" will be created containing the following two lines only.

```
1 2 3 4
6
```

This corresponds to the cycle $1, 2, 3, 4, 1$ with total weight $1 + 2 + 1 + 2 = 6$.

Deadline & Upload instructions

The deadline of the project is on Friday April 10th, 14h55, on the ENT.
NO FILES WILL BE ACCEPTED AFTER THE DEADLINE.

You should upload only **ONE** compressed file called "`team_XX.zip`", where "`XX`" is the number of your team. Only **ONE** member of the team should upload the file (not the four of you). This file should contain the following folders :

- "`Report`" : containing the pdf file of your report.
- One folder for each algorithm called "`Method`", where "`Method`" should be "`Exact`", "`Constructive`", "`Local_search`" and "`Grasp`". Each of these folders must contain two sub-folders.
  i. Sub-folder "`Source`" : containing the source code of the algorithm plus the "`readme.txt`" file with the compiling/executing instructions.
  ii. Sub-folder "`Instances`" : containing the instances you used to test your algorithms individually and/or to set configurable parameters (that should be referenced in the report). This folder should contain as well, the "`.out`" files with the obtained solutions for each input instance.
- "`Instances`" : containing the **new set of instances** you used to compare performance between algorithms as described before. You should again include, the "`.out`" files with the obtained solutions for each algorithm and input instance.

If you use `github` or any other repository manager, **DELETE** all hidden files and folders that the manager creates. **Not doing so will lead to a penalty of points**.

References

[1] Thomas A. Feo and Mauricio G. C. Resende. *Greedy randomized adaptive search procedures*. Journal of Global Optimization, pages 1–27, 1995.

[2] *An introduction to LaTeX*. LaTeX project (`https://www.latex-project.org/about/`).