Optimization and Decision

Asymmetric traveling salesman problem

Project report

Bologna Master Degree in Mechanical Engineering
Academic Year 2021 – 2022

2nd Semester

Students

Afonso Brandão Oliveira Fernandes, n.º 93204
Duarte Sequeira Correia, n.º 93242
Miguel André Milhazes, n.º 93305

Professors

Susana M. Vieira
Bernardo Marreiros Firme
Miguel de Sousa Esteves Martins

Lisbon, 25th of April 2022

Index

1 Introduction

The asymmetric travelling salesman problem (ATSP) is a special case of the travelling salesman problem (TSP). It is a NP-hard combinatorial problem and is described as, given a set of n nodes and distances for each pair of nodes, find a roundtrip of minimal total length, visiting each node exactly once. In this case, the distance from node i to node j and the distance from node j to node i may be different. Concluding, the goal is to find a Hamiltonian tour of minimal length on a fully connected graph

In order to solve it, three different methods are implemented and tested against two benchmark results obtained in the TSPList [1]. The methods used are the nearest neighbour, the tabu search and the ant colony optimization.The benchmarks used are *ft53*, with 53 nodes and *kro124p*, with 100 nodes. The best known solution for each problem is 6905 and 36230, respectively.

2 Formulation of the problem

The ATSP has for long been researched and has several possible formulations. In this report, the problem will be formulated as a binary integer linear programming problem. Firstly, the cities have to be enumerated from 1 to n, n being the number of nodes in the case study, followed by the "cost" for travelling from one node (i) to the other (j), $\mathbf{c_{ij}}$. This cost parameter in the problems at hand is the distance between two cities.

The Asymmetric nature of the problem makes it possible for the cost of travelling the arc (i, j) to not be the same as the other way around, (j, i), i.e., $\mathbf{c_{ij}} \neq \mathbf{c_{ji}}$. This values are retrieved from a text document in matrix form, where $\mathbf{c_{ij}}$ is the value in row i and column j in said matrix. As travelling from one city to the same city doesn't translate to anything in the problem, $\mathbf{c_{ii}}$ has a value far superior than the others, in order not to compromise the optimal solution, for every i = 1, ..., n.

A binary integer linear programming problem uses binary variables $\mathbf{x_{ij}}$, defined by:

$$\mathbf{x_{ij}} = \begin{cases} 1 & \text{, if arc } (i, j) \text{ is part of the optimal solution} \\ 0 & \text{, if not} \end{cases} \tag{2.1}$$

And presented as the following model:

$$Min : \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.2}$$

Subject to:

$$\sum_{i=1}^{n} x_{ij} = 1, \quad i = 1, ..., n \tag{2.3}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, ..., n \tag{2.4}$$

$$x_{ij} \in \{0, 1\}, \quad i, j = 1, ..., n \tag{2.5}$$

This model does not regard the possibility of sub-tours to appear in the solution, the solution must be a single tour passing through all cities, not a union of smaller tours.

2.1 Sub-tour Elimination Constraints

To suppress said possibility, a new constraint has to be added, using the Dantzig–Fulkerson–Johnson formulation, the last constraint becomes:

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \quad \forall S \subset \{1, ..., n\} : S \neq \varnothing \tag{2.6}$$

This guarantees that no subset S can form a sub-tour.

3 Methods and Algorithms

In order to solve the problem, three different methods are implemented: nearest neighbour (NN), tabu search (TS) and ant colony optimization (ACO).

The choice of algorithms was simple. To begin with, a simple algorithm is tested, the NN algorithm, since it was one of the first algorithms used to solve the TSP and the fact that it produces a fast solution, although sometimes not the optimal, and it is simple to compute. Then, the TS algorithm was used, with the initial condition guessed with the NN, explained in section 3.2. Through this, it is possible to directly compare the effectiveness of the meta-heuristic algorithm. In the end, a different meta heuristic algorithm was used, ACO, in order to have some diversification in the methods implemented.

3.1 Nearest neighbour

The nearest neighbour algorithm is a simple algorithm that works with the following steps:

1. Start in a random node;
2. Determine the node with the smallest distance to the starting point and connect to the starting point;
3. The last node is now the new starting point;
4. Repeat the procedure, excluding the nodes already visited
5. If all nodes are visited once, the algorithm is done. If not, go back to step 3.

This type of algorithm has a tendency to become stuck in suboptimal regions also known as local minimum.

In order to have a better comparison between the results and since the problem being studied is relatively small (50 to 100 nodes), instead of starting in a random node, the algorithm was run $(n_{nodes} - 1)$ times so that all the nodes can be the starting point. The results were stored and will be discussed in section 4.1.

3.2 Tabu Search

Tabu search is a meta-heuristic algorithm based on a local search method. It explores the solution space by replacing recent solutions with the best non visited neighbouring solution. In order to avoid the local minimum problem, TS allows moves that give a worse solution if no improving move is available. Also, it maintains a list of neighbour generation moves that are considered forbidden and solutions that can be obtained through those moves are ignored. This tabu list is always changing since, after entering the list, a move stays there for a pre-specified number of iterations. This number is called the tabu tenure of the move.In the end, the stopping criteria can either be a maximum number of iterations/time or if there is no better solution in the previous iterations. The steps for this algorithm are:

1. Start with an initial solution, in this implementation it is obtained through the NN algorithm;
2. Generate alternative neighbouring solutions to the current solution, if they are in the tabu list they are discarded;

3. Choose the best solutions out of the alternative ones found previously;
4. Update the tabu list by removing the moves that expired the tabu tenure and add the new moves obtained in step 3;
5. If the stopping criteria is reached, the algorithm is done. If not, go back to step 2.

3.3 Ant Colony Optimization

Ant colony optimization is a population-based meta-heuristic algorithm. A set of software agents called artificial ants incrementally build solutions by moving through a parameter space representing all possible solutions, in this case, on a weighted graph. The construction of the solution is a stochastic process biased by a pheromone model.

The steps to this iterative algorithm are:

1. An ant completes a tour (a solution), choosing nodes by a stochastic mechanism;
2. Pheromones are laid out according to the quality of the solution, in this case according to how small the distance was;
3. Another ant repeats the procedure;
4. The next ant selects the path to go, again via a stochastic mechanism, but this time biased by the pheromone quantity left in each node;
5. Update the pheromone quantity in each node as every ant tries a path;
6. If the majority of the ants is following the same path, the algorithm can stop and that path is the optimal one.

When implementing this algorithm, a few parameters have to be taken into the account, in this section goes a brief explanation of them and, in section 4.3 a comparison between them, in order to find the optimal values is done.

- Number of ants:

  The number of ants affects how many ants trying new paths the algorithm has. Fewer ants may lead to insufficient exploration, and too many ants can cause an inefficient algorithm.

- Pheromone (τ).

- Pheromone evaporation (ρ):

  Pheromone evaporation avoids unlimited accumulation of the pheromone trails leading to quick convergence to suboptimal path, reinforces good solutions and enables the algorithm to forget bad decisions previously taken.It is extremely important in early stages when ants provide various worse solutions.

- Pheromone constant (Q):

  The amount of pheromones laid.If this parameter is too high, it may lead to a quick convergence in a suboptimal path, if it is too low, it can take a long time to find an optimal path.

- Pheromone exponential weight (α):

  Controls the importance given to the pheromone trail.

- Heuristic exponential weight (β):

  Controls the importance given to the heuristic information.

4 Results obtained

4.1 Nearest neighbour results

4.2 Tabu Search results

4.3 Ant Colony Optimization results

4.4 Comparison of results

References

[1] Ruprecht-Karls-Universität Heidelberg. *TSPLib*. URL: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/. (accessed: 10.04.2022).

[2] Dorigo Marco and Stützle Thomas. *Ant Colony Optimization*. MIT Press. 2004.

[3] M Vieira Susana. *Course notes*. 2022.