University of Southern Denmark

DM852

14th of May 2020

Heuristics & Approximations Algorithms - Project 02

Submitted To:
Marco Chiarandini
Lene Monrad Favrholdt
IMADA
Institute of Mathematics & Computer Science

Submitted By:
Alexander Lerche Falk
Spring - Master of Computer Science

Contents

1 Introduction

The goal of this project is to show the knowledge acquired by the student in regards to Heuristic and Local Search Algorithms. The algorithms are developed to suit the Capacitated Vehicle Routing Problem (CVRP). CVRP is a combinatorial optimization problem, where a set of n customers have to be visited by m vehicles. Each customer can only be visited once and the vehicles have a capacity limit, which cannot be exceeded. All vehicles start and end their route at the depot (the starting point). The objective is to minimize the travelling cost with as few vehicles as possible. Metaheuristics are the next step from heuristics and local search algorithms. Metaheuristics are combinations of the two, where a metaheuristic algorithm starts of by constructing a solution to a problem and then tries to optimize the solution with a local search algrorithm. The difference between a metaheuristic algorithm and just combining a heuristic and a local search algorithm, is the extra set of mechanisms which comes with metaheuristics and the on-going execution until a criteria is met (time or a certain number of steps). E.g. of a mechanism could be: restart of the algorithm. If the algorithm is stuck in a local optimum, a restart can be a mechanism to escape and try again. Another mechanism is saving solutions in memory to try and tackle a problem from different solution generated through the execution cycle. The latter mechanism is being used in this project, where Tabu Search have been implemented in two versions.

At the end of the report, the performance results are shown of the algorithm. All the code can be found at GitHub

2 Tabu Search

Tabu Search [1] is a metaheuristic algorithm, which tries to accommodate the issue with being stuck in a local optimum. It does so by using a memory mechanism, which is called the Tabu List. This list contains solutions, which have improved solution, and should not be touched (they are tabu) until a criteria is met. A criteria could be the length of the tabu list. When exceeded, the latest is removed, and allowed to be touched again, and the new is added. The inmemory structure is commonly implemented in two ways:

- ShortTerm: stores a list of candidate solutions. The solutions in here are kept until they are removed by some threshold. This is to avoid revisiting candidate solutions as much as possible.
- LongTerm: this can be used in addition with ShortTerm to try and increase the diversity of the search space

In the implementation of the project, two ShortTerm memory structures have been implemented, and no LongTerm. The first ShortTerm structure contains the candidate solutions and the second ShortTerm structure contains a list of sets of edges. To generate a solution as starting point for Tabu Search, the heuristic Nearest Neightbour Algorithm [2] is used. From here, the local search twoopt algorithm [3] is ran on the solution to make swaps in the routes. The twoopt is the main driver of trying to generate candidate solutions by swapping edges to reduce the cost.
The two implementations of Tabu Search uses Nearest Neightbour to generate a starting point and twoopt for swapping edges, but the first version only uses twoopt on one route: the route a vehicle has been assigned. The issue here is it gets stuck in a Local Optimum quickly. To accommodate the problem, a second version of Tabu Search has been implemented. This version uses twoopt against two routes instead of one. A swap between two routes (two vehicles), the cost has to be reduced for both vehicles and their capacity cannot be exceeded.
The algorithm is described in pseudocode:

The algorithm 1 uses two ShortTerm structures: one for edges and one for solutions. When TwoOpt does its swapping, it has a check if the edges it is swapping is a part of the tabu list for edges. This is done to avoid a swap on an edge, which improved the solution, and therefore should not be touched until a criteria is met. The criteria is set on the length of the tabu list for edges. When the length exceed the defined max length, the edge in the list is removed, and the new one is appended. The tabu list for solutions is more broad and keeps a memory structure for the solutions itself. If a solution is found by swapping, and it is already contained in the list of solutions, then it is "discarded". If it is not in the list and not a better solution,

Algorithm 1 Tabu Search Version 1

Require: Set of customers

Ensure: *Solution* (solution to the CVRP instance)

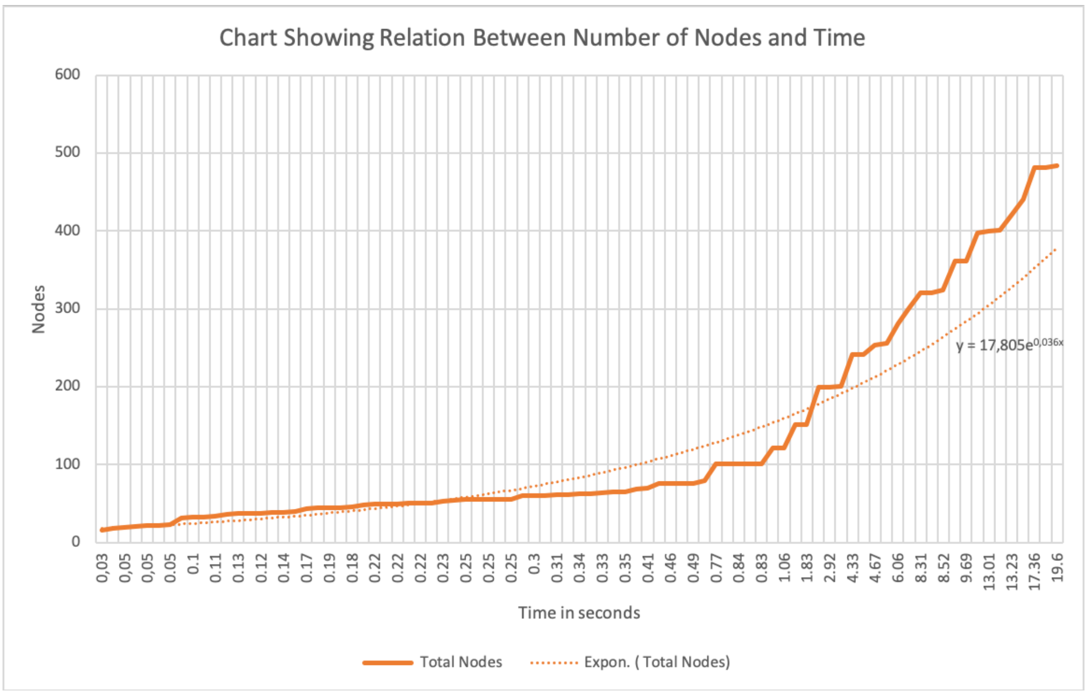
```
 1: localBest ← solution of NearestNeightbour
 2: tabuListSolutions ← []
 3: threshold ← $n$
 4: thresholdCounter ← 0
 5: stoppingCriteria ← 0
 6: function TabuSearch($instance, timeLimit$)
 7:     keepRunning ← $True$
 8:     while $keepRunning$ do
 9:         if time is up then
10:             return localBest
11:         tabuListEdges ← []
12:         for route in localBest.routes do
13:             if capacity not exceeded and edges not in tabuListEdges then
14:                 $candidate ← twoOpt(route)$
15:             if candidate not in tabuListSolution then
16:                 append tabuListSolution ← $candidate$
17:                 if cost of candidate < cost of localBest then
18:                     $localbest = candidate$
19:                     append candidate to tabu list
20:                 else
21:                     thresholdCounter ← +1
22:                     if thresholdCounter > threshold then
23:                         force swap or make worse move
24:                         thresholdCounter ← 0
25:                         stoppingCriteria ← +1
26:                         if stoppingCriteria > number to stop at then
27:                             return localBest
```

then increase the threshold counter. This counter is defined to have a condition on when to create a worsening move, which can be done by forcing the candidate as solution or make a random move. The last criteria is the stopping criteria. In this project, the stopping criteria is when enough worse moves have been made. When this condition is exceeded, then we return the latest localBest.
The second version of the Tabu Search algorithm is the exact same, but does swapping on two routes instead of the one route assigned to a vehicle.

3 Analysis

The image 1 shows the analysis of the Tabu Search algorithm. The difference from the algorithm from the pseudocode is the actual implementation has been done in such way, when the stopping criteria is met, the first version of Tabu Searh starts an executation of the second version of Tabu Search. This is to get a bit more diversity in the search space. Tabu Search version 1 (one route is being examined) has the problem with being stuck in a local optimum for the route assigned to a vehicle. To try and get around it, we use the second version to compare the local optimum of a route with another to see if we can make an improvement.
The idea seems good at first sight, but as the the analysis show, we are getting exponetial growth. The running time is: $O(logn)+O(n^2)$, which is Nearest Neightbour + TwoOpt. When the number of nodes are below 100, it performs acceptable, but after 100 nodes the increase in time flies off. An assumption to why this happens in the amount of compares it has to do with comparing all routes with each other to find a better solution.

Figure 1: Algorithm analysis of Tabu Search. $X = Time$, $Y = Nodes$



An idea of avoiding using a second Tabu Search to minimize the cost of the routes could be to use Nearest Neighbour with randomization to get a bit more diversity in the solutions.

4 Ending Note

When looking into information about Tabu Search, TSP is mentioned. Since CVRP is a subset of TSP, an intuition is, it should work nicely on CVRP as well. But it does have it challenges with CVRP. This might be due to the selection of the heuristic and local search algorithms of this project, but nevertheless it struggles. If we look at the TSP, the swapping and short term memory storage is smart with one big route to avoid messing up with a possible good solution, but when having multiple routes with capacity constraints, the short term memory does not offer a lot of improvements, and often does the single route in a local optimum, which is hard to escape. An issue with the heuristic chosen in this project was: routes with 1 node. Those routes made it difficult with swapping. To try and accommodate, a route with one node was added to the route in the solution with the most capacity left (only if total capacity was not exceeded the maximum allowed). It created a risk of a route having its cost (length of the route) exploding, but this was worth it, since a whole vehicle could be left out (gasoline prices). The threshold and length of the tabu list have been played around with, and the results did not come out better. Even though, the algorithm was allowed to run for a long time. My conclusion is: if Tabu Search is to be used in CVRP, a deeper investigation have to be made of the heuristic to be used as starting point and a local search algorithm to improve the solution. The ones picked for this project did not work as well as expected.

References

[1] Jason Brownlee. Clever algorithms: Nature-inspired programming recipes.

[2] Tavish Srivastava and Tavish Srivastava. K nearest neighbor: Knn algorithm: Knn in python & r, Mar 2018.

[3] Mikko Venhuis. Around the world in 90.414 kilometers, Apr 2019.

Appendices

A Tabu Search Results

The Comma Separated Results of Tabu Search can be found in the file "TabuSearch.csv" under the folder Doc. This has been done, since LaTeX have a hard time structuring it well into a PDF with so many results - at least in my experience