

TITOLO DA DECIDERE

Matteo Moratello
Christian Obetti
Marco Barison

Padua, June 24, 2020

Chapter 1

Introduction

Qui un intro sul problema TSP direi

Chapter 2

Compact Models

Qui la parte dei modelli compatti

Chapter 3

Exact Algorithms

qui la parte degli algoritmi esatti

Chapter 4

Matheuristic

Qui la parte dei matheuristic (hard fixing e local branching)

Chapter 5

Stand-alone heuristics

On this chapter we will illustrate some stand-alone heuristics. These algorithms are so called because they provide an heuristic solution without relying on CPLEX or any other solver. The literature in this case is huge: there are plenty of different approaches and each of them can have multiple variations. On this report we will cover some of the major ideas.

5.1 ?

5.1.1 Nearest neighborhood

Nearest neighborhood is a greedy algorithm ¹ that works in the following way: start from a random point that is considered the first visited node. Then pick as next node the nearest to the last visited (greedy choice) and iterate until all the nodes are visited. This procedure can be repeated until a time limit is reached, each time starting from a different random point, keeping in memory the best solution and eventually update it if a new one is found. The algorithm is pretty simple but provide an admissible solution (that of course is not the best one and in the very most of the cases is quite far from the optimum) in a short time.

On our implementation we decided to trade space for time. Since the opera-

¹algorithm that always makes the choice that looks best at the moment. That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.

tion to compute the nearest point is repeated several time, we decide to store for each point the list of all others points sorted by their distances from that one. This require $O(n^2)$ additional space. Considering the time, instead, to sort the points we implemented a simple *insertion sort*, that is $O(n^2)$, so in total we need an initial $O(n^3)$ (This part could be further improved with a better sort algorithm like *merge sort* that is $O(n \cdot \log(n))$, but since the initial time to order the nodes on our experiment was negligible, we decided to use insertion sort that was easier to implement). So, by paying this initial cost we were able to speed up each iteration of nearest neighborhood, in fact each time we needed to find the nearest point we just had to pick the first available one from the ordered list. Of course in the worst case this is $O(n)$, because it can happen that the firsts elements of the ordered list have already been visited, so I have to run across most of the list until I find an available point. However, what we saw in practice is that, except when there are left few nodes to visit, the first element of the list is also available, so the operation to find the nearest neighbor became $\Theta(1)$ in most of the cases.

The major problem of this technique is that visiting always the next closest point leads to the creation of lots of intersection between edges, that of course are inefficient. A possible solution to improve this initial result will be shown on the next section dedicated to the refining algorithms.

Another problem is that this algorithm is deterministic, so this mean that if it is run twice starting from the same initial point, it produce the same solution. The consequence of this is that the number of different circuit that it can explore is limited to the total number of nodes.

5.1.2 GRASP

GRASP is a variation of nearest neighborhood that introduce a random component. On each iteration, rather than picking the nearest point to the last visited, select the three closest points and choose one of them at random as next visited vertex.

Our implementation was almost the same of nearest neighborhood, with the construction for each point of the initial list of all other points sorted by their distance from that point, with the only difference that first three available points are picked from the list, rather than only one.

The introduction of the random component means that the probability to repeat the same sequence starting from the same point is very minimal, so

GRASP can explore lots of different circuit resolving the problem of determinism of nearest neighborhood.

5.1.3 Insertion heuristic

This algorithm starts from a simple circuit made with only two points. At each iteration, for each of the remaining vertices to insert, compute the minimal extra mileage. The extra mileage is the cost to insert a point on the circuit. So, for example, if we want to insert the vertex z between vertex x and vertex y , the extra mileage is: $c_{xz} + c_{zy} - c_{xy}$. The fact that for each point we search the minimal extra mileage means that we are looking to the best position where insert that point. Insert vertex that has the best extra mileage into the partial circuit and repeat the procedure until all the points are inserted. The algorithm than can be executed several time, starting every time from different initial points to explore new solutions and updating the incumbent if necessary.

The cost to compute the best extra mileage is $O(n^2)$, so this mean that the entire algorithm is $O(n^3)$.

It is also possible to add the GRASP rule, so rather then selecting the point with the best extra mileage, find the three smallest extra mileages and choose at random between them, in this way the algorithm doesn't converge to the same solution starting from the same initial circuit.

On our implementation to select the initial circuit we decided to pick at random one point and then to select the furthest point from that one. (Of course this is not the only solution: other possibility are, for example, to pick a vertex and its closest one, or to choose two points completely at random.) In addition, at the beginning of the algorithm, we built a matrix where the element $i\ j$ contains the distance between vertices i and j (So in total we payed an additional $O(n^2)$ in space). We made this choice because on this algorithm distances between points are computed several times and in addition the same distance can be calculated more than once.

5.1.4 Insertion with convex hull

5.2 Refining algorithms

Refining algorithm are so called because they start from an admissible solution that could be provided by any other algorithm and they try to improve it with a series of iteration.

5.2.1 TWO-OPT

The first algorithm we present is TWO-OPT (or 2-OPT).

5.2.2 THREE-OPT

5.3 Metaheuristics

5.3.1 VNS

5.3.2 Tabú search

Tenure = dim tab list

5.3.3 Simulated Annealing

5.3.4 Multi-start

5.3.5 Genetic Algorithm