University of Padua

Department of Information Engineering

Master's degree in

Computer Engineering

Project Report
Ricerca Operativa II

Davide Serratore M. 1207660
Omar Vesco M. 1197699

Academic Year 2019/2020

Contents

1 Introduction

The work presented in this report was developed during Ricerca Operativa II course and aims to propose and analyze several approach to solve the TSP problem. We studied two main different approaches to solve this type of problem:

- Exact resolution approach
- Heuristic approach

In the first case we use ILOG CPLEX, an optimization software package with a student commercial license provided by IBM. The goal is to return an optimal solution of the problem. In the second case we present the implementation of different algorithms found in the literature that permit to obtain an approximated solution for the problem. We'll see how to use CPLEX for approximated solution and, later, how to compute a good solution without it.
Moreover, in the first part of the report we present the problem and then we introduce in details the approaches and the results obtained from some instances provided by TSPLIB website [1].

2 TSP problem

Given a collection of cities and the links between them with relative costs, the Travelling Salesman Problem, or TSP, is to find the cheapest path that visits all the cities once and returns to the starting point [4]. This path is called tour. There is the possibility to study two main types of instances: symmetric and asymmetric, where the differences are on the definition of distances.
In the first one the costs between cities don't depend on the direction, given the possibility in a practical implementation to store only $\frac{n(n-1)}{2}$ edges.

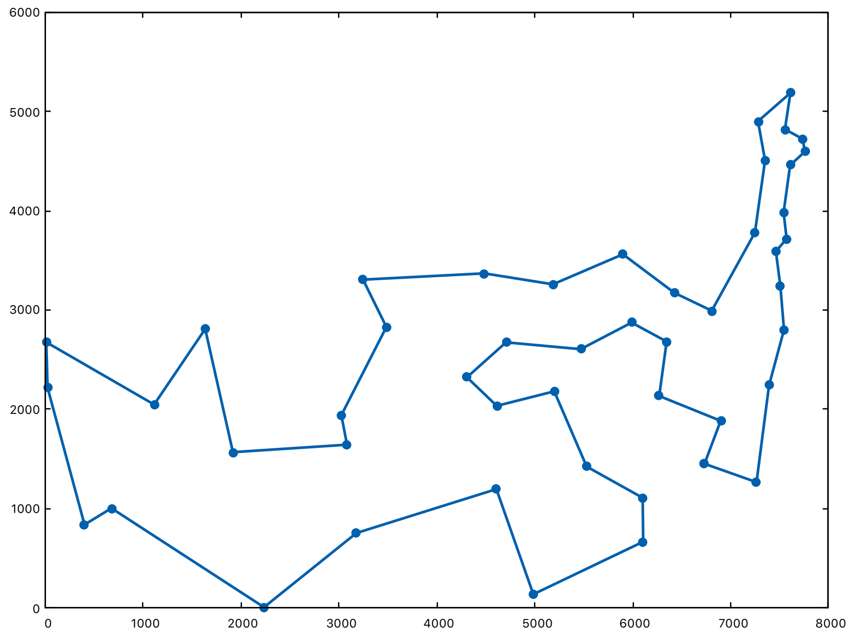

Figure 1: Solution for a TSP problem

Let V be the set of nodes, E the set of edges between nodes and $\delta(v)$ the set of edges linked to the node v. We can easily obtain the mathematical model for its symmetric formulation:

$$\begin{aligned}
\text{Minimize} \quad & \sum_{e \in E} c_e x_e \\
\text{subject to} \quad & \sum_{e \in \delta(v)} x_e = 2 && \forall v \in V \\
& \sum_{e \in E(S)} x_e \leq |S| - 1 && \forall S \subset V : |S| \geq 3 \\
& x_e \in \{0, 1\}, \forall e \in E
\end{aligned}$$

The objective function is the sum of the costs of the selected arcs. For each node there must be exactly one incoming selected arc and one outgoing.
With these constraints we visit every node once but solutions consisting of more than one subtour would be accepted. The last constraints discard all solutions containing subtours since they forbid the number of selected arcs within subset S of nodes to be equal or larger than the number of nodes in S. These are called **subtour elimination** constraints.
The asymmetric formulation can be defined as follow:

$$\begin{aligned}
\text{Minimize} \quad & \sum_{(i,j)\in A} c_{ij}x_{ij} \\
\text{subject to} \quad & \sum_{(i,j)\in\delta^-(j)} x_{ij} = 1 && \forall j \in V \\
& \sum_{(i,j)\in\delta^+(i)} x_{ij} = 1 && \forall i \in V \\
& \sum_{(i,j)\in\delta^+(S)} x_{ij} \geq 1 && S \subset V : 1 \in S \\
& x_{ij} \in \{0,1\}, (i,j) \in A
\end{aligned}$$

where c_{ij} is the cost to go from node i to node j, V the set of nodes, A the set of edges, $\delta^+(v)$ and $\delta^-(v)$ respectively the set of edges outgoing from v and incoming to v. To implement our algorithms we use mainly the asymmetric definition. However the compact models, as we will see, work with directed graph: in these cases we consider $c_{ij} = c_{ji}$

From the literature we know that this problem can't be solved in polynomial time.

3 Exact resolution

These methods use CPLEX to obtain the optimal solution for the model. In practice, the main problem for the solver is to manage the exponential number of subtour elimination (SEL) constraints. That's why the best approach is to define them in part only when needed.
For the first algorithms ("Loop" and "Lazy callback") we start therefore from a basic version of the problem that doesn't include them and we correct the intermediate solutions (obtained during the computation) re-solving the model, augmented with the constraints for the subtours found.
The basic model is:

$$\begin{aligned} \text{Minimize} \quad & \sum_{e\in E} c_e x_e \\ \text{subject to} \quad & \sum_{e\in\delta(v)} x_e = 2 \qquad \forall v \in V \\ & x_e \in \{0,1\}, \forall e \in E \end{aligned}$$

Then we explore the possibility to define subtour constraints with different mathematical equations. In these cases we start with a basic model based on directed graph that is the follow:

$$\begin{aligned} \text{Minimize} \quad & \sum_{(i,j)\in A} c_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{(i,j)\in\delta^-(j)} x_{ij} = 1 \qquad \forall j \in V \\ & \sum_{(i,j)\in\delta^+(i)} x_{ij} = 1 \qquad \forall i \in V \\ & x_{ij} \in \{0,1\}, (i,j) \in A \end{aligned}$$

3.1 Loop

In this approach we compute the optimal solution looping and solving the TSP model many times. After each step we verify if the solution is a set of subtours and, if so, we modify the problem adding the appropriate subtour elimination constraints. Then, we iterate solving the new problem with CPLEX until there is an unique tour that represents the solution.

Algorithm 1: Loop Algorithm

```
$build\_model()$;
$ncomp \leftarrow \infty$;
while $ncomp \geq 2$ do
    $CPLEX \rightarrow x^* \rightarrow ncomp$;
    if $ncomp \geq 2$ then
        $k \leftarrow 1$;
        while $k \leq ncomp$ do
            Add SELs to component $S_k$;
            $k \leftarrow k + 1$;
        end
    end
end
```

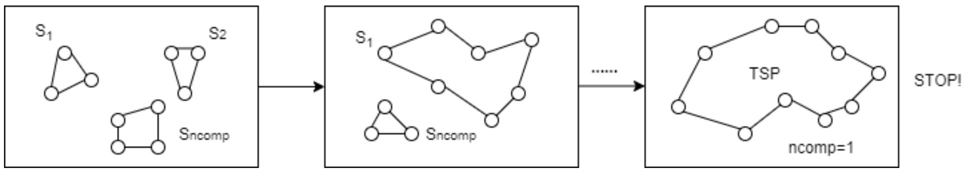

Figure 2: Loop Algorithm

3.2 Lazy callback

In this approach we use CPLEX callbacks that permit to call a user-written function for adding the so called "lazy constraints".
The user-written callback is invoked when CPLEX compares an integer-feasible solution to lazy constraints. If lazy constraints have been added, the current solution is rejected, the subproblem is resolved and evaluated and, if the LP solution is still integer feasible and not cut off, the lazy constraint callback is called again. Our lazy constraints are represented by the subtour elimination.

3.3 Sequential Formulation - Miller, Tucker and Zemlin (1960)

The goal of MTZ [6, 7] is to define subtour elimination constraints in a compact way. The trade-off of the model is to consider the instances as a directed graph, so we need to insert each edge twice in the problem formulation. Then, starting from the asymmetric TSP basic model, we need:

$$0 \leq u_i \leq n-2 \quad \forall i \in V \setminus \{1\}, \quad u_1 = 0 \tag{1}$$

$$u_j \geq u_i + 1 - M(1 - x_{ij}) \quad \forall i, j \in V \setminus \{1\}, \quad i \neq j \tag{2}$$

$$\text{with } M >> 0 \tag{3}$$

We add to the model a set of new variables u_i (sequence in which city i is visited, $i \neq 1$) and we impose the enumeration of nodes with these variables from 0 to $n-2$. With the second set of constraints we impose $x_{ij} = 1 \Rightarrow u_j \geq u_i + 1$: if $x_{ij} = 1$ the constraint is activated and the correspondents variables u_j and u_i are properly set.
Instead, if $x_{ij} = 0$, the constraint is deactivated and u_j is free.
The result is that every subset, e.g. a mini tour, is discarded and the unique legit tour is the biggest tour that includes all the nodes enumerated with the logic defined by the constraints. In practice we set $M = n$ obtaining

$$u_i - u_j + nx_{ij} \leq n - 1 \tag{4}$$

For this formulation we also tried to define this constraints as lazy to CPLEX.

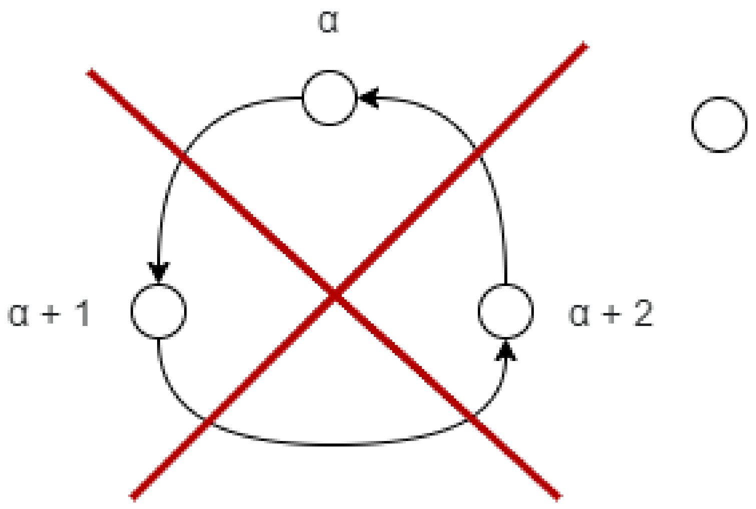

Figure 3: MTZ: forbidden subtour

3.4 Single Commodity Flow Formulation - Gavish and Graves (1978)

This formulation has the same goal and basis formulation problem of MTZ but use different subtour elimination constraints. We introduce some variables y_{ij} that represent an hypothetical flow on the (i, j) edges.

The constraints that augment our model are:

$$0 \leq y_{ij} \leq (n-1)x_{ij} \quad \forall i, j \in V, i \neq j \tag{5}$$

$$\sum_{j \in V, j \neq 1} y_{1j} = n - 1 \tag{6}$$

$$\sum_{j \in V, j \neq h} y_{hj} = \sum_{i \in V, i \neq h} y_{ih} - 1 \quad \forall h \in V \setminus \{1\} \tag{7}$$

$$y_{ii} = 0 \quad \forall i \in V \tag{8}$$

$$y_{i1} = 0 \quad \forall i \in V \tag{9}$$

The outgoing flow from the first node must be $n-1$. With the constraint (7) we impose that for each node $h \in V \setminus \{1\}$ the outgoing flow must be equal to the incoming flow minus 1. The incoming flow for the first node must be 0. The first coupling constraint permits the connection between the two sets of variables x_{ij} and y_{ij}: if $x_{ij} = 0$ then y_{ij} is set to 0, if $x_{ij} = 1$ then y_{ij} is free to assume a value between 0 and $n-1$.
These constraints permit to discard the subtours since it's not possible to assign legal values to variables y_i for them.
We tried to add this constraints as lazy to CPLEX in this formulation too.

4 Approximate resolution

Even if some exact resolution studied get good time performances, the problem is still NP-Hard and solving instances with high number of nodes is time consuming. The goal of the next approaches is to solve problems with number of nodes higher than 1000 returning to the user a tour that is not necessarily optimum. We'll see algorithms that both use CPLEX solver or not.

4.1 Hard-Fixing Approach

This algorithm is based on CPLEX solver. The main idea is that we want to extends our best exact approach in order to solve instances with many nodes by manipulating the solution returned after a local time limit (10 minutes in our case). In particular we want to get the problem easier fixing some of the variables that are set to 1 in the current solution (not necessarily the optimum). At each iteration, if edges variables have values equal to 1, we fix every one in the problem with some probability and resolve the new formulation. If we don't get improvement respect to the previous one, the probability value decreases until is 0. This approach needs an initial solution: at the beginning we give a local optimal solution returned by 2-Opt algorithm with Grasp initialization to CPLEX that try to improve it without fixing edges. Giving the first solution through some algorithm is very useful in practice because the solver might not find an initial solution in a short time.

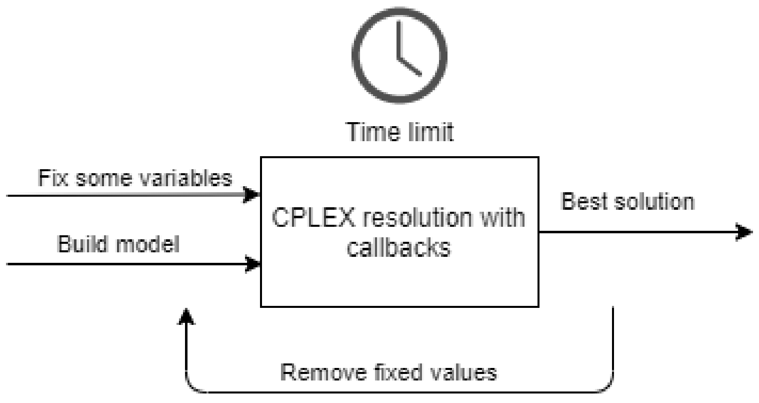

Figure 4: Hard-Fixing

4.2 Soft-Fixing Approach

This algorithm is based on CPLEX solver. It has the same main idea of Hard-Fixing approach with the differences that we let CPLEX decide which of them to fix. Let S be the set of edges' variables that are 1 in the current solution and n the number of nodes. The new rule to be added at each iteration is:

$$\sum_{x_e \in S} x_e \geq \alpha n \quad \text{with} \quad 0 \leq \alpha \leq 1 \tag{10}$$

where α is the fixing probability value. It means that at least αn variables of the previous solution must be equal to one.
Alternatively a similar equation that we use in our implementation is the follow:

$$\sum_{x_e \in S} x_e \geq n - k \quad \text{with} \quad k < n \tag{11}$$

It means that at least $n - k$ variables of the previous solution must be equal to one. We start with $k = 2$ and we increase its value every time the solution doesn't improve respect to the previous one. We also use the same initialization proposed in Hard-Fixing algorithm.

4.3 Greedy Initialization

In the different algorithms studied it is useful to start with some solution. To do so we try two different initialization:

- Nearest Neighborhood (NN) [5]
- Random GRASP (RG)

For the first one, starting from all different nodes, at each iteration we choose the closest node not yet visited as successor of the current node, until all nodes are selected. The best path found is the starting node solution for the next algorithms.

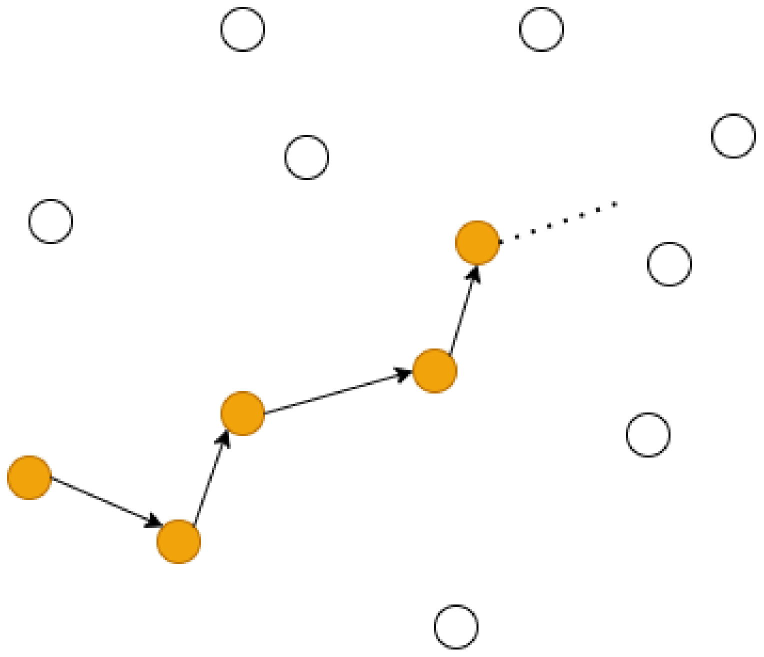

Figure 5: NN: we iteratively choose the closest node

The second one uses the same approach of NN with some differences: we select 3 nodes closest to the current one not visited yet and we choose at random one as the next node or the closest one with some probability. Also in this case the algorithm starts from different nodes selected at random: the best solution found is the starting node for local search approach.

Once the initialization is done, we can try to minimize path cost with:

- 2-Opt
- Variable Neighborhood Search (VNS)
- Simulated Annealing (SA)

The first one is used in every heuristic algorithm proposed to find an initial solution to optimize. VNS and SA are algorithms not based on CPLEX solver.

4.4 2-Opt Algorithm - Croes (1958)

2-Opt [5] is a local search algorithm which aim is to optimize as much as possible a solution. So, starting from a tour, we select two edges and reconnect associated nodes using a different pattern. After this, we compute the new tour distance and, if this modification gets a shorter tour, it becomes a candidate to update the current tour; the best path found is selected at each iteration. We iterate until there are no improvements.
To optimize the computation we can just verify cost tour's variation defining Δ as $(c_3 + c_4) - (c_1 + c_2)$ where c_1 and c_2 are the old edges, while c_3 and c_4 are the new ones. If $\Delta < 0$ this choice leads to improvements.
The main problem of this algorithm is that the solution is easily a local optimum. That's why we try to improve our solution using different approaches.

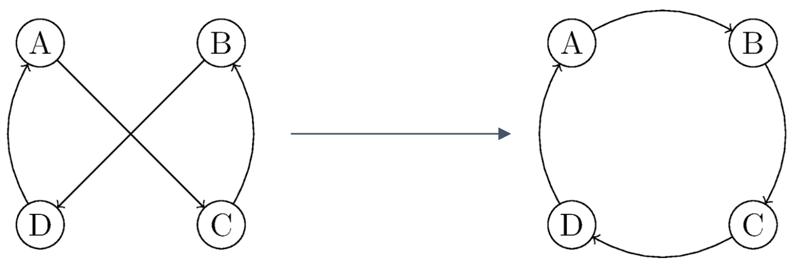

Figure 6: Example of 2-Opt move

4.5 Variable Neighborhood Search - Mladenović and Hansen (1997)

The VNS is a metaheuristic method that acts alternating two phases:

- Intensification
- Diversification

In the first phase we initialize a solution using a Greedy algorithm shown before and we apply 2-Opt algorithm to obtain a local optimum. Then, to escape from

it, we apply diversification: we randomly select some edges and we reconnect the associated nodes (opt move) using a different pattern, similarly to 2-Opt algorithm. This operation perturbs the current solution and, hopefully, permits to escape from a local optimum applying a new 2-Opt algorithm.
We implemented different opt move, from 4 to 6. The main idea that we used to do this operation is to split the tour in 5,6 or 7 part according to the edges chosen and reconnect them using a different pattern. Starting from a 4-Opt move, we decide to switch to another after a defined number of failures that are represented by a non escaping local optimum: an opt move and a 2-opt that falls in the current local optimum or a tour with a worse cost.

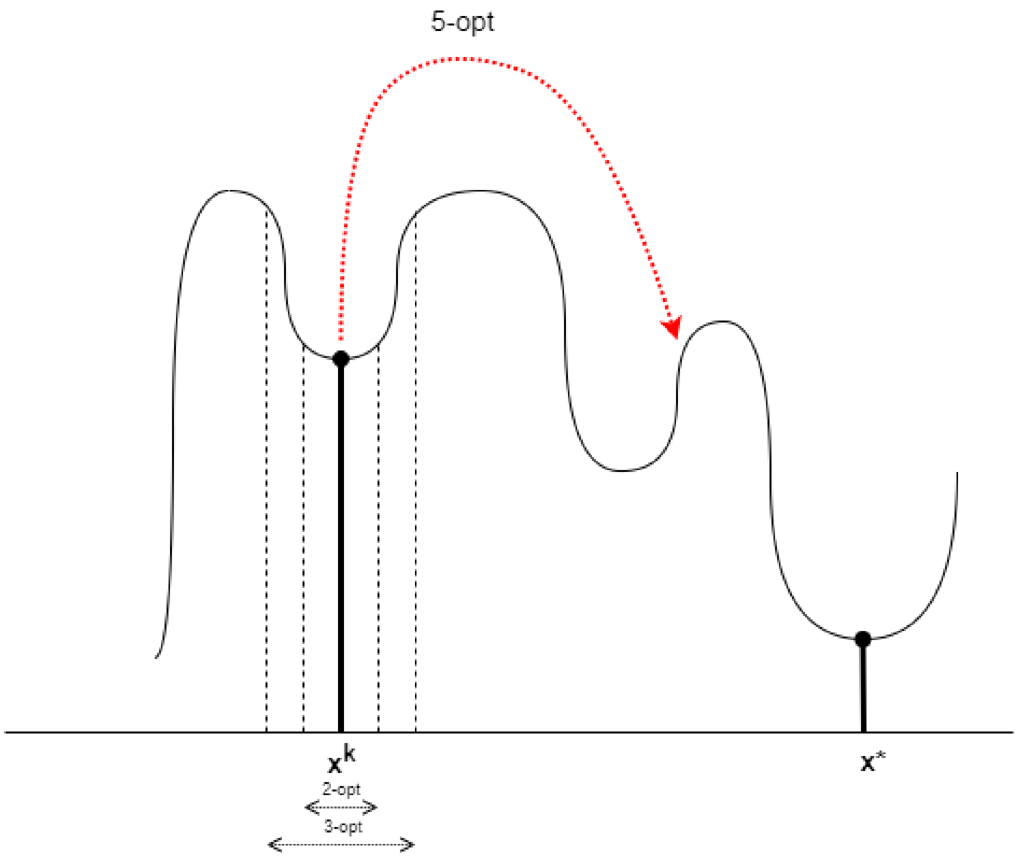

Figure 7: Example of escaping from a local optimum

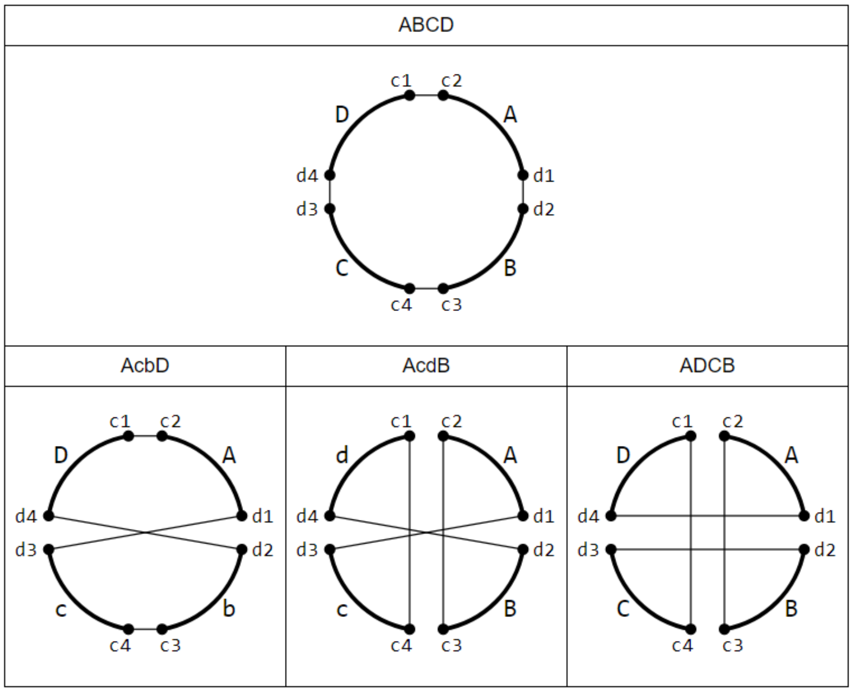

Figure 8: Different 4-opt moves. Source: TSP Basics

We repeat the procedure until iterations defined by the user end.

Algorithm 2: VNS Algorithm

```
$tour \leftarrow getInitialSolution()$;
$maxIter \leftarrow setMaxIteration()$;
$iteration \leftarrow 0$;
while $iteration < maxIter$ do
    $newTour \leftarrow perturbSolution(tour)$;
    if $cost(newTour) < cost(tour)$ then
        $tour \leftarrow newTour$;
    end
    $iteration \leftarrow iterations + 1$;
end
```

4.6 Simulated Annealing

Simulated Annealing [2] is inspired by annealing in metallurgy. The fundamental idea is to allow moves to solutions with objective function values that are worse than the current values. At the beginning we start with high temperature and after some time we decrease it until it's close to zero. At each iteration we randomly perturb the solution: if its value increases we always accept, otherwise we accept with some probability (Boltzmann Distribution) depending on temperature and cost function. In detail the function is $\exp(-\frac{\Delta}{T})$, where Δ is the cost tour's variation of the possible perturbation ($\Delta < 0$ means we are improving the solution) and T is the current temperature. There are different way to implement temperature behaviour. In this algorithm a simple linear function is used: $T_1 = \alpha T_0$ where α is a constant less than 1 and T_0 the previous temperature. User can set an execution time that will be equidistributed over the different phases. Once the temperature is too low, algorithm should accept every move. That's why at the end we use 2-Opt approach to compute an optimal solution.

Algorithm 3: SA Algorithm

```
$tour \leftarrow getInitialSolution()$;
$timeLimit \leftarrow setMaxTime()$;
$innerTime \leftarrow setInnerTime()$;
$T \leftarrow setInitialTemperature()$;
$T_{min} \leftarrow setMinTemperature()$;
while $T > T_{min}$ do
    $time \leftarrow 0$;
    while $time < innerTime$ do
        $newTour \leftarrow perturbSolution(tour)$;
        if $cost(newTour) < cost(tour)$ then
            $tour \leftarrow newTour$;
        else
            update tour with newTour with probability P()
        end
        $time \leftarrow updateTime()$;
    end
    $T \leftarrow adaptTemperature()$;
end
$tour \leftarrow Opt\_2(tour)$;
```

5 Performance Analysis

To test our algorithm we use some instances of TSPLIB where the distances between nodes are computed using euclidean 2D geometry. Working with ILOG CPLEX and approaches based on randomness, we also try different random seeds. All the tests are evaluated with seed 1995, 15, 16 and no seed defined. For the testing we use:

- berlin52
- pr76
- rat99
- bier127
- kroA200
- a280
- d493
- rat575
- d657
- vm1084
- nrw1379
- pr2392

For the analysis we also use the so-called Performance Profile [3], in which the Y-axis represents the winning percentage and the X-axis the ratio deviation from the winner (in terms of computation time) for a specific instance (performance acceptability). This permits to obtain a better description for the behaviour of the algorithms. For the exact models we decide to use instances with 700 as maximum number of nodes. To evaluate heuristic approach we use instances between 1000 and 2000 number of nodes.

5.1 Exact solver

In Figure 9 we can see a comparison between every algorithms that solve the problem returning the optimal solution. Even if compact models reduce the subtour elimination constraints, as expected, the formulation is not enough to get better time results related to Loop and lazy callback.

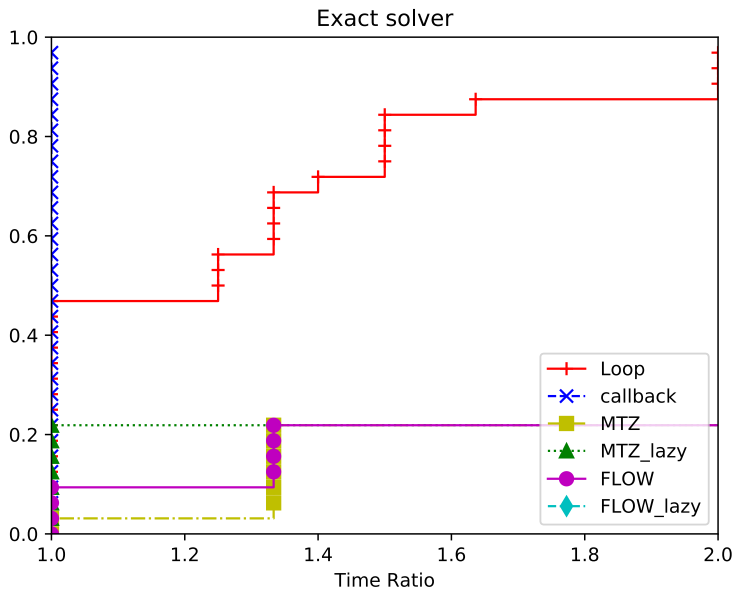

Figure 9: Exact solver Comparison

Anyway we try to optimize as much as possible time execution using lazy constraints for subtour eliminations, since we notice better results in the standard formulation. We can see the result in detail in figure 10. With this approach we didn't achieve improvements on both the models. In particular, model FLOW1 gets better results using no lazy. That's not happen for MTZ model that perform as the best related to the compact ones. Anyway, we limit the tests for this type of solver to instances with a maximum number of nodes equal to 200 because every model often ends with time limit not finding an optimal solution.

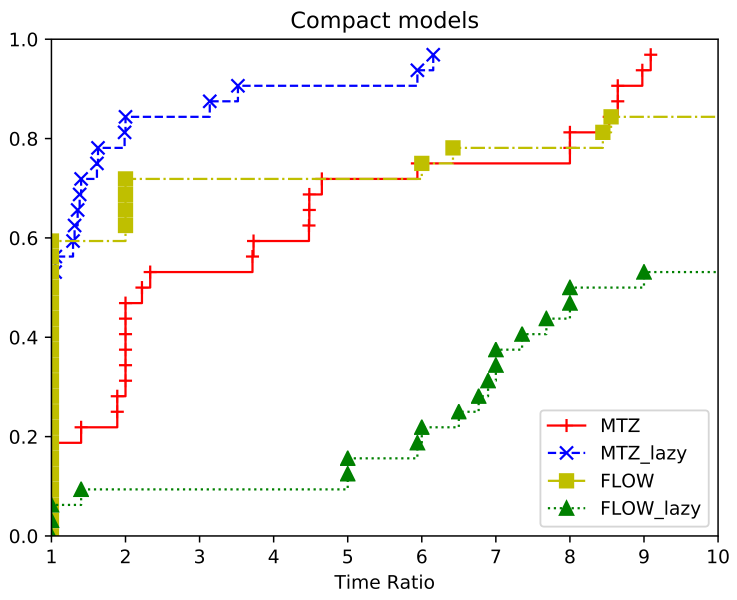

Figure 10: Detailed comparison between compact models

Using Loop and Lazy callback, CPLEX solver obtain our best results. For these types of model we extend tests using instances with a maximum of 700 number of nodes in order to understand better the behaviour of these two approaches. As we can see in figure 11 lazy callback approach is the one with best results. The reason can be related on how ILOG CPLEX decides when and how to check the presence of subtour in the solution. We also don't waste time calling many times the solver that probably needs to do some initialization.

Instance	NN initialization	GRASP initialization	Optimal solution
berlin52	7713,034759	7749,880842	7542
a280	2696,322022	2710,701832	2579
vm1084	251868,205847	254381,911066	239297
pr2392	401283,545938	401618,718999	378032

Table 1: 2-Opt Solution with different initialization

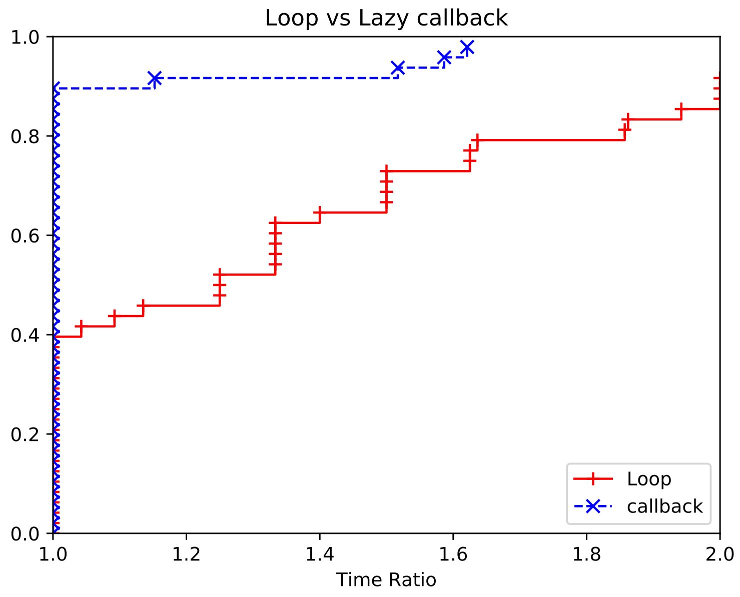

Figure 11: Detailed comparison between Loop and lazy callback

5.2 Approximation solver

In this section we report in the different tables the best result obtained with different executions.
The result returned by 2-Opt algorithm are in table 1. As we can see Nearest Neighborhood initialization gets better solution than random initialization using Random Grasp and, as expected, we find a local optimum.
VNS results are shown in tables 2 and 3. Increasing the number of iterations (user input) leads to increase computation time. For instances with many nodes this could be a problem, that's why for "pr2392" we reduced the number of iterations. This algorithm obtains the best results in the instances proposed compared to heuristic algorithm not based on ILOG CPLEX, getting tour cost close to the optimal solutions. Increasing the number of iterations doesn't seem to lead to improvements.

Instance	10000 iterations	20000 iterations	30000 iterations
berlin52	7598,442341	7544,365902	7544,365902
a280	2593,554662	2608,496278	2586,769648
vm1084	241695,09037	242752,583767	241254,550924

Instance	1000 iterations	2000 iterations	3000 iterations
pr23921	397505,347158	395682,7485	395861,9671

Table 2: Variable Neighborhood Search Solution with Nearest Neighborhood initialization

Instance	10000 iterations	20000 iterations	30000 iterations
berlin52	7544,365902	7544,365902	7544,365902
a280	2612,11621	2588,658191	2611,423965
vm1084	243072,106929	240846,051024	242839,044261

Instance	1000 iterations	2000 iterations	3000 iterations
pr23921	397070,502052	396419,24235	397389,169301

Table 3: Variable Neighborhood Search Solution with random GRASP initialization

SA results are shown in tables 4 and 5. We tried different combination for the parameters but in the final test we decided to use these values:

- $\alpha = 0.999$
- $T_0 = 100$
- $T_{minimum} = 0.001$. Once we get this temperature, the algorithm ends calling 2-Opt algorithm.

This algorithm seems to be the worst one tested. Higher times execution doesn't get always better solutions and, even if NN gets better cost on tours, sometimes SA obtains better results using random initialization. Moreover, this implementation is often worse than 2-Opt that requires less time to return a tour. Hard-Fixing and Soft-Fixing are compared in figure 12. For the final tests:

- Hard-Fixing: we start with fixing probability $p = 0.90$ and we decrease its value of 0.20 until we get $p < 0$

Instance	120 sec.	300 sec.	600 sec.
berlin52	8061,065372	8056,643669	8127,067017
a280	2902,531378	2815,435516	2878,506388
vm1084	260430,284846	257365,720032	261542,933843
pr2392	420000,824281	421999,935624	421308,402058

Table 4: Simulated Annealing Solution with Nearest Neighborhood initialization

Instance	120 sec.	300 sec.	600 sec.
berlin52	7854,786437	7892,888087	8018,219082
a280	2852,648067	2899,664779	2820,88199
vm1084	259150,508727	256295,133794	262140,137431
pr2392	421844,824054	423060,671927	425227,078704

Table 5: Simulated Annealing Solution with random GRASP initialization

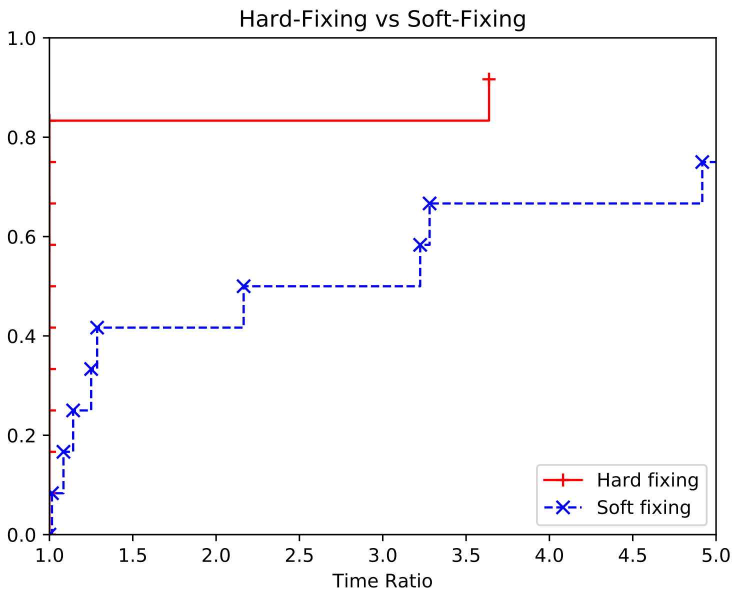

Figure 12: Caption

- Soft-Fixing: we start with $k = 2$ and we increase its value of 2 until we get $k \geq$ "number of nodes"

As we can see, even if Soft-Fixing gets good solutions, Hard-Fixing approach performs better.

The latest comparisons we want to show, are between CPLEX-based heuristics and VNS approach.

As we can see in table 6, we achieve better results using CPLEX. Moreover, the best heuristic approach we implemented is Hard-Fixing (also related to Figure 12). Anyway the results of VNS are very promising (also related to tables 2 and 3), so this implementation could be a good trade-off if we don't want to buy any software-license.

	vm1084	nrw1379	pr2392
Hard-Fixing	240216,02856	56669,38722	387303,89281
Soft-Fixing	241872,57182	57384,44466	395561,62379
VNS	240846,05102	57665,87324	395682,74849

Table 6: Comparison between Hard-Fixing, Soft-Fixing and VNS

6 Conclusion

After implementing several algorithms for the resolution of the TSP problem, we tested their performance using some instances of the TSPLIB. At first we experimented different approaches to find the optimal solution: we have found out that smart way to define subtour elimination aren't good anymore, since it's better to face the problem adding the violated constraints when needed. Using lazy constraints achieved our best results even if loop approach was very useful to understand and see how these implementations work.
Since the problem is NP-hard, in the second part we studied and analyzed approximation algorithms. CPLEX-based approaches got our best results. Moreover, VNS get good solutions to the problems, so we can consider it as a trade-off in case we don't want to buy any license. Future improvements would consist in the implementation of greater opts to escape from the local optimum, since we only implemented move from 4-opt to 6-opt. 2-Opt with NN or random GRASP helped our algorithms a lot to initialize a good solution thanks to its speed. It could be useful also if we want a solution as fast as possible instead to find a better one. In final Simulated Annealing, for its nature, performed worse. It would be useful to test different functions for the temperature and see if we can return better tour than the result we get in our tests.

A Appendix

A.1 Data structures and main functions

In this section we present some data structures and functions that we have defined for the implementation of our models.

A.1.1 Instance structure

```
typedef struct {
    //input data
    int nnodes;
    double *xcoord;
    double *ycoord;

    // parameters
    int model_type;
    int randomseed;
    double timelimit;
    int initialization;
    char input_file[1000];

    //heuristic data
    int *nodes_tour;
    double *distance;
} instance;
```

The struct **instance** contains all the useful information for the resolution of our TSP problem: the number of nodes with their coordinates (x,y) divided in two different arrays, plus some parameters such as the model we want to use for the resolution and the *timelimit*, both defined by the user.
If the model is heuristic, we use *nodes_ tour* to store the approximated solution.

A.1.2 Solution structure

```
typedef struct {
     int *successor;
     int *component;
     int ncomp;
} solution;
```

This type of structure permits to store the number of components of the solution and to plot it through *plotData()*.
ncomp is an integer indicating the number of subtour in the current solution, while *successor* and *component* are two arrays useful to define the tour: successor[i] indicates node's index that is after node i, while component[i] indicates which subtour the node "i" belongs to.

A.2 Main CPLEX functions

- *CPXcreateprob()* : the routine creates a CPLEX problem object in the CPLEX environment. The problem created is an LP minimization problem with zero constraints, zero variables and an empty constraint matrix
- *CPXsetintparam()* and *CPXsetdblparam()* : set the value of a CPLEX parameter of type int and double respectively. We use these functions to set the random seed and the time limit
- *CPXmipopt()* : used to find a solution for a problem created with *CPXcreateprob()*
- *CPXgetx()* : gets the solution values for a range of problem variables
- Generic Callback : as described in section 3.2, CPLEX allows callback to some user defined functions. The Generic Callback function is a callback that is invoked in many different places during the search.
  CPXcallbacksetfunc() sets the generic callback function and specifies in which contexts it will be invoked thanks to some constants. We uses this to set our *mycallback* function with context
  CPX_CALLBACKCONTEXT_CANDIDATE for the approach described in section 3.2. CPLEX invokes this generic callback when it has found a new candidate for an integer-feasible solution or has encountered an unbounded relaxation.
  It is possible to reject the candidate solution with *CPXcallbackrejectcandidate()* and optionally specify a number of additional constraints; CPLEX drops the current candidate solution or unbounded direction, considers it infeasible and uses these additional constraints to avoid finding the same point or direction again.

References

[1] {TSPLIB}: a library of sample instances for the tsp (and related problems) from various sources and of various types.

[2] C. Blum, M. Blesa, A. Roli, and M. Sampels. *Hybrid Metaheuristics: An Emerging Approach to Optimization*. 01 2008. ISBN 978-3-540-78294-0. doi: 10.1007/978-3-540-78295-7.

[3] E. D. Dolan and J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[4] M. FISCHETTI. *Lezioni Di Ricerca Operativa*. Independently Published. ISBN 9781980835011.

[5] C. Nilsson. Heuristics for the traveling salesman problem. 01 2003.

[6] A. Orman and H. Williams. A survey of different integer programming formulations of the travelling salesman problem. In E. J. Kontoghiorghes and C. Gatu, editors, *Optimisation, Econometric and Financial Analysis*, pages 91–104, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-36626-3.

[7] T. Sawik. A note on the miller-tucker-zemlin model for the asymmetric traveling salesman problem. *Bulletin of the Polish Academy of Sciences Technical Sciences*, 64, 01 2016. doi: 10.1515/bpasts-2016-0057.