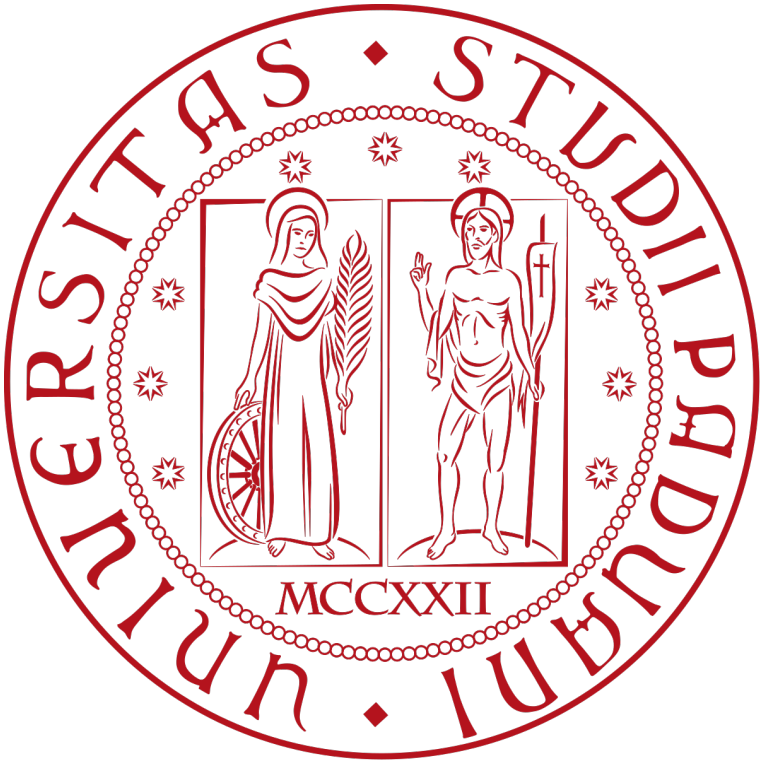
Progress Report

Operational Reseach 2

Thomas Porro

August 2021



School of Computer Engineering

Contents

1 The Problem

In this report we are going to describe, analyze and implement solutions for the Travelling Salesman Problem (from now on it will be called TSP).
Essentially the problem have this type of formulation is the following: "Given a list of cities and the distances between eachother, find the shortest path that connect all the cities". This could be "translated" to find the shortest (or the one with the lowest cost) hamiltonian circuit given an oriented graph $G = (V, A)$, where V are the cities of the problem and A are the paths that connect each city to the other ones.

Matematically the problem is the following. We start numbering all the cities that we have, from now on they will be called nodes, then we introduce a decisional variable x_{ij}

$$x_{ij} = \begin{cases} 1 & \text{if the arc } (i,j) \in A \text{ is chosen in the optimal solution} \\ 0 & \text{otherwise} \end{cases} \tag{1.1}$$

Now we can describe the first formulation of the problem:

$$min \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{1.2a}$$

$$\sum_{(i,j)\delta^-(j)} x_{ij} = 1, \quad j \in V \tag{1.2b}$$

$$\sum_{(i,j)\delta^+(j)} x_{ij} = 1, \quad i \in V \tag{1.2c}$$

$$x_{ij} \geq 0 \text{ intero}, \quad (i,j) \in A \tag{1.2e}$$

In these equations we use the value c_{ij} as the cost of the path from the node i to the node j. The equations 1.2b and 1.2c lead to the fact that each node must have only one arc incoming and one arc outgoing.

2 Code setup

In order to implement the models to be solved we decided to use the common and powerful tool IBM CPLEX. Usually this software isn't free but due the academic usage it was made available for all the students that needed it.
CPLEX allow its user to decide which programming language to use between Python and C; in this project we used C.

To visualize the nodes and the paths found by our program we used a Gnuplot which is a command-line driven utility. Its code is protected byt copyright but the download is completely free. The sofware needs to be installed on the machine where the code is executed because Gnuplot is executed as a pipe: in particular before the plotting all the data is wrote to a file (according to the documentation) and than Gnuplot read and create the plot from that file.

To build the performance profiles in this report we used a python program written by D. Salvagnin (2019).

The first thing we did was build a parser capable of interpreting the TSP problems provided by the TSPLIB. The main data to save was the number of nodes, the coordinates of the nodes (they will be or relative coordinates that describe the position of the nodes or real world coordinates), the type of distance function to use (for example when are used real world coordinates the distance function need to consider the sphericity of the world). For each tsp problem we assume that the datafile contains a complete graph, so each node is directly connected with all the others nodes.

In my particular case all the project was developed on a linux machine with Ubuntu 20.04.

2.1 CPLEX environment

In order to work properly CPLEX needs to build his internal data structure to hold all the information needed to solve the problem. So the first thing to do is to create a pointer to the environment of Cplex through the `CPXopenCPLEX(&error)`: this function will return a pointer to the CPLEX environment that will be needed to use his entire library.
Once the environment is build CPLEX needs an additional data structure to hold the constraints of the optimization

problem we want to solve, in order to use it we build an empty object using the function `CPXcreateprob(env, &error, "TSP")`: this will return a pointer to the problem where we will write all the constraints that we need.

```
CPXENVptr env = CPXopenCPLEX(&error);
CPXLPptr lp = CPXcreateprob(env, &error, "TSP");
```

2.1.1 Variable and constraint creation

Now that the enviroment is setted up we can start by creating our first variable. CPLEX handle the variable and constraints respectivetely as columns and rows; to unserstand it better let's make an example with a simple minization problem:

$$\begin{array}{lrcrcrl} \min & x_1 & + & 3x_2 & + & x_3 & \\ & 2x_1 & & & - & x_3 & \leq 60 \\ & & & 4x_2 & + & 7x_3 & \geq 20 \end{array}$$

$$x_i \text{intero}$$

Thius problem can be seen as a matrix composed by rows and columns, each column correspond to a variable, each row correspond to a constraint that the problem as to satisfy. Using the callable library of CPLEX we can build the rows and columns easily.
To build the variable we need to add a column to the problem, we can to this using the method `CPXnewcols`; this method allow us to create more than a single column per time through his parameters, in our case we built each variable singularly as shown in the code:

```
CPXnewcols(env, lp, 1, &obj, &lb, &ub, &variable_type, cname)
```

Here `env` and `lp` are respectively the pointers toi the environment and to the problem build in the section 2.1. The arguments `&obj, &lb, &ub, &variable_type, cname` are arrays that contains the values for the objective function, lower bound, upper bound, variable type and name for each variable. The last argument to analyze is the number 1, it represent the number of variables to add during this call to the CPLEX's library (as said before in this project we build the variables singularly).

At the end of this phase we have only the variable used in the objective function, now we will discuss how the constraints are implemented in the code. This time the process is more complicated that the previous one, in fact we need first to build an empty constraint, that means that on the left side there are no variables or numbers, than we change the coefficients for each variable. Let's explain it with and example:

```
CPXnewrows(env, lp, 1, &rhs, &sense, NULL, cname)
```

Here `env, lp, 1` and `cname` corresponds at the same variable as in the case of the columns, `&rhs` is the righthand side term for each constraint to be added to the problem object, `&sense` is the sense of each constraint to be added to the problem object, `NULL` contains the range values for the new constraints, for example it sets a maximum/minimum value to each constraint.
At this point we have something like this (example with random values):

$$\text{*empty*} \qquad \leq 30$$

So a part of the constraint is build, we need to decide what variables we want to put in this inequation. In order to do that we use the function `CPXchgcoef`.

Before seeing this method in detail we will explain how CPLEX reference to the rows and columns. In fact the name that we pass to the callable library are not directely used by CPLEX but are needed to the developer to understand if the problem is built correctely; for its function CPLEX simply enumerates each row and each column, so if we want to reference to a specific variable or constraint we need to now its number; by creating variables and constraint we know exactely which is built before and whic after, so for example in the optimization problem shown at the begginning of the section to x_1 is given the number 1, to x_2 the number 2 and so on. The same thing happens to the constraints. Now we can fully understand the method previously introduced.

`CPXchgcoef` uses the already known arguments `env` and `lp`, in addition to them in needs the number of the constraint we want to modify, the number of the variable we want to change the coefficent and the coefficient for that variable. For example to build the first contraint of the already used optimization problem we need two calls to this method since we need to modify two coefficients of the variables:

```
// Here the first number is the id of the constraint
// the second is the id of the variable (1 for x1, 3 for x3)
// and the third one id the value of the coefficents
// (2 for x1 and -1 for x2)
CPXchgcoef(env, lp, 1, 1, 2.0);
CPXchgcoef(env, lp, 1, 3, -1.0);
```

Now that we have introduced the code setup and how the models are built we can start by examinating the variuos methods that we have use to solve the TSP optimization problem.