Università degli Studi di Padova
Dipartimento di Ingegneria dell'Informazione

Corso di laurea magistrale in Ingegneria Informatica

Optimizing the positioning of medical facilities using linear programming techniques

Relatore: ***Prof. Geppino Pucci***

Tutor Aziendale: ***Dott. Federico Sitta***

Studente: ***Andrea Ialenti***
Matricola: ***1056894***

Anno Accademico: 2014 - 2015

If you fail to plan,
you are planning to fail
B.F.

Contents

Contents **ii**

1 The Plant Location Problem **1**
1.1 Simple and Capacitated Plant Location Problems 1
1.2 A survey on SPLP and CPLP 3
1.3 Erlenkotter: A Dual-Based Procedure for the SPLP 4
1.4 Kuehn and Hamburger: A Heuristic Program for Locating Warehouses 6
1.5 SPLP and CPLP Extensions 7
1.5.1 Multi-Commodity PLP 7
1.5.2 Other Plant Location Derived Problems 8
Multi-Echelon Plant Location 8
Multi Period Plant Location 9
Improving the Costs Definition 9
Plant Location and Uncertainities 9

2 Semi-Lagrangean Relaxation-based Algorithm for the SPLP **11**
2.1 Semi-Lagrangean Relaxation 11
2.2 The Dual-Ascent Method 13
2.3 Dual-Ascent Algorithm Applied to the SPLP 15
2.3.1 Properties of the SLR Dual Problem 16
2.4 The Core Subproblem 17

3 MIN-MAX Capacitated and Budgetized Plant Location **21**
3.1 Problem Description 21
3.2 NP-Hardness 23
3.3 The Semi-Lagrangean Relaxation with the New Problem 25
3.4 MMBPLP Feasibility 27
3.5 The Min Capacity Knapsack Problem 27

4 Progressive Plants Selector **31**
4.1 General Idea 31
4.2 Algorithm Description 32
4.3 Algorithm Correctness 34

5 Case Study **39**
5.1 Data . . . 39
5.2 Specifications . . . 40
System's Reactivity . . . 41
5.3 Development Tools . . . 41
5.3.1 MIP Solver . . . 41
Open-source and free solvers . . . 42
Commercial solvers . . . 42
5.3.2 Hadoop . . . 42
Hadoop Data Flow . . . 43
5.4 Oracle Database and Oracle Spatial . . . 46
5.5 System's Architecture . . . 48

6 Computational Results **59**
6.1 Random Instances . . . 60
6.1.1 Problem Hardness . . . 60
Instances Description . . . 60
Results . . . 61
6.1.2 SCIP vs. Progressive Plants Selector . . . 62
6.1.3 Playing with the different factors . . . 62
Users volumes . . . 63
6.2 The Feasibility Check . . . 68
6.3 Different Graphs . . . 71
6.3.1 Facilities Sorrounded by Users . . . 71
6.3.2 Users Sorrounded by Facilities . . . 73
6.3.3 Clustered Users . . . 74
6.4 Real Instances . . . 78

Bibliography **83**

Introuction

The Plant Location Problem is one of the most important branch of operations research concerned with the optimal placement of facilities to minimize transportation costs. The problem is known to be NP-hard [1], and there exists a high number of more complex problems that are based on the SPLP. Because it is widely used in real applications, many studies based on SPLP have been developed and today there exists a big number of exact and heuristic approaches for solving it.

We had to deal with a real problem related to the positioning of medical facilities on the territory of Emilia-Romagna. These facilities are called "Health Houses" and the aim of the Administrators is to use these structures in order to replace or integrate some of the services that are actually provided by big and centralized Hospitals.

We had several potential locations for our Health Houses and a set of users to be served by these plants, consisting in clusters of possible patients; in particular we were dealing with 1500 possible facilities locations and 38000 clusters. Our strategy was based on three steps: first we divided the problem by considering one optimization process for each of the 11 administrative sectors that divide the Region, second we applied a clustering algorithm (k-means) in order to reduce the size of each instance and third we applied the heuristic algorithm we developed on each problem that came out from the two previous steps.

We started from the Simple Plant Location to create a mathematical model for our real problem, but we needed to add some more constraints due to more particular requests: we had a budget limitation and there was a specification on the minimum and the maximum number of users to be served by a facility, to ensure that this can be opened. Moreover we considered three different classes of facilities: small, medium and big.

After we formulated a mixed-integer linear programming model, we tried to solve it by using a general MIP solver to estimate the hardness of the problem. Since it has been proved to be very hard to solve with only the MIP solver, we decided to develop a heuristic algorithm based on Linear Programming. The algorithm we implemented was designed to give to the end-user a rapid feedback from the solving system.

Basically, we used a Semi-Lagrangean realxation of the MIP model in combination with a Dual-Ascent Procedure [2] in order to progressively reduce the

instance size by selecting some of the most convenient facilities; once we fixed some of the plants, we positioned the remaining facilities by using the MIP solver. Once all the structures have been placed, we needed to understand how to assign to each user one facility in order to get the minimum transportation costs, obtaining a solution that was feasible in the sense of the original problem formulation. The idea behind our approach was also based on the studies of Erlenkotter [3] and Kuehn-Hamburger [4]; the first uses a Dual-Ascent Procedure that aims to increase the SPLP dual problem objective function value, while the second tries to progressively guess the best facilities choices in order to find an heuristic solution to the problem.

This procedure required a fast method to check the reduced instance feasibility with respect to the original constraints. We discovered a sufficient condition that was very fast to prove, even if it involved the resolution of two NP-hard problems.

Finally, we compared our algorithm results to those provided by the MIP solver. The values of the solutions found by the heuristic and the execution times have been satisfactory. Most of the time we have got better results from our approach than the MIP solver; obviously giving to the MIP solver an inifinite time limit, it would always give us the optimal solution, so in order to compare the two strategies, we gave to the Solver a time limit that was equal to the time used by the heuristic algorithm to solve the problem.

A further advantage of our method was a shorter "first-feedback-time", i.e. the time interval between the optimization starting instant and the first feedback to the user.

This thesis is organized in two parts. In the first part we describe the Plant Location Problem and its most important derived problems and we show some of the different approaches that were developed so far. Furthermore we explain the approach of [2] for solving the Simple version of the problem, reporting some of the results obtained in their paper that prove the correctness of the method.

The second part first describes the problem that we have derived, showing how it is possible to apply the tecnique of [2] to the mathematical model built around our case study, and second shows the sufficient condition to check the feasibility of the model intances. Moreover, we expose the details of the heuristic approach we have designed and, after a description of the case study on which we have applied all the theory above, we show the goodness of our method by comparing its results with those of a general MIP solver.

In the first Chapter we give an overview on the Uncapacitated and Capacitated Plant Location Problems, explaining how it is possible to model it by using Mixed-integer Linear Programming; we also show a survey on what we have found in literature relatively to the different techniques that were developed for solving it. We also report more details about the Erlenkotter [3] and Kuehn and Hamburger [4] approaches. Our resolution method is based on the approach, developed by [2], which works with the Semi-Lagrangean relaxation of the SPLP and in Chapter 2 we explain in details this method. Moreover we illustrate how it is possible to reduce the difficulty of the Lagrangean Oracle

by removing some of the unnecessary edges from the graph given in input. In Chapter 3 we describe a new derived problem coming from the Capacitated Plant Location that we call *MMBPLP - Min-Max Budgetized Plant Location Problem*. First we analyze the problem itself, showing that it is NP-hard, second we show how it is possible to apply the Semi-Lagrangean Relaxation in order to solve the problem, and finally we provide the sufficient condition useful to understand if an instance of the MMBPLP is feasible or not. Chapter 4 describes the Progressive Plants Selector, the heuristic algorithm we have designed and implemented in order to solve the problem described above. After a detailed description of the algorithm itself, we illustrate that it is correct, in the sense that all the solutions that it provides are feasible. Chapter 5 describes our case study, proving that the model we have formulated in previous Chapters, perfectly fits with our real needs. We describe which development tools we have used and the reasons behind their choice. In the end we provide a detailed description of the overall architecture of the solving system.

Finally, in Chapter 6, we report some computational results on both randomly generated and real instances of the problem we have described in this thesis. In addition we explain how its "hardness" increases or decreases according to different choices of input parameters and different structure of the graph that needs to be optimized.

Chapter 1

The Plant Location Problem

The Simple Plant Location Problem (SPLP) has been studied for many years. It is rooted in the seminal work of Weber (1909), but workable and realistic models and algorithms began to emerge only in mind-1960s with the arrival of automatic computation capabilities [5]. There exists a high number of variants of this problem each of which has a wide range of practical applications, and often the modeling process for a new real location problem starts from the capacitated and the uncapacitated version of the PLP.

1.1 Simple and Capacitated Plant Location Problems

The Simple Plant Location Problem (SPLP) takes as input a set $U = 1, 2, 3, .., n$ of clients, each having a unit demand, a set $F = 1, 2, 3, ..., m$ of sites in which plants can be located, a vector $K = (f_j)$ of fixed costs for setting up plants at sites $j \in F$ and a matrix $C = [c_{i,j}]$ of transportation costs from $i \in U$ to $j \in F$. It computes a set P^*, $\emptyset \subset P^* \subseteq F$ at which plants can be located so that the total cost of satisfying all clients demand is minimal. The costs involved in meeting the clients demand include the fixed costs of setting up plants and the transportation cost of moving the client from its original position to the plant. The SPLP forms the underlying model in several combinatorial problems, like set covering, set partitioning, information retrieval, simplification of logical Boolean expressions and it is a subproblem for various location analysis problems.

In many cases, it is more realistic to incorporete some capacity limitations on the facilities to be established and this new constraint leads to the Capacitated version of the PLP (CPLP). In the CPLP we have the same elements described above for the SPLP, but we need to consider the vector $D = (d_j),\ j \in F$ of the maximal number of users supported by facility j.

The SPLP is NP-hard [1][6], and several exact and heuristic algorithms for solving it have been discussed in the literature. This problem can be represented using a mathematical model whose requirements are expressed by linear

relationships, so it is possible to use linear optimization to solve the problem instance or approximate the optimal solution.

One possible linear programming (LP) formulation for the Simple Plant Location Problem is the following:

$$x_{ij} = \begin{cases} 1 & \text{if the user } i \text{ is served by facility } j \\ 0 & \text{otherwise} \end{cases}$$

$$y_j = \begin{cases} 1 & \text{if the facility } j \text{ is opened} \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{cases} min \sum_{i \in U} \sum_{j \in F} x_{ij} c_{ij} + \sum_{j \in F} y_j f_j & & (1.1a) \\ s.t. \sum_{j \in F} x_{ij} = 1 & \forall i \in U & (1.1b) \\ x_{ij} \leq y_j & \forall i \in U \; \forall j \in F & (1.1c) \\ 0 \leq x_{ij} \leq 1 & \text{integer} & (1.1d) \\ 0 \leq y_j \leq 1 & \text{integer} & (1.1e) \end{cases}$$

where:

- 1.1a wants to minize the sum of the connection costs between users and facilities, plus the costs for opening those facilities.
- 1.1b tells that each user is served by exactly one facility.
- 1.1c grants that an user can be connected only to an open facility.

It is possible to turn the problem into its capacitated version by adding some constraints on facilities' capacities: each plant cannot support more than $d_j \in D$ users; the constraints we need to add to the model are the following:

$$\sum_{i \in U} x_{ij} \geq d_j \qquad \forall j \in F \tag{1.2}$$

More in general, each user can have a different "*volume*": there may exist an user i such that facilities need to use more than one capacity unit to serve it. In this case we need to define a vector $R = (\rho_i),\ i \in U$ of users' volumes, and we must modify the constraint 1.2 as follows:

$$\sum_{i \in U} \rho_i x_{ij} \geq d_j \; \forall j \in F \tag{1.3}$$

Finally the full model of the Capacitated Plant Location Problem will be the following:

$$x_{ij} = \begin{cases} 1 & \text{if the user } i \text{ is served by facility } j \\ 0 & \text{otherwise} \end{cases}$$

$$y_j = \begin{cases} 1 & \text{if the facility } j \text{ is opened} \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{cases} min \sum_{i \in U} \sum_{j \in F} x_{ij} c_{ij} + \sum_{j \in F} y_j f_j & & (1.4a) \\ s.t. \sum_{j \in F} x_{ij} = 1 & \forall i \in U & (1.4b) \\ x_{ij} \leq y_j & \forall i \in U \; \forall j \in F & (1.4c) \\ \sum_{i \in U} r_i x_{ij} \geq d_j & \forall j \in F & (1.4d) \\ 0 \leq x_{ij} \leq 1 & \text{integer} & (1.4e) \\ 0 \leq y_j \leq 1 & \text{integer} & (1.4f) \end{cases}$$

1.2 A survey on SPLP and CPLP

Most of the exact algorithms for SPLP are based on its mathematical programming formulation. The seminal paper of Erlenkotter [3] presents a dual-based algorithm for solving the SPLP that remains as one of the most efficient solution techniques for this problem. Prior to Erlenkotter, the best-known approaches for solving the SPLP where the branch-and-bound algorithm developed by Efroymson and Ray [7] and the implicit enumeration technique of Spielberg [8]. Khumawala [9] developed efficient branching and separation strategies for the branch-and-bound algorithm. The Erlenkotter approach is based on the "tight formulation" of the SPLP that is known to often produce natural integer solutions. This property of the tight formulation was first highlighted by Scharge [10] and was used effectively by [1].

In Chapter 2 we will describe the approach to solve the SPLP developed by Beltran-Royo et al. [2]. In their work they use a Semi-Lagrangean Relaxation combined with a general mixed integer programming solver. Their idea is to take advantage of the steadily increasing power of general MIP codes and aiming to enhance the performance of the MIP solver at low programming cost. This approach has been used as the starting point for the strategy we developed to find a heuristic solution to the optimization problem we will define further.

One of the earliest linear programming-based heuristics for the CPLP was developed by Kuehn and Hamburger in 1963 [4]. Branch and bound procedures for this problem where presented by Akinc and Khumawala [11] using a linear programming relaxation, and by Nauss [12] through Lagrangean relaxation.

One of the most effective strategies for the CPLP, is the cross-decomposition algorithm of Van Roy [13]. The basic idea of Van Roy's algoritm is to obtain a SPLP structure by dualizing the capacity constraints. This Lagrangean relaxation provides values for the location and allocation variables given a set of multipliers. The location decisions are then used to fix the integer variables and solve the CPLP as a transportation problem.

In a supply chain that comprises suppliers, plants, distribution centers, warehouses and customers, these basic formulations are relevant for making location decisions involving two consecutive echelons. For example, the focus of a majority of the literature on warehouse location, the SPLP and CPLP formulations are equally relevant for choosing suppliers to satisfy the needs of a firm's plants [14][15].

In the next Sections we will summarize some of the most popular approaches to the Plant Location Problem, as seen in [15].

1.3 Erlenkotter: A Dual-Based Procedure for the SPLP

Erlenkotter provides the same problem formulation seen in Section 1.1, that we report below for convenience:

$$x_{ij} = \begin{cases} 1 & \text{if the user } i \text{ is served by facility } j \\ 0 & \text{otherwise} \end{cases}$$

$$y_j = \begin{cases} 1 & \text{if the facility } j \text{ is opened} \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{cases} min \sum_{i \in U} \sum_{j \in F} x_{ij} c_{ij} + \sum_{j \in F} y_j f_j & & (1.5a) \\ s.t. \sum_{j \in F} x_{ij} = 1 & \forall i \in U & (1.5b) \\ x_{ij} \leq y_j & \forall i \in U \; \forall j \in F & (1.5c) \\ 0 \leq x_{ij} \leq 1 & \text{integer} & (1.5d) \\ 0 \leq y_j \leq 1 & \text{integer} & (1.5e) \end{cases}$$

There exists a weak formulation of the SPLP that uses a more compact model, generated by aggregating constraints 1.5c into a single constraint for each facility location j:

$$\sum_{i \in U} x_{ij} \geq |U| y_j \;\; \forall j \in F \tag{1.6}$$

In developing the solution approach, Erlenkotter utilizes a condensed dual formulation of this relaxation. Let v_i and w_{ij} represent the dual variables associated with constraints 1.5b and 1.5c. By relaxing y_j variables the dual problem can be formulated as follows:

$$\begin{cases} max \sum_{i \in U} v_i & (1.7a) \\ s.t. \sum_{i \in U} w_{ij} \leq f_j \qquad \forall j \in F & (1.7b) \\ v_i - w_{ij} \leq c_{ij} \qquad \forall i \in U \ \forall j \in F & (1.7c) \\ w_{ij} \geq 0 \qquad \forall i \in U, \ j \in F & (1.7d) \end{cases}$$

Since w_{ij} variables are not part of the objective function, we can set:

$$w_{ij} = max\{0, v_i - c_{ij}\},$$

so the condensed dual formulation of [3] will be:

$$\begin{cases} max \sum_{i \in U} v_i & (1.8a) \\ s.t. \sum_{i \in U} max\{0, v_i - c_{ij}\} \leq f_j \qquad \forall j \in F & (1.8b) \end{cases}$$

[15] explains that the Dual-Ascent procedure that constitutes the core of Erlenkotter's algorithm aims at increasing the values of v_i so as to maximize their sum. The idea is to use a quick heuristic for solving the condensed dual problem. To this end, the algorithm starts by setting the v_i values to the smallest c_{ij} for each customer i. At each iteration of the dual ascent procedure, the customer zones are processed one by one and the v_i value at each zone is raised to the next higher c_{ij} value, unless such an increase is constrainted by 1.8b. When the inequality 1.8b becomes binding during this process, the v_i value is increased to the highest level allowed by the constraint. The heuristic terminates when no further increase is possible for the v_i variables.

Once we have got the dual solution, we want to obtain a primal feasible solution to the original problem. It is helpful to analyze the complementary slackness conditions for the condensed dual and the linear programming relaxation. If x^*_{ij} and y^*_j are the optimal values of the primal decision variables and v^*_i are the optimal values for the dual ones, we have:

$$y^*_j[f_j - \sum_{i \in U} max\{0, v^*_i - c_{ij}\}] = 0 \ \ \forall j \in F \tag{1.9}$$

$$[y^*_j - x^*_{ij}]max\{0, v^*_i - c_{ij}\} = 0 \ \ \forall i \in U, \ j \in F \tag{1.10}$$

The Dual-Ascent procedure, produces a feasible solution v^*_j with at least one binding constraint 1.8b. For each associated location j, the slack of the dual constraint is zero, and it is possibble to set $y_j = 1$. It is likely that the procedure terminates with a solution where, among open facilities, there is more than one facility k with $c_{ij} \geq v_i$ for some i. This would violate 1.10,

since each customer zone must be served from the lowest cost open facility. Therefore it is possible to set $x_{ij} = y_j = 1$ only for the smallest value of c_{ij}. We obtain a sub-optimal primal feasible integer solution.

It is possible that there is more than one c_{ij} with a smaller value than v_i; To close the duality gap in such cases, Erlenkotter first uses a dual adjustment procedure and if this not suffice, he executes a simple B&B algorithm [3],[15].

1.4 Kuehn and Hamburger: A Heuristic Program for Locating Warehouses

First of all, Kuehn and Hamburger (K-H) focus their attention on the potential advantages of positioning warehouses in a distribution network:

1. Through warehouses is possible to generate economies of scale in transportation costs between factories and warehouses;

2. It is possible to implement economies of scope from combining products from different factories into a single shipment in serving customer demand;

3. It is possible to improve delivery times by increasing proximity to the end-users locations.

In positioning the warehouses K-H trade off the potential cost savings with the costs of establishing and maintaining them.

They develop their heuristic approach, starting from three assumptions:

- The most promising locations will be at or near concentrations of demand.

- Near optimum warehousing systems can be developed by locating warehouses one at a time, adding progressively those warehouses which produce the maximum cost savings for the system.

- Only a small subset of warehouse locations need to be evaluated in detail at each stage to determine the next site to be added.

Basically K-H assume that the set of M possible warehouse locations is a subset of demand locations.

Their algorithm is based on a constructive phase ("the main program") and the improvement phase ("the bump and shift routine"). They use a data structure that they call "the buffer" which is used to confine the detailed evaluation at each iteration to a subset of N locations.

- At the beginning of the constructive phase, the buffer is initialized with the N sites where serving the local demand with a local warehouse results in the hightest cost savings. Then the N sites in the buffer are assessed one by one in terms of the system-wide cost savings that can be obtained by opening the warehouse. The site that brings the hightest savings is

selected as a position where a warehouse must be open. The algorithm cycles between the buffer recostruction and the detailed evaluation, until all the demand is satisfied.

- Starting from a solution determined in the constructive phase, the algorithm tries to improve the soluton by eliminating some warehouses and/or moving them to a nearby position.

1.5 SPLP and CPLP Extensions

The SPLP and CPLP constitute the basic discrete facility location problems forumlations and there is an abundance of papers based on their extensions. Follows an overview of the most important works that extended the classical formulations by increasing the number of products, the number of facility echelons or the number of time periods included in the model.

1.5.1 Multi-Commodity PLP

An immediate generalization of the SPLP is the *multi-commodity* facility location problem that relaxes the single product assumption. Even if Neebe and Khumawala [16] and Karkazis and Boffey [17] offered two alternative formulations for this problem, both papers assumed that each facility deals with a single product. The first paper that studied a multi-commodity plant location model without any restriction on the number of products at each facility was written by Klincewicz and Luss [18].

One modeling example of a Multi-Commodity Plant Location Problem comes from Shen [19]. He uses the following notation to define the problem:

- I is the set of customers.
- L is the set of commodities.
- I_l is the set of customers that have demand for commodity $l \in L$.
- J is the set of candidate facility locations.

The problem takes the following input parameters:

- μ_{il}, the annual demand from customer i for commodity l.
- f_j, the fixed annualized cost of locating a facility at j, for each $j \in J$.

When a facility j is used to serve the customer in set $S \subset I$, the associated total cost is given by the following three cost components:

1. f_j, the fixed location cost.
2. $\sum_{i \in S, l \in L} d_{ijl}\mu_{il}$, where the term $d_{ijl}\mu_{il}$ is linear in μ_{il} where d_{ijl} is constant.

3. $\sum_{l \in L} G_{jl}(\sum_{i \in S} \mu_{il})$, where the term $G_{jl}(\sum_{i \in S} \mu_{il})$ is concave and non-decreasing in the total mean demand for commodity l at facility j.

For example, when d_{ijl} corresponds to the unit transportation cost for commodity l between facility j and customer i, the second term captures the total transportation cost if facility j provides the customers in set S with commodity l. The term $G_{jl}(\sum_{i \in S} \mu_{il})$ can be interpreted as the economies of scale cost term within the supply chain. For example, it can represent the facility operation and inventory repleinshment cost.

The MIP model build by [19] is the following:

$$x_{ijl} = \begin{cases} 1 & \text{if the demand for commodity } l \text{ of customer } i \text{ is served by } j \\ 0 & \text{otherwise} \end{cases}$$

$$x_j = \begin{cases} 1 & \text{if the facility } j \text{ is opened} \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{cases} min \sum_{j \in J} \Big\{ f_j x_j + \sum_{l \in L} \Big[\sum_{i \in I} (d_{ijl}\mu_{il}) y_{ijl} + G_{jl}(\sum_{i \in I} \mu_{il} y_{ijl}) \Big] \Big\} & \text{(1.11a)} \\ s.t. \sum_{j \in J} y_{ijl} = 1 & \forall i \in I, l \in L \quad \text{(1.11b)} \\ y_{ijl} - x_j \leq 0 & \forall i \in I, j \in J, l \in L \quad \text{(1.11c)} \\ 0 \leq x_j \leq 1 & \text{integer } \forall j \quad \text{(1.11d)} \\ 0 \leq y_{ijl} \leq 1 & \text{integer } \forall i, j, l \quad \text{(1.11e)} \end{cases}$$

To solve the problem [19] uses a Lagrangean relaxation embedded in a branch-and-bound procedure. In his paper, Shen shows how to drive low and upper bounds of the problem. They also present a variable fixing technique to speed up the algorithm. Refer to his paper for further readings.

1.5.2 Other Plant Location Derived Problems

Multi-Echelon Plant Location

Another important extension involves increasing the number of echelons incorporated in the problem formulation. One of the earliest *multi-echelon* formulations is by Kaufman et al. [20], which determined the locations of a set of facilities and a set of warehouses simultaneously. One of the most important papers following the Capacitated Plant Location Problem formulation in Kuehn and Hamburger [4] was the paper developed by Geoffrion and Graves [21]. Their model wanted to minimize the total cost of transportation and warehousing over a distribution network comprising three different echelons.

Multi Period Plant Location

[15] underlines how a number of researchers focused on relaxing the single period assumption of the two problems and developed models and solutions for the *dynamic* facility location problem. The objective was to determine the facilities positions at each time period so as to minimize the total costs for satisfing the customer demand. The earliest work on this problem is by Van Roy and Erlenkotter [22]: they extended the Erlenkotter algorithm (described in the next Sections) to handle the time periods specifications. Some other papers on this PLP variation are by Lim et al. [23] and Canel [24], who solved the problem with capacity restrictions at the facilities.

Improving the Costs Definition

Another stream of research is about improving the realism of the cost metrics. These efforts where made because of possible implications on economies of scale and economies of scope in the fixed and/or variable costs. Moreover the optimal solution should probably consider economic factors such as capacity acquisition and technology selection. Soland [25] is one of the earliest researcher who tried to incorporate economies of scale in the problem formulation, while Holmberg [26] extended the CPLP by formulating the capacity acquisition costs as arbitrary piecewise linear functions.

Plant Location and Uncertainities

Finally, an important stream of efforts in PL problems highlighted by [15], is about the incorporation of uncertainities in the problem parameters. The earlier works on this stream were by Jucker and Hodder et al. . They used scenario-based approaches in modeling a risk-averse decision maker's choices.

Chapter 2

Semi-Lagrangean Relaxation-based Algorithm for the SPLP

The Semi-Lagrangean Relaxation (SLR) method was introduced in [27] to solve the p-median problem. Compared to the Lagrangean Relaxation, the SLR method closes the duality gap and gives an optimal integer solution, but the relaxed model is harder to solve. The SLR applies to problems with equality constraints. Like in Lagrangean Relaxation, the equality constraints are relaxed, but the definition of the Semi-Lagrangean dual problem incorporates those constraints under the weaker form of inequalities. On combinatorial problems with positive coefficients, it has the strong property of achieving a zero duality gap. This method has been used with success to solve large instances of the p-median problem.

In this Chapter, after defining the Semi-Lagrangean Releaxation and after prooving that it closes the duality gap with the original problem, we will present the Dual-Ascent Algorithm developed by Beltran-Royo et al. [2]. We will briefly show how the authors applied this tecnique to the SPLP and we will explayn "why" the algorithm works. Note that some of the assumptions made by Beltran-Royo et al. that are valid for the Simple Plant Location, will not be valid anymore for our new extended problem, so we will show how the same strategy can be applied to find an heuristc solution to our model.

2.1 Semi-Lagrangean Relaxation

In this section, we summarize the main results obtained in [2] taking advantage of the simplified versions of the proofs provided in [2].

Consider the following "primal" problem:

$$\begin{cases} z^* = minc^T x & (2.1a) \\ s.t. Ax = b & (2.1b) \\ \quad x \in S \subset X \cap \mathbb{N}^n & (2.1c) \end{cases}$$

Assumption 1. *The components of $A \in \mathbb{R}^n \times \mathbb{R}^m, b \in \mathbb{R}^n$ and $c \in \mathbb{R}^m$ are non-negative.*

Assumption 2. *X is a polyhedral set, $\in S$ and 2.1a is feasible.*

Assumptions 1 and 2 together imply that 2.1a has an optimal solution.

The Semi-Lagrangean Relaxation consists in adding the inequality constraint $Ax \leq b$ and removing $Ax = b$ only. We obtain the dual problem:

$$\max_{u \in \mathbb{R}^n} \mathcal{L}(u) \tag{2.2}$$

where $\mathcal{L}(u)$ is the Semi-Lagrangean dual function defined as:

$$\begin{cases} \mathcal{L}(u) = \min\limits_x c^T x + u^T(b - Ax) & (2.3a) \\ s.t.\ Ax \leq b & (2.3b) \\ \quad x \in S & (2.3c) \end{cases}$$

Note that with our assumptions the feasible set of 2.3a is bounded. We also have that $x = 0$ is feasible to 2.3a. $\mathcal{L}(u)$ is well-defined, but the minimizer in 2.3a is not necessarily unique. We can write:

$$x(u) = arg\min_x \{c^T x + u^T(b - Ax) | Ax \leq b, x \in S\} \tag{2.4}$$

to denote one such minimizer. With this notation we may write $\mathcal{L}(u) = (c - A^T u)^T x(u) + b^T u$. We denote $\mathcal{U}^*$ the set of optimal solutions of problem 2.13. Finally, given two sets A and B, its addition corresponds to:

$$A + B = \{a + b : a \in A \text{ and } b \in B\} \tag{2.5}$$

Theorem 1. *The following statements hold [27]*

1. *$\mathcal{L}(u)$ is concave and $b - Ax(u)$ is a subgradient at u.*
2. *$\mathcal{L}(u)$ is monotone and $\mathcal{L}(u') \geq \mathcal{L}(u)$ if $u' \geq u$ and $u' \notin \mathcal{U}^*$.*
3. *$\mathcal{U}^* + \mathbb{R}^n_+ = \mathcal{U}^*$; thus $\mathcal{U}^*$ is an unboundend set.*
4. *if $x(u)$ is such that $Ax(u) = b$, then $u \in \mathcal{U}^*$ and $x(u)$ is optimal for problem 2.1a.*
5. *Conversely, if $u \in int(\mathcal{U}^*)$, then any minimizer $x(u)$ is optimal for problem 2.1a.*

6. the SLR closes the duality gap for problem 2.1a.

Proof. From the definition of the SLR function, the inequality:

$$\mathcal{L}(u') \leq c^T x(u) + (u')^T(b - Ax(u)) = \mathcal{L}(u) + (b - Ax(u))^T(u' - u) \tag{2.6}$$

holds for any pair u, u'. This shows that $\mathcal{L}(u')$ is concave and that $(b - Ax(u))$ is a subgradient at u.

To prove statement 2, we note, in view of $b - Ax(u') \geq 0$ and $u' \geq u$, that we have the chain of inequalities:

$$\begin{aligned}
\mathcal{L}(u') &= c^T x(u') + (b - Ax(u'))^T u', && \text{(2.7a)}\\
&= c^T x(u') + (b - Ax(u'))^T u + (b - Ax(u'))^T(u' - u), && \text{(2.7b)}\\
&\geq c^T x(u') + (b - Ax(u'))^T u, && \text{(2.7c)}\\
&\geq c^T x(u) + (b - Ax(u))^T u = \mathcal{L}(u) && \text{(2.7d)}
\end{aligned}$$

This proves the first part of the third statement. If $u' \notin \mathcal{U}^*$, then $0 \notin \partial\mathcal{L}(u')$, the subdifferential of $\mathcal{L}$ at u' [28], and we have $(b - Ax(u'))_j > 0$ for some j. Thus, $u < u'$ implies $(b - Ax(u'))^T(u' - u) > 0$. Hence $\mathcal{L}(u') > \mathcal{L}(u)$.

The third statement is a consequence of the monotone property of $\mathcal{L}(u)$ and $\mathcal{U}^*$ is convex since it is the optimal set of a concave function [28].

To prove the fourth statement, we note that $Ax(u) = b$ implies $0 \in \partial\mathcal{L}(u)$, a necessary and sufficient condition of optimality for problem 2.13. Hence $u \in \mathcal{U}^*$. Finally, since $x(u)$ is feasible to 2.1a and optimal for its relaxation, it is also optimal for 2.1a.

To prove the fifth statement, assume $u \in int(\mathcal{U}^*)$. In this case there exists $u' \in \mathcal{U}^*$ such that $u' < u$; thus $(b - Ax(u))^T(u - u') \geq 0$, with strict inequality if $b - Ax(u) \neq 0$. In view of 2.6:

$$0 \geq (b - Ax(u))^T(u' - u)) \geq \mathcal{L}(u') - \mathcal{L}(u). \tag{2.8}$$

Thus $Ax(u) = b$, and $x(u)$ is optimal to 2.1a. It follows that the original problem and the Semi-Lagrangian dual problem have the same optimal value (the last statement). □

2.2 The Dual-Ascent Method

The main idea of a dual ascent algorithm is to start with a price vector and successively obtain new price vectors with improved dual cost value, with the aim of solving the dual problem (in our case we want to solve the 2.13 dual problem). In this section we will state the algorithm and then prove its finite convergence.

Theorem 2. *The following statements hold:*

1. *Algorithm 1 is a dual ascent method when applied to solve the SLR dual problem: for any two consecutive iterates u^k and u^{k+1} we have $\mathcal{L}(u^{k+1}) > \mathcal{L}(u^k)$.*

Algorithm 1 Dual-Ascent algorithm (basic iteration)

1: Solve the Oracle, compute:

$$x^k = arg\min_x\{c^T x + (u^k)^T(b - Ax) | Ax \leq b, x \in S\},$$

where u^k is the current dual iterate.

2: **if** $s^k := b - Ax^k$ is equal to 0 **then**

3: Stop. (x^k, u^k) is an optimal primal-dual point.

4: **end if**

5: Update the dual iterate. For $j = 1, 2, ..., n$, set

$$u_j^{k+1} = \begin{cases} u_j^k + \delta_j^k & \text{if } s_k^k > 0 \quad (2.9a) \\ u_j^k & \text{otherwise} \quad (2.9b) \end{cases}$$

where $\delta_j^k \geq \Delta$.

2. Let us suppose that $u^0 \geq 0$ and that $\mathcal{U}^* \neq \emptyset$. Algorithm 1 converges to an optimal dual point $u \in \mathcal{U}^*$ after finitely many iterations.

Proof. The updating procedure of the Algorithm, consists in increasing some components of the current dual point, so $u^{k+1} > u^k$ and for the second statement of Theorem 1 we have $\mathcal{L}(u^{k+1}) > \mathcal{L}(u^k)$ and the first statement is proved.

Let us consider the sequences $\{s^k\}$ of subgradients generated by the algorithm. We have two exclusive cases:

Case 1 There exists k_0 such that $s^{k_0} = 0$. Then $0 \in \partial\mathcal{L}(u^{k_0})$ and $u^{k_0} \in \mathcal{U}^*$.

Case 2 At least for one component of s^k,say the $1-$st, there exists a subsequence$\{s_1^{k_i} \subset \{s_1^k\}$ such that $s_1^{k_i} \neq 0$ for all $i = 0, 1, 2, ...$ [2] proves by contradiction that this case cannot happen.

By definition of the Algorithm, we have:

$$u_1^{k_i} \geq u_1^{k_0} + i\Delta. \quad (2.10)$$

Then the subsequence $\{\mathcal{L}(u^{k_i})\}$ is unbounded, which contradicts the hypotesis $\mathcal{U}^* \neq \emptyset$.

Define $J^{k_i} = \{j | s_j^{k_i} > 0\}$. Since x is a binary vector, it implies, by Assumption 1, that there exists an absolute constat $\eta > 0$ such that:

$$\min_j \min_x\{s_j = (b - Ax)_j | (b - Ax)_j > 0\} = \eta. \quad (2.11)$$

Thus $s_j^{k_j} \geq \eta \ \forall j \in J^{k_i}$ and $\forall i$. Using the fact that $c^T x \geq 0$ and that $u^{k_i} \geq 0$, we have:

$$\begin{aligned}
\mathcal{L}(u^{k_i}) &= c^T x(u^{k_i}) + (b - Ax(u^{k_i}))^T u^{k_i} \\
&= c^T x(u^{k_i}) + (s^{k_i})^T u^{k_i} \\
&\geq (s^{k_i})^T u^{k_i} \\
&= \sum_{j \in J^{k_i}} s_j^{k_i} u_j^{k_i} \\
&\geq u_1^{k_i} \eta \\
&\geq (u_1^{k_0} + i\Delta)\eta.
\end{aligned}$$

Thus $\lim_{i \to \infty} \mathcal{L}(u^{k_i}) = +\infty$

□

2.3 Dual-Ascent Algorithm Applied to the SPLP

Now that we proved the Algorithm convergence, we need to apply this method to the SPLP. Also in this case we will summarize the Dual-Ascent algorithm specialization developed by [2]. It is useful to remember the Simple Plant Location linear programming model. Given a set $F = \{1, 2, ..., m\}$ of possible positions for opening a facility and a set $U = \{1, 2, ..., n\}$ of users that have to be served by those facilities, we can build the following model:

$$x_{i,j} = \begin{cases} 1 & \text{if the user } i \text{ is served by facility } j \\ 0 & \text{otherwise} \end{cases}$$

$$y_j = \begin{cases} 1 & \text{if the facility } j \text{ is opened} \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{cases}
min \sum_{i \in U} \sum_{j \in F} x_{ij} c_{ij} + \sum_{j \in F} y_j f_j & & \text{(2.12a)} \\
s.t. \sum_{j \in F} x_{ij} = 1 & \forall i \in U & \text{(2.12b)} \\
x_{ij} \leq y_j & \forall i \in U \ \forall j \in F & \text{(2.12c)} \\
0 \leq x_{ij} \leq 1 & \text{integer} & \text{(2.12d)} \\
0 \leq y_j \leq 1 & \text{integer} & \text{(2.12e)}
\end{cases}$$

Let's formulate the Semi-Lagrangean Relaxation for this problem. We obtain the following problem:

$$\max_{u \in \mathbb{R}^n} \mathcal{L}(u) \tag{2.13}$$

and the following oracle:

$$\begin{cases} min \sum_{i\in U}\sum_{j\in F} x_{ij}(c_{ij}-u_j) + \sum_{j\in F} y_j f_j + \sum_{i\in U} u_i & (2.14a) \\ s.t \sum_{j\in F} x_{ij} \leq 1 \quad \forall i \in U & (2.14b) \\ x_{ij} \leq y_j \quad \forall i \in U \; \forall j \in F & (2.14c) \\ 0 \leq x_{ij} \leq 1 \quad \text{integer} & (2.14d) \\ 0 \leq y_j \leq 1 \quad \text{integer} & (2.14e) \end{cases}$$

Let us denote $(x(u), y(u))$ an optimal point for the oracle.

2.3.1 Properties of the SLR Dual Problem

In this section we summarize properties of the SLR dual problem that where underlined in [2]. These properties allow us to restrict the dual search to a box, that is, we will be able to remove some of the variables from the problem, proving that they do not belong to any optimal solution.

[2] first defines for each client i, its *best combined costs* as:

$$\tilde{c}_i := min_j\{c_{ij} + f_j\}. \tag{2.15}$$

Consider now the vector of best combined costs $\tilde{c} := (\tilde{c}_1, \tilde{c}_2, \tilde{c}_3, ..., \tilde{c}_n)$ and the vector of the *sorted costs* for each client i:

$$c_i^1 \leq c_i^2 \leq c_i^3 \leq ... \leq c_i^n. \tag{2.16}$$

Theorem 3. *$u \geq \tilde{c} \Rightarrow u \in U^*$ and $u > \tilde{c} \Rightarrow u \in int(U^*)$.*

Proof. Consider the oracle:

$$\begin{cases} min \sum_{i\in U}\sum_{j\in F} x_{ij}(c_{ij}-u_j) + \sum_{j\in F} y_j k_j + \sum_{i\in U} u_i & (2.17a) \\ s.t. \sum_{j\in F} x_{ij} \leq 1 \quad \forall i \in U & (2.17b) \\ x_{ij} \leq y_j \quad \forall i \in U \; \forall j \in F & (2.17c) \\ 0 \leq x_{ij} \leq 1 \quad \text{integer} & (2.17d) \\ 0 \leq y_j \leq 1 \quad \text{integer} & (2.17e) \end{cases}$$

Assume $u \geq \tilde{c}$. If there exists an optimal solution of the Oracle such that:

$$\sum_{j\in F} x_{ij} = 1 \; \forall i \in U, \tag{2.18}$$

then for Theorem 1, this is an optimal solution for the original problem. Suppose that we have an oracle solution such that, for some i,

$$\sum_{j\in F} x_{ij} = 0. \tag{2.19}$$

Let j_k be:

$$j_k \ : \ \tilde{c_i} = f_{j_k} + c_{ij_k}. \tag{2.20}$$

By hypotesis,$\tilde{c_i} - u_i \leq 0$ and $f_{j_k} + (c_{ij_k} - u_i) \leq 0$, so it is possible to set $x_{ij_k} = 1$ and $y_{j_k} = 1$ without increasing the objective value.

This implies that the new solution is also optimal, and there exists:

$$\sum_{j \in F} x_{ij} = 1 \ \forall i \in U. \tag{2.21}$$

The second statement follows from $\tilde{c} \in \mathcal{U}^*$ and statement 3 of Theorem 1. □

Theorem 4. *If $u \in int(\mathcal{U}^*)$, then $u \geq c^1$.*

Proof. [2] proves this theorem by contradiction.

Assume that $u_{i_0} < c^1_{i_0}$ for some $i_0 \in U$. If $u_{i_0} < c^1_{i_0}$ then $c^k_{i_0} - u_{i_0} > 0$ for all $k \in F$. Any optimal solution $x(u)$ is such that $x_{i_0 j}(u) = 0$, for all $j \in F$. Hence, $1 - \sum_{j \in F} x_{i_0 j}(u) = 1$ and by Theorem 1, u is not in $int(\mathcal{U}^*)$ which contradicts the Theorem hypotesis. □

Finally [2] proves that it is possible to define a box $\mathcal{B} \subset \mathcal{U}$ such that there exists at least one dual optimal solution $\dot{u}$ for which is true that:

$$\dot{u} \in \mathcal{B} \cap \mathcal{U}^* \tag{2.22}$$

Corollary 1. *Consider the scalar $\epsilon > 0$, the vector $\bar{\epsilon}$ where each compoment is equal to ϵ and the box:*

$$\mathcal{B} := \{u \in \mathbb{R}^n | c^1 < u \leq \tilde{c} + \bar{\epsilon}\}. \tag{2.23}$$

*Then, **for the SPLP**:*

$$int(\mathcal{U}^*) \cap \mathcal{B} \neq \emptyset. \tag{2.24}$$

This corollary implies that taking $\bar{u} = \tilde{c} + \bar{\epsilon}$ and solving the Oracle, yields to a primal optimal solution in one step. Of course at this point the oracle is too difficult to be solved, so it makes sense to apply the Dual-Ascent Algorithm described above, in order to detect a solution $u^* \in \mathcal{U}^*$ for which the oracle is smaller and easier.

2.4 The Core Subproblem

[2] explains how some primal variables x_{ij} can be set to zero when u_i is small enough. Moreover, we will present the specialized version of the Dual-Ascent Algorithm for the SPLP presented in [2].

Let $c(u)$ be the matrix of reduced costs such that $c(u)_{ij} = c_{ij} - u_i$. Let $G = (W \times V, E)$ be the bipartite graph associated to the SPLP problem, such

that each node in W represents a client, each node in V stands for a facility and each edge e_{ij} exists if facility j can serve user i. Let $E(u) \subset E$ be the subset of edges with strictly negative reduced cost for a given dual point u. Let $V(u) \subset V$ and $W(u) \subset W$ be adjacent vertices to $E(u)$. Then the subgraph $G(u) = (W(u) \times V(u), E(u))$ is the *core subgraph*.

Assumption 3. *For any $c(u)_{ij} \geq 0$, there exists $x(u)$ such that $x(u)_{ij} = 0$. Therefore we can restrict our search to the core subgraph $G(u)$ to compute $x(u)$.*

The advantage of solving the oracle created on the subgraph $G(u)$ is that we have a linear model with much less variables. This makes the model more handful for general MIP solvers. Another advantage is that $G(u)$ may be decomposable into independent subgraphs, allowing us to separate the core subproblem into smaller subgraphs.

[2] uses the sorted costs to partition the domain of each coordinate u_i into intervals like $[c_i^k, c_i^{k+1}]$, with $c_i^{m+1} = +\infty$. In this way the core subgraph can be partitioned in elementary boxes and the dual search can be restricted to one representative point per elementary box. Follows [2] algorithmic scheme.

Take a small $\epsilon > 0$ and for each client i, take some $l(i) \in F$ where $u_i = c_i^{l(i)} (i = 1, ..., n)$ and start the dual search at $u_i^0 = c_u^{l(i)}$. Query the Oracle and if the current dual iterate is not optimal (that is there exist at least one i such that $\sum_{j \in F} x_{ij} = 0$), we update u_i:

$$u_i = c_i^{l(i)} \rightarrow c_i^{l(i)+1} + \epsilon \text{ for some } l(j) \in F \tag{2.25}$$

We only update the coordinates for the iterate u^k whose corresponding subgradient coordinate is not null.

Algorithm 2 Dual-Ascent algorithm (basic iteration)

1: Set $k = 0$ and $\epsilon > 0$. For each client $i \in U$, set

- $u_i^0 = c_i^{l(i)} + \epsilon$ for some $l(i) \in F$
- $\tilde{c}_i = \min_j \{c_{ij} + f_j\}$
- $c_i^{m+1} = +\infty$

2: **while** Solution is not optimal **do**

3: Compute $\mathcal{L}(u^k), (x(u^k), y(u^k)))$ and s^k where

$$s_j^k = 1 - \sum_{j \in F} x_{ij}^k \ \ \forall i \in U$$

4: **if** $s^k = 0$ **then**

5: Stop. The pair $(u^k, (x(u^k), y(u^k)))$ is a primal-dual optimal point

6: **end if**

7: For each $i \in U$ such that $s_i^k = 1$, set

$$u_i^{k+1} = min\{c_i^{l(i)+1}, \tilde{c}_i\} + \epsilon \ \text{ and } \ l(i) = min\{l(i) + 1, m\}.$$

8: set $k = k + 1$

9: **end while**

Chapter 3

MIN-MAX Capacitated and Budgetized Plant Location

What we are going to do in this Chapter is to present a specialization of the Plant Location Problem, involving some constraints on facilities capacities and on the maximum budget that can be spent on facilities openings. Moreover, we also wanted to "optimize" the size of the facilities that were open in the optimal solution. We decided to integrate all the specifics in a single linear programming model. The problem that came out, was very hard to solve, expecally because, as we will see below, the instance's size grows very fast in relation to the number of possible facility locations.

3.1 Problem Description

We first recall sets and variables involved it the CPLP and then we explain the new constraints added to our Plant Location specialization. We have a set $U = 1, 2, 3, .., n$ of clients, a set $F = 1, 2, 3, ..., m$ of possible plants' locations, a vector $K = (f_j)$ of fixed costs for setting up plants at sites $j \in F$, a matrix $C = [c_{i,j}]$ of transportation costs from $i \in U$ to $j \in F$ as input, a vector $D = (d_j),\ j \in F$ of the maximal number of users supported by facility j and finally a vector $R = (r_i),\ i \in U$ for the users' volumes. In our problem, we first decided to let the user fix the maximum budget that the solution can spend in facilities openings. The relative constraint is quite simple:

$$\sum_{j \in F} f_j y_j \leq BUDGET \tag{3.1}$$

Each plant can support a maximum number of users (we call this number as "maximum capacity") and must support a minimum number of users ("minimum capacity"), that is a facility cannot be open if the number of users that want to be served by it are less than its minimum capacity. These specifications could be modeled by using $2|M|$ constraints where M is the number of possible positioning sites, but another specification makes this number much bigger.

Each facility belongs to one of three exclusive classes: "big facilities", "medium facilities" and "small facilities". To introduce this concept in the MIP model, we decided to triplicate the number of y variables, by positioning three virtual "opening sites" onto the real ones. For example if we have a point (X, Y) which is a possible opening site, we added three y variables for that point; of course we needed to forbid the possibility to have more than one opening at the same (X, Y) point. According to this specification, we needed to specify different minimum and maximum capacities for each different class. Note that, since we are triplicating the number of y variables, we are also triplicating the number of constraints about facilities capacities. We call $\mathcal{B}$, $\mathcal{M}$, $\mathcal{S}$ respectively the set of big, medium, and small facilities, with $F = \mathcal{B} \cup \mathcal{M} \cup \mathcal{S}$.

If we define $T = (MAX\_CAPACITY_j),\ j \in F$ as the vector of the maximum number of users supported by facility j and $t = (MIN\_CAPACITY_j),\ j \in F$ as the vector of the minimum number of user that must be supported by facility j if we want it to be open, then we have the following constraints about facilities capacities:

$$\sum_{i \in U} \rho_i x_{ij} \leq MAX\_CAPACITY_j \ \ \forall j \in F \tag{3.2}$$

$$\sum_{i \in U} \rho_i x_{ij} \geq y_j MIN\_CAPACITY_j \ \ \forall j \in F \tag{3.3}$$

To force the problem to open only one facility onto each selected (X, Y) point, we added the following constraints:

$$y_s + y_m + y_b \leq 1 \ \ \forall Y_{s,m,b} \subset F \ : \ s, m, b \in Y_{s,m,b} \tag{3.4}$$

where $s, m, b \in F$ are three facilities which have the same opening point.

The overall MIP problem is the following:

$$\begin{cases} min \sum_{i \in U} \sum_{j \in F} x_{ij} c_{ij} + \sum_{j \in F} y_j f_j & & \text{(3.5a)} \\ s.t. \sum_{j \in F} x_{ij} = 1 & \forall i \in U & \text{(3.5b)} \\ x_{ij} \leq y_j & \forall i \in U \ \forall j \in F & \text{(3.5c)} \\ \sum_{j \in F} f_j y_j \leq BUDGET & & \text{(3.5d)} \\ \sum_{i \in U} \rho_i x_{ij} \leq MAX\_CAPACITY_j & \forall j \in F & \text{(3.5e)} \\ \sum_{i \in U} \rho_i x_{ij} \geq y_j MIN\_CAPACITY_j & \forall j \in F & \text{(3.5f)} \\ y_s + y_m + y_b \leq 1 & \forall Y_{s,m,b} = \{s, m, b\} & \text{(3.5g)} \\ 0 \leq x_{ij} \leq 1 & \text{integer} & \text{(3.5h)} \\ 0 \leq y_j \leq 1 & \text{integer} & \text{(3.5i)} \end{cases}$$

Note that we can give different formulations for this problem. For example we could modify the constraint 3.5e as:

$$\sum_{i \in U} \rho_i x_{ij} \leq y_j MAX\_CAPACITY_j \ \ \forall j \in F. \tag{3.6}$$

This simple modification causes the constraints 3.5c to be unnecessary, since they are implicated by constraints 3.5f and 3.6.

Moreover, we can eliminate from the model the constraints 3.5g trasforming constraints 3.5f and 3.6 as follows:

$$\sum_{i \in U} \rho_i x_{ij} \leq y_s MAX\_C_s + y_m MAX\_C_m + y_b MAX\_C_b \ \ \forall Y_{s,m,b} = \{s, m, b\} \subset F \tag{3.7}$$

$$\sum_{i \in U} \rho_i x_{ij} \geq y_s MIN\_C_s + y_m MIN\_C_m + y_b MIN\_C_b \ \ \forall Y_{s,m,b} = \{s, m, b\} \subset F \tag{3.8}$$

Where MIN_C_j and MAX_C_j are the minimum capacity and the maximum capacity of facility j respectively.

We call this problem *MIN-MAX Capacitated and Budgetized Plant Location Problem* (MMBPLP).

3.2 NP-Hardness

In this section we will prove that the MMBPLP is NP-Hard by reducing it from the Simple Plant Location Problem.

Theorem 5.

$$SIMPLE\_PLANT\_LOCATION\_PROBLEM \prec_p MMBPLP. \tag{3.9}$$

Proof. Define $\mathcal{I}_{SPLP}$ and $\mathcal{I}_{MMBPLP}$ as the sets of the Simple Plant Location and MMBPLP problems' instances respectively, then to prove 3.9 we need to define a bijective function f such that:

$$f : \mathcal{I}_{SPLP} \rightarrow \mathcal{I}_{MMBPLP}. \tag{3.10}$$

Moreover, to be the reduction valid, we must show:

$$A_f \in P, \tag{3.11}$$

where P is the complexity class that contains all decision problems that can be solved by a deterministic Turing machine in polynomial time. To complete the reduction we have to:

- Assume to have a polynomial algorithm to solve the MMBPLP
- Prove that this algorithm can be used to solve in polynomial time every instance of th SPLP.

Starting from a Simple Plant Location Problem instace, we need to add all the constraints needed to turn it into a MMBPLP instance. First of all let we think about the budget. We can set the maximum budget that can be spent in the optimal solution as:

$$BUDGET = \sum_{j \in F} f_j \tag{3.12}$$

In this way we have no budget limitations in the MMBPLP, since it is not possible to spend more than the sum of all facilities costs.

A similar approach can be used to add constraints about maximum capacities:

$$MAX\_CAPACITY_j = \sum_{i \in U} \rho_i \ \ \forall j \in F. \tag{3.13}$$

For the minimum capacity we can set:

$$MIN\_CAPACITY_j = 1 \ \ \forall j \in F, \tag{3.14}$$

that is, a generic facility can be open only if at least one user is connected to it, just like in the SPLP.

Finally we need to handle constraints 3.5g; one solution is to simply triplicate the facilities, by positioning the new ones onto the facilities sites of the original SPLP instance. It is easy to see that each instance in $\mathcal{I}_{SPLP}$ can be transformed into an instance of $\mathcal{I}_{MMBPLP}$ and that if we solve the $\mathcal{I}_{MMBPLP}$ problem, we solve the original problem.

Define $\mathbb{F} = F \cup F \cup F$ (remember that we triplicated the number of facilities because we need to add this specification to turn SPLP in MMBPLP), then the resulting MIP model will be the following:

$$\begin{cases} min \sum_{i \in U} \sum_{j \in F} x_{ij} c_{ij} + \sum_{j \in F} y_j f_j & & (3.15a) \\ s.t. \sum_{j \in \mathbb{F}} x_{ij} = 1 & \forall i \in U & (3.15b) \\ x_{ij} \leq y_j & \forall i \in U, \ \forall j \in \mathbb{F} & (3.15c) \\ \sum_{j \in \mathbb{F}} f_j y_j \leq \sum_{j \in \mathbb{F}} f_j & & (3.15d) \\ \sum_{i \in U} \rho_i x_{ij} \leq \sum_{i \in U} \rho_i & \forall j \in \mathbb{F} & (3.15e) \\ \sum_{i \in U} \rho_i x_{ij} \geq y_j & \forall j \in \mathbb{F} & (3.15f) \\ y_{j_1} + y_{j_2} + y_{j_3} \leq 1 & \forall \tilde{Y} = \{j_1, j_2, j_3\} \subset \mathbb{F} & (3.15g) \\ 0 \leq x_{ij} \leq 1 & \text{integer} & (3.15h) \\ 0 \leq y_j \leq 1 & \text{integer} & (3.15i) \end{cases}$$

□

3.3 The Semi-Lagrangean Relaxation with the New Problem

As we described in previous sections, we want to use the same approach of [2] to find a heuristc solution to MMBPLP. Since the MIP model is very "dense" (there is a big number of non-zero variables in the constraints), we wanted to apply the same principle used by [2] for removing the x_{ij} variables from the Oracle, and approximate the problem's solution by using the dual Ascent method. We are talking about approximation and heuristics, because some theorems and results that are valid for the SPLP are not valid anymore for the MMBPLP; such these differences are generated by the constraints on facilities minimum capacities. In this Section, we will first describe how to generate the Semi-Lagrangean Relaxation for the MMBPL Problem and then we will show how we applied the Dual-Ascent Algorithm we have seen in 2.3.

Like in [2], we have to dualize the equality constraints 3.5b. The MIP model of the oracle will be the following:

$$
\begin{cases}
min \sum_{i \in U} \sum_{j \in F} x_{ij}(c_{ij} - u_j) + \sum_{j \in F} y_j f_j + \sum_{i \in U} u_i & & \text{(3.16a)} \\
\sum_{j \in F} x_{ij} \leq 1 & \forall i \in U & \text{(3.16b)} \\
s.t.\ x_{ij} \leq y_j & \forall i \in U\ \forall j \in F & \text{(3.16c)} \\
\quad \sum_{j \in F} f_j y_j \leq BUDGET & & \text{(3.16d)} \\
\quad \sum_{i \in U} \rho_i x_{ij} \leq MAX\_CAPACITY_j & \forall j \in F & \text{(3.16e)} \\
\quad \sum_{i \in U} \rho_i x_{ij} \geq y_j MIN\_CAPACITY_j & \forall j \in F & \text{(3.16f)} \\
\quad y_s + y_m + y_b \leq 1 & \forall Y_{s,m,b} \subset F & \text{(3.16g)} \\
\quad 0 \leq x_{ij} \leq 1 & \text{integer} & \text{(3.16h)} \\
\quad 0 \leq y_j \leq 1 & \text{integer} & \text{(3.16i)}
\end{cases}
$$

With this new model we decided to apply the same Dual-Ascent Algorithm of [2], keeping the Assumption 3 to remove some x_{ij} variables from the oracle. While the assumption is very useful in our heuristic approach, it is not mathematically correct, since the Oracle's optimal solution could contain some of the x_{ij} variables removed by using the Assumption. This happens because of the constraints on minimum facilities capacities:

$$\sum_{i \in U} \rho_i x_{ij} \geq y_j MIN\_CAPACITY_j \tag{3.17}$$

Proposition 1. *Given the Oracle for the Semi-Lagrangean Relaxation of the MMBPLP,* $x_{ij} - u_i < 0 \nRightarrow x_{ij} = 0$ *in the Oracle MIP model.*

Proof. To prove the Proposition, we can provide a counter example.

Suppose we have a facility y_q such that its opening cost $f_q = \epsilon > 0$. This facility's minimum capacity is γ_q and its maximum capacity is Γ_q.

Suppose there exists an user p such that $c_{pq} = \epsilon$ with volume ρ_p, with $|c_{pq}| < |f_q|$.

Assume now that there is a subset Q of users, such that each user $i \in Q,\ i \neq p$, has $c_{iq} < c_{ij} + \epsilon\ \forall j \in F, j \neq q$. Suppose the set Q to be such that:

$$\sum_{i \in Q} \rho_i = \gamma_q - \rho_p \tag{3.18}$$

Imagine that the budget is:

$$BUDGET = \sum_{j \in F} f_j. \tag{3.19}$$

Because of the Assumption 3, we removed the x_{pq} variable from the Oracle. Solving this (reduced) Oracle, we get a solution S^* (S^* is the set of both x and y variables selected in the solution). Imagine that every facility that is open in the actual solution could still be open if we "disconnect" all the users in Q (that is the remaining users have still enough volume to keep open all the facilities in S^*), and call $\tilde{Q}$ the set of disconnected users. Define S_Q as the set of x_{ij} variables that represent the connection between every $i \in Q$ and the facility q, and define the set $S_{\tilde{Q}}$ as the set of variables relative to the disconnections of users in $\tilde{Q}$ in S^*.

If we add to the problem the x_{pq} edge, then we have:

$$\sum_{i \in Q \cup \{p\}} \rho_i = \gamma_q, \tag{3.20}$$

that is, the users in $Q \cup \{p\}$ have enough volume to open the facility q. If we open the facility q, the new solution cost is:

$$\sum_{x_{ij} \in S^*} c_{ij} + \sum_{y_j \in S^*} f_j + f_q + (\sum_{x_{ij} \in S_Q} c_{ij} - \sum_{x_{ij} \in S_{\tilde{Q}}} c_{ij} + c_{pq}). \tag{3.21}$$

Now, since $c_{iq} < c_{ij} + \epsilon\ \forall j \in F, j \neq q,\ \forall i \in Q$, we have:

$$\sum_{x_{ij} \in S_Q} c_{ij} > \sum_{x_{ij} \in S_{\tilde{Q}}} c_{ij} \tag{3.22}$$

Moreover, since $f_q = c_{pq}$, we can state:

$$f_q + (\sum_{x_{ij} \in S_Q} c_{ij} - \sum_{x_{ij} \in S_{\tilde{Q}}} c_{ij} + c_{pq}) < 0, \tag{3.23}$$

and we prove the Proposition by noting that:

$$\sum_{x_{ij} \in S^*} c_{ij} + \sum_{y_j \in S^*} f_j > \sum_{x_{ij} \in S^*} c_{ij} + \sum_{y_j \in S^*} f_j + f_q + (\sum_{x_{ij} \in S_Q} c_{ij} - \sum_{x_{ij} \in S_{\tilde{Q}}} c_{ij} + c_{pq}). \tag{3.24}$$

□

Another key difference between SPLP and MMBPLP is that, when in the first problem SLR we have that an user's Lagrangean multiplier is greater then the "best combined cost" for that user, then we can stop to increase that multiplier, because the worst thing that can happen in the optimal solution is that we open the facility involved in the best combined cost and connect the user to that facility. Once we introduced constraints on facilities capacities, this assumption is not valid anymore.

For the reasons described above, we cannot state that the Dual-Ascent Algorithm previously exposed, mathematically reach the optimal solution after a finite number of steps.

3.4 MMBPLP Feasibility

Before we describe the main algorithm we used to solve the MMBPLP, we need to define a procedure to "fastly" define if a general instance of our problem is feasible. Of course it is not possible to check feasibility in polynomial time, unless $P = NP$, but it is still possible to decompose the MMBPLP in simpler derived problems, such that we can prove the original problem's feasibility by analyzing the subproblems. In the next Sections we will define how it is possible to derive the MMBPLP feasibility by analyzing two induced subproblems: the "Min Capacity Knapsack Problem" (MCKP) and the "Maximal Sum Problem" (MSP). We will first define the MCKP and the MSP, and then we will prove a sufficient condition to derive the original problem feasibility by solving them both.

All this process will be addressed in the below as the "Feasibility Check", and it will be used as a primitive function in the Progressive Plants Selector Algorithms described below.

3.5 The Min Capacity Knapsack Problem

The 0-1 Knapsack is one of the most important problems of combinatorial optimization. Is given a set of items J, each with a profit p_j and a weight w_j, $j \in J$, these items must be packed into a knapsack of capacity c and the objective is to choose a subset $\tilde{J} \subseteq J$ such that the total capacity of the items in $\tilde{J}$ is less or equal to c, and the profit of the chosen object is maximum.

The problem can be expressed by using Mixed-Integer Linear Programming:

$$x_j = \begin{cases} 1 & \text{if the item } j \text{ has to be included in the knapsack} \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{cases} max \sum_{j \in J} p_j x_j & (3.25a) \\ s.t. \sum_{j \in J} w_j x_j \leq c & (3.25b) \\ 0 \leq c_j \leq 1 \qquad \text{integer} & (3.25c) \end{cases}$$

Our first derived problem is a specialization of the 0-1 Knapsack Problem.

Consider the orignal problem. We would like to know what is the "best way" to allocate the budget we have at our disposal, that is we want to know the facilities sizes such that we can serve all the users whitout violating the budget constraint. In this problem we also need to keep constraints 3.5g. The resulting LP problem is the following:

$$y_j = \begin{cases} 1 & \text{if structure } j \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{cases} min \sum_{j \in F} y_j MIN\_CAPACITY_j \\ s.t. \sum_{j \in F} f_j y_j \leq BUDGET \\ \quad \sum_{j \in F} MAX\_CAPACITY_j y_j \geq \sum_{i \in U} \rho_i \\ \quad y_s + y_m + y_b \leq 1 & \forall Y_{s,m,b} = \{s, m, b\} \subset F \\ \quad 0 \leq y_j \leq 1 & \text{integer} \end{cases}$$

For the variables definition just look at the Sections above. Notice that the problem is still NP-Hard, indeed it is possible to reduce it from 0-1 Knapsack, by setting:

$$MAX\_CAPACITY_j = \sum_{i \in U} \rho_i. \tag{3.26}$$

Proposition 2. *If the induced MCKP is not feasible, then the original MMBPLP is not feasible.*

Proof. We have:

MMBPLP is feasible $\Rightarrow$ the induced MCKP is feasible.

and if we consider its contrapositive:

The induced MCKP is not feasible $\Rightarrow$ MMBPLP is not feasible.

□

Of course the inverse is not true, because we did not consider the constraints on facilities capacities. We define $\Phi^*(MCKP)$ as the optimal solution of this problem. The set $\Phi^*(MCKP)$ is a good candidate to use if we want to check the problem feasibility.

The only thing we have left to do is to check whether $\Phi^*(MCKP)$ can yeld to a feasible solution to the original MMBPLP or not. Starting from those $\Phi^*(MCKP)$ we want to satisfy the capacity constraints on facilities. To do that, we need to know if there exists any combination of x_{ij} variables such that they satisfy the following constraints:

$$\begin{cases} \sum\limits_{j \in \Phi^*(MCKP)} x_{ij} = 1 & \forall i \in U \\ s.t. \sum\limits_{i \in U} \rho_i x_{ij} \leq MAX\_CAPACITY_j & \forall j \in \Phi^*(MCKP) \\ \sum\limits_{i \in U} \rho_i x_{ij} \geq MIN\_CAPACITY_j & \forall j \in \Phi^*(MCKP) \\ 0 \leq x_{ij} \leq 1 & \text{integer} \end{cases}$$

Informally, we need to know if there is a combination of x_{ij} such that it is possible to open all the facilities in $\Phi^*(MCKP)$ while satisfing both minimum and maximum capacities constraints.

What we have got in this Section, is a sufficient condition to dermine the main problem feasibility. Even if we are splitting our problem in two different NP-Hard ones, in practice solving the MCKP and checking the feasibility of the second model, is much faster then determine the feasibility of the original MMBPLP.

As described above, we will use this method in the algorithm that follows.

Chapter 4

Progressive Plants Selector

In this Chapter we will describe the heuristic algorithm that we developed for finding a good feasible solution to MMBPLP. This method has been studied considering the necessity to have a rapid feedback from the optimization system without sacrificing too much quality.

The algorithm that follows is based on the Semi-Lagrangean Relaxation we described in Section 3.3; another possibility could have been to approximate the problem through a simple Lagrangean Relaxation, but we decided to follow the other way since the SLR fits the original problem much better than its simple version.

What we have got is a method that tipically yelds to a solution, on random graphs (the hardest to solve), with less then 10% duality gap.

4.1 General Idea

Since we wanted to have a rapid feedback from our system, we thought to use a greedy approach to solve the problem, because it allows to display fragments of the solution during the algorithm execution. Our method is based on the Dual-Ascent Algorithm described above and we exploited Lagrangean coefficients to have information about which facilities we wanted to be open.

During the Progressive Plants Selector (PPS) we reduce the instace size until we get to a problem that is solvable through the general MIP solver. To do that, we apply the Dual-Ascent Method to the current instance (until some stop conditions are verified), then we extract from the solution some of the most convenient facilities with all the users that are connected to them. When we arrive to an instance size that is good enough to be solved directly, we solve it to get the last facilities positions.

In Figure 4.1 it is possible to see the flowchart of our algorithm.

Since we are reducing the instance size by starting from a solution that is not feasible for the original problem (the Semi-Lagrangean relaxation allows users to not connect to any of the facilities) and since open facilities and users connections could be part of an infeasible solution for the original problem,

we need to certify, at each iteration of the reduction procedure, that the new problem is still feasible in the sense of the original formulation.

After we decided the facilities positions and their size, we re-optimize the users allocation in order to reduce the solution cost as much as possible.

4.2 Algorithm Description

We now describe step-by-step the Progressive Plants Selector Algorithm.

Before starting the instance resolution, we first check whether the sufficient condition for problem feasibility is valid or not. If the feasibility check is not passed, then we send a warning message as output.

The PPS algorithm inputs are:

- a *step size*, to be intended as the percentage of open facilities at each iteration of the reducion procedure to be removed from the problem instance. For example if the i-th solution provided by the reduction procedure has q open facilities and the step size is $\alpha \in [0, 1]$, the reduction will take place by removing from the problem the "best" $\lceil \alpha q \rceil$ facilities (with all the users connected to them);
- a "*reasonability threshold*", that is the size of the problem we belive to be small enough to be optimized by the MIP solver. Here when we speak about "size", we intend the number of users to be allocated.
- a "*time limit*" for the Dual-Ascent Method that is tipically set to some minutes.

The procedure that follows is iterated until both the following conditions are verified:

1. The problem size is not reasonable.
2. The last iteration yeld to a feasible problem: if the i-th iteration has generated a infeasible problem, we rollback the changes that were previously made (that is we re-insert both users and facilities that were removed from the instance) and we start the resolution with the MIP solver..

We run the Dual-Ascent Algorithm on the current problem instance until the time limit is reached and until we have found at least one new facility to open; these are the candidates from which to choose those elements to add to the solution we are building.

The criterion that we adopted to choose which facilities to extract from the problem, uses the Lagrangean coefficients that are computed during the Dual-Ascent Algorithm. These coefficients are user-related, but we want to extract the facilities from the problem. If u_i is the Lagrangean coefficients for user i, we define $score(j)$ as:

$$score(j) = \sum_{i \in U} u_i x_{ij}, \tag{4.1}$$

and we define as the "most convenient facility", the plant j such that:

$$j = \arg max\{score(j) : j \in F\}. \tag{4.2}$$

As mentioned, if the Dual-Ascent Procedure contains q open facilities and the *step size* is $\alpha \in [0, 1]$, then we extract from the solution the top $\lceil \alpha q \rceil$ facilities from the list of facilities sorted in decreasing order by their score.

At this point we use the feasibility check to determine whether the new problem is solvable or not [1]. If the check is not passed, then we rollback the changes and we start the MIP solver, else we iterate the reduction procedure.

Of course before iterating we need to update the problem data:

1. we need to update the budget subtracting all the budget we spent in the actual solution;

2. we need to remove all the facilities that we have open and all the users that are connected to them;

3. we need to remove from the problem all the variables that are relative to the same position of all facilities in the current solution, that is $\forall Y_{s,m,b} = \{s, m, b\} \subset F$ if $\exists j | j \in Y_{s,m,b}$ we remove all the elements of $Y_{s,m,b}$ from the problem.

Note that after we checked that the problem is feasible and, consequently, that the facilities we extracted will belong to the solution we are building, we have the possibility to output those plants, giving to the end-user a feedback even if the final solution will be provided in the future.

When we finally determined the facilities positions and sizes that cover all the demand, we solve the "Optimal Allocation Problem" we have seen in Section 3.5 in order to determine the optimal users allocation with the facilities we have open.

[1] Remark that the feasibility here is relative to the original problem specifications but with the new set of facilities, users and the new budget.

Algorithm 3 Progressive Plants Selector

procedure PROGRESSIVEPLANTSSELECTOR(STEPSIZE, REASONTHR, TIMELIMIT)
  $instance \leftarrow$ Build the problem
  while *istance.users* $>$ threshold **do**
    DualAscentProcedure(istance, timelimit)
    $\tilde{f} \leftarrow$ facilities open in DualAscentProcedure sorted by their score
    $oldInstace \leftarrow instance$
    $instance \leftarrow$ remove from $instance$ the top $\lceil stepSize \cdot |\tilde{f}| \rceil = \tilde{\tilde{f}}$ facilities
    if checkInstanceFeasibility($instance$) is true **then**
      Save the $\tilde{\tilde{f}}$ facilities and all the users connected to them
      Remove all the facilities in $\tilde{\tilde{f}}$ from the instance.
      Remove all the users connected to facilities in $\tilde{\tilde{f}}$ from the current instance.
      Remove all the facilities located on the same position of the facilities that are open[2]
      Update the maximum budget:

$$BUDGET \leftarrow BUDGET - \sum_{j \in \tilde{\tilde{f}}} f_j$$

    else
      $instance \leftarrow oldInstance$
      break the cycle
    end if
  end while
  solveToOptimum($instance$)
end procedure

4.3 Algorithm Correctness

For algorithm correctness, we intend the property of our method to return, if it exists, a feasible solution for the input instance. In this section we will prove this statement.

Proposition 3. *Selective Plants Selector Algorithm returns a feasible solution for any feasible instance of MMBPLP.*

Proof. To be feasible, a solution must satisfy all the following constraints:

$$\begin{cases} \sum_{j \in F} x_{ij} = 1 & \forall i \in U & (4.3a) \\ x_{ij} \leq y_j & \forall i \in U \; \forall j \in F & (4.3b) \\ \sum_{j \in F} f_j y_j \leq BUDGET & & (4.3c) \\ \sum_{i \in U} \rho_i x_{ij} \leq MAX\_CAPACITY_j & \forall j \in F & (4.3d) \\ \sum_{i \in U} \rho_i x_{ij} \geq y_j MIN\_CAPACITY_j & \forall j \in F & (4.3e) \\ y_s + y_m + y_b \leq 1 & \forall Y_{s,m,b} = \{s, m, b\} \subset F & (4.3f) \\ 0 \leq x_{ij} \leq 1 & \text{integer} & (4.3g) \\ 0 \leq y_j \leq 1 & \text{integer} & (4.3h) \end{cases}$$

All the constraints from 4.3b to 4.3f are satisfied by any solution $\tilde{S}$ provided by the Dual-Ascent Algorithm, with:

$$\tilde{S} = (\tilde{X}, \tilde{Y}), \tag{4.4}$$

where:

$$\tilde{X} = \{x_{ij} | x_{ij} = 1\} \;\; \tilde{Y} = \{y_j | y_j = 1\}. \tag{4.5}$$

If we take a subset $\dot{Y} \subseteq \tilde{Y}$ and we take the subset of $\dot{X} \subseteq \tilde{X}$ such that:

$$\dot{X} = \{x_{ij} | j \in \dot{Y} \; and \; x_{ij} = 1\}, \tag{4.6}$$

Then all the constraints from 4.3b to 4.3f remain verified. Moreover, every user relative to $\dot{X}$ variables is connected to only one facility.

Now, define

$$\dot{U} = \{i \in U | x_{ij} = 1, \; x_{ij} \in \dot{X}\} \tag{4.7}$$

$$\dot{F} = \{j \in F | y_j = 1, \; y_j \in \dot{Y}\}. \tag{4.8}$$

Consider the problem $\bar{\Phi}$ that can be build from the two subsets $\bar{U} \subseteq U$ and $\bar{F} \subseteq F$ with $\bar{U} = U \setminus \dot{U}$ and $\bar{F} = F \setminus \dot{F}$, obtained by removing those users and facilities that are selected in the reduction phase of PPS. The maximum budget that can be spent in a solution of $\bar{\Phi}$ is:

$$\overline{BUDGET} = BUDGET - \sum_{j \in \dot{F}} f_j \tag{4.9}$$

Then we have two cases:

1. $\bar{\Phi}$ is not feasible: the algorithm will rollback the modifications and it will solve, through a MIP solver, a feasible instance of the original problem.

2. $\bar{\Phi}$ is feasible: define $\bar{S} = (\bar{X}, \bar{Y})$ as a feasible solution of $\bar{\Phi}$. Then we have that the solution $\hat{S}$ defined as:

$$\hat{S} = (\dot{X} \cup \bar{X}, \dot{Y} \cup \bar{Y}) = (\hat{X}, \hat{Y}),$$

is feasible for the original problem, since:

- all the users have been allocated to exactly one facility, so constraints 4.3a are satisfied;
- all the constraints 4.3d, 4.3e, 4.3b and 4.3f are satisfied because both $\hat{S}$ and $\dot{S} = \{\dot{X}, \dot{Y}\}$ satisfy them;
- constraint 4.3c is satisfied since the maximum budget that can be spent is:

$$\begin{aligned} \widehat{BUDGET} &= \overline{BUDGET} + \dot{BUDGET} = \\ &= BUDGET - \sum_{j \in \dot{F}} f_j + \dot{BUDGET} \end{aligned}$$

with:

$$\dot{BUDGET} = \sum_{j \in \dot{F}} f_j \tag{4.10}$$

□

One can say that there is no need to remove a facility completely from the problem because that facility may still have some space available for new allocations.

For example if a facility j with maximum capacity $MAX\_CAPACITY_j$ has been open in an intermediate step of the algorithm, we may have the situation in which only a certain amount of its potential capacity has been filled. If we remove this facility completely we waste some space.

Since we are discussing about NP-Hard problems, we must face the fact that some instances could be very hard to solve. In our particular case we observed that if there is a high variation of capacities among the facilities, then the instances start to become very hard and we are not able to find a solution in useful time (remember that this algorithm was developed to give some feedback in at most one or two minutes).

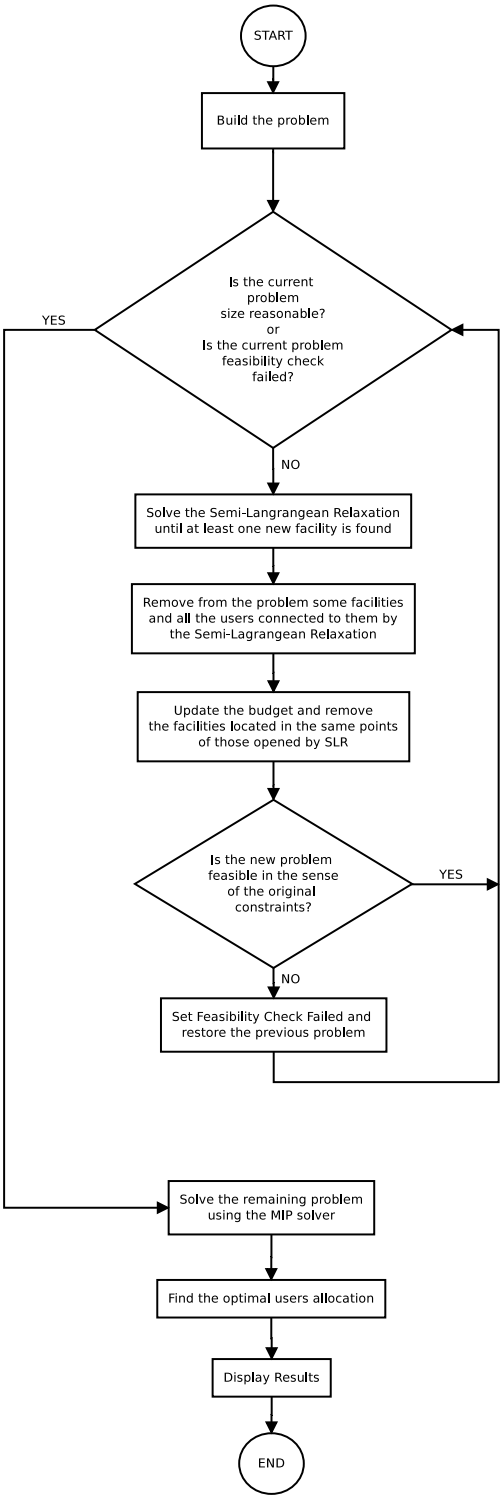

Figure 4.1: Progressive Plants Selector Algorithm flowchart

Chapter 5

Case Study

The theorethical study carried out so far and the strategy developed to solve the MMBPLP, were finalized to an application on a real situation: we needed a method to optimally place medical structures onto the area of an Italian administrative region.

In this Chapter we will describe the case study specifications and how we modeled the problem starting from the data at our disposal. Moreover, we will explain some technical choices on the tools we selected to solve the problem and we will describe the software architecture designed.

Finally, we will report some computational results of both the Progressive Plants Selector Algorithm and the overall system.

5.1 Data

Emilia-Romagna is an administrative Region of Northen Italy, comprising the historical regions of Emilia and Romagna. At present its healthcare is organized into big Hospitals, but the intent of the administration is to build new smaller structures in order achive a better distribution of the medical assistance, mainly for those patients affected by chronic deseases. Some of these structures ("*Health Houses*") have been already built on the region area. What we wanted to do is to position these structures onto the territory by minimizing the overall system's cost.

The healthcare administration of Emilia-Romagna is organized in eleven different administrative sectors, also called *USL*s, each of which has a different jurisdiction area. It's also possible to partition the territory in 36.000 census sections.

Since we wanted to position new facilities, we needed to have a set of possible locations. We decided to use both the set of positions of the plants that are already on the territory and a subset of the census sections, choosing those with the highest population. An evolution of the system (not yet implemented) will be to delegate, to the end-user, the selection of the new plant locations that will be used by the optimization system.

During the optimization we will assume that there are no facilities on the territory, so that our algorithm will try to find the overall optimal solution (without considering that some Health Houses are already positioned). Of course if the end-user wants to keep some facilities from the present configuration, we can force this facility to be open in our model.

The data at our disposal were organized into a well-structured datawarehouse (DWH) containing information about patients clinical history and geographical position. All the users data were, of course, anonymized, but each patient were geolocalized on the region area. Moreover, we knew all the present facilities locations.

5.2 Specifications

Given the 36.000 census sections and all the possible facilities positioning sites, we wanted to decide the plants openings and the opened facilities' capacities in order to minimize the sum of the plants set-up costs and the users' connections costs. We wanted each user to be served by exactly one facility and we knew that there exist a maximum budget that can be spent in new facilities openings. About facilities capacities, we knew that there were three different sizes: small, medium and big facilities.

One can see that these general specification can be modeled by the MMBPLP we described above. It is a good idea to formalize this evidence.

In order to model our problem as a MMBPLP instance, we need to define the following elements:

- the set $U = 1, 2, 3, .., n$ of users;
- the set $F = 1, 2, 3, ..., m$ of possible facilities locations;
- the vector $K = (f_j)$ of fixed costs for setting up plants at sites $j \in F$;
- the matrix $C = [c_{i,j}]$ of transportation costs from $i \in U$ to $j \in F$;
- the vector $D = (d_j),\ j \in F$ of the maximum number of users supported by facility j;
- the vector $T = (t_j),\ j \in F$ of the minimum number of users to be supported by facility j i we want it to be open;
- the vector $R = (r_i),\ i \in U$ for the users' volumes.

In our case the set U will be the set of census sections and the vector R will be formed by using the population for each section. Notice that, since we will not be able to solve instances with 36.000 users and about 1.500 possible facilities locations within a couple of minutes, we will need to reduce the problem size in some way.

The set F will be composed by the union of all the position relative to those facilities that are actually positioned onto Emilia-Romagna's territory, and the subset of those census sections with the hightest population.

The elements of C will be defined as follows:

$$c_{ij} = distance(i,j) \cdot r_i \cdot factor \cdot km\_cost, \quad r_i \in R$$

where:

- $distance(i,j)$ is the Euclidean distance, in km, between the census section i and the facility j centroids;
- $factor \in \mathbb{R}$ is a constant that represents how much "importance" we want to give to the users satisfcation in terms of distance from their facility (if factor is very small we would open few facilities because we give less relevance to the fact that a user must move far away to reach its facility);
- km_cost is how much a user spends to travel for one km;

About K,D and T, they will be defined by the end-user.

System's Reactivity

An important requrement we somehow already highlited, was about the system's response time: we wanted a well-approximated problem solution as fast as possible, with possibly the first feedback from the system in less than 1-2 minutes. The problems we have solved are all NP-Hard and the instances are non-trivial, so this specification has been very hard to be satisfied.

The motivation that led to the definition of this need, is relative to the intent of iCONSULTING to commercialize the final product. Since business users are more confident with Business Intelligence (BI) products, they will expect our system to have a response time comparable to BI tools.

5.3 Development Tools

In this section we are going to discuss about tools and technologies we decided to use to build our system. The general idea was to select open-source and free technologies instead of using commercial products. Moreover, we wanted to be as much "vendor independant" as possibile.

5.3.1 MIP Solver

Since we wanted to solve MIP models, the most important choice was about the MIP solver. Available LP-solvers differ in many ways. They come with different licences and features, for example in terms of how problems can be specified. What follows is a brief description of the most popular tools on the market:

Open-source and free solvers

- GLPK: the GNU Linear Programming Kit is a free and open-source software written in ANSI C.
- LP_SOLVE: is open-source and written in ANSI C as GLPK.
- CLP: is a solver created within the Coin-OR project written in C++. The Coin-OR project aims at creating open software for the operations research community. CLP is used in many othe project within Coin-OR such as SYMPHONY (a ibrary to solve MIP models) or CBC (a LP-based branch-and-cut library).
- SCIP: is a framework for solving integer and constraint programs which is available as a C callable library or a standalone solver. The codebase of SCIP is freely available and the framework ca be used without restrictions for academic research but not for commercial use.

Commercial solvers

- CPLEX: the IBM ILOG CPLEX Optimiaziont Studio is a commercial solver designed to deal with large scale MIP problems. The software supports several interfaces so that it is possible to connect the solver to different program languages. CPLEX is often acknowledged as the best MIP solver on the market.
- Xpress: Xpress is a commercial software that can solve MIP problems. As CPLEX it offers different interfaces to interact with different programming languages.
- Gurobi: is a modern solver for MIP linear (and also for non-linear) mathematical optimization problems.

Among all these tools, our choice fell on the open-source/free softwares, but our system is solver-independent, in the sense that we can replace the MIP solver very easily by implementing a C++ "interface".

During the development process we tested two MIP solvers: GLPK and SCIP (with SoPlex). GLPK gave us very poor results, expcially on non-trivial instances of MMBPLP, so we choose SCIP as the principal MIP Solver.

5.3.2 Hadoop

As we highlighted above, the problem of optimizing the Health Houses positioning onto the Emilia-Romagna area, can be divided into sub-problems relative to each USL in the region. This fact yelds to the possibility of parallelizing our computation by solving on different machines/threads the different USLs sub-problems.

In order to distribute our optimization on different machines without thinking about networking problems, fault tolerance issues and scalability, we decided to use Hadoop for the parallelization.

The Apache Hadoop project develops open-source software for reliable, scalable, distributed computing. The library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming model. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of cluster of computers, each of which may be prone to failures [29].

Hadoop consists of the Hadoop Common package, which provides filesystem and OS level abstractions, a MapReduce engine and the Hadoop Distributed File System (HDFS). The Hadoop Common package contains the necessary JAR files and the scripts needed to start Hadoop.

For effective scheduling of work, every Hadoop-compatible file system should provide location awareness: the name of the rack (more precisely, of the network switch) where a worker node is. Hadoop applications can use this information to run work on the node where the data is, and, failing that, on the same rack-/switch, reducing backbone traffic. HDFS uses this method when replicating data to try to keep different copies of the data on different racks. The goal is to reduce the impact of a rack power outage or switch failure, so that even if these events occur, the data may still be readable.

A small Hadoop cluster includes a single master and multiple worker nodes. The master node consists of a JobTracker, TaskTracker, NameNode and DataNode. A slave or worker node acts as both a DataNode and TaskTracker, though it is possible to have data-only worker nodes and compute-only worker nodes. These are normally used only in nonstandard applications.[30] Hadoop requires Java Runtime Environment (JRE) 1.6 or higher. The standard startup and shutdown scripts require that Secure Shell (ssh) be set up between nodes in the cluster.

In a larger cluster, the HDFS is managed through a dedicated NameNode server to host the file system index, and a secondary NameNode that can generate snapshots of the namenode's memory structures, thus preventing file-system corruption and reducing loss of data. Similarly, a standalone JobTracker server can manage job scheduling. In clusters where the Hadoop MapReduce engine is deployed against an alternate file system, the NameNode, secondary NameNode, and DataNode architecture of HDFS are replaced by the file-system-specific equivalents [31].

Hadoop Data Flow

To understand how we used Hadoop in the optimization process, it is useful to spend some words on MapReduce programming model and Hadoop's Data Flow.

Conceptually, MapReduce programs transform lists of input data elements into lists of output data elements . A MapReduce program will do this twice, using two different list processing idioms: map and reduce. The first phase of a MapReduce program is called mapping. A list of data elements are provided, one at a time, to a function called the Mapper, which transforms each element individually to an output data element (Figure 5.1).

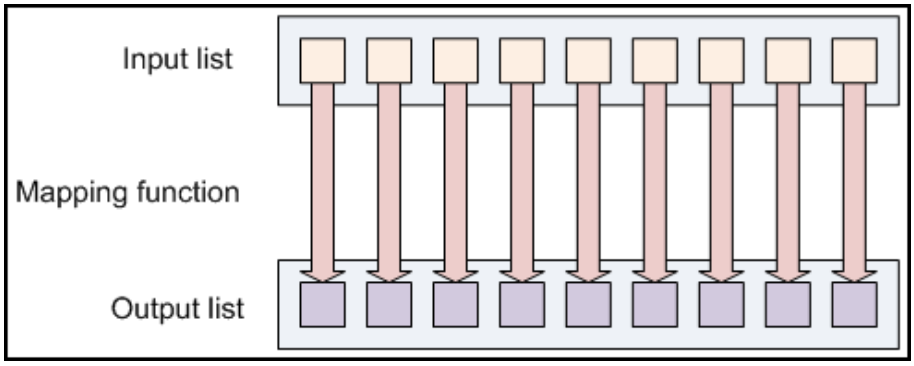

Figure 5.1: Hadoop transforms a list of input data into a list of output data

Reducing lets you aggregate values together. A reducer function receives an iterator of input values from an input list. It then combines these values together, returning a single output value (Figure 5.2).

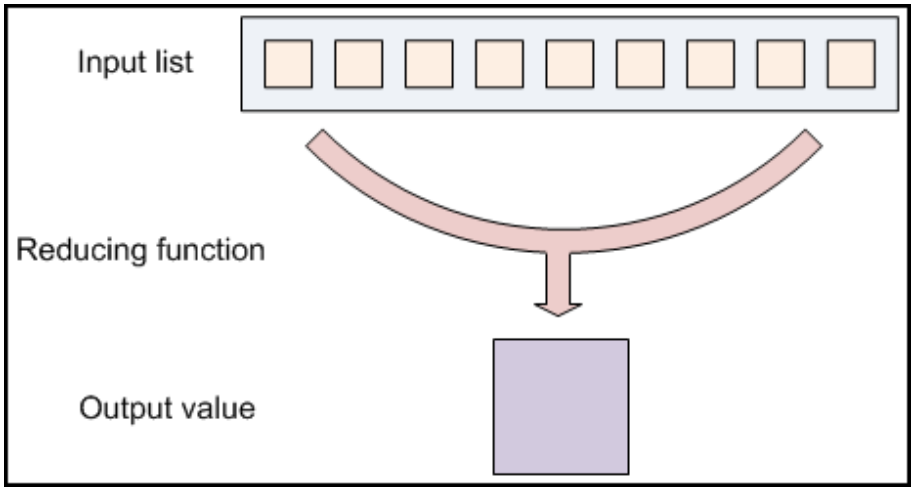

Figure 5.2: The reduce function transforms an input list into a single output value

The Hadoop MapReduce framework takes these concepts and uses them to process large volumes of information. A MapReduce program has two com-

ponents: one that implements the mapper, and another that implements the reducer.

In MapReduce, no value stands on its own. Every value has a key associated with it. Keys identify related values.

The mapping and reducing functions receive not just values, but (key, value) pairs. The output of each of these functions is the same: both a key and a value must be emitted to the next list in the data flow.

A reducing function turns a large list of values into one (or a few) output values. In MapReduce, all of the output values are not usually reduced together. All of the values with the same key are presented to a single reducer together. This is performed independently of any reduce operations occurring on other lists of values, with different keys attached (Figure 5.3).

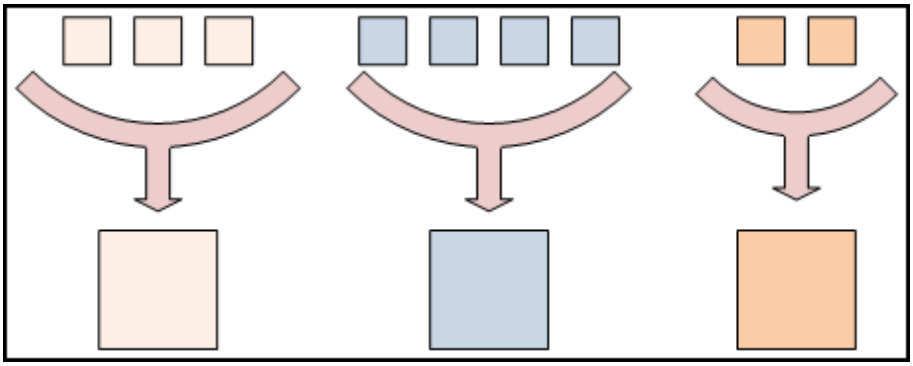

Figure 5.3: Every output element with the same key is processed by the same reduce task. In this picture, different colors mean different keys

When the mapping phase has completed, the intermediate (key, value) pairs must be exchanged between machines to send all values with the same key to a single reducer. The reduce tasks are spread across the same nodes in the cluster as the mappers. This is the only communication step in MapReduce. Individual map tasks do not exchange information with one another, nor are they aware of one another's existence. Similarly, different reduce tasks do not communicate with one another.

MapReduce inputs typically come from input files loaded onto the processing cluster in HDFS. These files are evenly distributed across all cluster's nodes. Running a MapReduce program involves running mapping tasks on many or all of the nodes in the cluster. Each of these mapping tasks is equivalent. Therefore, any mapper can process any input file. Each mapper loads the set of files local to that machine and processes them.

In Figure 5.4 is possible to see the whole map-reduce process described above.

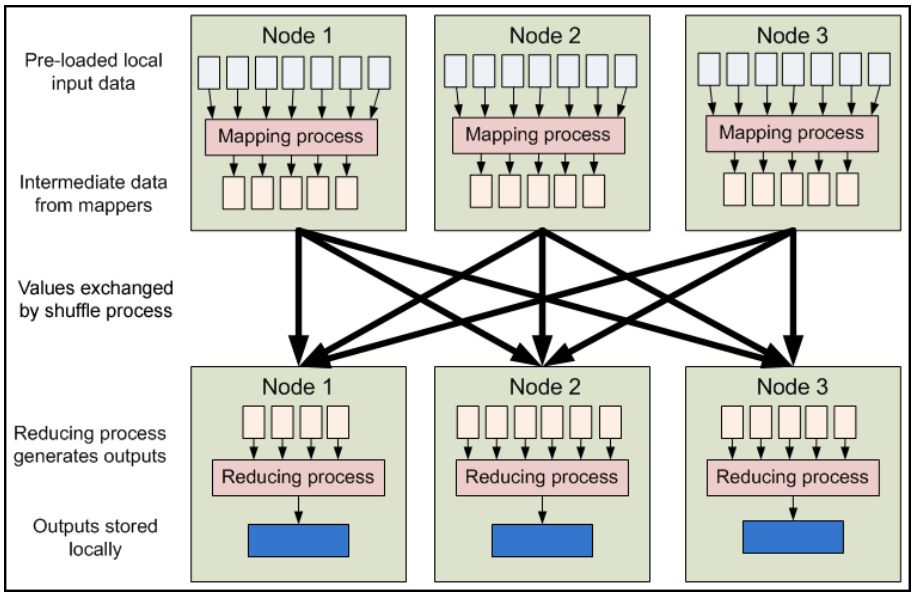

Figure 5.4: The map-reduce process in its entirety

5.4 Oracle Database and Oracle Spatial

All the data needed for the optimization were stored in a Oracle Database with Oracle Spatial extension.

Oracle Spatial (or Spatial) is an integrated set of functions and procedures that enables spatial data to be stored, accessed and analyzed quickly and effeciently in an Oracle database. Spatial data represents the essential location characteristics of real or conceptual objects relate to the real or conceptual space in which they exists. Spatial provides a SQL schema and functions to facilitate the storage, retrieval, udpate and query of collections of spatial features in an Oracle Database; it consists of the following components:

- A schema (MDSYS) that prescribes the storage, syntax and semantics for supported geometrica data types
- A spatial indexing mechanism
- A set of operators and functions for performing area-of-interest queries, spatial join queries and other spatial analysis operations.
- Administrative utilities

The spatial component of a spatial feature is the geoetric representation of its shape in some coordinate space (SRID). This is referred to as its geometry.

Once spatial data is stored in an Oracle database, it can be easily manipulated, retrieved and related to all the other data stored in the database. In our case, we had different types of spatial data, such as:

- The shapes of different sub-areas of Emilia-Romagna (such as the shape of each USL's jurisdiction area);
- Points relative to the geolocalization of clients and facilities
- Compound lines relative to different streets

Oracle Spatial extension has been very useful to let us present our results inside a map: in the database we stored geometries which can be straight-forward drawed in a map, without any particular effort. Notice that we used Oracle for convenience, but our system is independent from the database technology which can be replaced with minimal effort.

5.5 System's Architecture

In this section we will present the architecture of our optimization system. Each step of the optimization process will be described in detail, starting from the request made by the user for the optimization to the output presentation by the map engine.

In the following Figure, we reported all the steps of the optimization process and what follows is their description:

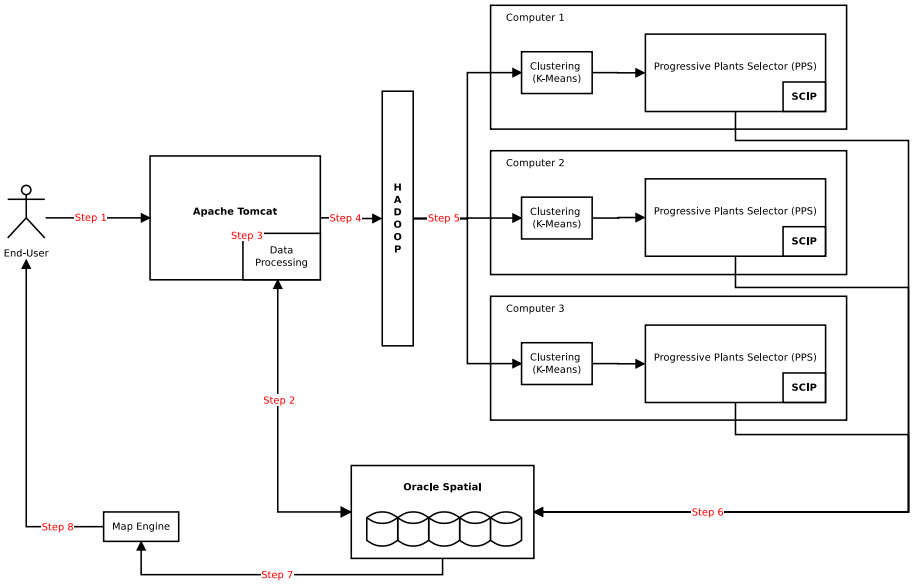

Figure 5.5: The overall optimization system architecture

- Step 1: The user sends the request for optimization with all the parameters relative to capacities and costs.
- Step 2: An Apache Tomcat server retrieves the data needed for the optimization from a database. These data are relative to the census sections centroids and all the possible facilities opening points.
- Step 3: The information obtained from the database is processed and the results of this processing step is written into the "problem's files":
  1. One file for each USL containing information about facility positions and sizes;
  2. One file for each USL with the information about census sections' population and position
  3. Since the optimization procedure (the PPS) will be called by "command line", one single file containing the command to be launched for each USL.

- Step 4: After Tomcat propagated all the problem's files in each machine that is delegated to perform an USL optimization procedure, a Hadoop job is started. The input file for Hadoop is the third file described above; such this file will have the following form:

  ```
  <PATH_TO_EXEC>/mipopt BOLOGNA [parameters]
  <PATH_TO_EXEC>/mipopt FERRARA [parameters]
  ...
  <PATH_TO_EXEC>/mipopt MODENA [parameters]
  ```

  where the first parameter is name of the USL that will be optimized by invoking the relative command from terminal. The MapReduce job that is working on the file described above will proceed as follows:

  - Map Step: The file is splitted into different key-value pairs. The key will be the name of the USL to be optimized while the value will be the command itself;
  - Reduce Step: Since each pair has a different key they will be sent to different Reduce functions, each of which just creates a process using the value of the pair it recived.

  Since we had three (virtual) machines and we forced Hadoop to create exactly one reducer at time on each computer, we had a parallelism-degree of three.

- Step 5 and 6: When the single Hadoop task is started, the real optimization begins. First we need to reduce the size of our input instance. Each USL has something like 5000 census sections (our users) and 150-180 possible facilities locations (counting the triplication of facilities due to the Health House size definition). For these reasons we want to limitate the number of users. To do that we applied a clustering algorithm on census sections.

  The algorithm we used for clustering is K-Means, with the distance between elements defined as the simple Euclidean distance between census sections. K-Means is executed for a fixed number of iterations and the starting points for centroids are randomly choosen among the census sections centroid. Note that the number of clusters desired is an input provided by the end-user. We can assert that it is likely that an execution with higher number of clusters will yeld to a better solution then an execution with a lower number of elements.

  An important observation is that, after K-Means has completed, the system starts to give an output to the user (the generated clusters).

  After K-Means has been executed, we start the Progressive Plants Selector where the set of users is the set of clusters we just generated.

  When the PPS finds a facility to be open, it writes the facility position and size in the Database. At this point the data is already ready to be displayed on the map.

The procedure just described continues until all the USLs have been optimized.

- Step 7 and 8: Since the Map Engine (Oracle Map Viewer) and the database are made to work together, once spatial data are written in the DB by the optimizer, then they are displayed straight forward on the map with very little effort.

Finally we show some screenshots from our application, starting from the empty scenario until all the USLs have been optimized.

Our starting point are the USLs and in Figure 5.6 we can see the shapes which represent the single USLs. Note that if we consider the union of those shapes, we get the Emila-Romagna's area.

When the user asks for the optimization, the system starts the clustering procedure. When K-means has been executed, clusters are written in the database and displayed to the client. Since showing the clusters involves Oracle Spatial functions dedicated to the aggregation of shapes (a cluster is an aggregation of census sections and, to display the cluster, we aggregate the census sections shapes), this step may take some minutes; this is the reason why the user will see clusters while they are being written in the database and not all at the same time. In Figure 5.7 and 5.8, we can see the clusters displaying process, while Figure 5.9 shows a different rapresentation of this aggregations: a circle positioned on the cluster's centroid which size and color depends on cluster's population.

Notice that the system starts K-means on three different USLs at time: these are the three Hadoop's reduce tasks on the three virtual machines.

When the clustering has finished for a particular USL, the Progressive Plants Selector starts its search for the facilities to open. Since those structures are found progressively, the user interface can display them while the algorithm is still running. In Figure 5.10 we show an instant in which the system found a certain number of facility. We represent with a blu square the small facilities, yellow square the medium facilities and with a red square the big facilities.

When all the users have been allocated in the PPS, the algorithm searches for the best users allocation. As usual this step is performed for each USL and in Figure 5.11 we can see the connections found for some USL. Once the Progressive Plants Selector found the connections for every user, the USL subproblem has been fully solved.

Finally, in Figure 6.13, we can see the solution found by our algorithm for the full Emilia-Romagna area with 450 users and 150 positioning sites for each USL.

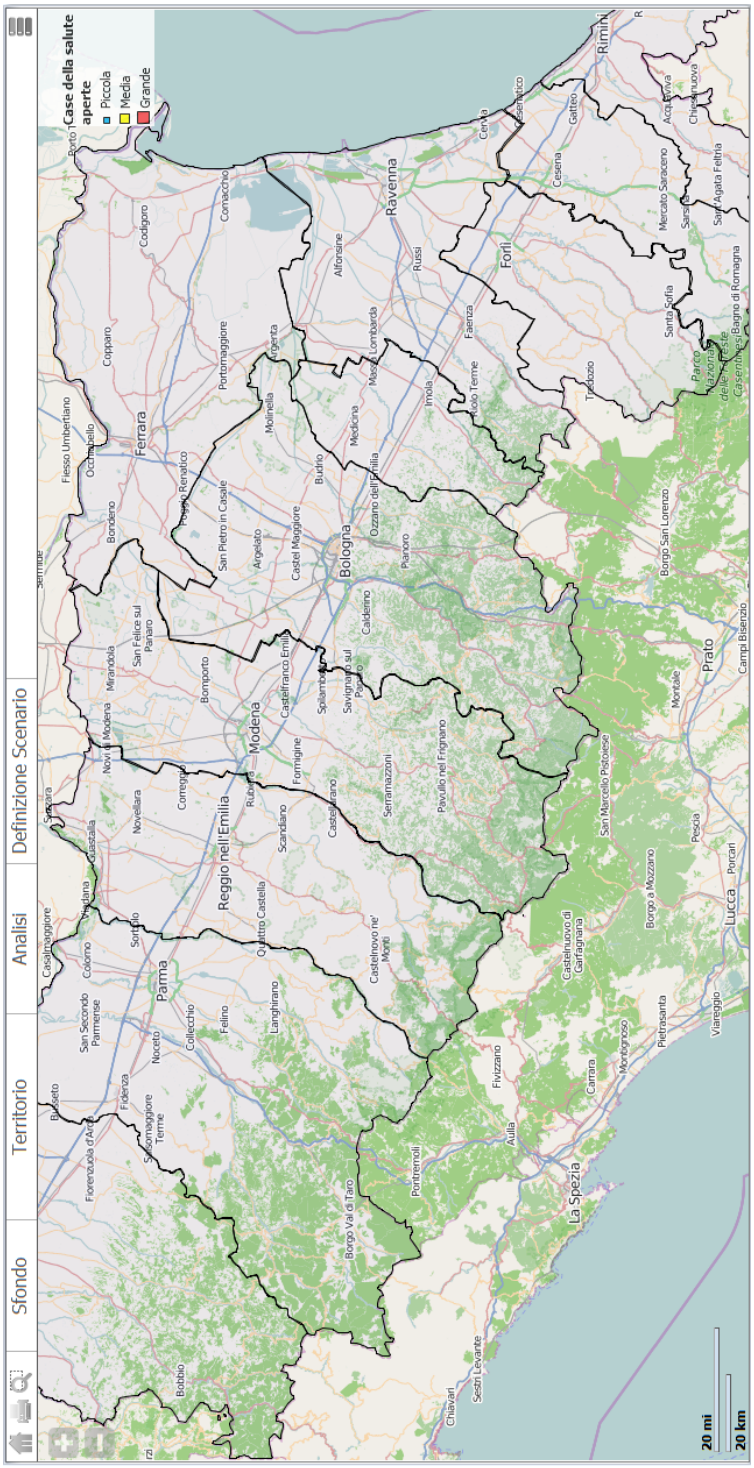

Figure 5.6: Emila-Romagna partitioned into the different USLs

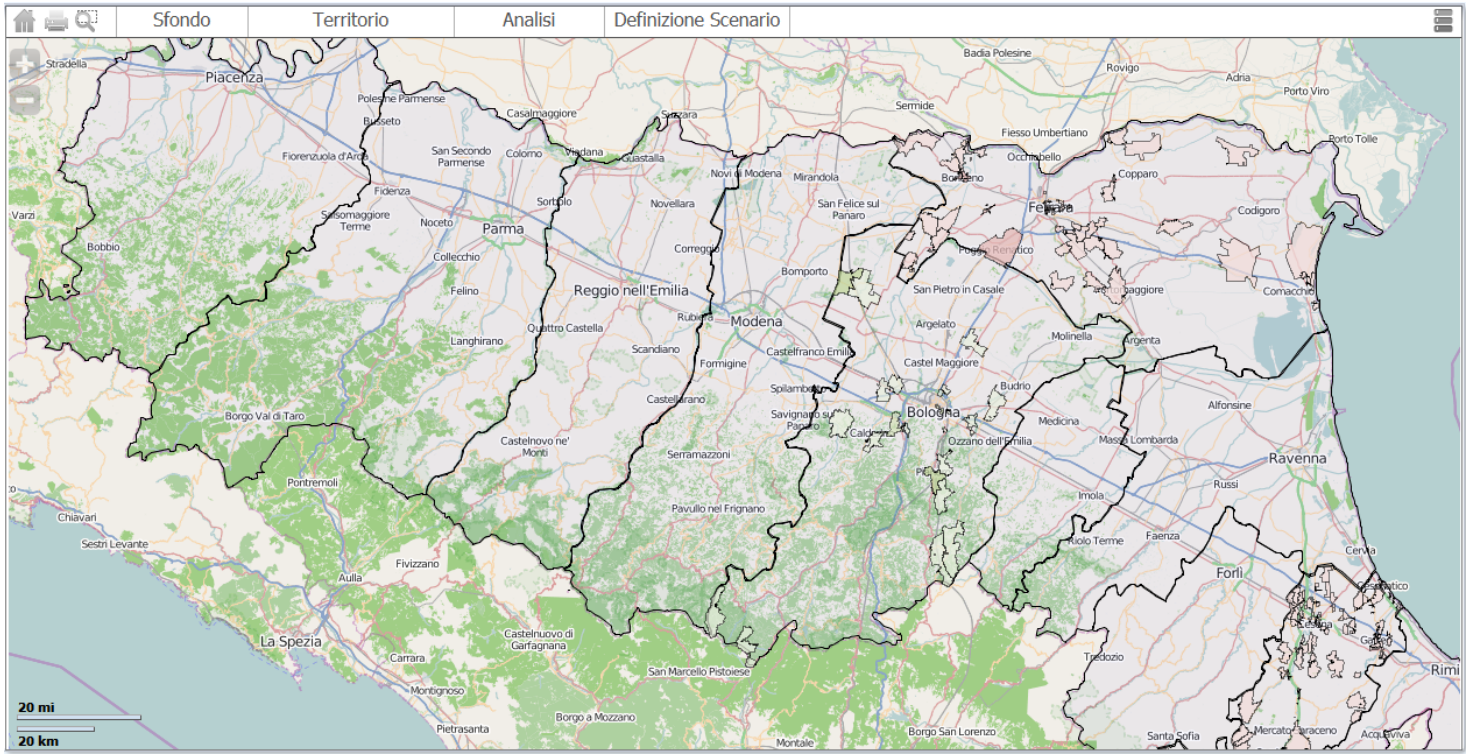

Figure 5.7: Clusters found by K-means begin to be displayed to the user

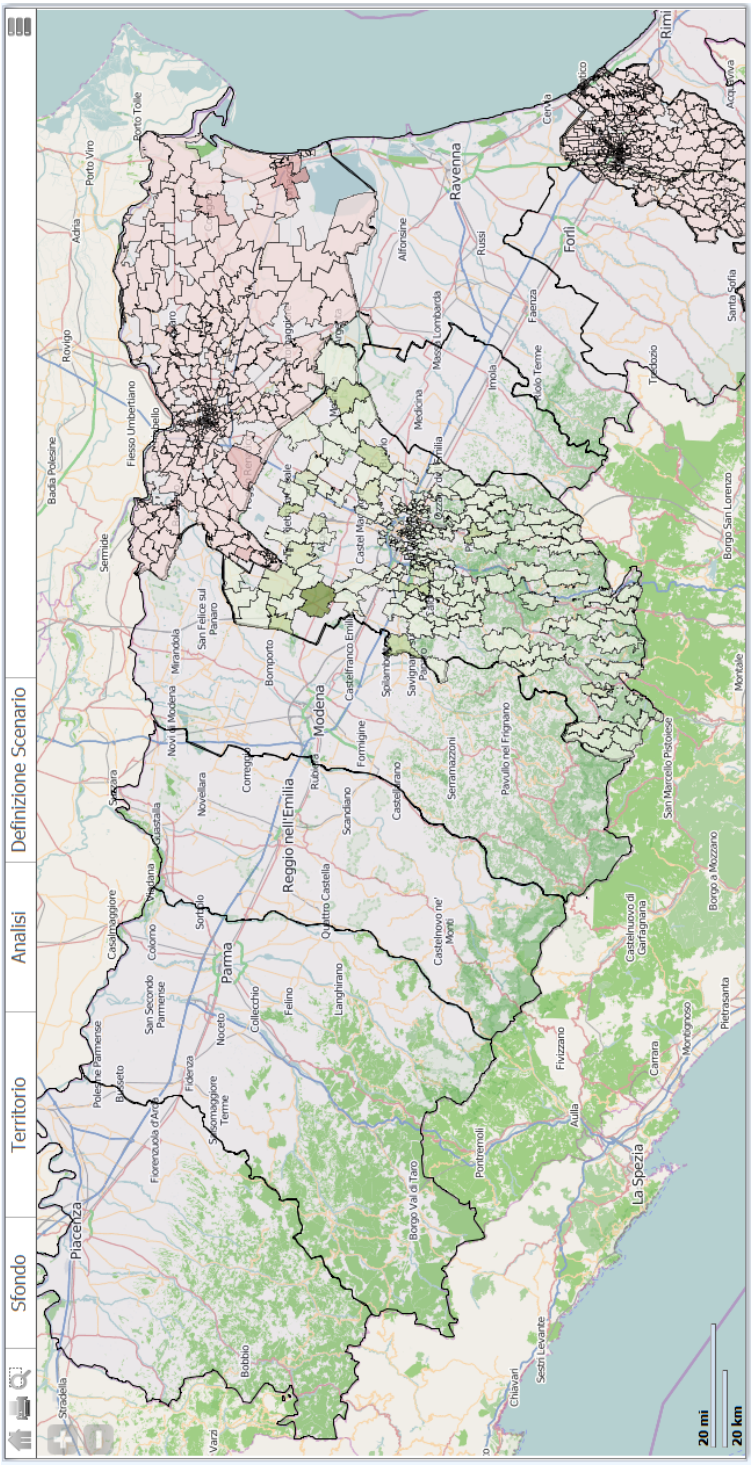

Figure 5.8: More clusters have been generated

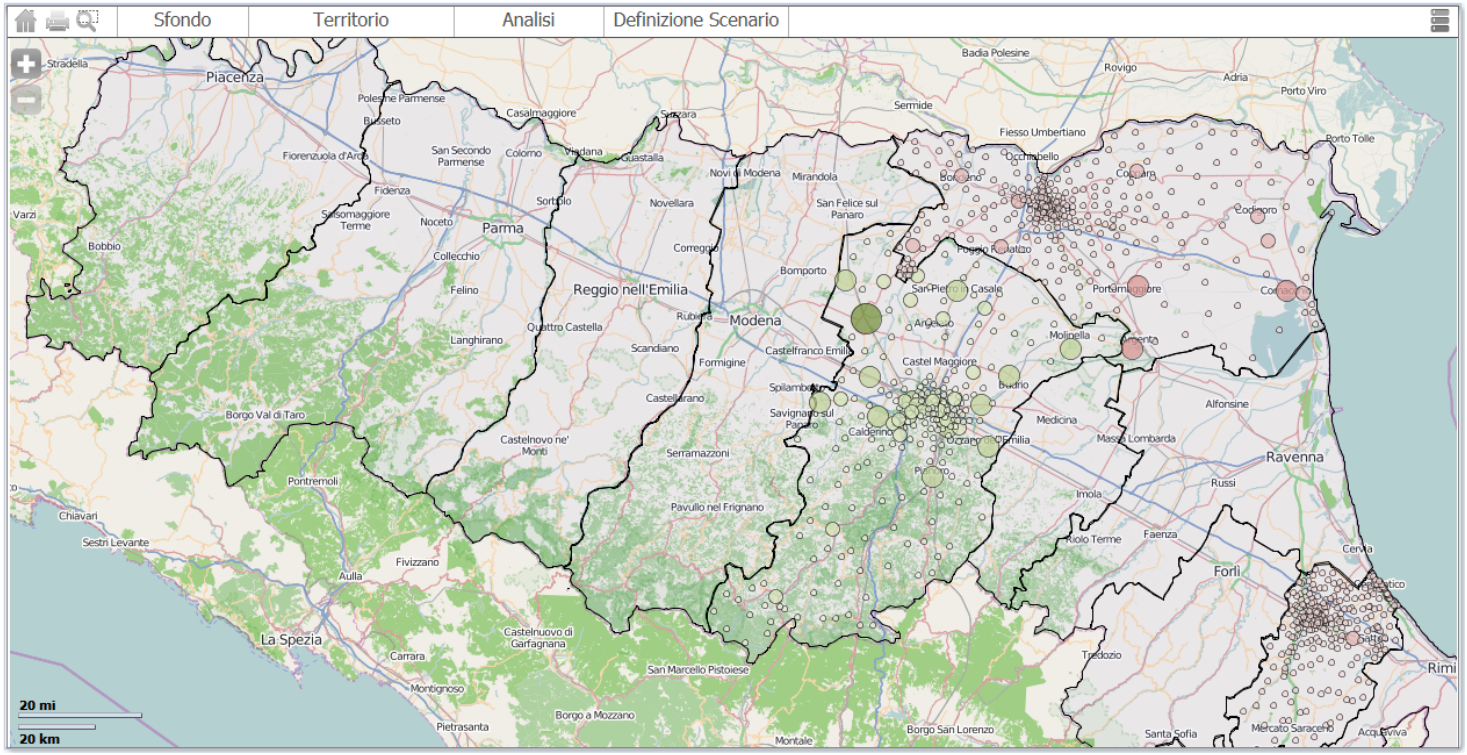

Figure 5.9: Bubble rapresentation for clusters

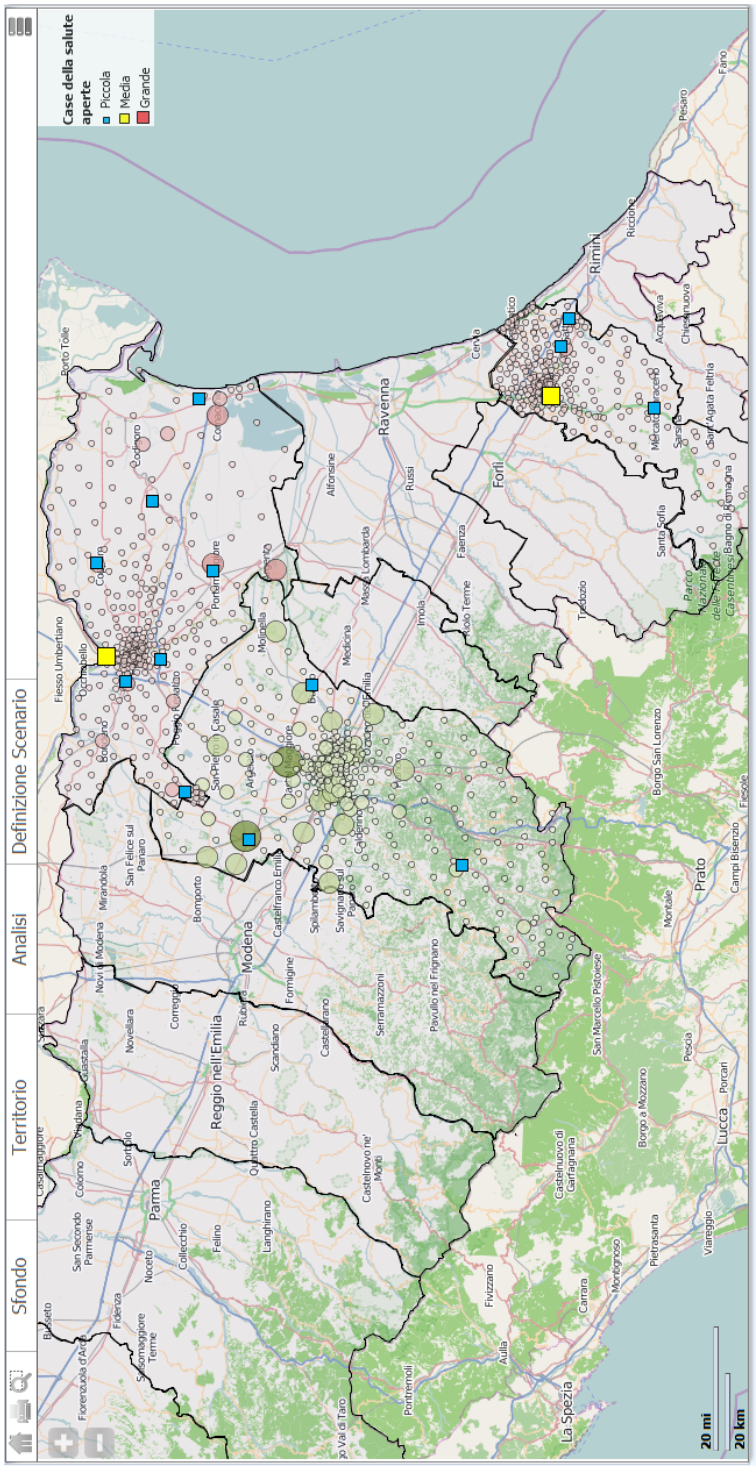

Figure 5.10: The PPS found some facilities. We represent with a blu square the small facilities, yellow square the medium facilities and with a red square the big facilities

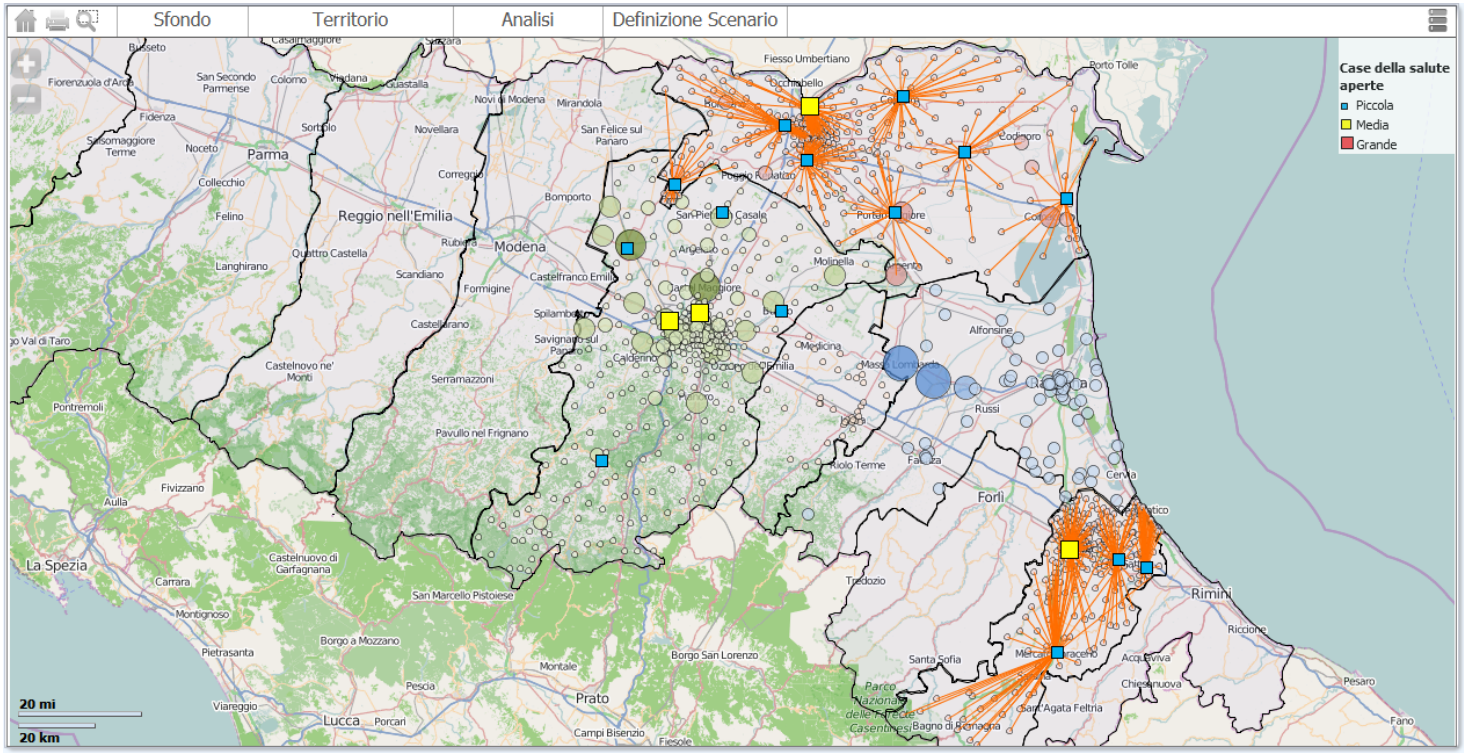

Figure 5.11: The PPS found all the connections for some USLs

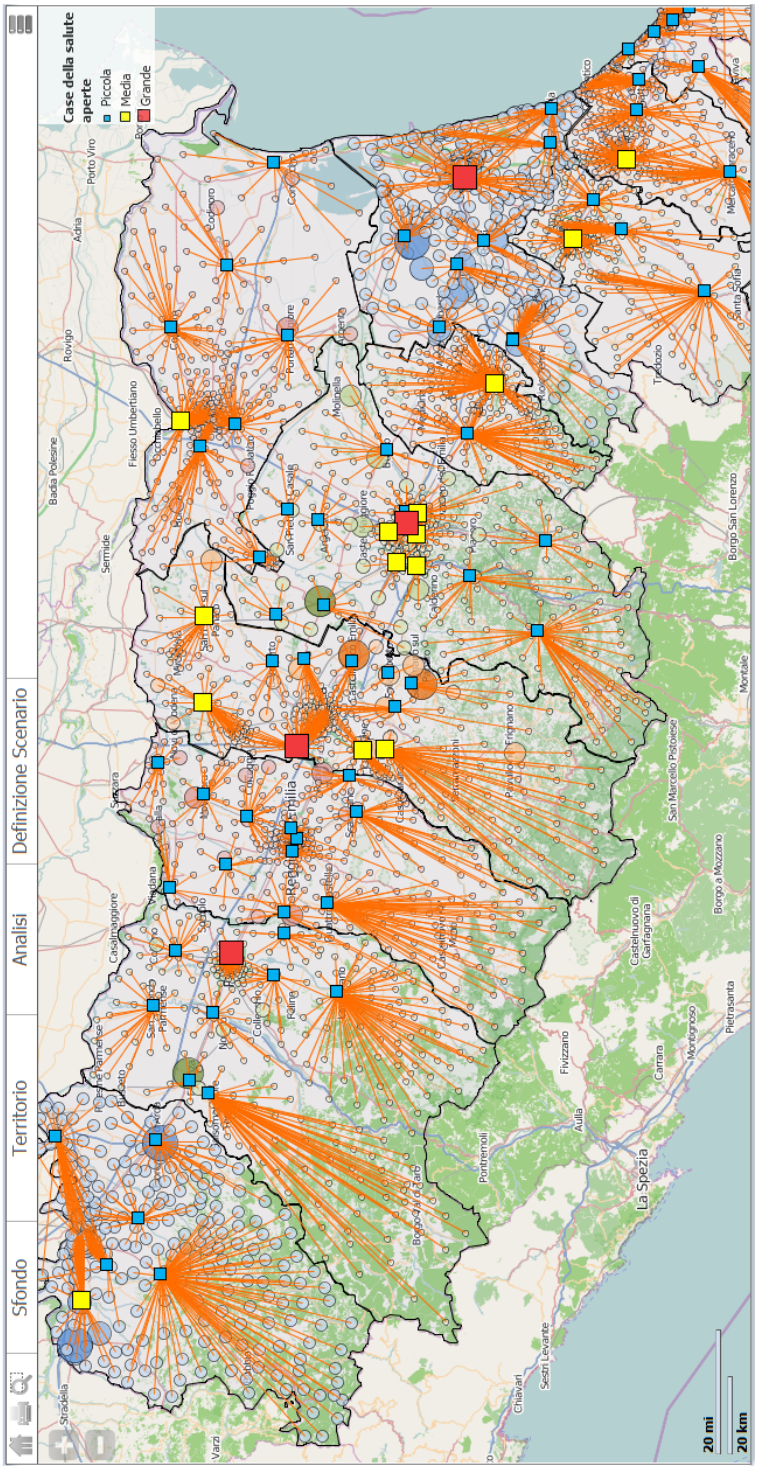

Figure 5.12: The solution found by PPS for the full Emilia-Romagna area

Chapter 6

Computational Results

In this Chapter we are going to present some results to show the performances of our methods. We will first show the behavior of our algorithm on randomly generated MMBPLP instances. Generating random problems allowed us to dermine those factors that make the instance harder or simpler to solve.

In the second part we will expose the computational results obtained by solving the real instances, id est those instaces generated starting from our real problem. Luckily these are much more simple than the random ones, and we will optimize problems involving a high number of users and facilities.

In addition to showing the Progressive Plants Selector performances and emphasizing the differences from an optimization performed by using the raw MIP Solver, we will show the results obtained by using the different MIP model formulations we have seen in Chapter 3.

All the tests were carried out on an Ubuntu 13.10 with LXDE GUI system, equipped with an Intel i5 480m processor (1st generation Intel's i5) and 4GB DDR3 RAM.

6.1 Random Instances

6.1.1 Problem Hardness

To give an idea on "how hard" our problem is, we show the performances of the optimization obtained by using SCIP with the MMBPLP expressed using the following MIP model:

$$x_{ij} = \begin{cases} 1 & \text{if the user } i \text{ is served by facility } j \\ 0 & \text{otherwise} \end{cases}$$

$$y_j = \begin{cases} 1 & \text{if the facility } j \text{ is opened} \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{cases} min \sum_{i \in U} \sum_{j \in F} x_{ij} c_{ij} + \sum_{j \in F} y_j f_j & & \text{(6.1a)} \\ s.t. \sum_{j \in F} x_{ij} = 1 & \forall i \in U & \text{(6.1b)} \\ \sum_{j \in F} f_j y_j \leq BUDGET & & \text{(6.1c)} \\ \sum_{i \in U} \rho_i x_{ij} \leq y_j MAX\_CAPACITY_j & \forall j \in F & \text{(6.1d)} \\ \sum_{i \in U} \rho_i x_{ij} \geq y_j MIN\_CAPACITY_j & \forall j \in F & \text{(6.1e)} \\ y_s + y_m + y_b \leq 1 & \forall Y_{s,m,b} = \{s, m, b\} & \text{(6.1f)} \\ 0 \leq x_{ij} \leq 1 & \text{integer} & \text{(6.1g)} \\ 0 \leq y_j \leq 1 & \text{integer} & \text{(6.1h)} \end{cases}$$

Instances Description

The objects involved in the instances we used for this test are generated as follows:

1. Facilities maximum and minimum capacities are fixed and equal for each facility of the same size:

   - Small Facilities

     $$(MIN\_CAPACITY, MAX\_CAPACITY) = (100, 200)$$

   - Medium Facilities

     $$(MIN\_CAPACITY, MAX\_CAPACITY) = (150, 450)$$

- Big Facilities

$$(MIN\_CAPACITY, MAX\_CAPACITY) = (200, 600)$$

2. Facilities costs are fixed:

$$\begin{cases} Small\ Facilities = 1000 \\ Medium\ Facilities = 2000 \\ Big\ Facilities = 3000 \end{cases}$$

3. Users volumes ρ_i, are randomly generated with $\rho_i \in [1, 10],\ \rho_i \in \mathbb{N}$

4. Users and Facilities positions are randomly generated in a 100×100 square. The maximum distance between an user and a facility is:

$$\sqrt{100^2 + 100^2} = 141.42 \tag{6.2}$$

5. The budget is set to:

$$BUDGET = Facilities\ Number \times Medium\ Facility\ Cost \tag{6.3}$$

Results

In the following Table one can see the execution times and the gap obtained by optimizing the instances described above. The timelimit was set to 3600 seconds. Moreover we highlighted the average amount of memory used during the optimization:

Seed	Bud	Usrs	yVars	Tim	SVS	Mem	GapS
999	50000	100	75	4.2	5014.4	49	0.00
2553	100000	400	150	186.4	11964.0	320	0.00
3330	150000	400	225	3600.0	15068.3	500	0.47
4107	200000	400	300	3065.6	22896.8	675	0.00
4884	140000	500	210	3600.0	27701.5	600	0.05
5661	150000	600	225	3600.0	17698.3	750	0.04
7215	300000	800	450	3600.0	72894.0	1900	∞
7992	200000	1000	300	3600.0	26785.0	1650	0.07

Table 6.1: Columns description: **Seed**: the seed used to generate tha random graph; **yVrs**: the number of y-variables in the model; **Bud**: the budget; **Usrs**: the number of users in the model; **Tim**: the time needed to PPS solve the problem; **SVS**: the value of the solution found by SCIP; **Mem**: the average amount of RAM memory used; **GapS**: the gap for SCIP

Looking at these results one can see that the instances becomes harder when the number of y variables starts growing: SCIP needs only 186 seconds to solve the 400×150 instance to optimum, but reaches the timelimit in almost all the

successive problems. We give more relevance to y variables because in our case $\#users > \#facilities$, so each new y-variable (i.e. a new facility positioning site) adds more edges to the problem's graph than an user.

In the bigger instances, i.e. the one with seed 7215, to represent the model for a graph of $800 \times 450 = 360000$ edges, SCIP needs about 2 GB of RAM.

One big advantage of the Progressive Plants Selector, will be the lower amount of memory needed to represent the problem in the computer main memory.

6.1.2 SCIP vs. Progressive Plants Selector

In order to highlight the differences between a raw SCIP optimization and the PPS we needed to use the same metric of costs. Considering only the solution quality (i.e. the duality gap) was not a good idea, because giving SCIP a sufficient amount of time, we will always find the optimal solution; moreover the main purpose of PPS is to provide a good solution to our problem using a small amount of time, so we need to involve in its evaluation both those factors.

What we did to compare the two methods, was to first solve the instances using PPS and then start SCIP by using the time used by PPS as timelimit for SCIP (e.g. if PPS needed 200 seconds to solve a particular instance, we used SCIP for 200 seconds to solve the same instance). We are somehow challenging SCIP to do better than PPS using the same amount of time.

Notice that once we obtained the PPS solution value, the SCIP solution value and the SCIP duality gap, we are able to compute the duality gap of the PPS solution.

6.1.3 Playing with the different factors

After we tested our algorithm with generally random generated instances, we started to tune all the problem factors to understand how much each element is involved in MMPLP instance hardness.

We discovered that the problem size, to be intended as the number of edges, users and facilities, is not a good indicator to represent the difficiulty of an instance. Other factors, such as the users volumes or the budget, if badly chosen, can make the problem much harder to solve than increasing the number of users or facilities.

Of course once the instances start to become very big (1500 users and 200×3 facilities, or even smaller), the problem is still very hard to solve even if we make the best choice in terms of the other factors.

In all the tests that follow, we will show a comparison between SCIP and PPS, to remark the advantages obtained by using the second method in terms of rapidity.

Users volumes

To understand how much the users volumes are involved in the instance hardness, we decided to set the budget to a very high value, so that it was possible to open all the facilities. We made this choice to reduce as much as possible the influence of this factor on problem hardness. The following Table collects the results obtained by solving instances generated using the same criteria of Section 6.1.1 with the key differences explained above.

In the following Table, we show the comparison described above using instances generated as in Section 6.1.1, with the difference that, in each instance, every user i has the same volume defined as:

$$\rho_i = \rho \ \ \forall i \in U \ with \ \rho \in \{1, 3, 5\} \tag{6.4}$$

Seed	yVars	ρ	Usrs	Tim	SVS	SVP	GapS	GapP
9111	150	1	500	45.23	∞	15456.6	∞	∞
18000	150	1	500	45.14	∞	13763.2	∞	∞
26889	150	1	500	46.54	∞	14206.7	∞	∞
35778	150	1	500	45.14	∞	13976.7	∞	∞
44667	150	1	500	52.86	13680.0	13892.8	0.00	0.01
53556	150	1	500	48.42	∞	14233.3	∞	∞
62445	150	1	500	44.87	∞	13972.8	∞	∞
71334	150	1	500	53.21	13586.5	13670.8	0.00	0.01
80223	150	1	500	44.90	∞	13843.7	∞	∞
89112	150	1	500	54.02	29022.6	14037.6	∞	∞
9111	150	3	500	96.29	35893.0	15652.6	∞	∞
18000	150	3	500	115.22	34344.6	15363.2	∞	∞
26889	150	3	500	138.24	34436.8	15576.2	∞	∞
35778	150	3	500	109.01	32991.9	14899.8	∞	∞
44667	150	3	500	130.17	38061.9	15656.4	∞	∞
53556	150	3	500	139.19	31077.4	15468.1	∞	∞
62445	150	3	500	100.21	35072.2	15315.6	∞	∞
71334	150	3	500	125.23	32980.2	15243.0	∞	∞
80223	150	3	500	112.21	33423.8	15182.1	∞	∞
89112	150	3	500	98.16	33174.8	15030.3	∞	∞
9111	150	5	500	298.11	38528.0	18579.0	1.13	0.03
18000	150	5	500	265.12	39096.6	18629.1	1.15	0.02
26889	150	5	500	412.18	19640.7	18942.7	0.06	0.02
35778	150	5	500	427.83	19333.5	18672.2	0.06	0.02
44667	150	5	500	436.14	20538.2	18922.4	0.11	0.02
53556	150	5	500	444.36	20031.7	18656.4	0.10	0.03
62445	150	5	500	320.12	19405.7	18616.9	0.06	0.02
71334	150	5	500	459.98	20627.9	19158.2	0.11	0.03
80223	150	5	500	263.13	38460.9	18577.0	1.11	0.02
89112	150	5	500	443.04	19663.0	18996.5	0.07	0.03
9111	150	[1,10]	500	286.81	43231.2	18631.6	∞	∞
18000	150	[1,10]	500	265.11	20922.5	21819.1	0.00	0.04
26889	150	[1,10]	500	121.24	34338.3	15569.8	∞	∞
35778	150	[1,10]	500	46.13	∞	14234.9	∞	∞
44667	150	[1,10]	500	291.16	39727.4	18893.7	1.15	0.02
53556	150	[1,10]	500	448.01	20285.4	18826.3	0.11	0.03
62445	150	[1,10]	500	1094.39	21128.8	21982.5	0.01	0.05
71334	150	[1,10]	500	925.30	20847.7	21137.2	0.01	0.03
80223	150	[1,10]	500	206.92	36431.6	16732.9	1.20	0.01
89112	150	[1,10]	500	244.33	20473.2	20963.5	0.00	0.02

Table 6.2: Columns description: **Seed**: the seed used to generate tha random graph; **yVrs**: the number of y-variables in the model; ρ: the volume of each user; **Usrs**: the number of users in the model; **Tim**: the time needed to PPS solve the problem; **SVS**: the value of the soluion found by SCIP; **SVP**: the value of the solution found by PPS; **GapS**: the gap for SCIP; **GapP**: the gap for PPS

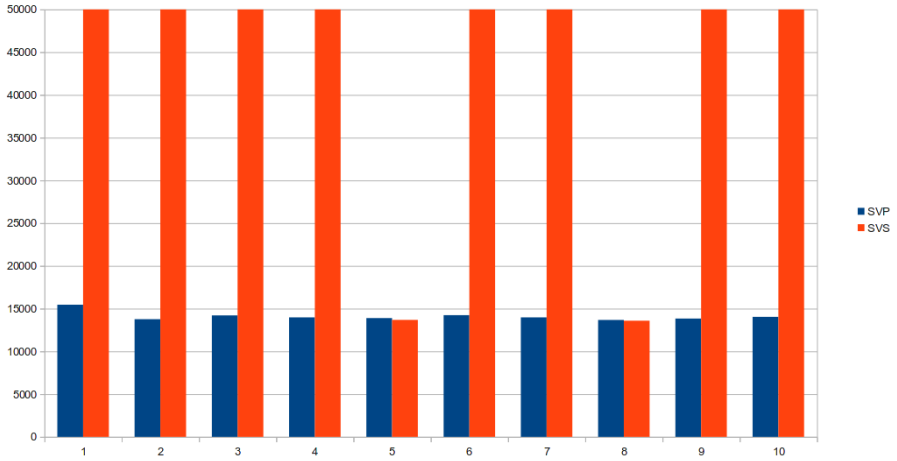

Figure 6.1: In the image it is possible to see the solution values found by SCIP (in orange) and by PPS (in blue) for instances with user's volumes equals to 1 with instances generated as described in Section 6.1.3. In most cases SCIP is not able to find any feasible solution within the same time of PPS and, when it does, the two values are almost equal.

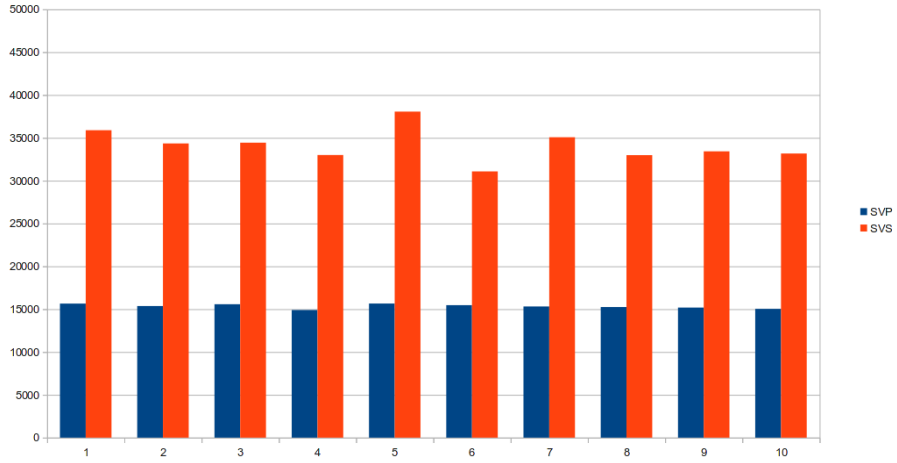

Figure 6.2: In the image it is possible to see the solution values found by SCIP (in orange) and by PPS (in blue) for instances with user's volumes equals to 3 with instances generated as described in Section 6.1.3. In this case SCIP always finds a solution that doubles the costs of the one discovered by PPS.

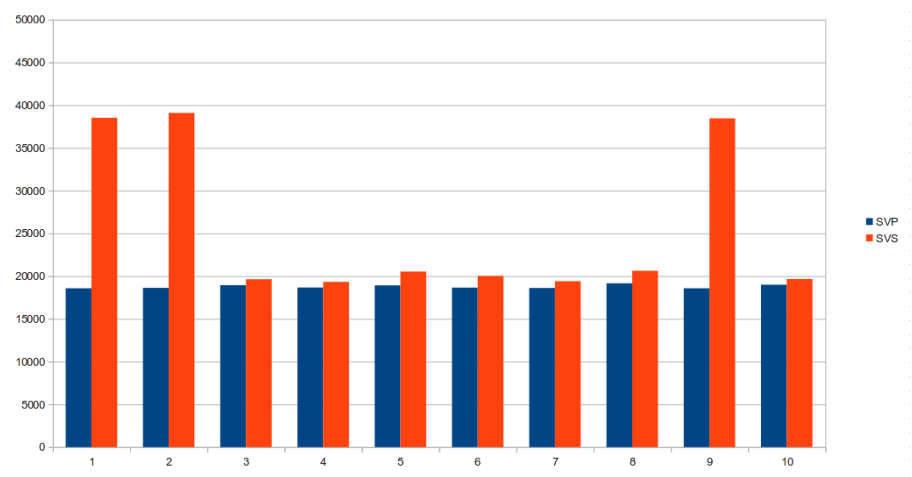

Figure 6.3: In the image it is possible to see the solution values found by SCIP (in orange) and by PPS (in blue) for instances with user's volumes equals to 5 with instances generated as described in Section 6.1.3. In this case SCIP always finds a solution that is worse than the one discovered by PPS.

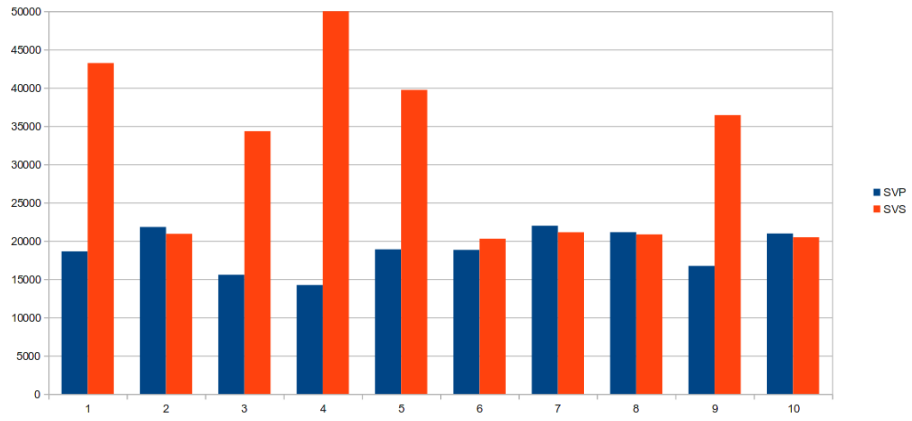

Figure 6.4: In the image it is possible to see the solution values found by SCIP (in orange) and by PPS (in blue) for instances with user's volumes randomly distributed in $[1, 10]$, with instances generated as described in Section 6.1.3. In this case SCIP often finds a solution that is worse than the one discovered by PPS.

Like before, we used the time needed to the PPS as timelimit for the SCIP-based solution method. Consider that when we set $SolVal(SCIP) = \infty$ we want to express the fact that SCIP did not find any solution for the instance within the timelimit. In this case we are not able to compute both the duality gap of SCIP and the duality gap of the PPS. If $SolVal(SCIP) \neq \infty$ but $Gap(SCIP) = \infty$, SCIP was not capable to find a lower bound for that instance, so that it is impossible to compute the duality gap for SCIP and PPS both.

Table 6.2 shows that the users volumes highly influence the overall solution process performance. If users capacities are all set to 1 the Progressive Plants Selector is able to find a good solution in very little time (we know for sure that the solution is good only when we have the duality gap found by SCIP).

If we increase the users capacities, the problem becomes harder, and we were not able to solve in useful time these instances with users capacity set to 7.

Moreover, we can see that the duality gap achieved by PPS is much better than the one obtained using SCIP with the timeout described above. Even when SCIP finds a feasible solution, but does not find the lower bound to compute the gap, we can see that PPS discovers a feasible solution with much better than the one found by the other method.

We also noticed that those instances with random generated users volumes (the last block of the Table) are simpler than those with all the users with the same volume, when it is different from 1.

6.2 The Feasibility Check

In Section 3.4 we described a sufficient condition to check the feasibility of a generic instance of MMBPLP, stating that applying this condtion is much faster than using SCIP to check the problem feasibility.

We now present some results that prove our statement. Remind that the feasibility checking procedure is often called during PPS execution, and it must be as fast as possible.

In the following table we report a comparison of the executions times of two methods: the feasibility check described above and a feasibility test directly performed by using SCIP. In this second case we optimized the SCIP parameters in order to find a feasible solution to our problem as fast as possible: whenever SCIP finds a feasible solution, the problem feasibility is proved and the execution can be stopped.

Seed	Bud	yVrs	Usrs	TimP	TimS	Diff%
1583	375000	225	100	0.05	14.39	286.80
1668	375000	225	100	0.04	14.42	359.50
1753	375000	225	100	0.07	14.44	205.28
1838	500000	300	100	0.07	18.89	268.85
1923	500000	300	100	0.06	18.86	313.83
2008	500000	300	100	0.06	18.74	311.33
2093	500000	300	100	0.06	18.89	313.83
2178	500000	300	100	0.07	18.85	268.28
2263	500000	300	100	0.06	18.88	313.66
2348	500000	300	100	0.06	18081.00	301349
2433	500000	300	100	0.06	18.87	313.05
2518	500000	300	100	0.07	18.76	267.00
2603	625000	375	100	0.07	24.04	342.43
2688	625000	375	100	0.07	23.41	333.42
2773	625000	375	100	0.03	23.77	791.33
2858	625000	375	100	0.11	23.58	213.36
2943	625000	375	100	0.08	22.78	283.75
3028	625000	375	100	0.06	23.72	394.33
3113	625000	375	100	0.08	23.65	294.62
3198	625000	375	100	0.08	23.62	294.25
3283	625000	375	100	0.09	23.93	264.88
3368	750000	450	100	0.11	28.95	262.18
3453	750000	450	100	0.15	29.65	196.66
1073	375000	225	300	0.13	47.73	366.15

1158	375000	225	300	0.13	42.85	328.61
1243	375000	225	300	0.13	48.42	371.46
1328	375000	225	300	0.13	46.01	352.92
1413	375000	225	300	0.13	43.37	332.61
1498	375000	225	300	0.13	46.47	356.46
1583	375000	225	300	0.13	46.86	359.46
1668	375000	225	300	0.13	46.27	354.92
1753	375000	225	300	0.14	46.62	332.00
1838	500000	300	300	0.13	55.65	427.07
1923	500000	300	300	0.14	62.56	445.85
2008	500000	300	300	0.14	56.45	402.21
2093	500000	300	300	0.14	63.26	450.85
2178	500000	300	300	0.15	56.35	374.66
2263	500000	300	300	0.15	62.65	416.66

Table 6.3: Columns description: **Seed**: the seed used to generate tha random graph; **Bud**: the budget; **yVrs**: the number of y-variables in the model; **Usrs**: the number of users in the model; **TimP**: the time needed to PPS to check the feasibility; **TimS**: the time needed to SCIP to check the feasibility; **Diff**%: the percentage difference between TimP and TimS

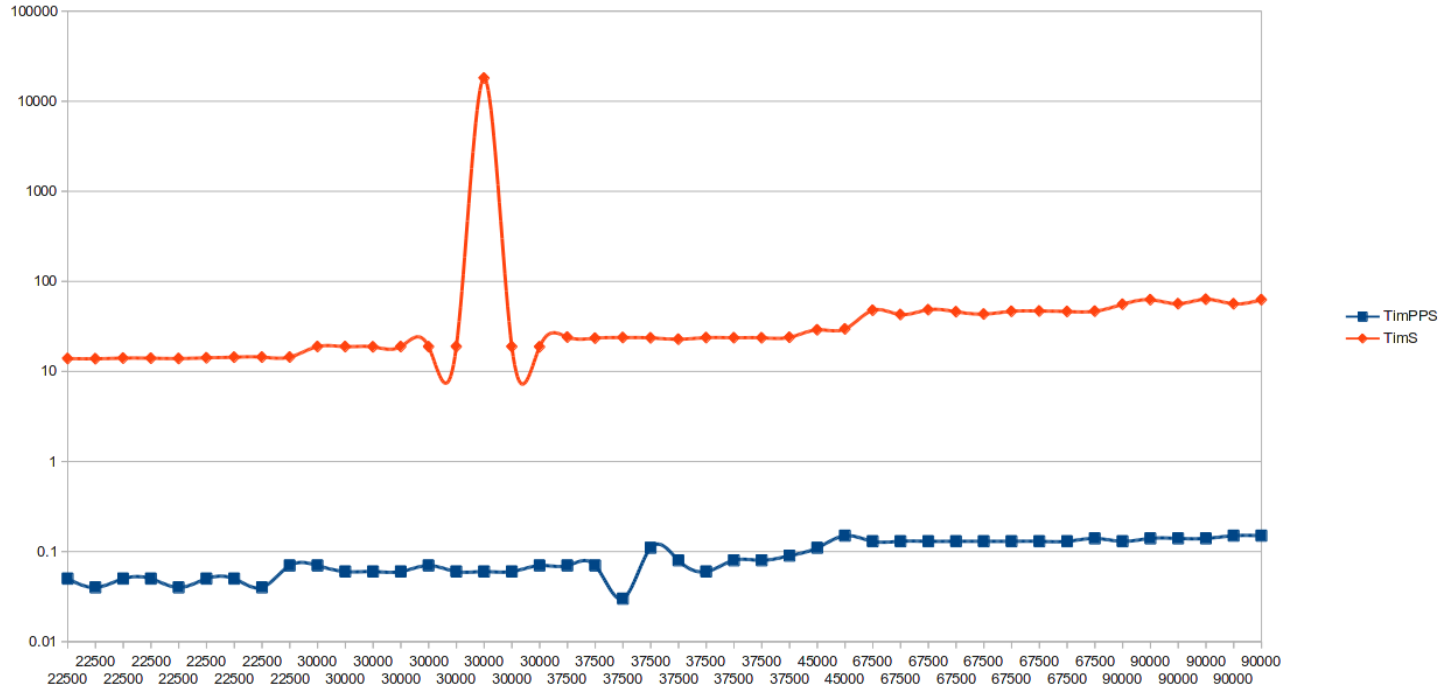

Figure 6.5: The orange line represents the time needed to SCIP to check the feasibility of our problem (that is it is the time it needs to find a feasible solution). The blue line is relative to our Feasibility Check. Since the difference is huge, we needed to represent the data in logaritmic scale.

6.3 Different Graphs

Since here we analyzed the performances of our model by using randomly generated graphs where both facilities and users were evenly distributed inside a square. In this section we will show the results obtained by optimizing different types of graphs. We will notice that these structures will increase/decrease the problem hardness.

6.3.1 Facilities Sorrounded by Users

In these instances facilities are placed inside a square and most of the users are generated all around that square (a small number of users is generated inside the square of facilities). An example of this kind of graph is shown in Figure 6.6 and Figure 6.7.

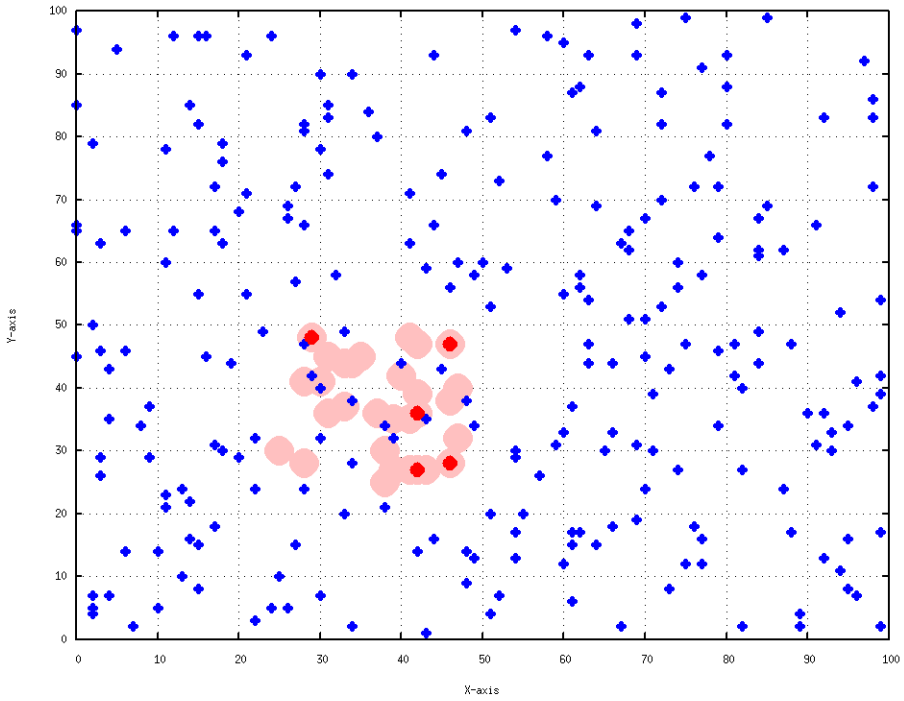

Figure 6.6: A graph with facilities (in pink) clustered in a square and users (in blue) mainly positioned outside that square. The red points are the facilities that are open in the solution provided by PPS. The edges are not displayed

These instaces have proved to be very hard to solve by both the Progressive Plants Selector algorithm and SCIP, even if SCIP gave better results in this case. We were not able to solve many medium-big instances, expecally if users volumes were randomly chosen (as usual in the interval $[1, 10]$. To give an idea

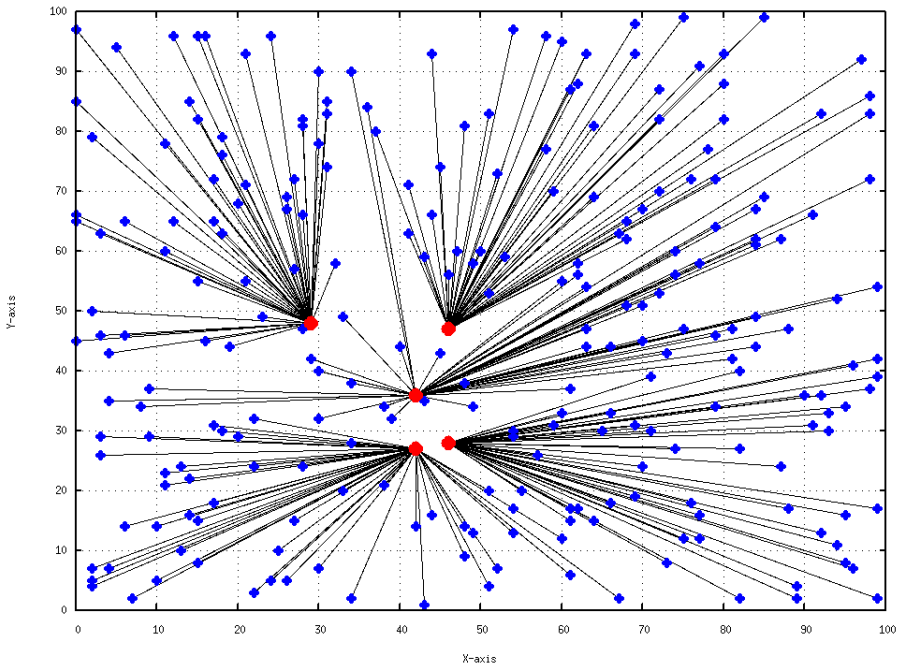

Figure 6.7: The solution provided by PPS of the graph in Figure 6.6

on how this structure is complicating, we show, in Table 6.4, some results on instances with random volumes for users.

Seed	yVrs	Usrs	TimP	SVP	TimS	SVS	GapS
980170	150	250	1690.63	14293.6	1698.24	14322.1	0.03
980255	150	250	1382.14	14321.4	1389.89	14303.3	0.06
980340	300	250	8499.88	14359.8	2470.72	14329.5	0.03

Table 6.4: Columns description: **Seed**: the seed used to generate tha random graph; **yVrs**: the number of y-variables in the model; **Usrs**: the number of users in the model; **TimP**: the time needed to PPS to solve the problem; **SVP**: the solution value found by PPS; **TimS**: the time needed to SCIP to solve the problem; **SVS**: the solution value found by SCIP; **GapS**: the gap value for SCIP (percentage/100)

We decided to show the results obtained by instances where the users volumes were constant and equals to 1 (the easiest case).

6.3.2 Users Sorrounded by Facilities

These instances are constructed symmetrically to those described in the previous Section. In this case users are placed in a square and this square is sorrounded by facilities. An example of these graphs is shown in Figure 6.8 and Figure 6.9.

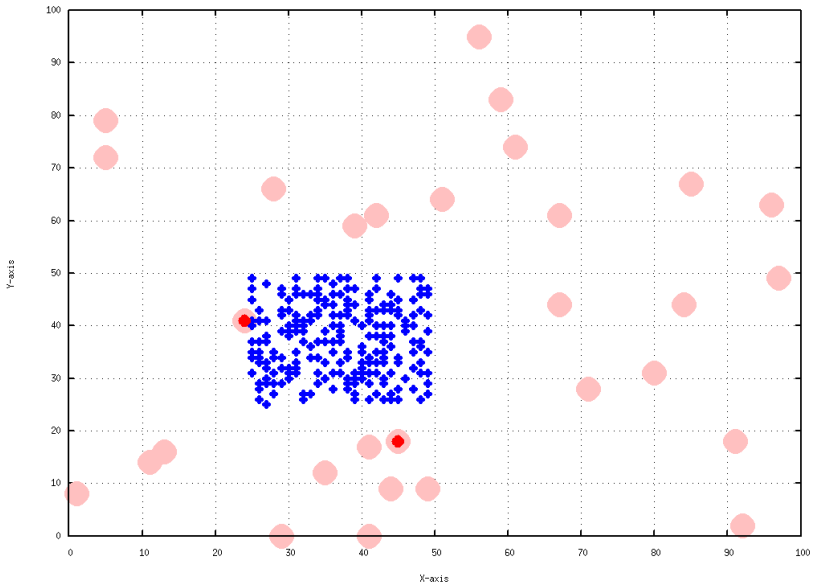

Figure 6.8: A graph with users (in blue) clustered in a square and facilities (in pink) positioned outside that square. The red points are the facilities that are open in the solution provided by PPS. The edges are not displayed

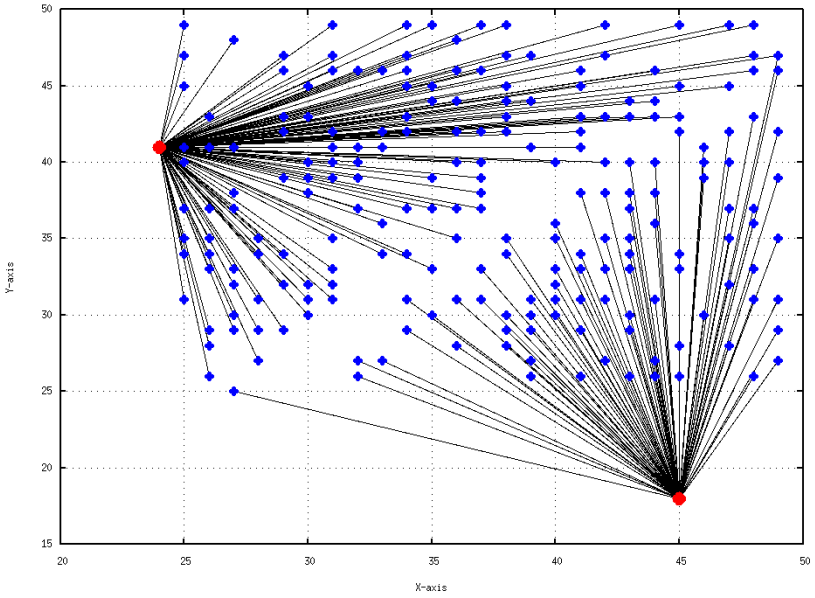

Figure 6.9: The solution provided by PPS of the graph in Figure 6.8

6.3.3 Clustered Users

We have a small number of users that is evenly distributed in a square and a big number of users organized into clusters. Facilities are evenly distributed in the square.

These kind of instances are those most similar to the real ones: Emilia-Romagna has the biggest number of users concentrated into the main cities, while a small number of clients is (evenly) distributed onto the remaining territory.

These instances have been solved more easy than the previous, and the algorithm behaved as for the real instances.

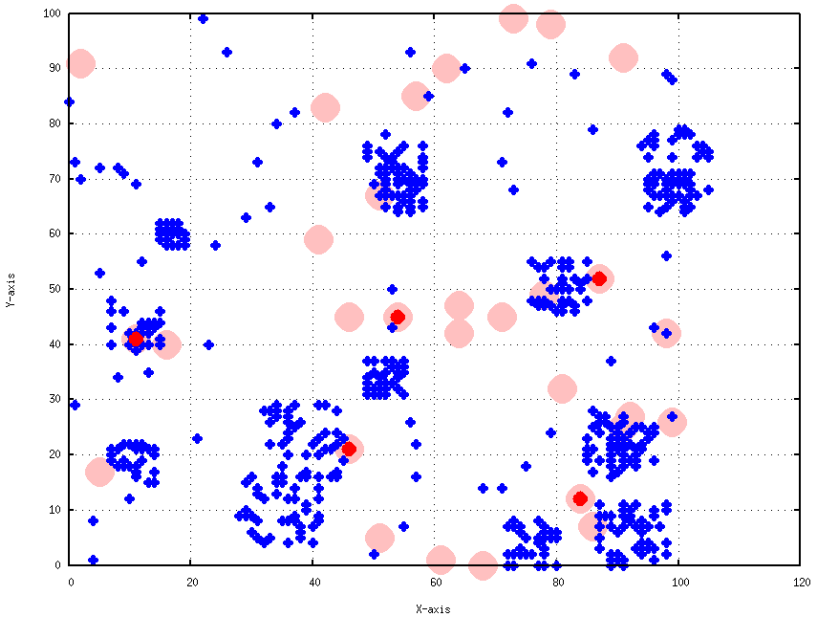

Figure 6.10: A graph with users (in blue) mostly organized in different clusters and facilities randomly distributed in the full square. These type of graphs are the most similar to the real instances, in which we have some big cities in which is concentrated the biggest number of users.

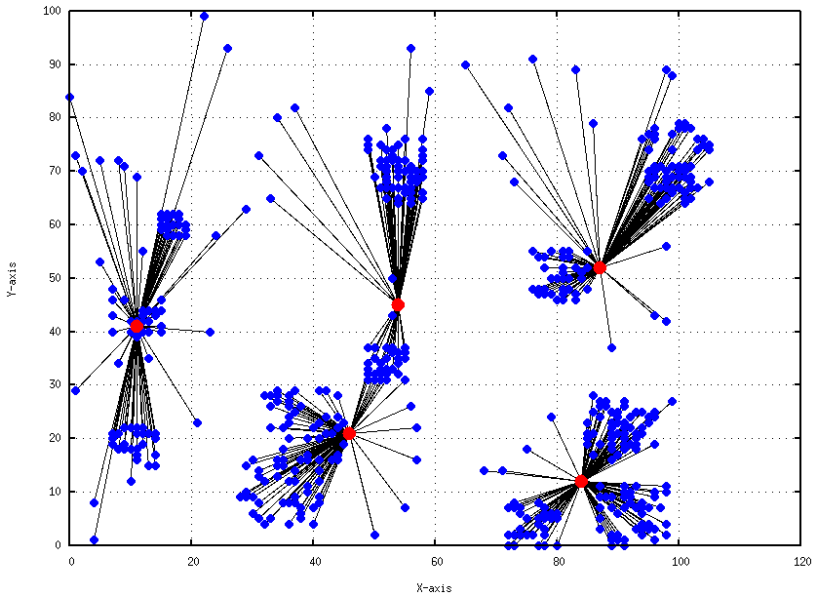

Figure 6.11: The solution provided by PPS of the graph in Figure 6.10

Seed	yVrs	Usrs	SVP	SVS	Tim	GapS
1287	90	600	15281.7	14817.8	197.74	0
1372	150	400	13216.5	11561.4	53.61	0
1457	210	400	11372.6	10202.7	65.47	0
1542	270	400	10056.9	10056.9	76.86	0
1627	300	400	11033.3	10521.7	366.24	0
1712	90	700	18025.6	17254.9	66.69	0
1797	150	700	15444.7	13861.8	73.56	0
1882	210	700	14347.6	14008.9	280.87	0
1967	270	700	15434.4	15282.8	193.13	0
2052	300	700	14864.9	14435.1	182.13	0
88164	90	600	15977.2	15538	142.57	0
88249	150	400	11537.2	11251.1	358.73	0
88334	210	400	11453.2	11152.4	211.89	0
88419	270	400	10843.2	10345.9	77.81	0
88504	300	400	9931.34	9637.53	269.62	0
88589	90	700	15965.6	15641.9	48.46	0
88674	150	700	16143	38288.3	364.81	1.45
88759	210	700	14278.9	13849.4	230.09	0
88844	270	700	13264.9	13264.9	142.49	0
88929	300	700	16043.1	14348	176.85	0

Table 6.5: Columns description: **Seed**: the seed used to generate tha random graph; **yVrs**: the number of y-variables in the model; **Usrs**: the number of users in the model; **SVP**: the solution value found by PPS; **SVS**: the solution value found by SCIP; **Tim**: the time needed to solve the problem; **GapS**: the gap value for SCIP (percentage/100)

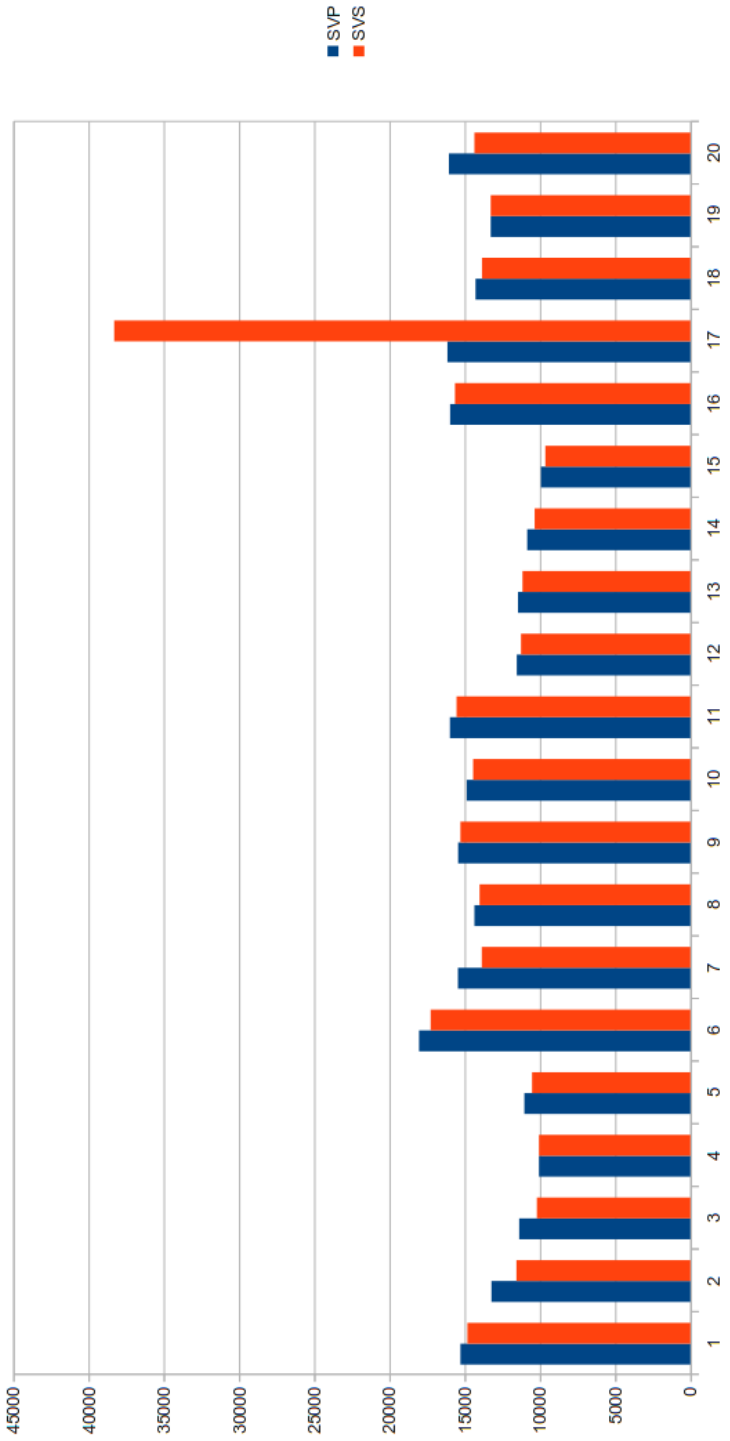

Figure 6.12: In the image it is possible to see the comparison between the solution values found by SCIP (in orange) and by PPS (in blue) for some instances generated as described in Section 6.3.3. The results given by PPS are very close to the solutions provided by SCIP itself. Notice that in this case SCIP almost always finds the optimal solution to the problem.

6.4 Real Instances

Finally we show some of the results obtained by solving the real instances. As we said in the previous sections, we needed to optimize the positioning of Health Houses on all the territory of Emilia-Romagna, but, due to the size of the problem, we decided to analyze the different USLs separately. This takes us to eleven different optimization problems, that we solved in parallel using Hadoop.

In the following table we report the solution values obtained by optimizing the graphs corresponding to each USL, in which we used a constant set of possible facility locations and we varied the number of clusters.

USL	yVrs	Usrs	Tim	SVP	SVS	GapS
BOLOGNA	198	250	81.25	1.85E+008	1.82E+008	0
BOLOGNA	198	350	99.44	1.89E+008	1.81E+008	0
BOLOGNA	198	400	152.6	1.85E+008	1.81E+008	0
BOLOGNA	198	600	172.85	1.85E+008	1.82E+008	0
BOLOGNA	198	800	190.74	1.86E+008	1.82E+008	0
CESENA	171	250	48.75	4.32E+007	4.31E+007	0
CESENA	171	350	72.44	4.35E+007	4.31E+007	0
CESENA	171	400	72.26	4.43E+007	4.31E+007	0
CESENA	171	600	79.64	4.35E+007	0	0
CESENA	171	800	144.01	4.37E+007	0	0
FERRARA	162	250	70.23	1.03E+008	1.03E+008	0
FERRARA	162	350	73.72	1.04E+008	1.02E+008	0
FERRARA	162	400	73.58	1.04E+008	1.02E+008	0
FERRARA	162	600	98.92	1.04E+008	1.03E+008	0
FERRARA	162	800	106.02	1.03E+008	1.06E+008	0.03
FORLI	180	250	46.53	3.76E+007	3.75E+007	0
FORLI	180	350	64.93	3.76E+007	3.75E+007	0
FORLI	180	400	65.93	3.76E+007	3.76E+007	0
FORLI	180	600	72.18	4.00E+007	1.50E+008	3.01
FORLI	180	800	71.08	3.77E+007	1.45E+008	2.89
IMOLA	153	250	24.32	2.85E+007	2.85E+007	0
IMOLA	153	350	44.7	2.85E+007	2.85E+007	0
IMOLA	153	400	50.66	2.85E+007	2.85E+007	0
IMOLA	153	600	88.62	2.92E+007	2.84E+007	0
IMOLA	153	800	96.76	2.92E+007	2.84E+007	0
MODENA	180	250	62.28	2.26E+008	2.07E+008	0
MODENA	180	350	75.43	2.19E+008	6.61E+008	2.21
MODENA	180	400	78.89	2.20E+008	2.07E+008	0

MODENA	180	600	154.65	2.21E+008	2.07E+008	0
MODENA	180	800	139.82	2.22E+008	6.87E+008	2.34
PARMA	198	250	48.07	1.36E+008	1.36E+008	0
PARMA	198	350	71.47	1.36E+008	1.36E+008	0
PARMA	198	400	72.1	1.37E+008	1.31E+009	8.65
PARMA	198	600	114.25	1.38E+008	1.34E+009	8.87
PARMA	198	800	139.4	1.37E+008	1.34E+009	8.83
PIACENZA	144	250	50.79	8.62E+007	8.24E+007	0
PIACENZA	144	350	70.53	8.26E+007	8.22E+007	0
PIACENZA	144	400	68.81	8.24E+007	8.20E+007	0
PIACENZA	144	600	100.83	8.35E+007	8.20E+007	0
PIACENZA	144	800	122.78	8.26E+007	8.20E+007	0
RAVENNA	180	250	71.92	1.12E+008	1.09E+008	0
RAVENNA	180	350	74.57	1.12E+008	4.59E+008	3.21
RAVENNA	180	400	56.59	1.10E+008	4.39E+008	3.03
RAVENNA	180	600	141.81	1.10E+008	1.09E+008	0
RAVENNA	180	800	149.46	1.12E+008	1.09E+008	0
REGGIO	207	250	131.31	1.30E+008	1.29E+008	0
REGGIO	207	350	162.92	1.35E+008	1.29E+008	0
REGGIO	207	400	183.62	1.36E+008	1.29E+008	0
REGGIO	207	600	169.14	1.30E+008	4.20E+008	2.26
REGGIO	207	800	344.7	1.41E+008	4.22E+008	2.26
RIMINI	171	250	89.66	5.89E+007	5.76E+007	0
RIMINI	171	350	119.08	5.78E+007	5.75E+007	0
RIMINI	171	400	91.98	5.96E+007	5.91E+007	0.03
RIMINI	171	600	140.35	5.90E+007	2.15E+008	2.74
RIMINI	171	800	164.47	5.94E+007	2.14E+008	2.74

Table 6.6: Columns description: **USL**: the name of the USL; **yVrs**: the number of y-variables in the model; **Usrs**: the number of users in the model; **Tim**: the time needed to solve the problem; **SVP**: the solution value found by PPS; **SVS**: the solution value found by SCIP; **GapS**: the gap value for SCIP (percentage/100)

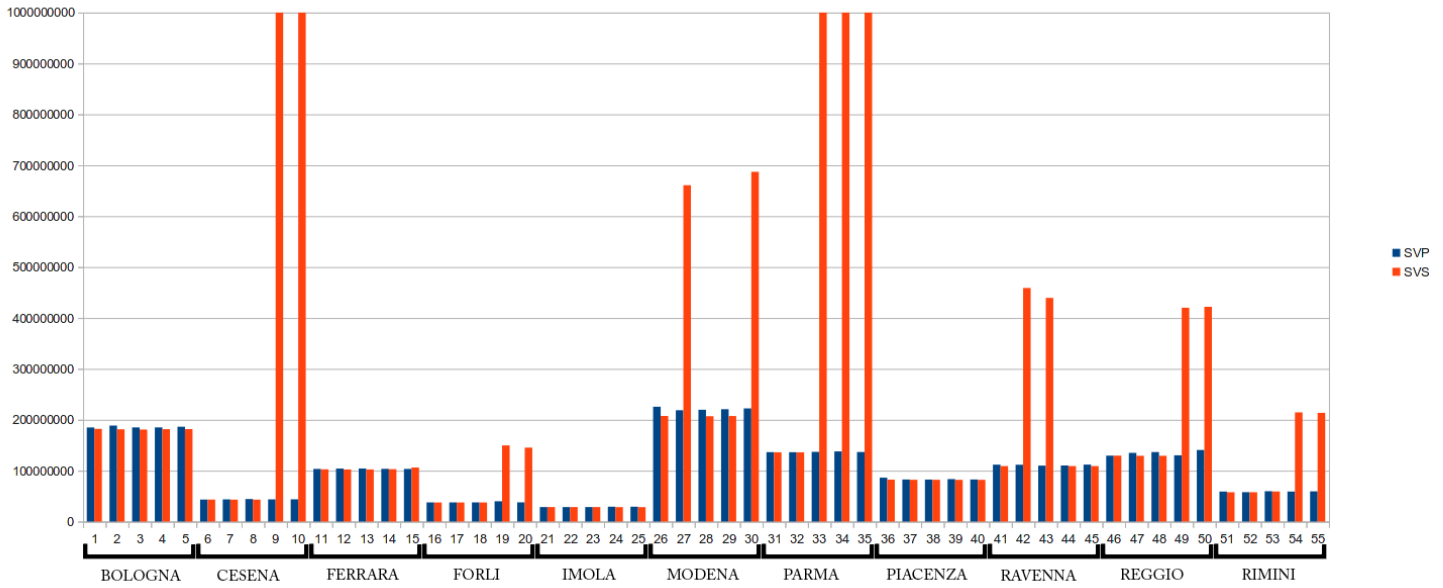

Figure 6.13: In the image it is possible to see the comparison between the solution values found by SCIP (in orange) and by PPS (in blue) for some real instances.

Conclusion

The two valuable assets we have obtained from the study we described so far, are the MIP model of a new problem derived from the well-known Plant Location Problem and a heuristic algorithm that has proved to be good both on the times and the quality of the solutions point of view. The Progressive Plant Selector performs pretty well in all our tests. It often gives better results than the MIP solver (within the same time) and it has a better "first-feedback-time".

"Luckily", the real instances of the MMBPLP are the easiest to solve with both the solution strategies. This makes the model good enough to be used in practical applications.

There are several aspect of the study we made that could be the starting point for future works.

One possibility could be to extend the model in order to manage k different classes of facility sizes, with $k \in \mathbb{N}$; by studying a problem with this new specification, one must consider that the number of y variables involved in the problem is directly proportional to the number of classes.

Another possible extension, could be to allow multi-utility facilities (i.e. in our particular case study, each Health House could perform different services). Also, users could ask for different performances, so we could have at the same time facilities providing diverse services and users having various necessities.

Finally, an improvement to our case study application, could be to use routing distances instead of Euclidean distances. This would allow us to give better solutions, involving more realistic traveling times between people and plants. An evoluton of this last idea, could be to consider the distances among users and facility positioning sites, as a set of aleatory variables.

Bibliography

[1] G. Cornuejols, G. L. Nemhauser, and L. A. Wolsey, "The uncapacitated facility location problem," *In Francis RL, MIrchandani PB (Eds.) Discrete Location Theory. New York: Wiley-Interscience*, pp. 119–171, 1983.

[2] C. Beltran-Royo, J.-P. Vial, and A. Alonso-Ayuso, "Semi-lagrangian relaxation applied to the uncapacitated facility location problem," *Computational Optimization and Applications*, vol. 51, pp. 387–409, 2012.

[3] D. Erlenkotter, "A dual-based procedure for uncapacitated facility location," *Oper Res*, vol. 26, pp. 992–1009, 1978.

[4] A. Kuehn and M. Hamburger, "A heuristic program for locating warehouses," *Manag Sci*, vol. 9, pp. 643–666, 1963.

[5] C. S. Revelle and G. Laporte, "The plant location problem: New models and research prospects," *Operations Research*, vol. 44(6), pp. 864–874, 1996.

[6] N. Megiddo and A. Tamir, "On the complexity of locating linear facilities in the plane," *Operations Research Letters*, vol. 5, pp. 194–197, 1982.

[7] M. Efroymson and T. Ray, "A branch and bound algorithm for plant location," *Oper Res*, vol. 14, pp. 361–368, 1966.

[8] K. Spielberg, "Algorithms for the simple plant location problem with some side constraints," *Oper Res*, vol. 17, pp. 85–111, 1969.

[9] B. Khumawala, "An efficient branch and bound algorithm for the warehouse location problem," *Manag Sci*, vol. 18, pp. B718–B731, 1972.

[10] L. Scharge, "Implicit representation of variable upper bounds in linear programming," *Math Program Study*, vol. 4, pp. 118–132, 1975.

[11] U. Akinc and B. Khumawala, "An efficient branch and bound algorithm for the capacitated warehouse location probelm," *Manag Sci*, vol. 23, pp. 585–594, 1977.

[12] R. Nauss, "An improved algorithm for the capacitated facility location problem," *J Oper Res Soc*, vol. 29, pp. 1195–1201, 1978.

[13] T. Van Roy, "A cross decomposition algorithm for capacitated facility location," *Oper Res*, vol. 34, pp. 145–163, 1986.

[14] G. Gutierrez and P. Kouvelis, "A robustness approach to international sourcing," *Ann Oper Res*, vol. 59, pp. 165–193, 1995.

[15] H. A. Eiselt and V. Marianov, eds., *Foundations of Location Analysis*. 2011.

[16] A. Neebe and B. Khumawala, "An improved algorithm for the multi-commodity location problem," *J Oper Res Soc*, vol. 32, pp. 143–169, 1981.

[17] J. Karkazis and T. Boffey, "The multi-commodity facilities location problem," *J Oper Res Soc*, vol. 32, pp. 803–814, 1981.

[18] J. Klincerwicz and H. Luss, "A dual based algorithm for multiproduct uncapacitated facility location," *Transp Sci*, vol. 21, pp. 198–206, 1987.

[19] Z. Scen, "A multi-commodity supply chain design problem," *IIE Trans*, vol. 7, pp. 753–762, 2005.

[20] L. Kaufman and H. P. Eede M.V., "A plant and warehouse location problem," *Oper Res Quart*, vol. 28, pp. 547–554, 1977.

[21] A. Geoffrion and G. Graves, "Multicommodity distribution system design by bender's decomposition," *Manag Sci*, vol. 20, pp. 822–844, 1974.

[22] T. Van Roy and D. Erlenkotter, "Dual-based procedure for dynamic facility location," *Manag Sci*, vol. 28, pp. 1091–1105, 1982.

[23] S.-K. Lim and Y.-D. Kim, "An integrated approach to dynamic plant location and capacity planning," *Journal of the Operational Research Society*, vol. 50, pp. 1205–1216, 1999.

[24] C. Canel, "An algorithm for the capacitated, multi-commodity multi-period facility location problem," *Comput Oper Res*, vol. 28, pp. 411–427, 2001.

[25] R. Soland, "Optimal facility location with concave costs," *Oper Res*, vol. 22, pp. 373–382, 1974.

[26] K. Holmberg, "Solving the staircase cost facility location problem with decomposition and piecewise linearization," *Eur J Oper Res*, vol. 75, pp. 41–61, 1994.

[27] C. Beltran, C. Tadonki, and J.-P. Vial, "Solving the p-median problem with a semi-lagrangian relaxation," *Computational Optimization and Applications*, vol. 35(2), pp. 239–260, 2006.

[28] J. Hiriart-Urruty and C. Lemaréchal, *Fundamentals of Convex Analysis*. 2001.

[29] T. A. S. Foundation, “What is apache hadoop,” 2014.

[30] M. G. Noll, “Running hadoop on ubuntu linux,” 2014.

[31] W. F. Inc., “Apache hadoop,” 2014.