On solving the Multi Objective Facility Location Problem with Single Sourcing and Capacity Constraints

Charly Lersteau

directed by

Dr Fabien Tricoire   Pr Xavier Gandibleux   Dr Anthony Przybylski

Université de Nantes — LINA, UMR CNRS 6241
UFR Sciences – 2 rue de la Houssinière BP92208, F44322 Nantes cedex 03 – France

University of Vienna — Department of Business Administration
Brünner Straße 72 – 1210 Wien – Austria

Abstract

The Multi-Objective Uncapacitated Facility Location Problem has been well solved by Gandibleux et al. [11] with an exact two-step method. This paper presents a direct extension of this algorithm for the *Multi-Objective Single Source Capacitated Facility Location*. The first step (*paving*) consists on a *branch and bound* on facility opening variables, which produces assignment subproblems. The power of this step is to eliminate entire subproblems by dominance tests. The second step (*generation*) solves each subproblem and merges all the nondominated solutions. Subproblems are solved using a *label setting* algorithm. Moreover, we provide an efficient *branch and bound* algorithm to solve single objective version of our subproblems. The computation times are compared to SCIP MIP solver.

Keywords: capacitated, facility location, multi objective, single source, exact

1. Introduction

Decision making is one of the most important things encountered in a life. As Matthias Ehrgott says, *Life is about decisions* [6]. There are so many situations in the real world where a decision has to be taken with conflicting criteria. Multi-Objective Combinatorial Optimization is a field of Operations Research which copes with these kind of problems in a scientific way. In particular, we will speak about *Facility Location Problems*.

The *Facility Location Problems* (FLP) involve locating or positioning a number of facilities in order to minimize their fixed *opening costs* and *delivering costs* by serving required demands. For example, we need to locate some warehouses which have building costs. Then we have to assign known customers to warehouses, the criteria can be the profit and travel distance between them. A good survey shows the state of the art in *Multi Objective Facility Location Problems* [7]. The *Uncapacitated Facility Location Problem* (UFLP) has been well studied in the literature and its multi-objective version is also well solved [11, 9].

The *Capacitated Facility Location Problem* (CFLP) is a generalization of the *UFLP*. The objective is still to minimize opening costs and delivering costs, but, in contrast, the customers have demand amounts and facilities have a capacity limit. In this paper, we will consider the case in which the customers can be delivered by only one facility, also called *Single Source Capacitated Facility Location* (SSCFLP).

Let us describe an integer programming model for the multi-objective SSCFLP. Let $I = \{1, \ldots, m\}$ be a set of customers and $J = \{1, \ldots, n\}$ be a set of facilities. Facility opening costs (f_j^k), delivery costs (c_{ij}^k), customer demands (d_i) and facility capacities (q_j) are given.

The objectives (1) are to minimize the total costs including facility opening costs and delivery costs.

$$
\text{(SSCFLP)} \quad
\begin{aligned}
\min z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}^k x_{ij} + \sum_{j=1}^{n} f_j^k y_j & \qquad \forall k \in \{1, \ldots, p\} && (1)\\
\text{s.t.} \sum_{j=1}^{n} x_{ij} = 1 & \qquad \forall i \in I && (2)\\
x_{ij} \le y_j & \qquad \forall i \in I, \forall j \in J && (3)\\
\sum_{i=1}^{m} d_i x_{ij} \le q_j y_j & \qquad \forall j \in J && (4)\\
x_{ij} \in \{0, 1\} & \qquad \forall i \in I, \forall j \in J && (5)\\
y_j \in \{0, 1\} & \qquad \forall j \in J, \forall j \in J && (6)
\end{aligned}
$$

The constraints (2) force all the demands to be covered by the facilities. The constraints (3) ensure a used facility will take opening costs into account. The constraints (4) make the facilities limited by their capacity. Moreover, (5) make sure the deliveries are not split and each customer is delivered by only one facility.

The method [11] runs in two steps named *paving* and *generation*. The idea is to decompose the problem into subproblems. In each subproblem, all y_j variables are already fixed and we only look at x_{ij} variables. During the *paving*, a *branch & bound* procedure is performed on y_j variables to filter subproblems which have no chance to contain nondominated solutions for the global problem. This produces a paving which bounds the objective space into boxes. Next, the generation procedure finds all the nondominated solutions in each remaining subproblem and merges them to keep only the nondominated points for the global problem.

First, we will study the subproblems and provide a method to solve them in their single objective version. Then, we will describe the paving and generation procedures more in details.

2. A theorical study of subproblems

In this section, we will consider the single objective version of the subproblems. In each subproblem, the decision to open a facility or not has already been taken. The only variables remaining are them for assigning customers to facilities. Subproblems are modeled using the following formulation.

$$
\text{(SP)} \quad \begin{aligned}
\min z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} & & (7)\\
\text{s.t.} \sum_{j=1}^{n} x_{ij} = 1 & \quad \forall i \in I & (8)\\
\sum_{i=1}^{m} d_i x_{ij} \leq q_j & \quad \forall j \in J & (9)\\
x_{ij} \in \{0,1\} & \quad \forall i \in I, \forall j \in J & (10)
\end{aligned}
$$

(SP) also models a *Single-Source Transportation Problem* (SSTP) [18] which is a special case of a *Generalized Assignment Problem* [19] in minimization (Min-GAP). Because of the combination of constraints (8) and (9), (SP) can have no feasible solution.

2.1. NP-completeness

Let us denote by $\mathcal{A}$ the following problem.

$\mathcal{A}$

INSTANCE : Positive integers $m > n \geq 1$, a vector of demands $d = (d_1, \ldots, d_m) \in \mathbb{Z}_+^m$ and a vector of capacities $q = (q_1, \ldots, q_n) \in \mathbb{Z}_+^n$.
QUESTION : Are there disjoint subsets $I_j \subseteq \{1, \ldots, m\}, \forall j \in \{1, \ldots, n\}$ such that

$$\bigcup_{j=1}^{n} I_j = \{1, \ldots, m\} \tag{11}$$

$$\sum_{i \in I_j} d_i \leq q_j, \forall j \in \{1, \ldots, n\} \tag{12}$$

Lemma 1 *$\mathcal{A}$ is* NP-complete.

PROOF. Let us give a definition of the PARTITION problem.

PART.

INSTANCE : A vector of positive integers $d = (d_1, \ldots, d_m) \in \mathbb{Z}_+^m$
QUESTION : Is there a subset $I \subseteq \Omega = \{1, \ldots, m\}$ such that

$$\sum_{i \in I} d_i = \sum_{k \in \overline{I}} d_k \tag{13}$$

Reduction : From a PARTITION instance $(d_1, \ldots, d_m)$, we create a $\mathcal{A}$ instance by defining $n = 2$ capacities : $q_1 = q_2 = \frac{1}{2}Q$ with $Q = \sum_{i \in \Omega} d_i$.
By contradiction, let us prove that $\sum_{i \in I_1} d_i = \sum_{k \in I_2} d_k$ iff the answer of $\mathcal{A}$ is YES. Due to constraints (12), if ever $\sum_{i \in I_1} d_i < \frac{Q}{2}$, then $\sum_{i \in I_1} d_i + \sum_{k \in I_2} d_k \leq \sum_{i \in I_1} d_i + \frac{Q}{2} < Q$ which contradicts $\sum_{i \in I_1} d_i + \sum_{k \in I_2} d_k = \sum_{i \in \Omega} d_i = Q$.
Moreover, as we have I_1 and I_2 disjoint, $I_2 = \overline{I_1}$. Thus, $\sum_{i \in I_1} d_i = \sum_{k \in \overline{I_1}} d_k$, which is equivalent to constraint (13).
Our reduction makes $\mathcal{A}$ be the same problem as PARTITION, which has been proven *NP-complete* [14, 12, 13].
□

Theorem 1 *Single Source Transportation Problem is* NP-hard.

Proof. Let $M = \{1, \ldots, m\}, N = \{1, \ldots, n\}, (c_{ij} \in \mathbb{Z}_+)_{M \times N}$ along with $C \in \mathbb{Z}_+$ be given. We add the constraint

$$\sum_{i \in M} \sum_{i \in N} c_{ij} \leq C \tag{14}$$

to problem $\mathcal{A}$ and denote the so-obtained problem by $\mathcal{B}$. $\mathcal{B}$ is also the decision version of SSTP. By setting $c_{ij} = C = 0, \forall (i, j) \in M \times N$, $\mathcal{B}$ restricts to $\mathcal{A}$, so $\mathcal{B}$ is *NP-complete*. So the optimization problem where (14) is a minimization objective (with c_{ij} set to the values of the SSTP instance) subject to the constraints of $\mathcal{A}$ is *NP-hard*. □

It is also obvious that SSCFLP and its multi-objective variant are *NP-complete*. In fact, SSTP is a specialization of SSCFLP where the opening costs are all zero (all y_j can be set to 1 without compromising the objective). Let us analyze what causes the hardness of the problem.

When the knapsack constraints (9) are loose enough, i.e. $\forall j \in \{1, \ldots, n\}, q_j \geq \sum_{i=1}^{m} d_i$, these constraints can be ignored. It is also a semi-assignment problem, easily solvable in polynomial-time. In this case, SSCFLP becomes a UFLP.

When the knapsack constraints (9) are the tightest they can, i.e. $\sum_{j=1}^{n} q_j = \sum_{i=1}^{m} d_i$, then the problem to decide whether a feasible solution exists is similar to an Exact Cover Problem, which is very hard to solve. In fact, the thing that makes the problem difficult is essentially the tightness of the knapsack constraints (9). Our experiments show that the tigher they are, the harder SSTP is.

We define the tightness coefficient as follows :

$$1 - \frac{\sum_{j=1}^{n} q_j - \sum_{i=1}^{m} d_i}{\sum_{j=1}^{n} q_j} \tag{15}$$

If this coefficient is near 1, the problem is considered potentially harder. In the opposite case, it is much easier. Our conjecture is : the more this coefficient is near 0, the more the solution of the continuous relaxation is similar to the optimal solution.

2.2. Modeling as a Transportation Problem

We can consider the contraints (8) as inequality constraints ($\geq$), since an optimal solution of the following linear program will never have more than one assignment per customer.

$$\text{(SP}'\text{)}\quad \begin{aligned} \min z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} & & (16)\\ \text{s.t.} \sum_{j=1}^{n} x_{ij} \geq 1 & \quad \forall i \in I & (17)\\ \sum_{i=1}^{m} d_i x_{ij} \leq q_j & \quad \forall j \in J & (18)\\ x_{ij} \in \{0,1\} & \quad \forall i \in I, \forall j \in J & (19) \end{aligned}$$

Theorem 2 *If x^* is optimal for (SP′) then $\sum_{j=1}^{n} x^*_{ij} = 1, \forall i \in I$.*

Proof. Suppose x^* is an optimal solution for (SP′) and $x^*_{ij} = 1$ and $x^*_{ik} = 1$ for at least one $i \in \{1,\ldots,m\}$ and $j,k \in \{1,\ldots,n\}$, $j \neq k$. As x^* already conforms to the knapsack constraints (18), we can remove one of the items x^*_{ij} or x^*_{ik} without compromsing them. By setting x^*_{ij} or x^*_{ik} to zero, we keep a feasible solution and reduce the costs in the objective function. Also we prove the first optimality assumption was wrong. □

By replacing x_{ij} by $y_{ij} = d_i x_{ij}$, we obtain an equivalent formulation :

$$\text{(SSTP)}\quad \begin{aligned} \min z = \sum_{i=1}^{m}\sum_{j=1}^{n} \frac{c_{ij}}{d_i} y_{ij} & & (20)\\ \text{s.t.} \sum_{j=1}^{n} y_{ij} \geq d_i & \quad \forall i \in I & (21)\\ \sum_{i=1}^{m} y_{ij} \leq q_j & \quad \forall j \in J & (22)\\ y_{ij} \in \{0,d_i\} & \quad \forall i \in I, \forall j \in J & (23) \end{aligned}$$

In fact, (SSTP) is a special case of the *Hitchcock Problem* [10] or *Transportation Problem* (TP) where q_j are the supplies and d_i the demands. This subproblem is single source because of the constraints (23) that force each demand to be served by only one supply.

The model (SSTP) is also convenient because the Transportation Problem is well studied and very fast algorithms are provided, in particular a primal-dual specialized simplex method [10, 20]. The primal-dual method can also be used to compute the continuous relaxation as a lower bound. To have a correct Transportation Problem, we have to balance the network. We add a dummy demand d_{m+1} with costs $c_{m+1j} = 0$ and consider the flows from this demand can be fractional.

2.3. Properties

Analyzing some properties about (SP) helps us to have a better understanding of the problem.

Theorem 3 *If $\exists(i,j) \in (I,J)$ such that $d_i > q_j$, then setting $x_{ij} = 1$ will never lead to a feasible solution.*

Proof. Contradiction : $d_i > q_j$ and $x_{ij} = 1 \implies \sum_{k=1}^{m} d_k x_{kj} > q_j$. □

Theorem 4 *If $\exists\, i \in I, \exists!\, j \in J$ such that $d_i \leq q_j$, then setting $x_{ij} = 0$ will never lead to a feasible solution.*

Proof. Setting $x_{ij} = 0$ will lead to the situation of the theorem 4 for each $k \in J, k \neq j$. □

Theorem 5 *If $m = n$ and $\forall i \in I,\ \forall j \in J,\ \frac{q_j}{2} < d_i \leq q_j$, then (SP) is equivalent to the* Assignement Problem.

Proof. It consists on proving the knapsack constraints are equivalent to $\forall j \in J\ ,\ \sum_{i=1}^{n} x_{ij} = 1$.
Suppose $\exists j \in J\ ,\ \sum_{i=1}^{n} x_{ij} > 1$ then $\exists i, k \in I, i \neq k$ such that $x_{ij} = 1$ and $x_{kj} = 1$.
Then $d_i\ x_{ij}\ +\ d_k\ x_{kj}\ = d_i\ +\ d_k\ >\ \frac{q_j}{2}\ +\ \frac{q_j}{2}\ =\ q_j$. So the knapsack constraint is violated. The first assumption is also wrong, so $\forall j \in J\ ,\ \sum_{i=1}^{n} x_{ij} \leq 1$.
Now suppose $\exists j \in J\ ,\ \sum_{i=1}^{n} x_{ij} = 0$, then $\sum_{j=1}^{n} \sum_{i=1}^{n} x_{ij} \leq n - 1$, which is infeasible because
$\forall i \in I\ ,\ \sum_{j=1}^{n} x_{ij} = 1$ and also $\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} = n$. To conclude, we have proven $\forall j \in J\ ,\ \sum_{i=1}^{n} x_{ij} = 1$. □

2.4. Dual problem

Let us give the dual model of (SSTP) and an interpretation. Let u_i and v_j be the dual variables respectively corresponding to the constraints (21) and (22).

$$\text{(DSSTP)} \quad \max z = \sum_{i=1}^{m} d_i u_i + \sum_{j=1}^{n} q_j v_j \tag{24}$$

$$\text{s.t. } u_i + v_j \leq \frac{c_{ij}}{d_i} \qquad \forall i \in I, \forall j \in J \tag{25}$$

$$u_i, v_j \in \mathbb{R} \qquad \forall i \in I, \forall j \in J \tag{26}$$

We could imagine that an intermediate company is charged to manage the logistics. Its goal is to maximize the profit (24). The company buys products to facilities and sells them to customers, respectively with the unit prices v_j (negative) and u_i. The company has to decide the prices u_i and v_j to be cheaper to involve them than to handle the transportation by ourselves (i.e. with a price of $\frac{c_{ij}}{d_i}$) (25).

3. A branch and bound algorithm for SSTP

In order to solve SSTP in a more efficient way than a generic MIP solver, we describe a *branch and bound* algorithm suitable for this subproblem. The main idea is to solve the problem with respect to the knapsack constraints which are very tight in our instances.

3.1. Lower bound

Let us first describe the behaviour in each node. Our branch and bound is a very simple method which lies on two bounds : a continuous and a Lagrangian relaxations. In each node, we first solve the continuous relaxation using a primal-dual transportation algorithm [10, 20], which is a very fast specialization of the simplex algorithm. It also gives the optimal objective value z^{LP*} and dual variables u_i, which will serve to compute the Lagrangian bound.

Let u_i be the optimal dual variables of constraints (21) of the continuous version of (SSTP) and $\lambda_i = d_i u_i$ the Lagrangian multipliers. The Lagrangian relaxation model is given.

$$(\text{LSP}_\lambda)\quad \begin{aligned} \min z^L &= \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{m}\lambda_i(1 - \sum_{j=1}^{n} x_{ij}) && && (27)\\ \text{s.t.} \sum_{i=1}^{m} d_i x_{ij} &\leq q_j && \forall j \in J && (28)\\ x_{ij} &\in \{0,1\} && \forall i \in I, \forall j \in J && (29) \end{aligned}$$

This model can be decomposed into independent problems. It is also equivalent to solve a sequence of knapsack problems.

$$(\text{KP}_j^\lambda)\quad \begin{aligned} \max z_j^{KP} &= \sum_{i=1}^{m}(\lambda_i - c_{ij})x_{ij} && && (30)\\ \text{s.t.} \sum_{i=1}^{m} d_i x_{ij} &\leq q_j && && (31)\\ x_{ij} &\in \{0,1\} && \forall i \in I && (32) \end{aligned}$$

We solve each (KP_j^λ) using a simple *best first search* branch and bound. The Lagrangian lower bound is obtained by the following formula.

$$z^{L*} = \sum_{i=1}^{m}\lambda_i - \sum_{j=1}^{n} z_j^{KP*}$$

Let z^{P*}, z^{LP*} and z^{L*} be respectively the parent node's bound, the continuous bound and the Lagrangian bound. The lower bound of the current node is also :

$$z^* = \max\left\{ z^{P*}\ ,\ z^{LP*}\ ,\ z^{L*} \right\}$$

The situation in which both continuous and Lagrangian bounds are less than the parent's bound can occur. It occurs when the Lagrangian bound behaves well on the parent (greater than the continuous relaxation) but bad on the child. The Lagrangian relaxation can be very unstable because of the changes of the multipliers at each computation. It is also convenient to take the parent node's lower bound into account.

3.2. Branching strategy

Our branch and bound is a *best first search*, i.e. it selects the node with the best lower bound between the candidates. The produced tree is binary as we branch on one boolean variable at a time.

Let x_{ij} be the optimal solution of (LSP_λ), u_i and v_j be the optimal dual variables of respectively constraints (21) and (22) of continuous version of (SSTP). Let r_j the residual capacity of the facility j, i.e. its remaining capacity unassigned in the partial solution.

Denote S a pool of variables such that :

$$S = \left\{ (i, j) \mid x_{ij} = 1 \text{ or } \sum_{k=1}^{n} x_{ik} = 0 \right\}$$

S is the set of variables x_{ij} such that, in the previous solution, the assignment (i, j) has been chosen or there is no chosen assignment for the customer i. This makes every variable potentially selectable as every variable which is not in S in the current node can be selected in a further node. The variable chosen for branching is the one in S which has the maximal Ψ_{ij} coefficient, computed as follows.

$$\Psi_{ij} = \frac{u_i - v_j - \dfrac{c_{ij}}{d_i}}{r_j}$$

Almost all the performance lies on this simple heuristic. In fact, modifying it may dramatically increase the number of nodes. This heuristic fits the customers which involve high revenues and cheap costs in the most filled facilities. In a case of equality, we take the variable for which $\dfrac{d_i u_i}{c_{ij}}$ is maximal.

3.3. Variable fixing

To restrict the choice of the variables during the branch and bound, one can fix some variables which will never lead to an optimal solution.

Let r_j the residual capacity of the facility j. It is obvious that, when $d_i > r_j$, x_{ij} can be set to 0.

Moreover, the primal-dual method for (SSTP) gives us the reduced costs $\bar{c}_{ij}$ of y_{ij}. Let y_{ij} be an optimal solution of the continuous relaxation of (SSTP). Let $\bar{c}_{ij}$ the reduced cost of y_{ij} and z^{UB} an upper bound for (SP). If $z(y_{ij}) + d_i\bar{c}_{ij} > z^{UB}$ then x_{ij} can be fixed to 0 without any risk. Indeed, $\bar{c}_{ij}$ is the unit cost of incrementing y_{ij}. As $y_{ij} = d_i x_{ij}$, we need to increment it d_i times to fix x_{ij} to 1. So the cost of fixing x_{ij} from 0 to 1 is $d_i\bar{c}_{ij}$.

The variables are fixed in a local scope. The fixing has only an effect in the current node and its children.

3.4. Initial upper bound

Before the beginning of the branch and bound, we try to create an initial feasible solution as an upper bound. The following method is inspired from *Vogel's Approximation Method* [18].

We denote $\min_j^k \{c_{ij}\}$ the k-th smallest element c_{ij} in the row i. Let $\text{reg}(i) = \min_j^2 \{c_{ij}\} - \min_j^1 \{c_{ij}\}$ be the *regret* of the row i. In the case that for a customer i, there is only one available facility j, we define $\text{reg}(i) = c_{ij}$.

Algorithm 1 Vogel's Approximation Method for SSTP

Require: I a set of unassigned customers.
Ensure: An initial solution x if the method succeed.

```
procedure VAM
    x_{ij} ← 0 , ∀i ∈ I, ∀j ∈ J
    r_j ← q_j , ∀j ∈ J
    while I ≠ ∅ do
        i* ← argmax_{i∈I} reg(i) such that d_{i*} ≤ r_j
        if i* does not exist then
            Failure
        j* ← argmin_{j∈J_{i*}} {c_{i*j}} such that d_{i*} ≤ r_{j*}
        if j* does not exist then
            Failure
        r_{j*} ← r_{j*} − d_{i*}
        I ← I \ {i*}
        x_{i*j*} ← 1
```

This method can fail, so in this case, no upper bound is available. Usually, this method is useful only for problems in which the knapsack constraints (18) are loose. In our case, the initial bound will fail in a large majority of cases, because they are tight. But this procedure has a very negligible time comparing to the saved effort when the method succed, because it finds a solution very close to the optimal.

3.5. Summary

We provide a simple *best first search* branch and bound procedure which relies on two different bounds. The *Lagrangian* and the *continuous* relaxations complete each other by giving multipliers to the other and giving an integer solution and/or a better lower bound. These two bounds are very fast to compute using a primal-dual algorithm for the continuous relaxation and a simple branch and bound for the knapsacks. A simple and efficient heuristic for choosing the variable and a fixing procedure are provided to reduce significantly the number of nodes. We have presented an approximation method to find a good initial upper bound in some cases.

4. A two-step method for multi-objective SSCFLP

4.1. Paving

The following method is based on the method in [11]. Probably the most natural way to solve this problem is to branch on opening variables y_j, in order to solve assignment subproblems later. We call this branching step *paving*. The main idea of the paving is to split the objective space into boxes. So, a *box* is a subproblem in which the opening facility variables are fixed to 0 or 1. These boxes are bounded by their lexicographically optimal solutions. All the non-dominated solutions of the subproblem are contained in the box.

With each box is associated an *origin* which is a point computed by accumulating the costs of opening the fixed facilities. The bounds of a box are computed by finding the lexicographically optimal solutions of its subproblem. In the bi-objective case, consider x_1^* and x_2^* these lexicographically optimal solutions. This means, when the horizontal axis is for z_1, the coordinates of the box are $(z_1(origin) + z_1(x_1^*), z_2(origin) + z_2(x_1^*))$ for the top-left corner and $(z_1(origin) + z_1(x_2^*), z_2(origin) + z_2(x_2^*))$ for the bottom-right corner.

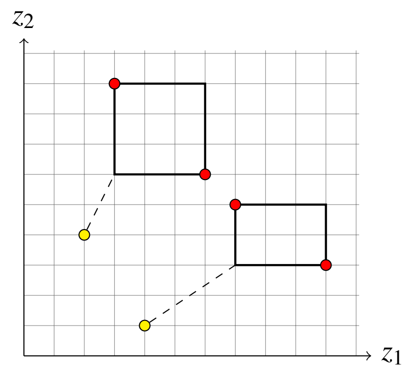


Figure 1: A paving with two boxes (with their respective origin) and two objectives

During the paving step, we consider two list of boxes : a *waiting* list $\mathcal{W}$ and a list of *computed* boxes C, i.e. boxes for which the lexicographically optimal solutions are known. During the whole algorithm, we maintain $\mathcal{S}$ as the pool of non-dominated integer solutions. When we add a solution in $\mathcal{S}$, we check whether it is dominated by an other solution of $\mathcal{S}$. In this case, the incoming solution is deleted. Otherwise, we delete the solutions in $\mathcal{S}$ that are dominated by the incoming.

4.1.1. Expansion

A *branch & bound* on the boxes is performed, with a special branching rule called *expansion*. At initialization, a fake box with no open facility is first expanded. At each expansion, several children are created and added to the waiting list $\mathcal{W}$, with respect to the following scheme. Let k be the greatest index of the open facilities. We create $n - k$ children with exactly one additional facility having a greater index than k by flipping one of the trailing zeros. The facilities before k always stay untouched. This ensures each box is unique during the procedure. The parent box is never deleted, unless it is dominated (see next sections).

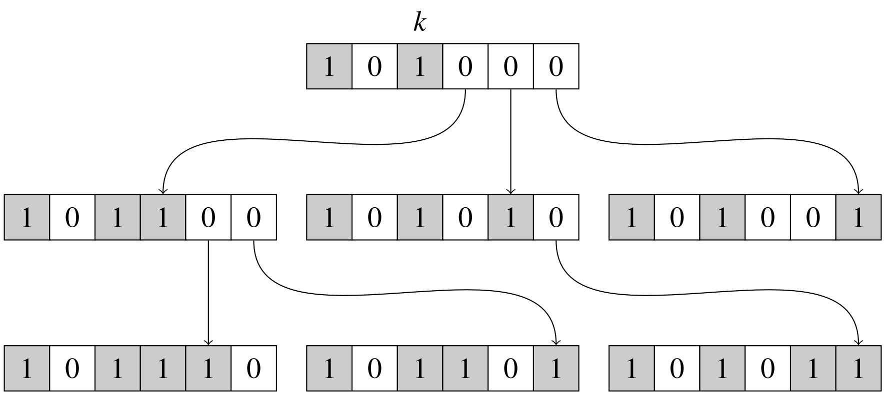


Figure 2: Expansion of a box and its children

To avoid some infeasible branches, children guaranteed to be infeasible in all cases are skipped. For that, we compute the *potential capacity* R, i.e. the sum of the capacities of the trailing facilities (i.e. that have an index greater than the k of the child). We call the *available capacity* Q the sum of the capacities of the open facilities. If the sum of available and potential capacities is strictly less than the total customer demand, the child is not created and the expansion stops. At the end, a sorting procedure is run. For more explainations, please see next sections.

Algorithm 2 Box expansion

Require: $\mathcal{W}$ a list of waiting boxes
Require: J a set of open facilities

procedure EXPAND($\mathcal{W}, J$)
  $k \leftarrow \max\{j \in J\}$
  $D \leftarrow \sum_{i=1}^{m} d_i$ ▷ Total demand
  $Q \leftarrow \sum_{j=1}^{k} y_j q_j$ ▷ Available capacity
  $R \leftarrow \sum_{j=k+1}^{n} q_j$ ▷ Potential capacity
  $j \leftarrow k + 1$
  while $j \leq n$ **and** $Q + R \geq D$ **do**
    $\mathcal{W} \leftarrow \mathcal{W} \cup (J \cup \{j\})$ ▷ Add a child box (with j open) to the waiting list
    $R \leftarrow R - q_j$ ▷ Update potential capacity
    $j \leftarrow j + 1$
  SORT($\mathcal{W}$) ▷ Sort the boxes by fronts of origins

To ensure the enumeration will not be exhaustive, several dominance tests are made for pruning. Dominance tests are made by comparing a lower bound point or the ideal point of a box with the solutions in $\mathcal{S}$, the pool of non-dominated solutions. If such a point is dominated by a solution in $\mathcal{S}$, the box can be eliminated, because all the feasible points of the box are inside the dominance cone of the previous solution. In other terms, $\mathcal{S}$ is an upper bound front of the final Pareto-front.

4.1.2. Dominance by origin

Dominance by origin was proposed for solving the *Uncapacitated Facility Location* [11, 3, 1]. Origin is a very fast lower bound to compute, as it is only a sum, so it is always the first test performed. All solutions of the subproblem are dominated by its origin. Therefore, by transitivity, solutions of the global problem that dominate the origin of a box, dominate also every solution inside the latter. Moreover, child subproblems will be dominated, because their origin is dominated by their parents' origin. Indeed, opening new facilities can only increase opening costs.

Origin is fast but not very efficient for our purpose. Only relying on the origin can run useless and very slow exact bound computations. It is also convenient to find other lower bounds which approximate the ideal point better.

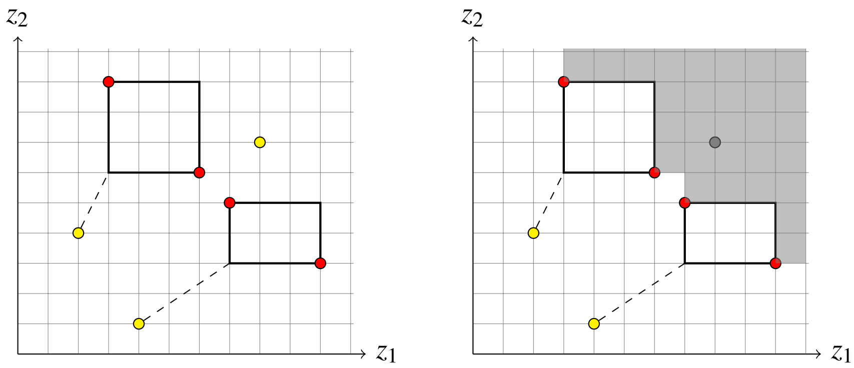


Figure 3: Dominance by origin or lower bound (exact bounds not computed yet)

4.1.3. Dominance by inherited lower bound

To speed up box elimination, we will use a *inherited lower bound* added to the *origin*. A lower bound is said *inherited* if it works for a subproblem and all its children subproblems. The idea is to prune a box and all its children.

A lower bound ($LB0$) convenient to compute is the integer solution of the subproblem in which we consider all the facilities open. We compute it independently for all objectives. This value can be computed once at initialization and added to the origin of each box. For that, we simply run our previous *branch and bound* solver for each objective. Because all the facilities are included, the knapsack constraints are loose and this makes the solving quite fast.

An other lower bound ($LB1$) to be checked is the solution of the continuous relaxation of the subproblem (i.e. with single source constraints relaxed) in which we open all the trailing closed facilities. Because of the expansion rule, which considers only the trailing facilities, even all the children will have objectives greater than this bound. Instead of using a general simplex algorithm, we run our specialized primal-dual transportation algorithm to compute this bound [10, 20]. The reason why we don't take the integer solution is the high computation time to get it for each box.

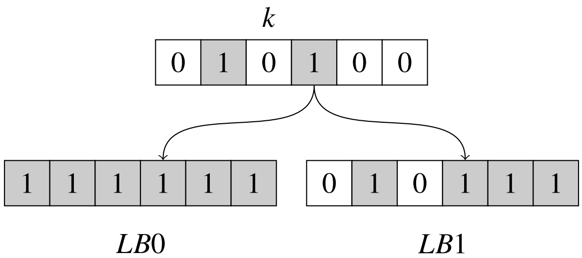


Figure 4: Open facilities in inherited lower bounds

We can skip the expansion of a box dominated by an *inherited* lower bound, because this kind of bound can only increase in children boxes. ($LB1$) can be less than ($LB0$), due to the fact that ($LB0$) is from an integer solution, while ($LB1$) is from a fractional solution. But ($LB1$) is much faster to compute than ($LB0$), due to relaxation of the single source constraints which avoid to use a heavy branch & bound.

4.1.4. *Dominance by non-inherited lower bound*

When previous dominance tests fail, the goal is to skip the expansive computation of the bounds of the box. We compute this lower bound ($LB2$) using the continuous relaxation of the subproblem. This bound is computed in the same way as ($LB1$) but here we do not touch at the closed facilities.

However, the children of the box cannot be pruned, because adding new facilities in children boxes can decrease the value of this bound. Indeed, adding new facilities can add more profitable assignments.

4.1.5. *Dominance by exact bounds*

After computing the exact bounds of a box, the last dominance test is to check the whole box is inside the dominance cone of one solution of the global problem. For that, we simply consider the *ideal point* of the box, i.e. the point given by the best objective values of the lexicographically optimal solutions. Then, we check if it is dominated by a solution in $\mathcal{S}$. In this case, the box is also deleted.

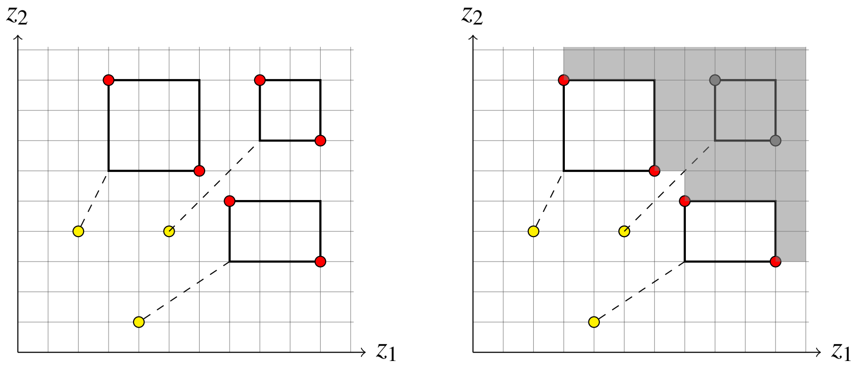


Figure 5: Dominance by bounds

4.1.6. *Sorting the pending boxes*

We can observe that non-dominated boxes have often non-dominated origins, and that the ordering of the pending boxes can influence the dominating rules. So, instead of using an unordered waiting list of boxes, we sort them by Pareto-fronts as *NSGA-II* [4]. All the non-dominated origins are assigned the rank 1, the next non-dominated, the rank 2, and so on. So the boxes with a non-dominated origin will always be treated in priority, in order to maximize the probability of pruning the next boxes. Instead of making a random choice in the first front, we can sort it by increasing available capacity, because in spite of the tightness of the subproblems, they are more likely to give good solutions for pruning. The sorting procedure is called at each expansion, when boxes are added to $\mathcal{W}$.

4.1.7. *Sorting the computed boxes*

In the same spririt as the pending boxes, we sort the computed boxes $\mathcal{C}$, but by fronts of ideal points. The boxes with non-dominated ideal points are more likely to compute non-dominated solutions for the global problem. This sorting procedure is run only once and at the end of the paving, since no more boxes will be added for the next step.

4.1.8. Computing supported solutions

It is convenient to compute some exact supported solutions in advance for two reasons. It improves the size of the dominated region which can prune boxes even more. And it speeds up the generation step by making earlier dominated labels in a label setting algorithm. To compute them, we can use *Aneja and Nair*'s dichotomic method [2] which is the first step of the *bi-objective two-step method* [6].

The deal is to make the generation step easier without making the paving step too long. Indeed, the higher the number of computed solutions is, the less these computations are profitable, because newly dominated regions are usually smaller. However, the number of supported solutions computed can be naturally limited by checking dominance. Indeed, when all the points that are used for the weighted sum coefficients are simultaneously dominated by a same point, the result will be a dominated point, because it is "between" the initial points. The dominance check is performed by the function IsDominatedByUnique.

We use an iterative version of the algorithm instead of a recursive one by using a queue of jobs $\mathcal{W}$. So before adding a new tuple of points as a new job in $\mathcal{W}$, we check that all points of the tuple are not dominated by $\mathcal{S}$. The number of computed jobs is limited by the parameter *maxpoints*.

Algorithm 3 Adapted Aneja and Nair (bi-objective)

Require: x^{1*}, x^{2*} lexicographically optimal solutions of the box, resp. for objective 1 and 2.
Require: $z : \mathbb{Z}_+^* \times \mathbb{Z}_+^* \to \mathbb{Z}_+^* \times \mathbb{Z}_+^*$ the 2-objective function.
Require: $maxpoints \in \mathbb{Z}_+^*$ a parameter.

procedure DichotomicMethod
  $\mathcal{W} \leftarrow \{(x^{1*}, x^{2*})\}$ ▷ Waiting list of pairs of solutions
  $k \leftarrow 0$ ▷ Counter of computations
  while $\mathcal{W} \neq \emptyset$ **and** $k < maxpoints$ **do**
    $(x^1, x^2) \leftarrow$ first pair of solutions in $\mathcal{W}$
    $\mathcal{W} \leftarrow \mathcal{W} \setminus \{(x^1, x^2)\}$
    $\lambda_1 \leftarrow z_2(x^1) - z_2(x^2)$
    $\lambda_2 \leftarrow z_1(x^2) - z_1(x^1)$
    $x^0 \leftarrow$ SolveWeightedSum(λ)
    $k \leftarrow k + 1$
    if $\lambda_1 z_1(x^0) + \lambda_2 z_2(x^0) < \lambda_1 z_1(x^1) + \lambda_2 z_2(x^1)$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{x^0\}$
      if not IsDominatedByUnique$(x^0, x^1, \mathcal{S})$ **then**
        $\mathcal{W} \leftarrow \mathcal{W} \cup \{(x^1, x^0)\}$
      if not IsDominatedByUnique$(x^0, x^2, \mathcal{S})$ **then**
        $\mathcal{W} \leftarrow \mathcal{W} \cup \{(x^0, x^2)\}$

4.1.9. Paving algorithm

The complete paving algorithm is given below.

Algorithm 4 Paving

Ensure: C is a list of non-dominated boxes

procedure Paving
    $\mathcal{W} \leftarrow \emptyset$ ▷ Waiting list
    $C \leftarrow \emptyset$ ▷ Computed boxes
    $D \leftarrow \sum_{i=1}^{m} d_i$ ▷ Total demand
    ComputeLB0() ▷ LB for which all facilities are open (precomputed)
    Expand($y = (0, \ldots, 0)$) ▷ Expand the first (fake) box
    while $\mathcal{W} \neq \emptyset$ **do**
        $B \leftarrow$ first box in $\mathcal{W}$
        $\mathcal{W} \leftarrow \mathcal{W} \setminus \{B\}$
        if not DominatedByInheritedLB($B, \mathcal{S}$) **then**
            Expand(B)
            if AvailableCapacity(B) $\geq D$ **and not** DominatedByNonInheritedLB($B, \mathcal{S}$) **then**
                ComputeExactBounds(B)
                if IsFeasible(*current*) **and not** DominatedByExactBounds($B, \mathcal{S}$) **then**
                    $\mathcal{S} \leftarrow \mathcal{S} \cup$ ComputedSolutions(B) ▷ Add new solutions to the pool
                    FilterDominatedBoxes($C, \mathcal{S}$) ▷ New $\mathcal{S}$ can dominate previous boxes
                    $C \leftarrow C \cup \{B\}$ ▷ B is added to the paving
    Sort(C) ▷ Sort the final paving by fronts of ideal points

4.2. Generation

When the paving is finished, we have a set of multi-objective SSTP subproblems to solve. The method proposed for the *UFLP* [11, 3, 1] transforms the subproblems into *Shortest Path Problems* (SPP) and uses *Martins' algorithm* [15], an extension of the *Dijkstra algorithm* for the multi-objective case, to find all efficient paths. We propose a direct extension of this method by transforming the subproblem into a *Constrained Shortest Path Problem* [5, 8].

First in a directed graph we create one vertex v_i per customer i and a starting vertex v_0. Between every pair (v_{i-1}, v_i) of consecutive vertices, we create one edge $(e^j_{i-1,i})$ per open facility j with the costs $c^k_{ij}, k \in \{1, \ldots, p\}$. This results in a *directed acyclic graph* in which every path between v_0 and v_m corresponds to a unique combination of assignments. Also selecting the edge $(e^j_{i-1,i})$ means the customer i is assigned to the facility j.

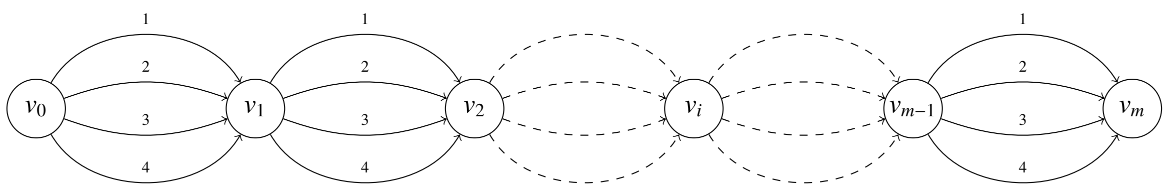


Figure 6: A graph associated to a subproblem with 4 open facilities

For our algorithm, we use a *label setting* principle. Each vertex has a list of labels, initially empty.

For the *UFLP*, labels contained only partial objective values. To cope with capacity constraints, we have to take into account the residual capacities of the open facilities. So in our algorithm, a label L contains partial objective values (z_k^L), residual capacities (r_j^L) and the total residual capacity (R^L) (i.e. sum of the residual capacities). To write labels, the following notation will be used : $[z_1^L, \ldots, z_p^L | r_1^L, \ldots, r_n^L]$.

We begin with the vertex v_0 in which we put the first label $[z_1^{orig}, \ldots, z_p^{orig} | r_1, \ldots, r_n]$, with z_k^{orig} the value of the origin for the objective k and $r_j = q_j$ the initial capacity of the open facility j. At each iteration $i \in \{1, \ldots, m\}$, we focus on a vertex v_{i-1} and try to expand its labels to the vertex v_i by passing through each leaving edge. At each expansion through an edge ($e_{i-1,i}^j$), the new label objectives are increased by $c_{ij}^k, k \in \{1, \ldots, p\}$ and the residual capacity of the facility j is decreased by d_i. Feasibility and dominance criteria are always checked in order to eliminate a maximum number of labels. When arrived to the last vertex v_m, the labels are filtered by objective-dominance only. We also have in the last vertex all the non-dominated solutions of the subproblem, to merge in a global list. The final global list is filtered to remove dominated solutions that were non-dominated only in the scope of the subproblem.

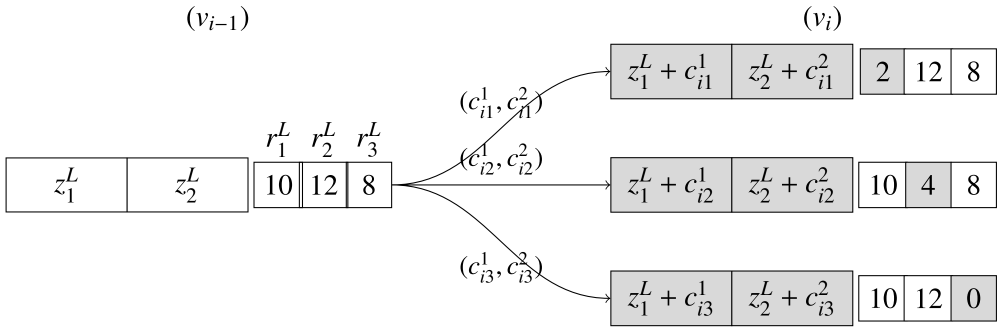


Figure 7: Expansion of a label from v_i to v_{i+1}, 3 open facilities, $d_i = 8$

4.2.1. Algorithms

The generation algorithm is a simple loop over boxes. A label setting is applied on each box. After a box is processed by the generation, we may have some new non-dominated solutions which dominate other boxes. FilterBoxes removes all newly dominated boxes according to the global non-dominated solution set $\mathcal{S}$ in order to process less boxes. FilterSolutions removes all the dominated solutions.

Algorithm 5 Generation

Require: *BoxList* a set of non-dominated boxes, $\mathcal{S}$ a global set of non-dominated solutions
Ensure: $\mathcal{S}$ a complete set of non-dominated solutions

```
procedure Generation(BoxList, S)
    for all B ∈ BoxList do
        G ← BuildGraph(B)
        S ← S ∪ LabelSetting(G)
        FilterBoxes(BoxList, S)
    S ← FilterSolutions(S)
    return S
```

Before applying the label setting, the graph associated to the subproblem is computed using BUILDGRAPH. It is possible to sort the nodes in order to measure the influence of the ordering. The recommended ordering is by decreasing demand, because it leads to more infeasible labels earlier and less enumeration. At this step, it is also recommended to precompute the values of total, minimal and maximal remaining demands at each vertex.

Algorithm 6 Build graph

Require: B a non-dominated box, I a set of customers, J^B the set of open facilities
Ensure: G a directed acyclic graph

```
procedure BuildGraph(B)
    V ← {v_0}
    E ← ∅
    I ← SortByDecreasingDemands(I)
    for all i ∈ I do
        V ← V ∪ {v_i}
        for all j ∈ J^B do
            E ← E ∪ {e^j_{i−1,i}}
    return G = (V, E)
```

The label setting (LABELSETTING) performs from v_0 to v_m and tests each label on each leaving edge ($e^j_{i-1,i}$). The acceptance criteria of LABELEXPANSION are given below. When expanding a label $L = [z^L_1, \dots, z^L_p | r^L_1, \dots, r^L_n]$ through an edge ($e^j_{i-1,i}$), the following operations are performed :

$$z^L_k \leftarrow z^L_k + c^k_{ij}\ , \quad \forall k \in \{1, \dots, p\}$$

$$r^L_j \leftarrow r^L_j - d_i$$

$$R^L \leftarrow R^L - d_i$$

Algorithm 7 Label Setting

Require: G a directed acyclic graph
Ensure: S a set of non-dominated solutions inside the box

```
procedure LabelSetting(G = (V, E))
    ℒ(v_0) ← {[z_1^origin, ..., z_p^origin | r_1, ..., r_n]}          ▹ First label
    for all v_i ∈ V do
        for all e^j_{i,i+1} ∈ E do
            for all L ∈ ℒ(v_i) do
                LabelExpansion(L, e^j_{i,i+1})
        FilterLabels(ℒ(v_{i+1}))
    S ← ℒ(v_m)                        ▹ The last vertex contains all the final solutions
    return S
```

The FilterLabels procedure assumes the algorithm is implemented using a linear list. It removes all the dominated labels. To avoid a huge number of dominance tests in a high-dimensional space, it is recommended to use a *quadtree* [17, 16]. This structure divides the objective space into 2^p parts on each node (where p is the dimension of the space), and skips the dominance tests in regions where the stored points are guaranteed to be non-dominated by new inserted points. In this case, the FilterLabels procedure is not needed.

4.2.2. Label dominance criteria

One of the main problems of the capacitated context is that an objective-dominant label does not necessarily lead to feasible solutions. The consequence is that a such a label cannot be deleted without taking into account the residual capacities.

Let $z_k^L, (k \in \{1 \dots p\})$ be the current objective values for the label L and r_j^L $(j \in J)$ be its residual capacities. The label dominance criteria are given by definition 1.

Definition 1. Let L_1 and L_2 be two given labels in the same list.
L_1 *dominates* L_2 if and only if :

$$z_k^{L_1} \leq z_k^{L_2} \ , \ \forall k \in \{1 \dots p\} \text{ and } r_j^{L_1} \geq r_j^{L_2} \ , \ \forall j \in J$$

with at least one strict inequality.

A label dominated according to the definition 1 will always expand dominated labels.

Using the non dominated front obtained by the paving and the previous boxes, we can delete even more labels. The following definition 2 is very convenient because it does not need to take the residual capacities into account. The labels dominated according to this criterion will always lead to dominated solutions.

Definition 2. Let L be a label and S a feasible solution obtained before.
S *dominates* L if and only if :

$$z_k^S \leq z_k^L \ , \ \forall k \in \{1 \dots p\}$$

with at least one strict inequality.

The previous definitions are also extensible for lower bounds of z^L (i.e. $z_k^S \leq LB(z_k^L)$). We have chosen to check the dominance by only using lower bounds. The lower bounds we have chosen are the continuous relaxation and the previously described Lagrangian relaxation used for our branch and bound for SSTP.

4.2.3. Label infeasibility criteria

A label L is said *infeasible* if itself and its successors do not satisfy the capacity constraints. A list of criteria is proposed in order to delete more labels.

Criterion 1 (Capacity violation) *One of the capacity contraints is violated.*

$$\exists j \in J \ , \ r_j^L < 0$$

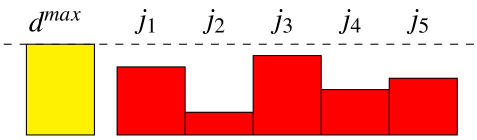


Figure 8: Residual capacities not able to cover the maximal demand

Criterion 2 (Max demand violation) *The maximal demand d^{max} to be assigned cannot fit into an existing facility.*

$$\forall j \in J \ , \ r_j^L < d^{max}$$

Definition 3 (Full facility). Let j be a facility, r_j^L its residual capacity and d^{min} the minimal demand to assign. The facility j is said *full* if and only if $r_j^L < d^{min}$.

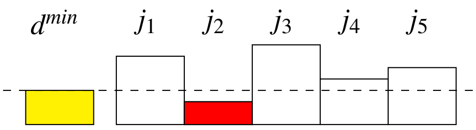


Figure 9: A *full* facility i.e. not fitting the minimal demand

When a facility is said *full*, its residual capacity can be set to zero, also reducing the total amount of residual capacity. The operations are $R^L \leftarrow R^L - r_j^L$ and $r_j^L \leftarrow 0$

Criterion 3 (Total demand violation) *The total amount of residual capacity R^L is not able to cover the total unassigned demand D^{v_i}.*

$$R^L < D^{v_i}$$

Criterion 4 ($P||C_{max}$ bound) *A lower bound of the $P||C_{max}$ associated problem is strictly greater than the maximal residual capacity.*

$$LB(P||C_{max}) > r^{maxL}$$

The $P||C_{max}$ problem can be seen as a relaxation of the SSTP, as the capacity constraints are ignored and the objective is not to reduce the costs but the makespan. For this purpose, we consider demands as tasks. One task per facility is also added, with a duration of $r^{maxL} - r_j^L$. Let I^L be the set of unassigned customers. Denote T the set of task durations and M the set of machines.

$$T = \left\{d_i \in I^L \mid d_i > 0\right\} \cup \left\{r^{maxL} - r_j^L \mid r^{maxL} - r_j^L > 0 \ , \ j \in J^B\right\}$$

$$M = \left\{j \in J^B \mid r_j^L > 0\right\}$$

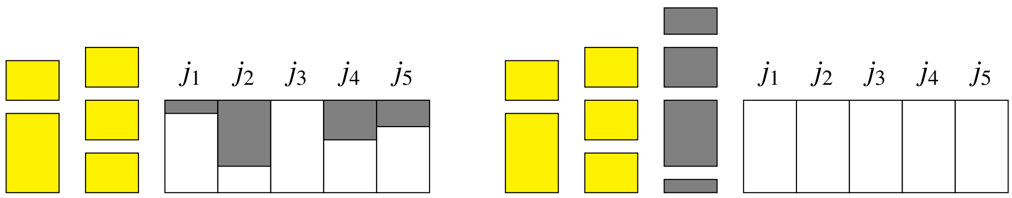
Let $p_i \in T$ the durations of the tasks. The linear program is given.

$$\min \; C_{max} \tag{33}$$

$$\sum_{j \in M} x_{ij} = 1, \qquad \forall p_i \in T \tag{34}$$

$$\sum_{p_i \in T} p_i x_{ij} \leq C_{max}, \qquad \forall j \in M \tag{35}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall p_i \in T, \forall j \in M \tag{36}$$



(a) SSTP problem (b) Its $P||C_{max}$ associated problem

Figure 10: A $P||C_{max}$ problem associated to SSTP

When p_i are ordered by decreasing durations, the lower bound of C_{max} can be computed as following :

$$C_{max} \geq \max \left(p_0 \; , \; p_{n-1} + p_n \; , \; \sum_{i \in T} \frac{p_i}{n} \right)$$

If C_{max} is greater than the maximal residual capacity, it means that the remaining demands cannot fit into the available space of the facilities anymore without splitting them. This label is also considered infeasible.

4.2.4. Other label deletion criteria

Criterion 5 (Wasted facilities) *The number of unused facilities is strictly greater than the number of remaining customers to assign.*

When a label has more unused facilities than the number of remaining customers to assign, it means the final solutions obtained will not use all the facilities. Such a solution is not interesting because it means that an useless extra facility was open ; the resulting solutions could also be computed in an other box without extra opening costs. This situation should never happen because of the tight knapsack constraints and more customers than facilities in our problems, but may be interesting in the case the number of open facilities is high.

4.2.5. Summary

The table 11 summarizes the different label deletion criteria we will use.

Criterion	Condition
Dominance (label)	$\exists L', z_k^{L'} \leq z_k^L, r_j^{L'} \geq r_j^L, \forall k \in \{1 \dots p\}, \forall j \in J$ with at least one strict inequality
Dominance (solution)	$\exists S \in \mathcal{S}, z_k^S \leq z_k^L, \forall k \in \{1 \dots p\}, \forall j \in J$ with at least one strict inequality
Capacity violation	$\exists j \in J, r_j^L < 0$
Max demand violation	$\forall j \in J, r_j^L < d^{max}$
Total demand violation	$R^L < D^{v_i}$
$P\|\|C_{max}$ bound	$LB(P\|\|C_{max}) > r^{maxL}$

Figure 11: Table of the label deletion criteria

The capacity, max demand and total demand violations are classified as *infeasibility* criteria.

The algorithm for label expansion on an edge is given.

Algorithm 8 Label Expansion

Require: L a label to expand, $e_{i,i+1}^j$ an edge

procedure LabelExpansion($L, e_{i,i+1}^j$)
    if $r_j^L - d_i < 0$ **then** ▷ Capacity violation
        return false
    if $r_j^L - d_i < d^{min}$ **and** $R^L - r_j^L < D^{v_i}$ **then** ▷ Total demand violation
        return false ▷ $R^L = \sum_{j \in J} r_j^L$
    $L' \leftarrow L$ ▷ Compute new label
    $\forall k \in \{1, \dots, p\}$, $z_k^{L'} \leftarrow z_k^{L'} + c_{ij}^k$
    $r_j^{L'} \leftarrow r_j^{L'} - d_i$
    $R^{L'} \leftarrow R^{L'} - d_i$
    if $r_j^{L'} < d^{min}$ **then** ▷ Full facility ?
        $R^{L'} \leftarrow R^{L'} - r_j^{L'}$
        $r_j^{L'} \leftarrow 0$
    if $r^{maxL'} < d^{max}$ **then** ▷ Max demand violation
        return false
    if $LB(P||C_{max}) > r^{maxL'}$ **then** ▷ $P||C_{max}$ bound
        return false
    if $LB(z_k^L)$ dominated by any feasible solution $S \in \mathcal{S}$ **then**
        return false
    $\mathcal{L}(v_{i+1}) \leftarrow \mathcal{L}(v_{i+1}) \cup \{L'\}$ ▷ Accept the label
    return true

5. Experimental results

5.1. Machines

Two machines were used for the experiments, denoted by $\mathcal{M}_1$ and $\mathcal{M}_2$. Let us give the configurations.

Machine	Processor	Memory
$\mathcal{M}_1$	Intel(R) Xeon(R) CPU X5550 @ 2.67GHz x 8	24GB
$\mathcal{M}_2$	Intel(R) Core(TM) 2 Duo CPU E8500 @ 3.16GHz x 2	3GB

Figure 12: The machines used for the experiments

5.2. Data set

The tested instances are aggregation of single objective SSCFLP instances from Elena Fernandez' website (`http://www-eio.upc.es/~elena/`). From two instances `px.txt` and `py.txt`, we construct `Fx-y.txt` by setting the first objective coefficients from `px.txt` and the second from `py.txt`. The demands and capacities are taken from `px.txt`. We have taken instances in order to have measurable times for our algorithm and our machines, with 20 customers and 10 facilities. Five arbitrary instances will be more subject to experiments than the others : they are called "main" instances.

Main	F1-2	F2-3	F3-4	F4-5	F5-6
Secondary	F1-3	F1-5	F2-4	F2-6	F3-6
	F1-4	F1-6	F2-5	F3-5	F4-6

Figure 13: Tested instances

5.3. Branch and bound for SSTP

We have made 10 runs on $\mathcal{M}_2$ on 278 SSTP problems taken from the obtained pavings of our main instances. We consider the computations of the lexicographical solutions and compare the results between our branch and bound method and SCIP. The detailed results are in the appendix.

We observe that our branch and bound is faster in 92.80% of cases. The average speedup factor is 5.46, but there are in average 34% more nodes in our method than in SCIP. So our method computes more nodes but each node is processed faster than in SCIP. Our method seems to be even faster than SCIP when the instances are very difficult.

We also measured the times according to the tightness coefficient defined in (15). It shows that when the tightness coefficient is greater, our algorithm is more likely to take more time to solve the instance, but it is not always the case.

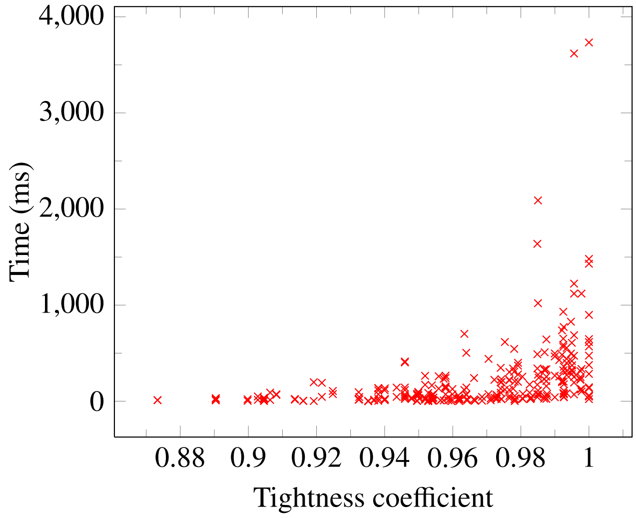


Figure 14: Relation between solving time and tightness coefficient

5.4. *Paving*

First, we will measure the pertinence of using lower bounds. Then, we will see the influence of the ordering of the waiting boxes.

5.4.1. *Using lower bounds*

For each instance, 100 runs on machine $\mathcal{M}_2$ have been done with and without lower bounds. We do not pre-compute supported solution here and the box ordering by rank and increasing total capacity is applied. As the lower bound computations are negligible comparing to the exact computations, the results show that using lower bounds reduces the execution time and many boxes are removed before being computed.

	Created boxes		Computed boxes		Non-dominated boxes
Instance	Only origin	Origin + LB	Only origin	Origin + LB	
F1-2	208	185	60	17	10
F1-3	218	201	76	27	16
F1-4	217	194	62	14	12
F1-5	221	205	91	57	42
F1-6	228	221	105	68	54
F2-3	204	200	38	16	10
F2-4	201	197	26	9	6
F2-5	199	189	34	17	6
F2-6	207	207	61	44	33
F3-4	289	285	47	29	17
F3-5	278	272	43	18	8
F3-6	304	295	101	65	54
F4-5	434	420	63	25	20
F4-6	462	442	130	56	36
F5-6	214	201	87	53	37

Figure 15: Comparison of number of boxes between an execution with only origin and with lower bounds

Instance	Computed solutions		Time (s)	
	Only origin	Origin + LB	Only origin	Origin + LB
F1-2	120	34	14.9479	7.97823
F1-3	152	54	23.2523	16.4067
F1-4	124	28	20.3808	9.2525
F1-5	182	114	12.281	10.4534
F1-6	210	136	18.4341	16.8221
F2-3	76	32	10.2544	6.03555
F2-4	52	18	10.7403	5.33655
F2-5	68	34	20.996	9.03586
F2-6	122	88	10.4762	9.57405
F3-4	94	58	36.2442	22.0244
F3-5	86	36	25.3635	9.61901
F3-6	202	130	50.1329	43.7725
F4-5	126	50	5.47842	3.02635
F4-6	260	112	10.473	5.28885
F5-6	174	106	48.0936	36.774

Figure 16: Comparison of times between an execution with only origin and with lower bounds

5.4.2. Ordering of the pending boxes

The ordering of the pending boxes matters in the box elimination procedure. Indeed, if the worst boxes are treated first, they will involve several bound computations that could be avoided by a dominance test with a good box computed before. It also useful to define an ordering.

The following data compares the paving with no ordering and with front-ordering by origin and increasing total capacity Q. We don't compute any supported solution here. For each instance, 100 runs on $\mathcal{M}_2$ have been done and we have taken the average time.

Instance	Created boxes		Computed boxes		Non-dominated boxes
	No ordering	By origin	No ordering	By origin	
F1-2	206	185	32	17	10
F1-3	207	201	36	27	16
F1-4	209	194	24	14	12
F1-5	216	205	65	57	42
F1-6	222	221	71	68	54
F2-3	200	200	16	16	10
F2-4	199	197	11	9	6
F2-5	198	189	22	17	6
F2-6	207	207	44	44	33
F3-4	286	285	31	29	17
F3-5	274	272	21	18	8
F3-6	295	295	65	65	54
F4-5	430	420	39	25	20
F4-6	442	442	56	56	36
F5-6	202	201	56	53	37

Figure 17: Number of created, computed and non-dominated boxes according to the ordering

On these instances, in most cases there are strictly less boxes computed. The consequence is generally a better or equivalent average time. The origin is a good indicator, but we may be able to do better in case

	Computed solutions		Time (s)	
Instance	No ordering	By origin	No ordering	By origin
F1-2	64	34	10.223	7.97823
F1-3	72	54	17.2818	16.4067
F1-4	48	28	10.0973	9.2525
F1-5	130	114	11.1162	10.4534
F1-6	142	136	16.605	16.8221
F2-3	32	32	6.16077	6.03555
F2-4	22	18	5.84697	5.33655
F2-5	44	34	13.4541	9.03586
F2-6	88	88	9.26711	9.57405
F3-4	62	58	22.6924	22.0244
F3-5	42	36	10.3465	9.61901
F3-6	130	130	42.824	43.7725
F4-5	78	50	4.46682	3.02635
F4-6	112	112	5.16381	5.28885
F5-6	112	106	38.9786	36.774

Figure 18: Number of computed solutions and time according to the ordering

the opening costs are low. Indeed, imagine all the opening costs are null, then the origin ordering does not work anymore and every box may be computed. A solution to that might be to begin with a box with all the facilities open and to expand by removing facilities instead of adding them. Because if the opening costs are low comparing to delivering costs, it is likely that we would open almost all the facilities in the efficient solutions. An other hint is to compute a lower bound at each expansion, so we could be able to take into account the delivering costs directly in the ordering. The last proposal has a very negligible influence on the results of our instances that have greater opening costs.

5.5. Generation

In the generation step, we will speak about the pertinence of the label deletion criteria. We will also analyze the progress of the number of labels during the generation. And we will focus on the importance of pre-computing supported points before the generation. Finally, we will compare our method with a generic epsilon constraint.

Erratum : a very small change in the *Adapted Aneja and Nair*'s method has been made after the measures. It is about the dominance checks : instead of checking whether the points used for the weighted sum were dominated by a same point, we checked whether the points were dominated by one or several points, not necessarily unique. So the previous implementation may skip supported points in special cases during the pre-computation. We think the following results are very close to the ones after the modification. It does not alter the final result because all the supported points are found a second time during the generation step.

5.5.1. Label deletion criteria and number of supported points

Because of the important number of boxes, we study only one of them, in particular the first generated box of the F1-2 instance. The following graphs give an idea of the typical behavior of our criteria. It presents the percentage of labels deleted according to 3 groups of criteria. Label dominance by other labels is not included. The y axis has been cutted for more visibility. We observe that the infeasibility criteria are the most involved (more than 70%). The "$P_m||C_{max}$" and the label dominance by $\mathcal{S}$ criteria occur later but remain less significant. The efficiency of the generation step also lies essentially on the infeasibility criteria, because they don't let some labels growing early.

Computing supported points before the generation makes the label dominance by $\mathcal{S}$ more occuring and earlier. In fact, the supported points restrict the search space and this permits to remove labels earlier. The increasing paving time due to computation of supported points is also compensated by a better computation time during the generation step.

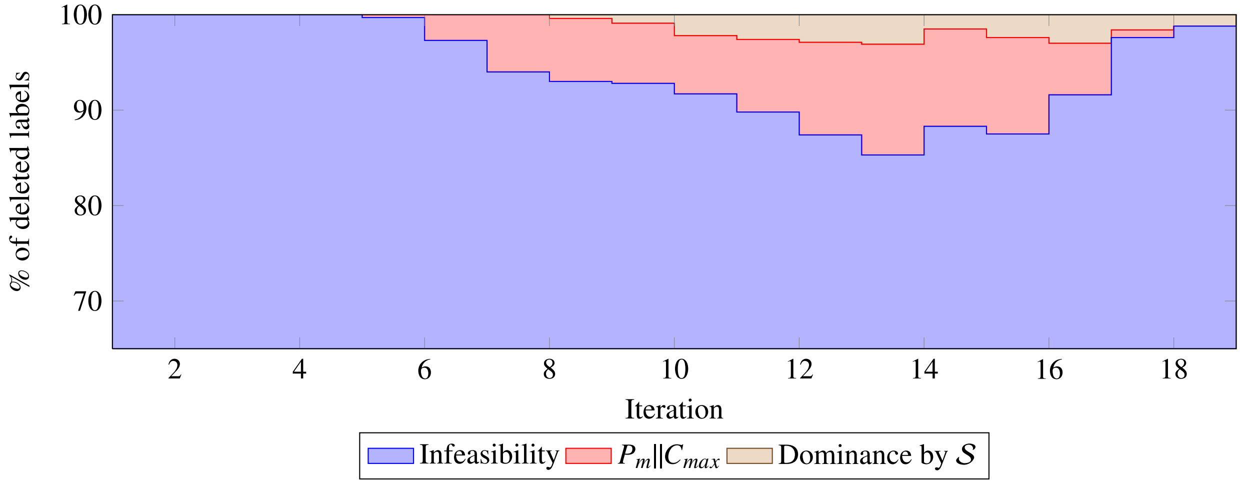


Figure 19: Part of the criteria involved in the label deletion in F1-2(1101101110) with no computation of supported points (y axis cutted)

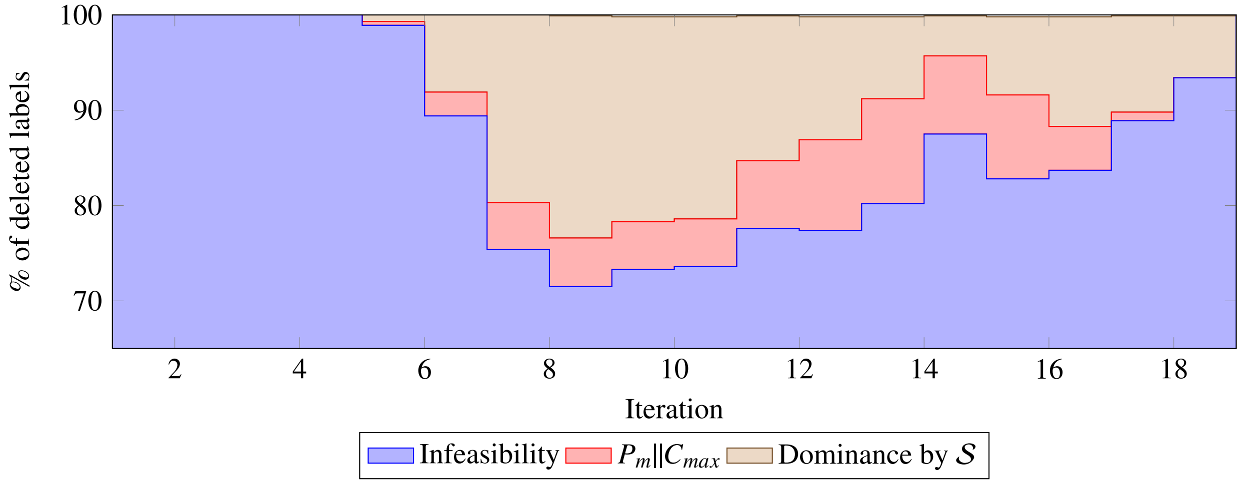


Figure 20: Part of the criteria involved in the label deletion in F1-2(1101101110) with computation of all supported points (y axis cutted)

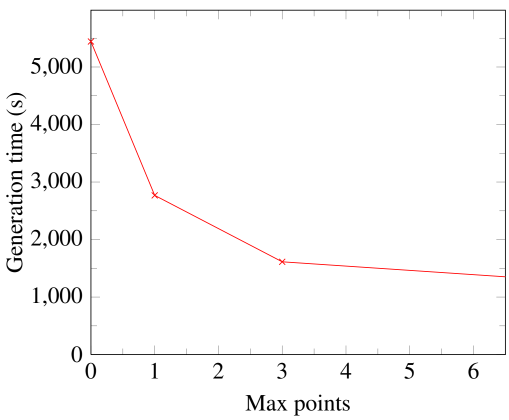
We can observe that the number of labels is increasing until the middle, reaching a peak and decreasing after. This is due to fact that almost all labels are potentially feasible at the beginning of the process. The exponentially growing number of labels makes difficult to handle them. Even if we use a quadtree structure to store them as we have done.

Iteration	#labels	Deleted by infeasibility	Deleted by $P_m\|\|C_{max}$	Deleted by domin. by $\mathcal{S}$	Elapsed time (s)
#0	7	0	0	0	0.00178
#1	43	6	0	0	0.011627
#2	222	79	0	0	0.057805
#3	1114	440	0	0	0.258304
#4	4383	2708	0	0	1.07586
#5	20766	9728	26	0	4.56381
#6	68475	54289	1472	3	20.456
#7	272292	191831	12423	140	82.3387
#8	680296	903662	64469	3138	355.809
#9	1035647	2580593	175795	24410	966.057
#10	2345762	4383348	293518	99523	1828.73
#11	3094552	10620218	902007	303787	3375.46
#12	2420455	15095700	1687309	479830	4820.16
#13	821398	12787414	1739571	454156	5283.08
#14	307825	4659950	540185	76929	5368.4
#15	108583	1753102	203779	45576	5386.06
#16	41713	624103	37399	19759	5391.04
#17	10223	252579	1736	4237	5392.32
#18	2419	59842	0	695	5392.56
#19	450	14514	0	0	5392.56

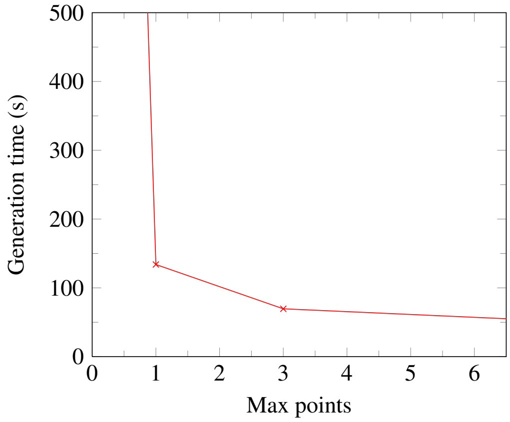
Figure 21: Execution of box F1-2(1101101110) with no pre-computed supported points (on machine $\mathcal{M}_1$), showing the number of labels deleted per criterion

Iteration	#labels	Deleted by infeasibility	Deleted by $P_m\|\|C_{max}$	Deleted by domin. by $\mathcal{S}$	Elapsed time (s)
#0	7	0	0	0	0.001837
#1	43	6	0	0	0.011894
#2	222	79	0	0	0.058722
#3	1114	440	0	0	0.262486
#4	4383	2708	0	0	1.10654
#5	20700	9728	26	73	4.69763
#6	65514	54078	1467	4900	21.2845
#7	215869	181935	11701	47585	84.7399
#8	414842	704015	50481	229459	262.756
#9	498299	1546909	106220	454442	594.1
#10	600538	2117137	145197	611520	851.375
#11	563629	2719616	248578	532088	1100.54
#12	291977	2760494	339514	462274	1241.88
#13	61811	1568317	215893	169345	1264.14
#14	19328	358235	33764	17263	1266.9
#15	3059	109315	11704	10919	1267.56
#16	951	16993	943	2353	1267.68
#17	344	5576	57	634	1267.71
#18	207	1972	0	139	1267.72
#19	96	1242	0	0	1267.72

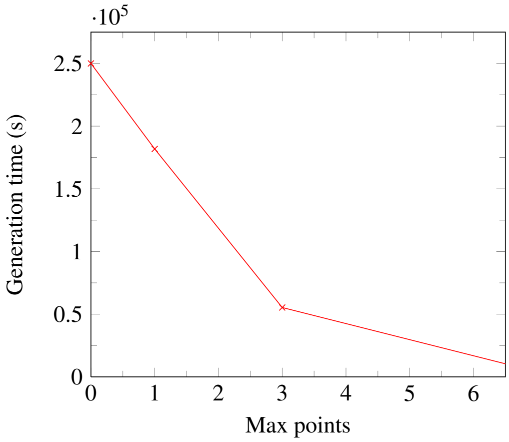
Figure 22: Execution of box F1-2(1101101110) with all pre-computed supported points (on machine $\mathcal{M}_1$)



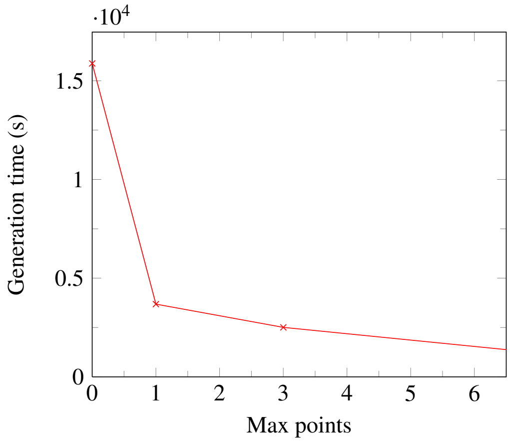
(a) F1-2



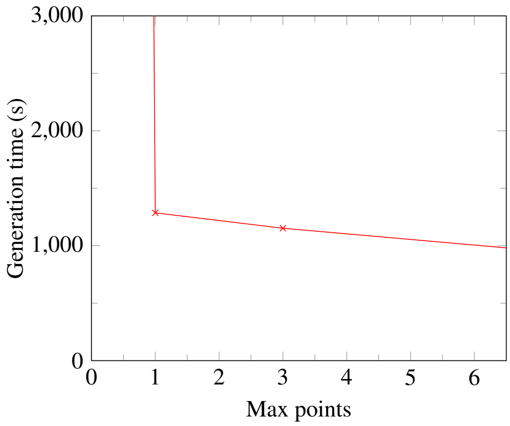
(b) F2-3



(c) F3-4



(d) F4-5



(e) F5-6

Figure 23: Relation between max number of supported points per box and generation time (on machine $\mathcal{M}_1$)

Instance	Paving time (s) No supported	All supported	Generation time (s) No supported	All supported
F1-2	5.9	11	5443	1315
F2-3	4.9	13.9	2961	53.1
F3-4	15.6	39.1	>250000	4012
F4-5	2.4	6	15876	1219
F5-6	26.3	61.9	78428	957

Figure 24: Time with or without computing the supported points (on machine $\mathcal{M}_1$)

The figures 23 show clearly that computing supported solutions is very profitable. The more we limit the number of supported points to compute, the more the generation step will take time. The modification of the *Aneja and Nair*'s algorithm makes computing all supported points not so expensive, because we skip a lot of them which are sure to be dominated. The paving step remains also quite fast.

We compared our method with a generic *epsilon constraint* method [6]. The *epsilon constraint* we implemented uses SCIP MIP solver. We present briefly the procedure. It takes the first objective as the main objective and expresses the others in the form of inequality constraints ($z_k(x) \leq \epsilon_k$), to have a mono-objective problem. Then it solves series of problems by adjusting the ϵ_k values in order to have all possible efficient solutions.

Instance	Paving	Generation	All	SCIP
F1-2	16.19	2117.41	2133.6	258.84
F2-3	19.9	104.15	124.05	136.8
F3-4	67.48	8140.08	8207.57	492.56
F4-5	6.75	2448.38	2455.14	465.79
F5-6	91.3	2014.74	2106.04	209.4

Figure 25: Comparison of times in seconds of our method and a SCIP epsilon constraint (on machine $\mathcal{M}_2$)

For these tests, we computed all the supported solutions at the paving step. We observe that the generation step is very slow compared to SCIP solver. The difficulty of handling the exponentially growing number of labels makes our method less efficient. However, the paving step is quite fast. It gives all supported points and even more : when points are supported for the subproblem but non-supported efficient for the global problem. Because the paving step gives already computed solutions and does not depend on the generation step, one can use other ways to solve the subproblems than a label setting in order to get non-supported points.

6. Conclusion

We have explored a method in two steps for the *Single Source Capacitated Facility Location Problem* in its *multi-objective* version. The single objective subproblems are solved using a specialized *branch and bound* lying on a *continuous* and a *Lagrangian lower bounds* and a good heuristic, which is measured faster than the generic MIP solver SCIP. The first step of *paving* has been improved by introducing *lower bounds* and an *ordering* on the boxes. Moreover, the computation of *supported points* using a modified *Aneja and Nair*'s method, improves the whole algorithm. The *generation* step lies on a *label setting*, shown actually inefficient comparing to an epsilon-constraint using SCIP. As we consider the paving is quite fast and gives some flexibility, the second step is open to a lot of potential. We have also to consider that the instances were small and the problem *NP-hard*, even in its single objective version. Thus, solving efficiently the *Multi Objective Single Source Capacitated Facility Location Problem* is still an open question.

6.1. Further research

We think that the *paving* step in reasonably fast to be kept. The *label setting* may be improved by introducing an even tighter lower bound which is not too long to compute, but may still fail due to exponentially growing number of labels.

An other direction to follow might be using a *ranking* method instead of a *label setting*, if possible. To do that, we would compute all the supported solutions during the paving and find the *k best solutions* in the non-dominated triangles given by the latter, in order to get the non-supported efficient solutions. This method would be inspired by the *two phase method for the multi-objective assignment problem* [6].

It remains that the problem is *NP-hard* and we would *parallelize* the algorithm to make it suitable in concrete situations. The *paving* step may be easily parallelizable as the *generation* step in which each box can be independently processed. A common pool of non-dominated solutions would be shared between the processes.

6.2. Acknowledgments

I would like to thank the research team of the University of Vienna, who is very nice. A particular and friendly thank to Thibault Barthélemy who gave me hints to improve my research. I express my gratitude to Dr Fabien Tricoire, Pr Xavier Gandibleux and Dr Anthony Przybylski to have made the internship be realized under good conditions. Thanks to my professors of the MSc Optimisation in Operations Research of the University of Nantes and the Université libre de Bruxelles who permitted to get the skills to do the research.

References

[1] , March 2012. Biobjective Uncapacitated Facility Location Solver (GPL software). Http://oro.univ-nantes.fr/software.

[2] Aneja, Y. P., Nair, K. P. K., 1979. Bicriteria transportation problem. Management Science 25 (1), pp. 73–78.

[3] Bourougaa, S., Derrien, A., Grimault, A., Gandibleux, X., Przybylski, A., April 11-13 2012. Sur le calcul des solutions efficaces du problème bi-objectif de localisation de services sans contrainte de capacité. In: 13e congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision. ROADEF'2012, Angers, France.

[4] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2000. A fast elitist multi-objective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6, 182–197.

[5] Desrochers, M., Desrosiers, J., Solomon, M., 1992. A new optimization algorithm for the vehicle routing problem with time windows. Operations Research 40 (2), 342–354.

[6] Ehrgott, M., 2005. Multicriteria Optimization (2. ed.). Springer.

[7] Farahani, R. Z., SteadieSeifi, M., Asgari, N., 2010. Multiple criteria facility location problems: A survey. Applied Mathematical Modelling 34 (7), 1689–1709.

[8] Feillet, D., Dejax, P., Gendreau, M., Gueguen, C., 2004. An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems. Networks 44 (3), 216–229.

[9] Fernández, E., Puerto, J., 2003. Multiobjective solution of the uncapacitated plant location problem. European Journal of Operational Research 145 (3), 509–529.

[10] Ford, L., Fulkerson, D., 1956. A Primal Dual Algorithm for the Capacitated Hitchcock Problem. P (Rand Corporation). Rand Corporation.

[11] Gandibleux, X., Przybylski, A., Bourougaa, S., Derrien, A., Grimault, A., June 11-13 2012. Computing the efficient frontier for the 0/1 biobjective uncapacitated facility location problem. In: 10th International Conference on Multiple Objective Programming and Goal Programming. MOPGP'2012, Niagara Falls, Canada.

[12] Garey, M. R., Johnson, D. S., July 1978. Strong np-completeness results: Motivation, examples, and implications. J. ACM 25, 499–508.

[13] Garey, M. R., Johnson, D. S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA.

[14] Karp, R., 1972. Reducibility among combinatorial problems. In: Miller, R., Thatcher, J. (Eds.), Complexity of Computer Computations. Plenum Press, pp. 85–103.

[15] Martins, E., 1984. On a multicriteria shortest path problem. European Journal of Operational Research 16, 236–245.

[16] Mostaghim, S., Teich, J., 2003. Quad-trees: A data structure for storing pareto-sets in multi-objective evolutionary algorithms with elitism. In: In Evolutionary Computation Based Multi-Criteria Optimization: Theoretical Advances And Applications.

[17] Mostaghim, S., Teich, J., Tyagi, A., May 12-17 2002. Comparison of data structures for storing pareto-sets in moeas. In: IEEE Proceedings, World Congress on Computational Intelligence (CEC 2002). Honolulu, USA, pp. 843–849.

[18] Nagelhout, R., Thompson, G., 1979. A Single Source Transportation Algorithm. Carnegie-Mellon University, Design Research Center.

[19] Ross, G., Soland, R., 1975. A branch and bound algorithm for the generalized assignment problem. Mathematical Programming 8 (1), 91–103.

[20] Teghem, J., 2012. Recherche Opérationnelle Tome 1 : Méthodes d'optimisation. Ellipses.

7. Appendix

7.1. Branch and bound times

Instance	Box	k	Time(BnB)	Time(SCIP)
F1-2	1101101110	0	827.651	1550.76
F1-2	1101101110	1	495.215	842.344
F1-2	1111001110	0	198.494	1067.73
F1-2	1111001110	1	483.605	876.032
F1-2	1101011110	0	737.83	1157.62
F1-2	1101011110	1	208.274	614.771
F1-2	0101111110	0	400.247	802.074
F1-2	0101111110	1	201.798	263.554
F1-2	0111101110	0	278.437	291.362
F1-2	0111101110	1	370.571	586.158
F1-2	0111011110	0	117.127	281.131
F1-2	0111011110	1	304.131	438.493
F1-2	1100111110	0	11.8373	38.945
F1-2	1100111110	1	18.9666	58.6369
F1-2	1110101110	0	12.0836	42.2645
F1-2	1110101110	1	114.372	152.587
F1-2	1101110110	0	130.886	221.056
F1-2	1101110110	1	88.0681	167.483
F1-2	1111100110	0	196.282	519.283
F1-2	1111100110	1	347.565	250.208
F1-2	1101111100	0	32.2031	68.9573
F1-2	1101111100	1	223.065	423.997
F1-2	1101101111	0	45.4079	106.225
F1-2	1101101111	1	191.369	86.9984
F1-2	1100111011	0	229.38	377.279
F1-2	1100111011	1	27.165	160.167
F1-2	1111101100	0	54.5262	97.9359
F1-2	1111101100	1	216.495	184.567
F1-2	1101111010	0	72.0695	156.194
F1-2	1101111010	1	37.9114	110.784
F1-2	1001111110	0	15.6152	59.1295
F1-2	1001111110	1	22.7073	198.543
F1-2	0111101111	0	66.1123	120.196
F1-2	0111101111	1	72.4167	118.93
F2-3	1001011111	0	312.854	681.48
F2-3	1001011111	1	373.17	956.257
F2-3	1011011001	0	54.0359	322.637
F2-3	1011011001	1	76.1333	579.63
F2-3	1110011101	0	313.181	2319.22
F2-3	1110011101	1	229.462	922.793
F2-3	0101111111	0	772.135	908.325
F2-3	0101111111	1	204.093	1425.28
F2-3	1010111101	0	73.1691	249.815
F2-3	1010111101	1	244.347	663.843
F2-3	1011001111	0	26.3834	202.145
F2-3	1011001111	1	21.5903	115.97
F2-3	1110011011	0	255.404	446.678
F2-3	1110011011	1	30.7282	339.992
F2-3	1101111101	0	306.527	519.599
F2-3	1101111101	1	544.687	489.594
F2-3	1010111011	0	45.2595	73.9428
F2-3	1010111011	1	5.6127	39.9912
F2-3	1111101101	0	20.5214	137.146
F2-3	1111101101	1	241.468	499.076
F2-3	1101111011	0	27.0583	97.277
F2-3	1101111011	1	61.161	191.116
F2-3	1111101011	0	25.2199	78.0853
F2-3	1111101011	1	30.0138	110.195
F2-3	0010111111	0	646.534	979.315
F2-3	0010111111	1	284.329	680.805
F2-3	1101011111	0	141.929	111.167
F2-3	1101011111	1	45.176	101.49
F2-3	1111011001	0	7.855	91.1436
F2-3	1111011001	1	16.2338	34.8707
F2-3	1111001111	0	44.0789	117.98
F2-3	1111001111	1	19.1396	206.19
F3-4	0111101101	0	1480.98	7299.97
F3-4	0111101101	1	615.74	2420.96
F3-4	0111111001	0	229.859	1110.78
F3-4	0111111001	1	223.53	1213.17
F3-4	1001111101	0	608.067	3069.2
F3-4	1001111101	1	418.215	823.918
F3-4	0111101011	0	284.081	1145.12
F3-4	0111101011	1	326.771	2022.27
F3-4	1011101101	0	339.981	655.279
F3-4	1011101101	1	130.847	201.621
F3-4	1011111001	0	616.108	1286.32
F3-4	1011111001	1	121.129	442.206
F3-4	1011101011	0	23.5481	219.659
F3-4	1011101011	1	60.5606	150.702
F3-4	1111110001	0	1020.01	2151.07
F3-4	1111110001	1	110.879	775.464
F3-4	1010111110	0	570.307	3281.22
F3-4	1010111110	1	145.688	1231.74
F3-4	0010110111	0	332.699	1214.25
F3-4	0010110111	1	157.349	1112.44
F3-4	1111100011	0	173.825	409.757
F3-4	1111100011	1	88.6196	889.1
F3-4	1010011111	0	650.897	1412.65
F3-4	1010011111	1	565.104	854.042
F3-4	1001111011	0	497.144	1218.5
F3-4	1001111011	1	466.19	1944.05
F3-4	1111100101	0	255.29	1455.83
F3-4	1111100101	1	209.597	971.372
F3-4	1011111100	0	440.761	763.798
F3-4	1011111100	1	73.6237	145.674

Figure 26: Times in milliseconds of our branch and bound algorithm and SCIP solver

Instance	Box	k	Time(BnB)	Time(SCIP)	Instance	Box	k	Time(BnB)	Time(SCIP)
F3-4	0011110101	0	40.67	62.999	F4-5	1011011101	0	37.2595	121.203
F3-4	0011110101	1	9.507	51.7963	F4-5	1011011101	1	1.3736	25.8088
F3-4	0011100111	0	8.8203	68.0207	F4-5	1101111001	0	206.159	752.266
F3-4	0011100111	1	9.4989	29.0101	F4-5	1101111001	1	55.3524	364.024
F3-4	0111111100	0	930.469	1579.2	F4-5	1001100111	0	21.4518	200.159
F3-4	0111111100	1	84.5502	526.784	F4-5	1001100111	1	0.6997	37.8111
F3-4	0111101110	0	266.747	567.111	F4-5	0011011111	0	41.1626	574.118
F3-4	0111101110	1	41.4343	256.428	F4-5	0011011111	1	359.06	1851.45
F3-4	0111011101	0	2089.8	1948.35	F4-5	1000111011	0	10.9044	40.6555
F3-4	0111011101	1	276.67	1333.46	F4-5	1000111011	1	4.2758	54.7396
F3-4	1011011101	0	701.089	667.356	F4-5	1010111001	0	86.1267	238.26
F3-4	1011011101	1	117.035	210.774	F4-5	1010111001	1	0.3209	22.8254
F3-4	0001110111	0	65.9624	225.273	F4-5	0010111011	0	40.0976	223.074
F3-4	0001110111	1	38.9942	102.493	F4-5	0010111011	1	31.4579	265.128
F3-4	0011110011	0	30.2832	292.866	F4-5	1100001111	0	29.7371	89.6646
F3-4	0011110011	1	3.6318	22.2301	F4-5	1100001111	1	9.0534	63.9979
F3-4	1110111101	0	15.5885	61.0407	F4-5	1001101011	0	0.8694	24.286
F3-4	1110111101	1	63.0712	111.165	F4-5	1001101011	1	3.2322	89.9094
F3-4	1001101111	0	321.931	1574.5	F4-5	1010111010	0	76.1137	683.328
F3-4	1001101111	1	585.857	1700.71	F4-5	1010111010	1	23.7082	364.813
F3-4	0111111010	0	643.182	1518	F4-5	1011010011	0	22.4176	62.094
F3-4	0111111010	1	24.7471	158.366	F4-5	1011010011	1	7.7291	27.4751
F3-4	1110111011	0	63.9719	205.072	F4-5	1011101001	0	2.83	56.5709
F3-4	1110111011	1	77.0474	233.811	F4-5	1011101001	1	0.368	23.14
F3-4	0011110110	0	6.4858	33.982	F4-5	1000101111	0	2.5076	21.6622
F3-4	0011110110	1	2.205	20.6125	F4-5	1000101111	1	5.12	33.5343
F3-4	0011111101	0	20.5025	42.6801	F4-5	1100111101	0	233.742	321.597
F3-4	0011111101	1	4.3234	49.0111	F4-5	1100111101	1	8.6718	48.653
F4-5	1001110011	0	21.9673	853.003	F5-6	1101110110	0	3733.12	3566.62
F4-5	1001110011	1	60.1301	479.706	F5-6	1101110110	1	898.298	5567.55
F4-5	1011110001	0	48.5135	545.368	F5-6	1100011111	0	313.815	1171.79
F4-5	1011110001	1	54.8205	292.785	F5-6	1100011111	1	555.47	1610.68
F4-5	1001111101	0	123.972	690.312	F5-6	1101111010	0	171.62	410.192
F4-5	1001111101	1	251.823	1083.96	F5-6	1101111010	1	489.257	1140.79
F4-5	1010000111	0	45.8349	451.795	F5-6	1111001100	0	20.1856	76.2312
F4-5	1010000111	1	81.3157	434.673	F5-6	1111001100	1	229.394	470.195
F4-5	1101001011	0	79.7759	573.567	F5-6	1001111110	0	441.454	1919.04
F4-5	1101001011	1	109.148	627.108	F5-6	1001111110	1	398.1	1503.86
F4-5	1111001001	0	176.489	714.108	F5-6	1110001110	0	36.9987	146.901
F4-5	1111001001	1	71.2888	737.835	F5-6	1110001110	1	222.681	369.71
F4-5	1000110111	0	21.1502	188.046	F5-6	1101011110	0	55.1873	191.439
F4-5	1000110111	1	0.9316	30.1068	F5-6	1101011110	1	116.86	121.465
F4-5	1010110101	0	25.021	174.34	F5-6	1110010111	0	135.753	901.926
F4-5	1010110101	1	17.9172	276.118	F5-6	1110010111	1	376.894	431.731
F4-5	1010001011	0	10.7779	49.4133	F5-6	1101011101	0	1119.77	4512.48
F4-5	1010001011	1	4.5512	52.01	F5-6	1101011101	1	252.399	516.498
F4-5	1001011111	0	25.027	96.8295	F5-6	1111110010	0	257.265	1685.96
F4-5	1001011111	1	17.7133	113.753	F5-6	1111110010	1	428.138	1174.85

Figure 27: Times in milliseconds of our branch and bound algorithm and SCIP solver

Instance	Box	k	Time(BnB)	Time(SCIP)
F5-6	1110101100	0	23.6579	162.548
F5-6	1110101100	1	198.803	338.781
F5-6	1101001111	0	133.161	264.611
F5-6	1101001111	1	155.144	157.119
F5-6	1111010110	0	27.0729	107.512
F5-6	1111010110	1	171.219	230.712
F5-6	1110110110	0	33.4094	338.96
F5-6	1110110110	1	264.549	183.836
F5-6	1100111110	0	40.1625	146.981
F5-6	1100111110	1	413.701	242.08
F5-6	1111011010	0	4.0222	37.7128
F5-6	1111011010	1	134.663	120.692
F5-6	1110011011	0	42.4231	183.888
F5-6	1110011011	1	1638	2294.43
F5-6	1111110100	0	64.0541	429.513
F5-6	1111110100	1	165.452	236.395
F5-6	1101101011	0	1119.76	4678.64
F5-6	1101101011	1	125.72	936.834
F5-6	1101111100	0	24.4145	107.786
F5-6	1101111100	1	36.6507	59.1671
F5-6	1100111101	0	687.264	2043.13
F5-6	1100111101	1	194.153	1154
F5-6	1111000111	0	72.3725	190.105
F5-6	1111000111	1	143.023	245.562
F5-6	1010011111	0	82.6332	1013.49
F5-6	1010011111	1	271.996	1532.28
F5-6	1011111010	0	70.8915	732.411
F5-6	1011111010	1	341.194	799.3
F5-6	1100101111	0	71.3696	346.067
F5-6	1100101111	1	242.12	274.116
F5-6	1110111010	0	28.0528	224.726
F5-6	1110111010	1	112.092	80.9518
F5-6	1011110110	0	474.624	1628.49
F5-6	1011110110	1	1430.7	9651.13
F5-6	1011011110	0	23.2393	264.856
F5-6	1011011110	1	71.6383	120.49
F5-6	1110111001	0	331.583	1851.7
F5-6	1110111001	1	322.057	1943.77
F5-6	1010111110	0	142.565	338.037
F5-6	1010111110	1	403.511	540.008
F5-6	1110101011	0	58.8941	313.354
F5-6	1110101011	1	64.5436	49.5619
F5-6	1011101011	0	335.599	2154.21
F5-6	1011101011	1	101.114	731.965
F5-6	1101110111	0	137.594	403.264
F5-6	1101110111	1	72.611	60.1827
F5-6	1101111011	0	74.5019	106.515
F5-6	1101111011	1	106.468	108.357
F5-6	1111011100	0	1.4351	30.3608
F5-6	1111011100	1	35.4993	102.833
F5-6	1010111101	0	473.489	4992.65
F5-6	1010111101	1	70.3685	1543.37
F5-6	1001111111	0	16.2212	61.0499
F5-6	1001111111	1	93.9752	71.6479
F5-6	1110011110	0	14.1888	35.3267
F5-6	1110011110	1	49.6034	58.2228
F5-6	1100111111	0	8.2176	38.5708
F5-6	1100111111	1	15.6712	49.4291
F5-6	1111011001	0	49.6461	157.911
F5-6	1111011001	1	509.193	1509.58
F5-6	1010101111	0	265.623	199.044
F5-6	1010101111	1	146.092	187.094
F5-6	1111100110	0	2.0748	38.6249
F5-6	1111100110	1	197.588	84.9502
F5-6	1101101110	0	23.7791	66.4797
F5-6	1101101110	1	15.8959	48.7659
F5-6	0111111011	0	3618.5	8623.42
F5-6	0111111011	1	1223.62	4838.82
F5-6	1110100111	0	65.8393	339.151
F5-6	1110100111	1	503.892	1355.81
F5-6	1111001011	0	84.9318	313.029
F5-6	1111001011	1	97.1082	78.092
F5-6	1111111000	0	23.6073	51.0823
F5-6	1111111000	1	133.876	229.599
F5-6	1101011111	0	25.6706	63.1066
F5-6	1101011111	1	32.0541	66.6499
F5-6	1101101101	0	126.931	613.646
F5-6	1101101101	1	72.0836	148.863
F5-6	1111101010	0	13.5015	49.7086
F5-6	1111101010	1	90.1027	62.6063
F5-6	0110111111	0	131.753	310.19
F5-6	0110111111	1	259.604	194.559
F5-6	1110111100	0	15.6348	45.5799
F5-6	1110111100	1	29.3389	74.563
F5-6	1110101110	0	11.6174	32.513
F5-6	1110101110	1	10.9836	51.5002

Figure 28: Times in milliseconds of our branch and bound algorithm and SCIP solver

7.2. Branch and bound nodes

Instance	Box	k	Nodes(BnB)	Nodes(SCIP)	Instance	Box	k	Nodes(BnB)	Nodes(SCIP)
F1-2	1101101110	0	4957	8559	F2-3	1101111101	0	1683	1946
F1-2	1101101110	1	3919	3356	F2-3	1101111101	1	3421	1832
F1-2	1111001110	0	1643	5455	F2-3	1010111011	0	313	173
F1-2	1111001110	1	4437	4184	F2-3	1010111011	1	35	117
F1-2	1101011110	0	5645	5533	F2-3	1111101101	0	103	355
F1-2	1101011110	1	1709	2275	F2-3	1111101101	1	1581	1451
F1-2	0101111110	0	2967	5383	F2-3	1101111011	0	143	318
F1-2	0101111110	1	1187	436	F2-3	1101111011	1	377	343
F1-2	0111101110	0	2393	512	F2-3	1111101011	0	107	246
F1-2	0111101110	1	3811	1149	F2-3	1111101011	1	239	235
F1-2	0111011110	0	1007	599	F2-3	0010111111	0	3405	7019
F1-2	0111011110	1	3297	681	F2-3	0010111111	1	2143	3386
F1-2	1100111110	0	83	133	F2-3	1101011111	0	851	242
F1-2	1100111110	1	171	166	F2-3	1101011111	1	259	181
F1-2	1110101110	0	79	156	F2-3	1111011001	0	55	155
F1-2	1110101110	1	1145	269	F2-3	1111011001	1	137	86
F1-2	1101110110	0	825	379	F2-3	1111001111	0	339	262
F1-2	1101110110	1	663	334	F2-3	1111001111	1	201	330
F1-2	1111100110	0	1719	2746	F3-4	0111101101	0	10605	114360
F1-2	1111100110	1	3803	437	F3-4	0111101101	1	4493	32717
F1-2	1101111100	0	255	264	F3-4	0111111001	0	1179	7164
F1-2	1101111100	1	2135	961	F3-4	0111111001	1	1811	11565
F1-2	1101101111	0	423	217	F3-4	1001111101	0	3761	46936
F1-2	1101101111	1	1443	188	F3-4	1001111101	1	3945	4440
F1-2	1100111011	0	1751	711	F3-4	0111101011	0	2247	11509
F1-2	1100111011	1	207	352	F3-4	0111101011	1	2877	30406
F1-2	1111101100	0	391	280	F3-4	1011101101	0	2851	3241
F1-2	1111101100	1	1621	357	F3-4	1011101101	1	1113	292
F1-2	1101111010	0	393	351	F3-4	1011111001	0	4593	10827
F1-2	1101111010	1	333	252	F3-4	1011111001	1	1153	1275
F1-2	1001111110	0	81	151	F3-4	1011101011	0	213	333
F1-2	1001111110	1	281	360	F3-4	1011101011	1	571	305
F1-2	0111101111	0	497	143	F3-4	1111110001	0	5943	20767
F1-2	0111101111	1	641	209	F3-4	1111110001	1	1083	3884
F2-3	1001011111	0	1811	1629	F3-4	1010111110	0	3193	50544
F2-3	1001011111	1	2157	5266	F3-4	1010111110	1	933	9130
F2-3	1011011001	0	455	828	F3-4	0010110111	0	3525	11270
F2-3	1011011001	1	745	2958	F3-4	0010110111	1	2201	12224
F2-3	1110011101	0	1203	32822	F3-4	1111100011	0	1513	665
F2-3	1110011101	1	1867	5784	F3-4	1111100011	1	647	6549
F2-3	0101111111	0	4497	4510	F3-4	1010011111	0	3357	13293
F2-3	0101111111	1	1439	11695	F3-4	1010011111	1	4605	2549
F2-3	1010111101	0	447	478	F3-4	1001111011	0	2529	7524
F2-3	1010111101	1	1385	2998	F3-4	1001111011	1	5193	22683
F2-3	1011001111	0	137	471	F3-4	1111100101	0	1917	17761
F2-3	1011001111	1	131	289	F3-4	1111100101	1	1405	6624
F2-3	1110011011	0	1281	1069	F3-4	1011111100	0	3371	3726
F2-3	1110011011	1	365	398	F3-4	1011111100	1	713	224

Figure 29: Number of nodes of our branch and bound algorithm and SCIP solver

Instance	Box	k	Nodes(BnB)	Nodes(SCIP)	Instance	Box	k	Nodes(BnB)	Nodes(SCIP)
F3-4	0011110101	0	297	131	F4-5	1011011101	0	321	226
F3-4	0011110101	1	73	139	F4-5	1011011101	1	11	66
F3-4	0011100111	0	61	118	F4-5	1101111001	0	2063	3548
F3-4	0011100111	1	93	89	F4-5	1101111001	1	509	834
F3-4	0111111100	0	7145	14072	F4-5	1001100111	0	177	346
F3-4	0111111100	1	531	1443	F4-5	1001100111	1	5	91
F3-4	0111101110	0	2623	2029	F4-5	0011011111	0	221	1643
F3-4	0111101110	1	295	455	F4-5	0011011111	1	2525	23266
F3-4	0111011101	0	16375	22042	F4-5	1000111011	0	105	84
F3-4	0111011101	1	2503	9471	F4-5	1000111011	1	39	127
F3-4	1011011101	0	5859	3442	F4-5	1010111001	0	1019	455
F3-4	1011011101	1	1277	314	F4-5	1010111001	1	3	37
F3-4	0001110111	0	815	407	F4-5	0010111011	0	377	499
F3-4	0001110111	1	511	243	F4-5	0010111011	1	167	443
F3-4	0011110011	0	401	641	F4-5	1100001111	0	213	212
F3-4	0011110011	1	39	130	F4-5	1100001111	1	39	198
F3-4	1110111101	0	107	196	F4-5	1001101011	0	9	89
F3-4	1110111101	1	537	233	F4-5	1001101011	1	31	122
F3-4	1001101111	0	1933	11148	F4-5	1010111010	0	813	5344
F3-4	1001101111	1	6383	17337	F4-5	1010111010	1	177	531
F3-4	0111111010	0	3867	9834	F4-5	1011010011	0	133	192
F3-4	0111111010	1	179	267	F4-5	1011010011	1	43	74
F3-4	1110111011	0	351	372	F4-5	1011101001	0	31	118
F3-4	1110111011	1	655	386	F4-5	1011101001	1	3	31
F3-4	0011110110	0	49	143	F4-5	1000101111	0	23	29
F3-4	0011110110	1	29	32	F4-5	1000101111	1	65	53
F3-4	0011111101	0	283	71	F4-5	1100111101	0	2469	626
F3-4	0011111101	1	47	101	F4-5	1100111101	1	47	125
F4-5	1001110011	0	175	5712	F5-6	1101110110	0	20349	53539
F4-5	1001110011	1	313	1449	F5-6	1101110110	1	4499	91931
F4-5	1011110001	0	337	1589	F5-6	1100011111	0	1841	10644
F4-5	1011110001	1	399	577	F5-6	1100011111	1	4047	16645
F4-5	1001111101	0	965	2196	F5-6	1101111010	0	1579	1059
F4-5	1001111101	1	1801	8405	F5-6	1101111010	1	3599	8462
F4-5	1010000111	0	381	2393	F5-6	1111001100	0	173	166
F4-5	1010000111	1	665	1616	F5-6	1111001100	1	2973	1520
F4-5	1101001011	0	595	2767	F5-6	1001111110	0	3323	23015
F4-5	1101001011	1	539	3071	F5-6	1001111110	1	2283	9637
F4-5	1111001001	0	1235	3160	F5-6	1110001110	0	293	264
F4-5	1111001001	1	549	5329	F5-6	1110001110	1	2287	834
F4-5	1000110111	0	183	336	F5-6	1101011110	0	435	358
F4-5	1000110111	1	7	108	F5-6	1101011110	1	1407	278
F4-5	1010110101	0	193	380	F5-6	1110010111	0	587	5096
F4-5	1010110101	1	139	598	F5-6	1110010111	1	2189	892
F4-5	1010001011	0	53	132	F5-6	1101011101	0	6109	72627
F4-5	1010001011	1	43	118	F5-6	1101011101	1	1387	1434
F4-5	1001011111	0	139	238	F5-6	1111110010	0	1469	23440
F4-5	1001011111	1	189	235	F5-6	1111110010	1	2015	7923

Figure 30: Number of nodes of our branch and bound algorithm and SCIP solver

Instance	Box	k	Nodes(BnB)	Nodes(SCIP)
F5-6	1110101100	0	165	299
F5-6	1110101100	1	1731	667
F5-6	1101001111	0	1155	463
F5-6	1101001111	1	1421	260
F5-6	1111010110	0	179	292
F5-6	1111010110	1	1561	423
F5-6	1110110110	0	409	624
F5-6	1110110110	1	2143	322
F5-6	1100111110	0	339	192
F5-6	1100111110	1	3909	407
F5-6	1111011010	0	45	100
F5-6	1111011010	1	1553	217
F5-6	1110011011	0	241	316
F5-6	1110011011	1	10701	24978
F5-6	1111110100	0	719	1385
F5-6	1111110100	1	1383	393
F5-6	1101101011	0	5737	66153
F5-6	1101101011	1	635	3309
F5-6	1101111100	0	219	204
F5-6	1101111100	1	347	140
F5-6	1100111101	0	3797	24238
F5-6	1100111101	1	1049	5368
F5-6	1111000111	0	495	370
F5-6	1111000111	1	1477	490
F5-6	1010011111	0	439	5957
F5-6	1010011111	1	2069	17442
F5-6	1011111010	0	581	3702
F5-6	1011111010	1	2393	2163
F5-6	1100101111	0	537	404
F5-6	1100101111	1	2265	479
F5-6	1110111010	0	321	452
F5-6	1110111010	1	871	203
F5-6	1011110110	0	2077	9791
F5-6	1011110110	1	7547	138222
F5-6	1011011110	0	303	659
F5-6	1011011110	1	663	191
F5-6	1110111001	0	1903	24417
F5-6	1110111001	1	2035	21350
F5-6	1010111110	0	1511	566
F5-6	1010111110	1	3473	2178
F5-6	1110101011	0	525	740
F5-6	1110101011	1	405	142
F5-6	1011101011	0	1883	29803
F5-6	1011101011	1	545	1109
F5-6	1101110111	0	1309	1062
F5-6	1101110111	1	573	191
F5-6	1101111011	0	755	168
F5-6	1101111011	1	1133	276

Instance	Box	k	Nodes(BnB)	Nodes(SCIP)
F5-6	1111011100	0	15	84
F5-6	1111011100	1	373	180
F5-6	1010111101	0	2043	71649
F5-6	1010111101	1	365	15961
F5-6	1001111111	0	139	187
F5-6	1001111111	1	727	153
F5-6	1110011110	0	77	91
F5-6	1110011110	1	711	121
F5-6	1100111111	0	67	107
F5-6	1100111111	1	161	151
F5-6	1111011001	0	227	363
F5-6	1111011001	1	3769	18534
F5-6	1010101111	0	1885	283
F5-6	1010101111	1	1501	372
F5-6	1111100110	0	27	104
F5-6	1111100110	1	1375	152
F5-6	1101101110	0	283	100
F5-6	1101101110	1	147	112
F5-6	0111111011	0	14649	106084
F5-6	0111111011	1	5161	71768
F5-6	1110100111	0	571	566
F5-6	1110100111	1	4571	8832
F5-6	1111001011	0	695	541
F5-6	1111001011	1	1081	181
F5-6	1111111000	0	227	121
F5-6	1111111000	1	1017	389
F5-6	1101011111	0	221	114
F5-6	1101011111	1	343	179
F5-6	1101101101	0	1163	1606
F5-6	1101101101	1	807	268
F5-6	1111101010	0	115	118
F5-6	1111101010	1	859	144
F5-6	0110111111	0	899	653
F5-6	0110111111	1	1677	339
F5-6	1110111100	0	95	100
F5-6	1110111100	1	229	109
F5-6	1110101110	0	113	70
F5-6	1110101110	1	93	100

Figure 31: Number of nodes of our branch and bound algorithm and SCIP solver

7.3. *Relation between max number of supported points per box and paving and generation times*

Instance	Paving time (s)				Generation time (s)			
maxpoints:	0	1	3	Infinity	0	1	3	Infinity
F1-2	5.9	7.1	7	11	5443	2768	1613	1315
F2-3	4.9	6.6	8.5	13.9	2961	134	69.6	53.1
F3-4	15.6	16.6	22.3	39.1	>250000	181787	55332	4012
F4-5	2.4	2.4	3.3	6	15876	3685	2508	1219
F5-6	26.3	24.6	32.4	61.9	78428	1286	1152	957

Figure 32: Relation between max number of supported points per box and paving and generation times

7.4. *Efficient solutions graphs*

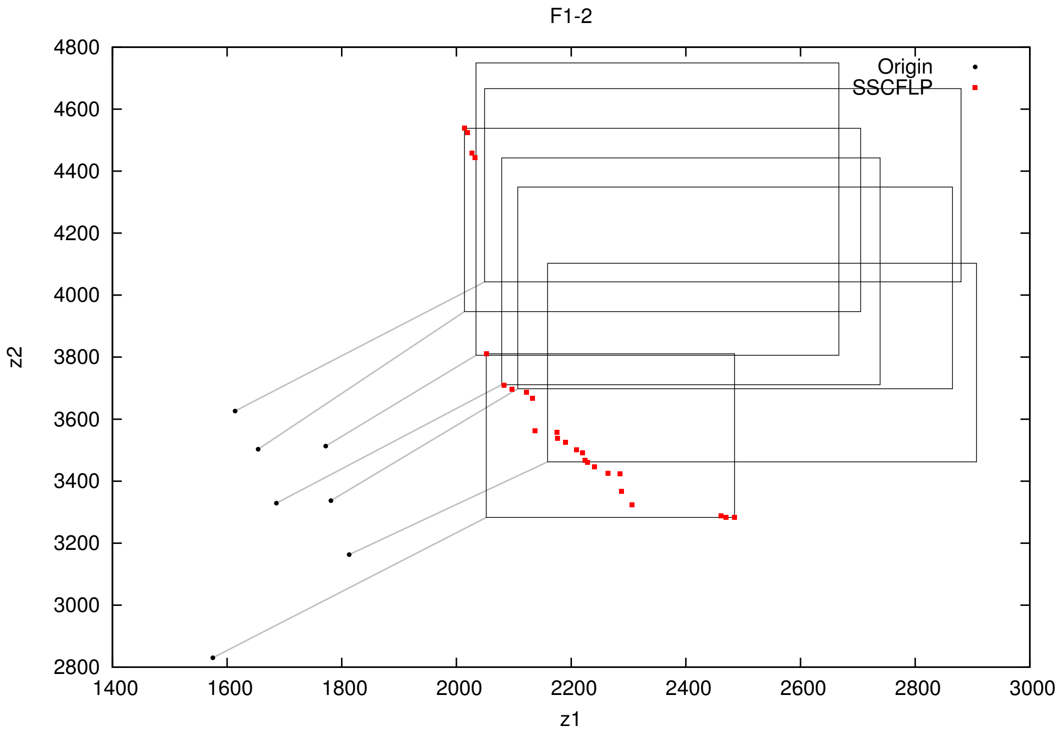


Figure 33: Efficient solutions for F1-2

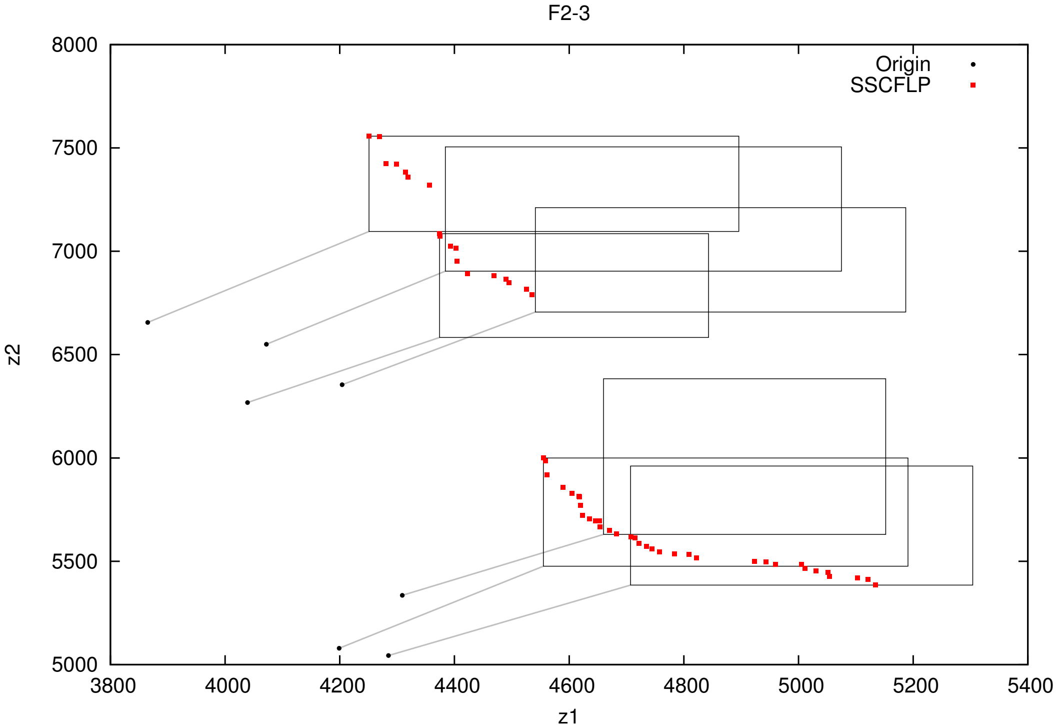


Figure 34: Efficient solutions for F2-3

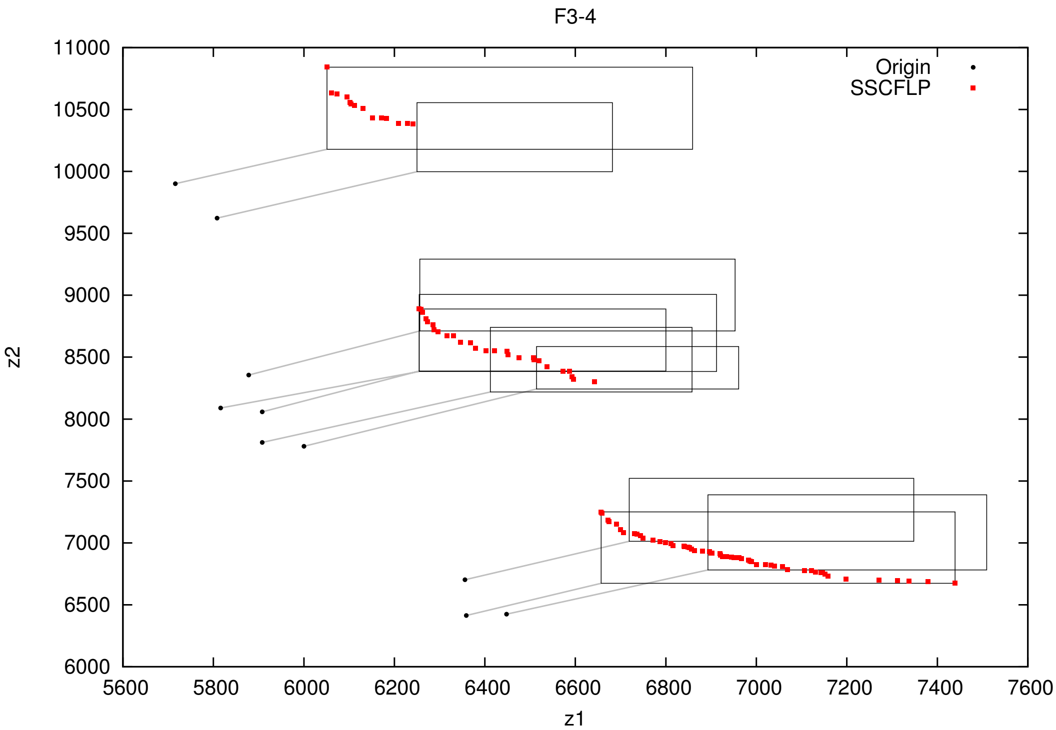


Figure 35: Efficient solutions for F3-4

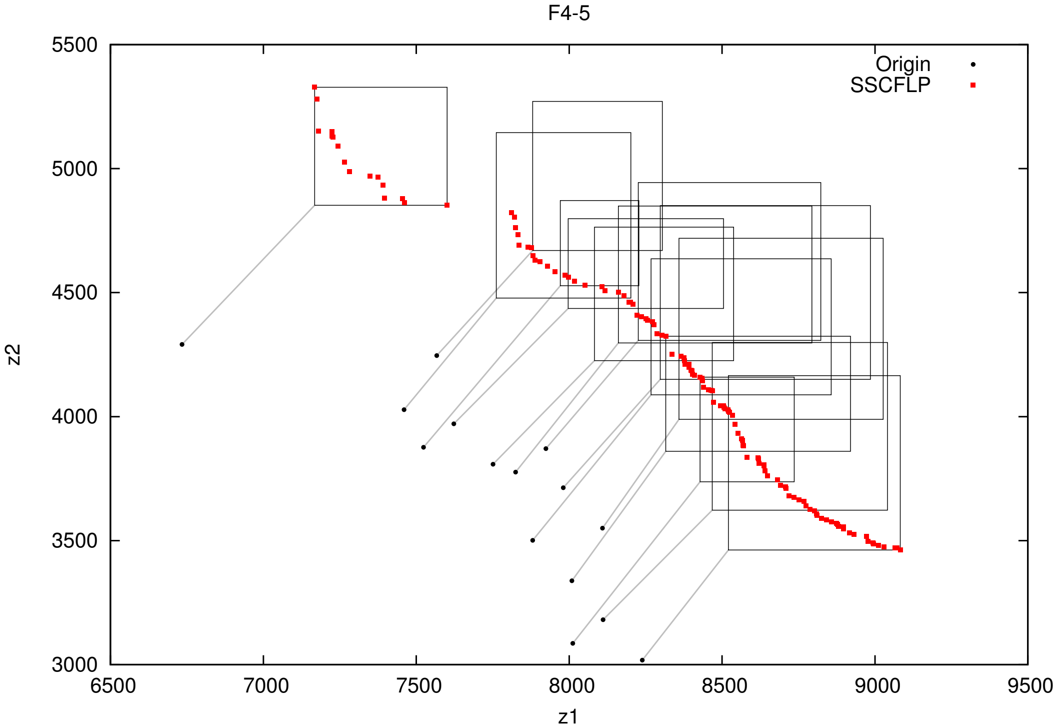


Figure 36: Efficient solutions for F4-5

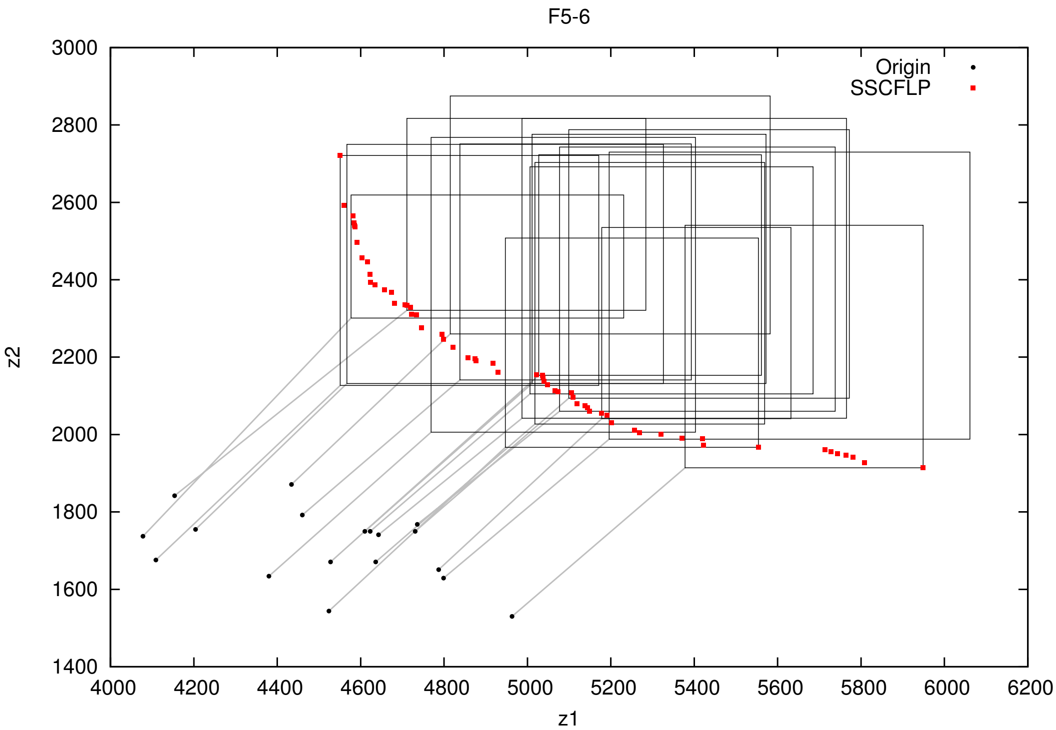


Figure 37: Efficient solutions for F5-6

Contents