Multi-objective branch-and-bound. Application to the bi-objective spanning tree problem

Francis Sourd, Olivier Spanjaard
Laboratoire d'Informatique de Paris 6 (LIP6-UPMC), 4 Place Jussieu, F-75252 Paris Cedex 05, France,
{FirstName.Name@lip6.fr}

This paper focuses on a multi-objective derivation of branch-and-bound procedures. Such a procedure aims to provide the set of Pareto optimal solutions of a multi-objective combinatorial optimization problem. Unlike previous works on this issue, the bounding is performed here via a set of points rather than a single ideal point. The main idea is that a node in the search tree can be discarded if one can define a separating hypersurface in the objective space between the set of feasible solutions in the subtree and the set of points corresponding to potential Pareto optimal solutions. Numerical experiments on the bi-objective spanning tree problem are provided that show the efficiency of the approach.

Key words: multi-objective combinatorial optimization; branch-and-bound; bi-objective spanning tree problem

---

1. Introduction

Branch-and-bound methods (which belong to the class of *implicit enumeration* methods) have proved to perform well on many combinatorial optimization problems, provided good bounding functions are known. They have been designed under the assumption that the quality of a solution is evaluated by a single objective function on a totally ordered scale. However, many real world problems involve multiple, potentially conflicting, objectives. This has led practioners to investigate problems where each solution x in the set $\mathcal{X}$ of feasible solutions is evaluated by p objective functions $f(x) = (f_1(x), \ldots, f_p(x))$ to minimize (without loss of generality). A thorough presentation of the field can be found for instance in a book byEhrgott (2000). Most of the classical exact and approximate methods for single objective discrete optimization have been revisited under multiple objectives, e.g. dynamic programming (Daellenbach and De Kluyver, 1980), greedy algorithm (Serafini, 1986), many heuristic and metaheuristic methods (Ehrgott and Gandibleux, 2004)... Quite surprisingly, as emphasized by Ehrgott and Gandibleux (2000), branch-and-bound methods have not

been studied widely in this multi-objective setting. Actually, the few existing papers on this topic concern mainly multi-objective integer linear programming (Bitran and Rivera, 1982; Kiziltan and Yucaoglu, 1983; Marcotte and Soland, 1986; Mavrotas and Diakoulaki, 1998). This type of method yet combines optimality of the returned solutions (like dynamic programming or greedy algorithm when the problem fulfills the required properties) with adaptability to a wide range of problems (like metaheuristics). An efficient multi-objective branch-and-bound schema is therefore likely to be useful in many contexts.

In multi-objective combinatorial optimization problems, one has to find the Pareto front of $\mathcal{X}$, denoted by $\mathcal{X}^\star$. More precisely, one usually looks for one solution for each point in $\mathcal{Y}^\star$, where $\mathcal{Y}^\star = f(\mathcal{X}^\star)$ denotes the set of Pareto points in the objective space. Therefore, instead of a *single* incumbent, one keeps, during the branch-and-bound algorithm, the *set* **UB** of the best solutions found so far. To be efficient, the branch-and-bound procedure has also to manage a *lower bound* of the sub-problems. In the literature, the proposed branch-and-bound algorithms all use the ideal of a sub-problem (i.e., the point achieving on every component the lowest value among solutions of the sub-problem) for that goal. The methodological contribution of this paper is to show that this approach can be greatly improved by introducing the concept of *separation* between *reachable* and *improving* solutions. The reachable solutions are the solutions that derives from the partial solution of the current node of the branch-and-bound tree while the improving solutions are the solutions that are not dominated by **UB**.

The second contribution of this paper is related to solving the bi-objective spanning tree problem that is presented in detail in Section 3. We introduce new multi-objective derivations of the well-known cut and cycle rules that are useful to identify some dominating edges (that are made mandatory) and to remove some dominated ones from the input graph. We also present dedicated data structures and algorithms to efficiently implement our multi-objective branch-and-bound. Experimental tests show that the resulting algorithm is able to solve instances with up to 400-500 nodes whereas previous algorithms could only solve problems with at most 120 nodes (or even less).

In Section 2, we give a formal framework to design multi-objective branch-and-bound procedures that mainly relies on a generalization of the lower bounding concept. Then, in Section 3, we study more specifically the bi-objective spanning tree problem. Finally, Section 4 is devoted to the numerical experimentations on that problem.

2. Multi-objective branch-and-bound

2.1. Preliminary definitions

A multi-objective branch-and-bound procedure aims to solve multi-objective combinatorial optimization (MOCO) problems. We first recall some preliminary definitions concerning this type of problems. They differ from the standard single-objective ones principally in their cost structure, as solutions are valued by p-vectors instead of scalars. Hence, the comparison of solutions reduces to the comparison of the corresponding vectors. In this framework, the following notions prove useful:

Definition 1 *The* weak dominance *relation on cost-vectors of* $\mathbb{R}^p_+$ *is defined, for all* $u, v \in \mathbb{R}^p_+$ *by:*

$$u \preccurlyeq v \iff [\forall i \in \{1, \ldots, p\}, u_i \leq v_i)]$$

The dominance *relation on cost-vectors of* $\mathbb{R}^p_+$ *is defined as the asymmetric part of* $\preccurlyeq$*:*

$$u \prec v \iff [u \preccurlyeq v \text{ and } not(v \preccurlyeq u)]$$

The strong dominance *relation on cost-vectors of* $\mathbb{R}^p$ *is defined, for all* $u, v \in \mathbb{R}^p$*, by:*

$$u \lessdot v \iff [\forall i \in \{1, \ldots, p\}, u_i < v_i]$$

Definition 2 *Within a set* $\mathcal{Y}$ *any element* u *is said to be* dominated *(resp.* weakly dominated*) when* $v \prec u$ *(resp.* $v \preccurlyeq u$*) for some* v *in* $\mathcal{Y}$*, and* non-dominated *when there is no* v *in* $\mathcal{Y}$ *such that* $v \prec u$*. The set of non-dominated elements in* $\mathcal{Y}$*, denoted by* $\mathcal{Y}^\star$*, is called the* Pareto front *of* $\mathcal{Y}$*. The vector* v^I *such that* $v^I_i = \min_{v \in \mathcal{Y}^\star} v_i$ *for* $i = 1, \ldots, p$ *is called the* ideal point *of* $\mathcal{Y}$*, and the vector* v^N *such that* $v^N_i = \max_{v \in \mathcal{Y}^\star} v_i$ *for* $i = 1, \ldots, p$ *is called the* nadir point *of* $\mathcal{Y}$*.*

By abuse of language, when the cost of a solution is dominated by the cost of another one, we say that the solution is *dominated* by the other one. Similarly, we use the term of *non-dominated* solutions, as well as *Pareto front* of a set of solutions. The non-dominated solutions that minimize a weighted sum of the objectives are called *supported*. A supported solution is called *extreme* if its image is an extreme point of the convex hull of $\mathcal{Y}^\star$.

We are now able to formulate a generic MOCO problem: given a set $\mathcal{X}$ of feasible solutions evaluated by p objective functions $f(x) = (f_1(x), \ldots, f_p(x))$, find a subset $\mathcal{S} \subseteq \mathcal{X}$ such that $f(\mathcal{S})$ equals $\mathcal{Y}^\star = f(\mathcal{X}^\star)$ the set of Pareto points in the objective space.

2.2. The multi-objective branch-and-bound procedure

We now describe our multi-objective branch-and-bound (MOBB) procedure. Let us first remark that single-objective branch-and-bound algorithms are notoriously more efficient when a good solution is known even before starting the search. In the case of the multi-objective branch-and-bound, having a good initial approximation of the Pareto front seems to be even more important. Indeed, while the branching scheme of the single-objective branch-and-bound method can usually be guided by a heuristic in order to quickly find a good feasible solution, a similar heuristic used in the multi-objective context can lead to quickly find some good solutions but is likely to have some difficulties to find some other Pareto optimal points. In practice, it is worth considering any existing heuristic approach to compute this approximate front (constructive methods, search for supported solutions, local search and metaheuristics...). We will denote by **UB** (for Upper Bound) the set of non-dominated costs $(f_1(x), \ldots, f_p(x))$ of the solutions found by the algorithm. **UB** is clearly initialized with the initial approximation of the Pareto front.

We recall that a branch-and-bound method explores an enumeration tree, i.e. a tree such that the set of leaves represents the set of solutions of $\mathcal{X}$. We call *branching part* the way the set of solutions associated with a node of the tree is separated into two subsets, and *bounding part* the way the quality of the best solutions in a subset is optimistically evaluated. The MOBB method is identical to the classical branch-and-bound in the branching part but differs in the bounding part. That is to say the branching scheme must be able to enumerate in a search tree all the feasible —possibly Pareto optimal— solutions of $\mathcal{X}$. When a new feasible solution x is found at some node, its cost $f(x)$ is included in **UB** if it is not weakly dominated by any $u \in \mathbf{UB}$. Conversely, all the costs $u \in \mathbf{UB}$ such that $f(x) \prec u$ are removed from **UB** once $f(x)$ is inserted. The pure enumerative approach, that is an algorithm that enumerates all the elements of $\mathcal{X}$, clearly finds all the Pareto front but is computationally impracticable to solve problems of even moderate size.

The role of the bounding phase is to make the enumeration implicit, which means that, at each node of the search, some computational effort is devoted to try to prove that all the solutions enumerated in the subtree of the current node are unable to improve the current **UB**. Figure 1 illustrates how the bounding procedure is generalized to the multi-objective case (on the right part of the figure, there are two objective functions f_1 and f_2, and c_i denotes the cost w.r.t. f_i). Let us denote by $\mathbf{UB}^{\prec}$ the set of points in the objective space that are not

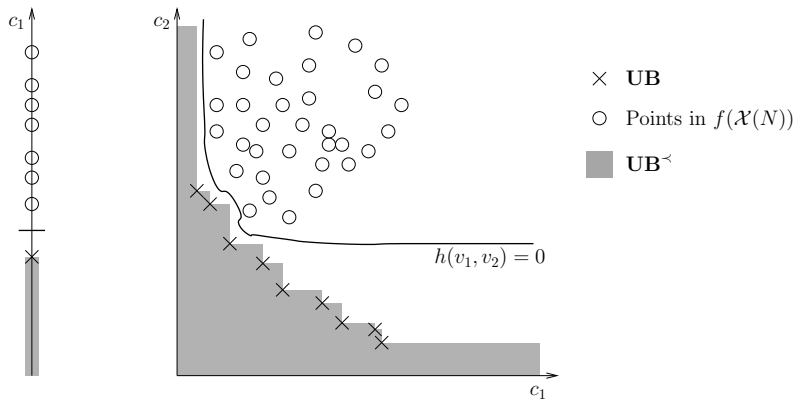

Figure 1: Bounding phase

weakly dominated by any current point of **UB**, that is $\mathbf{UB}^{\prec} = \{v \in \mathbb{R}^p | \forall u \in \mathbf{UB}, u \not\preccurlyeq v\}$. In the single-objective case, there is a single incumbent value **UB** and $\mathbf{UB}^{\prec} = (-\infty, \mathbf{UB})$. Let us also denote by $\mathcal{X}(N)$, the set of feasible solutions which are enumerated in the subtree of the current node N and let $\mathcal{Y}(N)$ be their values in the objective space.

The main point of the bounding procedure is that the current node N can be discarded if we can find a separating hypersurface in the objective space between $\mathbf{UB}^{\prec}$ and $f(\mathcal{X}(N))$, that is a function $h(v_1, \ldots, v_p)$ such that $h(f(x)) \geq 0$ for all $x \in \mathcal{X}(N)$ and $h(v) < 0$ for all $v \in \mathbf{UB}^{\prec}$. This latter property ensures that, when a node N is discarded by this way, for any solution $x \in \mathcal{X}(N)$ there exists $u \in \mathbf{UB}$ such that $u \preccurlyeq f(x)$. In other words, no new non-dominated cost $f(x)$ to be included in **UB** can be found for any solution $x \in \mathcal{X}(N)$. In the single-objective case, the separating function is simply $h(v) = v - \mathbf{LB}(N)$ where $\mathbf{LB}(N)$ is the usual lower bound for $f(\mathcal{X}(N))$. Indeed, if $\mathbf{LB}(N) \geq \mathbf{UB}$, then $h(f(x)) = f(x) - \mathbf{LB}(N) \geq 0$ for all $x \in \mathcal{X}(N)$ and $h(v) = v - \mathbf{LB}(N) < \mathbf{UB} - \mathbf{LB}(N) \leq 0$ for all $v \in \mathbf{UB}^{\prec}$. In the multi-objective case, a linear function h cannot generally separates $f(\mathcal{X}(N))$ from **UB** because **UB** is far from being convex. A general family of good separating functions can be defined as

$$h_\Lambda(v_1, \ldots, v_p) = \min_{\lambda \in \Lambda} \left(\langle \lambda, v \rangle - \mathbf{LB}_\lambda(N) \right)$$

where the $\lambda \in \Lambda$ are weight vectors of the form $(\lambda_1, \ldots, \lambda_p) \geq 0$, $\langle ., . \rangle$ denotes the scalar product and $\mathbf{LB}_\lambda(N) \in \mathbb{R}$ is a lower bound for $\langle \lambda, f \rangle(\mathcal{X}(N))$. The value of $h_\Lambda(v)$ is positive if v is included in $\mathcal{Y}_\Lambda(N) = \bigcap_{\lambda \in \Lambda} \mathcal{Y}_\lambda(N)$, where $\mathcal{Y}_\lambda(N)$ denotes the set $\{v \in \mathbb{R}^p | \langle \lambda, v \rangle \geq \mathbf{LB}_\lambda(N)\}$. $\mathcal{Y}_\Lambda(N)$ plays here the role of a relaxation of $f(\mathcal{X}(N))$, the membership of which can be guessed without enumerating the whole set of solutions in $\mathcal{X}(N)$. Furthermore, we have that $f(\mathcal{X}(N)) \subseteq \mathcal{Y}_\Lambda(N)$, which implies:

Lemma 1 *If h_Λ separates $\mathbf{UB}^\prec$ from $\mathcal{Y}_\Lambda(N)$, then it also separates $\mathbf{UB}^\prec$ from $f(\mathcal{X}(N))$.*

Proof. We only show that $h_\Lambda(f(x)) \geq 0$ for all $x \in \mathcal{X}(N)$ (the other part is immediate). Consider $x \in \mathcal{X}(N)$. We have $\langle \lambda, f(x) \rangle \geq \mathbf{LB}_\lambda(N)$ for all $\lambda \in \Lambda$ since $\mathbf{LB}_\lambda(N)$ is a lower bound for $\langle \lambda, f \rangle(\mathcal{X}(N))$. Hence, $\langle \lambda, f(x) \rangle - \mathbf{LB}_\lambda(N) \geq 0$ for all $\lambda \in \Lambda$, which implies that $\min_{\lambda \in \Lambda} (\langle \lambda, f(x) \rangle - \mathbf{LB}_\lambda(N)) \geq 0$. ∎

Note that the corresponding separating hypersurface is the frontier of $\mathcal{Y}_\Lambda(N)$, which can be defined as $\mathcal{Y}^\star_\Lambda(N) = \{v \in \mathbb{R}^p | h_\Lambda(v) = 0\}$. Clearly, the larger $|\Lambda|$ is, the better the separating function becomes. However, it also becomes more complex and longer to compute. Function h_Λ is convex and piecewise linear and its graph has at most $|\Lambda|$ facets.

Remark 1 *The idea consisting in bounding the Pareto front by a piecewise linear function was also used by Murthy and Sarkar (1998) to optimize a piecewise linear utility function in stochastic shortest path problems, as well as by* **?** *to evaluate the quality of an approximation of an unknown Pareto front.*

Conversely, an implementation of MOBB should also implement a computationally tractable representation of $\mathbf{UB}^\prec$, which may be approximate. In practice, we can consider a finite set $\mathcal{N}$ of vectors in $\mathbb{R}^p$ such that, for any $v \in \mathbf{UB}^\prec$, we have $v \prec\!\!\!\prec w$ for some $w \in \mathcal{N}$. For instance, when there are two objectives and $\{(u_1^i, u_2^i) \,|\, 1 \leq i \leq k\}$ are the points of $\mathbf{UB}$ maintained in lexicographical order, we can set $\mathcal{N} = \{(u_1^{i+1}, u_2^i) \,|\, 0 \leq i \leq k\}$, where $u_2^0 = +\infty$ and $u_1^{k+1} = +\infty$. With this setting, we have an exact covering of $\mathbf{UB}^\prec$ in the sense that $\mathbf{UB}^\prec = \{v \in \mathbb{R}^p \,|\, \exists w \in \mathcal{N},\, v \prec\!\!\!\prec w\}$. Note that, if $u_1^1 = \min_{v \in \mathcal{Y}^\star} v_1$ and $u_2^k = \min_{v \in \mathcal{Y}^\star} v_2$, the definition of $\mathcal{N}$ can be restricted to $\{(u_1^{i+1}, u_2^i) \,|\, 1 \leq i \leq k-1\}$ since there cannot be any feasible solution with a cost strictly smaller than u_1^1 on the first component, nor a solution of cost strictly smaller than u_2^k on the second component[1]. It typically occurs when the determination of the lexicographically smallest solutions is easy, which is typically the case when the single-objective version of the problem is solvable in polynomial time. Hence, the set $\mathcal{N}$ can here be viewed as a generalization of the nadir point: indeed, if we want that $|\mathcal{N}| = 1$, then the best point we can choose is the nadir of $\mathbf{UB}$. As for Λ, the larger $|\mathcal{N}|$ is, the better the approximation of $\mathbf{UB}^\prec$ becomes, but it requires a greater computational

[1] In such a case, we have $\mathbf{UB}^\prec \neq \{v \in \mathbb{R}^p \,|\, \exists w \in \mathcal{N},\, v \prec\!\!\!\prec w\}$, but this does not invalidate the approach since the difference is due to parts of the objective space corresponding to no feasible solution.

effort. This set $\mathcal{N}$ can be used instead of $\mathbf{UB}^{\prec}$ to find a separating function h, as shown by the following lemma.

Lemma 2 *When h is monotonic w.r.t. strong dominance (i.e., $v \prec\!\!\!\prec w \Rightarrow h(v) < h(w)$), a sufficient condition for a function h to separate $\mathbf{UB}^{\prec}$ and $\mathcal{Y}_\Lambda(N)$ is:*

$$\begin{cases} h(w) \leq 0 & \forall w \in \mathcal{N} \\ h(v) \geq 0 & \forall v \in \mathcal{Y}_\Lambda(N) \end{cases}$$

Proof. Assume that h separates $\mathcal{N}$ and $\mathcal{Y}_\Lambda(N)$. We show that $[\forall w \in \mathcal{N},\ h(w) \leq 0] \Rightarrow [\forall v \in \mathbf{UB}^{\prec},\ h(v) < 0]$. Consider $v \in \mathbf{UB}^{\prec}$. By definition of $\mathcal{N}$, there exists $w \in \mathcal{N}$ such that $v \prec\!\!\!\prec w$. Consequently, by monotonicity of h, we have $h(v) < h(w)$. Hence, $h(v) < 0$ since $h(w) \leq 0$ by assumption. Therefore h separates $\mathbf{UB}^{\prec}$ and $\mathcal{Y}_\Lambda(N)$. ∎

Since h_Λ is monotonic, this result implies that if $h_\Lambda(w) \leq 0$ for all $w \in \mathcal{N}$ then h_Λ separates $\mathbf{UB}^{\prec}$ and $\mathcal{Y}_\Lambda(N)$ (note that one always has $h_\Lambda(v) \geq 0$ for all $v \in \mathcal{Y}_\Lambda(N)$). By Lemma 1, it then also separates $\mathbf{UB}^{\prec}$ and $f(\mathcal{X}(N))$. Finally, a sufficient condition to discard the current node N is therefore that, for all $w \in \mathcal{N}$, we have $h_\Lambda(w) \leq 0$. In the single objective case, we clearly have $\mathcal{N} = \{\mathbf{UB}\}$, which means that the node is discarded if $\mathbf{UB} \leq \mathbf{LB}(N)$, which is the well-known condition.

Example 1 *Consider a node N such that the set of feasible values in the sub-tree is $f(\mathcal{X}(N)) = \{(4,10),(5,9),(6,6),(7,8),(10,4)\}$. Assume first that $\mathbf{UB} = \{(3,7),(7,3)\}$. Thus, we can set $\mathcal{N} = \{(7,7)\}$, as represented on the left part of Figure 2. Note that $(6,6) \in f(\mathcal{X}(N))$ and $(6,6) \prec\!\!\!\prec (7,7)$. Hence, $(6,6) \in \mathbf{UB}^{\prec}$ and therefore there cannot be a separating hypersurface between $\mathbf{UB}^{\prec}$ and $f(\mathcal{X}(N))$. Consequently, the node is not discarded. Assume now that $\mathbf{UB} = \{(3,7),(5,5),(7,3)\}$. We can then set $\mathcal{N} = \{(5,7),(7,5)\}$, as represented on the right part of Figure 2. By taking $\Lambda = \{(2,1),(\frac{1}{2},1),(1,0),(0,1)\}$, the frontier of $\mathcal{Y}_\Lambda(N)$ is the convex hull of $\mathcal{X}(N)$ and is a separating hypersurface between $\mathbf{UB}^{\prec}$ and $f(\mathcal{X}(N))$. Indeed, from $h_\Lambda(5,7) = \min\{-1, \frac{1}{2}, 1, 3\} < 0$ and $h_\Lambda(7,5) = \min\{1, -\frac{1}{2}, 3, 1\} < 0$, we deduce that h_Λ separates $\mathbf{UB}^{\prec}$ and $f(\mathcal{X}(N))$. Consequently, the node is discarded. Note that the ideal point of $f(\mathcal{X}(N))$ is $(4,4)$. Hence, if we had used this point to decide whether or not to discard N, the node would not have been discarded since $(4,4) \prec (5,5) \in \mathbf{UB}$.*

We complete the presentation of MOBB by two pratical observations about managing $\mathcal{N}$. Let us first consider a node N of the search tree that cannot be discarded. Function

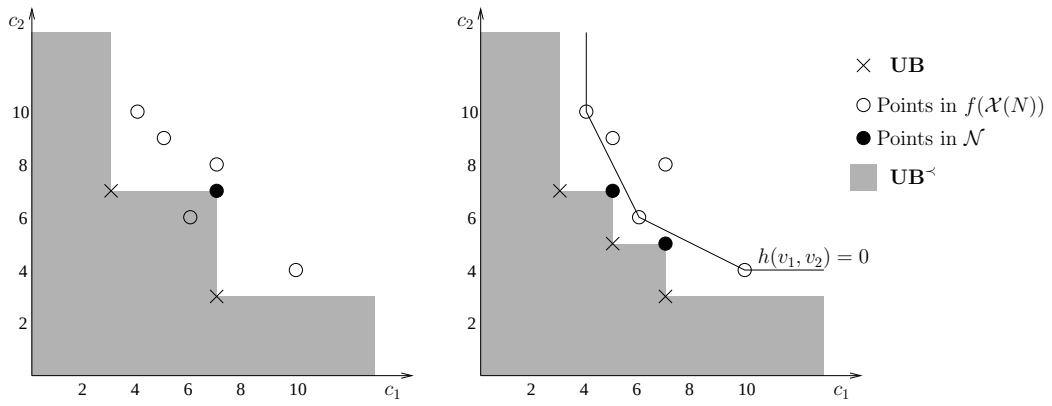

Figure 2: Piecewise linear separating hypersurface

h_Λ has been calculated and $h_\Lambda(v) \geq 0$ for all $v \in \mathcal{Y}_\Lambda(N)$. Since N is not discarded, we have $h_\Lambda(w) > 0$ for some $w \in \mathcal{N}$ but we may also have $h_\Lambda(w') \leq 0$ for some $w' \in \mathcal{N}$. Let us consider such a w' if exists. Since h_Λ is monotonic w.r.t. weak dominance (i.e., $v \preccurlyeq w \Rightarrow h(v) \leq h(w)$), any feasible solution that is a descendant of the current node N cannot have a cost $v \prec\!\!\!\prec w'$. Therefore, we can avoid to test whether $h_\Lambda(w') \leq 0$ in any node that is a descendant of N or, in other words, w' can be removed from $\mathcal{N}$ while exploring the subtree rooted at N. In practice, decreasing the size of $\mathcal{N}$ makes the separation test significantly faster.

The second practical improvement assumes that the possible values of the objective functions are integer, which is quite common in both single and multi-objective combinatorial optimization problems. In such a case, we can observe that if we replace $\mathcal{N}$ (the vectors of which have integer components) by $\mathcal{N}' = \{w - (1-\varepsilon, \ldots, 1-\varepsilon) \,|\, w \in \mathcal{N}\}$ $(0 < \varepsilon \leq 1)$ then no integer point is removed from the set of points covered by $\mathcal{N}'$ that is

$$\{v \in \mathbb{Z}^p \,|\, \exists w \in \mathcal{N},\, v \prec\!\!\!\prec w\} = \{v \in \mathbb{Z}^p \,|\, \exists w \in \mathcal{N}',\, v \prec\!\!\!\prec w\}.$$

Therefore, MOBB can use $\mathcal{N}'$ instead of $\mathcal{N}$ without losing any optimal solution.

3. Bi-objective minimum spanning tree

While the single-objective minimum spanning tree problem is easily solved, the introduction of multiple objectives significantly complicates the task. The bi-objective version can be formulated as follows: given a connected graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges, where each edge $e \in \{e_1, \ldots, e_m\}$ is valued by an integer-valued vector $w^e = (w_1^e, w_2^e)$,

the set $\mathcal{X}$ of feasible solutions is the set of spanning trees of G. Each spanning tree is characterized by a m-tuple x of booleans such that $x_i = 1$ if edge e_i belongs to the tree, $x_i = 0$ otherwise. The value of a tree in the objective space is then $f(x) = (f_1(x), f_2(x))$, where $f_j(x) = \sum_{i=1}^{m} x_i w_j^{e_i}$ $(j = 1, 2)$. The goal is to find (at least) one spanning tree for each Pareto point in the objective space. After briefly recalling previous works on this problem, we describe in the following the different phases of the branch-and-bound we propose for the bi-objective spanning tree problem (BOST): preprocessing of the graph, initialization of the set **UB**, presolve methods to speed up the search, and search algorithm itself.

3.1. Related works

The multi-objective spanning tree problem (MOST) has been introduced by Corley (1985), and proved NP-hard by Emelichev and Perepelitsa (1988) (see also Hamacher and Ruhe, 1994), even when there are only two objectives. Several practical approximation methods have been proposed (Hamacher and Ruhe, 1994; Zhou and Gen, 1999; Knowles and Corne, 2001b). On the theoretical side, a fully polynomial time approximation scheme has been provided (Papadimitriou and Yannakakis, 2000), and very recently the expected runtimes of a simple evolutionary algorithm have been studied (Neumann, 2006). Concerning more specifically the BOST problem, to the best of our knowledge, only two operational exact methods (Ramos et al., 1998; Steiner and Radzik, 2006) have been proposed until now, both based on a two-phase (exact) procedure (Visée et al., 1998). They first calculate the set of *extreme* Pareto optimal solutions (i.e., vertices of the convex hull of all solutions in the objective space) then they compute the set of non-extreme Pareto optimal solutions located in the triangles generated in the objective space by two successive extreme solutions[2]. The methods mainly differ in the way they compute Pareto-optimal solutions in the triangles: in the method by Ramos et al. (1998), these solutions are computed by a branch-and-bound procedure discarding any node such that an ad-hoc bounding point (that is weakly dominated by the ideal point of the corresponding sub-problem) falls outside the triangles; in the method by Steiner and Radzik (2006), they are computed by a sequence of applications of a k-best algorithm for the single-objective version of the problem. Both methods have been implemented by Steiner and Radzik (2006): the latter is shown to run significantly

[2]Strictly speaking, a two-phase procedure is often described as the generation of *supported* Pareto optimal solutions and then non-supported Pareto optimal solutions. For the sake of simplicity, we do not elaborate here.

faster on instances with up to 121 vertices for grid or planar graphs, and up to 38 vertices for complete graphs.

3.2. Preprocessing of the graph

Section 3.2 to Section 3.5 are devoted to preliminary computations that are all processed at the root node of the search tree. Then Section 3.6 considers the instanciation of the MOBB procedure to our BOST problem. This section first presents a preprocessing of the graph which aims to remove some edges of G or make some edges mandatory (that is they must be in any spanning tree) without changing the set $\mathcal{Y}^\star$ of Pareto points in the objective space. Note that the rules presented in this section are not only valid for the bi-objective problem but for the general multi-objective spanning tree problem with p objectives.

As often done when presenting algorithms for constructing spanning trees (see e.g. Tarjan, 1983), we describe our preprocessing phase as an edge coloring process. Initially all edges are uncolored. We color one edge at a time either blue (mandatory) or red (forbidden). At each step of the preprocessing phase, we have therefore a partial coloring $c(\cdot)$ of the edges of G. For a partial coloring c, we denote by $\mathcal{X}(c)$ the set of trees including all blue edges, some of the uncolored ones (possibly none), and none of the red ones. We denote by $\mathcal{Y}(c)$ the corresponding values in the objective space. As usual, the Pareto front of $\mathcal{Y}(c)$ is denoted by $\mathcal{Y}^\star(c)$. The aim of the preprocessing phase is to color as many edges as possible, so as to get a maximal coloring c such that $\mathcal{Y}^\star(c) = \mathcal{Y}^\star$. We now give conditions under which an edge can be colored blue or red, which are adaptations of the well-known *cut optimality condition* and *cycle optimality condition* to the multi-objective case. Note that Hamacher and Ruhe (1994) also give some properties related to these rules but they cannot be applied in our context. Indeed, Hamacher and Ruhe provide *necessary* conditions for a *spanning tree* to be Pareto optimal. However, we need here *sufficient* conditions for an *edge* to be mandatory (for each Pareto point $y \in \mathcal{Y}^\star(c)$, there exists at least one spanning tree $x \in \mathcal{X}(c)$ including that edge for which $f(x) = y$) or forbidden (for each Pareto point $y \in \mathcal{Y}^\star(c)$, there exists at least one spanning tree $x \in \mathcal{X}(c)$ without that edge for which $f(x) = y$), so as to be able to recover one spanning tree for each Pareto point in the objective space.

Before introducing the optimality conditions, we recall some definitions from graph theory. A *cut* in a graph is a partition of the vertices into two disjoint sets and a *crossing edge* (with respect to a cut) is one edge that connects a vertex in one set to a vertex in the other.

When there is no ambiguity, the term *cut* will also be used to refer to the set of crossing edges defined by the partition of the vertices.

Proposition 1 (optimality conditions) *Let G be a connected graph with coloring c of the edges. The following properties hold:*
(i) Cut optimality condition. *Let us consider a cut in G with no blue edge and let C denote the set of crossing edges. If there is some uncolored edge $e \in C$ such that $e \preccurlyeq e'$ for all uncolored edges $e' \in C$, then e is in at least one tree of cost y for all $y \in \mathcal{Y}^\star(c)$.*
(ii) Cycle optimality condition. *Let us consider a cycle C in G containing no red edge. If there is some uncolored edge $e \in C$ such that $e' \preccurlyeq e$ for all uncolored edges $e' \in C$, then e can be removed from G without changing $\mathcal{Y}^\star(c)$.*

Proof. The proof is similar to the single objective case (see e.g. Tarjan, 1983).
Proof of (i). Suppose that there exists a cut C and a crossing uncolored edge e that satisfy the cut optimality condition. Let x be a Pareto optimal spanning tree of $\mathcal{X}(c)$ that does not contain e. Now consider the graph formed by adding e to x. This graph has a cycle that contains e, and that cycle must contain at least one other uncolored crossing edge — say f — such that $f \in C$, and therefore $e \preccurlyeq f$. We can get a new spanning tree $x' \in \mathcal{X}(c)$ by deleting f from x and adding e. Its cost $f(x')$ is equal to $f(x) - w^f + w^e \preccurlyeq f(x)$. By Pareto optimality of x, we have $f(x') = f(x)$ and x' is therefore a Pareto optimal tree in $\mathcal{X}(c)$ with the same value as x.
Proof of (ii). Suppose that there exists a cycle C containing no red edge with an uncolored edge $e \in C$ such that $e' \preccurlyeq e$ for all non-blue edges $e' \in C$. Let x be a Pareto optimal spanning tree of $\mathcal{X}(c)$ that contains e. Now consider the graph formed by removing e from x. This graph is compounded of two connected components. The induced cut contains at least one other uncolored crossing edge — say f — such that $f \in C$, and therefore $f \preccurlyeq e$. We can get a new spanning tree $x' \in \mathcal{X}(c)$ by deleting e from x and adding f. Its cost $f(x')$ is equal to $f(x) - w^e + w^f \preccurlyeq f(x)$. By Pareto optimality of x, we have $f(x') = f(x)$ and x' is therefore a Pareto optimal tree in $\mathcal{X}(c)$ with the same value as x. Hence, for any Pareto optimal tree in $\mathcal{X}(c)$ containing e, there is an equivalent tree in $\mathcal{X}(c)$ without e. ∎

The single objective versions of these conditions make it possible to design a generic greedy method (see e.g. Tarjan, 1983), from which Kruskal's and Prim's algorithms can be derived. The multi-objective counterpart of this generic greedy method is indicated on

Algorithm GREEDY(G,c)
Input : A MOST problem on a connected graph $G = (V, E)$ with coloring c_1 of the edges
Output : Returns a coloring c_2 such that the MOST problem with coloring c_2 is equivalent to the input MOST problem
if one of the following rules can be applied:
Blue rule: **if** there is an uncolored edge e s.t. the cut optimality condition holds **then** set $c(e) = \texttt{blue}$
Red rule: **if** there is an uncolored edge e s.t. the cycle optimality condition holds **then** set $c(e) = \texttt{red}$
then apply non-deterministically one of the enforceable rule and set $c = \text{GREEDY}(G,c)$
return c

Figure 3: Greedy algorithm for the MOST problem

Figure 3. However, unlike the single objective case, it does not necessarily yield a complete coloring of the graph. As indicated earlier, the returned coloring c is such that $\mathcal{Y}^\star(c) = \mathcal{Y}^\star$, i.e. for all $y \in \mathcal{Y}^\star$ there is a spanning tree of cost y containing all the blue edges and no red one. Actually, this property is an invariant of the greedy algorithm, the validity of which directly follows from Proposition 1. Clearly, as soon as some edges remain uncolored, the coloring is insufficient to deduce the set of Pareto optimal trees. Therefore, we use this algorithm as a preprocessing of the graph prior to the exact resolution of the BOST problem by our MOBB algorithm.

The complexity of the greedy method strongly depends on the complexity of detecting uncolored edges satisfying an optimality condition. In practice, to determine whether an uncolored edge $e = \{v, w\}$ satisfies the cut optimality condition, one performs a depth first search from v in the partial graph $G_e^{\text{cut}} = (V, E_e^{\text{cut}})$ where $E_e^{\text{cut}} = \{e' \in E \,|\, \text{not}(e \preccurlyeq e')\} \cup \{e' \in E \,|\, c(e') = \texttt{blue}\}$. If w belongs to the set of visited vertices, then the partition between visited and non-visited vertices constitutes a cut for which e satisfies the cut optimality condition. Similarly, to determine whether an uncolored edge $e = \{v, w\}$ satisfies the cycle optimality condition, one performs a depth first search from v in the partial graph $G_e^{\text{cyc}} = (V, E_e^{\text{cyc}})$ where $E_e^{\text{cyc}} = \{e' \in E \,|\, e' \preccurlyeq e\} \setminus \{e\} \cup \{e' \in E \,|\, c(e') = \texttt{blue}\}$. If w is visited, then the chain from v to w in the search tree, completed with $\{v, w\}$, constitutes a cycle for which e satisfies the cycle optimality condition. Since the number of edges in the graph is m and the complexity of a depth first search is within $O(m)$ in a connected graph, the complexity of carrying out the blue rule and the red rule is within $O(m^2)$. Since each recursive call of the algorithm in Figure 3 colors one edge, there are at most m recursive calls, which means that

the algorithm runs in $O(m^3)$. However, we are going to show that one single pass is enough to find a *maximal* coloring, i.e. a coloring for which no additional edge can be colored (this coloring is not unique). This algorithm is based on the two following observations:

- *Coloring an edge does not make it possible to color additional edges in red.* By contradiction, assume that the coloring of an edge — say f — makes it possible to color in red an edge e for which the red rule did not apply before. Clearly, f is colored blue and there exists a cycle C including f and e where f is the unique edge such that $f \not\preccurlyeq e$. Since f is colored blue, there exists a cut C' for which the cut optimality condition is fulfilled. Consider now $f' \in C \cap C'$ (such an edge necessarily exists and $f' \neq e$). Edge f' cannot be blue (since it belongs to C') nor red (since it belongs to C). Hence, f' is uncolored and therefore $f \preccurlyeq f'$ by the cut optimality condition in C'. Furthermore, by the cycle optimality condition in C, we also have $f' \preccurlyeq e$. By transitivity of $\preccurlyeq$, it follows that $f \preccurlyeq e$. It contradicts the initial assumption that $f \not\preccurlyeq e$.

- *Coloring an edge does not make it possible to color additional edges in blue.* By contradiction, assume that the coloring of an edge — say f — makes it possible to color in blue an edge e for which the blue rule did not apply before. Clearly, f is colored red and there exists a cut C including f and e where f is the unique edge such that $e \not\preccurlyeq f$. Since f is colored red, there exists a cycle C' for which the cycle optimality condition is fulfilled. Consider now $f' \in C \cap C'$ (such an edge necessarily exists and $f' \neq e$). Edge f' cannot be blue (since it belongs to C) nor red (since it belongs to C'). Hence, f' is uncolored and therefore $f' \preccurlyeq f$ by the cycle optimality condition in C'. Furthermore, by the cut optimality condition in C, we also have $e \preccurlyeq f'$. By transitivity of $\preccurlyeq$, it follows that $e \preccurlyeq f$. It contradicts the initial assumption that $e \not\preccurlyeq f$.

From these observations, it follows that the $O(m^2)$ algorithm given in Figure 4 finds a maximal edge coloring.

3.3. Initial Pareto front

As mentioned in the beginning of Section 2.2, a branch-and-bound algorithm is notoriously more efficient when good solutions are known even before starting the search. In our approach, **UB** is initialized by a two-phase (approximation) procedure, similar to the one used by Hamacher and Ruhe (1994): first, the extreme solutions are computed and second, local

Algorithm $O(m^2)$ IMPLEMENTATION
Input : A MOST problem on a connected graph $G = (V, E)$
Output : Returns a maximal coloring c
for each edge e of E **do**
  if the cut optimality condition holds for e **then**
    set $c(e) =$ `blue`
  else if the cycle optimality condition holds for e **then**
    set $c(e) =$ `red`
return c

Figure 4: An $O(m^2)$ algorithm to compute blue and red edges

search (starting with the extreme solution) is launched. These two phases are more precisely described below:

1. *Computation of the set ES of extreme solutions.* First, the two lexicographically optimal solutions are computed by resorting to Kruskal's algorithm. Second, the set ES of extreme solutions is initialized with the two obtained solutions, and maintained in increasing order w.r.t. the first objective. Set ES is computed recursively as follows: given two consecutive solutions x_1 and x_2 in ES, a new extreme solution is computed by solving a standard minimum spanning tree problem after scalarizing the vector valuations (α_i, β_i) of each edge e_i by a weighted sum $\lambda_1\alpha_i + \lambda_2\beta_i$, with $\lambda_1 = f_2(x_2) - f_2(x_1)$ and $\lambda_2 = f_1(x_1) - f_1(x_2)$. Actually, a superset of the extreme solutions can be returned at the end of this phase. Andersen et al. (1996) observed that the algorithm is polynomial, using a result of Chandrasekaran (1977) proved in the context of minimal ratio spanning trees. The polynomial complexity of the algorithm is also a corollary of the following lemma that is also very important in Section 3.5.

   Lemma 3 *Let us consider the m pairs (α_i, β_i) for $1 \leq i \leq m$. Let $S(\lambda)$ be the sequence of indices $1, \ldots, m$ lexicographically sorted according to the $\lambda\alpha_i + (1-\lambda)\beta_i, i$ values when λ varies in $[0, 1]$. Then we have that $\{S(\lambda) \,|\, 0 \leq \lambda \leq 1\}$ contains at most $m(m-1)/2 + 1$ different sequences.*

   Proof. The proof is geometric. Parallel graduated axes are associated with both objectives, as represented on Figure 5. We call first axis the one associated with objective one, its equation is the line $y = 1$. The second axis is the line $y = 0$ and is

associated with objective two. Each pair (α_i, β_i) is then in correspondence to a segment I_i ($i \in \{1, \ldots, m\}$) linking the point $(\alpha_i, 1)$ on the first axis to the point $(\beta_i, 0)$ on the second axis. Given $\lambda \in [0, 1]$, consider the line $y = \lambda$. The sequence in regard to which the segments I_i are crossed by this line gives the sequence of increasing values $\lambda\alpha_i + (1 - \lambda)\beta_i$. When several segments are crossed simultaneously on a same point, they are ranked according to index i. The sequence gives therefore precisely the lexicographical sorting according to the $\lambda\alpha_i + (1 - \lambda)\beta_i, i$ values. Consequently, the changes in sequence $S(\lambda)$ correspond to crossings of segments i. Since the number of crossing points is at most $m(m - 1)/2$, we deduce that the cardinal of this set of sequences is at most $m(m - 1)/2 + 1$. ∎

Note that we use index i in Lemma 3 in order that there is a unique possible sequence $S(\lambda)$ for a given λ, otherwise several edges with the same weighted sum could be sorted in many (maybe exponential) ways.

2. *Approximation of the Pareto front.* The initial set **UB** is computed from ES by iteratively performing a local search around the solutions of **UB**. More precisely, the algorithm is initialized by setting **UB** $= ES$, and all the points are set as "unvisited". Two spanning trees are neighbours if they have $(n - 2)$ edges in common. The local search selects an unvisited point in **UB** (and its corresponding spanning tree s) and computes the set $N(s)$ of neighbors of s. Then **UB** is replaced by the set of non-dominated points in **UB** $\cup\, f(N(s))$. If $f(s)$ is still in **UB**, its status becomes visited and the search continues with a new unvisited point. The algorithm stops when all the points in **UB** are visited. Local search seems to be especially convenient for BOST, since it appears that the set of Pareto optimal spanning trees is often connected with respect to the usual definition of neighbourhood (Ehrgott and Klamroth, 1997). Furthermore, if p is considered as a constant ($p = 2$ for BOST), this phase is performed within a pseudopolynomial number of iterations since $|N(S)| \leq m(n - 1)$ and the number of points in the objective space is at most $(n - 1)^p(w_{\max} + 1)^p$ with $w_{\max} = \max_{e,i} w_i^e$.

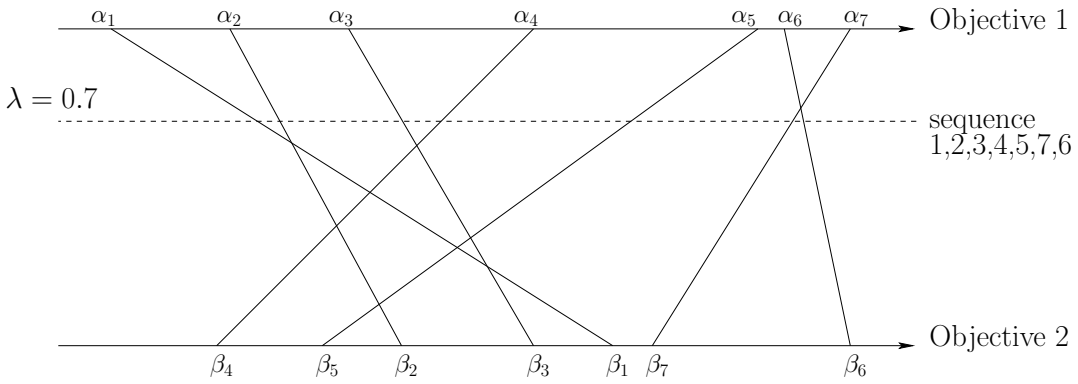

Figure 5: Proof of Lemma 3

3.4. Root lower bound

Let us now consider the bounding phase. The computation of $h_\Lambda(c_1, c_2)$ is greatly eased by the fact that minimizing the weighted sum of the two objectives is a polynomial problem. Therefore, by calculating all the extreme solutions of the sub-problem attached to the current node, we have the best possible h_Λ function. Since $|ES|$ is polynomial, h_Λ is also computed in polynomial time.

3.5. Presorting edges

As a consequence of Lemma 3, there are $O(m^2)$ interesting sequences of the edges, so that they all can be efficiently stored in memory. The benefit of storing all the pre-sorted sequences of edges is that for any $\lambda \in [0, 1]$, $S(\lambda)$ can be retrieved in $O(\log(m(m-1)/2))$ that is in $O(\log n)$ while computing it from scratch takes $O(m \log n)$ time (since $m \in O(n^2)$). We simply use a balanced binary tree data structure to store the $O(m^2)$ sequences, which means that the memory requirement of the structure is in $O(m^3)$.

Moreover, the pre-sorting and the initialization of the structure can be done in only $O(m^3)$ time by a simple sweeping algorithm where λ is successively increased from 0 to 1. This algorithm has $O(m^2)$ steps that corresponds to the crossovers in Figure 5. At each step, a new array of m integers is build to store the new sequence, which is done in $O(m)$ time.

Therefore, as soon as the branch-and-bound search requires a sufficiently large number of nodes, the time spent in this preliminary phase is easily balanced by the time spared during the search. Implementation details are given in Section 4.1.

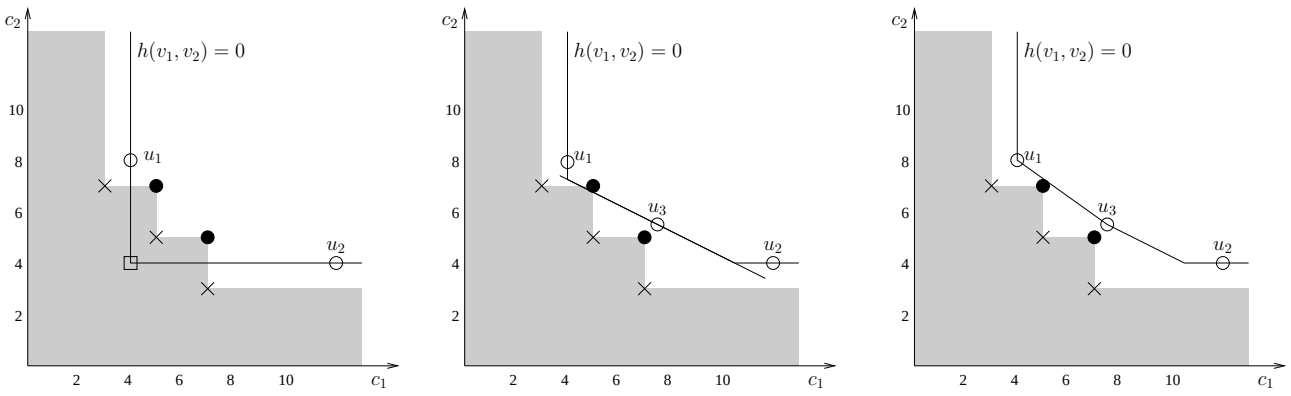

Figure 6: Computing the lower bound

3.6. Search algorithm

3.6.1. Branching scheme

The branching scheme is very simple: at each node, an edge e of G is selected and we create two subproblems. In the first one, edge e is mandatory (it must be in the spanning tree) while in the second one, edge e is forbidden (it is removed from G). The heuristic to select e searches for the uncolored edge such that $\min(w_1^e, w_2^e)$ is minimal.

3.6.2. Computing the lower bound

When a descendant node is created, the computation of the separating hypersurface can be speeded up by using the separating hypersurface of the father node. Indeed, to each extremal point of the hypersurface is attached a feasible solution for the father node. Clearly, if this solution is also feasible for the descendant node, it is also extremal for the separating hypersurface of the descendant node.

Therefore, instead of computing it from scratch, we initialize the computation of the separating hypersurface using the "feasible" extremal points of the father. Then the computation continues as indicated in Section 3.4 for the root lower bound. However, the computation can be made lazy, in the sense that some superfluous calculation may be avoided. It is illustrated by Figure 6. Initially, u_1 and u_2 are computed: they represent the optimal points when the two objectives are sorted lexicographically according to (c_1, c_2) and (c_2, c_1) respectively. Therefore, we know that $\mathcal{Y}(N)$ is above u_2 and to the right of u_1. At this step, we cannot conclude whether node N can be discarded because some points of $\mathcal{N}$ are in this set. Then, we search for an optimum, say u_3, of the problem with (single) criterion $\lambda f_1(x) + (1-\lambda) f_2(x)$ where λ is defined such that $\lambda c_1 + (1-\lambda) c_2 = \mu$ is a linear equation of the straight line

D defined by u_1 and u_2. By construction, the points of $\mathcal{X}(N)$ are above the line parallel to D containing u_3. We observe that there is still a nadir point above this line but we can eventually get the separating hypersurface by optimizing $\lambda' f_1(x) + (1 - \lambda' f_2(x)$ where λ' is defined such that $\lambda' c_1 + (1 - \lambda')c_2 = \mu'$ is a linear equation of the straight line D' defined by u_1 and u_3. We say that the computation is lazy because we avoided to check whether the line defined by u_2 and u_3 supports the convex hull of $\mathcal{Y}(N)$: indeed, this information is not required to discard N.

3.6.3. Updating the incumbent Pareto front

When a point $w \in \mathcal{N}$ satisfies $h_\Lambda(w) > 0$ then the node N is not discarded but, before branching, the algorithm checks whether a newly computed extremal point of the separating hypersurface can be inserted into **UB**. Indeed, we take advantage of the property that these extremal points of the separating hypersurface correspond to feasible solutions.

4. Experimental results

4.1. Implementation details

The branch-and-bound algorithm has been implemented in C#[3] and was run on a 3.6 GHz personal computer with a memory of 2GB. In this implementation, the presorting procedure is called only if the set $\mathcal{N}$ computed at the root node contains at least 10 points. Indeed, if $|\mathcal{N}| < 10$, it means that the gap between the root **UB** and the convex envelop of $\mathcal{Y}$ is very narrow and we heuristically consider that the search will be short —and therefore refined preprocessing is not required. Note that in practice we can indeed have $|\mathcal{N}| < 10$ even if $|\mathbf{UB}|$ is large (e.g. greater than 100): both improvements presented at the end of Section 2.2 lead to a drastic decrease of the size of $\mathcal{N}$ even at the root node.

In order to reduce at most the number of uncolored edges at the root node, we have also implemented a *shaving* procedure. The term "shaving" was introduced by Martin and Shmoys (1996) for the job-shop scheduling problem. This procedure works as follows: for each edge e left uncolored by the application of the cycle and cut rules, we build a subproblem in which e is colored blue. If the computation of the lower bound proves that the subproblem cannot improve the incumbent **UB**, then it means that e can be colored red at the root problem. In our tests, this procedure colors red about 20% of the the uncolored edges and is

[3]Available online `http://www-poleia.lip6.fr/~sourd/project/nadei/`

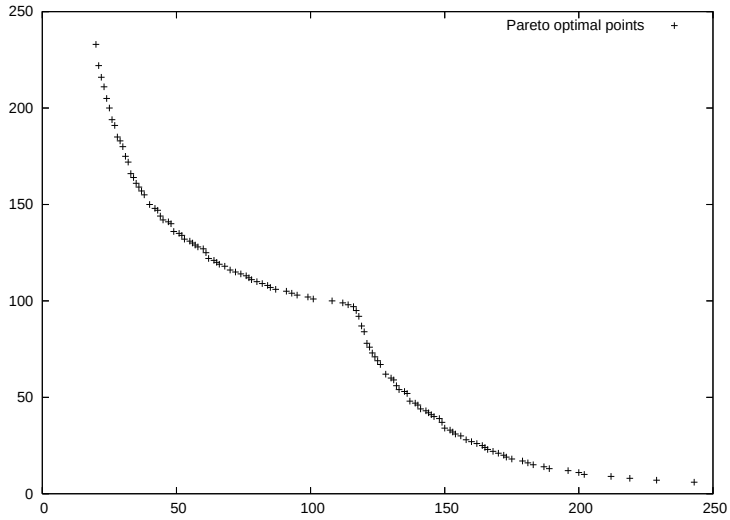

Figure 7: Typical Pareto front for "hard" instances

therefore very useful to limit the total computation time. As for the presorting, the shaving procedure is run only if $|\mathcal{N}| < 10$.

4.2. Instances

The experimental tests are based on three classes of graphs. First, the *random* graphs of density d are graphs with n nodes and about $dn(n-1)/2$ edges. More precisely, for each pair of nodes $\{i, j\}$, the edge is generated with probability d. We check that the generated graph is connected. When $d = 1$, the generated graphs are called *cliques*, that is the graphs are complete. Second, the *grid* graphs have $n = a^2$ nodes, which are generated by creating an $a \times a$ grid and joining the neighboring vertices horizontally and vertically. For both clique and grid graphs, the weights w_1^e and w_2^e of each edges are drawn from the uniform distribution $[0, K]$ where the parameter K represents the maximum edge weight. The construction of the instances of the last class (called *hard* instances) is based on the work of Knowles and Corne (2001a). These instances are said to be hard because there are some Pareto optimal solutions that are far from any supported solution (see Figure 7). Typically, the k-best approach is very bad for such instances (that have not been tested by Steiner and Radzik, 2006). We first build a clique instance for some given parameters n and K. Let (v_1, v_2) be the minimum cost when minimizing the two objectives lexicographically. We then define the constant $M = \frac{2}{3}\max(v_2 - v_1, 0)$. We randomly select a node x of the clique and add two nodes y and z to the graph with the edges $e = \{x, y\}$, $f = \{x, z\}$ and $g = \{y, z\}$ and the

Graph	n	m'	blue	LS	Sort	Shaving	BnB	Total
Clique	50	156.5	1.4	0.62	0.05	0.05	0.75	1.54
	100	339.8	2.8	4.34	0.56	0.23	3.17	8.56
	150	534.3	3.0	12.71	1.89	0.40	3.87	19.68
	200	727.0	3.5	28.22	4.10	0.45	4.10	38.64
	250	905.3	4.8	52.26	7.06	0.46	2.28	65.74
	300	1075.9	5.2	87.94	9.47	0.43	7.61	105.82
	350	1181.8	8.7	137.92	13.27	0.58	4.87	154.01
	400	1650.4	7.4	208.94	-	-	1.56	225.68
	450	1839.3	10.1	308.38	-	-	0.44	331.68
	500	1980.9	13.6	438.75	-	-	1.88	473.34
Grid	25	34.7	10.8	0.03	0.01	0.01	0.01	0.1
	100	147.9	41.7	1.21	0.03	0.03	1.16	2.48
	144	215.8	56.4	5.02	0.07	0.04	5.46	10.69
	225	336.4	86.3	30.50	0.25	0.13	31.42	62.54
	256	380.2	96.2	48.75	0.39	0.17	53.54	103.22
	289	434.5	110.0	81.47	0.56	0.26	87.22	170.06
	324	482.8	126.9	123.22	0.77	0.30	124.39	249.39
	361	542.1	136.3	199.48	1.12	0.41	203.82	405.79
	400	595.0	156.1	261.43	1.46	0.51	226.22	490.87
Hard	50	200.3	3.2	1.2	0.1	0.0	2.6	4.0
	100	430.0	3.7	8.2	0.5	0.2	11.6	20.9
	150	665.0	4.5	24.9	1.8	0.3	19.3	47.5
	200	891.8	4.2	53.6	3.9	0.4	22.0	82.6
	250	1115.3	5.3	102.2	6.8	0.3	14.0	129.2
	300	1310.5	6.7	173.2	9.8	0.3	11.8	205.3
	350	1501.4	7.3	270.7	13.5	0.3	10.1	310.8
	400	1661.6	10.6	396.3	16.6	0.3	4.8	442.3

Table 1: Mean CPU time of each phase of the algorithm.

weights $w^e = (0, 0)$, $w^f = (M, 0)$ and $w^g = (0, M)$.

4.3. Results

In the first series of experiments, the parameter K is set to 100 (as in the tests of Steiner and Radzik, 2006). We study the computation times of the different phases of the branch-and-bound algorithm for each class of instances when the size of the graphs varies. Table 1 reports the results: each line reports the average results for 10 graphs of the same class generated for the corresponding value of n. Note that the results for random graphs with $d < 1$ are not reported here. Column m' displays the number of remaining edges after applying the

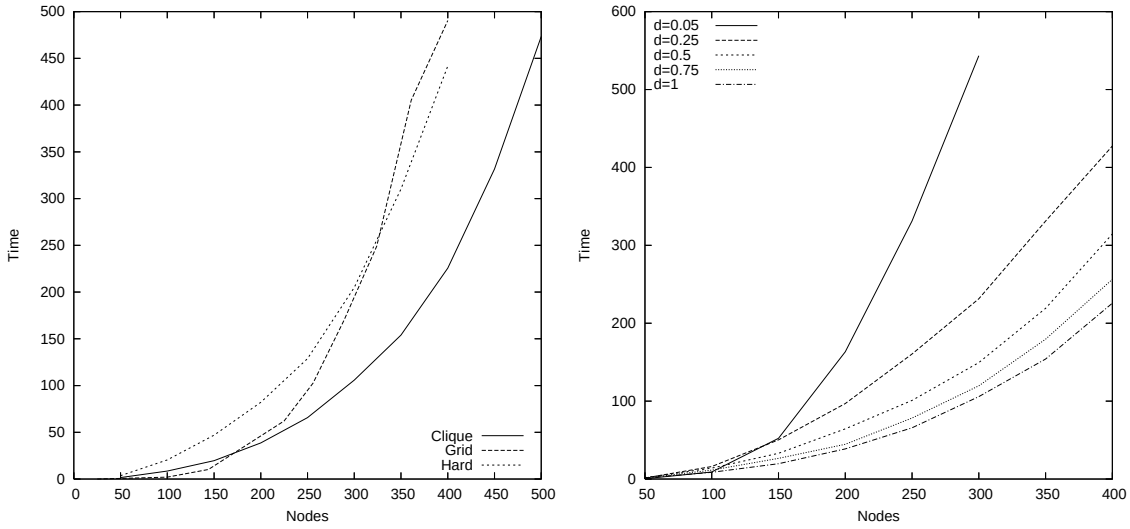

Figure 8: Performance for different class of graphs

cycle optimality conditions (red rules) and processing the shaving procedure. Column "blue" displays the number of edges that are colored blue with the cut optimality conditions. The four following columns respectively report the CPU times (in seconds) for the local search, presorting, shaving and branch-and-bound phases and the last column reports the total CPU time in seconds. Note that some cells in the table are left empty: it means that presorting and shaving procedures are never called for these graphs (we discuss this point later).

Figure 8 shows the computation times in function of the number of nodes of the graphs for each class of instances (the right part is devoted to random graphs). Before we study more carefully the behavior of our algorithm, we compare it to the results obtained by Steiner and Radzik (2006). Roughly speaking, they can solve grid instances with up to 121 nodes and random instances with at most 38 nodes. Conversely, for all the classes, our algorithm can solve instances with 400 nodes —and even more— in a similar computation time. An important difference between the two algorithms is that the k-best approach of Steiner and Radzik (2006) leads to compute $\mathcal{X}^\star$ whereas our branch-and-bound computes $\mathcal{Y}^\star$. Therefore, one could argue that the comparaison in unfair, however we should note that:

- $|\mathcal{X}^\star|$ is generally exponential while $|\mathcal{Y}^\star|$ is pseudopolynomial since it contains at most nK points. Therefore any approach that searches for $\mathcal{X}^\star$ is necessary limited to smaller instances because computation time is in $\Omega(|\mathcal{X}^\star|)$ and it seems questionable to render to the decision maker an exponentially large output. Moreover $\mathcal{Y}^\star$ can easily be represented by a graph (as in Figure 7).

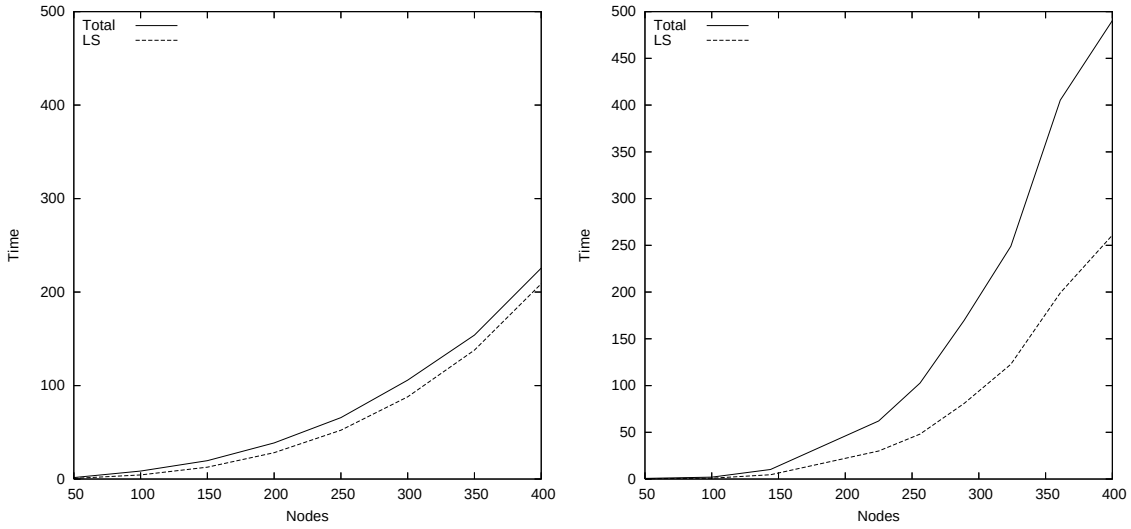

Figure 9: CPU time according to the size of the clique graph (left) or the grid graph (right)

- The k-best approach works on elements of $\mathcal{X}$ therefore, it seems unlikely to have an adaptation of this approach to efficiently compute $\mathcal{Y}^\star$. Conversely, our branch-and-bound approach can be adapted in order to directly search for either $\mathcal{Y}^\star$ or $\mathcal{X}^\star$. Some tests have shown that we are able to compute $\mathcal{X}^\star$ for 15×15 grid graphs and cliques with 65 nodes in less than 10 minutes whereas Steiner and Radzik (2006) only solve 9×9 grid graphs and cliques with 26 nodes.

- The k-best approach seems highly inappropriate to solve instances where some Pareto optimal points are not in the very best solutions for any linear combination of the objectives, such as the instances of our "hard" class. Table 1 and Figure 8 show that these instances remain satisfactorily tractable with our branch-and-bound.

A surprising conclusion can be drawn from the global study of these experimental results. For small graphs —typically $n < 100$— instances with low density graphs —that is random graphs with $d = 0.05$ and grid graphs— are easier than instances with high density graphs such as cliques. However, when n is larger, the conclusion is opposite: large grid graphs lead to more difficult instances than cliques of the same size. This fact is surprising because a clique contains significantly more spanning trees that a grid. To explain this behavior, we compare in Figure 9 the ratio between the local search phase and the total CPU time for both cliques and grids. Clearly, the branch-and bound remains very fast for the large clique instances. It is due to the fact that at the root node there are very few points in the objective space between the initial **UB** and the separating hypersurface. In fact, due

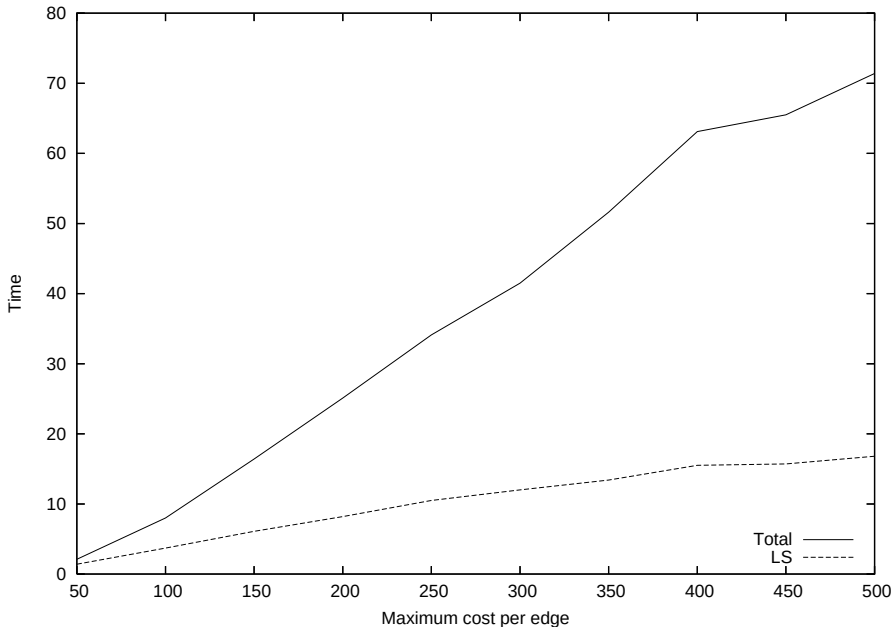

Figure 10: Computation time according to the edge cost parameter K

to the high number of spanning trees, most of the "efficient" costs in the two-dimensional objective space have at least one corresponding spanning tree. This assertion can be verified by checking the cardinality of $\mathcal{N}$ at the root node. Recall that the points of $\mathcal{N}$ that are not lower bounded by the separating hypersurface are removed from $\mathcal{N}$: so $|\mathcal{N}|$ reflects the number of areas in the objective space where **UB** might be improved. For large clique instances, $|\mathcal{N}|$ is indeed small: for all the tested instances with $n \geq 400$ we have $|\mathcal{N}| < 10$ — this is why the presorting is not processed for these instances (see Table 1).

We can explain this phenomenon because we have an exponential number of spanning trees but a pseudopolynomial number of different points in the objective space because any spanning tree cost is bounded by nK. To check this assertion, we have run our algorithms for clique graphs with 100 nodes but with different values of K. Figure 10 shows the computation times: clearly, instances are more difficult when K increases.

Finally, we would like to mention the quality of the simple local search heuristic. For more than 99% of the instances, it finds $\mathcal{Y}^\star$ at the root node; for these cases, the branch-and-bound only proves the optimality of the front. This last observation confirms the experience of Ehrgott and Klamroth (1997) with smaller graphs: they indeed mention that the set of efficient spanning trees is "only very rarely" disconnected.

5. Conclusion

In this paper, we have proposed a new multi-objective branch-and-bound procedure, strongly relying on the convex hull of the image in the objective space of the potential feasible solutions. Numerical experiments clearly show that this procedure, combined with an efficient presolve method, makes it possible to outperform state-of-the-art algorithms on the bi-objective spanning tree problem.

An interesting extension of this work would be to apply our procedure to problems involving more than two objectives. In particular, an important issue is to investigate how much the separating hypersurface must be refined to obtain good numerical performance: indeed, the convex hull of $\mathcal{Y}(N)$ will be exponentially large. Another research issue would be to test our procedure on combinatorial problems the single objective version of which is NP-hard (the computation of the convex hull of $\mathcal{Y}(N)$ is here exponential even in the bi-objective case).

References

Andersen, K.A., K. Jrnsten, M. Lind. 1996. On bicriterion minimal spanning trees: an approximation. *Computers and Operations Research* **23** 1171–1182.

Bitran, G., J.M. Rivera. 1982. A combined approach to solve binary multicriteria problems. *Naval Research Logistics Quarterly* **29** 181–201.

Chandrasekaran, R. 1977. Minimal ratio spanning trees. *Networks* **7** 335–342.

Corley, H.W. 1985. Efficient spanning trees. *Journal of Optimization Theory and Applications* **45** 481–485.

Daellenbach, H.G., C.A. De Kluyver. 1980. Note on multiple objective dynamic programming. *Journal of the Operational Research Society* **31** 591–594.

Ehrgott, M. 2000. *Multicriteria Optimization*, *Lecture Notes in Economics and Mathematical Systems*, vol. 491. Springer Verlag, Berlin. 2nd edition: 2005.

Ehrgott, M., X. Gandibleux. 2000. A survey and annoted bibliography of multiobjective combinatorial optimization. *OR Spektrum* **22** 425–460.

Ehrgott, M., X. Gandibleux. 2004. Approximative solution methods for multiobjective combinatorial optimization. *TOP, the OR journal of the Spanish Statistical and Operations Research Society* **12** 1–88.

Ehrgott, M., X. Gandibleux. 2007. Bound sets for biobjective combiantorial optimization problems. *Computers & Operations Research* **34** 2674–2694.

Ehrgott, M., K. Klamroth. 1997. Connectedness of efficient solutions in multiple criteria combinatorial optimization. *European Journal of Operational Research* **97** 159–166.

Emelichev, V.A., V.A. Perepelitsa. 1988. Multiobjective problems on the spanning trees of a graph. *Soviet Mathematics Doklady* **37** 114–117.

Hamacher, H.W., G. Ruhe. 1994. On spanning tree problems with multiple objectives. *Annals of Operations Research* **52** 209–230.

Kiziltan, G., E. Yucaoglu. 1983. An algorithm for multiobjective zero-one linear programming. *Management Science* **29** 1444–1453.

Knowles, J.D., D.W. Corne. 2001a. Benchmark problem generators and results for the multiobjective degree-constrained minimum spanning tree problem. *Proceedings of the Genetic and Evolutionary Computation Conference GECCO2001*. Morgan Kaufmann Publishers, 424–431.

Knowles, J.D., D.W. Corne. 2001b. A comparison of encodings and algorithms for multiobjective spanning tree problems. *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*. 544–551.

Marcotte, O., R.M. Soland. 1986. An interactive branch-and-bound algorithm for multiple criteria optimization. *Management Science* **32** 61–75.

Martin, P.D., D.B. Shmoys. 1996. A new approach to computing optimal schedules for the job shop scheduling problem. S.T. McCormick W.H. Curnigham, M. Queyranne, eds., *Proceedings fifth international IPCO conference, Vancouver, Canada*. LNCS 1084, 389–403.

Mavrotas, G., D. Diakoulaki. 1998. A branch and bound algorithm for mixed zero-one multiple objective linear programming. *European Journal of Operational Research* **107** 530–541.

Murthy, I., S. Sarkar. 1998. Stochastic shortest path problems with piecewise-linear concave utility functions. *Management Science* **44** 125–136.

Neumann, F. 2006. Expected runtimes of a simple evolutionary algorithm for the multi-objective minimum spanning tree problem. *European Journal of Operational Research* Doi:10.1016/j.ejor.2006.08.005.

Papadimitriou, C.H., M. Yannakakis. 2000. On the approximability of trade-offs and optimal access of web sources. *IEEE Symposium on Foundations of Computer Science FOCS 2000*. Redondo Beach, California, USA, 86–92.

Ramos, R.M., S. Alonso, J. Sicilia, C. Gonzales. 1998. The problem of the optimal biobjective spanning tree. *European Journal of Operational Research* **111** 617–628.

Serafini, P. 1986. Some considerations about computational complexity for multiobjective combinatorial problems. J. Jahn, W. Krabs, eds., *Recent advances and historical development of vector optimization*, *Lecture Notes in Economics and Mathematical Systems*, vol. 294. Springer-Verlag, Berlin.

Steiner, S., T. Radzik. 2006. Computing all efficient solutions of the biobjective minimum spanning tree problem. *Computers and Operations Research* .

Tarjan, R.E. 1983. *Data structures and network algorithms*. Society for Industrial and Applied Mathematics.

Visée, M., J. Teghem, M. Pirlot, E.L. Ulungu. 1998. Two-phases method and branch and bound procedures to solve biobjective knapsack problem. *Journal of Global Optimization* **12** 139–155.

Zhou, G., M. Gen. 1999. Genetic algorithm approach on multi-criteria minimum spanning tree problem. *European Journal of Operational Research* **114** 141–152.