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

Francis Sourd, Olivier Spanjaard, Patrice Perny

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

Francis Sourd* Olivier Spanjaard* Patrice Perny*

*LIP6, University of Paris 6
8, rue du Capitaine Scott — F75015 Paris, France
FirstName.Name@lip6.fr

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. Quite surprisingly, as emphasized in [3], they have not been studied widely in a multi-objective setting, i.e. when each solution x in the set $\mathcal{S}$ of feasible solutions is evaluated by p objective functions $f(x) = (f_1(x), \ldots, f_p(x))$ to minimize (without loss of generality). Actually, the few existing papers on this topic concern mainly multi-objective integer linear programming [1, 6, 8, 9].

In multi-objective branch-and-bound procedures, one has to find the Pareto front of $\mathcal{S}$ (in fact one solution for each Pareto point in the objective space). Therefore, at each time of the search, one keeps the *set* of best solutions found so far instead of a single incumbent. Futhermore, unlike the single-objective case, there possibly exist several Pareto optimal solutions (more precisely, Pareto optimal solutions with *distinct* images in the objective space) that can be reached from a given node in the search tree. Hence, a natural extension of conventional branch-and-bound procedures should associate each node with a *set* of lower bounds. However, in previous works, the proposed branch-and-bound algorithms share the common property to use the ideal of a sub-problem as a lower bound for that sub-problem. This feature impacts badly on the efficiency of the approach since it often leads to loose bounds.

In the present work, we give a formal framework to design multiobjective branch-and-bound procedures, and we provide a generalization of the lower bounding concept, able to more tightly bound the Pareto front of a sub-problem. We tested our approach on the bi-objective spanning tree problem. It significantly improves the existing results for the problem.

2 Multi-objective branch-and-bound

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

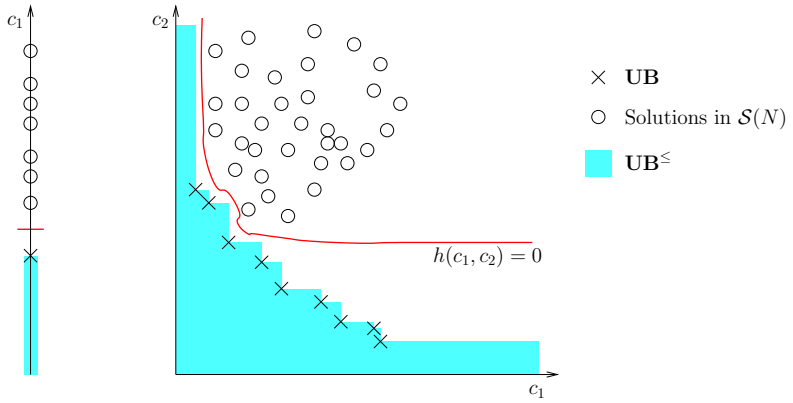

Figure 1: Bounding phase

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** the set of the 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.

The multi-objective branch-and-bound method is identical to the classical branch-and-bound in the branching part but differs in the bounding part. Therefore the branching scheme must be able to enumerate in a search tree all the feasible —possibly Pareto optimal— solutions of $\mathcal{S}$. When a new feasible solution x is found at some node, it is included to **UB** if it is not dominated by any $y \in \mathbf{UB}$ and, conversely, all the solutions $u \in \mathbf{UB}$ such that $f(x) \leq u$ are removed from **UB**. The pure enumerative approach clearly find all the Pareto front but would be computationally impracticable to solve problems of 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 sub-tree 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. Let us denote by $\mathbf{UB}^{\leq}$ the set of points in the objective space that are not dominated by any current point of **UB**, that is $\mathbf{UB}^{\leq} = \{v \in \mathbb{R}^p | \forall u \in \mathbf{UB}, u \not\leq v\}$. In the single-objective case, $\mathbf{UB}^{\leq} = (-\infty, \mathbf{UB}]$. Let us also denote by $\mathcal{S}(N)$, the set of feasible solutions which are enumerated in the subtree of the current node N.

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 $\mathbf{UB}^{\leq}$ and $\mathcal{S}(N)$, that is a function $h(c_1, \ldots, c_p)$ such that $h(f(x)) \geq 0$ for all $x \in \mathcal{S}(N)$ and $h(v) < 0$ for all $v \in \mathbf{UB}^{\leq}$. In the single-objective case, the separating function is simply $h(c) = c - \mathbf{LB}(N)$ where $\mathbf{LB}(N)$ is the usual lower bound for $f(\mathcal{S}(N))$. In the multi-objective case, a linear h function cannot generally separate $f(\mathcal{S}(N))$ from **UB** because **UB** is far from being convex. A general family

of good separating functions can be defined as

$$h_\Lambda(c_1, \ldots, c_p) = \min_{\lambda \in \Lambda} \langle \lambda, c \rangle - \mathbf{LB}_\lambda(N)$$

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 $\langle \lambda, f \rangle(\mathcal{S}(N))$. Clearly, the larger Λ is, the better the separating function becomes. However, it also becomes more complex and longer to compute. In general, h_Λ is convex and piecewise linear and its graph has at most $|\Lambda|$ facets.

Conversely, an implementation of MOBB should also implement a computationally tractable representation of $\mathbf{UB}^{\leq}$, which may possibly be approximate. In practice, we can consider a finite set $\mathcal{N}$ of vectors in $\mathbb{R}^p$ such that, for any $v \in \mathbf{UB}^{\leq}$, we have $v \leq w$ for some $w \in \mathcal{N}$. The set $\mathcal{N}$ can 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 $\mathbf{UB}$. Once again, a large $|\mathcal{N}|$ will help in having a better approximation of $\mathbf{UB}^{\leq}$ but requires a greater computational effort.

Finally, a sufficient condition to discard the current node N is that, for all $w \in \mathcal{N}$, we have $h_\Lambda(w) < 0$ if we want to enumerate all the optima in the solution space or $h_\Lambda(w) \leq 0$ if we want the optima in the objective space. 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.

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 graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges, where each edge e is valued by a vector (w_1^e, w_2^e), find the set of Pareto optimal spanning trees of G. This problem has been introduced in [2], and proved NP-hard in [4, 5]. Several approximation methods have been proposed [5, 13, 7]. To the best of our knowledge, only two operational exact methods [10, 11] have been proposed until now, both based on a two-phases (exact) procedure [12], first calculating the set of *extreme* Pareto optimal solutions (i.e., vertices of the convex hull of all solutions in the objective space) and second the set of non-extreme Pareto optimal solutions located in the triangles generated in the objective space by two successive extreme solutions[1]. The methods mainly differ in the way they compute Pareto-optimal solutions in the triangles: in [10] these solutions are computed by a branch-and-bound discarding any node such that an ad-hoc bounding point (that seems to be dominated by the ideal point of the corresponding sub-problem) falls outside the triangles; in [11] 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 in [11] and the latter is shown to run significantly faster.

In our approach, $\mathbf{UB}$ is also initialized by a two-phases (approximation) procedure, similar to the one used in [5]: first, the extreme solutions are computed and second, local search (starting with the extreme solution) is launched. The branching scheme is very simple: at each node, an edge e of G is selected (according to a heuristic we do not detail here) and we

[1] Strictly speaking, a two-phases 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.

create two subproblems. In the first one, the edge e must belong to the spanning tree while in the second one, the edge is removed from G.

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. Furthermore, we have proved the following:

Lemma 1. *Let us consider the m pairs (α_i, β_i) for $1 \leq i \leq m$. Let $S(\lambda)$ be the sequence of the 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 $m(m-1)/2$ different sequences.*

As Kruskal's algorithm —used to compute the extreme spanning trees— is based on the order of the edges according to their weighted cost $\lambda w_1^e + (1-\lambda)w_2^e$, a corollary is that there are at most $m(m-1)/2$ extreme spanning trees of distinct costs and therefore h_Λ can be computed in polynomial time. As a second consequence of this lemma, we have that all the possible sequences 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. Moreover, the pre-sorting can be done in only $O(m^3)$ time: in general the time spent in this phase is easily balanced by the time spared during the search phase.

Computing $\mathcal{N}$ is easy when there are only two objectives. Indeed, let $\{(u_1^i, u_2^i) \,|\, 1 \leq i \leq k\}$ be the points of **UB** which are maintained in the lexicographical order. Then, we can define $\mathcal{N} = \{(u_1^{i+1}, u_2^i) \,|\, 1 \leq i < k\}$ in order to have a strict covering of $\mathbf{UB}^{\leq}$, that is $v \leq w \in \mathcal{N}$ implies that $v \in \mathbf{UB}^{\leq}$. When a point $w \in \mathcal{N}$ satisfies $h_\Lambda(w) > 0$ then the node N is not fathomed but, before branching, the algorithm checks whether a newly computed supported point of $f(\mathcal{S}(N))$ can be inserted into **UB**.

4 Experimental results

The branch-and-bound algorithm has been implement in C# and was run on a 1.6 GHz personal computer. Table 1 shows computation times obtained on complete graphs with costs randomly drawn in [0,100]. Instances with 150 nodes are easily solved, while the approach of [11] can only solve problems with 25 nodes on the same class of instances.

n	Solutions	Nodes	LS	Sort	Search	Total
25	250	7729	0.13	0.03	0.26	0.47
50	639	38504	1.31	0.92	4.10	6.42
75	899	60101	4.39	6.63	15.02	26.32
100	1168	76147	11.15	28.35	39.63	79.65
125	1343	70189	20.61	81.48	66.38	169.46
150	1512	66052	35.71	169.31	169.31	401.71

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

References

[1] Bitran, G. and Rivera, J.M. (1982): "A combined approach to solve binary multicriteria problems". In: *Naval Research Logistics Quarterly*. **29**, 181–201.

[2] Corley, H.W. (1985): "Efficient spanning trees". In: *Journal of optimization theory and applications*. **45** (3), 481–485.

[3] Ehrgott, M. and Gandibleux, X. (2000). "A survey and annoted bibliography of multiobjective combinatorial optimization". In: *OR Spektrum*. **22**, 425–460.

[4] Emelichev, V.A. and Perepelitsa, V.A. (1988): "Multiobjective problems on the spanning trees of a graph". In: *Soviet Mathematics Doklady*. **37** (1), 114–117.

[5] Hamacher, H.W. and Ruhe, G. (1994): "On spanning tree problems with multiple objectives". In: *Annals of Operations Research*. **52**, 209–230.

[6] Kiziltan, G. and Yucaoglu, E. (1983): "An algorithm for multiobjective zero-one linear programming". In: *Management Science*. **29** (12), 1444–1453.

[7] Knowles, J.D. and Corne, D.W. (2001): "A Comparison of Encodings and Algorithms for Multiobjective Spanning Tree Problems". In: *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, 544–551.

[8] Marcotte, O. and Soland, R.M. (1986): "An interactive branch-and-bound algorithm for multiple criteria optimization". In: *Management Science*. **32** (1), 61–75.

[9] Mavrotas, G. and Diakoulaki, D. (1998): "A branch and bound algorithm for mixed zero-one multiple objective linear programming". In: *European Journal of Operational Research*. **107**, 530–541.

[10] Ramos, R.M. and Alonso, S. and Sicilia, J. and Gonzales C. (1998): "The problem of the optimal biobjective spanning tree". In: *European Journal of Operational Research*. **111**, 617–628.

[11] Steiner, S. and Radzik, T. (2003): "Solving the biobjective minimum spanning tree problem using a k-best algorithm". Technical Report [TR-03-06]. Department of Computer Science, King's College London.

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

[13] Zhou, G. and Gen, M. (1999): "Genetic algorithm approach on multi-criteria minimum spanning tree problem". In: *European Journal of Operational Research*. **114**, 141–152.