The Bi-Objective Pareto Constraint

Renaud Hartert* and Pierre Schaus

UCLouvain, ICTEAM,
Place Sainte Barbe 2,
1348 Louvain-la-Neuve, Belgium
{renaud.hartert,pierre.schaus}@uclouvain.be

Abstract. Multi-Objective Combinatorial Optimization (MOCO) problems are ubiquitous in real-world applications. Gavanelli proposed a complete constraint programming approach to find the exact set of optimal solutions – known as efficient solutions – of MOCO problems. This approach has recently been extended in a new global constraint called the `Pareto` constraint. In this paper, we bring some complementary information on the `Pareto` constraint. Particularly, we discuss its efficiency for generic MOCO problems and present two ways of improving this efficiency for bi-objective combinatorial optimization problems.

Keywords: constraint programming, multi-objective combinatorial optimization, bi-objective combinatorial optimization, global constraint.

Despite the increasing interest in the field of Multi-Objective Combinatorial Optimization (MOCO) during the last decades, only a small number of approaches have been introduced to tackle MOCO problems with Constraint Programming (CP) [2]. Among them, Gavanelli [3] proposed a dedicated branch-and-bound algorithm to find the exact set of efficient solutions in a single exploration of the search tree.

The `Pareto` constraint (briefly introduced in [6]), extends the original idea of Gavanelli in a more flexible way. In this paper, we complete our brief introduction of the `Pareto` constraint with a detailed formalization. Furthermore, we discuss the complexity of its general case and show how it can be improved for bi-objective combinatorial optimization problems.

Outline. In Section 1, we introduce the definitions and concepts necessary to the understanding of this work. Section 2 presents the original approach of Gavanelli. Then, we formalize the `Pareto` constraint in Section 3. Section 4 describes two ways of improving the efficiency of the `Pareto` constraint to tackle bi-objective combinatorial optimization problems. Finally, we conclude this paper in Section 5 by discussing future work on the `Pareto` constraint.

1 Multi-Objective Combinatorial Optimization

The following definitions, adapted from [1], introduce Multi-Objective Combinatorial Optimization (MOCO) and the class of problems we are trying to solve.

* MSc student.

Definition 1 (MOCO problem). *A MOCO problem $\mathcal{P}$ is a tuple $\langle X, D, C, f_1, \ldots, f_m \rangle$ where X is a set of variables, D is the set of the domains, C is a set of constraints, and $f_1, \ldots, f_m$ are integer objective functions to minimize. We assume that m objective variables $obj_1, \ldots, obj_m$ have been added to the set of variables X and constrained to be equal to their corresponding objective function i.e. $obj_i = f_i(X)$.*

Definition 2 (Solution). *A solution is a complete assignment of the variables of $\mathcal{P}$ that satisfies the constraints of this problem. In the following, we represent a solution $sol = (sol_1, \ldots, sol_m)$ as the vector of the values assigned to its objective variables.*

As it is not likely that a solution is simultaneously optimal for all objectives, one is generally interested in the set of all the *efficient solutions* i.e. all the optimal compromises between the objectives.

Definition 3 (Weak Pareto dominance). *Let sol and sol' be two solutions of a MOCO problem $\mathcal{P}$. We say that sol dominates sol', denoted $sol \preceq sol'$, if and only if:*

$$\forall i \in \{1, \ldots, m\} : sol_i \leq sol'_i. \tag{1}$$

Definition 4 (Efficient solution). *Let $sols(\mathcal{P})$ denote the set of all the feasible solutions of a MOCO problem $\mathcal{P}$. A solution sol of a MOCO problem $\mathcal{P}$ is efficient if and only if there is no solution sol' in $sols(\mathcal{P})$ that dominates sol:*

$$\nexists sol' \in sols(\mathcal{P}) : sol' \preceq sol. \tag{2}$$

Definition 5 (Efficient set). *The efficient set of a MOCO problem $\mathcal{P}$ is the set of all the efficient solutions of the problem:*

$$\{sol \in sols(\mathcal{P}) \mid \nexists sol' \in sols(\mathcal{P}) : sol' \preceq sol\}. \tag{3}$$

In practice, it is difficult to find the exact efficient set of challenging MOCO problems [1]. We are thus interested in finding a good approximation of the efficient set, also known as an *archive*. It is formalized as follows:

Definition 6 (Archive). *An archive $\mathcal{A}$ is a set of solutions such that there is no solution in the archive that dominates an other solution in the archive. This property is known as the domination-free property:*

$$\forall sol \in \mathcal{A}, \nexists sol' \in \mathcal{A} : sol' \preceq sol. \tag{4}$$

An archive may contain non-efficient solutions as well as efficient solutions.

2 Related work

In [3], Gavanelli proposed a branch-and-bound algorithm to find the exact efficient set of MOCO problems. This algorithm makes use of the previously discovered solutions (i.e. the solutions contained in an archive $\mathcal{A}$) to prune branches of the search tree that cannot lead to an efficient solution. In other words, this algorithm ensures that each newly discovered solution is nondominated by any solution in the archive:

$$\nexists sol \in \mathcal{A} : sol \preceq (obj_1, \ldots, obj_m). \tag{5}$$

Particularly, each time a new solution is discovered, it is inserted into the archive $\mathcal{A}$ to tighten the bounds of the objective variables.[1]

[1] According to the domination-free property, dominated solutions are removed from the archive. Hence, discovering a new solution could reduce the size of the archive.

Definition 7 (Dominated point). *Let $obj_i^{\min}$ and $obj_i^{\max}$ denote the lower and upper bounds of the objective variable obj_i, the dominated point DP_i is defined as follows:*

$$DP_i = (obj_1^{\min}, \ldots, obj_{i-1}^{\min}, obj_i^{\max}, obj_{i+1}^{\min}, \ldots, obj_m^{\min}). \tag{6}$$

Observe that DP_i dominates all the solutions sol contained in the Cartesian product of the domain of the objective variables such that $sol_i \geq obj_i^{\max}$. Hence, if the dominated point DP_i is dominated by a solution in the archive, one can use this solution to adjust the upper bound of the objective variable obj_i:

$$\exists sol \in \mathcal{A}, sol \preceq DP_i \ : \ obj_i^{\max} \leftarrow sol_i - 1. \tag{7}$$

The left part of Fig. 1 illustrates the filtering of the bounds of the objective variables on an arbitrary archive. If a dominated point DP_i is dominated by several solutions at the same time, the filtering rule of Equation 7 has to be called until no solution in the archive dominates DP_i. Clearly, the order in which the dominating solutions are selected affects the number of calls of the filtering rule. This situation is illustrated in the right part of Fig. 1 where a, b, c is the worst possible selection order.

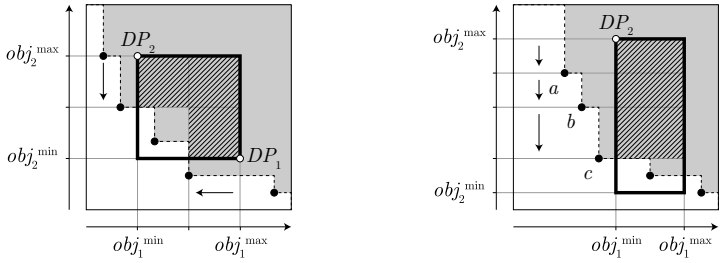

Fig. 1. Filtering of the bounds of the objective variables. Gray areas represent the subspaces of solutions that are dominated by the archive.

3 The Pareto Constraint

The `Pareto` constraint [6] is a global constraint defined over the set of m objective variables of a MOCO problem $\mathcal{P}$ and the solutions contained in an archive $\mathcal{A}$:

$$\texttt{Pareto}(obj_1, \ldots, obj_m, \mathcal{A}). \tag{8}$$

As for the approach of Gavanelli, the `Pareto` constraint guarantees that the archive $\mathcal{A}$ is equal to the efficient set of $\mathcal{P}$ once the search tree is entirely explored.

The aim of the `Pareto` constraint is to extend the approach of Gavanelli by taking advantage of the information available during propagation and offering more flexibility by not requiring the implementation of a dedicated branch-and-bound algorithm.

3.1 Filtering and efficiency

From Section 2, it appears that the `Pareto` constraint can reach its fix point in one step if it is able to access directly the solution that dominates DP_i with the lowest value for

objective i (we call this solution the *tightest solution* in objective i). We formalize this improved filtering rule as follows:

$$obj_i^{\max} \leftarrow \min(\{obj_i^{\max}\} \cup \{sol_i - 1 \mid sol \in \mathcal{A},\ sol \preceq DP_i\}) \tag{9}$$

Clearly, the time needed to access the tightest solutions is the bottle-neck of the `Pareto` constraint and is strongly related to the data structure used to implement the archive. In [3], Gavanelli suggested the use of domination-free quad-trees[2] (simply quad-trees in the sequel) to implement the archive. However, quad-trees suffer from several weaknesses:

- As for binary search tree, a quad-tree is not auto-balanced and is strongly influenced by the order in which solutions are inserted in its structure. The worst possible case being a quad-tree structured as a linked-list i.e. only one son by node. Hence, the theoretical worst case complexity of accessing a solution is larger than $\mathcal{O}(\log n)$.[3]
- Removing nodes in the structure of a quad-tree (e.g. to maintain the domination-free property) often yield to important computational costs [4].

The next section is an attempt to provide more efficient methods to access solutions in the archive when applied on bi-objective combinatorial optimization problems.

4 Efficient Bi-Objective Implementations

According to Definitions 3 and 4, bi-objective problems have the particularity that improving the first objective of a Pareto optimal solution cannot be done without degrading the value of the second objective (and vice-versa). Hence, sorting the solutions of a bi-objective problem in increasing order w.r.t. one objective amounts to sort these solutions in decreasing order w.r.t. the other objective. We call this property the *ordering property* of bi-objective problems. In the following, we denote $\mathcal{A}^{\gg_1}$ (resp. $\mathcal{A}^{\gg_2}$) the archive $\mathcal{A}$ ordered by decreasing value of obj_1 (resp. obj_2).

We introduce two possible uses of the ordering property to implement efficiently the idempotent filtering rule (Equation 9) of the `Pareto` constraint when considering bi-objective combinatorial optimization problems.

4.1 Balanced Linked-Tree

A balanced linked-tree (or braided balanced trees [5]) is an ordered linked-list and a balanced binary tree at the same time. Balanced linked-trees ensure a worst case time complexity of $\mathcal{O}(\log n)$ for operations as access, insertion and deletion while allowing to access the successor and the predecessor of a given element in constant time.

Let T_1 be a binary linked-tree containing all the solutions of a bi-objective archive where sol_1 is the key value of a solution sol. The following algorithm is able to access the tightest solution of obj_2 (if it exists) within a time complexity of $\mathcal{O}(\log n)$. The idea consists in finding the position of the key $obj_1^{\min}$ in T_1. If the tree is not empty, one of the three following situations has to be considered:[4]

[2] Quad-trees can be seen as the extension of binary search trees to m dimensional data. We refer to Habenicht [4] for more information about quad-trees and their efficiency.

[3] This affirmation is particularly true when considering more than two dimensions since more than one branch could be explored at each node when searching for a specific element [4].

[4] The tightest solution of obj_1 can be accessed similarly in T_2.

1. If T_1 already contains a solution sol with $obj_1^{\min}$ as key value, then, sol is the tightest solution of obj_2.
2. If $obj_1^{\min}$ has to be inserted in the left branch of a solution sol, then, the direct successor of sol in $\mathcal{A}^{>_1}$ is the tightest solution of obj_2.
3. If $obj_1^{\min}$ has to be inserted in the right branch of a solution sol, then, sol is the tightest solution of obj_2.

Fig. 2 illustrates the access of the tightest solution of both objective.

Clearly, accessing the tightest solution of an objective is performed within a time complexity of $\Theta(\log n)$. The insertion of a new solution sol^{new} has a time complexity of $\Theta(k \log n)$ where k is the number of solutions in the archive that are dominated by sol^{new}. These complexities are reported in Table 1.

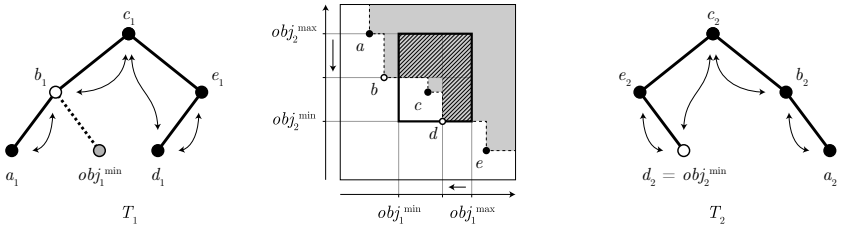

Fig. 2. Access of the tightest solutions (i.e. b and d) with balanced linked-trees.

4.2 Reversible Ordered Linked-List

An alternative to the balanced linked-tree based implementation is to exploit the ordering property of bi-objective problems to maintain the tightest solution of each objective incrementally during the exploration of the search tree. Here, the archive is implemented as an ordered linked-list following the bi-objective ordering property.

Definition 8 (Support). *Let sol be a solution in a bi-objective archive $\mathcal{A}$. We say that sol is the support of obj_1 (resp. obj_2) if and only if:*

$$sol = \text{argmax}_{sol' \in \mathcal{A}}\{sol'_2 \mid sol'_2 < obj_2^{\min}\}. \tag{10}$$

Proposition 1. *Supports are never included in the Cartesian product of the domain of the objective variables i.e. supports cannot be dominated by newly discovered solutions.*

Proposition 2. *If it exists, the tightest solution of obj_i is its support or its direct successor in $\mathcal{A}^{>_i}$.*

Proof. Propositions 1 and 2 are trivially derived from Definitions 3 and 8. □

Proposition 3. *Let $sol^{\text{new}} = (obj_1^{\min}, obj_2^{\min})$ be a newly discovered solution. If the archive is ordered according to the value of one objective (i.e. $\mathcal{A}^{>_1}$ or $\mathcal{A}^{>_2}$), then, sol^{new} dominates all the solutions contained between the supports.*

Proof. Let sol^{sup} be the support of obj_i, and let S_i denote the set of all the successors of sol^{sup} in $\mathcal{A}^{>_i}$. By Definition 8, we know that $S_i = \{sol \in \mathcal{A} \mid sol_i \geq obj_i^{\min}\}$. Since $sol^{\text{new}} = (obj_1^{\min}, obj_2^{\min})$, the solutions contained in the intersection of the successors of both supports are dominated by sol^{new}:

$$\forall sol \in S_1 \cap S_2 = \{sol' \in \mathcal{A} \mid sol'_1 \geq obj_1^{\min} \wedge sol'_2 \geq obj_2^{\min}\} : sol^{\text{new}} \preceq sol. \quad \square$$

We describe now a second algorithm to adjust the upper bound of obj_1 (resp. obj_2) based on Propositions 1, 2, and 3:

1. Each time the lower bound of obj_2 is adjusted, we have to reconsider the support sol^{sup} of obj_1. To do so, we iterate on the direct successors of sol^{sup} in $\mathcal{A}^{>_1}$ until we reach a new support. Let Δ denote this number of iterations. Clearly, the sum of the Δ cannot exceed the size of the archive along a branch of the search tree.
2. When the new support is found, we use Proposition 2 to apply the idempotent filtering rule from Equation 9 to adjust the upper bound of obj_1.
3. To insert a new solution sol^{new} in $\mathcal{A}$, we simply have to replace the sublist contained between the supports of both objectives by sol^{new} (see Proposition 3). This operation is performed in constant time.

Assuming a trailed based CP solver, *reversible pointers* can be used to maintain the supports. Thus, each time a backtrack occurs, the bi-objective `Pareto` constraint is able to recover its previous supports in constant time (see Table 1).

Table 1. Best and worst time complexity of the bi-objective `Pareto` constraints to access the tightest solutions and to insert a new solution in an archive of n solutions.

Structure	Access (Θ)	Insertion (Θ)
Linked-list	n	n
Balanced Linked-Tree	$\log n$	$k \log n$
Reversible Ordered Linked-List	Δ	1

5 Conclusion

We have detailed and formalized the `Pareto` constraint while pointing out the importance of the underlying data structure in the efficiency of the filtering algorithm. Furthermore, we have presented two different ways of taking advantage of the ordering property to improve the efficiency of the `Pareto` constraint on bi-objective combinatorial optimization problems. As future work, we would like to compare the efficiency of our bi-objective approaches on challenging MOCO problems.

References

1. Matthias Ehrgott. *Multicriteria optimization*, volume 2. Springer Berlin, 2005.
2. Matthias Ehrgott and Xavier Gandibleux. Hybrid metaheuristics for multi-objective combinatorial optimization. *Hybrid metaheuristics*, pages 221–259, 2008.
3. M. Gavanelli. An algorithm for multi-criteria optimization in CSPs. *ECAI*, 2:136–140, 2002.
4. W Habenicht. Quad trees, a datastructure for discrete vector optimization problems. In *Essays and Surveys on Multiple Criteria Decision Making*, pages 136–145. Springer, 1983.
5. Stephen V Rice. Braided AVL trees for efficient event sets and ranked sets in the SIMSCRIPT III simulation programming language. In *Proceedings of the 2007 Western Multiconference on Computer Simulation*, pages 150–155, 2007.
6. Pierre Schaus and Renaud Hartert. Multi-Objective Large Neighborhood Search. In *19th International Conference on Principles and Practice of Constraint Programming*, 2013.