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 when applied on standard MOCO problems and present two ways of improving this efficiency when applied on bi-objective combinatorial optimization problems.

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

1 Introduction

Since the pioneering work of Pareto [8], many progresses were done in the field of Multi-Objective Combinatorial Optimization (MOCO). However, during the last decades, only a small number of approaches have been introduced to tackle MOCO problems with constraint programming (CP) [3]. Among them, Gavanelli [5] proposed a dedicated branch-and-bound algorithm to find the exact set of efficient solutions in a single constraint programming search.

The `Pareto` constraint (briefly introduced in [11]), extends the original idea of Gavanelli in a more time efficient and flexible way. In this paper, we complete our brief introduction of the `Pareto` constraint with a detailed formalization. Besides, we discuss the complexity of its general case and show how it can be improved when applied on bi-objective combinatorial optimization problems.

Outline. We introduce the definitions and concepts necessary to the understanding of this work in Section 2. We introduce the `Pareto` constraint in its general form in Section 3. Then, we describe in Section 4 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.

* MSc student.

2 Multi-Objective Combinatorial Optimization

The typical MOCO problem we want to solve has m finite integer objective variables to minimize while satisfying a set of constraints:

$$\begin{aligned} &\text{Minimize} \quad obj = (obj_1, obj_2, \ldots, obj_m) \\ &\text{Subject to} \quad constraints \end{aligned} \tag{1}$$

Solutions of this problem are defined as follows:

Definition 1 (Solution). *Let $\mathcal{P}$ be a MOCO problem, a solution of the problem $\mathcal{P}$ is a complete assignment of the decision variables and objective variables of $\mathcal{P}$ that satisfies all the constraint of this problem. In the following, we represent a solution sol by the vector of its objective values $sol = (sol_1, \ldots, sol_m)$.*

As it is not likely that a solution is simultaneously optimal in all objectives, one is generally interested in all the optimal compromises known as *efficient solutions*.

Definition 2 (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:*

$$\forall i \in \{1, \ldots, m\} : sol_i \leq sol'_i. \tag{2}$$

Definition 3 (Efficient solution). *Let $sols(\mathcal{P})$ denote the set of all the feasible solutions of a MOCO problem $\mathcal{P}$. A solution sol is efficient if and only if there is no solution sol' in $sols(\mathcal{P})$ that dominates sol:*

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

In other words, a solution is efficient if it is impossible to improve the value of one objective without degrading the value of at least one other objective.

The set of all the efficient solutions is called the *efficient set* of the problem and is defined as follows:

Definition 4 (Efficient set). *The efficient set of a MOCO problem $\mathcal{P}$ is the set of all the efficient solutions of the problem:*

$$\{sol \in sols(\mathcal{P}) \mid \nexists sol' \in sols(\mathcal{P}) : sol' \preceq sol\}. \tag{4}$$

Unfortunately, discovering the exact efficient set of difficult MOCO problems may be impracticable. We are thus interested in finding a good approximation of this set, also known as the *archive*. It is formalized as follows:

Definition 5 (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:*

$$\forall sol \in \mathcal{A}, \nexists sol' \in \mathcal{A} : sol' \preceq sol. \tag{5}$$

3 The Pareto Constraint

The `Pareto` constraint is a global constraint defined over the set of objective variables of a MOCO problem $\mathcal{P}$ and the solutions contained in a domination-free archive $\mathcal{A}$ of current size n:

$$\texttt{Pareto}(obj_1, \ldots, obj_m, \mathcal{A}) \tag{6}$$

where obj_i is an objective variable and sol_i is a feasible solution of $\mathcal{P}$.

The aim of the `Pareto` constraint is to prune branches of the search tree that are dominated by the previously discovered solutions i.e. the solutions contained in the archive $\mathcal{A}$. In other words, the `Pareto` constraint ensures that each new discovered solution is nondominated w.r.t. the archive:

$$\nexists sol \in \mathcal{A} : sol \preceq (obj_1, \ldots, obj_m). \tag{7}$$

Particularly, each time a new solution is discovered, it is inserted into the archive $\mathcal{A}$ to strengthen the filtering of the `Pareto` constraint.[1] This approach can be seen as a specialized branch-and-bound algorithm for MOCO.

3.1 Filtering

The filtering rule of the `Pareto` constraint aims at reducing the upper bound of each objective variable. Let $obj_i^{\min}$ and $obj_i^{\max}$ denote the lower and upper bounds of the objective variable obj_i, we define the dominated point DP_i of objective i has follows:

$$DP_i = (obj_1^{\min}, \ldots, obj_{i-1}^{\min}, obj_i^{\max}, obj_{i+1}^{\min}, \ldots, obj_m^{\min}). \tag{8}$$

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, we can use this solution to adjust the upper bound of the objective variable obj_i:

$$\exists sol \in \mathcal{A}, sol \preceq DP_i \ : \ obj_i^{\max} \leftarrow sol_i - 1. \tag{9}$$

This filtering rule (due to [5]) has to be executed until there exists no solution in the archive that dominates the dominated point. Fig. 1 illustrates this situation where the dominating solutions are selected in the worst possible order.

From the observation made in Fig. 1, it appears that the `Pareto` constraint can reach its fix point in one step (i.e. the constraint becomes idempotent) 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). The filtering rule thus becomes:

$$obj_i^{\max} \leftarrow \min(\{obj_i^{\max}\} \cup \{sol_i - 1 \mid sol \in \mathcal{A},\ sol \preceq DP_i\}) \tag{10}$$

[1] According to the domination-free property, solutions that are dominated by a new inserted solution are removed from the archive.

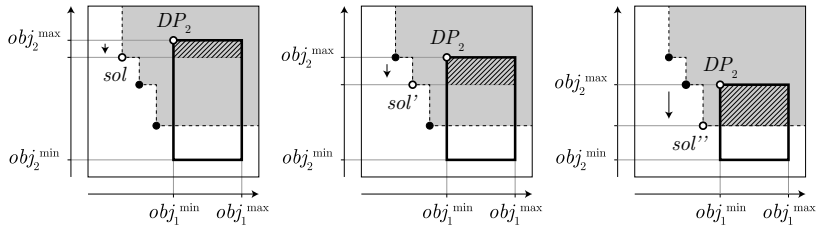

Fig. 1. A dominated point DP_i may be dominated by several solutions. The order in which the dominating solutions are selected affects the number of iterations to reach the fix point of the `Pareto` constraint.

3.2 Efficiency

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 store the archive. In [5], Gavanelli suggested to use domination-free quad-trees [6,12,13] (simply quad-trees in the following) to store the archive. However, we do not encourage the use of quad-trees for the following reasons:

- To the best of our knowledge, there is no efficient algorithm to access the tightest solutions of an archive implemented with a quad-tree;
- Despite the existence of algorithms [4,10] to reduce its size, a quad-tree is not auto-balanced. Hence, the efficiency of a quad-tree is strongly influenced by the order in which solutions are inserted in its structure;
- Since quad-trees are not auto-balanced, the worst time complexity of finding a dominating solution in a quad-tree is worse than $\mathcal{O}(\log n)$.[2] This is especially true when considering more than two objectives [6].

Further researches have to be conducted to provide the `Pareto` constraint with an efficient data structure. Meanwhile, we recommend (when considering more than two objectives) the use of linked-lists which allow to find the tightest solutions in a single traversal of the archive.[3]

4 Efficient Bi-Objective Implementations

According to Definitions 2 and 3, 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.

We introduce two possible uses of the ordering property to implement efficiently the idempotent filtering rule (Equation 10) of the `Pareto` constraint when considering bi-objective combinatorial optimization problems.

[2] This affirmation contradicts the claim of Gavanelli [5].

[3] Standard implementations of quad-trees and linked-lists are compared in [7].

4.1 Balanced Linked-Tree

A balanced linked-tree (or braided balanced trees [9]) is an ordered linked-list and a balanced binary tree [1,2] at a same time. Balanced linked-trees ensure a worst 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 BLT_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 BLT_1. If the tree is not empty, one of the three following situations has to be considered:[4]

1. If BLT_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 objective 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 (see Fig. 2).

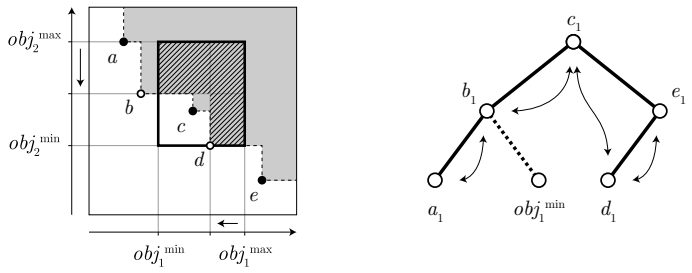

Fig. 2. Balanced linked-trees allow to access the tightest solutions in a worst time complexity of $\mathcal{O}(\log n)$. In this example, d is the tightest solution of objective 2.

As mentioned in Section 3, each new solution sol^{new} is inserted into the archive to strengthen the filtering of the `Pareto` constraint. Using a balanced linked-tree based archive, the insertion of sol^{new} has a worst time complexity of $\mathcal{O}(k \log n)$ where k is the number of solutions in the archive that are dominated by sol^{new} (see Table 1).

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.

Definition 6 (Support solution). *Let sol be a solution in a bi-objective archive $\mathcal{A}$. We say that sol is the support solution of obj_1 (resp. obj_2) if and only if:*

$$sol = \min_{sol' \in \mathcal{A}} \{sol'_1 \mid sol'_2 < obj_2^{\min}\} \tag{11}$$

[4] The tightest solution of obj_1 can be accessed similarly in BLT_2.

Proposition 1. *Support solutions are never included in the Cartesian product of the domain of the objective variables. Hence, support solutions cannot be dominated be a new discovered solution.*

Proposition 2. *If it exists, the tightest solution of an objective i is its support solution or the direct successor in objective i of the support solution.*

Let us consider the left part of Fig. 2 to illustrate these propositions. Solutions e and b are the supports of obj_1 and obj_2 respectively and are not contained in the Cartesian product of the domain of the objective variables (Proposition 1). Besides, d and b are the tightest solutions of obj_1 and obj_2 respectively and can thus be used to adjust the upper bounds of the objective variables (Proposition 2).

We describe now a second algorithm to adjust the upper bound of obj_1 (resp. obj_2) based on support solutions. The idea is as follows:

1. Each time the lower bound of obj_2 is adjusted, we have to reconsider the support of obj_1. To do so, we iterate on the direct successors in objective 1 of the old support until reaching a new support solution. Let Δ denote this number of iterations. Clearly, the sum of the Δ cannot exceed n along a branch of the search tree.
2. Then, we use Proposition 2 to apply the idempotent filtering rule from Equation 10 to adjust the upper bound of obj_1.

Assuming a trailed based CP solver, *reversible pointers* can be used to maintain the support solutions. Thus, each time a backtrack occurs, the bi-objective `Pareto` constraint is able to recover its previous support solutions in constant time. The time complexity of this approach is reported in Table 1. Observe that this approach can also be used with a balanced linked-tree as ordered list.

Table 1. Best and worst time complexity of the bi-objective `Pareto` constraints when filtering the objective variables and inserting a new solution in an archive of n solutions.

Algorithms	Filtering		Insertion	
	Ω	$\mathcal{O}$	Ω	$\mathcal{O}$
Linked-list	n	n	n	n
Balanced Linked-Tree (BLT)	$\log n$	$\log n$	$\log n$	$k \log n$
Reversible with ordered list	1	Δ	1	n
Reversible with BLT	1	Δ	$\log n$	$k \log n$

5 Conclusion

We have detailed and formalized the `Pareto` constraint while pointing out the importance of the chosen underlying data structure in the efficiency of the filtering algorithm. Besides, we have presented two different ways of taking advantage of the ordering property to improve the efficiency of the `Pareto` constraint when applied on bi-objective combinatorial optimization problems.

As future work, we would like to compare the practical efficiency of our bi-objective approaches over general implementations of the `Pareto` constraint.

References

1. G. Adelson-Velskii and E. Landis. An algorithm for the organization of information. Technical report, DTIC Document, 1963.
2. Rudolf Bayer. Symmetric binary b-trees: Data structure and maintenance algorithms. *Acta informatica*, 1(4):290–306, 1972.
3. Matthias Ehrgott and Xavier Gandibleux. Hybrid metaheuristics for multi-objective combinatorial optimization. *Hybrid metaheuristics*, pages 221–259, 2008.
4. Raphael A. Finkel and Jon Louis Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.
5. M. Gavanelli. An algorithm for multi-criteria optimization in csps. *ECAI*, 2:136–140, 2002.
6. 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.
7. Sanaz Mostaghim and Jürgen Teich. Quad-trees: A data structure for storing pareto sets in multiobjective evolutionary algorithms with elitism. In *Evolutionary Multiobjective Optimization*, pages 81–104. Springer, 2005.
8. Vilfredo Pareto. Manual of political economy. 1927.
9. 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: International Conference on High Level Simulation Languages and Applications*, pages 150–155, 2007.
10. Hanan Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.
11. Pierre Schaus and Renaud Hartert. Multi-objective large neighborhood search. In *Proceedings of The 19th International Conference on Principles and Practice of Constraint Programming*, 2013.
12. Minghe Sun. A primogenitary linked quad tree data structure and its application to discrete multiple criteria optimization. *Annals of Operations Research*, 147(1):87–107, 2006.
13. Minghe Sun and Ralph E Steuer. Quad-trees and linear lists for identifying nondominated criterion vectors. *INFORMS Journal on Computing*, 8(4):367–375, 1996.