MDD Archive for Boosting the Pareto Constraint

Steve Malalel
Université Côte d'Azur, CNRS, I3S, Nice, France

Arnaud Malapert
Université Côte d'Azur, CNRS, I3S, Nice, France

Marie Pelleau
Université Côte d'Azur, CNRS, I3S, Nice, France

Jean-Charles Régin
Université Côte d'Azur, CNRS, I3S, Nice, France

Abstract

Multi-objective problems are frequent in the real world. In general they involve several incomparable objectives and the goal is to find a set of Pareto optimal solutions, i.e. solutions that are incomparable two by two. In order to better deal with these problems in CP the global constraint Pareto was developed by Schaus and Hartert to handle the relations between the objective variables and the current set of Pareto optimal solutions, called the archive. This constraint handles three operations: adding a new solution to the archive, removing solutions from the archive that are dominated by a new solution, and reducing the bounds of the objective variables. The complexity of these operations depends on the size of the archive. In this paper, we propose to use a multi-valued Decision Diagram (MDD) to represent the archive of Pareto optimal solutions. MDDs are a compressed representation of solution sets, which allows us to obtain a compressed and therefore smaller archive. We introduce several algorithms to implement the above operations on compressed archives with a complexity depending on the size of the archive. We show experimentally on bin packing and multi-knapsack problems the validity of our approach.

2012 ACM Subject Classification Applied computing → Multi-criterion optimization and decision-making; Theory of computation → Constraint and logic programming; Mathematics of computing → Decision diagrams

Keywords and phrases Constraint Programming, Global Constraint, MDD, Multi-Objective Problem, Pareto Constraint

Digital Object Identifier 10.4230/LIPIcs.CP.2023.24

Funding This work has been supported by the French government, through the 3IA Côte d'Azur Investments in the Future project managed by the National Research Agency (ANR) with the reference number ANR-19-P3IA-0002.

1 Introduction

Multi-objective combinatorial optimization (MOCO) problems are present in many industrial applications [13, 14]. They involve several incomparable objectives represented by objective variables.

For the sake of clarity and without loss of generality, we will consider that we have to solve a problem where all objective variables must be minimized.

A solution S_1 of objective variables is dominated by another solution S_2 if for each objective variable obj_i the value of obj_i in S_2 is better than or equal to the value of obj_i in S_1. For instance the solution $(4, 6, 3, 1)$ is dominated by $(4, 3, 2, 1)$ but it is not dominated by $(1, 1, 1, 3)$. The set of non dominated solutions defines the set of Pareto optimal solutions. In MOCO, the goal is to compute that set of Pareto optimal solutions. Usually the set of non

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).
Editor: Roland H. C. Yap; Article No. 24; pp. 24:1–24:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

dominated solutions are saved in an archive that is maintained during the search for solutions. Two operations are involved: *insert* that manages the addition of a new non dominated solution and *delete* that removes the solutions that are dominated by the new solution.

In addition, solvers dealing with MOCO problems have to deal with another question: Is a new solution dominated by an archive solution?

In Constraint Programming, we answer this question by avoiding the generation of dominated solutions. To do this, we add a constraint to the problem that ensures that no dominated solution can be computed. This added constraint is called Pareto constraint and was proposed by Schaus and Hartert [11, 4]. It implements the ideas of Gavanelli [3]. This constraint reduces the bounds of the objective variables such that dominated solutions cannot be produced. This result is obtained by preventing a new solution from being dominated by a solution from the archive. In order to understand this process, let us create a tuple composed of the current minimum of all objective variables. This tuple will dominate all solutions that can be constructed from the current objective variables. Thus, if there is a solution in the archive that dominates this tuple then clearly it will dominate all future solutions and so we can stop the search. Now, consider the objective variable obj_i. If we replace in our tuple the value of obj_i by its maximum possible value and if we found S, a solution of the archive, dominating this tuple, then obj_i must take a value less than or equal to that of S otherwise the future solutions will be dominated by S. By applying this process for each variable, Schaus and Hartert establish the bound consistency of the constraint. We will denote by *filter* this process. The complexity of this operation is not detailed in their paper. A simple implementation will require to traverse n (the number of objectives) times the archive. It is not straightforward to reach a time complexity linear in the size of the archive.

It may be tempting to use multi-valued decision diagrams (MDDs) for this constraint because MDDs are a compressive data structure for representing solution sets. Perez [7] defined the MDD representing the Pareto constraint, i.e. the set of tuples allowed by the constraint that are the possible future non-dominated solutions. As mentioned by Perez, this approach failed mainly because at the beginning the MDD compresses very strongly the set of solutions since everything is almost possible, then it will decompress because we only delete tuples and the chances of recompression are low.

In this article we propose to use MDDs to represent the archive, that are the current non dominated solutions. Currently the archive is often represented as lists. We can also use quad-trees but this is only efficient if we have few objectives [6], which is not our case of study. With lists (or quad-trees for that matter), the complexity of the operations *insert* and *delete* is linear with the size of the archive (i.e. the number of elements multiplied by the number of objectives). Representing the archive by an MDD will allow parts of common solutions to be merged. As with an MDD, all the solutions are treated globally, we can therefore hope to save time thanks to these groupings.

The operation *delete* will therefore potentially save time. The operation *insert* may be slower because MDDs are a heavier data structure than lists. However, this operation takes much less time than *delete* or *filter*. For this last operation, we will benefit from the global view of the MDD. We propose to improve the algorithm of Schaus and Hartet in two ways: we define an algorithm to find the tightest solutions (i.e. the largest possible value of an objective) faster and we introduce a variant of this algorithm that processes all objective variables at once, and not successively.

Our algorithms are based on the following idea: Consider the objective variable obj_i. Let us remove from the MDD that represents the archive all the values of the objective variables different from obj_i that are less than or equal to the minimum of their variable. Then we perform a reduction of the MDD in order to obtain MDD_D. If MDD_D is not empty then it means that there exist paths from root to tt in the MDD, that is solutions in the archive which will dominate any new solution involving some values of obj_i. More precisely, there exist solutions of the form $(v_1, v_2, \ldots, v_i, \ldots, v_n)$ such that $\forall j = 1 \ldots n, j \neq i : v_j \leq min(obj_j)$. These solutions dominate (or are equal to) any future solution with $obj_i \geq v_i$. Hence, the maximum possible value for obj_i is the smallest value of obj_i in MDD_D. The obtained algorithms have a linear time complexity in the size of the MDD, which improves the algorithm of Schaus and Hartet.

The paper is organized as follows. First, we recall some concepts and definitions of Constraint Programming, Multi-Objective Optimization Problems and Multi-valued Decision Diagrams. Then, we introduce the representation of the archive by an MDD. We present how the *insert* and *delete* operations are implemented. We detail two algorithms for the operation *filter* that establishing the bound consistency of the Pareto Constraint associated with an MDD. Next, we experiment with these methods on bin packing and multi-knapsack problems. At last, we conclude.

2 Preliminaries

2.1 Constraint Programming

A finite constraint network $\mathcal{N}$ is defined as a set of n variables $X = \{x_1, \ldots, x_n\}$, a set of current domains $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ where $D(x_i)$ is the finite set of possible values for variable x_i, and a set $\mathcal{C}$ of constraints between variables. If x is a variable, then $x^{min} = \min(D(x))$ and $x^{max} = \max(D(x))$. We introduce the particular notation $\mathcal{D}_0 = \{D_0(x_1), \ldots, D_0(x_n)\}$ to represent the set of initial domains of $\mathcal{N}$ on which constraint definitions were stated. An element of $D_0(x_1) \times \cdots \times D_0(x_n)$ on the ordered set $\mathcal{D}$ is called a tuple and is denoted τ. In a tuple τ, the assignment of the i^{th} variable is denoted τ_i. A solution of $\mathcal{N}$ is a tuple τ that satisfies all the constraints in $\mathcal{C}$. A constraint C on the ordered set of variables $X(C) = (x_{i_1}, \ldots, x_{i_r})$ is a subset $T(C)$ of the Cartesian product $D_0(x_{i_1}) \times \cdots \times D_0(x_{i_r})$ that specifies the allowed combinations of values for the variables $x_{i_1}, \ldots, x_{i_r}$.

2.2 Multi-Objective Optimization

A multi-objective problem in combinatorial optimization is a problem where several objectives have to be improved while satisfying constraints. For the sake of clarity and without loss of generality, these objectives are represented by integer variables and have to be minimized. We denote by $O = (obj_1, \ldots, obj_m)$ the ordered set of the objective variables in X, the set of variables of the whole problem. Then, this problem can be modeled as follows:

$$\begin{array}{ll} \text{Minimize} & O \\ \text{Subject to} & \mathcal{C} \end{array} \tag{1}$$

However, the minimization of several objectives simultaneously may seem ambiguous. Indeed, there is no order of priority between the different objectives and improving one often means degrading at least one of the others. This generally introduces the need to make compromises during the solving process: the goal is not to find only one optimal solution but a set of solutions that are considered *Pareto optimal*.

In the rest of the paper, we will focus only on the objective variables. As a consequence, a tuple will always be understood only in relation to the set O, and the same applies to a solution. Thus, for a tuple τ, the assignment of the i^{th} objective variable is denoted by τ_i.

The following definitions are taken from [11]:

▶ **Definition 1** (Pareto dominance). *Let τ and τ' be two solutions of a multi-objective problem represented by a constraint network $\mathcal{N}$.*
We say that τ dominates τ', denoted $\tau \prec \tau'$, if and only if:

$$\begin{array}{ll} & \forall i \in [1 \dots m] : \tau_i \leq \tau'_i \\ \wedge & \exists i \in [1 \dots m] : \tau_i < \tau'_i \end{array} \tag{2}$$

We say that τ weakly-dominates τ', denoted $\tau \preceq \tau'$, if and only if $\forall i \in [1 \dots m] : \tau_i \leq \tau'_i$.

▶ **Definition 2** (Pareto optimality). *Let $\mathcal{S}$ be the set of all the feasible solutions of a multi-objective problem represented by a constraint network $\mathcal{N}$. A solution τ is Pareto optimal if and only if there is no solution τ' in $\mathcal{S}$ that dominates τ:*

$$\nexists \tau' \in \mathcal{S} : \tau' \prec \tau \tag{3}$$

▶ **Definition 3** (Pareto set). *Let $\mathcal{N}$ be the constraint network representing a multi-objective problem and $\mathcal{S}$ be the set of all the feasible solutions of $\mathcal{N}$. The Pareto set of $\mathcal{N}$ is the set of all the Pareto optimal solutions in $\mathcal{S}$:*

$$\{\tau \in \mathcal{S} | \nexists \tau' \in \mathcal{S} : \tau' \prec \tau\} \tag{4}$$

The search for the exact Pareto set of a multi-objective problem can be impossible to achieve in a reasonable time. This leads to search for an approximation of the Pareto set: the *archive*.

▶ **Definition 4** (Archive). *An archive $\mathcal{A}$ is a set of solutions such that there is no solution τ' in the archive that dominates another solution τ in the archive. This property is known as the domination-free property:*

$$\tau \in \mathcal{A}, \nexists \tau' \in \mathcal{A} : \tau' \prec \tau \tag{5}$$

A basic way to maintain an archive $\mathcal{A}$ is to verify if a solution τ found during the search is dominated by a solution of the archive:

- If τ is dominated by at least one solution, do not add it in $\mathcal{A}$.
- If τ is not dominated by any solution, add it and remove from $\mathcal{A}$ all the solutions dominated by τ.

2.3 Pareto Constraint

We reformulate the definition introduced in [11] in order to avoid the notion of "next discovered solution".

▶ **Definition 5** (Pareto Constraint). *Let X be a set of objective variables and $\mathcal{A}$ be an archive defined on O. A* Pareto *constraint is a constraint C associated with $\mathcal{A}$ defined by* $\textsc{Pareto}(O, \mathcal{A}) = \{\tau$ *s.t.* τ *is a tuple on* O *and* $\nexists \tau' \in \mathcal{A}$ *with* $\tau' \preceq \tau$ $\}$.

When using this constraint during the search for solutions, newly found solutions must be inserted in the archive. One could notice that this constraint prevents finding a solution τ such that $\tau' \in \mathcal{A}$ and $\tau = \tau'$.

$$\begin{array}{l} D(obj_1) = \{2,4\} \\ D(obj_2) = \{1,2,5\} \\ D(obj_3) = \{1,3,4,5\} \\ D(obj_4) = \{2,5,6\} \end{array} \quad \wedge \quad \begin{array}{c} \mathcal{A} \\ \boxed{\begin{array}{c} (3,\ 1,\ 3,\ 1) \\ (2,\ 1,\ 4,\ 2) \end{array}} \end{array} \quad \Longrightarrow \quad obj_3 < 4$$

Figure 1 Application of the propagator of the Pareto constraint.

We adapt the definition of ideal point of multi-objective problems to our purpose:

▶ **Definition 6** (Ideal tuple). *Let* C =Pareto$(O, \mathcal{A})$ *be a Pareto constraint with* $O = (obj_1, \ldots, obj_n)$.

- *The ideal tuple of* C *denoted by* $\tau^*(O)$ *is the tuple composed of the best objective values, that is* $(obj_1^{min}, \ldots, obj_n^{min})$.
- *The ideal tuple for the value* a *of the variable* obj_i *denoted by* $\tau^*(O, i, a)$ *is the tuple composed of* $obj_i = a$ *and the best objective values for the other objective variables, that is* $(obj_1^{min}, \ldots, obj_{i-1}^{min}, a, obj_{i+1}^{min}, \ldots, obj_n^{min})$.

▶ **Proposition 7.** *Let* C =Pareto$(O, \mathcal{A})$ *be a Pareto constraint; the following two properties are equivalent:*

- C *is consistent;*
- $\tau^*(O)$ *is not weakly-dominated by any tuple of* $\mathcal{A}$

Proof. By definition $\tau^*(O)$ weakly-dominates any tuple defined on O, thus if $\tau^*(O)$ is not weakly-dominated then it is a possible solution and the constraint is consistent. Otherwise, there is no solution and C is not consistent. ◀

We reformulate the filtering algorithm associated with the Pareto constraint given in [11].

▶ **Proposition 8.** *Let* C =Pareto$(O, \mathcal{A})$ *be a Pareto constraint. The value* a *of the objective variable* obj_i *is not consistent with* C *if and only if* $\exists \tau \in \mathcal{A}$ *such that* $\tau \preceq \tau^*(O, i, a)$.

Proof. $\Rightarrow$ If the value a of obj_i is not consistent with C then every tuple τ of C with $\tau_i = a$ is weakly-dominated by a tuple of $\mathcal{A}$. Therefore $\tau^*(O, i, a)$ is weakly-dominated by a tuple of $\mathcal{A}$.
$\Leftarrow$ The tuple $\tau^*(O, i, a)$ weakly-dominates all the possible tuples τ of C with $\tau_i = a$. Thus if this tuple is weakly-dominated then there is no tuple with $\tau_i = a$ consistent with the constraint and the value a of obj_i is not consistent with C. ◀

From this proposition we can identify all values of all variables that are inconsistent with the constraint and so we can establish the arc consistency of the constraint which is equivalent in our case to the bound consistency. Figure 1 gives an example of domain reduction.

2.4 Multi-valued Decision Diagram

The decision diagrams considered in this paper are reduced ordered multi-valued decision diagrams (MDD) [5, 12, 1], which are a generalization of binary decision diagrams [2]. They use a fixed variable ordering for canonical representation and shared sub-graphs for compression obtained by means of a reduction operation. An MDD is a rooted directed acyclic graph (DAG) used to represent some multi-valued functions $f : \{0 \ldots d-1\}^n \rightarrow true, false$.

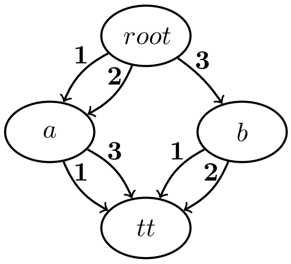


Figure 2 An MDD representing the tuple set {(1, 1), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)}.

Given the n input variables, the DAG contains $n+1$ layers of nodes, such that each variable is represented at a specific layer of the graph. Each node on a given layer has at most d outgoing arcs to nodes in the next layer of the graph. Each arc is labeled by its corresponding integer. The arc (u, a, v) is from node u to node v and labeled by a. Sometimes it is convenient to say that v is a child of u. The set of outgoing arcs from node u is denoted by $\omega^+(u)$. All outgoing arcs of the layer n reach tt, the true terminal node (the false terminal node is typically omitted). There is an equivalence between $f(a_1, \ldots, a_n) = true$ and the existence of a path from the root node to the tt whose arcs are labeled $a_1, \ldots, a_n$. Figure 2 shows an example of MDD and the kind of compression it can offer.

The reduction of an MDD is one of the most important operations that may reduce the MDD size by an exponential factor. It consists in removing nodes that have no successor and merging equivalent nodes, i.e., nodes having the same set of neighbors associated with the same labels. This means that only nodes of the same layer can be merged. Other operations used in this paper are the addition and deletion of tuples of an MDD. They can be performed with in-place operations provided by Perez and Régin [8].

The advantage of using MDDs instead of the usual data structures is their compression capability which is useful for reducing memory consumption. Moreover, this compression may also improve the time performance of algorithms computing on a set.

3 Pareto Constraint Using MDD

We propose to use MDD$_\mathcal{A}$, an MDD, to represent the solution archive. Consider τ a new solution. Without loss of generality, we assume that τ is not weakly-dominated by any tuple of the archive. As mentioned in the introduction we need to implement the operations *insert* and *delete*:

- *insert*: τ has to be added to MDD$_\mathcal{A}$.
- *delete*: All the tuples dominated by τ must be removed from MDD$_\mathcal{A}$.

In addition we need to design algorithms for implementing the *filter* operation of the Pareto constraint.

3.1 Insert and Delete Operations

Adding τ to MDD$_\mathcal{A}$ can be done thanks to the in-place addition operation [8]. The deletion from MDD$_\mathcal{A}$ of all the tuples that are dominated by τ can be done by creating MDD$_{dom}(\tau)$, the MDD of all the tuples weakly-dominated by τ. MDD$_{dom}(\tau)$ is really simple: each layer contains only one node, and for each layer $i \in [1 \ldots (n-1)]$ there are arcs labeled with all the values that belong to the range $[\tau_i \ldots max(D_0(obj_i))]$ between the node of layer i and the node of layer $i+1$. Figure 3 shows an example of such an MDD. Then, the operation MDD$_\mathcal{A}$

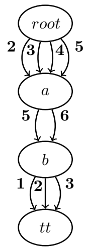


Figure 3 An MDD representing all the tuples weakly-dominated by (2, 5, 1) with $D_0(obj_1) = [1 \ldots 5]$, $D_0(obj_2) = [1 \ldots 6]$ and $D_0(obj_3) = [1 \ldots 3]$.

– $\text{MDD}_{dom}(\tau)$ can be performed in-place thanks to the in-place difference operator [8]. It should be noted that this operation also deletes the tuple τ from $\text{MDD}_{\mathcal{A}}$, so it is better to perform first the *delete* operation and then the *insert* operation.

3.2 Filtering algorithm of the Pareto Constraint

Let $C = \text{Pareto}(O, \text{MDD}_{\mathcal{A}})$ be a Pareto constraint whose archive is represented by an MDD. We present two methods that eliminate all values of the variables satisfying Proposition 8 (i.e., that are not consistent with C). That establishes the bound consistency of C. The first one has to be executed for each objective variable, and the second one uses the concept of the first method to filter all the objective variables at the same time.

3.2.1 Unidirectional Marking

This method operates only on one objective variable obj_i at a time. It must be repeated for each objective variable to be complete.

$\text{MDD}_{\mathcal{A}}$ is traversed using a Depth-First Search (DFS) procedure from *root* following two rules:

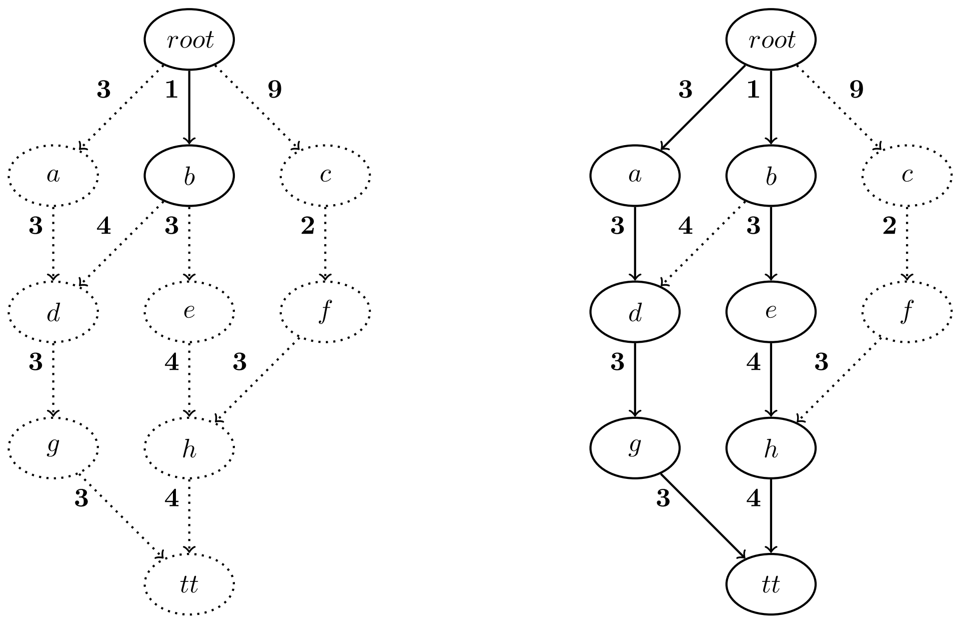
- For each layer $j \neq i$ it is possible to go only through arcs whose values are less than or equal to obj_j^{min}.
- For the layer i, it is possible to go only through arcs whose values are less than or equal to obj_i^{max}.

Each time the node *tt* is reached, the value of the arc of the layer i belonging to the current path is memorized if it is less than the previous memorized value. Then, obj_i^{max} takes the lower value between the current upper bound and the memorized value minus one. Figure 4 shows two examples based on the same archive with different states for the domains. Algorithm 3.1 is a possible implementation of this filtering for all the objective variables.

▶ **Proposition 9.** *The unidirectional marking method eliminates all the values of obj_i that are not consistent (c.f. Proposition 8).*

Proof. The unidirectional marking method finds paths corresponding to the tuples that weakly-dominate $\tau^*(O, i, obj_i^{max})$ and it eliminates all the values a such that $\tau^*(O, i, a)$ is weakly-dominated, by setting the maximum of obj_i to $minV - 1$ where $minV = \min(\{a$ such that $\tau^*(O, i, a)$ is weakly-dominated$\})$. ◀

The time complexity of this method for one objective variable is linear in the size of the MDD, because it traverses the MDD with a DFS. However, as it is repeated for each objective variable and the size of the MDD depends on the number of objective variables, the overall



(a)

- $D(obj_1) = \{2, 4\}$
- $D(obj_2) = \{2, 3, 4\}$
- $D(obj_3) = \{1, 2, 3, 4, 5\}$
- $D(obj_4) = \{3, 5, 6\}$

tt is never reached, so obj_3 does not need to be filtered.

(b)

- $D(obj_1) = \{4, 5, 8\}$
- $D(obj_2) = \{3, 7\}$
- $D(obj_3) = \{1, 2, 3, 4, 5\}$
- $D(obj_4) = \{5, 6\}$

tt is reached two times with the tuples $(3, 3, 3, 3)$ and $(1, 3, 4, 4)$. The domain of the obj_3 becomes $\{1, 2\}$.

Figure 4 Application of the unidirectional marking in $\text{MDD}_\mathcal{A}$ for the objective variable obj_3. $\text{MDD}_\mathcal{A}$ contains the set of tuples $\{(3, 3, 3, 3), (1, 4, 3, 3), (1, 3, 4, 4), (9, 2, 3, 4)\}$. All the nodes and arcs reached with the unidirectional marking method are represented with plain lines while those not reached with dotted lines.

time complexity is then quadratic in the number of objectives. This method takes advantage of the compression offered by MDDs. Nonetheless, we can notice that a large part of the DFS is shared between the filtering of each objective variable, that is to say there are many repetitions. We then propose an improvement of this method that will execute only two DFSs.

3.2.2 Bidirectional Marking

The first step is to identify in $\text{MDD}_\mathcal{A}$, for all $j \in (1 \ldots n)$, all the beginning of tuple $\tau_{(1\ldots j)} = (\tau_1, \ldots, \tau_j)$ such that $\tau_{(1\ldots j)}$ weakly-dominates $(obj_1^{min}, \ldots, obj_j^{min})$. This can be done by using a DFS from *root* to find the corresponding paths by following one rule: for the layer j it is possible to go only through arcs whose values are less than or equal to obj_j^{min}. All the nodes reached with this method are considered as marked from *root*.

▶ **Proposition 10.** *If tt is reached by the first step of the bidirectional marking method, then the* Pareto *constraint C is not consistent.*

Proof. If *tt* is reached by the first step of the bidirectional marking method it means that there exists a path corresponding to a tuple that weakly-dominates $\tau^*(O)$. Then C is not consistent according to Proposition 7. ◀

Algorithm 3.1 unidirectional marking algorithm.

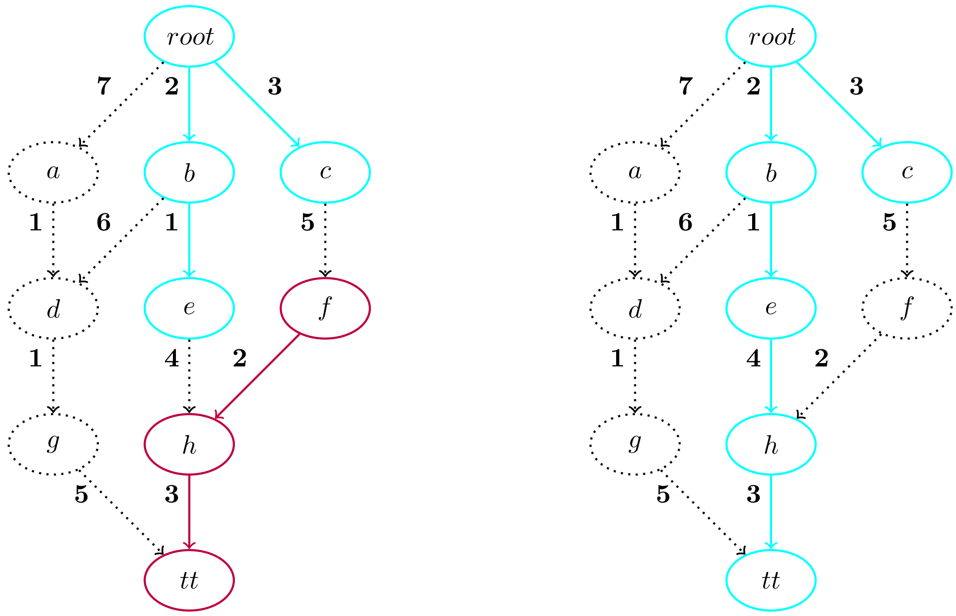
```
// min is passed by reference
recursiveDFS(MDD_A, D, i, l, u, path[], current, min)
    u.isVisited ← true
    path[l] ← u
    for each arc (u, a, v) do
        if (l = i and a ≤ obj_i^max) or a ≤ obj_l^min then
            if l = i then current ← a
            if v = tt then
                for each node ∈ path do node.reachTt ← true
                if current ≤ min then min ← current
            else
                if v.reachTt and l ≥ i and current ≤ min then min ← current
                if not v.isVisited then
                    recursiveDFS(MDD_A, D, i, l + 1, v, path, current, min)

oneWayMarkingFiltering(O, D, MDD_A)
    for each obj_i ∈ O do
        for each node ∈ MDD_A do
            node.isVisited ← false
            node.reachTt ← false
        path[] ← ∅ for each index
        currentValue ← MAX_INTEGER
        minValue ← MAX_INTEGER
        recursiveDFS(MDD_A, D, i, 1, root, path, currentValue, minValue)
        obj_i^max ← min(obj_i^max, minValue − 1)
```

The second step is similar to the first one. It consists of identifying in $\text{MDD}_{\mathcal{A}}$, for all $j \in (1 \dots n)$, all the end of tuple $\tau_{(j\dots n)} = (\tau_j, \dots, \tau_n)$ such that $\tau_{(j\dots n)}$ weakly-dominates $(obj_j^{min}, \dots, obj_n^{min})$. This time the DFS starts from *tt*, takes arcs only in reverse, and follows the same rule as for the first step. All the nodes reached during this step are considered as marked from *tt*.

The inconsistent edge can now be identified: these are the arcs (u, a, v) on the layer i, such that u is marked from *root* and v is marked from *tt*. More formally we have:

▶ **Proposition 11.** *Let Λ_i be the set of all the arcs (u, a, v) on the layer i, such that u is marked from root and v is marked from tt. The value a of obj_i satisfies Proposition 8 if and only if $\exists(u, a, v) \in \Lambda_i$*

Proof. For all the arcs (u, l, v) in Λ_i there is at least one path going from *root* to u that weakly-dominates $(obj_1^{min}, \dots, obj_{i-1}^{min})$, and there is also at least one path going from v to *tt* that weakly-dominates $(obj_{i+1}^{min}, \dots, obj_n^{min})$. It means that there is at least one tuple τ such that $\tau_i = l$ and $\tau \preceq \tau^*(O, i, l)$. Conversely, if there exists one tuple τ such that $\tau_i = l$ and $\tau \preceq \tau^*(O, i, l)$, then there is at least one path going from *root* to u that weakly-dominates $(obj_1^{min}, \dots, obj_{i-1}^{min})$, and there is also at least one path going from v to *tt* that weakly-dominates $(obj_{i+1}^{min}, \dots, obj_n^{min})$. Therefore (u, l, v) belongs to Λ_i ◀



(a)

- $D(obj_1) = \{4, 5, 8\}$
- $D(obj_2) = \{3, 4, 6, 8\}$
- $D(obj_3) = \{2, 3, 4, 7\}$
- $D(obj_4) = \{3, 6\}$

The nodes b, c and e are marked from *root*, and the nodes f and h are marked from *tt*. $D(obj_2)$ becomes {3, 4} and $D(obj_3)$ becomes {2, 3}. obj_1 and obj_4 do not need to be filtered.

(b)

- $D(obj_1) = \{4, 6\}$
- $D(obj_2) = \{3, 4, 6, 8\}$
- $D(obj_3) = \{4, 9\}$
- $D(obj_4) = \{3, 5, 6\}$

tt is reached from *root*: the constraint is not consistent.

Figure 5 Application of the bidirectional marking in $\text{MDD}_\mathcal{A}$. It represents the set of tuples {(7, 1, 1, 5), (2, 6, 1, 5), (2, 1, 4, 3), (3, 5, 2, 3)}. All the nodes and arcs reached with the bidirectional marking method are represented with plain lines while those not reached with dotted lines.

We immediately have:

▶ **Proposition 12.** *The bidirectional marking method finds and eliminates for each objective variable obj_i all the value a such that $\tau^*(O, i, a)$ is weakly-dominated, by setting the maximum of obj_i to $minV - 1$ where $minV = \min(\{a \text{ such that } \tau^*(O, i, a) \text{ is weakly-dominated}\})$.*

Algorithm 3.2 is a possible implementation of this method.

Figure 5 shows two examples based on the same archive with different states for the domains. In Figure 5 (a), obj_1 and obj_4 are not filtered because for both cases there is no arc between a node marked from *root* and a node marked from *tt* at their corresponding layer. Concerning obj_2 there is the arc $(c, 5, f)$ that satisfies this condition so $obj_2^{max} = min(8, 5$ - 1) and the domain becomes {3, 4}. For obj_3, the arc $(e, 4, h)$ is the only one that satisfies this condition so $obj_3^{max} = min(7, 4$ - 1) and the domain becomes {2, 3}. In Figure 5 (b), *tt* is reached from *root* during the first step of the bidirectional marking. Therefore there is no possible assignment with current domains.

The MDD is traversed two times with DFS and one time with the search of minimal value for each objective variable, then the time complexity of this method is linear in the size of the MDD.

Algorithm 3.2 Bidirectional marking algorithm.

```
topDownMarking(MDD_A, D, l, u, markedFromRoot)
    markedFromRoot[l].add(u)
    for each arc (u, a, v) do
        if a ≤ obj_l^min and v ∉ markedFromRoot[l + 1] then
            topDownMarking(MDD_A, D, l + 1, v, markedFromRoot)

bottomUpMarking(MDD_A, D, l, v, markedFromTt)
    markedFromTt[l + 1].add(v)
    for each arc (u, a, v) do
        if a ≤ obj_l^min and u ∉ markedFromTt[l] then
            bottomUpMarking(MDD_A, D, l - 1, u, markedFromTt)

twoWaysMarkingFiltering(O, D, MDD_A)
    size ← O.size
    for i ∈ 1 ... size + 1 do
        markedFromRoot[i] ← ∅
        markedFromTt[i] ← ∅
    topDownMarking(MDD_A, D, 1, root, markedFromRoot)
    // If tt si reached, it means that there is no
    // more possible assignment with current domains.
    if tt ∈ markedFromRoot[size + 1] then
        backtrack
    else
        bottomUpMarking(MDD_A, D, size, tt, markedFromTt)
        for i ∈ 1 ... size do
            minValue ← MAX_INTEGER
            for each u ∈ markedFromRoot[i] do
                for each arc (u, a, v) do
                    if v ∈ markedFromTt[i + 1] and a < minValue then
                        minValue ← a
            obj_i^max ← min(obj_i^max, minValue − 1)
```

4 Experiments

The methods presented in this paper have been implemented in Java 17 using Choco-solver version 4.10.10 [9]. All the experiments were run in sequential on a machine with an Intel(R) Xeon(R) W-2175 CPU @ 2.50GHz using Ubuntu 20.04.6 LTS version 5.4.0-146-generic.

In these experiments, three implementations of the Pareto constraint are compared:

- *List*: the Pareto constraint of Choco, using a list for representing the archive. When a new solution is inserted, all the solutions in the list are compared with this solution to determine if they must be removed from the list. Concerning the filtering, for each objective variable obj_i the inconsistent values are found by comparing all the solutions in the list with the *dominated point* DP_i, defined in [11, 3].
- M-U: the Pareto constraint associated with an MDD for representing the archive and using the Unidirectional marking algorithm as filtering algorithm.

Table 1 Time (s) comparison between the use of lists and MDDs for the PARETO constraint with 10 objectives. The search ends when all solutions are found or if 30000 solutions are found, or if it exceeds 30 minutes (TO).

n	Data	# Solutions found	#Solutions in $\mathcal{A}$	Total time			Filtering time			Deletion and insertion time		
				List	M-U	M-B	List	M-U	M-B	List	M-U	M-B
12	b1	6770	1687	11	46	**9**	**3**	23	8	6	**0.7**	0.8
	b2	7443	3004	36	61	**11**	7	26	**6**	25	**0.8**	0.9
	b3	7351	2651	29	55	**11**	**6**	23	7	19	**0.6**	0.7
	b4	7754	3187	40	61	**13**	**7**	32	8	29	**0.8**	0.9
	b5	8502	3284	49	129	**18**	**10**	65	13	35	**0.9**	**0.9**
16	b6	30000	4894	420	781	**103**	87	360	**70**	309	**4**	**4**
	b7	30000	5805	708	TO	**239**	170	TO	**166**	487	TO	**5**
	b8	30000	5015	615	901	**159**	**106**	460	116	478	**3**	**3**
	b9	30000	7763	1336	1171	**178**	236	467	**128**	1055	6	**5**
	b10	30000	7144	1150	TO	**312**	**215**	TO	236	881	TO	**5**
20	b11	30000	2085	265	504	**115**	100	247	**67**	123	**5**	**5**
	b12	30000	3812	386	938	**142**	131	508	**86**	207	**6**	**6**
	b13	30000	1198	**143**	827	159	**56**	439	77	17	**3**	4
	b14	30000	2647	246	1430	**205**	**107**	709	108	56	**3**	**3**
	b15	30000	2707	713	TO	**406**	427	TO	**216**	108	TO	**5**

- M-B: the PARETO constraint associated with an MDD for representing the archive and using the Bidirectional marking algorithm as filtering algorithm.

The considered problems are the bin packing and the multi-criteria knapsack problems.

4.1 Bin Packing Problem

The bin packing problem is a problem where n items have to be placed into bins. Each item has m types of weight, and each bin has a limit for each type of weight. For each type of weight the objective is to minimize the maximum weight among all the bins, so there are m objectives. These objectives encourage an equitable distribution of weights. The datasets used involve items with weights randomly chosen between 1 and 40, and a limit of 120 for each type of weight for each bin. These items have to be distributed between 8 bins. The problem is modeled using the matrix-based symmetry-breaking constraints proposed by Salem and Kieffer [10].

In order to compare the different methods, we focused on the time taken to find at most 30000 solutions. Moreover, we measured the total time taken by the filtering of the Pareto constraint throughout the search (Filtering time), and the total time taken to maintain the domination-free property of the archive throughout the search (Deletion and Insertion time, or D&I time). Table 1 shows the evolution of these times depending of the number of items n while Table 2 shows this evolution depending of the number of objectives m.

The first thing that comes out of the results of Table 1 is that M-B generally performs better than the list for this problem. These performances seem to depend on the size of the archive: the larger it is, the more the compression of MDDs shows its advantage. When we look at the time spent on the different operations, we can notice that this time saving is mainly done during the D&I part. For example with data b9, which has the largest archive with 7763 solutions, M-B is 7.5 times faster than the list and its D&I part is 200 times faster than the D&I part of the list. Concerning the filtering part M-B is sometimes slower than the list but when this is the case, it is not much slower. Moreover, when M-B is faster on the filtering it can be up to almost 2 times faster as with data b15.

The experiments in Table 2 show another interesting phenomenon about the filtering

Table 2 Time (s) comparison between the use of lists and MDDs for the PARETO constraint with 16 items. The search ends when all solutions are found or if 30000 solutions are found, or if it exceeds 30 minutes (TO).

m	Data	# Solutions found	#Solutions in $\mathcal{A}$	Total time			Filtering time			Deletion and insertion time		
				List	M-U	M-B	List	M-U	M-B	List	M-U	M-B
5	b16	4619	121	**7**	17	9	**0.5**	5	2	**0.1**	0.2	0.2
	b17	5452	640	**14**	58	19	**4**	28	9	1	**0.3**	**0.3**
	b18	5679	382	**3**	12	5	**0.6**	6	2	**0.3**	**0.3**	**0.3**
	b19	8078	597	**16**	52	20	**4**	24	9	2	**0.6**	0.7
	b20	4658	234	**3**	6	4	**0.3**	2	1	**0.1**	0.2	0.2
10	b6	30000	4894	420	781	**103**	87	360	**70**	309	**4**	**4**
	b7	30000	5805	708	TO	**239**	170	TO	**166**	487	TO	**5**
	b8	30000	5015	615	901	**159**	**106**	460	116	478	**3**	**3**
	b9	30000	7763	1336	1171	**178**	236	467	**128**	1055	6	**5**
	b10	30000	7144	1150	TO	**312**	**215**	TO	236	881	TO	**5**
15	b21	30000	2242	78	290	**44**	**16**	179	30	53	**2**	**2**
	b22	30000	4651	583	1005	**116**	105	545	**91**	456	**4**	**4**
	b23	30000	3442	271	363	**68**	70	180	**39**	178	**3**	4
	b24	30000	3379	296	1180	**97**	**54**	614	63	223	5	**4**
	b25	30000	8421	1359	1213	**121**	305	532	**77**	1014	10	**9**
20	b26	30000	7145	911	1098	**77**	185	568	**47**	706	6	**5**
	b27	30000	6828	1059	TO	**149**	202	TO	**108**	828	TO	**7**
	b28	30000	5327	684	889	**87**	140	529	**61**	521	**5**	**5**
	b29	30000	5791	819	TO	**247**	243	TO	**176**	516	TO	**6**
	b30	30000	7142	864	855	**106**	178	342	**63**	650	**7**	**7**

part: compared to the list, the more objectives there are, the faster the filtering with M-B. For example with data b26 where there are 20 objectives, the filtering with M-B is almost 4 times faster than the filtering with the list. However, when there are few objectives the list is globally better as shown by the results for $m = 5$.

The results with M-U are not as good as those with M-B, and are even worse than those with the list. Indeed, even if M-U has an advantage on the D&I part compared to the list, the filtering takes too much time which negates totally the advantage.

4.2 Multi-Criteria Knapsack Problem

This problem is a variant of the knapsack problem with n items: the goal is not to maximize only one type of profit but m types of profit. Then, each item has m values and there are m objectives to maximize. The data sets used represent items with weights and values randomly chosen between 1 and 40. For each data set, the limit of the knapsack is equal to $(\sum_{i=1}^{n} w_i)/2$ with w_i the weight of the i-th item.

We took the same types of measures as for the bin packing problem (i.e. the total time, the filtering time and the D&I time) but we only ran the methods with the list and M-B. Table 3 lists the results obtained.

The results between the methods are similar for this problem, the solving times are more or less equivalent for each instance. However, we can observe a behavior that we have already seen with the bin packing problem: M-B is generally better when the size of the archive is large. The results show also that for this problem the list is generally more efficient for the filtering while M-B is better for the D&I.

Table 3 Time (s) comparison between the use of lists and MDDs for the PARETO constraint with 10 objectives. The search ends when all solutions are found or if 10000 solutions are found.

n	Data	# Solutions found	#Solutions in $\mathcal{A}$	Total time		Filtering time		Deletion and insertion time	
				List	M-B	List	M-B	List	M-B
20	k1	1809	640	**182**	186	**1**	4	0.3	**0.2**
	k2	3667	1750	**309**	317	**6**	13	4	**0.9**
	k3	1661	1661	**134**	135	**0.5**	2	**0.2**	**0.2**
21	k4	10000	6714	786	**672**	**52**	97	175	**6**
	k5	7806	3265	**865**	868	**27**	59	15	**2**
	k6	4537	815	472	**467**	**5**	11	**1**	**1**
22	k7	3705	2750	**1011**	1050	**20**	35	8	**1**
	k8	1895	601	450	**446**	**2**	6	**0.3**	**0.3**
	k9	10000	5192	1123	**1073**	**59**	92	73	**6**
23	k10	10000	4561	1116	**1101**	**54**	99	66	**6**
	k11	4634	1593	**1360**	1420	**21**	46	3	**1**
	k12	5390	2816	1225	**1193**	**22**	40	11	**2**
24	k13	1791	823	**848**	873	**4**	12	0.5	**0.3**
	k14	10000	5781	1402	**1335**	**77**	112	89	**5**
	k15	10000	3528	**1545**	1570	**54**	92	45	**5**

5 Conclusion

In this paper we presented methods to use an MDD as an archive for the PARETO Constraint. The insertion of a new solution into the MDD and the deletion from the MDD of the solutions dominated by this new solution are made by applying classical operators. We presented two methods for establishing the bound consistency of this constraint: the unidirectional marking method and the bidirectional marking method. The second method is linear in the size of the MDD.

We have shown that, depending on the problem, using an MDD as an archive with the bidirectional marking method can be very effective compared to the classical list representation of the archive. The use of MDDs is particularly well suited to maintaining the dominance-free relation of the archive. It is also interesting for the filtering algorithm. These performances are even more important as the number of objectives increases.

References

1 David Bergman, André A. Ciré, Willem-Jan van Hoeve, and John N. Hooker. *Decision Diagrams for Optimization*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer, 2016. doi:10.1007/978-3-319-42849-9.

2 Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986. doi:10.1109/TC.1986.1676819.

3 Marco Gavanelli. An algorithm for multi-criteria optimization in csps. In *ECAI'02: Proceedings of the 15th European Conference on Artificial Intelligence*, pages 136–140, January 2002.

4 Renaud Hartert and Pierre Schaus. A support-based algorithm for the bi-objective pareto constraint. *Proceedings of the National Conference on Artificial Intelligence*, 4:2674–2679, June 2014. doi:10.1609/aaai.v28i1.9119.

5 T.Y.K. Kam and Robert K. Brayton. Multi-valued decision diagrams. Technical Report UCB/ERL M90/125, EECS Department, University of California, Berkeley, 1990. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1990/1671.html.

6 Sanaz Mostaghim and Jürgen Teich. *Quad-trees: A Data Structure for Storing Pareto Sets in Multiobjective Evolutionary Algorithms with Elitism*, pages 81–104. Springer London, London, 2005. doi:10.1007/1-84628-137-7_5.

7 Guillaume Perez. *Decision diagrams: constraints and algorithms*. PhD thesis, Université Côte d'Azur, 2017.

8 Guillaume Perez and Jean-Charles Régin. Constructions and in-place operations for mdds based constraints. In Claude-Guy Quimper, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 279–293, Cham, 2016. Springer International Publishing.

9 Charles Prud'homme and Jean-Guillaume Fages. Choco-solver: A java library for constraint programming. *Journal of Open Source Software*, 7(78):4708, 2022. doi:10.21105/joss.04708.

10 Khadija Salem and Yann Kieffer. An experimental study on symmetry breaking constraints impact for the one dimensional bin-packing problem. In *Conference: 2020 Federated Conference on Computer Science and Information Systems*, pages 317–326, September 2020. doi:10.15439/2020F19.

11 Pierre Schaus and Renaud Hartert. Multi-objective large neighborhood search. In Christian Schulte, editor, *Principles and Practice of Constraint Programming*, pages 611–627, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

12 A. Srinivasan, T. Ham, S. Malik, and R. K. Brayton. Algorithms for discrete function manipulation. In *1990 IEEE International Conference on Computer-Aided Design. Digest of Technical Papers*, pages 92–95, 1990. doi:10.1109/ICCAD.1990.129849.

13 Taha Vafaeenezhad, Reza Tavakkoli-Moghaddam, and Naoufel Cheikhrouhou. Multi-objective mathematical modeling for sustainable supply chain management in the paper industry. *Computers & Industrial Engineering*, 135:1092–1102, 2019. doi:10.1016/j.cie.2019.05.027.

14 Yihan Wang, Zongguo Wen, Jianguo Yao, and Christian Doh Dinga. Multi-objective optimization of synergic energy conservation and co2 emission reduction in china's iron and steel industry under uncertainty. *Renewable and Sustainable Energy Reviews*, 134:110128, 2020. doi:10.1016/j.rser.2020.110128.