Combining an ϵ-Constraint Method with the Pareto Global Constraint

Manuel Combarro Simón
University of Luxembourg, Esch-sur-Alzette, Luxembourg
Interdisciplinary Centre for Security, Reliability and Trust (SnT), Esch-sur-Alzette, Luxembourg

Pierre Talbot
University of Luxembourg, Esch-sur-Alzette, Luxembourg

Pascal Bouvry
University of Luxembourg, Esch-sur-Alzette, Luxembourg

Abstract

Many real-life problems involve multiple conflicting objectives; hence, the decision maker is provided with a set of trade-off solutions, the Pareto front. While many methods to compute Pareto fronts have been proposed in the mathematical programming literature, comparatively few approaches are available for constraint programming (CP). One of the main state-of-the-art algorithms in CP is a branch-and-bound method that uses a Pareto global constraint, denoted here as MOBAB-CP. In this work, we adapt the SAUGMECON algorithm, a well-known and efficient ϵ-constraint method, in a CP solver. We also propose a new algorithm that combines SAUGMECON with the Pareto global constraint. Experimental results show that the proposed algorithm consistently achieves better results than our CP implementation of SAUGMECON and is competitive with MOBAB-CP, outperforming it on several of the studied problems.

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

Keywords and phrases Multi-objective combinatorial optimization, constraint programming, Pareto global constraint, ϵ-constraint method, SAUGMECON

Digital Object Identifier 10.4230/LIPIcs.CP.2026.14

Supplementary Material *Software (Source Code)*: `https://github.com/mancs20/choco-solver/tree/CP-2026` [6], archived at `swh:1:rel:5f25b7af157a3f8a5c71eb72ba4de0f07fc7c56e`

Funding *Manuel Combarro Simón*: This work is partially funded by the Luxembourg National Research Fund (FNR) – ASTRAL Project, ref. 17043604, and by the joint research programme UL/SnT-ILNAS on Technical Standardization for Trustworthy ICT, Aerospace, and Construction.

Acknowledgements The experiments presented in this paper were carried out using the HPC facilities of the University of Luxembourg [26].

1 Introduction

Many real-world optimization problems involve multiple, potentially conflicting objectives. When preferences among objectives are not fixed a priori, the outcome is not a single optimum but a set of trade-off solutions, known as the Pareto front. In this work, we consider multi-objective combinatorial optimization (MOCO) problems, defined over integer decision variables with integer-valued objectives.

Exact methods for computing Pareto fronts have been extensively studied in the mathematical programming literature, with around 70 different algorithms identified for multi-objective integer linear programming problems [11]. In contrast, comparatively few exact approaches have been proposed within constraint programming (CP). Among them, one of the main

state-of-the-art exact algorithms in CP is the branch-and-bound approach of Gavanelli [10], whose dominance-filtering mechanism was later formalized as a Pareto global constraint in [24]. We denote this algorithm as MOBAB-CP (Section 3).

This more limited algorithmic landscape in CP raises the question of whether algorithms from the mathematical programming literature can be adapted and combined with CP-specific features, such as the Pareto global constraint, to yield new exact methods for MOCO in CP solvers.

In this paper, we make three contributions. First, we adapt the state-of-the-art ϵ-constraint algorithm SAUGMECON [28] to CP (Section 5). Second, we propose PAUGMECON, a new CP method that combines ideas from SAUGMECON with the Pareto global constraint used in MOBAB-CP (Section 6). Third, we compare SAUGMECON, PAUGMECON, and MOBAB-CP on a benchmark set of MOCO problems (Section 7).

The benchmark consists of multi-objective versions of four problems: knapsack and N-Queens (with two to five objectives), a bi-objective version of the resource-constrained project scheduling problem, and the satellite image selection problem (SIMS) [7] with two and three objectives. The results show that PAUGMECON consistently outperforms SAUGMECON and is competitive with MOBAB-CP, achieving the best performance on three of the four benchmarks for bi-objective and tri-objective problems.

2 Preliminaries

2.1 Constraint Programming

In the following, we consider constraint programming over integer variables only. Let X be a finite set of variables and C be a finite set of constraints. A *constraint network* is a pair $P = \langle d, C \rangle$ such that $d \in X \to \mathcal{P}(\mathbb{Z})$ is the domain function. For any non-empty finite set $S \subseteq \mathbb{Z}$, we write $\texttt{LB}(S) \triangleq \min S$ and $\texttt{UB}(S) \triangleq \max S$. Hence, for any variable $x \in X$, $\texttt{LB}(d(x))$ and $\texttt{UB}(d(x))$ denote the current lower and upper bounds of its domain.

An *assignment* is a function $asn \in X \to \mathbb{Z}$. Let **ASN** be the set of all assignments. An *extended assignment* is a function $asn \in X \to \mathbb{Z} \cup \{-\infty\}$. Let $Y \subseteq X$ be a set of variables. A *partial assignment* is a function $Y \to \mathbb{Z}$, and an *extended partial assignment* is a function $Y \to \mathbb{Z} \cup \{-\infty\}$.

For each constraint $c \in C$, $rel(c) \subseteq$ **ASN** denotes the set of assignments that satisfy c. For example, $rel(x = y + 3) \triangleq \{asn \in \textbf{ASN} \mid asn(x) = asn(y) + 3\}$. The set of solutions of a constraint network $\langle d, C \rangle$ is:

$$sol(d, C) \triangleq \left\{ asn \in \textbf{ASN} \;\middle|\; \forall c \in C,\ asn \in rel(c) \ \wedge\ \forall x \in X,\ asn(x) \in d(x) \right\}.$$

2.2 Multi-objective Optimization in Constraint Programming

A MOCO problem is defined by a triplet $\langle d, C, O \rangle$ such that $\langle d, C \rangle$ is a constraint network and $O \subseteq X$ is the set of objective variables. Let $\succeq$ be a preorder on **ASN**, such that $asn \succeq asn' \Leftrightarrow \forall o \in O, asn(o) \geq asn'(o)$.

For two assignments $asn, asn' \in$ **ASN**, we define:

- asn and asn' are **equivalent**, denoted as $asn \sim asn'$, if and only if $asn \succeq asn'$ and $asn' \succeq asn$. We write $asn \not\sim asn'$ to indicate that asn and asn' are **not equivalent**.
- asn **weakly dominates** asn', if $asn \succeq asn'$.
- asn **dominates** asn', denoted as $asn \succ asn'$ if and only if $asn \succeq asn' \wedge asn \not\sim asn'$.
- asn **strictly dominates** asn', denoted as $asn \succ^* asn'$, if and only if $\forall o \in O,\ asn(o) > asn'(o)$.
- asn' is **not dominated** by asn, denoted as $asn \not\prec asn'$ if and only if $asn \sim asn' \vee \exists o \in O : asn(o) < asn'(o)$.

A solution $s \in sol(d, C)$ is *weakly Pareto-optimal* (also called *weakly efficient* by other authors) if there is no $s' \in sol(d, C)$ such that $s' \succ^* s$. A solution $s \in sol(d, C)$ is called *Pareto-optimal* if $\nexists\, s' \in sol(d, C)$ such that $s' \succ s$, that is, there is no other solution that dominates s. The set of all Pareto-optimal solutions is the *Pareto set*, and its image in the objective space is the *Pareto front*. We refer to each element of the Pareto front as a *Pareto point*. The Pareto set is defined as:

$$sol(d, C, O) \triangleq \{s \in sol(d, C) \mid \nexists s' \in sol(d, C),\ s' \succ s\}.$$

In this work, we are interested in computing the Pareto front. If two or more distinct Pareto-optimal solutions have the same objective values, that is, correspond to the same Pareto point, we do not distinguish between them and keep only one representative. Consequently, the MOCO algorithms presented here return a set

$$REP(d, C, O) \subseteq sol(d, C, O)$$

that contains exactly one representative of each equivalence class of Pareto-optimal solutions.

An archive A is an approximation of $REP(d, C, O)$, such that no solution in A is dominated by any other solution in A, i.e., $\forall a \in A, \nexists b \in A : b \succ a$. We define the operator $\sqcup$ to merge two archives as

$$A \sqcup B \triangleq \{c \in A \cup B \mid \nexists d \in A \cup B,\ d \succ c\}$$

Let $s \in sol(d, C)$ be a solution with objective vector $\mathbf{u} = (s(o_1), \ldots, s(o_p))$. The objective space can be divided into the following regions with respect to $\mathbf{u}$:

- **Region weakly dominated by s.** Contains all solutions $t \in sol(d, C)$ such that $s \succeq t$.
- **Region that dominates s.** Contains all solutions $t \in sol(d, C)$ such that $t \succ s$. If s is Pareto-optimal, then no solution belongs to this region, since a Pareto-optimal solution cannot be dominated by any other solution.
- **Region that neither dominates nor is dominated by s.** Contains all solutions $t \in sol(d, C)$ such that $s \not\succ t \wedge t \not\succ s$.

When we have an order on a subset of the variables, it is sometimes clearer to forget about the variables' names. For instance, given $\{y_1, \ldots, y_n\} = Y \subseteq X$ and an assignment $asn \in X \to \mathbb{Z}$, we write $\mathbf{u} = (asn(y_1), \ldots, asn(y_n))$ for the vector associated with asn on Y. The i^{th} value of $\mathbf{u}$ corresponds to the value of the variable y_i in asn and its name is left implicit. Given a vector $\mathbf{u} = (u_1, \ldots, u_n)$ on the variables $Y = \{y_1, \ldots, y_n\}$, its associated assignment is $toasn(\mathbf{u}) \triangleq \{y_i \mapsto u_i \mid i \in [1, n]\}$. The preorder $\succeq$ on $\mathbf{ASN}$ extends naturally to vectors as follows: $\mathbf{v} \succeq \mathbf{u}$ holds iff $toasn(\mathbf{v}) \succeq toasn(\mathbf{u})$, and similarly for other operations on $\mathbf{ASN}$. In the following, we will often use the vector notation on objective variables $O = \{o_1, \ldots, o_p\}$. We abuse notation and sometimes compare a vector to an assignment, for instance, given $a \in \mathbf{ASN}$ and a vector $\mathbf{u}$, we write $a \succeq \mathbf{u}$ for $a|_Y \succeq toasn(\mathbf{u})$ where $a|_Y$ is the restriction of the assignment a to the subset Y.

Given a vector $\mathbf{u} = (u_1, \ldots, u_n)$, we write $\mathbf{u}|_{i:j}$ for the subvector $(u_i, \ldots, u_j)$, where $1 \leq i \leq j \leq n$.

Let $\langle d, C, O\rangle$ be a MOCO problem with $O = (o_1, \ldots, o_p)$. The *ideal point* is the integer vector $\mathbf{i} = (i_1, \ldots, i_p)$ where $i_k = \max\{\, s(o_k) \mid s \in sol(d, C, O)\,\}$ for $k = 1, \ldots, p$. We write $\texttt{ideal}(d, C, O)$ to denote a procedure that computes the ideal point of $\langle d, C, O\rangle$.

The *nadir point* is the integer vector $\mathbf{n} = (n_1, \ldots, n_p)$ whose k-th component is the minimum value of objective o_k among Pareto-optimal solutions, i.e., $n_k = \min\{\, s(o_k) \mid s \in sol(d, C, O)\,\}$ for $k = 1, \ldots, p$. Computing the exact nadir point is challenging, so it is typically approximated [8]. In this paper, we use $\texttt{nadir}(d, C, O)$ to denote a procedure that returns an approximation of the nadir point.

3 Multi-objective Constraint Programming: MOBAB-CP

MOBAB-CP is a branch-and-bound algorithm that computes the Pareto front in one run, without restarting the search from the root for each Pareto point. In contrast to single-objective branch-and-bound, where the incumbent is a single solution, here the incumbent is an archive A of currently known non-dominated solutions. In the multi-objective setting, improving the incumbent means finding a solution that is not weakly dominated by A; this is enforced by the Pareto global constraint $\texttt{ParetoConstraint}(O, A)$ [24], which prunes regions of the objective space that are weakly dominated by A and ensures that any newly found solution can be added to A. As in single-objective branch-and-bound, the search ends when the incumbent cannot be improved, meaning here that no additional solution can be found that is not weakly dominated by A; at this moment the image of A in the objective space corresponds to the Pareto front.

$\texttt{ParetoConstraint}(O, A)$ removes weakly dominated regions of the objective space by tightening the lower bounds of the objective variables. For each objective variable $o_i \in O$, the constraint checks whether there can exist a solution $s \in sol(d, C)$ such that $s(o_i) = \texttt{LB}(d(o_i))$ and s is not weakly dominated by the archive A. To do so, the constraint generates a vector $\mathbf{dp}^i$ by assigning to o_i its current lower bound $\texttt{LB}(d(o_i))$ and to all other objectives their current upper bounds:

$$\mathbf{dp}^i = \big(\texttt{UB}(d(o_1)), \ldots, \texttt{UB}(d(o_{i-1})), \texttt{LB}(d(o_i)), \texttt{UB}(d(o_{i+1})), \ldots, \texttt{UB}(d(o_p))\big). \tag{1}$$

If there exists $a \in A$ such that $a \succeq \mathbf{dp}^i$, then any solution $s \in sol(d, C)$ with $s(o_i) = \texttt{LB}(d(o_i))$ satisfies $\mathbf{dp}^i \succeq s$; hence by transitivity $a \succeq s$. Therefore, no non-dominated solution can satisfy $o_i = \texttt{LB}(d(o_i))$, and $\texttt{LB}(d(o_i))$ must be increased. Let $b \in A$ be the solution with the highest value of objective o_i that weakly dominates $\mathbf{dp}^i$. Then, in the current subproblem, all branches with $o_i \leq b(o_i)$ can be pruned by tightening $\texttt{LB}(d(o_i))$ to $b(o_i) + 1$.

For example, consider a bi-objective problem where in the current subproblem we have $\mathbf{dp}^1 = (3, 8)$, and $A = \{a, b, c\}$ where $a|_O = (5, 9)$, $b|_O = (4, 11)$ and $c|_O = (2, 15)$. Since $a \succeq \mathbf{dp}^1$ and $b \succeq \mathbf{dp}^1$, the largest value of o_1 among such solutions is 5, hence $\texttt{LB}(d(o_1))$ must be tightened to 6.

4 The ϵ-Constraint Method

The ϵ-constraint method is a classical approach in the mathematical programming literature to solve multi-objective integer linear problems [5]. Solutions are sought in specific regions of the objective space by iteratively solving single-objective optimization problems, prioritizing one objective over the others; in its classical form, it simply maximizes the objective o_p.

The regions are defined by partitioning the objective space using a $(p-1)$-dimensional grid whose axes correspond to the objective variables $o_1, \ldots, o_{p-1}$, with each axis k ranging from the (approximated) nadir n_k to the ideal value i_k computed in a preprocessing step. When the goal is to find the entire Pareto front and the objectives are integer-valued, consecutive grid points differ by 1 in each dimension. Each grid point is represented by a vector $\boldsymbol{\epsilon}$, indexed by $O \setminus \{o_p\}$, where ϵ_k is the coordinate of that point along the axis o_k. The vector $\boldsymbol{\epsilon}$ then defines the lower bound of a region in the objective space where $o_k \geq \epsilon_k$, with $k = 1, \ldots, p-1$. See Figure 1 for an example with three objectives.

Intuitively, the grid is swept in lexicographic order. To move to a new value of ϵ_k in dimension k, all combinations for the previous dimensions $1, \ldots, k-1$ with the current value of ϵ_k must already have been explored. When a component ϵ_k reaches its ideal value i_k and

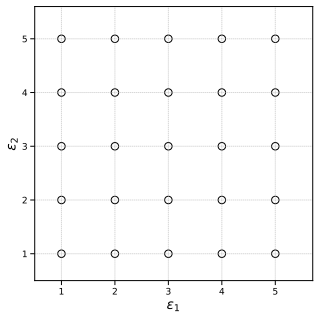


Figure 1 Example of the $(p-1)$-dimensional grid used by the ϵ-constraint method for three objectives ($p = 3$). The axes correspond to the first two objectives (o_1, o_2), whose values lie in $\{1, \ldots, 5\}$. Each grid point is a vector $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2)$ that defines the region $o_1 \geq \epsilon_1 \wedge o_2 \geq \epsilon_2$, with $\epsilon_1, \epsilon_2 \in \{1, \ldots, 5\}$.

needs to be increased, it is reset to n_k and the "carry" is propagated to the next component ϵ_{k+1}. The only exception is the last component: once ϵ_{p-1} has reached i_{p-1} and would need to be increased, the whole grid has been explored, and the Pareto front has been found.

5 SAUGMECON

Several improvements have been proposed to enhance the classical ϵ-constraint method described in Section 4. In particular, ϵ-constraint variants such as AUGMECON [20], AUGMECON2 [21], and SAUGMECON [28] aim to avoid returning weakly dominated points and to reduce the number of single-objective subproblems solved while still computing the entire Pareto front. SAUGMECON is inspired by AUGMECON and has been widely used in the mathematical programming literature [13, 27, 2, 19].

SAUGMECON follows the same grid-based framework as the classical method: it explores a $(p-1)$-dimensional grid over $O \setminus \{o_p\}$ and, at each grid point, solves a single-objective optimization problem. However, the grid is explored more efficiently thanks to three acceleration mechanisms. These mechanisms also ensure that SAUGMECON does not explore more grid points than AUGMECON [28]. Moreover, unlike in AUGMECON2, the use of pessimistic nadir estimates does not increase the computational effort [28].

In the remainder of this section, we use symbols Δ_k, Δ_j, etc. to denote non-negative integers. The three mechanisms proposed in SAUGMECON are the following:

- **Reuse of previous solutions.** In the grid, several points may produce the same solution. Information from previously solved grid points can therefore be exploited to avoid solving redundant ones. Given two grid points $\boldsymbol{\epsilon}'$ and $\boldsymbol{\epsilon}$ such that $\boldsymbol{\epsilon} \succeq \boldsymbol{\epsilon}'$ and $\boldsymbol{\epsilon}'$ is explored first, if the optimal solution found for $\boldsymbol{\epsilon}'$ weakly dominates $\boldsymbol{\epsilon}$, then the same solution will be obtained when exploring $\boldsymbol{\epsilon}$. If no solution is found for $\boldsymbol{\epsilon}'$, then no solution can be found for $\boldsymbol{\epsilon}$ either.
- **Skipping infeasible grid points.** Let $\boldsymbol{\epsilon} = (v_1, \ldots, v_{p-1})$ and suppose that the constraints $o_k \geq v_k$ for $k = 1, \ldots, p-1$ make the constraint network infeasible. In the traditional ϵ-constraint method, the next step would be to increase ϵ_1. However, any vector $\boldsymbol{\epsilon}' = (v_1 + \Delta_1, \ldots, v_{p-1})$ with $\Delta_1 \geq 1$ yields a tighter set of constraints and therefore is also infeasible. SAUGMECON then, instead of increasing ϵ_1, skips these unnecessary updates by jumping to a new vector that resets ϵ_1 to n_1 and increments ϵ_2, thus skipping all values with $\epsilon_1 > v_1$ that would also yield an infeasible subproblem. For

the general case in which infeasibility is detected when the first j components are equal to their smallest values, e.g., $\boldsymbol{\epsilon} = (n_1, \ldots, n_j, v_{j+1}, \ldots, v_{p-1})$, any vector obtained by increasing at least one of the first j components while keeping the remaining components unchanged is also infeasible. Consequently, all these grid points can be skipped. Since the grid is traversed in lexicographic order, the algorithm would next try to increase ϵ_{j+1}, i.e., move to $(n_1, \ldots, n_j, v_{j+1} + 1, v_{j+2}, \ldots, v_{p-1})$, but this again leads to infeasibility because it makes the domains at least as restrictive as for $(n_1, \ldots, n_j, v_{j+1}, \ldots, v_{p-1})$. Therefore, the entire block obtained by varying the first $j+1$ components while keeping $(\epsilon_{j+2}, \ldots, \epsilon_{p-1})$ fixed can be skipped; the next candidate point resets $\epsilon_1, \ldots, \epsilon_{j+1}$ to $n_1, \ldots, n_{j+1}$ and increases ϵ_{j+2}, yielding $(n_1, \ldots, n_{j+1}, v_{j+2} + \Delta_{j+2}, v_{j+3}, \ldots, v_{p-1})$ with $\Delta_{j+2} \geq 1$. In practice, SAUGMECON may increase ϵ_{j+2} by more than 1 thanks to the bouncing mechanism described below.

- **Bouncing steps.** Let $\boldsymbol{\epsilon} = (v_1, \ldots, v_{p-1})$. If a solution s is obtained with $s(o_1) > v_1$, then increasing ϵ_1 to $v_1 + 1$ will produce the same solution. Therefore, all grid points with $\boldsymbol{\epsilon}' = (v'_1, v_2, \ldots, v_{p-1})$ and $v'_1 \in \{v_1 + 1, \ldots, s(o_1)\}$ can be skipped. This can be generalized for any component $j > 1$. Assume that all points on the grid from $(n_1, \ldots, n_{j-1}, v_j, \ldots, v_{p-1})$ to $(i_1, \ldots, i_{j-1}, v_j, \ldots, v_{p-1})$ have been explored, then the next step is to increase ϵ_j. Let s be, among the solutions obtained for these grid points, the one with the smallest value of o_j, and suppose that $s(o_j) = v_j + \Delta_j$ with $\Delta_j > 1$. If ϵ_j is increased to any value in $\{v_j + 1, v_j + 2, \ldots, s(o_j)\}$, while the other components satisfy $\epsilon_k \in \{n_k, n_k + 1, \ldots, i_k\}$ for $k < j$ and $\epsilon_k = v_k$ for $k > j$, the constraints $o_k \geq \epsilon_k$ for $k = 1, \ldots, p-1$ will produce the same results as when ϵ_j is fixed to v_j. Then, SAUGMECON can safely increase ϵ_j to $s(o_j) + 1$. To apply this mechanism, SAUGMECON maintains a vector storing, for each objective o_k, the smallest value obtained since the last time ϵ_k was increased; this vector is called the *relatively worst-values vector*.

In the classical ϵ-method, the objective expression is o_p. SAUGMECON augments this objective expression by adding the other objectives multiplied by small coefficients. This acts as a lexicographic tie-breaking rule, ensuring that the returned optimum is Pareto-optimal.

In our CP adaptation, we perform this tie-breaking directly via lexicographic optimization rather than using the augmented term. This avoids floating-point arithmetic, since the augmented term relies on small non-integer coefficients. Moreover, floating-point objectives are unsupported in some CP solvers (e.g., OR-Tools CP-SAT [22]). Concretely, we use the global constraint for lexicographic ordering [9], to obtain a lexicographic optimum within a single search run. In the following, without loss of generality, we assume the lexicographic order $(o_p, o_1, \ldots, o_{p-1})$ for a problem with p objectives.

6 PAUGMECON

To find a Pareto front with n points, SAUGMECON solves at least n subproblems, one per Pareto point. In practice, it solves more subproblems, as some explored regions are infeasible and do not return any solution, while in other regions the returned solution has already been found in a previous subproblem[1].

To avoid rediscovering Pareto points already present in the archive, we propose a new algorithm that extends SAUGMECON by adding `ParetoConstraint`(O, A) to the constraint network, where A is the archive. The Pareto constraint forbids any solution weakly dominated

[1] The optimal solution for a grid point $\boldsymbol{\epsilon}$ may coincide with the optimal solution for a previously explored grid point $\boldsymbol{\epsilon}'$ such that $\boldsymbol{\epsilon} \not\succ \boldsymbol{\epsilon}'$; in this case, the *reuse of previous solutions* mechanism cannot be applied.

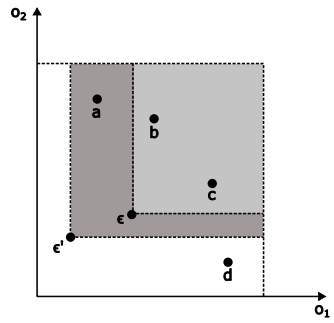


Figure 2 Use of archive subset and upper bound for objective o_3 in PAUGMECON for $p = 3$ (shown in the (o_1, o_2)- plane; $\boldsymbol{\epsilon}$ constrains o_1, o_2). **Archive subset.** If the archive contains the solutions a, b, c, d, then, in the region defined by $\boldsymbol{\epsilon}$, `ParetoConstraint`(O, A) only needs to use the archive subset $A_{\boldsymbol{\epsilon}} = \{b, c\}$. **Upper bound.** If $\boldsymbol{\epsilon}' \succ \boldsymbol{\epsilon}$ was previously solved and has associated solution a, then $a(o_3)$ is a valid upper bound for the current region; we add $o_3 \leq a(o_3)$.

by A. Consequently, when solving the subproblem for a grid point $\boldsymbol{\epsilon}$, the solver can only return a not yet discovered Pareto-optimal solution for that region. If all Pareto-optimal solutions within that region are already in A, the subproblem becomes infeasible and the *skipping infeasible grid points* mechanism is applied. We refer to this algorithm as PAUGMECON. We further strengthen PAUGMECON by (i) updating the archive with intermediate solutions, (ii) posting the Pareto constraint in the preprocessing phase to build a warm start archive, (iii) reducing Pareto-constraint overhead, (iv) extending the *reuse of previous solutions* mechanism, and (v) deriving an upper bound on o_p within each region.

Updating the archive A with only the optimal solution returned for each grid point prevents the redundant solves discussed above. Beyond that, the Pareto constraint can also strengthen propagation by pruning regions of the objective space that are weakly dominated by A. As A becomes a better approximation of the Pareto front, this dominance-based pruning becomes stronger, allowing later searches to benefit from information gathered in earlier ones and making the returned solutions more likely to be Pareto-optimal. Therefore, we also add intermediate solutions encountered during the search, i.e., solutions found before the final lexicographic optimum is returned, to improve A earlier.

Another advantage of using the Pareto constraint is that the preprocessing step for computing the ideal vector can also produce an initial archive to warm start the subsequent search. If we post `ParetoConstraint`(O, A) to the constraint network before computing the ideal value of each objective, any intermediate solution that is not dominated by the current archive can be inserted into A. One consequence of this is that the maximum value of an objective may be reached while optimizing a different objective and stored in A before the corresponding single-objective optimization is executed; when that objective is later optimized, the Pareto constraint prevents returning this already-known solution and yields a different non-dominated solution instead. Therefore, after all single-objective optimizations, we define the ideal value of each objective as the maximum value achieved for that objective in the final archive A. The resulting archive provides a first approximation of the Pareto front and is used as a warm start to strengthen pruning in the subsequent ϵ-constraint phase.

In PAUGMECON, each subproblem explores a region of the objective space defined by the current grid point $\boldsymbol{\epsilon}$. Only archive solutions that weakly dominate $\boldsymbol{\epsilon}$ can prune solutions within that region. Indeed, since $\boldsymbol{\epsilon}$ defines the lower bound of the region (i.e., $o_k \geq \epsilon_k$ for $k = 1, \ldots, p-1$), any solution in the region weakly dominates $\boldsymbol{\epsilon}$; hence, an archive solution that does not weakly dominate $\boldsymbol{\epsilon}$ cannot weakly dominate any solution in

the region. Therefore, we can restrict the archive A used in $\texttt{ParetoConstraint}(O, A)$ to the subset $\{a \in A \mid a \succeq \boldsymbol{\epsilon}\}$, which preserves correctness while reducing the number of dominance checks performed during propagation. See Figure 2 for an example in a three-objective problem. Reducing dominance check overhead is well motivated, as several works propose dedicated archive data structures to speed up archive operations (e.g., dominance checks and insertions) [10, 12, 18].

In addition, we can disable the Pareto constraint for a region, once the current search has found a solution with o_p strictly greater than the maximum o_p value among the archive points used for pruning in that region. Since subsequent solutions are sought with lexicographic priority on o_p, no archive point can then weakly dominate them, and the constraint can no longer tighten objective bounds.

PAUGMECON also extends the SAUGMECON *reuse of previous solutions* mechanism. In SAUGMECON, each explored grid point $\boldsymbol{\epsilon}$ is associated with infeasibility or with the returned lexicographic optimum for the corresponding region. In PAUGMECON, it is possible to follow the same structure, but this can miss useful reuse information. More precisely, there might be Pareto-optimal solutions obtained in the preprocessing phase or as intermediate solutions, that are not returned after the lexicographic optimization and consequently never associated with a grid point. Let A' be the subset of the archive used for the explored region defined by a given $\boldsymbol{\epsilon}$. We write $a \geq_{\texttt{lex}} b$ when the objective values of a are lexicographically greater than or equal to those of b in the order $(o_p, o_1, \ldots, o_{p-1})$. To overcome this situation, PAUGMECON associates more than one solution with $\boldsymbol{\epsilon}$: every solution $a \in A'$ such that $a \geq_{\texttt{lex}} s$ is also associated with $\boldsymbol{\epsilon}$, since it could have been returned if it were not already in the archive. This allows a grid point $\boldsymbol{\epsilon}$ that dominates a previously explored grid point $\boldsymbol{\epsilon}'$ to reuse not only the returned solution for $\boldsymbol{\epsilon}'$, but any solution associated with $\boldsymbol{\epsilon}'$ that weakly dominates $\boldsymbol{\epsilon}$. PAUGMECON maintains a set Q with information about previously obtained solutions. Each element $q \in Q$ is a pair $\langle \boldsymbol{\epsilon}, S \rangle$, where S is the set of solutions associated with $\boldsymbol{\epsilon}$ (which may be empty if the problem was infeasible). For convenience, we write $q.\boldsymbol{\epsilon}$ and $q.S$ to refer to these two components.

In PAUGMECON, we additionally derive an upper bound on the objective with the highest lexicographic priority o_p, before solving a region. This bound is obtained while reviewing previous solutions to apply the extended *reuse of previous solutions*: consider previously explored grid points $\boldsymbol{\epsilon}'$ dominated by the current $\boldsymbol{\epsilon}$. If none of the solutions associated with such grid points dominates $\boldsymbol{\epsilon}$ (otherwise one of them would be reused and the region would not be solved), then the minimum o_p value among all these associated solutions is a valid upper bound for objective o_p in the region defined by $\boldsymbol{\epsilon}$. This bound is correct, since any solution in the region defined by $\boldsymbol{\epsilon}$ also belongs to the region defined by each such $\boldsymbol{\epsilon}'$, and therefore cannot have an o_p value greater than the lexicographic optimum previously obtained for that region. Figure 2 shows an example of using a subset of the archive and applying the upper bound when solving the next region.

Algorithm 1 shows the pseudocode for PAUGMECON. The algorithm starts by adding $\texttt{ParetoConstraint}(O, A)$ to the constraint network and computing the ideal and nadir vectors $\mathbf{i}$ and $\mathbf{n}$ for the first $p-1$ objectives. To compute $\mathbf{i}$, we use an extended version of $\texttt{ideal}(d, C, O \setminus \{o_p\})$ that additionally takes the archive A as argument and inserts intermediate solutions found during search into A, so that it serves as a warm start for the grid exploration. In our implementation, `nadir` simply returns the declared lower bounds of these objective domains to avoid additional optimization runs, which is sufficient in PAUGMECON for the same reason as in SAUGMECON: pessimistic nadir values do not

Algorithm 1 PAUGMECON.

1: **function** paugmecon$(d, C, O = \{o_1, \ldots, o_p\})$
2: $A \leftarrow \emptyset$, $Q \leftarrow \emptyset$
3: $P \leftarrow$ ParetoConstraint(O, A) *(A is shared with* ParetoConstraint(O, A)*.)*
4: $C \leftarrow C \cup \{P\}$
5: $\mathbf{i} \leftarrow$ ideal$(d, C, O \setminus \{o_p\}, A)$ *(A is updated with intermediate solutions.)*
6: $\mathbf{n} \leftarrow$ nadir$(d, C, O \setminus \{o_p\})$, $\boldsymbol{\epsilon} \leftarrow \mathbf{n}$, $\mathbf{w} \leftarrow \mathbf{i}$, $best \leftarrow \{o \mapsto -\infty \mid o \in O\}$)
7: $C \leftarrow C \cup \{$Lex$((o_p, o_1, \ldots, o_{p-1}), best)\}$ *(Add lexicographic ordering constraint.)*
8: **while** $\epsilon_{p-1} \leq i_{p-1}$ **do**
9: $Q_{\succeq} \leftarrow \{q \in Q \mid \boldsymbol{\epsilon} \succeq q.\boldsymbol{\epsilon}\}$ *(Reuse of previous solutions.)*
10: **if** $\exists q \in Q_{\succeq},\ \big(q.S = \emptyset \ \vee\ \exists s \in q.S,\ s \succeq \boldsymbol{\epsilon}\big)$ **then**
11: **if** $q.S = \emptyset$ **then**
12: $s \leftarrow \{o \mapsto -\infty \mid o \in O\}$
13: **else**
14: choose $s \in q.S$ such that $s \succeq \boldsymbol{\epsilon}$
15: **end if**
16: **else**
17: $up \leftarrow \min(\{+\infty\} \cup \{s(o_p) \mid q \in Q_{\succeq},\ s \in q.S\})$
18: $d' \leftarrow d$
19: $d'(o_p) \leftarrow \{v \in d(o_p) \mid v \leq up\}$
20: **for** $k = 1$ **to** $p - 1$ **do**
21: $d'(o_k) \leftarrow \{v \in d(o_k) \mid v \geq \epsilon_k\}$ *(Restrict objectives to the current grid point.)*
22: **end for**
23: $A' \leftarrow \{a \in A \mid a \succ \boldsymbol{\epsilon}\}$
24: $u \leftarrow \max(\{-\infty\} \cup \{a(o_p) \mid a \in A'\})$
25: $s \leftarrow$ lexmax$(d', C, P, u, best, A')$
26: **if** $\exists o \in O,\ s(o) = -\infty$ **then** *(No solution was found.)*
27: $Q \leftarrow Q \cup \{\langle \boldsymbol{\epsilon}, \{\}\rangle\}$
28: **else**
29: $Q \leftarrow Q \cup \{\langle \boldsymbol{\epsilon}, \{a \in A' \mid a \geq_{\texttt{lex}} s\}\rangle\}$ *($\geq_{\texttt{lex}}$ uses the order $(o_p, o_1, \ldots, o_{p-1})$.)*
30: $C \leftarrow C \cup \{P\}$
31: $best \leftarrow \{o \mapsto -\infty \mid o \in O\}$ *(Restart best.)*
32: $A \leftarrow A \sqcup A'$
33: **end if**
34: **end if**
35: updateEpsilon$(s, \boldsymbol{\epsilon}, \mathbf{i}, \mathbf{n}, \mathbf{w})$
36: **end while**
37: **return** A
38: **end function**

increase computational effort [28]. The grid vector $\boldsymbol{\epsilon}$ is initialized to $\mathbf{n}$ and the relatively worst-values vector $\mathbf{w}$ to $\mathbf{i}$. As in the classical ϵ-constraint method, each iteration of the main loop (lines 8–36) corresponds to the evaluation of a point in the grid.

For each grid point, the algorithm first computes the set $Q_{\succeq}$ of previously explored grid points weakly dominated by $\boldsymbol{\epsilon}$. If one of them is associated either with infeasibility or with a solution that weakly dominates $\boldsymbol{\epsilon}$, then the current region can be skipped and the corresponding result is reused directly. Otherwise, the algorithm solves the region defined

Algorithm 2 Update $\boldsymbol{\epsilon}$.

```
1:  function updateEpsilon(s, ε, i, n, w)
2:      if ∃o ∈ O, s(o) = −∞ then                (Skip grid points that would lead to infeasibility.)
3:          j ← min({ k ∈ {1, ..., |ε| − 1} | ε_k ≠ n_k } ∪ {|ε| − 1})
4:          ε|_{1:j} ← i|_{1:j}
5:      else                                      (Update the relative worst value for each objective.)
6:          w_1 ← s(o_1)
7:          for k = 2 to |ε| do
8:              w_k ← min(s(o_k), w_k)
9:          end for
10:     end if
11:     for k = 1 to |ε| do                       (Update ε to represent the next grid point.)
12:         if ε_k < i_k ∧ w_k < i_k then
13:             ε_k ← w_k + 1
14:             w_k ← i_k
15:             return
16:         else if k < |ε| then
17:             ε_k ← n_k
18:         else
19:             ε_k ← i_k + 1                     (The whole grid is explored.)
20:         end if
21:     end for
22: end function
```

by $\boldsymbol{\epsilon}$. It first restricts the domains of $o_1, \ldots, o_{p-1}$ according to the current grid point and posts an upper bound on o_p derived from the solutions associated with $Q_{\succeq}$. Then it extracts the region-specific archive subset $A' = \{a \in A \mid a \succeq \boldsymbol{\epsilon}\}$ and computes u, the maximum value of o_p among the solutions in A'. The procedure `lexmax` then performs lexicographic optimization in the order $(o_p, o_1, \ldots, o_{p-1})$ using the global constraint $\texttt{Lex}((o_p, o_1, \ldots, o_{p-1}), \mathit{best})$, which enforces that any subsequent solution must be lexicographically greater than *best*. Each time a feasible solution is found, it is assigned to *best* and inserted into A', so that the search continues only for lexicographically better solutions. Moreover, once a solution s' with $s'(o_p) > u$ is found, the Pareto constraint can no longer prune any subsequent solution in the region and is therefore removed from the constraint network. If no feasible solution is found, `lexmax` returns the original value of *best*, which indicates infeasibility. After the call, the algorithm stores in Q either infeasibility or the set of reusable solutions associated with $\boldsymbol{\epsilon}$, adds P back to the constraint network for the next grid point, and merges A' into the global archive A.

After evaluating a point of the grid, it is necessary to update $\boldsymbol{\epsilon}$ to continue exploring the grid; this is done in the function `updateEpsilon` (Algorithm 2), where the other two SAUGMECON mechanisms are applied: *skipping infeasible grid points* and *bouncing steps*. If a solution was obtained, the vector with the relatively worst values $\mathbf{w}$ is updated (lines 6–9). The first component, w_1, is set to $s(o_1)$, and for each $k = 2, \ldots, p-1$, the component w_k is updated to $s(o_k)$ if $s(o_k) < w_k$. If instead no solution is found for the current grid point, the *skipping infeasible grid points* mechanism (lines 3–4) is applied by setting the first j components of $\boldsymbol{\epsilon}$ to their ideal values, where j is the first index such that $\epsilon_j \neq n_j$. This update simulates that all grid points in that prefix range have been explored and that those components should be reset to their nadir values in the loop between lines 11 and 21.

Finally, $\boldsymbol{\epsilon}$ is updated in order from ϵ_1 to ϵ_{p-1} (lines 11–21). The first component k such that $\epsilon_k < i_k$ and $w_k < i_k$ triggers a bounce: the algorithm sets $\epsilon_k \leftarrow w_k + 1$ and resets $w_k \leftarrow i_k$; no later component needs to be modified, and the exploration continues from the next grid point (*bouncing steps* mechanism). For all preceding components $j < k$, the algorithm is in the "carry" case: if $j < p-1$, ϵ_j is reset to its nadir value n_j, which means that all values for that component have been explored in the current round and the next component should be updated. If the loop reaches $k = p-1$ without triggering a bounce, then ϵ_{p-1} is set to $i_{p-1} + 1$; consequently, the condition of the main while-loop (line 8) fails in the next iteration, and the algorithm terminates with the image of A equal to the Pareto front.

7 Experimental Results

This section compares the previously presented multi-objective algorithms, MOBAB-CP, SAUGMECON, and PAUGMECON, on four MOCO problems: multi-objective 0/1 unidimensional knapsack (MUKP), multi-objective N-Queens (MN-Queens), multi-objective resource-constrained project scheduling (MORCPSP), and SIMS, considering objectives such as cost, cloud coverage, and incidence angle.

When the Pareto front was found for most instances of a problem by all algorithms, we compared the algorithms based on execution time. We report normalized runtime: for each instance, the completion time is divided by the minimum completion time achieved by any algorithm on that instance. Thus, a normalized value of 1 corresponds to the best time on that instance, and a value of $x > 1$ means the algorithm is x times slower than the best.

If the Pareto front was not found for most instances, we compared the approximate fronts returned within the time limit using hypervolume (HV) [29] and inverted generational distance with modified distance calculation (IGD^+) [14]. Both indicators are computed using `pymoo` [3].

To avoid HV and IGD^+ being overly influenced by objectives with larger numeric ranges, we apply a per-objective min–max normalization for each instance. For each objective, the best and worst values are taken over all solutions returned by any algorithm for that instance. After normalization, each objective lies in $[0, 1]$, with 0 corresponding to the best value and 1 to the worst value.

The HV measures the volume of the region dominated by the approximated front and delimited by a reference point; higher values indicate better convergence and diversity. In our case, after normalization, we used the reference point $(1.1, \ldots, 1.1)$, which is component-wise worse than all points in the considered fronts. To enable comparison across instances, we normalize the HV values relative to the maximum HV obtained per instance.

IGD^+ measures the average minimum Euclidean distance from each point on a reference front to the weakly dominated region induced by the approximated front. The reference front is the Pareto front when it is known; otherwise, we approximate it by merging the solution sets returned by all algorithms using the operator $\sqcup$. Lower values indicate better convergence and diversity.

We report ranked-instance profiles (cactus plots): for each algorithm, instances are sorted independently from best to worst according to the reported metric; the x-axis shows the rank (not a shared instance index) and the y-axis shows the corresponding metric value. For maximization metrics (e.g., HV) higher curves are better, while for minimization metrics (e.g., runtime, IGD^+) lower curves are better. When curves cross, we consider that a strategy is better than another if it is above/below the other over larger portions of the rank axis.

The experiments were conducted using the Choco open-source constraint programming solver [23], running on an AMD EPYC ROME 7H12 64-core processor (2.6 GHz). Each instance was solved using a single core, with a timeout of 1.5 hours for MUKP and MN-Queens, and 3 hours for MORCPSP and SIMS.

All algorithms use the same search procedure. The search follows a deep-first exploration of the search tree, branching on the domains of the decision variables. For variable and value selection, we use the well-known dom/wdeg search strategy [4], which is also the default Choco search strategy.

7.1 MUKP

Given a set of items with weight and value, the 0/1 uni-dimensional knapsack problem consists of selecting a subset of these items, where each item can be selected at most once, such that the total value is maximized and the total weight remains below a given weight limit. MUKP is the multi-objective extension of this problem, where each item has one value per objective and the goal is to maximize all objectives.

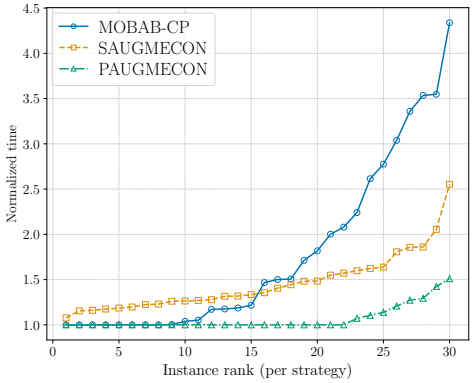
The instances used for the experiments vary the number of items from 10 to 50 and the number of objectives from two to five. All instances with more than two objectives were taken from [15] and are available in MOOLibrary (`https://github.com/srplsyn/MOOLibrary/tree/main`). The bi-objective instances were generated following the same procedure as in [15], where v_i^j and w_i are random integers drawn from the interval $[1, 1000]$, and the weight limit is $\frac{1}{2}\sum_{i=1}^{n} w_i$.

Figure 3 reports the ranked-instance profiles for normalized runtime. PAUGMECON consistently improves over SAUGMECON for all objective settings. For two and three objectives, PAUGMECON achieves the best overall profile, while MOBAB-CP is the worst for two objectives and only slightly behind PAUGMECON for three objectives. For four and five objectives, MOBAB-CP becomes the best method, but PAUGMECON remains substantially better than SAUGMECON, whose worst normalized runtimes grow much more sharply. Overall, these results suggest that PAUGMECON is competitive for bi-objective and tri-objective MUKP, while its advantage over MOBAB-CP decreases as the number of objectives increases.

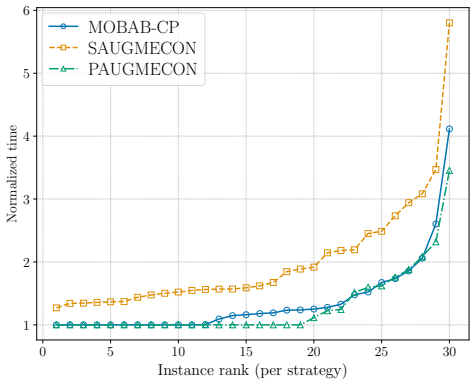
7.2 MN-Queens

N-Queens is a classical constraint problem where the goal is to place N queens on an $N \times N$ chessboard such that no two queens attack each other. Following the formulation proposed in [17], an objective is defined as follows: a random score between 0 and N is assigned to each square. A square is active if a queen is placed on it. The objective value is the sum of the scores of the active squares. For p objectives, p scores are generated per square. All objectives are maximized. For the experiments, we generated instances with 8, 10, 12, and 14 queens and between two and five objectives.

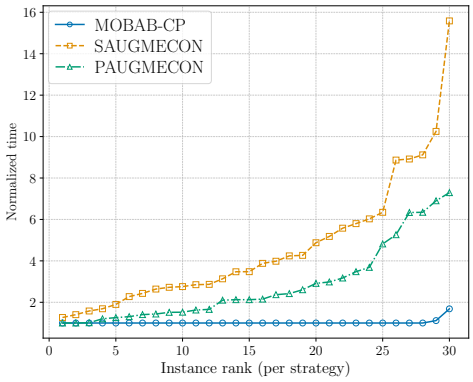
Figure 4 reports the ranked-instance profiles for normalized runtime. Overall, MOBAB-CP is by far the best-performing algorithm on MN-Queens, being the fastest on all but one instance. It achieves the best profile for all objective settings, and the gap with SAUGMECON and PAUGMECON widens as the number of objectives increases. Moreover, MOBAB-CP is the only method that solves all instances within the time limit, whereas SAUGMECON misses 25 instances and PAUGMECON 10. As in MUKP, PAUGMECON consistently improves over SAUGMECON, sometimes by nearly one order of magnitude, but it remains clearly behind MOBAB-CP on this problem.



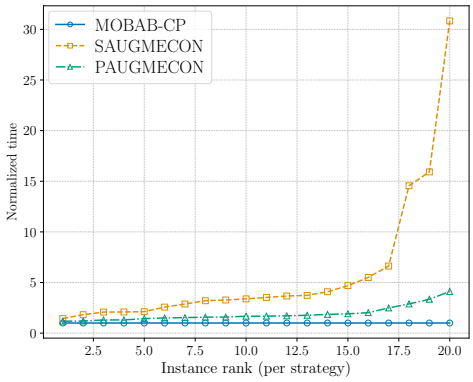
(a) 2 objectives.



(b) 3 objectives.



(c) 4 objectives. Logarithmic scale on the y-axis.



(d) 5 objectives. Logarithmic scale on the y-axis.

Figure 3 Normalized runtime profiles for MUKP. Lower curves indicate better performance.

7.3 MORCPSP

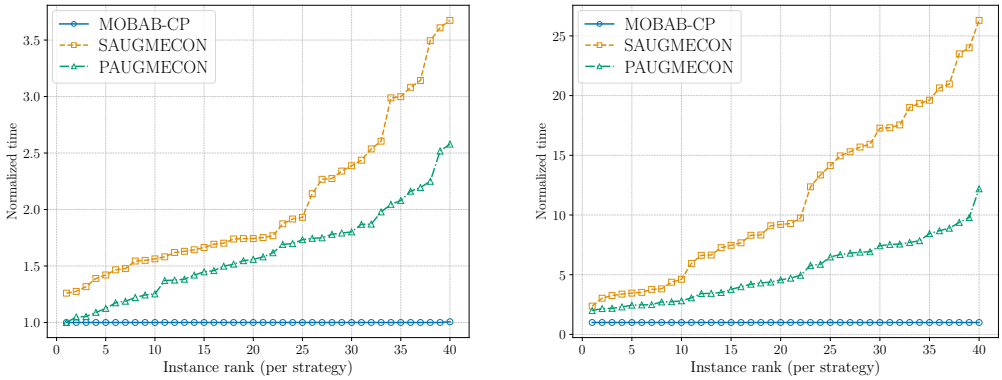
The resource-constrained project scheduling problem (RCPSP) is a well-known scheduling problem. It involves computing a schedule for a project with J activities subject to precedence constraints. Each activity j has a fixed duration and cannot be interrupted once it starts. An activity can only begin after all its predecessors have completed. Activities require renewable resources to be executed, which may be shared, but resource usage at any time must not exceed the resource capacity. In its classical form, the objective is to minimize the duration of the project, i.e., the makespan.

In addition to the makespan, other performance criteria may be relevant, such as the weighted earliness-tardiness penalty. Each activity j has a due date d_j, an earliness cost e_j, and a tardiness cost t_j (per unit of time). Let c_j be the completion time of activity j. The weighted earliness-tardiness objective is defined as:

$$\sum_{j=1}^{J} (e_j E_j + t_j T_j), \quad \text{where } E_j = \max(0, d_j - c_j), \quad T_j = \max(0, c_j - d_j) \tag{2}$$

The MORCPSP is a bi-objective extension of the RCPSP, where the objectives are the makespan and the weighted earliness-tardiness penalty.

For the experiments, we used 70 instances from the PSPLIB benchmark [16]: 30 instances with 30 activities, 30 with 60, and 10 with 90. We added the due dates to the instances following the methodology described in [25]. For each instance, a maximum due date was



(a) 2 objectives.

(b) 3 objectives.

(c) 4 objectives. Logarithmic scale on the y-axis.

(d) 5 objectives. Logarithmic scale on the y-axis.

Figure 4 Normalized runtime profiles for MN-Queens. Lower curves indicate better performance.

calculated by multiplying the critical path by 2.5. Then, J random numbers were generated between 1 and the maximum due date. These values were sorted in increasing order and assigned as the due dates for activities $1, 2, \ldots, J$. For each activity, the earliness and tardiness costs were generated as random integers between 0 and 5.

Among the 70 MORCPSP instances, only 22 were completed within the time limit by at least one algorithm. We therefore compare algorithms using HV and IGD$^+$.

Figure 5 reports ranked-instance profiles for normalized HV and IGD$^+$, respectively. PAUGMECON is the best strategy according to both HV and IGD$^+$. Moreover, it finds the Pareto front for 22 instances, whereas SAUGMECON does so for 20 and MOBAB-CP for none. SAUGMECON and MOBAB-CP show similar performance overall, remaining clearly behind PAUGMECON on both indicators.

7.4 SIMS

The SIMS problem involves selecting optical satellite images to study a given area of interest (AOI) on Earth. Because the AOI is too large to be covered by a single image, multiple images must be selected and stitched together to form a larger image, called a mosaic. The goal is to cover the entire AOI while optimizing several objectives. Following the preprocessing step described in [7], the problem is reformulated as a special case of the set covering problem: the AOI becomes the universe, each image is a set, and the elements are the non-overlapping polygons formed by image intersections.

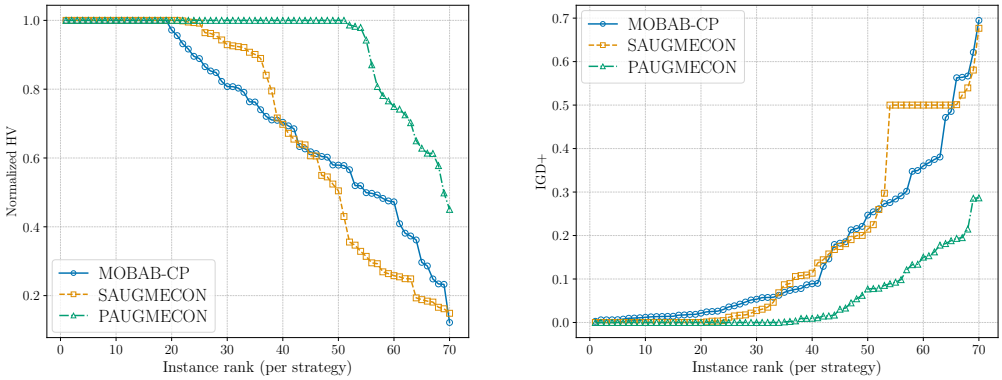


Figure 5 Normalized HV and IGD$^+$ ranked-instance profiles for MORCPSP. Higher curves indicate better performance for HV; lower curves indicate better performance for IGD$^+$.

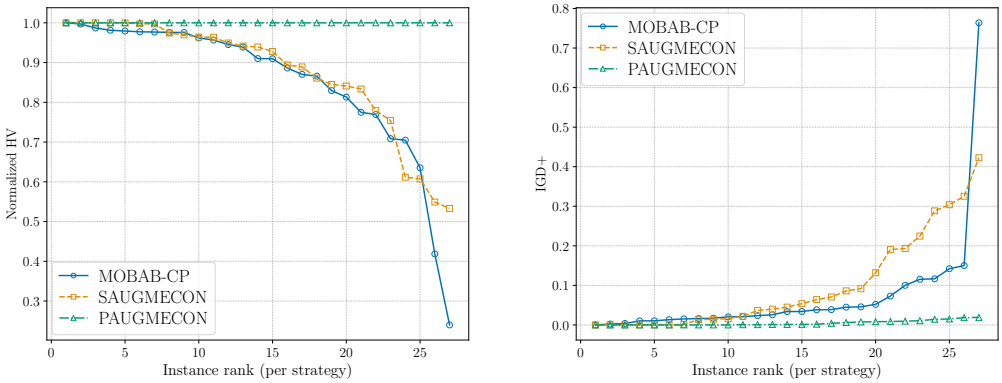


Figure 6 Normalized HV and IGD$^+$ ranked-instance profiles for SIMS. Higher curves indicate better performance for HV; lower curves indicate better performance for IGD$^+$.

We consider three objectives to be minimized, as defined in [7]: the total cost of the selected images, the maximum incidence angle, and the cloud coverage of the resulting mosaic. The incidence angle refers to the angle between the satellite and the ground normal at the time of image capture [1]. Cloud coverage is the percentage of cloudy areas in the mosaic. Cloudy regions are modeled as special elements, which can be excluded if they are overlapped by non-cloudy regions from other images.

We conducted two sets of experiments: one considering cost and cloud coverage, and another including the incidence angle as a third objective. We used the same benchmark as in [7], which includes five AOIs, each with five scenarios of image availability (30, 50, 100, 150, and 200 images), except for one AOI with only four scenarios. This resulted in 24 instances for both the bi-objective and tri-objective settings.

From the 48 instances, only 27 were completed by at least one algorithm. We therefore compare algorithms using HV and IGD$^+$, following the same evaluation protocol as in Section 7.3.

Figure 6 reports ranked-instance profiles for normalized HV and IGD$^+$ for both SIMS settings. PAUGMECON clearly outperforms the other algorithms on both indicators, achieving the best HV on all but one instance. Moreover, it completes the largest number of instances, 26, whereas SAUGMECON solves 24 and MOBAB-CP 22. Only SAUGMECON

completes an instance not completed by PAUGMECON; this is also the only instance for which PAUGMECON does not achieve the best HV. However, its normalized HV for that instance remains extremely close to 1, indicating that its approximated front is very close to the Pareto front. SAUGMECON is slightly better in HV than MOBAB-CP, while MOBAB-CP is better in IGD^+.

8 Conclusions

In this paper, we proposed a new CP algorithm, PAUGMECON, to compute the Pareto front of MOCO problems by combining the Pareto global constraint with the ϵ-constraint algorithm SAUGMECON. The experimental results show that PAUGMECON consistently improves over SAUGMECON and is the best-performing method on three of the four benchmarks for bi-objective and tri-objective problems. In addition, it can outperform MOBAB-CP in settings where SAUGMECON does not, showing that integrating the Pareto constraint together with the additional mechanisms substantially strengthens the original ϵ-constraint approach. However, the performance of both PAUGMECON and SAUGMECON degrades significantly when considering more than three objectives. A practical takeaway is that PAUGMECON is generally preferable for bi-objective and tri-objective problems, whereas for more than three objectives MOBAB-CP should be preferred. Overall, these results suggest that combining ideas from the mathematical programming literature with the Pareto global constraint is a promising direction for developing new exact methods for MOCO problems in CP.

References

1 Airbus. Incidence angle. URL: `https://www.intelligence-airbusds.com/en/8719-angle-conversion`.

2 Paulina Avila-Torres, Rafael Caballero, Igor S. Litvinchev, Fernando López Irarragorri, and Pandian Vasant. The urban transport planning with uncertainty in demand and travel time: a comparison of two defuzzification methods. *Journal of Ambient Intelligence and Humanized Computing*, 9(3):843–856, 2018. `doi:10.1007/s12652-017-0545-X`.

3 Julian Blank and Kalyanmoy Deb. Pymoo: Multi-objective optimization in python. *IEEE Access*, 8:89497–89509, 2020. `doi:10.1109/ACCESS.2020.2990567`.

4 Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *Proceedings of the 16th European Conference on Artificial Intelligence*, ECAI'04, pages 146–150, NLD, 2004. IOS Press.

5 Vira Chankong and Yacov Y. Haimes. *Multiobjective Decision Making: Theory and Methodology*. Elsevier Science Ltd, 1983.

6 Manuel Combarro Simón, Pierre Talbot, and Pascal Bouvry. Multi-objective algorithms implemented in Choco solver. Software, swhId: `swh:1:rel:5f25b7af157a3f8a5c71eb72ba4de0f07fc7c56e` (visited on 2026-06-29). URL: `https://github.com/mancs20/choco-solver/tree/CP-2026`, `doi:10.4230/artifacts.26934`.

7 Manuel Combarro Simón, Pierre Talbot, Grégoire Danoy, Jedrzej Musial, Mohammed Alswaitti, and Pascal Bouvry. Constraint Model for the Satellite Image Mosaic Selection Problem. In Roland H. C. Yap, editor, *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 44:1–44:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.CP.2023.44`.

8 Matthias Ehrgott. *Multicriteria Optimization*. Springer-Verlag, Berlin, Heidelberg, 2005.

9 Alan Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, and Toby Walsh. Global constraints for lexicographic orderings. In Pascal Van Hentenryck, editor, *Principles and Practice of Constraint Programming – CP 2002*, pages 93–108, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. `doi:10.1007/3-540-46135-3_7`.

10 Marco Gavanelli. An algorithm for multi-criteria optimization in csps. In Frank van Harmelen, editor, *Proceedings of the 15th European Conference on Artificial Intelligence, ECAI'2002, Lyon, France, July 2002*, pages 136–140. IOS Press, 2002.

11 Pascal Halffmann, Luca E. Schäfer, Kerstin Dächert, Kathrin Klamroth, and Stefan Ruzika. Exact algorithms for multiobjective linear optimization problems with integer variables: A state of the art survey. *Journal of Multi-Criteria Decision Analysis*, 29(5-6):341–363, 2022. doi:10.1002/mcda.1780.

12 Renaud Hartert and Pierre Schaus. A support-based algorithm for the bi-objective pareto constraint. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, June 2014. doi:10.1609/aaai.v28i1.9119.

13 Mahyar Lasemi Imeni, Mohammad Sadegh Ghazizadeh, Mohammad Ali Lasemi, and Zhenyu Yang. Optimal scheduling of a hydrogen-based energy hub considering a stochastic multi-attribute decision-making approach. *Energies*, 16(2), 2023. doi:10.3390/en16020631.

14 Hisao Ishibuchi, Hiroyuki Masuda, Yuki Tanigaki, and Yusuke Nojima. Modified distance calculation in generational distance and inverted generational distance. In António Gaspar-Cunha, Carlos Henggeler Antunes, and Carlos A. Coello Coello, editors, *Evolutionary Multi-Criterion Optimization – 8th International Conference, EMO 2015, Guimarães, Portugal, March 29 -April 1, 2015. Proceedings, Part II*, volume 9019 of *Lecture Notes in Computer Science*, pages 110–125. Springer, 2015. doi:10.1007/978-3-319-15892-1_8.

15 Gokhan Kirlik and Serpil Sayin. A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems. *European Journal of Operational Research*, 232(3):479–488, 2014. doi:10.1016/j.ejor.2013.08.001.

16 Rainer Kolisch and Arno Sprecher. PSPLIB – a project scheduling problem library: OR software – ORSEP operations research software exchange program. *European Journal of Operational Research*, 96(1):205–216, 1997. doi:10.1016/S0377-2217(96)00170-1.

17 Martin Lukasiewycz, Michael Glaß, Christian Haubelt, and Jürgen Teich. Solving multi-objective pseudo-boolean problems. In João Marques-Silva and Karem A. Sakallah, editors, *Theory and Applications of Satisfiability Testing – SAT 2007, 10th International Conference, Lisbon, Portugal, May 28-31, 2007, Proceedings*, volume 4501 of *Lecture Notes in Computer Science*, pages 56–69. Springer, 2007. doi:10.1007/978-3-540-72788-0_9.

18 Steve Malalel, Arnaud Malapert, Marie Pelleau, and Jean-Charles Régin. MDD Archive for Boosting the Pareto Constraint. In Roland H. C. Yap, editor, *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CP.2023.24.

19 Nancy M. Arratia Martínez, Rafael Caballero-Fernandez, Igor S. Litvinchev, and Fernando López Irarragorri. Research and development project portfolio selection under uncertainty. *Journal of Ambient Intelligence and Humanized Computing*, 9(3):857–866, 2018. doi:10.1007/s12652-017-0564-7.

20 George Mavrotas. Effective implementation of the epsilon-constraint method in multi-objective mathematical programming problems. *Applied Mathematics and Computation*, 213(2):455–465, 2009. doi:10.1016/j.amc.2009.03.037.

21 George Mavrotas and Kostas Florios. An improved version of the augmented ϵ-constraint method (AUGMECON2) for finding the exact pareto set in multi-objective integer programming problems. *Appl. Math. Comput.*, 219(18):9652–9669, 2013. doi:10.1016/j.amc.2013.03.002.

22 Laurent Perron and Frédéric Didier. CP-SAT Solver. https://developers.google.com/optimization/cp/cp_solver/, 2025. Version v9.12, 2025-02-17.

23 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.

24 Pierre Schaus and Renaud Hartert. Multi-objective large neighborhood search. In Christian Schulte, editor, *Principles and Practice of Constraint Programming – 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 611–627. Springer, 2013. doi:10.1007/978-3-642-40627-0_46.

25 Mario Vanhoucke, Erik Demeulemeester, and Willy Herroelen. An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem. *Annals of Operations Research*, 102(1-4):179–196, 2001. doi:10.1023/A:1010958200070.

26 Sebastien Varrette, Hyacinthe Cartiaux, Sarah Peter, Emmanuel Kieffer, Teddy Valette, and Abatcha Olloh. Management of an Academic HPC & Research Computing Facility: The ULHPC Experience 2.0. In *Proceedings of the 6th ACM High Performance Computing and Cluster Technologies Conf. (HPCCT 2022)*, Fuzhou, China, July 2022. Association for Computing Machinery (ACM).

27 Behzad Zahiri, Jun Zhuang, and Mehrdad Mohammadi. Toward an integrated sustainable-resilient supply chain: A pharmaceutical case study. *Transportation Research Part E: Logistics and Transportation Review*, 103:109–142, 2017. doi:10.1016/j.tre.2017.04.009.

28 Weihua Zhang and Marc Reimann. A simple augmented ϵ-constraint method for multi-objective mathematical integer programming problems. *European Journal of Operational Research*, 234(1):15–24, 2014. doi:10.1016/j.ejor.2013.09.001.

29 Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999. doi:10.1109/4235.797969.