# A Worst-Case Analysis of Constraint-Based Algorithms for Exact Multi-Objective Combinatorial Optimization

Jianmei Guo[1⋆], Eric Blais[2], Krzysztof Czarnecki[2], and Peter van Beek[2⋆]

[1] East China University of Science and Technology, China
gjm@ecust.edu.cn
[2] University of Waterloo, Canada
{eric.blais,k2czarnecki,peter.vanbeek}@uwaterloo.ca

**Abstract.** In a multi-objective combinatorial optimization (MOCO) problem, multiple objectives must be optimized simultaneously. In past years, several constraint-based algorithms have been proposed for finding Pareto-optimal solutions to MOCO problems that rely on repeated calls to a constraint solver. Understanding the properties of these algorithms and analyzing their performance is an important problem. Previous work has focused on empirical evaluations on benchmark instances. Such evaluations, while important, have their limitations. Our paper adopts a different, purely theoretical approach, which is based on characterizing the search space into subspaces and analyzing the worst-case performance of a MOCO algorithm in terms of the expected number of calls to the underlying constraint solver. We apply the approach to two important constraint-based MOCO algorithms. Our analysis reveals a deep connection between the search mechanism of a constraint solver and the exploration of the search space of a MOCO problem.

## 1 Introduction

In a *multi-objective combinatorial optimization (MOCO)* problem, multiple objectives must be optimized simultaneously. MOCO problems arise in many areas where there are tradeoffs, such as engineering, finance, and logistics. For example, in the design of systems, one often has to choose between different candidate designs that balance multiple objectives, such as low cost, high performance, and high reliability. Since these objectives are often conflicting, there is usually no single optimal solution that excels in all objectives. Therefore, decision makers would like to know various, ideally all, Pareto-optimal solutions that trade off the multiple objectives, such that they can choose a posteriori the solution that best meets their needs.

Over the last four decades, *constraint programming* has emerged as a fundamental technology for solving hard combinatorial problems, as it provides

⋆ Corresponding Authors.

rich languages to express combinatorial structures and to specify search procedures at a high level of abstraction [16]. Building on the strengths of this work, constraint-based algorithms and improvements to those algorithms have been proposed for finding Pareto-optimal solutions to MOCO problems [12, 8, 13, 15, 11]. These MOCO algorithms rely on modeling using constraint programming and on solving by repeated calls to a constraint solver to find feasible solutions.[3]

Understanding the properties of these constraint-based MOCO algorithms and analyzing their performance is an important problem. This is true both to understand their strengths and weaknesses, and also for the design of improved algorithms. Previous work has focused on empirical evaluations on benchmark instances. Such evaluations, while important, have their limitations, as any conclusions may not necessarily generalize to all instances. Our paper adopts a different, purely theoretical approach, which is based on characterizing the search space into subspaces and analyzing the worst-case performance of a MOCO algorithm in terms of an upper bound on the expected number of calls to the underlying solver to find each Pareto-optimal solution to a given MOCO instance. Our worst-case analysis holds for every MOCO instance and, in contrast to an average-case analysis, our bounds do not rely on any assumptions about the distribution of the input instances.

To determine the expected number of calls to a constraint solver, we build a general probability model that takes into account two uncertain factors: (a) how are all solutions distributed in the search space of a given MOCO instance, and (b) how likely is an arbitrary solution to be returned by the constraint solver. To address the first factor, we introduce a *good ordering property* that labels all solutions in the search space and identifies an important total-order relation on all solutions. To address the second factor, we introduce a *bounded bias assumption* where it is (weakly) assumed that for every solution $s$ in the search space, the probability that a call to the constraint solver returns $s$ is bounded from below and non-zero.

Our analysis framework reveals a deep connection between the search mechanism of the constraint solver and the exploration of the search space of a MOCO instance. In brief, if the probability that the solver returns a solution is bounded from below by $c/n$, where $n$ is the number of all solutions in the search space, for some constant $c$, a constraint-based MOCO algorithm $\mathcal{A}$ satisfying the good ordering property finds each Pareto-optimal solution in $O(\log n)$ expected calls to the solver. If $c$ is a function of $n$ and $c$ is in $\omega(\frac{\log n}{n})$—intuitively, $c$ grows asymptotically faster than $\frac{\log n}{n}$—$\mathcal{A}$ finds each Pareto-optimal solution in $o(n)$ expected calls to the solver, which is strictly better than the naive worst-case bound $O(n)$. Our study thus has implications for the best choice and design of the underlying constraint solver for a constraint-based MOCO solver.

We apply our framework to two important constraint-based MOCO algorithms: (i) Le Pape et al.'s [12] influential and widely-used algorithm for bi-objective optimization problems (see also [20] for an earlier proposal restricted

[3] From here after, we use *solution* unqualified to refer to a feasible solution and *Pareto-optimal solution* to refer to an optimal solution to a MOCO instance.

to a particular scheduling problem), and (ii) Rayside et al.'s [15] guided improvement algorithm (GIA), which has shown good performance empirically on several benchmark instances and has been incorporated into a widely-used system to support MOCO: the Z3 constraint solver, developed at Microsoft Research [2]. Both algorithms are designed to find one or more exact Pareto-optimal solutions. We prove that both algorithms satisfy the good ordering property and thus fit our analysis framework and theoretical results.

## 2 Notation and Preliminaries

In the context of constraint programming, a MOCO problem is a quadruple $\mathcal{P} = \langle X, D, C, F \rangle$, where $X = \{x_1, \ldots, x_L\}$ is a set of variables, $D = \{D_1, \ldots, D_L\}$ is a set of finite domains of variables in $X$, $C$ is a set of constraints on variables in $X$, and $F = \{f_1, \ldots, f_m\}$ is a set of $m$ objective functions to minimize simultaneously.[4] A solution $s$ to a MOCO problem $\mathcal{P}$ is a total assignment of all variables in $X$ to values in the corresponding domains, such that all constraints in $C$ are satisfied. For a combinatorial optimization problem [14], all solutions constitute a *finite* search space $\mathcal{S}$, and we denote $|\mathcal{S}|$ by $n$. Each objective function $f_i(\mathcal{S})$ assigns a discrete value to each solution $s \in \mathcal{S}$, and is bounded by the minimal and maximal values regarding the corresponding objective.

There are two classes of approaches to solving MOCO problems [7]: *a priori*, where weights or rankings are specified for the objectives prior to solving, and *a posteriori*, where a representative set of Pareto-optimal solutions are presented to a decision maker. Our interest is in *a posteriori* methods and, in particular, constraint-based MOCO algorithms for finding sets of Pareto-optimal solutions.

Given two solutions $s$ and $s'$ to a MOCO problem $\mathcal{P}$, we say that $s$ *dominates* $s'$, denoted by $s \prec s'$, if and only if $s$ is not worse than $s'$ regarding all objectives and $s$ is better than $s'$ regarding at least one objective:

$$\begin{aligned} &\forall i \in \{1, \ldots, m\} : f_i(s) \leq f_i(s') \ \textit{and} \\ &\exists j \in \{1, \ldots, m\} : f_j(s) < f_j(s') \end{aligned} \tag{1}$$

A solution to $\mathcal{P}$ is Pareto-optimal, which we denote by $\tilde{s}$, if and only if no other solution $s$ in $\mathcal{S}$ dominates $\tilde{s}$: $\nexists s \in \mathcal{S} : s \prec \tilde{s}$. All the Pareto-optimal solutions constitute the Pareto front $\tilde{\mathcal{S}}$, and we denote $|\tilde{\mathcal{S}}|$ by $p$.

## 3 Exact Constraint-Based MOCO Algorithms

A common approach to solving *single*-objective constraint optimization problems is to find an optimal solution by solving a sequence of constraint satisfaction problems [19, 1]. The idea is to, at each iteration, post an additional constraint that excludes solutions that are worse than the most recently found solution. This approach has been generalized in various ways to obtain constraint-based

[4] Without loss of generality, we consider minimization problems.

**Algorithm 1:** Le Pape et al. (1994) Algorithm

**input :** BOCO instance $\mathcal{P}$ with objectives $f_1$ and $f_2$
**output:** Pareto front $\tilde{\mathcal{S}}$

```
1  S̃ ← ∅
2  SupC_2 ← true
3  s ← solver(P)
4  while s ≠ null do                                   /* while a SAT call */
5      while s ≠ null do                               /* minimizing f_1 */
6          s' ← s
7          InfC ← f_1(S) ≥ f_1(s')
8          s ← solver(P ∧ ¬InfC ∧ SupC_2)
9      SupC_1 ← f_1(S) ≤ f_1(s')
10     while s' ≠ null do                              /* minimizing f_2 */
11         s ← s'
12         InfC ← f_2(S) ≥ f_2(s)
13         s' ← solver(P ∧ SupC_1 ∧ ¬InfC)
       /* s becomes Pareto-optimal                                        */
14     S̃ ← S̃ ∪ {s}
15     SupC_2 ← f_2(S) < f_2(s)
16     s ← solver(P ∧ SupC_2)
17 return S̃
```

algorithms for solving *multi*-objective problems. In this section, we examine in detail the two algorithms that we subsequently analyze.

The first algorithm we consider was proposed by Le Pape et al. [12], denoted by LePape from here after, which is a classical algorithm for bi-objective combinatorial optimization (BOCO) problems (see Algorithm 1). The idea is to find the optimal value for one of the objective functions, constrain its value, and restart the search to find the optimal value for the second objective function. The subroutine solver($\mathcal{P}$) (Line 3) indicates a call to a constraint solver for a solution to a MOCO problem $\mathcal{P}$. A call is SAT if the solver returns a solution successfully, and UNSAT otherwise. The inner loop in Lines 5–8 finds the optimal value for the objective function $f_1$: every time a solution $s$ is returned, the algorithm incrementally searches for a better solution than $s$ regarding $f_1$ by excluding all solutions scoped by the constraint $\mathit{InfC} = f_1(\mathcal{S}) \geq f_1(s)$, which indicates all $s' \in \mathcal{S}$, subject to $f_1(s') \geq f_1(s)$. Then, the algorithm constrains the optimal value regarding $f_1$ (Line 9) and finds the optimal value for the other objective function $f_2$ (Lines 10–13). A Pareto-optimal solution is found when there is no solution better than the currently-found solution regarding $f_2$ (Line 14). Next, the algorithm constrains the optimal value regarding $f_2$ (Line 15), implicitly retracts the constraint regarding $f_1$, and keeps searching for other Pareto-optimal solutions (the outer loop in Lines 4–16).

The second algorithm we consider was proposed by Rayside et al. [15], called guided improvement algorithm (GIA), which is designed for MOCO problems

(see Algorithm 2). On some benchmarks GIA has been shown empirically to outperform other constraint-based algorithms [8, 13] in terms of the actual running time of solving a given problem instance [15]. To formalize GIA, we introduce the following notation. According to the definition of Pareto dominance, a solution $s$ to a MOCO problem $\mathcal{P}$ partitions the original search space $\mathcal{S}$ into three subspaces: `inferior`, `superior`, and `incomparable`. Correspondingly, we define three types of constraints scoping these subspaces. The `inferior` constraint of a solution $s$ defines the `inferior` subspace leading to all solutions that are dominated by $s$:

$$\texttt{inf}(s) = \{s' \in \mathcal{S} : s \prec s'\}. \tag{2}$$

The `superior` constraint of solution $s$ defines the `superior` subspace leading to all solutions dominating $s$:

$$\texttt{sup}(s) = \{s' \in \mathcal{S} : s' \prec s\}. \tag{3}$$

The `incomparable` constraint of solution $s$ defines the `incomparable` subspace leading to all solutions that do not dominate $s$ and are not dominated by $s$ either:

$$\texttt{incp}(s) = \mathcal{S} \setminus \big(\texttt{inf}(s) \cup \texttt{sup}(s) \cup \{s\}\big). \tag{4}$$

GIA, at each iteration, uses the `superior` constraint of a newly-found solution, defined in Equation (3), to augment constraints for the next search. GIA always searches for the next solution only in the `superior` subspaces that lead to better solutions, regarding all objectives, than the ones already found. This results in inexpensive operations during the search as GIA only needs to keep track of the one solution that is currently the best. A Pareto-optimal solution is found when there is no solution in its `superior` subspace. Afterwards, GIA searches for other Pareto-optimal solutions in the `incomparable` subspace of the Pareto-optimal solution already found, defined in Equation (4).

Algorithm 2 lists the pseudo-code of GIA. The inner loop (Lines 5–8) is the procedure of searching for one Pareto-optimal solution: every time a solution $s$ is returned by a SAT call in the `superior` subspace of the previous solution, the current constraints are incrementally augmented by the `superior` constraint (denoted by $SupC$) of solution $s$. The outer loop (Lines 4–11) serves finding all Pareto-optimal solutions: every time an UNSAT call in the `superior` subspace of solution $s'$ is returned, solution $s'$ becomes a Pareto-optimal one and the constraints are incrementally augmented by the `incomparable` constraint (denoted by $IncpC$) of solution $s'$.

Theoretically, in the best case, LePape finds one Pareto-optimal solution using one SAT call and two UNSAT calls to a constraint solver, while GIA needs one SAT call and one UNSAT call to reach a Pareto-optimal solution. However, a naive analysis suggests that in the worst case, both algorithms may take $O(n)$ calls to find even one Pareto-optimal solution.

**Algorithm 2:** Guided Improvement Algorithm (GIA)

**input :** MOCO instance $\mathcal{P}$ with objectives $f_1, \ldots, f_m$
**output:** Pareto front $\tilde{\mathcal{S}}$

```
1  S̃ ← ∅
2  IncpC ← true
3  s ← solver(P)
4  while s ≠ null do                                   /* while a SAT call */
5      while s ≠ null do                               /* while a SAT call */
6          s' ← s
7          SupC ← getSupC(P, s)
8          s ← solver(P ∧ IncpC ∧ SupC)
           /* s' becomes Pareto-optimal                                    */
9      S̃ ← S̃ ∪ {s'}
10     IncpC ← IncpC ∧ getIncpC(P, s')
11     s ← solver(P ∧ IncpC)
12 return S̃
```

## 4 A Framework for Worst-Case Analysis

In this section, we propose an analysis framework for systematically investigating the worst-case performance of a constraint-based MOCO algorithm for finding each Pareto-optimal solution in terms of the expected number of calls to a constraint solver. In general, the performance of a constraint-based MOCO algorithm for finding a Pareto-optimal solution in a search space $\mathcal{S}$ of a given MOCO problem $\mathcal{P}$ is determined by the following two uncertain factors:

**Factor 1:** How are the solutions distributed in $\mathcal{S}$?
**Factor 2:** How likely is any particular solution in $\mathcal{S}$ to be returned by the solver?

Given an underlying constraint solver, a constraint-based MOCO algorithm that finds one or more Pareto-optimal solutions, denoted by $\mathcal{A}$, searches for a Pareto-optimal solution following a process that we call *improvement search*:

**Step 1 (constraint solving):** $\mathcal{A}$ asks the solver to return an arbitrary solution;
**Step 2 (decision making):** If no solution exists (an UNSAT call), the process terminates as all Pareto-optimal solutions have been found; if a solution $s$ is returned (a SAT call), go to Step 3;
**Step 3 (constraint improvement):** $\mathcal{A}$ delimits the next search space by augmenting the constraints using the currently-found solution $s$; go to Step 1.

Given a MOCO problem $\mathcal{P}$, the search space $\mathcal{S}$ with $n$ solutions to $\mathcal{P}$ is fixed. In all cases, an UNSAT call is returned at the end of the improvement search. If $n = 0$ (i.e., $\mathcal{P}$ has no solution), then no SAT call happens in $\mathcal{S}$. If $n = 1$, only one SAT call is required to reach the only solution in $\mathcal{S}$, which is Pareto-optimal as

well. If $n > 1$, a SAT call is required to return an arbitrary solution $s$ in $\mathcal{S}$, and subsequently the algorithm $\mathcal{A}$ defines the next search space using the constraints of solution $s$.

Let $Q \subseteq \mathcal{S}$ be any subspace of $\mathcal{S}$, which we call a *query space*. Let $target(s)$ be the *target* subspace defined by the algorithm $\mathcal{A}$ for the next search after finding solution $s$. Note that a target subspace, $target(s) \subseteq \mathcal{S}$, is a query space. Also, a query space can represent the original search space $\mathcal{S}$. For any query space $Q$, let $T(Q)$ be the expected number of SAT calls that $\mathcal{A}$ makes until it reaches any Pareto-optimal solution in $Q$, and $Pr[\text{solver}(Q) = s]$ the probability of returning solution $s$ in $Q$ by the solver. We define the following general probability model of $T(Q)$ when the first query space of algorithm $\mathcal{A}$ is $Q$:

**Definition 1 (General probability model).** *If $Q = \emptyset$ has no solution, then $T(Q) = 0$. If $Q \neq \emptyset$ contains at least one solution, then:*

$$T(Q) = \sum_{s \in Q} \Pr[\text{solver}(Q) = s] \cdot T\big(target(s)\big) + 1. \tag{5}$$

Consider Factor 1: the target subspace $target(s)$ is determined by the specific algorithm $\mathcal{A}$. Moreover, given a certain solution $s \in Q$, the number of solutions contained in $target(s)$ is uncertain. To address the uncertainty, we define the good ordering property of the algorithm $\mathcal{A}$ as follows:

**Definition 2 (Good ordering property).** *Given any query space $Q \subseteq \mathcal{S}$, let $|Q| = n$. The algorithm $\mathcal{A}$ has the good ordering property, if we can label all solutions $s_1, s_2, \cdots, s_n$, such that*

$$s_i \in target(s_j) \rightarrow i < j \tag{6}$$

*Remark 1.* By Equation (6), $target(s_i) \subseteq \{s_1, s_2, \cdots, s_{i-1}\}$, and thus $|target(s_i)| < i$.

For any integer $k \geq 0$, let $T(k)$ be the maximum of $T(Q)$ for any query space $Q$ that contains at most $k$ solutions:

$$T(k) = \max_{Q \subseteq \mathcal{S}: |Q| \leq k} T(Q). \tag{7}$$

*Remark 2.* By Definition (7), $T(k) \geq T(k-1)$ for every integer $k \geq 1$.

Consider Factor 2: the probability of returning any particular solution in a given query space by the underlying solver is uncertain. The probability is determined by the search mechanism designed in the solver, which varies for different solvers, depending, for example, on the amount of *randomization* used in the variable and value ordering heuristics adopted by the solvers [17, 9]. As an approximation of realistic constraint solvers, we make the following assumption regarding the probability. We merely suppose that the probability of returning any solution by the solver has a bounded bias:

**Definition 3 (Bounded bias assumption).** *Given a search space $\mathcal{S}$ including $n$ solutions to a given MOCO problem $\mathcal{P}$, there exists a non-zero parameter $c_n \leq 1$ such that for every $Q \subseteq \mathcal{S}$ and every solution $s \in Q$,*

$$\Pr[\text{solver}(Q) = s] \geq \frac{c_n}{|Q|}.$$

*Remark 3.* When the parameter $c_n = 1$, the solver returns any solution in $Q$ uniformly at random.

**Theorem 1.** *Let $\mathcal{A}$ be a constraint-based MOCO algorithm with the good ordering property and let $c_n$ be a parameter for which the bounded bias assumption holds for the underlying constraint solver used by $\mathcal{A}$. Given a search space $\mathcal{S}$ including $n$ solutions to a given MOCO problem $\mathcal{P}$, the expected number of SAT calls that $\mathcal{A}$ requires to find a Pareto-optimal solution is at most $(2 \log n)/c_n$.*

*Proof.* Given $\mathcal{S}$ including $n$ solutions to problem $\mathcal{P}$, we want to show that $T(\mathcal{S}) \leq \frac{2}{c_n} \log n$. To do so, it suffices to show that for any $k \leq n$, $T(k) \leq \frac{2}{c_n} \log k$.

Let $Q^* = \text{argmax}_{Q \subseteq \mathcal{S}: |Q| \leq k} T(Q)$. Then by Definition (7), $T(Q^*) = T(k)$. By the good ordering property and Remark 1, let $|Q^*| = \ell \leq k$ and label all the solutions in $Q^*$ by $s_1, \ldots, s_\ell$. Let $E$ denote the event where solver$(Q^*)$ returns one of the solutions in $\{s_1, \ldots, s_{\lceil k/2 \rceil}\}$. By the bounded bias assumption,

$$\Pr[E] = \sum_{i=1}^{\lceil k/2 \rceil} \Pr[\text{solver}(Q^*) = s_i] \geq \frac{k}{2} \cdot \frac{c_n}{\ell} \geq \frac{k}{2} \cdot \frac{c_n}{k} = \frac{c_n}{2}.$$

When event $E$ occurs, by Remark 1, the solver returns a solution $s$ such that $|target(s)| \leq \lceil k/2 \rceil - 1 \leq \lfloor k/2 \rfloor$ solutions; furthermore, by Remark 2, $T\big(target(s)\big) \leq \max_{Q \subseteq \mathcal{S}: |Q| \leq \lfloor k/2 \rfloor} T(Q) = T(\lfloor k/2 \rfloor)$. Likewise, when event $E$ does not occur, the solver returns a solution $s$ such that $|target(s)| \leq k - 1 < k$ solutions and $T\big(target(s)\big) < T(k)$. Therefore, we have that

$$\begin{aligned} T(Q^*) &\leq \Pr[E] \cdot T(\lfloor k/2 \rfloor) + (1 - \Pr[E]) \cdot T(k) + 1 \\ &\leq T(k) + \frac{c_n}{2}\big(T(\lfloor k/2 \rfloor) - T(k)\big) + 1. \end{aligned} \tag{8}$$

Since $T(Q^*) = T(k)$, the above inequality implies that

$$T(k) \leq T(\lfloor k/2 \rfloor) + \frac{2}{c_n}.$$

Hence, $T(k) \leq \frac{2 \log k}{c_n}$ and thus $T(\mathcal{S}) \leq T(n) \leq \frac{2 \log n}{c_n}$.

**Corollary 1.** *In Theorem 1, if parameter $c_n$ is a constant, the algorithm $\mathcal{A}$ finds each Pareto-optimal solution in $O(\log n)$ expected SAT calls; if $c_n$ is a function of $n$ and $c_n = \omega(\log n / n)$, $\mathcal{A}$ finds each Pareto-optimal solution in $o(n)$ expected SAT calls.*

According to Definition 3 and Corollary 1, if parameter $c_n$ is a positive constant $c \leq 1$, then the probability of returning any solution $s$ in the search space $\mathcal{S}$ of all $n$ solutions is not less than $\frac{c}{n}$, i.e., $\Pr[\text{solver}(\mathcal{S}) = s] \geq \frac{c}{n}$; in such a case, $T(n)$ reaches an ideal logarithmic bound $O(\log n)$. Note that if $c = 1$, then the solver returns any solution uniformly at random, i.e., $\Pr[\text{solver}(\mathcal{S}) = s] = \frac{1}{n}$. If parameter $c_n$ is a function bounded in $\omega(\frac{\log n}{n})$, then for any positive constant $c$, there exists a positive constant $n_0$ such that $0 \leq \frac{c \log n}{n} < c_n$ for all $n \geq n_0$, and the probability of returning any solution $\Pr[\text{solver}(\mathcal{S}) = s] > \frac{c \log n}{n^2}$; in such a case, $T(n) < \frac{2n}{c}$ is bounded in $o(n)$. Intuitively, if $c_n$ grows asymptotically faster than $\frac{\log n}{n}$, then $T(n)$ is bounded in $o(n)$, which is strictly (i.e., asymptotically) better than the naive worst-case bound $O(n)$.

## 5 Application of the Framework

To apply the proposed analysis framework to a certain MOCO algorithm, one has to check if the algorithm meets the good ordering property and if the underlying constraint solver used by the algorithm satisfies the bounded bias assumption. The bounded bias assumption embodies and relaxes an active research point of generating uniformly-distributed solutions (Remark 3) [6, 10], and it only needs a lower bound on the probability of returning any solution. Moreover, our analysis proves that not only uniformly random generators but also bounded bias generators are ideal. However, it is non-trivial to design a constraint solver that works efficiently and simultaneously guarantees either rigorous the uniformity or bounded bias assumptions. A state-of-the-art scalable generator supports *near-uniformity* [4] or *almost-uniformity* [3, 5], which also guarantees the bounded bias assumption, but only for Boolean satisfiability (SAT) problems. For general constraint-satisfaction problems (CSPs), a uniformly random generator has been proposed but suffers from exponential complexity [6]. In practice, we believe that an efficient generator that approximately guarantees the bounded bias assumption might be sufficient, e.g., by designing randomization heuristics based on variable and value ordering, which would be explored in future. In this section, we apply the proposed framework to two constraint-based optimization algorithms. We suppose that the underlying solver used in the algorithms satisfies the bounded bias assumption, and we prove that the good ordering property holds for the algorithms.

### 5.1 LePape

According to Algorithm 1, LePape essentially performs two improvement search processes to find one Pareto-optimal solution: the first process (Lines 5–8) minimizes one of the objective functions $f_1$, and the second one (Lines 10–13) minimizes the objective function $f_2$. In each process, the search retains the constraints of the optimal value for one of the objective functions and incrementally finds a better solution in the target subspace of the currently-found one regarding the

other objective function. For example, in the first process, $target(s)$ retains the constraint $SupC_2$ for $f_2$ and scopes all solutions constrained by $f_1(\mathcal{S}) < f(s)$ (i.e., $\neg InfC$ in Line 8); while in the second process, $target(s)$ retains the constraint $SupC_1$ for $f_1$ (Line 9) and scopes all solutions constrained by $f_2(\mathcal{S}) < f(s)$ (i.e., $\neg InfC$ in Line 13). According to the characteristics of the target subspaces in two processes, we have the following claim:

*Claim.* The good ordering property holds for LePape.

*Proof.* The good ordering property holds when the condition (6) is satisfied. To label all $n$ solutions in the entire search space, we perform a "bucket sorting": firstly, suppose that there are $n_1$ distinct values for one of objective functions $f_1$, we partition all solutions into $n_1$ buckets, each of which contains all solutions with the same value for $f_1$, and we sort buckets in the ascending order of the values for $f_1$; secondly, we sort all solutions inside each bucket in the ascending order of the values for the other objective function $f_2$; finally, we label all solutions from index 1 to $n$ firstly following the bucket order and secondly following the solution order inside each bucket, with breaking ties arbitrarily. For example, after the above bucket sorting, the first bucket of solutions are labeled as $s_1, \cdots, s_{|b_1|}$ ($b_1$ is the size of the first bucket) that have the minimum value for $f_1$ and are ordered ascendingly by their values regarding $f_2$.

Following the above labeling tactic, given a solution $s_j$ with the bucket index $b$, in the first search process that minimizes $f_1$, any solution $s_i \in target(s_j)$ must belong to a bucket with a smaller index than $b$; while in the second search process that minimizes $f_2$, any solution $s_i \in target(s_j)$ must be a solution that has a smaller index than $j$ in the same bucket as $s_j$. Thus, in both improvement search processes of LePape, we have $i < j$.

By Claim 5.1 and the bounded bias assumption, Theorem 1 and Corollary 1 hold for LePape. Moreover, LePape has two improvement search processes, and it requires at least one SAT call and two UNSAT calls to identify one Pareto-optimal solution. Hence, LePape finds all $p$ Pareto-optimal solutions using at most $(4p \log n)/c_n$ SAT calls and at most $2p$ UNSAT calls. In addition, the Pareto-optimal solutions founded by LePape follows a certain order, i.e., the ascending order of values regarding either of two objectives.

### 5.2 GIA

According to Algorithm 2, GIA performs the improvement search process once to reach some Pareto-optimal solution (Lines 5–8), and we have:

*Claim.* The good ordering property holds for GIA.

*Proof.* For GIA, the target subspace $target(s)$ of the currently-found solution $s$ is the superior subspace $\texttt{sup}(s)$. If we label all $n$ solutions in the entire search space, such that

$$|\texttt{sup}(s_1)| \leq |\texttt{sup}(s_2)| \leq \cdots \leq |\texttt{sup}(s_n)| \tag{9}$$

then for every $i = 1, \ldots, n$, we have

$$|\mathtt{sup}(s_i)| < i. \tag{10}$$

To prove Equation (10), by contradiction, suppose that all solutions have been indexed following the rule defined in Equation (9) and that there is a solution $s_i$ for which $|\mathtt{sup}(s_i)| \geq i$. Then, there must exist some index $j > i$, such that $s_j \in \mathtt{sup}(s_i)$. But then $\mathtt{sup}(s_j) \subseteq \mathtt{sup}(s_i) \setminus \{s_j\}$ and so $|\mathtt{sup}(s_j)| \leq |\mathtt{sup}(s_i)| - 1$, in contradiction to Equation (9). According to Equation (10), any solution $s_i \in \mathtt{sup}(s_j)$ must have a smaller index than $j$, i.e., $i < j$.

By Claim 5.2 and the bounded bias assumption, Theorem 1 and Corollary 1 hold for GIA. Moreover, GIA requires at least one SAT call and at least one UNSAT calls to identify one Pareto-optimal solution. Hence, GIA finds all $p$ Pareto-optimal solutions using at most $(2p \log n)/c_n$ SAT calls and at most $p$ UNSAT calls. In addition, the Pareto-optimal solutions founded by GIA does not follow a certain order, which is different from LePape.

## 6 Conclusion

We presented the first theoretical analysis of the worst-case performance of two constraint-based MOCO algorithms. The algorithms rely on modeling using constraint programming and on solving the MOCO problem by repeated calls to an underlying constraint solver. We characterized the original search space into subspaces during the search and developed a general probability model of $T(n)$, the expected number of (SAT) calls to the underlying constraint solver that the algorithms require to find each Pareto-optimal solution to a given MOCO problem. We identified a total-order relation on all solutions by introducing a good ordering property. Under only a (weak) bounded bias assumption—the probability that a call to the underlying solver returns any particular solution is bounded from below and non-zero, we proved that $T(n)$ is bounded in $O(\log n)$ or $o(n)$, determined by a parameter $c_n$ that depends on how the underlying solver behaves for the MOCO problem.

Our analysis reveals the connection between the search mechanism of a constraint solver and the exploration of the search space of a MOCO problem. Our study has implications for the best choice and design of the underlying constraint solver for a constraint-based MOCO solver. In brief, the underlying constraint solver used in a constraint-based MOCO solver should randomize the generation of feasible solutions, ideally meeting the uniformly random or bounded bias assumption, such that the MOCO solver is able to find each Pareto-optimal solution in a bound strictly better than the naive worst-case $O(n)$ .

Some extensions to our current analysis framework would be considered in future. We plan to perform a worst-case analysis of other constraint-based MOCO algorithms. Furthermore, we plan to investigate a smoothed analysis [18] of constraint-based MOCO algorithms.

## Acknowledgments

This work has been partially supported by Shanghai Municipal Natural Science Foundation (No. 17ZR1406900) and NSERC Discovery Grant.

## References

1. Baptiste, P., Le Pape, C., Nuijten, W.: Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems. Kluwer (2001)
2. Bjørner, N., Phan, A.D.: $\nu$Z - maximal satisfaction with Z3. In: Proc. SCSS. pp. 632–647 (2014)
3. Chakraborty, S., Fremont, D.J., Meel, K.S., Seshia, S.A., Vardi, M.Y.: Distribution-aware sampling and weighted model counting for SAT. In: Proc. AAAI. pp. 1722–1730 (2014)
4. Chakraborty, S., Meel, K.S., Vardi, M.Y.: A scalable and nearly uniform generator of sat witnesses. In: Proc. CAV. pp. 608–623 (2013)
5. Chakraborty, S., Meel, K.S., Vardi, M.Y.: Balancing scalability and uniformity in sat witness generator. In: Proc. DAC. pp. 1–6 (2014)
6. Dechter, R., Kask, K., Bin, E., Emek, R.: Generating random solutions for constraint satisfaction problems. In: Proc. AAAI. pp. 15–21 (2002)
7. Ehrgott, M.: Multicriteria Optimization. Springer, 2nd edn. (2005)
8. Gavanelli, M.: An algorithm for multi-criteria optimization in CSPs. In: Proc. ECAI. pp. 136–140 (2002)
9. Gomes, C., Selman, B., Kautz, H.: Boosting combinatorial search through randomization. In: Proc. AAAI. pp. 431–437 (1998)
10. Gomes, C.P., Sabharwal, A., Selman, B.: Near-uniform sampling of combinatorial spaces using xor constraints. In: Proc. NIPS. pp. 481–488 (2006)
11. Hartert, R., Schaus, P.: A support-based algorithm for the bi-objective pareto constraint. In: Proc. AAAI. pp. 2674–2679 (2014)
12. Le Pape, C., Couronné, P., Vergamini, D., Gosselin, V.: Time-versus-capacity compromises in project scheduling. In: Proceedings of the Thirteenth Workshop of the UK Planning Special Interest Group. Strathclyde, UK (1994)
13. Lukasiewycz, M., Glaß, M., Haubelt, C., Teich, J.: Solving multi-objective pseudo-boolean problems. In: Proc. SAT. pp. 56–69 (2007)
14. Papadimitriou, C., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Dover (1998)
15. Rayside, D., Estler, H.C., Jackson, D.: The guided improvement algorithm for exact, general purpose, many-objective combinatorial optimization. Tech. rep., MIT-CSAIL-TR-2009-033 (2009)
16. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming. Elsevier (2006)
17. Sadeh, N., Fox, M.: Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. Artificial Intelligence 86(1), 1–41 (1996)
18. Spielman, D., Teng, S.H.: Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. J. ACM 51(3), 385–463 (2004)
19. Van Hentenryck, P.: Constraint Satisfaction in Logic Programming. MIT Press (1989)
20. van Wassenhove, L., Gelders, L.: Solving a bicriterion scheduling problem. Eur. J. Oper. Res. 4(1), 42–48 (1980)