Slide&Drill, a New Approach for Multi-Objective Combinatorial Optimization

João Cortes
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal

Inês Lynce
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal

Vasco Manquinho
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal

Abstract

Following the successful use of Propositional Satisfiability (SAT) algorithms in Boolean optimization (e.g., Maximum Satisfiability), several SAT-based algorithms have been proposed for Multi-Objective Combinatorial Optimization (MOCO). However, these new algorithms either provide a small subset of the *Pareto front* or follow a more exploratory search procedure and the solutions found are usually distant from the Pareto front.

We extend the state of the art with a new SAT-based MOCO solver, Slide and Drill (`Slide&Drill`), that hones an *upper bound set* of the exact solution. Moreover, we show that `Slide&Drill` neatly complements proposed UNSAT-SAT algorithms for MOCO. These algorithms can work in tandem over the same shared "blackboard" formula, in order to enable a faster convergence.

Experimental results in several sets of benchmark instances show that `Slide&Drill` can outperform other SAT-based algorithms for MOCO, in particular when paired with previously proposed UNSAT-SAT algorithms.

2012 ACM Subject Classification Computing methodologies → Optimization algorithms

Keywords and phrases Multi-Objective Combinatorial Optimization, Satisfiability Algorithms

Digital Object Identifier 10.4230/LIPIcs.CP.2024.8

Funding This work was supported by Portuguese national funds through FCT, under projects UIDB/50021/2020 (DOI: 10.54499/UIDB/50021/2020), PTDC/CCI-COM/2156/2021 (DOI: 10.54499/-PTDC/CCI-COM/2156/2021) and 2022.03537.PTDC (DOI: 10.54499/2022.03537.PTDC).

1 Introduction

In real-world problems it is common to have several objective functions to optimize [17, 19, 29]. For instance, when updating a system such as a Linux installation [13], one can try to maximize the number of packages to be updated from the current version to the most recent one, while at the same time minimizing the number of software packages from the current installation to be removed in the update process. It is usually the case that the objective functions are conflicting, i.e., decreasing one objective function results in having to increase the value of another objective function. Hence, in Multi-Objective Combinatorial Optimization (MOCO), the goal is to try to find all Pareto-optimal solutions, i.e., all solutions for which one cannot improve the value of a function without worsening the value of another one. The set of all Pareto-optimal solutions is known as the Pareto front.

Following the success of Propositional Satisfiability (SAT) algorithms in Boolean optimization problems such as Maximum Satisfiability (MaxSAT) [2] or Pseudo-Boolean Optimization (PBO) [24], several algorithms for MOCO have been proposed based on iterative calls to a satisfiability solver [10, 22, 28, 26]. For instance, the Guided-Improvement Algorithm (GIA) [22] starts with a feasible solution and iteratively checks if there is some other solution that is better on all objective functions. When the iterative process ends, the algorithm has found a Pareto-optimal solution and new constraints are added so that only assignments that improve on at least one objective function are feasible (i.e., solutions that are worse on all objectives are blocked). More recently, the notion of P-minimal models [26] was introduced, where this blocking is done using a propositional clause.

The issue with GIA and P-minimal algorithms is that the search process is focused on iteratively improving upon one solution until a Pareto-optimal solution is found. Considering that the set of solutions in the Pareto front can be large, in many instances, these algorithms are only able to find a very small subset of the Pareto front within a given time limit. Moreover, it can be the case that the Pareto-optimal solutions are skewed to optimize some objective function and do not provide a broad representation of solutions in the Pareto front.

This paper proposes `Slide&Drill`, a new exact [9], generic algorithm for MOCO that maintains an *upper cover* of the Pareto front, made of feasible of solutions. `Slide&Drill` repeatedly selects a point from the cover to improve upon. This improvement starts with a *drill* operation followed by a series of *slide* operations that generate another upper cover that is closer to the Pareto front. Hence, at any point of time, one can obtain a diversified set of solutions that approximate the Pareto front. Experimental results on representative sets of MOCO instances show that `Slide&Drill` provides better approximations of the Pareto front than previous SAT-based MOCO solvers since `Slide&Drill` is able to find a more diverse set of solutions for the end user.

The paper is organized as follows. Section 2 formally defines the MOCO problem and provides an overview of previous SAT-based MOCO algorithms. Section 3 defines lower and upper bound sets. Section 4 introduces the new `Slide&Drill` algorithm for MOCO based on iterative refinement of an upper bound set. Additionally, Section 4 also proves the correctness of the `Slide&Drill` algorithm and shows how to pair it in tandem with other MOCO algorithms. Section 5 explores different configurations of the `Slide&Drill` algorithm and compares it against other state-of-the-art SAT-based MOCO solvers using three different metrics. Finally, the paper concludes in Section 6.

2 Preliminaries

We start with the definitions that fall under SAT's domain. Next, we introduce the definitions specific to MOCO. Moreover, we briefly review previous approaches to solving MOCO.

2.1 Boolean Satisfiability

▶ **Definition 1** (CNF Formula). *Let $V = \{x_1, \ldots, x_n\}$ denote a set of n Boolean variables. A literal is either a variable $x_i \in V$ or its negation $\bar{x}_i$. A clause is a disjunction of literals. A formula in Conjunctive Normal Form (CNF) ϕ is a conjunction of clauses.*

An *assignment* or *model* ν defines a truth value for all variables. Let $\nu(x_i)$ denote the truth value of variable x_i and let $\nu(l_i)$ denote the truth value of a literal l_i. We have $\nu(l_i) = \top$ if $l_i = x_i$ and $\nu(x_i) = \top$, or if $l_i = \bar{x}_i$ and $\nu(x_i) = \bot$. Otherwise, we have $\nu(l_i) = \bot$. A clause c is satisfied if at least one of its literals is true. An assignment ν is said to satisfy a formula ϕ if it satisfies all its clauses. We extend the notation of assignments to define the truth value of a clause c and a CNF formula ϕ as $\nu(c)$ and $\nu(\phi)$, respectively. In the remainder of the paper, we use the set notation for formulas (set of clauses, meaning its conjunction) and clauses (set of literals, meaning its disjunction).

▶ **Definition 2** (Boolean Satisfiability (SAT)). *Given a CNF formula ϕ, the Boolean Satisfiability (SAT) problem is to decide if there is any assignment ν to the variables in ϕ that satisfies it or prove that no such assignment exists.*

Let ϕ be a CNF formula and α a set of unitary clauses. A SAT solver call is denoted by ϕ-$\texttt{SAT}(\alpha)$, and its value decides the satisfiability of $\phi \cup \alpha$, i.e., it checks the satisfiability of ϕ assuming all literals in α are true. Note that if $\alpha = \emptyset$, then the solver checks the satisfiability of ϕ. If the query is satisfiable, then the call returns a satisfiable model. Otherwise, it returns a null value, written as $\emptyset$.

2.2 Single and Multi-Objective Combinatorial Optimization

▶ **Definition 3** (Linear Pseudo-Boolean function and Pseudo-Boolean formulas). *A linear*[1] pseudo-Boolean (PB) function $f : \{0, 1\}^n \to \mathbb{N}$ *computes a weighted sum of its literals,*

$$f(\boldsymbol{x}) = f(x_1 \dots x_n) = \sum_{i=1}^{n} w_i l_i \quad , w_i \in \mathbb{N},\ x_i \in V,\ l_i \in \{x_i, \bar{x}_i\}. \tag{1}$$

Pseudo-Boolean constraints *generalize propositional clauses, and can be written as $f(\boldsymbol{x}) \bowtie k$, $\bowtie \in \{\leq, \geq, =\}$. A PB formula is a conjunction of PB constraints.*

▶ **Definition 4** (Pseudo-Boolean Optimization (PBO)). *Given a PB formula ϕ, an assignment ν is said* (ϕ-)feasible *if it satisfies all constraints in ϕ. Given a PB formula ϕ and a PB function f to minimize, the goal of Pseudo-Boolean Optimization (PBO) is to find an assignment ν that satisfies ϕ and minimizes the value of $f(\boldsymbol{x})$, where $\boldsymbol{x} \equiv (\nu(x_1), \dots, \nu(x_n))$.*

Next, we generalize PBO to the multi-objective case. Multi-objective optimization builds upon a criterion of comparison (or order) of tuples of numbers. This paper uses the *Pareto order or dominance.*

▶ **Definition 5** (Pareto partial order ($\prec$)). *Let Y be some subset of $\mathbb{N}^n$. For any $\boldsymbol{y}, \boldsymbol{y}' \in Y$,*

$$\boldsymbol{y} \preceq \boldsymbol{y}' \iff \forall i, \boldsymbol{y}_i \leqslant \boldsymbol{y}'_i,$$
$$\boldsymbol{y} \prec \boldsymbol{y}' \iff \boldsymbol{y} \preceq \boldsymbol{y}' \wedge \boldsymbol{y} \neq \boldsymbol{y}'.$$

We say $\boldsymbol{y}$ dominates *$\boldsymbol{y}'$ iff $\boldsymbol{y} \preceq \boldsymbol{y}'$. We say $\boldsymbol{y}$* strictly-dominates *$\boldsymbol{y}'$ iff $\boldsymbol{y} \prec \boldsymbol{y}'$.*

Given a tuple of objective functions sharing a common domain X, we can compare two elements $\boldsymbol{x}, \boldsymbol{x}' \in X$ by comparing the corresponding tuples in the objective space. We use the term *multi-objective function* to denote an array of functions.

▶ **Definition 6** (Pareto Dominance ($\prec$)). *Let $F : X \to Y \subseteq \mathbb{N}^n$ be a* multi-objective function, *mapping the* decision space *X into the* objective space *Y. For any $\boldsymbol{x}, \boldsymbol{x}' \in X$,*

$$\boldsymbol{x} \prec \boldsymbol{x}' \iff F(\boldsymbol{x}) \prec F(\boldsymbol{x}'),$$
$$\boldsymbol{x} \preceq \boldsymbol{x}' \iff F(\boldsymbol{x}) \preceq F(\boldsymbol{x}').$$

We say $\boldsymbol{x}$ dominates *$\boldsymbol{x}'$ iff $\boldsymbol{x} \preceq \boldsymbol{x}'$. We say $\boldsymbol{x}$* strictly-dominates *$\boldsymbol{x}'$ iff $\boldsymbol{x} \prec \boldsymbol{x}'$.*

[1] Note We will drop the *linear* qualifier hereafter, as we will only work with linear functions and constraints.

As a consequence of this comparison criterion, different *optimal* solutions may be mapped to different points in the objective space, which does not happen in the single objective case. Therefore, the solution to the problem is actually a set, called *Pareto front*. These solutions are optimal in the sense that for each, there is no other feasible solution that strictly dominates them.

▶ **Definition 7** (Pareto front). *Given a a multi-objective function $F : X \to Y$ and a* feasible space *$Z \subseteq X$, the* Pareto front *of Z is a subset $P \subseteq Z$ containing all elements that are not strictly-dominated,*

$$P = \{\, \boldsymbol{x} \in Z : \nexists \boldsymbol{x}' \in Z : \boldsymbol{x}' \prec \boldsymbol{x} \,\} .$$

Let the image front of Z*, or simply* front of Z*, be the unique subset $\overline{Y} \subseteq Y$ that is the image of P under F,*

$$\overline{Y} \equiv \operatorname{front}_Z F = \{\, \boldsymbol{y} \in Y : \exists \boldsymbol{x} \in P : \boldsymbol{y} = F(\boldsymbol{x}) \,\} .$$

Finally, let argument front of Z*,denoted by* $\arg\operatorname{front}_Z$*, be any subset $\overline{Z}$ of the Pareto Front P that is mapped under F into $\overline{Y}$ in a one-to-one fashion.*

▶ **Definition 8** (Multi-Objective Combinatorial Optimization (MOCO)). *Let $F : X \to Y \subseteq \mathbb{N}^n$ be a* multi-objective PB function*, mapping the* decision space *$X \subseteq \{\,0, 1\,\}^n$ into the* objective space *Y. Let $Z \subseteq X$ be the image under $\nu \mapsto \boldsymbol{x} = \nu(V) \equiv (\nu(x_1), \dots, \nu(x_n))$ of the feasible space of a PB formula ϕ, with variables in V.*

The goal of MOCO is to find a $\operatorname{front}_\phi F \equiv \operatorname{front}_{Z(\phi)} F$, i.e., the complete set of non-dominated objective points $\boldsymbol{y} \in Y$ whose preimage under F is ϕ-feasible. A MOCO instance will be denoted by the triple $\langle \phi, V, F \rangle$.

A remark: most applications require the production of $\arg\operatorname{front}_Z F$, which is one of the preimages under F of $\operatorname{front}_Z F$. Our non-standard choice was made bearing in mind the clarity of the discussion and of the algorithms's presentation. In any case, the implementation of the algorithms returns an $\arg\operatorname{front}_{Z(\phi)} F$, as usual. The pseudo-code can be adapted to do the same, but it will get significantly clobbered without adding much in the way of ideas.

▶ **Example 9.** Let $\langle \phi, V, F \rangle$ denote a MOCO instance defined over $V = \{x_1, x_2, x_3\}$, with two objective functions to minimize $F = (f_1, f_2)$ where $f_1(\boldsymbol{x}) = 2x_1 + x_2$, $f_2(\boldsymbol{x}) = 2\bar{x}_2 + 2x_3$ and $\phi = \{x_1 + x_2 + x_3 \geq 2\}$. In this case, there are two Pareto-optimal solutions: $\nu_1 = \{(x_1, 0), (x_2, 1), (x_3, 1)\}$ with costs $(1, 2)$ and $\nu_2 = \{(x_1, 1), (x_2, 1), (x_3, 0)\}$ with costs $(3, 0)$. Note that ν_1 provides a better value for f_1, while ν_2 is able to improve on f_2. All other satisfiable assignments to ϕ are dominated by either $\nu_1(V)$ or $\nu_2(V)$.

2.3 Encoding of Pseudo-Boolean Functions

In several SAT-based optimization algorithms, PB objective functions are encoded into CNF [8, 24]. In MOCO, we are interested in blocking feasible solutions that are dominated by some other feasible solution. In order to achieve this goal, one can use *unary counter* [3, 15, 16] encodings.

▶ **Definition 10** (Unary Counter). *Let $f_i : \{\,0, 1\,\}^n \to \mathbb{N}$ be a PB function and set V be an ordered set of variables that parametrize the domain of f_i,*

$$V = \{\, x_1, \dots, x_n \,\}, f_i(\boldsymbol{x}) = f_i(x_1, \dots, x_n) \tag{2}$$

Algorithm 1 `P-Minimal` Algorithm.

Input : $\langle \phi, V, F \rangle$ `// MOCO instance`
Output : $\text{front}_\phi\, F$ `// one img-front`
1 $(\widetilde{\phi}, O) \leftarrow$ `EncodeCNF`(F, V) `// build unary counters`
2 $\phi \leftarrow \phi \cup \widetilde{\phi}$
3 $I \leftarrow \emptyset$
4 $\nu' \leftarrow \phi$-`SAT`$(\emptyset)$ `// find first feasible model`
5 **while** $\nu' \neq \emptyset$ **do**
6   **while** $\nu' \neq \emptyset$ **do**
7         $\boldsymbol{x} \leftarrow \nu'(V)$, $\boldsymbol{y} \leftarrow F(\boldsymbol{x})$
8         $\alpha \leftarrow \left\{ \left\{ \bar{o}_{i,\boldsymbol{y}_i+1} \right\}, 1 \leqslant i \leqslant m \right\}$
9         $c \leftarrow \left\{ \bar{o}_{i,\boldsymbol{y}_i}, 1 \leqslant i \leqslant m \right\}$
10         $\phi \leftarrow \phi \cup \{ c \}$ `// block region dominated by` $\boldsymbol{y}$
11         $\nu' \leftarrow \phi$-`SAT`(α) `// look for` $\boldsymbol{y}'$ `that dominates` $\boldsymbol{y}$
12   **end**
13     $I \leftarrow I \cup \{ \boldsymbol{y} \}$ `// save optimal solution` $\boldsymbol{y}$
14     $\nu' \leftarrow \phi$-`SAT`$(\emptyset)$ `// find new non-dominated solution`
15 **end**
16 **return** I

Consider the CNF formula $\widetilde{\phi}$ with variables $V \cup O$, where $V \cap O = \emptyset$ and O contains one variable $o_{i,k}$ for each value $k \in \mathbb{N} : \exists \boldsymbol{x} : k = f_i(\boldsymbol{x})$. The elements of O are the order variables. *We call the tuple $\left\langle f_i, V, O, \widetilde{\phi} \right\rangle$ an unary counter of f_i iff all feasible models ν of $\widetilde{\phi}$ satisfy*

$$f_i(\boldsymbol{x}) \geqslant k \Leftrightarrow o_{i,k}, \quad \boldsymbol{x} = \nu(V). \tag{3}$$

2.4 SAT-based algorithms for MOCO

One approach for solving MOCO is through Minimal Correction Subset (MCS) enumeration since all Pareto-optimal solutions are MCSs of the MOCO formula [28]. After enumerating the formula's MCSs, one can filter out the non-optimal solutions. The main advantage of the MCS enumeration is that it is not necessary to encode the objective functions into CNF since, in some cases, the encoding of objective functions can dominate the size of the resulting CNF formula [8].

Soh et al. [26] show that with a unary representation of the objective functions (see section 2.3), it is possible to establish a one-to-one correspondence between the P-minimal models and Pareto-optimal solutions of a MOCO instance.

Algorithm 1 illustrates the `P-Minimal` algorithm. It starts by finding any feasible solution (line 4). Next, it iteratively improves that solution until a Pareto-optimal solution is found (lines 6-12). Each time a new solution is found, all dominated solutions are blocked using a single clause (line 10). Afterwards, the process repeats if there are other non-dominated solutions (line 14). Otherwise, the algorithm ends and returns the Pareto front (line 16).

The `P-Minimal` algorithm can be seen as a particular case of the Guided-Improvement Algorithm (GIA) [22]. The algorithm structure is the same, but `P-Minimal` uses a single clause to block dominated solutions instead of a disjunction of PB constraints. Recently, new UNSAT-SAT and Hitting Set-based algorithms have also been proposed [5] and can be seen as a generalization of core-guided Maximum Satisfiability (MaxSAT) algorithms for MOCO. Other adaptations of MaxSAT techniques have been proposed for MOCO [14, 12], including preprocessing techniques [11].

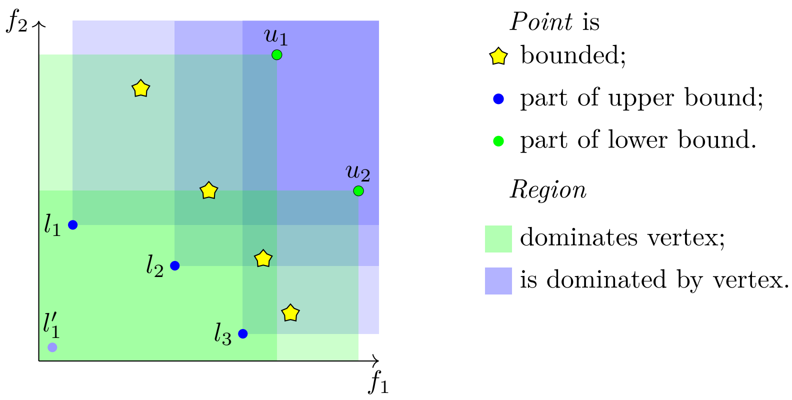


Figure 1 Bound sets (Definition 13) of some starred set $\overline{Y}$. The points $\{ l_1, l_2, l_3 \}$ form a *lower bound set* L. Dropping l_1 breaks *coverage*. This lower bound set is also *thin*, and adding l'_1 would make it "thick". The singleton set $\{ l'_1 \}$ is also a thin, lower bound set, not only of $\overline{Y}$ but also of L. The points $U = \{ u_1, u_2 \}$ form a thin, *upper bound set* of $\overline{Y}$. Note how it implies that U is an upper set of L too. And, by the same token, of $\{ l'_1 \}$. The set $\overline{Y}$ could be the image front of some MOCO instance.

3 Upper and Lower Bound Sets

Given that the Pareto order is just a *partial order* in the mathematical sense, there is no warrant to expect the existence of a *least element* of the feasible objective space. At the same time, the Pareto order reduces to the canonical *total order* of the integers when there is only one objective. The generalization of the order requires a generalization of the concept of "bounds". In particular, it is useful to deal in *bound sets* (Definition 13) that can contain more than one element.

We consider two different comparison predicates over sets. Let A and B be any two sets of points in the objective space. Then, 1) is A a *lower/upper cover* of B?, and 2) is A *a lower/upper bound set of* B?

▶ **Definition 11** (*Lower* and *upper covers*). *Let A and B be subsets of some decision space X, equipped with a multi-objective function F. Then, A* covers B from below*, or A* is a lower cover of B*, iff every element of B is dominated by some element of A,*

$$\forall b \in B, \exists a \in A : a \preceq b.$$

A strictly covers B*, or A* is a strict lower-cover of B*, iff*

$$\forall b \in B, \exists a \in A : a \prec b.$$

Also, we define an upper cover *analogously. In particular, B is an* upper cover *of A iff for every element of A there is some element of B that is dominated,*

$$\forall a \in A, \exists b \in B : a \preceq b.$$

The strict version trivially follows.

▶ **Definition 12** (Thin/thick sets). *A set A is* thin *if it does not contain distinct comparable elements,*

$$\neg\exists a_1, a_2 \in A : a_1 \neq a_2 \wedge a_1 \preceq a_2 \tag{4}$$

Otherwise, A is thick.

▶ **Definition 13** (Lower and upper bound sets). *Let L, U and Z be subsets of some decision space X^2, equipped with a multi-objective function F. $L \subseteq X$ is a* (strictly) lower bound set of $Z \subseteq X$ *iff L (strictly) covers Z from below and L is thin. If L is a lower bound set of Z, we say $L \preceq Z$. If it is a strictly lower bound set, we say $L \prec Z$.*

$U \subseteq X$ is a (strictly) upper bound set of $Z \subseteq X$ *iff U (strictly) covers Z from above and U is thin. If U is an upper bound set of Z, we say $U \succ Z$. If it is a strictly upper bound set, we say $U \succeq Z$.*

Figure 1 provides examples of lower and upper bound sets. Let the starred points correspond to the optimal front in the objective space. Any optimal element in the front will be dominated by at least one element of any lower bound set (e.g., $\{l_1, l_2, l_3\}$). Similarly, any element of the front dominates at least one element of the upper bound set (e.g., $\{u_1, u_2\}$).

Let u_{max} be the *maximal point*, that is, the point whose coordinates are the largest values of each objective. Then, the singleton set $\{ u_{max} \}$ is clearly an upper bound set, although not necessarily satisfiable. Analogously, the singleton set containing the origin is a lower bound set.

By computing a *satisfiable* upper bound set, we get an approximated view of the real front. If we improve this upper bound set slowly but surely, we will eventually stop, given a sufficient amount of time. At that point, the upper bound set coincides with the front.

4 `Slide&Drill`, an Upper-Bound Set Improver

We propose a new algorithm for MOCO, named `Slide&Drill` (Algorithm 2). Like `P-Minimal`, it is a SAT-UNSAT algorithm backed by a SAT oracle. By design, `P-Minimal` drills down the objective space, following a "greedy" path to optimal solutions. In contrast, `Slide&Drill` is a comprehensive algorithm that interleaves the drilling phase with a sliding one that diversifies [25] the flushed-out solutions.

4.1 Algorithm Description

We will go over the details of `Slide&Drill` (Algorithm 2). There is an illustration of the intuition behind the algorithm's dynamic in Figure 2.

`P-Minimal` (Algorithm 1) attempts to get to optimal solutions quickly by always moving to a dominator of the current point, and so it tries to go "down" towards the origin, so to speak. It assumes good approximations of the sought-after front should, above all, contain optimal solutions as soon as possible and that by diving in this fashion, it will flush them out quicker. But that may not be the case for every problem and application domain. And even if it is true that Pareto-optimal solutions can be found sooner, it may be more important to have a broad, diverse approximation with solutions that are feasible but not necessarily optimal.

In comparison, `Slide&Drill` moves less eagerly and more comprehensively in the direction of the front. It interleaves two mechanisms, *drill* and *slide*, that communicate through a *waiting list* of points and move the incumbent set down until it matches the exact front. The union of the incumbent set and the waiting list will contain an upper bound set of the exact result whenever a new drill is started.

[2] Although we define bound sets as part of the decision space, we will use their image in the objective space as a proxy throughout the description of the algorithms.

Algorithm 2 `Slide&Drill`, Slide and Drill MOCO solver.

Input : $\langle \phi, V, F \rangle$ // MOCO instance
Output : $\text{front}_\phi\, F$ // one img-front
1 $(\widetilde{\phi}, O) \leftarrow$ `EncodeCNF`(F, V) // build unary counters
2 $\phi \leftarrow \phi \cup \widetilde{\phi}$
3 $W \leftarrow \{\, u_{max} \,\}$ // maximal point
4 $I \leftarrow \emptyset$
5 **while** $W \neq \emptyset$ **do** // drill
6     $\boldsymbol{\omega} \leftarrow \langle \textit{select and remove} \rangle\, (W)$
7     $\alpha \leftarrow \{\{\, \bar{o}_{i,\boldsymbol{\omega}_i+1} \,\} : i \in 1 \ldots m \,\}$ // set up drill at $\boldsymbol{\omega}$
8     $\alpha' \leftarrow \emptyset$
9     $\nu \leftarrow \phi$-`SAT`(α)
10   **while** $\nu \neq \emptyset$ **do** // slide
11         $\boldsymbol{x} \leftarrow \nu(V),\ \boldsymbol{y} \leftarrow F(\boldsymbol{x})$
12         $\phi \leftarrow \phi \cup \{\, \{\, \bar{o}_{i,\boldsymbol{y}_i} : i \in 1 \ldots m \,\} \,\}$ // block $\boldsymbol{y}$ dominated region
13         $I \leftarrow I \backslash \{\, \boldsymbol{y}' \in I : \boldsymbol{y} \preceq \boldsymbol{y}' \,\} \cup \{\, \boldsymbol{y} \,\}$ // update incumbent set
14         $W \leftarrow W \cup \{\, \boldsymbol{y} \,\}$ // update waiting list
15         $\alpha' \leftarrow \alpha' \cup \{\, \{\, o_{i,\boldsymbol{y}_i} : i \in 1 \ldots m \,\} \,\}$ // temp. focus non-dominating
16         $\nu \leftarrow \phi$-`SAT`$(\alpha \cup \alpha')$
17   **end**
18 **end**
19 **return** I

After the initialization and the encoding of the unary counters, the external *drill* loop (line 5) hones the incumbent set I, as long as it is possible to do so. When we drill at site $\boldsymbol{\omega}$ (line 6), we look for points that dominate $\boldsymbol{\omega}$ (i.e., solutions "below" $\boldsymbol{\omega}$). This is accomplished by line 7 and the semantics of the unary counters. The first drill site is the maximal point u_{max}, (line 3), i.e., the point whose coordinates in the objective space are the maximal values of the objective functions. The $\langle$*select and remove*$\rangle$ procedure fetches an element of W while removing it and can be implemented using different strategies. When the waiting list is depleted, the drill loop stops. At that point, the incumbent set I is the complete solution, and the algorithm returns. The waiting list is expanded by the inner *slide* loop (lines 10-16). The *waiting list* W takes in freshly found solutions that will eventually be used to start another drill. Besides, the solutions are also placed into the incumbent set I that represents the best approximation of the front so far. The incumbent set will be reported if the solver cannot finish under the resource limits.

This slide loop is the main distinction between `Slide&Drill` and `P-Minimal`. Instead of drilling until striking an optimal solution, as the `P-Minimal` algorithm does, we steer the oracle so as to slide across the objective space, collecting solutions that do not dominate each other. This is accomplished by building the auxiliary formula α' while accruing the waiting list. The formula α' contains one clause per point found since the start of the last slide loop (line 15) and blocks the region under the known solutions. As soon as the solver fails to find an extra point the slide loop is complete and the implicit upper bound set contained in the union of the waiting list and the incumbent set was made whole again. Figure 2 provides a small example of the execution of the `Slide&Drill` algorithm.

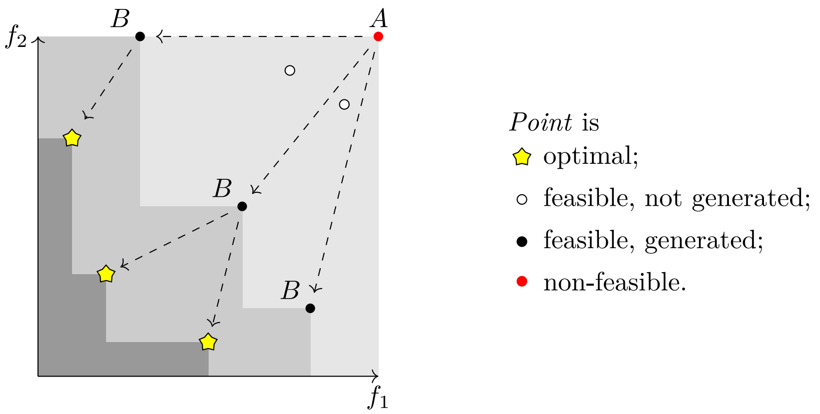


Figure 2 Illustration of a run of the `Slide&Drill` (Algorithm 2). Three upper bound sets are produced, marked by A, B and the star. The first drill site is the maximal point. We drill and find one of the elements marked with B. The remaining ones are generated by the slide loop. Note this is our first satisfiable upper bound set. Assume the next drill site, chosen by ⟨*select and remove*⟩, is B's midpoint. As we drill again, either of the two optimal solution is found, and the slide generates the other. The uppermost B point is chosen next, and the missing optimal solution is found during the subsequent drill. There are 4 remaining drill sites to consider: the three optimal solutions and the lowermost B point. Neither will produce new solutions, and the algorithm terminates after four more "blank" drills. All B elements are dominated, and they are pushed out of I by the addition of the optimal solutions. The shading levels vary as the number of upper bound sets that dominate the region. The lighter tone is painted by A only, while the darker is painted by all three.

The waiting list can be backed by different containers. We consider both a *stack* (i.e., FIFO container) and a *queue* (i.e., LIFO container). Different containers result in different implementations of ⟨*select and remove*⟩ (line 6), and hence a different concrete `Slide&Drill`.

- If a *stack* is used the algorithm resembles `P-Minimal`, but it is not quite the same. It is safer because it will perform a slide step, and hence diversify the incumbent set before drilling further. If the computation results in timeout, the pool of solutions will differ from what `P-Minimal` would have found. There is a trade-off between the number of optimal points (probably larger with `P-Minimal`) and the diversity of the points obtained;
- If a *queue* is used, the algorithm is substantially different from `P-Minimal`. We expect less optimal solutions but more robust approximations. This is a more extreme approach than the one resulting from using a stack. It will further tilt the scale in the favor of diverse but suboptimal points.

4.2 Algorithm Properties

Let us prove `Slide&Drill` (Algorithm 2) is *sound* and *complete* (Lemma 17)

▶ **Lemma 14.** *Any optimal point that dominates the drill site $\boldsymbol{\omega}$ will dominate at least one of the points generated by the associated slide loop (line 10).*

Proof. Assume that Lemma 14 is not true. Then, there must exist an optimal point $\boldsymbol{y}$ that dominates $\boldsymbol{\omega}$ but fails to dominate any of the generated points. In that case, the temporary constraints added at line 15 do not render $\boldsymbol{y}$ unsatisfiable, and because $\boldsymbol{y}$ is optimal, neither do the permanent constraints added at line 12. And therefore, $\boldsymbol{y}$ must have been generated. And that contradicts the assumption because $\boldsymbol{y}$ dominates itself. ◀

▶ **Lemma 15.** *At the start of the outer loop (line 5), the union of the optimal points in the incumbent set I with the waiting list W contains an upper bound set of the front $\overline{Y} = \mathrm{front}_Z\, F$.*

Proof. This is true for the first run because the waiting list contains the maximal point.

Assume Lemma 15 true at the start of iteration i, and let U be an upper bound set contained in $I \cap \overline{Y} \cup W$. We want to prove that an upper bound set U' is contained in $I' \cap \overline{Y} \cup W'$, where I' and W' are the incumbent set and waiting list at the start of iteration $i + 1$.

Let $W' = W \setminus \{\boldsymbol{\omega}\} \cup \Delta W$, where ΔW is the set accrued by the successive executions of line 14. We will prove that $C = U \setminus \{\boldsymbol{\omega}\} \cup \Delta W$ is an upper-cover of $\overline{Y}$. All solutions $\boldsymbol{y} \in \overline{Y}$ that do not dominate $\boldsymbol{\omega}$ are covered by elements in U. Solutions $\boldsymbol{y}$ that do dominate $\boldsymbol{\omega}$ are covered by elements in ΔW, by Lemma 14.

If the upper cover C is thin, then $U' = C$. Otherwise, for any pair of comparable elements $\boldsymbol{y} \preceq \boldsymbol{y}' \in C$, drop $\boldsymbol{y}$. The obtained set is a cover because any point dominating $\boldsymbol{y}$ dominates $\boldsymbol{y}'$ too. The remaining elements of C are incomparable and are collected into U' so that $U' \subseteq C$.

To see that $U' \subseteq I' \cap \overline{Y} \cup W'$,

$$U' \subseteq C = U \setminus \{\boldsymbol{\omega}\} \cup \Delta W \implies \tag{5}$$

$$U' \subseteq (I \cap \overline{Y} \cup W) \setminus \{\boldsymbol{\omega}\} \cup \Delta W \implies \tag{6}$$

$$U' \subseteq (I' \cap \overline{Y} \cup W) \setminus \{\boldsymbol{\omega}\} \cup \Delta W \implies \tag{7}$$

$$U' \subseteq (I' \cap \overline{Y}) \setminus \{\boldsymbol{\omega}\} \cup W \setminus \{\boldsymbol{\omega}\} \cup \Delta W \implies \tag{8}$$

$$U' \subseteq (I' \cap \overline{Y}) \setminus \{\boldsymbol{\omega}\} \cup W' = (I' \cap \overline{Y} \cup W') \setminus \{\boldsymbol{\omega}\} \subseteq I' \cap \overline{Y} \cup W', \tag{9}$$

where Equation (7) follows because only dominated solutions can be removed from I, and Equation (9) follows because $\boldsymbol{\omega}$ does not belong to W'. ◀

▶ **Lemma 16.** *At the start of the outer loop (line 5), any point in I that does not belong to W is optimal.*

Proof. All points are added to both I and W. If some point $\boldsymbol{\omega}$ does not belong to W, then it must have been removed by line 6. After that, the query will return an empty model iff $\boldsymbol{\omega}$ is optimal because the restrictions in ϕ block only dominated regions, and the assumptions focus the search over the region dominating $\boldsymbol{\omega}$. If $\boldsymbol{\omega}$ is not optimal, the query at line 16 will generate a point that dominates it, and that point will push off $\boldsymbol{\omega}$ from I at line 13. ◀

▶ **Proposition 17.** *Algorithm 2 is sound and complete.*

Proof. Let us prove soundness first. If the algorithm returns, W is empty. By Lemma 15, I contains an upper bound set. By Lemma 16, all its elements are optimal. Every element of the front dominates at least one element of I. Assume $\boldsymbol{y}$ is optimal and is not part of I. It must be dominated by some element of I, but an optimal point is dominated only by itself. Hence, $\boldsymbol{y}$ cannot be absent from I.

Let us move on to show the algorithm is complete. The clauses added by line 12 block at least one feasible model each, as they block the dominated region, including its defining vertex. Because no blocking clause is ever dropped, the number of satisfiable queries is bounded by the number of satisfiable models, which is finite.

After entering the slide loop at line 10, it will fail to return iff there is an infinite number of satisfiable queries, which cannot happen, given the former argument.

Therefore, every operation occurring in the drill loop (line 5) ends successfully in a finite amount of time. Therefore, the loop exits iff W becomes empty.

Note that the waiting list receives new elements only at line 14. Based on the argument above, the number of inserted elements must always be finite. Each iteration of the drill loop takes one element out. Assume this loop never ends. Eventually, the number of removals would catch up to the number of insertions, and the waiting list would be empty. But then, the loop would end, which contradicts the hypothesis. ◀

4.3 Tandem `Slide&Drill`

Two different solvers working together will most likely produce better results than any of them would by themselves.

Suppose we have two different approximated fronts A and B of a MOCO instance, produced respectively by solvers a and b. Consider also the combined solution $A \hat{} B$, built from $A \cup B$ by weeding out any dominated point from the union. Most likely, $A \hat{} B$ is a better approximation of the front than any of the solutions A and B by themselves. And it cannot be worse. Even more, had they shared the incrementally built approximations on the fly, the workers would have guided each other and avoided regions of the objective space that were already branded as dominated by some feasible solution produced by the other contributor.

Because `Slide&Drill` is a SAT-UNSAT solver, it makes sense to consider for its companion an UNSAT-SAT solver. We chose a previously proposed UNSAT-SAT algorithm named `Core-Guided` [5]. The workers (i.e., `Slide&Drill` and `Core-Guided`) will share a single, *incrementally* built formula. Note that the unary counters representing the objective functions are shared, as is the SAT oracle.

In order to synchronize their work, there is a *conflict budget*. The solvers will work in turn: as soon as the assigned budget is fully depleted on SAT calls the current worker stops, and the other contributor kicks in with a restored budget.

For the `Slide&Drill` algorithm, we simply reinsert the last drill site into the waiting list and proceed. For the `Core-Guided` algorithm, we keep track of the current upper-fence and bootstrap the next search session by setting the upper fence to the backed-up value.

5 Results and Analysis

5.1 Benchmark Sets and Experimental Setup

In order to evaluate our MOCO algorithms against other state-of-the-art MOCO solvers, we consider two publicly available benchmark sets of MOCO instances that have already been used in previous research works.

The Development Assurance Level (DAL) [4] benchmark set [3] is composed of 95 instances encoding different levels of rigour in the development of a software or hardware component of an aircraft. The development assurance level defines the assurance activities aimed at eliminating design and coding errors that could affect the safety of an aircraft. The goal is to allocate the smallest DAL to functions to decrease the development costs.

The Package Upgradeability (PU) benchmark set is composed of 687 instances from the Mancoosi International Solver Competition [18]. Each instance encodes the upgradeability of packages in an open-source system. The `packup` tool [13] was used to generate variants containing between two and five objectives to optimize. This results in 3570 instances.

[3] `https://www.lifl.fr/LION9/challenge.html`.

All of the experiments were conducted on a computer with Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz running Linux Debian 10.2. Each problem instance was executed for each MOCO solver with a memory limit of 32 GB and a CPU timeout of 10 minutes (600 seconds) imposed using the `runsolver` [23] tool.

5.2 Evaluated Algorithms

We evaluate our algorithms against several SAT-based MOCO solvers. The `ParetoMCS` algorithm [4] is based on the enumeration of MCSs of the MOCO instance [28] and the `P-Minimal` algorithm implements the SAT-UNSAT approach presented in Algorithm 1 [26]. Additionally, the `Core-Guided` algorithm implements a complementary UNSAT-SAT approach [5].

The `Slide&Drill` algorithm implements our new approach proposed in Algorithm 2. Furthermore, both the `Slide&Drill` and `P-Minimal` approaches are combined with the `Core-Guided` algorithm, (as described in section 4.3).

All algorithms are implemented using the publicly available codebase [5] from the authors of the `Core-Guided` algorithm [5]. Hence, all algorithms use the selection delimiter encoding [16] to represent the objective functions. Furthermore, the underlying SAT solver is also the same and used incrementally [7, 21, 1]. As a result, the observed differences in performance are mainly from the algorithmic techniques employed and a more fair comparison is achieved.

5.3 Evaluation Metrics

Finding the Pareto front of MOCO instances is computationally harder than solving single-objective optimization problems. In most cases, given an acceptable time limit, solvers can only provide an approximation of the Pareto front.

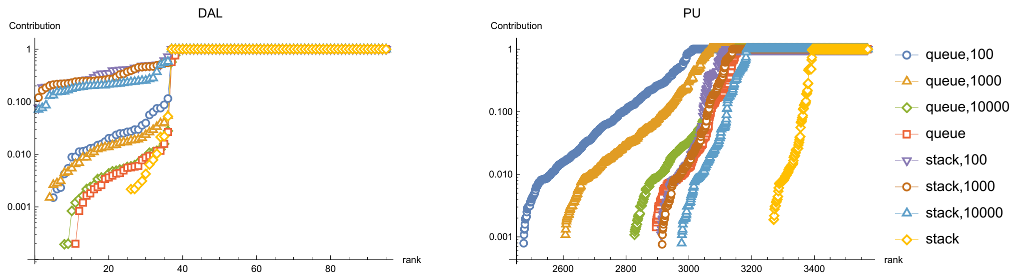
Let $\mathcal{A}$ denote a set of algorithms and variants to be evaluated and let $\mathcal{I}$ denote the set of instances. Let $Y_{i,j}$ denote the approximation of the Pareto front provided by algorithm A_i ($A_i \in \mathcal{A}$) for instance I_j ($I_j \in \mathcal{I}$). Let R_j denote the reference set for instance I_j defined as $R_j = \cup_{A_i \in \mathcal{A}} Y_{i,j}$, where only the incomparable elements are kept, i.e., all dominated solutions are filtered out of R_j. Hence, the reference set R_j contains only the best solutions found by any of the evaluated algorithms in $\mathcal{A}$.

To evaluate the quality of the approximations provided by each tool, we use three different metrics. The first metric is the *Contribution indicator* that measures the contribution of a given algorithm to the reference set. Hence, the contribution indicator of algorithm $A_i \in \mathcal{A}$ in a MOCO instance I_j is defined as $\frac{|Y_{i,j} \cap R_j|}{|R_j|}$. Clearly, *larger* values are preferable since the metric is maximized when the algorithm is able to identify all solutions in the reference set.

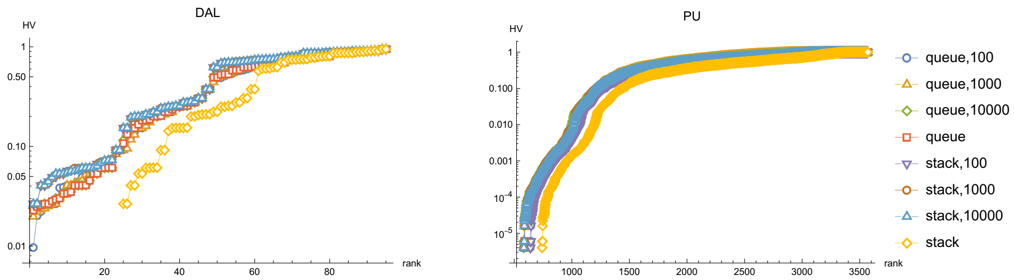
The second metric is the Hypervolume (HV) indicator [31]. This indicator measures the volume of the objective space between the set of nondominated solutions $Y_{i,j}$ and a given reference point u_r. The reference point depends on the benchmark. For a given instance I_j, the reference point is set to the largest possible objective values in the reference set R_j [6]. As in the previous indicator, *larger* values of HV are preferable since the volume of the dominated objective space is maximized at the Pareto front.

Finally, the third metric is the Inverted Generational Distance (IGD) indicator [30, 6]. IGD measures the average Euclidean distance, in the objective space, between the reference set R_j and the solution set $Y_{i,j}$ returned by the algorithm. In this case, *smaller* values of IGD are preferable, meaning that the solution set $Y_{i,j}$ is closer to the reference set R_j.

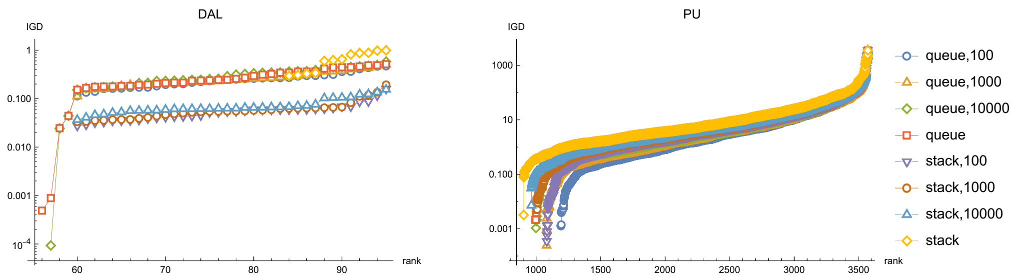
[4] `https://gitlab.ow2.org/sat4j/moco`
[5] `https://gitlab.inesc-id.pt/u001810/moco`
[6] If the reference set R_j is the Pareto front, then the reference point u_r is the Nadir point [20].



(a) *Contribution* metric on DAL instances. **(b)** *Contribution* metric on PU instances.



(c) *HV* metric on DAL instances. **(d)** *HV* metric on PU instances.



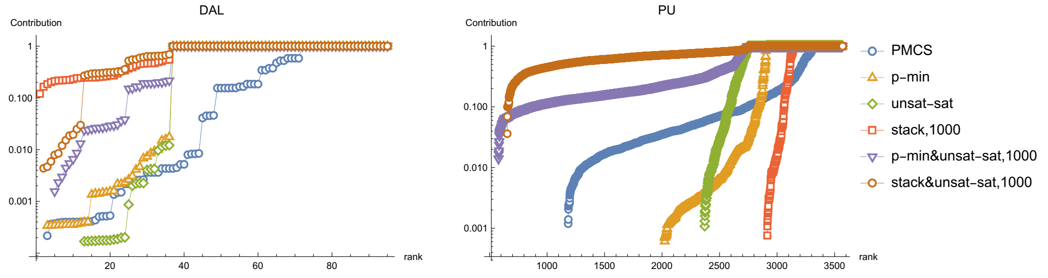
(e) *IGD* metric on DAL instances. **(f)** *IGD* metric on PU instances.

Figure 3 Comparison of the *Contribution*, *IGD* and *HV* results for each set of instances. `Slide&Drill` variants only. Each series is sorted independently, smaller values first. Vertical scale is logarithmical. Each series is labelled by the type of waiting list and the value of the conflict budget.

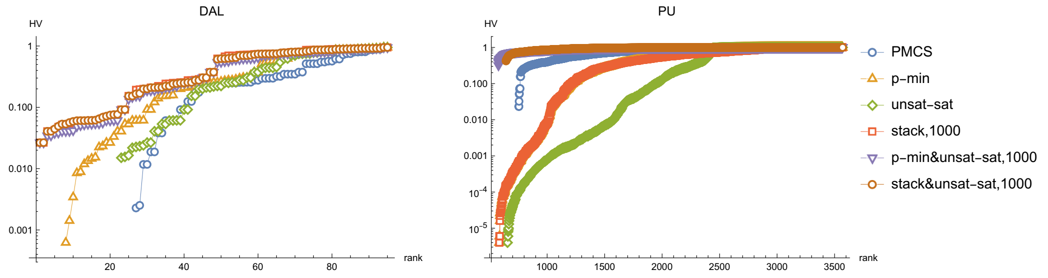
5.4 Slide and Drill Variants

The `Slide&Drill` algorithm (Algorithm 2) can be configured in different ways. In this section we focus on the management of the waiting list and the SAT solver call. As mentioned in section 4.1, the waiting list can be managed as a stack or as a queue and this results in exploring the search space in different ways. Additionally, one can set the SAT solver call with a limited budget of conflicts in order for `Slide&Drill` not to get "stuck". Setting up a conflict budget will not violate neither soundness nor completeness, since the site of the unfinished drill goes back into the waiting list and all SAT calls are done in an incremental fashion (i.e., the same SAT solver instance is always used and no learned clause is ever removed).

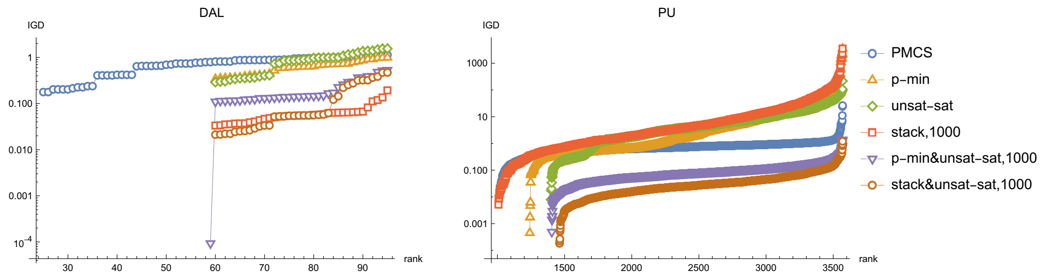
Figure 3 shows the results of several variants of the `Slide&Drill` algorithm for the three metrics defined in section 5.3 for both the DAL (left) and PU (right) benchmark sets. The `stack` and `queue` variants denote that the waiting list is managed as a stack and queue, respectively. Moreover, whenever the `stack` and `queue` variants are followed by a number C,



(a) *Contribution* metric on DAL instances. **(b)** *Contribution* metric on PU instances.



(c) *HV* metric on DAL instances. **(d)** *HV* metric on PU instances.



(e) *IGD* metric on DAL instances. **(f)** *IGD* metric on PU instances.

Figure 4 Comparison of the *Contribution*, *IGD* and *HV* results for each set of instances. Each series is sorted independently, smaller values first. Vertical scale is logarithmical.

then C denotes the conflict limit in the SAT call. Whenever the conflict limit is reached, the SAT call ends and the `Slide&Drill` algorithm retrieves a new starting point from the waiting list. Otherwise, no limit is imposed on the SAT call.

The experimental results in these benchmark sets show that the algorithm performs better when a conflict limit is imposed. This occurs for all metrics in both benchmark sets. The budgeted SAT call allows the algorithm to choose a new element of the waiting list, allowing it to find a wider variety of solutions that better approximates the Pareto front.

We obtained mixed results regarding the waiting list's management. While the `stack` variants perform better for the DAL benchmark set, the `queue` variants perform better on PU instances. This assay is based on the contribution metric, as the overall values for HV and IGD are similar.

5.5 Comparison with Other MOCO Solvers

We compare the `stack, 1000` variant of the `Slide&Drill` algorithm (`stack` strategy for management of the waiting list and $C = 1000$ for the conflict limit on the SAT solver) against other state-of-the-art MOCO solvers. We chose this variant of the `Slide&Drill` algorithm since it seems to be the most balanced one, considering the results from the previous section.

The results on DAL (left) and PU (right) benchmarks considering the three metrics are available in Figure 4. For the DAL benchmarks, the new `Slide&Drill` algorithm is able to outperform the `ParetoMCS` (`PMCS`), `Core-Guided` (`unsat-sat`) and `P-Minimal` (`p-min`) algorithms on all metrics. The approximation of the Pareto front provided by `Slide&Drill` on these instances is clearly better than the ones produced by all other algorithms. Due to the newly proposed strategy, `Slide&Drill` is able to find a more diverse set of solutions and, thus, a more accurate approximation of the Pareto front. Furthermore, even when `Slide&Drill` and `Core-Guided` work in tandem (`stack&unsat-sat`), there are only very slight improvements to the contribution metric.

On the PU benchmarks, the `Slide&Drill` (`stack, 1000`) algorithm is able to find solutions close to the `P-Minimal` (`p-min`) algorithm considering both the HV and IGD metrics. Moreover, it is able to outperform the `Core-Guided` (`unsat-sat`) algorithm. However, the `ParetoMCS` (`PMCS`) is the best standalone algorithm in terms of HV and IGD. Nevertheless, when `Slide&Drill` is paired with `Core-Guided` in tandem (`stack&unsat-sat`), then this approach is clearly better on all metrics on the PU benchmark set. This is due to the high complementarity of these algorithms when applied on the PU instances. Observe that the `P-Minimal`, when paired with `Core-Guided` in tandem (`p-min&unsat-sat`), also improves its performance. However, the tandem `Slide&Drill` and `Core-Guided` still performs better on all metrics due to the higher diversification of solutions provided by our new `Slide&Drill` algorithm.

6 Conclusions and Future Work

This paper introduces the Slide and Drill approach for solving MOCO problems. The proposed `Slide&Drill` algorithm is a SAT-based algorithm with a strategy to diversify the set of solutions found such that a better approximation of the Pareto front can be found. Previously proposed algorithms either disregard the objective function representation (e.g., through the enumeration of MCS) or have too much focus on proving that a given solution is Pareto-optimal, resulting in being able to identify only a small set of the Pareto front.

Experimental results on two representative sets of benchmarks show that the new `Slide&Drill` algorithm outperforms previous SAT-based MOCO solvers on three different metrics. Moreover, the performance of the `Slide&Drill` algorithm can be additionally boosted when paired with a complementary `Core-Guided` approach. Hence, the newly proposed algorithms further enhance the usage of SAT-based approaches for MOCO.

The Slide and Drill approach introduced in this paper can be configured using different techniques to diversify the exploration of the search space. In this paper we exploit several strategies to choose elements of a waiting list that correspond to areas of the search space still to explore. In future work, we propose to manage the waiting list as a priority queue using a performance metric such as the Hypervolume as the selection criterion. Although this criterion has already been used in other algorithmic contexts [27], using it in a tandem algorithm with both `Slide&Drill` and `Core-Guided` approaches poses new additional challenges.

References

1 Gilles Audemard, Jean-Marie Lagniez, and Laurent Simon. Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction. In Matti Järvisalo and Allen Van Gelder, editors, *16th International Conference on Theory and Applications of Satisfiability Testing, SAT 2013, Helsinki, Finland, July 8-12, 2013*, volume 7962 of *Lecture Notes in Computer Science*, pages 309–317. Springer, 2013. doi:10.1007/978-3-642-39071-5_23.

2 Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. Maximum satisfiabiliy. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 929–991. IOS Press, 2021. doi:10.3233/FAIA201008.

3 Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In Francesca Rossi, editor, *International Conference on Principles and Practice of Constraint Programming (CP)*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2003. doi:10.1007/978-3-540-45193-8_8.

4 Pierre Bieber, Remi Delmas, and Christel Seguin. Dalculus - theory and tool for development assurance level allocation. In Francesco Flammini, Sandro Bologna, and Valeria Vittorini, editors, *Computer Safety, Reliability, and Security - 30th International Conference, SAFECOMP 2011, Naples, Italy, September 19-22, 2011. Proceedings*, volume 6894 of *Lecture Notes in Computer Science*, pages 43–56. Springer, 2011. doi:10.1007/978-3-642-24270-0_4.

5 João Cortes, Inês Lynce, and Vasco M. Manquinho. New core-guided and hitting set algorithms for multi-objective combinatorial optimization. In Sriram Sankaranarayanan and Natasha Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings, Part II*, volume 13994 of *Lecture Notes in Computer Science*, pages 55–73. Springer, 2023. doi:10.1007/978-3-031-30820-8_7.

6 Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. *IEEE Trans. Evol. Comput.*, 18(4):577–601, 2014. doi:10.1109/TEVC.2013.2281535.

7 Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. In Ofer Strichman and Armin Biere, editors, *First International Workshop on Bounded Model Checking, BMC@CAV 2003, Boulder, Colorado, USA, July 13, 2003*, volume 89 of *Electronic Notes in Theoretical Computer Science*, pages 543–560. Elsevier, 2003. doi:10.1016/S1571-0661(05)82542-3.

8 Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, 2006. doi:10.3233/sat190014.

9 Matthias Ehrgott, Xavier Gandibleux, and Anthony Przybylski. Exact methods for multi-objective combinatorial optimisation. In Salvatore Greco, Matthias Ehrgott, and José Rui Figueira, editors, *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 817–850. Springer New York, New York, NY, 2016. doi:10.1007/978-1-4939-3094-4_19.

10 Marco Gavanelli. An algorithm for multi-criteria optimization in csps. In *European Conference on Artificial Intelligence*, pages 136–140. IOS Press, 2002.

11 Christoph Jabs, Jeremias Berg, Hannes Ihalainen, and Matti Järvisalo. Preprocessing in sat-based multi-objective combinatorial optimization. In Roland H. C. Yap, editor, *29th International Conference on Principles and Practice of Constraint Programming, CP 2023, August 27-31, 2023, Toronto, Canada*, volume 280 of *LIPIcs*, pages 18:1–18:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.CP.2023.18.

12 Christoph Jabs, Jeremias Berg, Andreas Niskanen, and Matti Järvisalo. Maxsat-based bi-objective boolean optimization. In *International Conference on Theory and Applications of Satisfiability Testing*, volume 236 of *LIPIcs*, pages 12:1–12:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SAT.2022.12.

13 Mikolás Janota, Inês Lynce, Vasco M. Manquinho, and João Marques-Silva. Packup: Tools for package upgradability solving. *J. Satisf. Boolean Model. Comput.*, 8(1/2):89–94, 2012. doi:10.3233/sat190090.

14 Mikolás Janota, António Morgado, José Fragoso Santos, and Vasco M. Manquinho. The seesaw algorithm: Function optimization using implicit hitting sets. In *International Conference on Principles and Practice of Constraint Programming*, volume 210 of *LIPIcs*, pages 31:1–31:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CP.2021.31.

15 Saurabh Joshi, Ruben Martins, and Vasco M. Manquinho. Generalized totalizer encoding for pseudo-boolean constraints. In *International Conference Principles and Practice of Constraint Programming*, volume 9255 of *LNCS*, pages 200–209. Springer, 2015. doi:10.1007/978-3-319-23219-5_15.

16 Michal Karpinski and Marek Piotrów. Encoding cardinality constraints using multiway merge selection networks. *Constraints*, 24(3-4):234–251, 2019. doi:10.1007/s10601-019-09302-0.

17 Rui Li, Qinghua Zheng, Xiuqi Li, and Zheng Yan. Multi-objective optimization for rebalancing virtual machine placement. *Future Gener. Comput. Syst.*, 105:824–842, 2020. doi:10.1016/j.future.2017.08.027.

18 Mancoosi international solver competition 2011. https://www.mancoosi.org/misc-2011/index.html.

19 Rafael Marques, Luís M. S. Russo, and Nuno Roma. Flying tourist problem: Flight time and cost minimization in complex routes. *Expert Syst. Appl.*, 130:172–187, 2019. doi:10.1016/j.eswa.2019.04.024.

20 Kaisa Miettinen. *Nonlinear Multiobjective Optimization*, volume 12. Springer Science & Business Media, 2012.

21 Alexander Nadel and Vadim Ryvchin. Efficient SAT solving under assumptions. In Alessandro Cimatti and Roberto Sebastiani, editors, *15th International Conference on Theory and Applications of Satisfiability Testing - SAT 2012, Trento, Italy, June 17-20, 2012*, volume 7317 of *Lecture Notes in Computer Science*, pages 242–255. Springer, 2012. doi:10.1007/978-3-642-31612-8_19.

22 Derek Rayside, H.-Christian Estler, and Daniel Jackson. The guided improvement algorithm for exact, general-purpose, many-objective combinatorial optimization. Technical Report Technical Report MIT-CSAIL-TR-2009-033, MIT Massachusetts Institute of Technology, 2009.

23 Olivier Roussel. Controlling a Solver Execution with the runsolver Tool: System description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(4):139–144, November 2011. doi:10.3233/SAT190083.

24 Olivier Roussel and Vasco M. Manquinho. Pseudo-boolean and cardinality constraints. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 695–733. IOS Press, 2009. doi:10.3233/978-1-58603-929-5-695.

25 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.

26 Takehide Soh, Mutsunori Banbara, Naoyuki Tamura, and Daniel Le Berre. Solving multiobjective discrete optimization problems with propositional minimal model generation. In *International Conference Principles and Practice of Constraint Programming*, volume 10416 of *LNCS*, pages 596–614. Springer, 2017. doi:10.1007/978-3-319-66158-2_38.

27 Satya Tamby and Daniel Vanderpooten. Enumeration of the nondominated set of multiobjective discrete optimization problems. *INFORMS J. Comput.*, 33(1):72–85, 2021. doi:10.1287/IJOC.2020.0953.

28 Miguel Terra-Neves, Inês Lynce, and Vasco M. Manquinho. Introducing pareto minimal correction subsets. In *International Conference on Theory and Applications of Satisfiability Testing*, volume 10491 of *LNCS*, pages 195–211. Springer, 2017. doi:10.1007/978-3-319-66263-3_13.

29 Yuan Yuan and Wolfgang Banzhaf. ARJA: automated repair of java programs via multi-objective genetic programming. *IEEE Trans. Software Eng.*, 46(10):1040–1067, 2020. doi:10.1109/TSE.2018.2874648.

30 Qingfu Zhang and Hui Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Trans. Evol. Comput.*, 11(6):712–731, 2007. doi:10.1109/TEVC.2007.892759.

31 E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, University of Zurich, Zürich, Switzerland, 1999.