Integrating Pseudo-Boolean Constraint Reasoning in Multi-Objective Evolutionary Algorithms

Miguel Terra-Neves[1,2] , **Inês Lynce**[1] and **Vasco Manquinho**[1]
[1] INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, Portugal
[2] OutSystems, Portugal
miguel.neves@outsystems.com, {ines,vmm}@sat.inesc-id.pt

Abstract

Constraint-based reasoning methods thrive in solving problem instances with a tight solution space. On the other hand, evolutionary algorithms are usually effective when it is not hard to satisfy the problem constraints. This dichotomy has been observed in many optimization problems. In the particular case of Multi-Objective Combinatorial Optimization (MOCO), new recently proposed constraint-based algorithms have been shown to outperform more established evolutionary approaches when a given problem instance is hard to satisfy. In this paper, we propose the integration of constraint-based procedures in evolutionary algorithms for solving MOCO. First, a new core-based smart mutation operator is applied to individuals that do not satisfy all problem constraints. Additionally, a new smart improvement operator based on Minimal Correction Subsets is used to improve the quality of the population. Experimental results clearly show that the integration of these operators greatly improves multi-objective evolutionary algorithms MOEA/D and NSGAII. Moreover, even on problem instances with a tight solution space, the newly proposed algorithms outperform the state-of-the-art constraint-based approaches for MOCO.

1 Introduction

In several real-world problems (e.g. software product lines [Henard *et al.*, 2015] or virtual machine consolidation in data centers [Zheng *et al.*, 2016]), there are several conflicting objectives to be optimized such that a set of Boolean linear constraints must be satisfied. These Multi-Objective Combinatorial Optimization (MOCO) problems have multiple optimal solutions, known as Pareto optimal solutions, each of them favoring certain objectives at the expense of others. The enumeration of all Pareto optimal solutions is known to be very hard. In practice, most algorithms can only approximate the Pareto front, i.e. the set of all Pareto optimal solutions.

Multi-Objective Evolutionary Algorithms (MOEAs) have been at the forefront for solving MOCO problems [Deb *et al.*, 2000; Zhang and Li, 2007; Zheng *et al.*, 2016]. However, in problem instances where it is hard to satisfy all problem constraints, the performance of MOEAs starts to decline. On the other hand, recently proposed constraint-based methods [Soh *et al.*, 2017; Terra-Neves *et al.*, 2018] thrive in solving tightly constrained instances. This duality, between evolutionary algorithms and constraint-based methods, motivated the selective integration of constraint solving in MOEAs [Henard *et al.*, 2015; Xiang *et al.*, 2018].

In previous works, MOEA tools use a constraint solver for correcting individuals or adding diversity to the population. An individual corresponds to a complete assignment to all problem variables. Given a complete assignment that does not satisfy all problem constraints, a partial assignment can be generated by unassigning the variables in unsatisfied constraints, while the remaining variables are fixed. Next, a constraint solver is used to extend the partial assignment into a complete satisfying assignment. As a result, a new feasible individual is added to the population. However, in general, there is no guarantee that the partial assignment can be extended to satisfy all problem constraints. In this paper, we propose a core-based approach to solve this issue.

The main contributions of the paper are: (1) a new core-based smart mutation operator that corrects individuals in any MOEA for MOCO; (2) a novel smart improvement operator based on Minimal Correction Subsets (MCSs); (3) the usage of incremental solving to avoid the constraint solver from generating the same individual; and (4) an extensive experimental evaluation showing that MOEA solvers with the proposed operators can outperform both the classic MOEA tools and recently proposed constraint-based solvers for MOCO.

The paper is organized as follows. Section 2 defines MOCO and MCS, and briefly reviews MOEAs and the virtual machine consolidation problem. Next, Section 3 describes the new core-based smart mutation operator, while Section 4 proposes the usage of smart improvements using MCSs. Section 5 presents an extensive experimental evaluation of the proposed techniques and the paper concludes in Section 6.

2 Preliminaries

This section starts by describing MOCO and MOEAs. Next, MCS based approaches for MOCO solving are introduced. Finally, we describe Virtual Machine Consolidation (VMC), a multi-objective benchmark problem used to evaluate the smart evolutionary operators proposed in this paper.

2.1 Multi-Objective Combinatorial Optimization

Let $X = \{x_1, \ldots, x_n\}$ be a set of n Boolean variables. A literal is either a variable x_i or its complement $\overline{x_i}$. Given a set of m literals $l_1, \ldots, l_m$ and respective coefficients $\omega_1, \ldots, \omega_m \in \mathbb{N}$, a Pseudo-Boolean (PB) expression is a weighted sum of literals $\sum_{i=1}^{m} \omega_i \cdot l_i$. Given an integer $k \in \mathbb{N}$, a linear PB constraint has the form:

$$\sum_{i=1}^{m} \omega_i \cdot l_i \bowtie k, \quad \bowtie \in \{\leq, \geq, =\}. \tag{1}$$

Note that a clause $(l_1 \vee \cdots \vee l_m)$ is equivalent to the PB constraint $(\sum_{i=1}^{m} l_i \geq 1)$. In some cases, we use clause notation for ease of explanation.

Given a set $\phi = \{c_1, \ldots, c_m\}$ of m PB constraints, the Pseudo-Boolean Satisfiability (PBS) problem consists of deciding if there exists a complete assignment $\alpha : X \to \{0, 1\}$ that satisfies all constraints in ϕ. If that is the case, we say that ϕ is satisfiable and α is a model of ϕ, denoted $\alpha(\phi) = 1$. If α does not satisfy ϕ, we say that α is unfeasible, denoted $\alpha(\phi) = 0$. If ϕ is unsatisfiable, then $\alpha(\phi) = 0$ for any assignment α. An instance of the Pseudo-Boolean Optimization (PBO) problem [Boros and Hammer, 2002] is a pair $\Phi = (\phi, f(X))$, where $\phi = \{c_1, \ldots, c_m\}$ is a set of m PB constraints and $f(X)$ is a PB expression representing a cost function. The goal is to find a model α of ϕ that minimizes its cost, denoted as $f(\alpha)$.

Example 2.1 *Let $(\phi, f(X))$ denote a PBO instance where $\phi = \{x_1 + x_2 + x_3 \geq 2, x_1 + x_2 = 1\}$ and $f(X) = x_1 + 2x_2 + 3x_3$. In this case, an optimal solution would be $\alpha = \{(x_1, 1), (x_2, 0), (x_3, 1)\}$ with an overall cost of 4.*

In this work, we consider that a MOCO [Ulungu and Teghem, 1994] instance is a pair $\Phi = (\phi, F(X))$, where ϕ is a set of PB constraints to satisfy and $F(X) = (f_1(X), \ldots, f_k(X))$ is a vector of k PB expressions to minimize, defined over a set X of Boolean variables. Given a complete assignment α such that $\alpha(\phi) = 1$, its cost vector is defined as $F(\alpha) = (f_1(\alpha), \ldots, f_k(\alpha))$. Let α and α' be two distinct complete assignments such that $\alpha(\phi) = \alpha'(\phi) = 1$. We say that $F(\alpha)$ dominates $F(\alpha')$ (α dominates α'), denoted $F(\alpha) \prec F(\alpha')$ ($\alpha \prec \alpha'$), if and only if $f_i(\alpha) \leq f_i(\alpha')$ for all $i = 1..k$ and $f_j(\alpha) < f_j(\alpha')$ for some $j = 1..k$. The Pareto front P_Φ of Φ is the set of its nondominated cost vectors, i.e., $F(\alpha) \in P_\Phi$ if and only if $\alpha(\phi) = 1$ and no α' exists such that $\alpha'(\phi) = 1$ and $F(\alpha') \prec F(\alpha)$. α is said to be a Pareto-optimal solution of Φ if and only if $F(\alpha) \in P_\Phi$. In MOCO, the goal is to find the Pareto front of Φ.

Example 2.2 *Let $(\phi, F(X))$ denote a MOCO instance where $\phi = \{x_1 + x_2 + x_3 \geq 2\}$ and $F(X) = (f_1(X), f_2(X)) = (2x_1 + x_2, 2\overline{x_2} + 2x_3)$. In this case, there are two Pareto optimal solutions: $\alpha_1 = \{(x_1, 0), (x_2, 1), (x_3, 1)\}$ with costs $(1, 2)$ and $\alpha_2 = \{(x_1, 1), (x_2, 1), (x_3, 0)\}$ with costs $(3, 0)$. Note that α_1 provides a better value for f_1, while α_2 is able to improve on f_2. All other satisfiable assignments to ϕ are dominated by either α_1 or α_2.*

For most problems, it is not possible to enumerate all Pareto optimal solutions P_Φ in a reasonable amount of time. Therefore, we focus on finding a set of solutions that best approximates the Pareto front P_Φ within a given time limit.

Algorithm 1 Typical MOEA framework for MOCO

```
Input: $\Phi = (\phi, F(X))$
1: $P \leftarrow$ InitialSolutions($\Phi$)
2: while stopping criteria not triggered do
3:     $Q \leftarrow$ GenerateOffsprings($P$)
4:     $P \leftarrow$ SelectSurvivors($P, Q$)
5: end while
6: return FilterDominated($P$)
```

2.2 Multi-Objective Evolutionary Algorithms

The typical MOEA framework is presented in Algorithm 1. Such algorithms maintain a fixed size population P of complete assignments, referred to as individuals. At each iteration, referred to as a generation, a set Q of $|P|$ offsprings is generated by applying crossover and mutation operators to selected individuals in P (line 3). Then, $|P|$ individuals among P and Q are selected to become the population in next generation (line 4). This process is repeated until some stopping criteria is met (e.g. number of generations, time limit, etc).

The crossover operator produces a new offspring α from two individuals α_1 and α_2 by mixing their variable assignments. For example, given some variable $x \in X$, uniform crossover [Spears and De Jong, 1995; Syswerda, 1989] has a fixed probability (typically 0.5) of setting $\alpha(x) = \alpha_1(x)$, otherwise it sets $\alpha(x) = \alpha_2(x)$. Note that crossover is applied with a given probability p_{cr}, referred to as crossover rate, and therefore, with probability $1 - p_{cr}$, α is a clone of either α_1 or α_2 instead. The mutation operator produces an offspring α' by applying perturbations (e.g. flipping the values of some variables) to an individual α. Similarly to crossover, the frequency of mutation is also controlled by a configurable mutation rate p_{mr}. Note that crossover and mutation may produce unfeasible individuals. In this work, the three feasibility rules of Deb [2000] are employed to handle constraints in MOEAs: (1) if two individuals are feasible, the best one is chosen; (2) if one individual is feasible and the other is not, the feasible one is chosen; (3) if two individuals are infeasible, the one with the smallest degree of constraint violation is chosen.

MOEAs employ a ranking scheme to define probabilities for individuals to participate in crossover/mutation and decide which individuals survive to the next generation. Typically, the ranking scheme is the main distinguishing factor between MOEAs, while the crossover/mutation operators remain the same. A good ranking scheme should promote fast convergence and ensure that the final population is well distributed. For example, NSGAII [Deb *et al.*, 2000] is a popular genetic algorithm that favors nondominated individuals with cost vectors in less crowded areas of the cost space. In MOEA/D [Zhang and Li, 2007], each individual is assigned a scalarized single-objective version of the multi-objective problem. An individual is replaced by offsprings with smaller costs in regard to its scalarized problem.

2.3 Unsatisfiable Cores and Minimal Correction Subsets

Let ϕ be an unsatisfiable set of PB constraints. A subset $\phi_C \subseteq \phi$ is an unsatisfiable core of ϕ if and only if ϕ_C is also unsatisfiable.

Let ϕ_H and ϕ_S be sets of hard and soft PB constraints, respectively, such that ϕ_H is satisfiable and $\phi_H \cup \phi_S$ is unsatisfiable. Note that all hard constraints must be satisfied. On the other hand, soft constraints are not required to be satisfied, but the goal is to satisfy as many as possible. A subset $C \subseteq \phi_S$ is an MCS if and only if $\phi_H \cup (\phi_S \setminus C)$ is satisfiable and $\phi_H \cup (\phi_S \setminus C) \cup \{c\}$ is unsatisfiable for all $c \in C$.

Example 2.3 *Let $\phi_H = \{x_1 + x_2 + x_3 \geq 2, x_1 + x_2 = 1\}$ be the set of hard constraints and $\phi_S = \{x_1 = 0, x_2 = 0, x_3 = 0\}$ the set of soft constraints. In this case, we have two MCS: $C_1 = \{x_1 = 0, x_3 = 0\}$ and $C_2 = \{x_2 = 0, x_3 = 0\}$.*

Several techniques exist in the literature for computing unsatisfiable cores [Goldberg and Novikov, 2003; Dershowitz *et al.*, 2006] and MCSs [Bailey and Stuckey, 2005; Felfernig *et al.*, 2012; Marques-Silva *et al.*, 2013; Mencía *et al.*, 2015]. MCSs can be used to find approximate solutions of PBO instances as follows. Let $\Phi = (\phi, f(X))$ be a PBO instance, where $f(X) = \sum_{i=1}^{m} \omega_i \cdot l_i$. Let $\overline{L_f} = \bigcup_{i=1}^{m} \{(\overline{l_i})\}$ denote the set of clauses built from the negation of the literals that appear in $f(X)$. We set $\phi_H = \phi$ and $\phi_S = \overline{L_f}$ and obtain an MCS C through the application of an MCS algorithm. We abuse notation and use $f(C)$ to denote the cost of C, defined as follows: $f(C) = \sum_{(\overline{l_i}) \in C} \omega_i$.

Observe that any complete assignment α that satisfies $\phi \cup (\overline{L_f} \setminus C)$ will have cost $f(\alpha) = f(C)$, which provides an approximation of the optimal cost for Φ. Actually, any PBO instance can be reduced to finding the MCS $C \subseteq \overline{L_f}$ with minimum value of $f(C)$ [Birnbaum and Lozinskii, 2003].

Recently, it was shown that MCSs can also be used to find the Pareto front of MOCO instances. Let $\Phi = (\phi, F(X))$ be a MOCO instance with $F(X) = (f_1(X), \ldots, f_k(X))$, and $\overline{L_F} = \bigcup_{i=1}^{k} \overline{L_{f_i}}$. Terra-Neves *et al.* [2017] proved that one can find P_Φ by setting $\phi_H = \phi$ and using an MCS algorithm to enumerate the MCSs of $\overline{L_F}$.

Example 2.4 *The MOCO instance $(\phi, (f_1(X), f_2(X)))$ from example 2.2 can be solved through MCS-enumeration as follows. Let $\phi_H = \phi$ and let $\overline{L_{f1}} = \{\overline{x_1}, \overline{x_2}\}$ and $\overline{L_{f2}} = \{x_2, \overline{x_3}\}$ derived from $f_1(X)$ and $f_2(X)$ as described. Consider $\overline{L_F} = \overline{L_{f1}} \cup \overline{L_{f2}}$ the set of soft constraints. Next, enumerate the MCSs of $(\phi_H, \overline{L_F})$. In this case, the MCSs corresponding to the Pareto frontier would be $C_1 = \{\overline{x_1}, \overline{x_2}\}$ and $C_2 = \{\overline{x_2}, \overline{x_3}\}$. $C_3 = \{\overline{x_1}, x_2, \overline{x_3}\}$ is also an MCS, but its solution is dominated by the solution of C_1.*

2.4 Virtual Machine Consolidation

In this section the Virtual Machine Consolidation (VMC) problem is introduced, since it will be used in Sections 3 and 4 for a better understanding of the new operators. Moreover, VMC instances will also be used to evaluate the performance of the operators proposed in this work. VMC is a well known resource allocation problem that arises in the context of a cloud provider, where a set of running Virtual Machines (VMs) must be allocated to the servers in a data center. All VMs must be placed in some server, while minimizing energy consumption, resource wastage and VM migration costs.

The VM placement is subject to the following constraints: (1) each VM must be placed in exactly one server; (2) the total resource requirements (e.g. CPU, memory, etc) of the VMs placed in some server cannot exceed its resource capacities; (3) some VMs belong to the same job and must be placed in different servers in order to achieve higher levels of fault tolerance; and (4) the VM migration costs cannot exceed a given fixed migration budget enforced by the cloud provider. In this paper, we focus solely on VM placement variable encodings. The interested reader is referred to the literature for further details on the mathematical formulation of the VMC problem's constraints and objectives [Zheng *et al.*, 2016; Terra-Neves *et al.*, 2017].

Consider a set $V = \{v_1, \ldots, v_m\}$ of m VMs and a set $S = \{s_1, \ldots, s_n\}$ of n servers. An integer variable x_i^I is introduced for each VM $v_i \in V$, with domain $\{1, \ldots, n\}$. The value of x_i^I indicates the server that accommodates v_i, i.e., $x_i^I = j$ means that v_i is placed in s_j. We consider this encoding for the individuals in a MOEA's population. Hence, the crossover of two individuals α_1 and α_2 produces a new individual α_{12} with approximately half of the VMs placed as in α_1 and the remaining ones as in α_2. The mutation of α_{12} corresponds to picking a VM v at random and choosing a random server to accommodate v.

However, the smart operators proposed in this paper require all variables to be Boolean. For each VM-server pair $v_i \in V$ and $s_j \in S$, we introduce a Boolean variable $x_{i,j}^B$ that indicates whether v_i is placed in server s_j. Therefore, if $x_i^I = j$, we have $x_{i,j}^B = 1$ and $x_{i,k}^B = 0$ for all $k \neq j$ and vice-versa. A different encoding is used in the evolutionary algorithm because the integer encoding naturally captures the constraint that each VM must be placed in exactly one server, improving the performance of the evolutionary process. Ideally, the MOEA should also operate on the Boolean encoding in order to better generalize to other kinds of MOCO problems, but the purpose of this paper is to highlight the benefits of combining MOEAs with logic-based methods, and thus we leave its generalization for future work.

3 Smart Mutation

Smart mutation is the process of using a constraint solver to turn an unfeasible individual of a MOEA's population into a feasible one. Given an unfeasible individual, smart mutation finds a set of variable assignments that are not responsible for constraint violations, enforces those assignments, and uses a constraint solver to assign the remaining variables, thus producing a feasible individual.

Smart mutation was recently proposed by Henard *et al.* [2015] in the context of Software Product Line Configuration (SPLC). The constraints of an SPLC instance can be expressed as a set of clauses, and thus the authors proposed the SATIBEA algorithm, which uses a Boolean satisfiability solver within IBEA [Zitzler and Künzli, 2004] to correct unfeasible individuals. Given such an individual, SATIBEA considers the assignments of variables appearing in unsatisfied clauses to be the ones responsible for constraint violations. Although this rule suffices for the SPLC problem, it does not generalize to other applications, specially when

Algorithm 2 Generalized smart mutation

```
Input: $\phi, \alpha$
1: $X_A \leftarrow$ InitAssigned($\phi, \alpha$)
2: $\phi_A \leftarrow \bigcup_{x \in X_A, \alpha(x)=1}\{(x)\} \cup \bigcup_{x \in X_A, \alpha(x)=0}\{(\overline{x})\}$
3: while not SAT($\phi \cup \phi_A$) do
4:     $\phi_C \leftarrow$ UnsatCore($\phi \cup \phi_A$)
5:     $\phi_A \leftarrow \phi_A \setminus (\phi_C \cap \phi_A)$
6: end while
7: $\alpha_{feasible} \leftarrow$ Model($\phi \cup \phi_A$)
8: return $\alpha_{feasible}$
```

more general types of constraints are involved, such as PB constraints. Hence, in general, SATIBEA's smart mutation operator is not guaranteed to produce a feasible individual.

We propose a generalized smart mutation operator that, given a set of constraints ϕ and an individual α, it uses unsatisfiable cores to determine which variable assignments in α are responsible for constraint violations. Its pseudo-code is presented in Algorithm 2. It starts by initializing a set X_A of variables that are believed to not be responsible for α's constraint violations (line 1). Then, a set ϕ_A is built with clauses enforcing the partial assignment that results from restricting α to the variables in X_A (line 2). A PBS oracle is used to check if $\phi \cup \phi_A$ is satisfiable (line 3), in which case a model of $\phi \cup \phi_A$ is returned as the new individual (line 7). Otherwise, a core of $\phi \cup \phi_A$ is extracted (line 4) and the clauses of ϕ_A present in the core are removed from ϕ_A (line 5). This process is repeated until $\phi \cup \phi_A$ becomes satisfiable.

In line 1, X_A can be initialized with all problem variables and let generalized smart mutation itself determine the variables responsible for constraint violations using unsatisfiable cores. However, one may speed-up the process by exploiting domain knowledge to initialize X_A. For VMC, we unassign the following variables: (1) all variables of VMs placed in servers with overloaded resource capacities; (2) all variables of VMs of the same job placed in the same server; (3) all variables of migrated VMs if the migration budget is exceeded.

Given a model α, we prevent smart mutation from producing α again in the future by adding the clause $(\bigvee_{x \in X, \alpha(x)=1} \overline{x} \vee \bigvee_{x \in X, \alpha(x)=0} x)$ to ϕ. As a result, our smart mutation operator works in an incremental way and adds diversity to the population by ensuring to always produce a different individual. Although very unlikely, if the smart mutation operator produces a core ϕ_C such that $\phi_C \cap \phi_A = \emptyset$, then we know that the whole feasible search space has been explored and the MOEA can be terminated.

Note that PBS is an NP-Hard problem [Roussel and Manquinho, 2009]. Therefore, smart mutation should be executed sparingly. Otherwise, most of the time will be spent on the PBS oracle instead of evolving the population. The frequency of smart mutation is controlled through a configurable smart mutation rate p_{smr}. Additionally, in order to prevent smart mutation from wasting unreasonable amounts of time on hard PBS instances, one may impose a configurable conflict[1] budget b_{sm} on each run of smart mutation. If the oracle exhausts that budget, smart mutation is interrupted and the original unfeasible individual is returned instead.

Algorithm 3 Smart improvement

```
Input: $\phi, \overline{L_F}, \alpha$
1: $X_A \leftarrow$ InitAssigned($\phi, \overline{L_F}, \alpha$)
2: $\phi_A \leftarrow \bigcup_{x \in X_A, \alpha(x)=1}\{(x)\} \cup \bigcup_{x \in X_A, \alpha(x)=0}\{(\overline{x})\}$
3: $C \leftarrow$ MCS($\phi \cup \phi_A, \overline{L_F}$)
4: $\alpha_{improved} \leftarrow$ Model($\phi \cup \phi_A \cup (\overline{L_F} \setminus C)$)
5: return $\alpha_{improved}$
```

4 Smart Improvement

We propose a novel smart improvement operator, inspired on MCS-based algorithms for enumerating the Pareto front of MOCO instances [Terra-Neves *et al.*, 2017]. Given a feasible individual α, smart improvement uses an MCS oracle to produce a similar offspring α'. This process is outlined in Algorithm 3. First, similarly to smart mutation, we build a set ϕ_A of clauses enforcing a subset of α's variable assignments (lines 1 and 2). In VMC, this step corresponds to selecting a subset of VMs to relocate and enforcing the variables of the remaining VMs. We select relocating VMs randomly with probability given by a configurable relaxation rate p_{rr}. Then, an MCS oracle is invoked with $\phi_H = \phi \cup \phi_A$ and $\phi_S = \overline{L_F}$, producing an MCS C (line 3). Finally, the improved offspring is a model of $\phi \cup \phi_A \cup (\overline{L_F} \setminus C)$ (line 4). Similarly to smart mutation, the clause $(\bigvee_{(l) \in C} l)$ is added to ϕ to prevent smart improvement from producing C again in the future.

Observe that $\phi_H = \phi \cup \phi_A$ is defined as the set of hard constraints given to the MCS algorithm. In our incremental implementation, a new blocking clause is added to ϕ whenever a new solution is found in the smart mutation operator, or when a new MCS is generated. As a result, there is no guarantee that ϕ_H is satisfiable. In those situations, no new offspring is generated. Note also that finding an MCS is NP-Hard. Hence, as in smart mutation, a conflict budget b_{si} is imposed on each smart improvement run.

Any MCS algorithm can be used in the smart improvement operator. In this work, MCSs are computed using the Stratified CLD (SCLD) algorithm for MOCO proposed by Terra-Neves *et al.* [2018]. SCLD first splits the literals in $\overline{L_F}$ into a sequence of partitions $P_1 \ldots P_k$. The partitioning is such that the literals in the first few partitions are the ones with the highest impact on the cost function values. Then, for each partition P_i, SCLD sets $\phi_H = \phi \cup \phi_A \cup \bigcup_{j=1}^{i-1}(P_j \setminus C_j)$ and $\phi_S = P_i$, where C_j is the MCS computed at iteration j, and uses CLD to compute C_i. In the end, $C = \bigcup_{j=1}^{k} C_j$ is an MCS for $\phi_H = \phi \cup \phi_A$ and $\phi_S = \overline{L_F}$.

5 Experimental Results

This section evaluates the performance of the smart operators proposed in Sections 3 and 4 on instances of the VMC problem. We consider the benchmark set publicly available

[1]Modern PB solvers apply an adaptation of the conflict-driven DPLL procedure used in state-of-the-art Boolean satisfiability solvers [Roussel and Manquinho, 2009].

on the DOME project website[2]. The smart operators were implemented in the VMAlloc solver[3], a collection of algorithms for solving instances of the VMC problem which includes implementations of MOEA/D [Zhang and Li, 2007], NSGAII [Deb *et al.*, 2000] and SCLD. PBS instances were solved using Sat4j [Le Berre and Parrain, 2010][4]. We evaluate the impact of the smart operators on the performance of NSGAII and MOEA/D, and make a comparison with SCLD as well. All algorithms were configured as suggested in the literature [Terra-Neves *et al.*, 2017; Terra-Neves *et al.*, 2018]. Smart mutation was applied to each offspring produced by the regular genetic operators with probability $p_{smr} = 0.01$. If the offspring was already feasible, smart improvement was used instead with relaxation rate $p_{rr} = 0.2$. Conflict budgets were set as follows: $b_{sm} = 20000$ and $b_{si} = 500000$.

The quality of the Pareto front approximations was evaluated using the Inverted Generational Distance (IGD) [Coello Coello and Sierra, 2004] and Hypervolume (HV) [Zitzler, 1999] performance metrics. IGD is a combined measure of convergence and diversity. Given some approximation P, IGD measures the average Euclidean distance from the cost vectors in some reference front P_R (ideally the Pareto front if known) to the closest vector in P. Smaller values of IGD indicate that the approximation is composed of solutions of higher quality in terms of convergence and/or diversity. We combined the approximations produced by all algorithms evaluated in this section to build the reference fronts for each VMC instance. HV is another combined metric that measures the volume of the cost space dominated by the approximation P, up to a given reference point, and thus larger values are preferred. All cost vectors in P and P_R are normalized before computing IGD and HV. The reference point coordinates for HV are set to $1 + \frac{1}{|P_R|}$.

We consider one additional metric, based on the primal integral used in the evaluation of mixed integer programming heuristics [Berthold, 2013], referred to as Unfeasible Integral (UI). Let $u(t)$ denote the number of unfeasible solutions in the population observed at instant t during the execution of some MOEA. Assuming a time limit of T seconds, the UI is the integral of $u(t)$, in interval $[0, T]$, divided by T. Smaller values of UI indicate that the algorithm maintains a larger average count of feasible individuals in the population throughout its execution. Observe that, in order to perform a fair comparison, UI should only be considered if the population size is the same for all algorithms (100 in this evaluation).

Each algorithm was executed 10 times with different seeds for each instance, and the analysis is performed using the median values over all executions. Statistical tests were performed with confidence intervals of 95% in order to assess the significance of the results. These were successful for at least 60% of all instances for all algorithm pairs. Memory and time limits of 4 GB and 1800 seconds respectively were used. The evaluation was conducted on an AMD Opteron 6376 (2.3 GHz) with 128 GB of RAM, running Debian jessie.

Figures 1 to 9 show the median UI, IGD and HV distributions obtained with each algorithm for VMC instances with migration budgets equal to 100%, 5% and 1% of the data centers total memory capacity. We refer to NSGAII (MOEA/D) with smart mutation enabled as NSGAII-SM (MOEA/D-SM). NSGAII-SI (MOEA/D-SI) corresponds to NSGAII (MOEA/D) with smart mutation plus improvement. A point (x, y) in an UI/IGD (HV) distribution plot indicates that the given algorithm obtained an UI/IGD (HV) equal to or lower (greater) than y for x instances. For example, the point $(100, 81)$ on MOEA/D-SM's line in Figure 1 indicates that MOEA/D-SM obtained UIs equal to or smaller than 81 for 100 instances. In order to improve the readability of Figures 4 to 6, we only display IGD values up to 1.2.

5.1 Impact of Smart Mutation

Figures 2 and 3 show that, in the 5% and 1% migration budget instances, smart mutation significantly improves the number of feasible solutions in the population for both MOEA/D and NSGAII. Moreover, we have also observed a faster convergence when smart mutation is used, since feasible individuals are found in earlier iterations of MOEA/D and NSGAII. Because the constraints of the 100% instances are easier to satisfy, we observe a lack of improvement in Figure 1. MOEA/D and NSGAII are already able to find many feasible solutions for the 100% instances without smart mutation, and smart operators are significantly slower than typical genetic operators. Nonetheless, NSGAII-SM is able to find feasible solutions for all 300 100% instances, 28 (64) more than NSGAII (MOEA/D). MOEA/D-SM does so for 297 of those instances.

In Figures 5, 6, 8 and 9, we can see a significant improvement in terms of IGD and HV for both NSGAII-SM and MOEA/D-SM for the 5% and 1% instances. Therefore, by injecting the population with feasible solutions, smart mutation enables the MOEAs to find high quality Pareto front approximations and even outperform SCLD, the state-of-the-art for such instances. Figures 4 and 7 show that smart mutation barely impacts the performance of MOEA/D in the 100% instances. However, we see some performance degradation for NSGAII. We can conclude that smart mutation improves the performance of MOEAs in tightly constrained problems, at the expense of some performance in loosely constrained ones.

5.2 Impact of Smart Improvement

Figures 1 to 3 show a degradation in terms of UI when smart improvement is enabled. This is expected since smart improvement is much more expensive than smart mutation, and thus less time is spent searching for feasible solutions. Nonetheless, the number of instances for which at least one feasible solution is found is similar with and without smart improvement. Moreover, Figures 4 to 9 show a considerable improvement for both MOEA/D-SI and NSGAII-SI in terms of approximation quality, particularly in HV. Not only does smart improvement enhances the performance in tightly constrained instances, it compensates for the performance degradation observed in loosely constrained ones for NSGAII.

6 Conclusion and Future Work

Recently, constraint-based methods for MOCO have been shown to outperform state-of-the-art evolutionary algorithms,

[2]http://sat.inesc-id.pt/dome

[3]https://github.com/MiguelTerraNeves/VMAlloc

[4]https://gitlab.ow2.org/sat4j/sat4j (27 Jan. 2019)

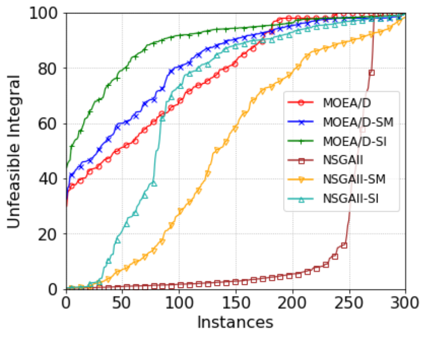

Figure 1: UI distributions (100%).

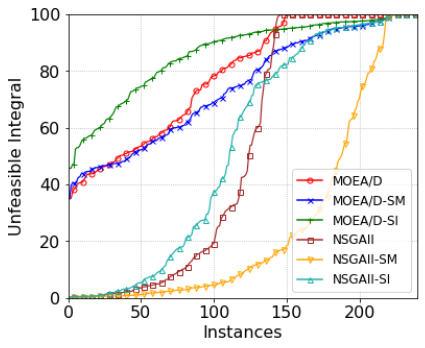

Figure 2: UI distributions (5%).

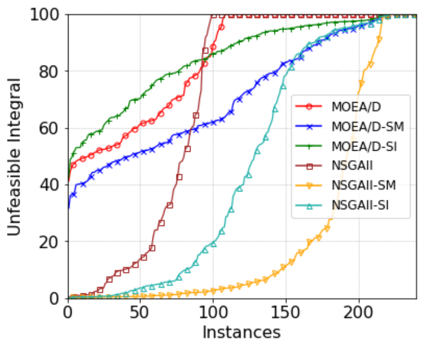

Figure 3: UI distributions (1%).

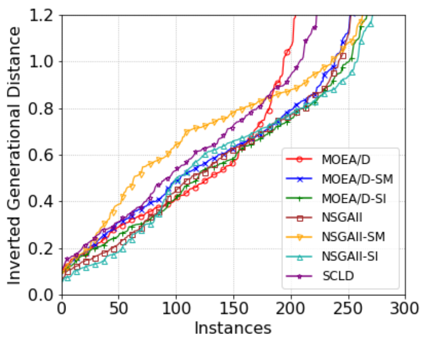

Figure 4: IGD distributions (100%).

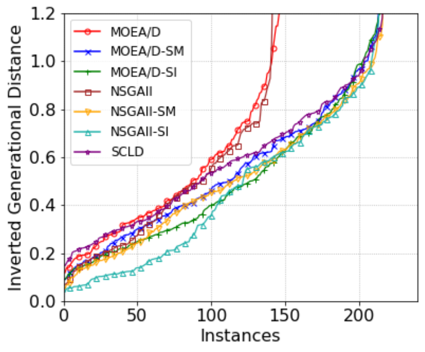

Figure 5: IGD distributions (5%).

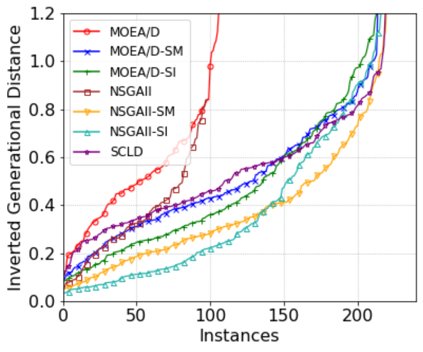

Figure 6: IGD distributions (1%).

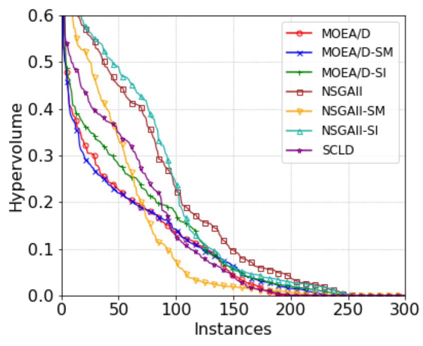

Figure 7: HV distributions (100%).

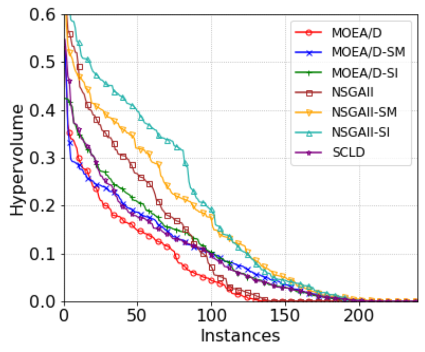

Figure 8: HV distributions (5%).

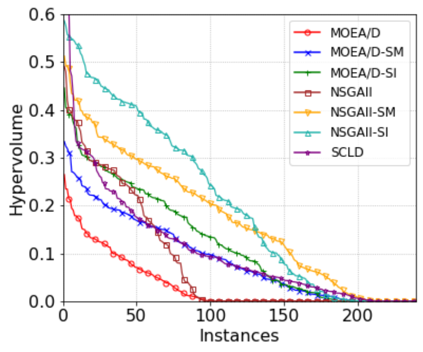

Figure 9: HV distributions (1%).

such as NSGAII and MOEA/D, when the problem's constraints are hard to satisfy [Terra-Neves *et al.*, 2018]. Following previous work on using constraint solvers in MOEAs [Henard *et al.*, 2015; Xiang *et al.*, 2018], we propose a generalized smart mutation operator for correcting unfeasible individuals, and a smart improvement operator that uses MCSs to improve already feasible individuals. Experimental results in a large set of VMC instances show that the combination of evolution and constraint solving significantly outperforms pure evolutionary and constraint-based approaches.

Recall that, currently, an integer encoding for individuals is used that is more suitable for VMC than a Boolean encoding. As future work, we propose to evaluate the performance of the Boolean encoding and investigate possible new encodings suitable for Boolean MOCOs. Moreover, general heuristics should be developed for the assigned variable initialization step of the smart mutation and improvement procedures. One should also explore deeper integration between MOEAs and constraint solvers (e.g. exploiting the scalarization weights when applying smart improvement within MOEA/D). Lastly, the performance of these techniques should be evaluated on other MOCO benchmarks besides VMC, such as SPLC.

Acknowledgments

This work was supported by national funds through FCT with references UID/CEC/50021/2019, SFRH/BD/111471/2015, PTDC/CCI-COM/31198/2017 and DSAIPA/AI/0044/2018.

References

[Bailey and Stuckey, 2005] J. Bailey and P. Stuckey. Discovery of Minimal Unsatisfiable Subsets of Constraints Using Hitting Set Dualization. In *Symposium on Practical Aspects of Declarative Languages*, pages 174–186, 2005.

[Berthold, 2013] T. Berthold. Measuring the Impact of Primal Heuristics. *Operations Research Letters*, 41(6):611–614, 2013.

[Birnbaum and Lozinskii, 2003] E. Birnbaum and E. Lozinskii. Consistent Subsets of Inconsistent Systems: Structure and Behaviour. *Journal of Experimental & Theoretical Artificial Intelligence*, 15(1):25–46, 2003.

[Boros and Hammer, 2002] E. Boros and P. L. Hammer. Pseudo-Boolean Optimization. *Discrete Applied Mathematics*, 123(1):155–225, 2002.

[Coello Coello and Sierra, 2004] C. A. Coello Coello and M. R. Sierra. A Study of the Parallelization of a Coevolutionary Multi-objective Evolutionary Algorithm. In *Mexican International Conference on Artificial Intelligence*, pages 688–697, 2004.

[Deb *et al.*, 2000] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimisation: NSGA-II. In *International Conference on Parallel Problem Solving from Nature*, pages 849–858. Springer, 2000.

[Deb, 2000] K. Deb. An Efficient Constraint Handling Method for Genetic Algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):311–338, 2000.

[Dershowitz *et al.*, 2006] N. Dershowitz, Z. Hanna, and A. Nadel. A Scalable Algorithm for Minimal Unsatisfiable Core Extraction. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 36–41, 2006.

[Felfernig *et al.*, 2012] A. Felfernig, M. Schubert, and C. Zehentner. An Efficient Diagnosis Algorithm for Inconsistent Constraint Sets. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 26(1):53–62, 2012.

[Goldberg and Novikov, 2003] E. Goldberg and Y. Novikov. Verification of Proofs of Unsatisfiability for CNF Formulas. In *Conference on Design, Automation and Test in Europe*, pages 10886–10891, 2003.

[Henard *et al.*, 2015] C. Henard, M. Papadakis, M. Harman, and Y. Le Traon. Combining Multi-Objective Search and Constraint Solving for Configuring Large Software Product Lines. In *International Conference on Software Engineering*, pages 517–528, 2015.

[Le Berre and Parrain, 2010] D. Le Berre and A. Parrain. The Sat4j Library, Release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2-3):59–6, 2010.

[Marques-Silva *et al.*, 2013] J. Marques-Silva, F. Heras, M. Janota, A. Previti, and A. Belov. On Computing Minimal Correction Subsets. In *International Joint Conference on Artificial Intelligence*, pages 615–622, 2013.

[Mencía *et al.*, 2015] C. Mencía, A. Previti, and J. Marques-Silva. Literal-Based MCS Extraction. In *International Joint Conference on Artificial Intelligence*, pages 1973–1979, 2015.

[Roussel and Manquinho, 2009] O. Roussel and V. Manquinho. Pseudo-Boolean and Cardinality Constraints. In *Handbook of Satisfiability*, pages 695–733. 2009.

[Soh *et al.*, 2017] T. Soh, M. Banbara, N. Tamura, and D. Le Berre. Solving Multiobjective Discrete Optimization Problems with Propositional Minimal Model Generation. In *International Conference on Principles and Practice of Constraint Programming*, pages 596–614. Springer, 2017.

[Spears and De Jong, 1995] W. M. Spears and K. D. De Jong. On the Virtues of Parameterized Uniform Crossover. Technical report, Naval Research Lab, 1995.

[Syswerda, 1989] G. Syswerda. Uniform Crossover in Genetic Algorithms. In *International Conference on Genetic Algorithms*, pages 2–9, 1989.

[Terra-Neves *et al.*, 2017] M. Terra-Neves, I. Lynce, and V. Manquinho. Introducing Pareto Minimal Correction Subsets. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 195–211, 2017.

[Terra-Neves *et al.*, 2018] M. Terra-Neves, I. Lynce, and V. Manquinho. Stratification for Constraint-Based Multi-Objective Combinatorial Optimization. In *International Joint Conference on Artificial Intelligence*, pages 1376–1382, 2018.

[Ulungu and Teghem, 1994] E. L. Ulungu and J. Teghem. Multi-Objective Combinatorial Optimization Problems: A Survey. *Journal of Multi-Criteria Decision Analysis*, 3(2):83–104, 1994.

[Xiang *et al.*, 2018] Y. Xiang, Y. Zhou, Z. Zheng, and M. Li. Configuring Software Product Lines by Combining Many-Objective Optimization and SAT Solvers. *ACM Transactions on Software Engineering and Methodology*, 26(4):14:1–14:46, 2018.

[Zhang and Li, 2007] Q. Zhang and H. Li. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

[Zheng *et al.*, 2016] Q. Zheng, R. Li, X. Li, N. Shah, J. Zhang, F. Tian, K. Chao, and J. Li. Virtual Machine Consolidated Placement Based on Multi-objective Biogeography-based Optimization. *Future Generation Computer Systems*, 54:95–122, 2016.

[Zitzler and Künzli, 2004] E. Zitzler and S. Künzli. Indicator-Based Selection in Multiobjective Search. In *International Conference on Parallel Problem Solving from Nature*, pages 832–842, 2004.

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