The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Enhancing Constraint-Based Multi-Objective Combinatorial Optimization

Miguel Terra-Neves, Inês Lynce, Vasco Manquinho
INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, Portugal
{neves, ines, vmm}@sat.inesc-id.pt

Abstract

Minimal Correction Subsets (MCSs) have been successfully applied to find approximate solutions to several real-world single-objective optimization problems. However, only recently have MCSs been used to solve Multi-Objective Combinatorial Optimization (MOCO) problems. In particular, it has been shown that all optimal solutions of MOCO problems with linear objective functions can be found by an MCS enumeration procedure.

In this paper, we show that the approach of MCS enumeration can also be applied to MOCO problems where objective functions are divisions of linear expressions. Hence, it is not necessary to use a linear approximation of these objective functions. Additionally, we also propose the integration of diversification techniques on the MCS enumeration process in order to find better approximations of the Pareto front of MOCO problems. Finally, experimental results on the Virtual Machine Consolidation (VMC) problem show the effectiveness of the proposed techniques.

1 Introduction

The integration of propositional satisfiabilty (SAT) solvers in algorithms to solve optimization problems such as Maximum Satisfiability (MaxSAT) or Pseudo-Boolean Optimization (PBO) resulted in a new generation of highly effective tools to solve optimization problems. Nevertheless, there are still a wide variety of problems where state of the art MaxSAT or PBO solvers cannot find an optimal solution in useful time. A well-known approach to approximate optimal solutions for these optimization problems is by enumerating Minimal Correction Subsets.

A Minimal Correction Subset (MCS) of an unsatisfiable set of constraints F is a minimal subset C such that if all constraints in C are removed from F, then F becomes satisfiable. Recently, several algorithms for effective MCS enumeration have been proposed (Bacchus et al. 2014; Felfernig, Schubert, and Zehentner 2012; Grégoire, Lagniez, and Mazure 2014; Marques-Silva et al. 2013) and successfully applied to obtain good quality solutions for problems where complete algorithms are unable to do it.

In many real-world applications, such as scheduling (Iturriaga, Dorronsoro, and Nesmachnow 2017) or green computing (Zheng et al. 2016), there is more than one objective (also known as cost function) to be optimized. In Multi-Objective Combinatorial Optimization (MOCO), there may exist multiple optimal solutions, known as Pareto optimal solutions, each of them favoring certain objectives at the expense of others. Furthermore, finding the Pareto front, i.e. all Pareto optimal solutions, is well-known to be very hard. Hence, for large MOCO problem instances, current algorithms are usually just able to approximate the Pareto front.

There is a wide variety of algorithms to solve MOCO problems, ranging from evolutionary algorithms that approximate the Pareto front (Deb et al. 2000; Xu and Fortes 2010) to exact methods (Jackson et al. 2009). Nevertheless, despite all its success in optimization with a single cost function, only very recently have constraint-based methods been proposed to solve MOCO. For instance, the Guided Improvement Algorithm (GIA) (Jackson et al. 2009), is implemented in the optimization engine of solver Z3 for finding Pareto optimal solutions of Satisfiability Modulo Theories (SMT) instances with multiple cost functions. Recently, it has been shown that one can find the Pareto front of a MOCO instance by enumerating the set of MCSs (Terra-Neves, Lynce, and Man qui nho 2017) or the P-minimal models (Soh et al. 2017) of a propositional formula.

This paper enhances constraint-based approaches for solving MOCO with the following contributions: (1) a proof that enumeration of MCSs can be used even when the cost functions in MOCO are defined as divisions of linear expressions; (2) two new diversification methods for enumerating MCSs in order to quickly find better approximations of the Pareto front; (3) an extensive experimental evaluation on instances of the virtual machine consolidation (VMC) problem taken from the Google Cluster Data project that show the impact of the proposed ideas in solving MOCO instances.

The paper is organized as follows. Section 2 introduces several basic definitions, including a formal definition of MOCO. In section 3, a constraint-based method is proposed for solving MOCO problem instances containing cost functions that are divisions of linear expressions. Section 4 proposes MCS sampling and path-based enumeration to obtain a wider diversification of MCS, resulting in an improved approximation of the Pareto front. Section 5 shows the effectiveness of the proposed techniques in a MOCO formulation of the VMC problem, and section 6 concludes the paper.

Copyright © 2018, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

2 Preliminaries

This section introduces the necessary definitions and notations that will be used in the reminder of the paper. First, Pseudo-Boolean Optimization (PBO) and Minimal Correction Subsets (MCSs) are defined. Next, the ClauseD (CLD) algorithm for MCS enumeration is reviewed. Finally, Multi-Objective Combinatorial Optimization (MOCO) is defined.

2.1 Pseudo-Boolean Optimization

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of n Boolean variables. A literal is either a variable x_i or its complement $\neg x_i$. Given a set of m literals $l_1, l_2, \ldots, l_m$ and their respective coefficients $\omega_1, \omega_2, \ldots, \omega_m \in \mathbb{Z}$, a Pseudo-Boolean (PB) expression has the following form:

$$\sum \omega_i \cdot l_i. \tag{1}$$

Given an integer $k \in \mathbb{Z}$, a PB constraint is a linear inequality with the form:

$$\sum \omega_i \cdot l_i \bowtie k, \quad \bowtie \in \{\leq, \geq, =\}. \tag{2}$$

Given a set $F = \{c_1, c_2, \ldots, c_k\}$ of k PB constraints defined over a set of X Boolean variables, the Pseudo-Boolean Satisfiability (PBS) problem consists of deciding if there exists a complete assignment $\alpha : X \rightarrow \{0, 1\}$, such that all PB constraints in F are satisfied. If that is the case, we say that F is satisfiable and α satisfies F, denoted $\alpha(F) = 1$. Otherwise, we say that F is unsatisfiable and $\alpha(F) = 0$ for any assignment α. Analogously, given a PB constraint c, $\alpha(c) = 1$ ($\alpha(c) = 0$) denotes that α satisfies (does not satisfy) c. In PBO (Boros and Hammer 2002), given a set of PB constraints F and a cost function f defined as a linear PB expression over a set of Boolean variables X, the goal is to compute a complete assignment to the variables in X that satisfies all of the constraints in F and minimizes the value of f. Given a complete assignment α, we denote as $f(\alpha)$ the cost of α.

Example 2.1. *Let $F = \{(x_1 + x_2 + x_3 \geq 2)\}$ be the set of PB constraints and $f(X) = 4 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_3$ the cost function of a PBO instance. $\alpha_1 = \{(x_1, 0), (x_2, 1), (x_3, 1)\}$ is an optimal assignment and has a cost of 5. $\alpha_2 = \{(x_1, 1), (x_2, 0), (x_3, 1)\}$ is not an optimal assignment because it has a cost of 7. $\alpha_3 = \{(x_1, 0), (x_2, 1), (x_3, 0)\}$ is an invalid assignment because it does not satisfy F.*

For simplicity reasons, given p PB constraints $c_1, c_2, \ldots, c_p$, the disjunction operator $\vee$ is used to represent the constraint that at least one of the PB constraints must be satisfied (e.g. $c_1 \vee c_2 \vee \cdots \vee c_p$). Note that such disjunctions can be easily converted to sets of PB constraints using auxiliary variables. Finally, note that propositional clauses are special cases of PB constraints. For example, $l_1 \vee \ldots \vee l_m$ is equivalent to $\sum_{j=1}^{m} l_j \geq 1$. Hence, for ease of notation, in some cases, the usual clause notation is used.

2.2 Minimal Correction Subsets

Given an unsatisfiable set of PB constraints F, a Minimal Correction Subset (MCS) is a minimal subset $C \subseteq F$ such that $F \setminus C$ is satisfiable.

Algorithm 1: CLD algorithm for computing an MCS

```
   Input: F_H, F_S
1  S ← F_H
2  C ← F_S
3  status ← SAT
4  while status = SAT do
5  |  D ← ⋁_{c∈C} c
6  |  (status, α) ← PBS(S ∪ {D})
7  |  if status = SAT then
8  |  |  S ← S ∪ ⋃_{c∈C, α(c)=1} {c}
9  |  |  C ← F_S \ S
10 return C
```

Definition 2.1. *Let F be an unsatisfiable set of PB constraints. A subset $C \subseteq F$ is an MCS of F if, and only if, $F \setminus C$ is satisfiable and $(F \setminus C) \cup \{c\}$ is unsatisfiable for all $c \in C$.*

Example 2.2. *Consider the unsatisfiable set of PB constraints $F = \{(x_1 + x_2 = 1), (x_1 \geq 1), (x_2 \geq 1)\}$. F has three MCSs $C_1 = \{(x_1 \geq 1)\}$, $C_2 = \{(x_2 \geq 1)\}$ and $C_3 = \{(x_1 + x_2 = 1)\}$.*

Several algorithms exist for finding MCSs (Bailey and Stuckey 2005; Felfernig, Schubert, and Zehentner 2012; Marques-Silva et al. 2013; Mencía, Previti, and Marques-Silva 2015). For the purpose of this work, the state-of-the-art CLD algorithm is used (Marques-Silva et al. 2013). CLD's pseudo-code is presented in algorithm 1. It receives as argument a set of hard constraints F_H that must be satisfied, and a set F_S of soft constraints for which we want to find an MCS $C \subseteq F_S$. If looking for an MCS of a PBS F, we have $F_H = \emptyset$ and $F_S = F$. For simplicity, we assume that F_H is always satisfiable (this can be checked using a single call to a PBS solver). The CLD algorithm starts by initializing the sets S and C of satisfied and not satisfied PB constraints respectively (lines 1 and 2). Initially, all constraints in F_S are not satisfied. Then, it repeatedly checks if it is possible to satisfy at least one of the constraints in C, while satisfying all constraints in S (lines 5 and 6). If so, then sets S and C are updated accordingly (lines 8 and 9). If not, then C is an MCS and is returned by the algorithm (line 10).

Algorithm 1 computes a single MCS C, but it can be used to find another MCS by incorporating the constraint $\bigvee_{c \in C} c$ in F_H and re-executing the algorithm. Such a constraint *blocks* MCS C from being identified again by the algorithm. Hence, the CLD algorithm can be used to enumerate all MCSs of F_S by blocking previous MCSs in subsequent invocations of the algorithm.

MCSs can be used to find approximate solutions of PBO instances as follows. Let F be the set of PB constraints and $f(X) = \sum \omega_i \cdot l_i$ the cost function of a PBO formula. Let $L(f)$ be the set of all literals in f and $L^{\neg}(f)$ the set of clauses built from the negation of the literals in $L(f)$, i.e., $L^{\neg}(f) = \bigcup_{l_i \in L(f)} \{(\neg l_i)\}$. Applying algorithm 1 with $F_H = F$ and $F_S = L^{\neg}(f)$, produces an MCS C of $L^{\neg}(f)$. We abuse notation and denote as $f(C)$ the cost of C, thus

Table 1: Possible assignments and respective costs for the instance in example 2.4.

x_1	x_2	$x_1 + 2 \cdot \neg x_2$	$\neg x_1$
0	0	2	1
0	**1**	**0**	**1**
1	**0**	**3**	**0**
1	1	-	-

$f(C)$ being given by

$$f(C) = \sum_{(\neg l_i) \in C} \omega_i. \tag{3}$$

Any assignment that satisfies $F \cup L^{\neg}(f) \setminus C$ will have a cost of $f(C)$, which provides an approximation of the optimum of the PBO instance. Actually, the PBO problem can be reduced to finding the MCS $C \subseteq L^{\neg}(f)$ that minimizes $f(C)$ (Birnbaum and Lozinskii 2003).

Example 2.3. *Consider again example 2.1 where* $F = \{(x_1 + x_2 + x_3 \geq 2)\}$ *and* $f(X) = 4 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_3$ *We have* $L^{\neg}(f) = \{(\neg x_1), (\neg x_2), (\neg x_3)\}$. *Hence, there are three MCSs* $C_1 = \{(\neg x_2), (\neg x_3)\}$, $C_2 = \{(\neg x_1), (\neg x_3)\}$ *and* $C_3 = \{(\neg x_1), (\neg x_2)\}$, *with costs 5, 7 and 6 respectively.* C_1 *is the minimum cost MCS. Therefore, any assignment that satisfies* $\{(x_1 + x_2 + x_3 \geq 2), (\neg x_1)\}$ *is an optimal solution of the PBO instance.*

2.3 Multi-Objective Combinatorial Optimization

A Multi-Objective Combinatorial Optimization (MOCO) (Ulungu and Teghem 1994) instance is composed of two sets: a set $F = \{c_1, c_2, \ldots, c_k\}$ of constraints that must be satisfied and a set $O = \{f_1, f_2, \ldots, f_l\}$ of cost functions to minimize. In this work, we focus on the special case where $c_1, c_2, \ldots, c_k$ are PB constraints and $f_1, f_2, \ldots, f_l$ are PB expressions over a set X of Boolean variables.

Definition 2.2. *Let* $\mathbb{M} = (F, O)$ *be a MOCO instance, where* F *and* O *are the constraint and cost function sets, respectively. Let* $\alpha, \alpha' : X \to \{0, 1\}$ *be two complete assignments such that* $\alpha \neq \alpha'$ *and* $\alpha(F) = \alpha'(F) = 1$. *We say that* α *dominates* α', *written* $\alpha \prec \alpha'$, *if, and only if,* $\forall_{f \in O} f(\alpha) \leq f(\alpha')$ *and* $\exists_{f' \in O} f'(\alpha) < f'(\alpha')$.

Definition 2.3. *Let* $\mathbb{M} = (F, O)$ *be a MOCO instance and* $\alpha : X \to \{0, 1\}$ *a complete assignment.* α *is said to be Pareto optimal if, and only if,* $\alpha(F) = 1$ *and no other complete assignment* α' *exists such that* $\alpha'(F) = 1$ *and* $\alpha' \prec \alpha$.

In MOCO, the goal is to find the set of Pareto optimal solutions, also referred to as solution set. Terra-Neves, Lynce, and Man qui nho(2017) proved that finding all Pareto optimal solutions can be reduced to enumerating all MCSs of $L^{\neg}(O) = \bigcup_{f \in O} L^{\neg}(f)$ and filtering out the ones that correspond to dominated assignments.

Example 2.4. *Let* $F = \{(x_1 + x_2 \leq 1)\}$ *be the set of PB constraints and* $O = \{(x_1 + 2 \cdot \neg x_2), (\neg x_1)\}$ *the set of objective functions of a MOCO instance. Table 1 shows the costs for each possible assignment. The lines that correspond to Pareto optimal solutions are highlighted in bold. Note that* $\{(x_1, 1), (x_2, 1)\}$ *violates the constraint in* F. *Hence, it is not a valid assignment.* $\{(x_1, 0), (x_2, 0)\}$ *is not Pareto optimal because it is dominated by* $\{(x_1, 0), (x_2, 1)\}$. *However,* $\{(x_1, 0), (x_2, 1)\}$ *and* $\{(x_1, 1), (x_2, 0)\}$ *are Pareto optimal solutions because they are not dominated by any other assignment that satisfies* F.

3 Handling Cost Function with Divisions

This section describes a new method to handle cost functions that are divisions of PB expressions, i.e., cost functions with the following form:

$$\frac{\sum \omega_i \cdot l_i}{\sum \omega_j \cdot l_j}. \tag{4}$$

Note that these objective functions occur frequently in scheduling and timetabling problems where one of the goals is to optimize the ratio of occupancy of the selected resources. Our method is based on the observation that, under certain conditions, the minimization of a cost function $f(X) = \frac{g(X)}{h(X)}$ can be reduced to finding the Pareto optimal solutions of the multi-objective problem with cost functions $O = \{g, -h\}$ and choosing the one that minimizes f.

Proposition 1. *Consider a set of Boolean variables* X *and let* F *be the set of PB constraints and* f *the cost function to optimize defined as* $f(X) = \frac{g(X)}{h(X)}$. *If* $g(X) \geq 0$ *and* $h(X) > 0$, *then an assignment* α *which satisfies* F *and minimizes* f *is a Pareto optimal solution of the multi-objective problem with constraint set* F *and cost function set* $O = \{g, -h\}$.

Proof. Let α be an assignment that satisfies F and minimizes f. Therefore, no α' exists such that α' satisfies F and $f(\alpha') < f(\alpha)$. Considering the multi-objective problem with constraint set F and $O = \{g, -h\}$, let α' be an assignment such that $\alpha' \prec \alpha$. There are two possible scenarios: (1) $g(\alpha') < g(\alpha)$ and $-h(\alpha') \leq -h(\alpha)$; or (2) $g(\alpha') \leq g(\alpha)$ and $-h(\alpha') < -h(\alpha)$. In the first scenario, because $g(X) \geq 0$ and $h(X) > 0$, we have

$$f(\alpha') = \frac{g(\alpha')}{h(\alpha')} \leq \frac{g(\alpha')}{h(\alpha)} < \frac{g(\alpha)}{h(\alpha)} = f(\alpha), \tag{5}$$

which is a contradiction. The same occurs in the second scenario. Therefore, α' cannot exist and α must be a Pareto optimal solution of the multi-objective problem. □

Corollary 1. *Let* (F, O) *be a MOCO problem with constraint set* F *and cost function set* O, *defined over a set of Boolean variables* X. *Let* $f \in O$ *be a cost function of the form* $f(X) = \frac{g(X)}{h(X)}$. *Let* A *and* A' *be the Pareto optimal solution sets of* (F, O) *and* $(F, O \setminus \{f\} \cup \{g, -h\})$, *respectively. If* $g(X) \geq 0$ *and* $h(X) > 0$, *then* $A \subseteq A'$.

Proof. Let A be the solution set of (F, O) and $\alpha \in A$ some Pareto optimal solution. Let $f \in O$ be a cost function of the form $f(X) = \frac{g(X)}{h(X)}$. Consider the single-objective optimization problem with constraint set $F' = F \cup \bigcup_{f' \in O, f' \neq f} \{f'(X) = f'(\alpha)\}$ and cost function f. From proposition 1, it follows that α is a Pareto optimal solution of $(F', \{g, -h\})$ and, consequently, of $(F, O \setminus \{f\} \cup \{g, -h\})$. □

Let (F, O) be a MOCO problem with cost functions composed of divisions of PB expressions. Corollary 1 entails that (F, O) can be reduced to a linear MOCO as follows: (1) for each $f \in O$ with the form $f(X) = \frac{g(X)}{h(X)}$, replace f in O with g and $-h$, producing a cost function set O'; (2) find the solution set A' of (F, O'); (3) build the solution set of (F, O) from A' by filtering out dominated solutions according to the original cost function set O.

It may be the case that f is not a division but a sum of divisions, i.e., $f(X) = \frac{g_1(X)}{h_1(X)} + \frac{g_2(X)}{h_2(X)} + \cdots + \frac{g_k(X)}{h_k(X)}$. In this scenario, we can replace f with $2k$ cost functions $g_1, g_2, \ldots, g_k, -h_1, -h_2, \ldots, -h_k$, as long as $g_i(X) \geq 0$ and $h_i(X) > 0$ for all $i \in \{1, 2, \ldots, k\}$. The proof for this is very similar to the proofs of proposition 1 and corollary 1, and is omitted due to space restrictions.

4 Diversification

In this section, two diversification mechanisms to improve diversity of MCSs are introduced. Several techniques have already been proposed to generate diverse solutions in propositional formulas (Nadel 2011). In this work, first we propose a technique that makes use of XOR hash functions (Gomes, Sabharwal, and Selman 2006; Chakraborty, Meel, and Vardi 2013) in order to sample MCSs instead of enumerating them. The second technique partitions the search tree based on the literals with the largest coefficients in each cost function. Then, CLD algorithm is used to search for an MCS in each partition.

4.1 MCS Sampling

Given positive integers n and m, we consider the family $H_{xor}(n, m)$ of hash functions, which is defined as follows:

$$H_{xor}(n, m) = \left\{ h : h(y)[i] = a_{i,0} \oplus \left(\bigoplus_{k=1}^{n} a_{i,k} \wedge y[k] \right), \; a_{i,k} \in \{0, 1\}, 1 \leq i \leq m \right\} \tag{6}$$

where $\oplus$ denotes the XOR Boolean operator. Observe that a hash function $h \in H_{xor}(n, m)$ maps a bit-string of length n to another of length m. (Gomes, Sabharwal, and Selman 2006) proved that $H_{xor}(n, m)$ is 3-universal. Consequently, $H_{xor}(n, m)$ is also 2-universal, which guarantees that, given $h \in H_{xor}(n, m)$ and $x, y \in \{0, 1\}^n$ such that $x \neq y$, the probability that $h(x) = h(y)$ is 2^{-m} (Carter and Wegman 1977). One can choose a random hash function in $H_{xor}(n, m)$ by uniformly and independently choosing the values of the $a_{i,k}$ coefficients in equation (6).

In algorithm 2, we present the SampleMCS algorithm for sampling a single MCS of a MOCO instance. SampleMCS has a single configuration parameter: the bit-string length m. It starts by randomly choosing a hash function $h \in H_{xor}(|X|, m)$ (line 1) and a sequence β of m bits (line 2). The CLD algorithm is called with the additional constraint that $h(X) = \beta$ (line 3), producing a potential

Algorithm 2: SampleMCS algorithm for sampling a single MCS of a MOCO instance

Input: F, O, m

1 $h \leftarrow$ random hash function in $H_{xor}(|X|, m)$
2 $\beta \leftarrow$ random bit sequence in $\{0, 1\}^m$
3 $C \leftarrow \texttt{CLD}(F \cup \texttt{Encode}(h(X) = \beta), L^{\neg}(O))$
4 **if** $F \cup (L^{\neg}(O) \setminus C) \cup \{(\bigvee_{l \in C} l)\}$ *is satisfiable* **then**
5 $\quad C \leftarrow \texttt{CLD}(F \cup (L^{\neg}(O) \setminus C), C)$
6 **return** C

MCS C. Then, SampleMCS checks if C is an MCS of the original instance (line 4). If so, then the algorithm terminates returning C (line 6). Otherwise, the hash function constraint is discarded and CLD continues its execution (line 5). Similarly to CLD, SampleMCS can also be used to sample multiple MCSs by including the constraint $(\bigvee_{l \in C} l)$ in F and re-running the algorithm.

Finally, we note that other MCS algorithms (Bailey and Stuckey 2005; Felfernig, Schubert, and Zehentner 2012; Mencía, Previti, and Marques-Silva 2015) can be used instead of CLD. Although not explicitly stated in the pseudocode, it is possible that adding an hash function $h(X) = \beta$ might result in an unsatisfiable formula (line 3). In that case, no MCS is generated.

4.2 Path Diversification

Given the set of variables X of a MOCO, the idea behind path diversification is to select a subset $X_P \subset X$ and partition the search tree considering all possible assignments to the variables in X_P. For each cost function $f \in O$, n_P variables of f are selected for X_P as follows: (1) sort the literals in $L(f)$ in decreasing order of their respective coefficients in f; (2) add the variables of the first n_P literals to X_P, skipping literals of variables already in X_P.

Example 4.1. *Let $O = \{f_1, f_2\}$, where $f_1 = 2 \cdot x_1 + 4 \cdot \neg x_2 + 3 \cdot x_3$ and $f_2 = 5 \cdot x_2 + \neg x_4 + 3 \cdot x_5$, be the cost function set of a MOCO instance. After sorting, we have $L(f_1) = \{\neg x_2, x_3, x_1\}$ and $L(f_2) = \{x_2, x_5, \neg x_4\}$. Assuming $n_P = 2$, we select variables x_2 and x_3 from $L(f_1)$, and then x_5 and x_4 from $L(f_2)$. In the end, we have $X_P = \{x_2, x_3, x_4, x_5\}$.*

After variable selection, a queue P of diversification paths is generated. A diversification path is a complete assignment of the variables in X_P. We prioritize switching the values of the variables that correspond to the literals with the largest coefficients in the cost functions, alternating between functions. In other words, we begin by considering the first variable selected from $L(f_1)$, then the first from $L(f_2)$, and so on. Only after traversing all cost functions we consider the second variable selected from each set.

Example 4.2. *Recall the MOCO instance in example 4.1. Variables x_2 and x_3 (x_5 and x_4) have coefficients 4 and 3 in f_1 (3 and 1 in f_2). Therefore, the variables are considered in the order x_2, x_5, x_3, x_4, and the corresponding diversifi-*

Algorithm 3: MCSEnumPD algorithm for enumerating MCSs of a MOCO instance using path diversification

Input: F, O, n_P
1 $M \leftarrow \emptyset$
2 $P \leftarrow$ `GeneratePaths` (F, O)
3 **while** $P \neq \emptyset$ **do**
4 $\quad \alpha_P \leftarrow$ `PopFront` (P)
5 $\quad$ **if** $F \cup \bigcup_{(x,b)\in\alpha_P} (x = b)$ *is satisfiable* **then**
6 $\quad\quad C \leftarrow$ `CLD` $(F \cup \bigcup_{(x,b)\in\alpha_P} (x = b),\ L^{\neg}(O))$
7 $\quad\quad$ **if** $F \cup (L^{\neg}(O) \setminus C) \cup \{\left(\bigvee_{l\in C} l\right)\}$ *is satisfiable* **then**
8 $\quad\quad\quad C \leftarrow$ `CLD` $(F \cup (L^{\neg}(O) \setminus C),\ C)$
9 $\quad\quad M \leftarrow M \cup \{C\}$
10 $\quad\quad F \leftarrow F \cup \left(\bigvee_{l\in C} l\right)$
11 $\quad\quad$ `PushBack` $(P,\ \alpha_P)$
12 **return** M

cation path queue is

$$\begin{aligned} P = <&\{(x_2, 0), (x_5, 0), (x_3, 0), (x_4, 0)\},\\ &\{(x_2, 1), (x_5, 0), (x_3, 0), (x_4, 0)\},\\ &\{(x_2, 0), (x_5, 1), (x_3, 0), (x_4, 0)\},\\ &\{(x_2, 1), (x_5, 1), (x_3, 0), (x_4, 0)\},\\ &\cdots >. \end{aligned} \tag{7}$$

The pseudo-code of MCSEnumPD, MCS enumeration with path diversification, is presented in algorithm 3. It starts by building the queue of diversification paths (line 2). Then, it removes the first path from the queue (line 4) and checks if an MCS exists for that path (line 5). If so, then an MCS C is generated (line 6), followed by a check to validate if C is an MCS of the original instance (line 7). After obtaining a valid MCS, the algorithm stores it (line 9) and "blocks" it (line 10). The path is then placed at the end of the queue (line 11). Note that paths for which an MCS no longer exists are discarded by the algorithm. This process is repeated until the queue becomes empty (line 3), i.e., no more MCSs exist for any path.

5 Experimental Results

In this section, the performance of the techniques proposed in sections 3 and 4 is evaluated on instances of the Virtual Machine Consolidation (VMC) problem. In VMC, we have several servers with fixed resource capacities and Virtual Machines (VMs) with requirements of those same resources. Each VM must be placed in some server, but server capacities cannot be exceeded and some VMs cannot be placed in the same server. There exists an initial placement, i.e., a VM can be associated with an initial server, incurring a migration cost if the VM is placed in a different one. This cost is estimated to be equal to the memory requirement of the VM. A migration budget constraint can be used to enforce an upper limit on the migration costs, and is specified as a percentile bp of the total memory capacity of the servers. The goal is to find a placement that satisfies the constraints and simultaneously minimizes energy consumption, migration costs and resource wastage. The latter is a measure of the imbalance of server resource usage.

A detailed description of the VMC problem, as well as the MOCO formulation, can be found in the work of Terra-Neves, Lynce, and Man qui nho(2017). Note that the authors use a resource wastage cost function that is an approximation of the one used by Zheng et al.(2016), which is a sum of divisions. The approximation consists in discarding the denominators of the divisions in order to obtain a linear cost function. In this paper, we consider the exact version of the wastage cost function and compare the performance of the methods proposed in section 3 with the linear approximation of the wastage function.

The evaluation is performed on the VMC benchmarks used in the work of Terra-Neves, Lynce, and Man quinho(2017), which are based on subsets of workload traces randomly selected from the Google Cluster Data project[1]. The benchmark set includes instances with 32, 64 and 128 servers. For each benchmark instance, the sum of VM resource requirements can be approximately 25%, 50%, 75% and 90% of the total capacity of the servers. The initial placements comprises approximately 0% (no placement), 25%, 50%, 75% and 100% (all VMs initially placed) of the total VM resource requirements. There are five different benchmarks for each number of servers (32, 64 and 128), total VM resource requirement (25%, 50%, 75% and 90%) and placement requirement percentile combination (0%, 25%, 50%, 75% and 100%), amounting to a total of 300 benchmarks. For each server, the energy consumption parameters $energy_{idle}$ and $energy_{full}$ were chosen from the ranges $[110, 300]$ and $[300, 840]$, respectively, depending on their resource capacities. The benchmark set and the prototype that implements the algorithms evaluated in this paper are publicly available online[2].

The Hypervolume (HV) quality indicator (Zitzler and Thiele 1999) provides a combined measure of convergence and diversity and is used to compare the performance of the algorithms. HV measures the volume of the cost space between the set of non-dominated solutions returned by the algorithm and a given reference point. The reference point depends on the benchmark, and is set to the worst possible costs. Larger values of HV mean that the solution set is composed of solutions of better quality and/or diversity.

The Inverted Generational Distance (IGD) indicator (Zhang and Li 2007) is also used to assess the quality of the solution sets produced by the algorithms. IGD measures the average Euclidean distance, in the cost space, between the Pareto optimal solutions and the solution set returned by the algorithm, and a smaller value is preferred. In this scenario, since the set of Pareto optimal solutions is unknown, we use instead an approximation obtained by combining the solution sets produced by all algorithms evaluated in this section.

All algorithms were implemented in Java and Sat4j-PB (Le Berre and Parrain 2010) (version 2.3.4) was used as the

[1] http://code.google.com/p/googleclusterdata/
[2] http://sat.inesc-id.pt/dome

PBS solver. Each algorithm was executed with a memory limit of 4 GB and a time limit of 1800 seconds. Randomized algorithms were executed with 10 different seeds for each instance, and the analysis is performed using the median values over all executions. The evaluation was conducted on an AMD Opteron 6376 (2.3 GHz) with 128 GB of RAM.

5.1 Division Reduction vs Approximation

In this section, we evaluate how the CLD algorithm performs when handling resource wastage with division reduction, compared to approximating by discarding the denominators. For the purpose of this evaluation, 100 instances with 8 servers from Google's workload traces were generated. Because those instances are smaller, the algorithms are able to produce better approximations of the Pareto front. Therefore, we only show results for those instances, as they better illustrate the impact of division reduction. We also obtained results for the instances with 32, 64 and 128 servers, but despite having positive results, the impact of division reduction was smaller for those instances.

Figures 1 and 2 show the distributions of the HV and IGD values, respectively, obtained by both approaches with a migration budget of 100%. A point (x, y) in the HV (IGD) distribution plot indicates that the given approach obtained an HV (IGD) equal to or greater (lower) than y for x instances. For example, the point $(40, 0.15)$ on division reduction's line in figure 1 indicates that, with division reduction, CLD obtained HVs equal to or greater than 0.15 for 40 instances. Note that points that correspond to IGD values larger than 3 are not displayed in figure 2. This is explained by the fact that, for some harder instances, the corresponding IGD values were too large (sometimes larger than 40), resulting in all lines to overlap for $x < 80$. Therefore, in order to improve the readability of IGD distribution plots, only points up to some fixed value are displayed.

Observe that division reduction allows CLD to find solutions of better quality, both in terms of IGD and HV. Figure 3 shows the distribution of the best wastage costs obtained with both approaches. In general, CLD is able to find solutions with smaller resource wastage costs when using division reduction. Results with budgets of 5%, 1% and 0.5% were also obtained. In terms of IGD, the results were very similar. In terms of HV and best wastage costs, the gains become less significant as the budget decreases. These results are not shown due to space restrictions.

5.2 Comparison of Diversification Techniques

Figures 4 and 5 show the HV and IGD distributions, respectively, for CLD, SampleMCS and MCSEnumPD when using a budget of 100%. The m (n_P) parameter of SampleMCS (MCSEnumPD) was set to 1 (4), as suggested by our empirical evaluation. We can see that SampleMCS degrades IGD without improving HV. On the other hand, MCSEnumPD shows a large improvement in terms of HV. It also improves IGD for some instances, but we can see a negative impact on many of them. We observed that this IGD deterioration occurs mostly on the instances with VM loads of 75% and 90%. Such instances have considerably less feasible solutions, causing MCSEnumPD to spend its time proving that no solution exists for most of the diversification paths, instead of actually finding solutions. This occurs because MCSEnumPD does not consider the problem constraints when generating the diversification paths. Instead, it focuses on variables with larger coefficients in the cost functions.

Figures 6 and 8 show the HV distributions when using budgets of 5% and 1% respectively. Observe that migration budgets are not relevant for the 60 instances with no VMs initially placed. Hence, the results for these instances are not illustrated in these figures. We can see that MCSEnumPD's improvement in HV becomes less significant as the budget decreases. Migration budget constraints reduce the space of feasible solutions, triggering the same behavior in MCSEnumPD for the 75% and 90% load instances. Also, these constraints prune the search space, making it easier for all MCS algorithms to find good solutions. This is supported by the observation that, in general, HV values tend to increase as the budget decreases. Additionally, with a budget of 5%, SampleMCS is able to find feasible solutions for more instances than the original CLD algorithm. The IGD distributions are presented in figures 7 and 9. The impact in terms of IGD also becomes less significant as the budget decreases.

5.3 Comparison with State-of-the-Art

This section compares the proposed enhancements with state-of-the-art VMC algorithms, VMPMBBO (Zheng et al. 2016) and MGGA (Xu and Fortes 2010). We note that VMPMBBO was originally designed for minimizing energy consumption and resource wastage. When no initial placement exists, no migrations occur and VMPMBBO was run with the configuration suggested by Zheng et al.(2016). However, when some VMs are initially placed, we have to consider migration costs. VMPMBBO's population is divided into subsystems and each subsystem minimizes a single cost function. The suggested configuration uses 4 subsystems, 2 per objective. When an initial placement exists, we use 6 subsystems instead to account for migration costs.

MGGA was originally designed for minimizing thermal dissipation instead of migration costs. We adapted MGGA to consider migration costs instead and was configured to use a population size of 12, and crossover rate and mutation rate as suggested by Xu and Fortes(2010).

Figures 4 and 5 show the HV and IGD distributions, respectively, for all algorithms when using a migration budget of 100%. We can see that VMPMBBO has the best performance both in terms of HV and IGD, being able to find solution sets for more instances and of better quality. However, note that MCSEnumPD is able to obtain HV values better than MGGA's in far more instances than the original CLD algorithm. Finally, when migration budgets of 5% and 1% are considered, VMPMBBO's and MGGA's performance degrades considerably, while CLD and its variants remain robust, as we can see in figures 6, 7, 8 and 9. Note that MGGA's performance degrades the most, barely being able to find feasible solutions. In fact, with a budget of 1%, MGGA fails to find a single solution for all instances. Note in figure 6 that MCSEnumPD is able to surpass VMPMBBO in terms of HV for far more instances than CLD, thus improving the constraint-based approach for MOCO.

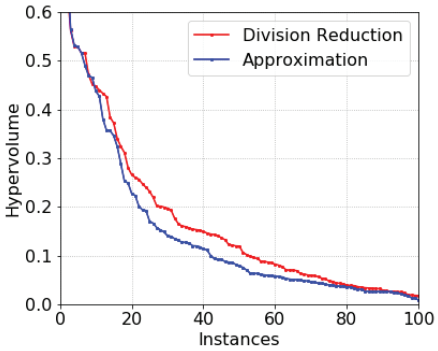

Figure 1: HV distributions for division handling ($bp = 100\%$).

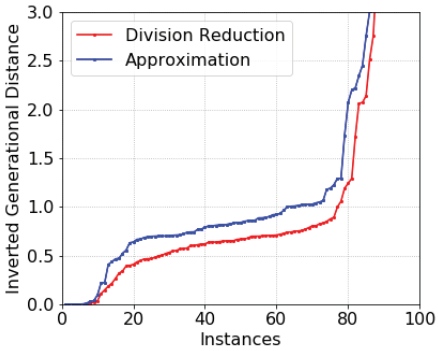

Figure 2: IGD distribution for division handling ($bp = 100\%$).

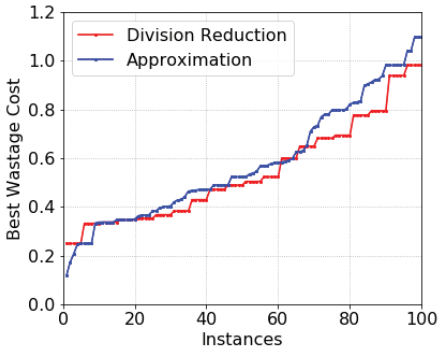

Figure 3: Best wastage distribution for division handling ($bp = 100\%$).

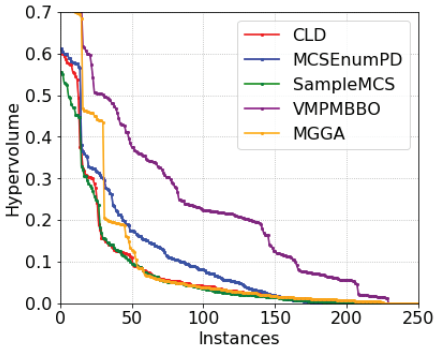

Figure 4: Algorithm HV distributions ($bp = 100\%$).

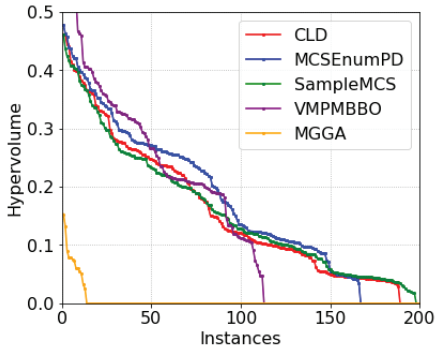

Figure 6: Algorithm HV distributions ($bp = 5\%$).

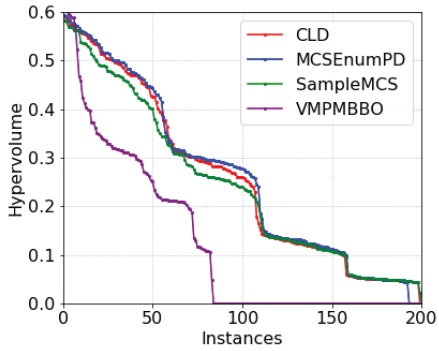

Figure 8: Algorithm HV distributions ($bp = 1\%$).

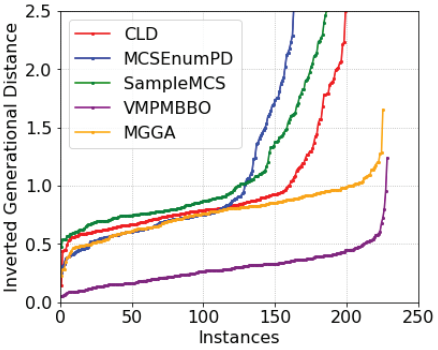

Figure 5: Algorithm IGD distributions ($bp = 100\%$).

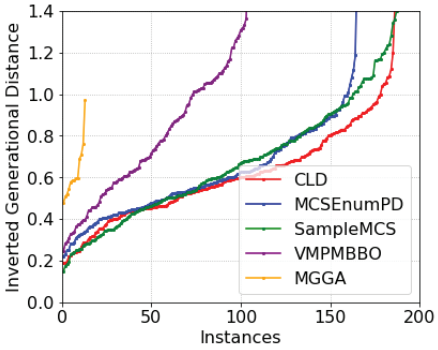

Figure 7: Algorithm IGD distributions ($bp = 5\%$).

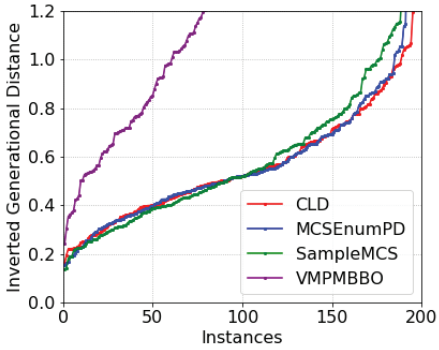

Figure 9: Algorithm IGD distributions ($bp = 1\%$).

6 Conclusions and Future Work

The usage of constraint-based methods for solving MOCO formulations is still in its early stages. This paper further enhances the state of the art in several ways. In particular, this work goes beyond linear expressions in cost functions in PBO and MOCO formulations, and defines a new scheme to deal with divisions of linear expressions. Furthermore, in order to provide better approximations of the Pareto front, two new diversification procedures are proposed, namely sampling of MCSs and path diversification. Experimental results in a large set of MOCO instances from the VMC problem show the improvements due to the proposed techniques.

The usage of constraint-based methods for MOCO can be expanded in several ways. First, other non-linear cost functions should be considered in constraint-based methods. Additionally, the current diversification scheme based in sampling is computationally heavy and can be improved with new procedures for MCS sampling (Achlioptas and Theodoropoulos 2017). Moreover, the proposed path diversification strategy focus mainly on the coefficients in the cost

functions, but other procedures for guiding path generation can also be used (Heule et al. 2012).

Acknowledgments

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with references UID/CEC/50021/2013 and SFRH/BD/111471/2015.

References

Achlioptas, D., and Theodoropoulos, P. 2017. Probabilistic model counting with short xors. In *Proceedings of the Twentieth International Conference Theory and Applications of Satisfiability Testing*, 3–19.

Bacchus, F.; Davies, J.; Tsimpoukelli, M.; and Katsirelos, G. 2014. Relaxation Search: A Simple Way of Managing Optional Clauses. In *Proceedings of the Twenty-Eighth Conference on Artificial Intelligence*, 835–841. AAAI Press.

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

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

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

Carter, L., and Wegman, M. N. 1977. Universal Classes of Hash Functions (Extended Abstract). In *Proceedings of the Ninth Annual Symposium on Theory of Computing*, 106–112. ACM.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2013. A Scalable and Nearly Uniform Generator of SAT Witnesses. In *Proceedings of the Twenty-Fifth International Conference on Computer Aided Verification*, 608–623. Springer.

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

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

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2006. Near-Uniform Sampling of Combinatorial Spaces Using XOR Constraints. In *Proceedings of the Twentieth Annual Conference on Advances in Neural Information Processing Systems*, 481–488.

Grégoire, É.; Lagniez, J.; and Mazure, B. 2014. An Experimentally Efficient Method for (MSS, CoMSS) Partitioning. In *Proceedings of the Twenty-Eighth Conference on Artificial Intelligence*, 2666–2673. AAAI Press.

Heule, M. J.; Kullmann, O.; Wieringa, S.; and Biere, A. 2012. Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads. In *Hardware and Software: Verification and Testing*. Springer. 50–65.

Iturriaga, S.; Dorronsoro, B.; and Nesmachnow, S. 2017. Multiobjective Evolutionary Algorithms for Energy and Service Level Scheduling in a Federation of Distributed Datacenters. *International Transactions in Operational Research* 24(1-2):199–228.

Jackson, D.; Estler, H.; Rayside, D.; et al. 2009. The Guided Improvement Algorithm for Exact, General-Purpose, Many-Objective Combinatorial Optimization. Technical Report MIT-CSAIL-TR-2009-033, MIT.

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

Marques-Silva, J.; Heras, F.; Janota, M.; Previti, A.; and Belov, A. 2013. On Computing Minimal Correction Subsets. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, 615–622. AAAI Press.

Mencía, C.; Previti, A.; and Marques-Silva, J. 2015. Literal-Based MCS Extraction. In *Proceedings of the Twenty-Forth International Joint Conference on Artificial Intelligence*, 1973–1979. AAAI Press.

Nadel, A. 2011. Generating diverse solutions in SAT. In *Proceedings of the International Conference in Theory and Applications of Satisfiability Testing*, 287–301.

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

Terra-Neves, M.; Lynce, I.; and Manquinho, V. 2017. Introducing Pareto Minimal Correction Subsets. In *Proceedings of the Twentieth International Conference Theory and Applications of Satisfiability Testing*, 195–211. Springer.

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

Xu, J., and Fortes, J. A. B. 2010. Multi-Objective Virtual Machine Placement in Virtualized Data Center Environments. In *Proceedings of the International Conference on Green Computing and Communications, & International Conference on Cyber, Physical and Social Computing*, 179–188. IEEE.

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

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

Zitzler, E., and Thiele, L. 1999. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation* 3(4):257–271.