TÉCNICO LISBOA

Stochastic Multi-objective Combinatorial Optimization Algorithms

Miguel António dos Santos Machado Tavares

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisors: Professor Vasco Miguel Gomes Nunes Manquinho
Doctor Miguel Ângelo da Terra Neves

Examination Committee

Chairperson: Professor José Luís Brinquete Borbinha
Supervisor: Professor Vasco Miguel Gomes Nunes Manquinho
Member of the Committee: Professor Alexandre Paulo Lourenço Francisco

January 2021

Acknowledgments

I would like to start by thanking my advisors Professor Vasco Manquinho and Miguel Neves for all their help and guidance throughout this master thesis.

I would like to thank my parents for everything they have done for me, for believing and supporting me all these years. You shaped me into who I am and I could not be more grateful.

To my girlfriend and friends, thank you for helping me. Could not have got through all these years without you.

This work was partially supported by national funds through FCT with references UID/CEC/50021/2020, PTDC/CCI-COM/31198/2017 and DSAIPA/AI/0044/2018.

Resumo

Otimização Combinatória Multi-Objetivo (MOCO) tem diversas aplicações no mundo real como Consolidação de Máquinas Virtuais, Alocação do Nível de Garantia de Desenvolvimento, Problema de Cobertura do Conjunto, entre outros. Novas abordagens para MOCO foram propostas baseadas em iterativamente resolver formulações de Lógica Proposicional. Além disso, esta abordagem demonstrou que consegue resolver efetivamente instâncias mais restritas. Por outro lado, abordagens estocásticas são normalmente melhores em instâncias menos restritas. Recentemente, abordagens híbridas que combinam técnicas estocásticas com técnicas de Lógica Proposicional, na forma de operadores inteligentes, têm sido propostas. Estes algoritmos demonstraram resolver muitos problemas de forma eficaz, como a Consolidação de Máquinas Virtuais, que utiliza variáveis inteiras na formulação do problema.

Neste trabalho apresentamos Neon, uma abordagem híbrida genérica para resolver instâncias MOCO compostas por fórmulas Booleanas Multi-Objetivo, que utilizam variáveis Booleanas. Além disso, Neon incorpora dois operadores inteligentes. O primeiro é o de mutação inteligente, que transforma uma atribuição inviável numa viável utilizando um solucionador Pseudo-Booleano. O outro chama-se melhoria inteligente, e melhora uma atribuição já viável ao extrair Subconjuntos de Correção Mínimos do problema. Utilizar variáveis Booleanas na formulação do problema é normalmente mais díficil para o algoritmo estocástico do que usar variáveis inteiras, portanto, o Neon implementa uma técnica chamada melhoria da estrutura que explora a estrutura de algumas restrições, ao criar variáveis inteiras que representam as referidas restrições e obrigam-nas a que sejam sempre satisfeitas.

Palavras-chave: Otimização Combinatória Multi-Objetivo, Pseudo-Booleano, Algoritmo estocástico, Algoritmo baseado em restrições, Algoritmo híbrido, Mutação inteligente, Melhoria inteligente, Melhorias da estrutura

Abstract

Multi-Objective Combinatorial Optimization (MOCO) has several real-world applications such as Virtual Machine Consolidation, Development Assurance Level Allocation, Set Covering Problem, among others. New approaches for MOCO have been proposed based on iteratively solving Propositional Logic formulations. Moreover, this approach has been shown to effectively solve more constrained problem instances. On the other hand, stochastic approaches are usually better on less constrained instances. Recently, hybrid approaches which combine stochastic techniques with Propositional Logic techniques, in the form of smart operators, have been proposed. These algorithms have shown to effectively solve many problems, such as Virtual Machine Consolidation that uses integer variables in the formulation of the problem. Therefore, one should be able to create a hybrid algorithm that can solve generic MOCO instances.

In this work, we present Neon, a generic hybrid approach for solving MOCO instances formulated as Multi-Objective Boolean formulas, which uses Boolean variables. Moreover, Neon incorporates two smart operators. The first one is smart mutation, which transforms an unfeasible assignment into a feasible one using a Pseudo-Boolean solver. The other one is called smart improvement, and improves an already feasible assignment by extracting Minimal Correction Subsets from the problem. Using Boolean variables in the formulation of the problem is usually more difficult for the stochastic algorithm than using integer variables, therefore, Neon implements a technique called structure improvements which exploits the structure of some constraints by creating integer variables that represent the said constraints and force them to always be satisfied.

Keywords: Multi-Objective Combinatorial Optimization, Pseudo-Boolean, Stochastic algorithm, Constraint based algorithm, Hybrid algorithm, Smart mutation, Smart improvement, Structure improvements

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Many real-world problems can be formulated as multi-objective problems [9, 30, 11]. While in single-objective problems there is only one optimal solution (known as a Pareto-optimal solution) to the problem, in the multi-objective problem, the same usually does not happen. Normally, the cost functions contradict themselves, meaning that there is no point that can minimize all the cost functions simultaneously. If there is no other information given, we cannot say that a Pareto-optimal solution is better than the others, which demands the user to find the set of all Pareto-optimal solutions (known as Pareto-front) or at least a good approximation of this set, since the number of Pareto-optimal solutions can grow exponentially with the number of cost functions.

Many algorithms have been proposed in order to solve these multi-objective problems. These algorithms can be divided into two main categories: stochastic and constraint-based algorithms. Stochastic algorithms are better in finding a good approximation of Pareto-front when problems are not highly constrained. These algorithms work by applying genetic operators on existing assignments from a population, in order to create new assignments, which then can be selected for a new population. Examples of these types of algorithms can be the NSGA-II [6] and the MOEA/D [32] algorithms. On the other hand, constraint-based algorithms can find the exact Pareto-front, however, it is very time consuming. These algorithms iteratively use solvers based on Propositional Logic to find new feasible assignments. Examples of these algorithms are the GIA [13] and the MCS enumeration [24] algorithms.

It has been shown that both stochastic and constraint based algorithms fail to provide solutions on some real-world problems [21, 19]. For this reason, a recent category called hybrid algorithms, has been gaining a lot of interest. Hybrid algorithms use techniques from both stochastic and constraint-based algorithms. An example of a hybrid algorithm is the SATIBEA [12]. However, this algorithm is specific to a single real-world problem and to our knowledge, no generic hybrid algorithm exists. For this reason, our goal is to create a tool that uses hybrid algorithms to solve generic multi-objective instances.

Problems such as the Virtual Machine Problem can be formulated using integer variables, which makes it easier for the stochastic algorithm to satisfy the formula. However, the generic instances that

our tool will solve, are formulated using only Boolean variables .

Take the following example where 1 virtual machine v_1 has to be placed in 1 out of the 3 servers s_1, s_2 and s_3. In the integer formulation we can create the variable $v_1 \in \{1, 2, 3\}$, where having assigned the value x means that the virtual machine v_1 is placed in the server s_x. Using a Boolean formulation we have the following:

$$\begin{aligned} x_1 + x_2 + x_3 &= 1 \\ x_1, x_2, x_3 &\in \{0, 1\} \end{aligned} \tag{1.1}$$

For this case, variables x_1, x_2 and x_3 represent v_1 being placed in s_1, s_2 and s_3, respectively. There are 2^3 possible assignments for this constraint, however, only 3 satisfy it. For this reason, a technique called structure improvements is created, where the structure of these constraints is exploited by replacing them with integer variables, in order to have a similar structure to the integer formulation.

1.2 Contributions

In this work, we introduce Neon, a tool used to solve generic Multi-Objective Combinatorial Optimization instances. Neon is built on top of sat4j-moco, a tool that uses a constraint based algorithm to find feasible assignments on generic Multi-Objective Combinatorial Optimization instances. The main contributions of this thesis are as follows:

- We start by extending sat4j-moco, by adding the stochastic algorithms NSGA-II and MOEA/D.
- We implement two smart operators, which are operators that use constraints based techniques, and apply them to NSGA-II and MOEA/D, thus creating hybrid algorithms.
- We create the structure improvements technique, which exploits exactly-1 and at-most-1 constraints, forcing them to always be satisfied by the stochastic approach of the algorithms.
- We perform an analysis on different types of real-world problems formulated as generic Multi-Objective Combinatorial Optimization instances, in order to evaluate the proposed algorithms.

1.3 Organization of the Document

This document is structured as follows. First, some general concepts related to Multi-Objective Combinatorial Optimization, Minimal Correction Subsets and performance metrics used to evaluate our results, are presented in Chapter 2. Next, algorithms used to solve Multi-Objective Combinatorial Optimization problems are presented in Chapter 3, separated into three different categories: stochastic algorithms, constraint based algorithms and hybrid algorithms. that implement techniques from both the stochastic and constraint based algorithms in order to find new feasible individuals. In Chapter 4, a hybrid algorithm

used to solve generic MOCO instances, is proposed, along with a technique called structure improvements, that exploits the structure of some constraints in order to facilitate the formulation. The evaluation of the algorithms for instances of Set Covering Problem, Development Assurance Level Allocation, Flying Tourist Problem and Virtual Machine Consolidation, is presented in Chapter 5. Finally, we conclude this work and present the future work in Chapter 6.

Chapter 2

Preliminaries

In this section, we start by defining several terms and notations that will be used in the rest of the document. First, Conjunctive Normal Form is introduced in Section 2.1. Next, Pseudo-Boolean Optimization (PBO) is introduced in Section 2.2. Then, Minimal Correction Subsets (MCS) are defined in Section 2.3. Multi-Objective Combinatorial Optimization (MOCO) and Pareto-optimal solutions are introduced in Section 2.4. Finally, some performance metrics are described in Section 2.5.

2.1 Boolean Satisfiability

Let $X = \{x_1, \ldots, x_n\}$ be a set of n Boolean variables. A literal can be positive or nagative. A positive literal is a variable x_i while a negative literal is its complement $\neg x_i$. Literals are aggregated using the binary disjunction operator $\vee$ to form a clause C. Each clause C_i has the form:

$$C_i = \bigvee_{j \in P_i} x_j \vee \bigvee_{j \in N_i} \neg x_j \tag{2.1}$$

where $P_i \subseteq \{1, \ldots, n\}$ and $N_i \subseteq \{1, \ldots, n\}$, $P_i \cap N_i = \emptyset$, are sets of variable indices that correspond to the positive and negative literals, respectively. Clauses are aggregated using the binary conjunction operator $\wedge$ to form a formula, that has the form:

$$\psi = C_1 \wedge \ldots \wedge C_n \tag{2.2}$$

A variable assignment on a Boolean variable consists of assigning it the value 0 or 1. We say that α is a complete assignment if it assigns a value to all the variables $x_i \in X$. The goal of the SAT problem is to find an assignment that satisfies a given formula. A model is an assignment that satisfies all the clauses in the CNF formula.

Example 1. *Consider a SAT instance defined by* $\psi = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$*. Consider the assignment* $\alpha_1 = \{(x_1, 1), (x_2, 1), (x_3, 1)\}$*.* α_1 *does not satisfy* ψ*, since the constraint* $(\neg x_2 \vee \neg x_3)$ *is not satisfied. Consider another assignment* $\alpha_2 = \{(x_1, 1), (x_2, 0), (x_3, 1)\}$*.* α_2 *satisfies all the constraints in* ψ*, making* α_2 *a model of* ψ*.*

2.2 Pseudo-Boolean Optimization

Let $X = \{x_1, \ldots, x_n\}$ be a set of n Boolean variables. Given a set of p literals $l_1, \ldots, l_p$, their corresponding weights $w_1, \ldots, w_p$ and a positive integer $k \in \mathbb{N}$ we have that a Pseudo-Boolean (PB) constraint has the form:

$$\sum_{i=1}^{p} w_i \cdot l_i \bowtie k, \ \bowtie \in \{\leq, =, \geq\} \tag{2.3}$$

Given a set $F = \{c_1, \ldots, c_m\}$ of m PB constraints, the Pseudo-Boolean Satisfiability (PBS) problem consists on finding a complete assignment $\alpha : X \longrightarrow \{0, 1\}$, such that all the PB constraints are satisfied. Given a Pb constraint c, $\alpha(c) = 1$ denotes that c is satisfied by the assignment. We say that α is a model of F if it satisfies all its constraints $c_i \in F$, denoted by $\alpha(F) = 1$. Otherwise we say that α is unfeasible, denoted by $\alpha(F) = 0$. If $\forall \alpha, \alpha(F) = 0$, then F is unsatisfiable.

In the Pseudo-Boolean Optimization (PBO) problem, besides the set of constraints F, a PB expression f known as cost function that assigns a cost to the models of F is also present. In PBO, the goal is to find an assignment that satisfies all the constraints while minimizing the value of the cost function. The cost function is denoted by $f(X)$ and has the form:

$$\sum_{i=1}^{p} w_i \cdot l_i \tag{2.4}$$

For a particular assignment α, $f(\alpha)$ denotes the value of $f(X)$ for the assignment α.

Example 2. *Consider a PBO instance defined by a set of variables $X = \{x_1, x_2, x_3\}$, a set of constraints $F = \{(x_1 + x_2 \leq 1), (4x_1 + 2x_2 + 2x_3 \geq 4)\}$ and a cost function $f(X) = 3x_1 + x_2 + x_3$. Consider the assignment $\alpha_1 = \{(x_1, 1), (x_2, 1), (x_3, 1)\}$. We can see that $\alpha_1(F) = 0$, i.e. α_1 is unfeasible, because it does not satisfy the first constraint. On the other hand, $\alpha_2 = \{(x_1, 1), (x_2, 0), (x_3, 0)\}$ satisfies all the constraints, thus α_2 is a model of F and has a cost of $f(\alpha_2) = 3$. Additionally, consider $\alpha_3 = ((x_1, 0), (x_2, 1), (x_3, 1))$ which is also a model of F and has a cost of 2. α_3 is an optimal assignment for this problem since all the other assignments either do not satisfy F or have a cost higher than 2.*

2.3 Minimal Correction Subsets

Let F be an unsatisfiable set of PB constraints. A subset $C \subseteq F$ is said to be a Minimal Correction Subset (MCS) of F if and only if $F \setminus C$ is satisfiable and $\forall c \in C, F \setminus C \cup \{c\}$ is unsatisfiable.

Example 3. *Let F be a set of PB constraints over variables $\{x_1, x_2, x_3\}$ and defined by $\{(x_1 + x_2 = 1), (x_1 = 1), (x_1 = 0), (x_2 = 1), (x_3 = 1)\}$. This set is unsatisfiable, having 3 MCSs: $C_1 = \{(x_1 + x_2 = 1), (x_1 = 0)\}$, $C_2 = \{(x_1 = 1)\}$ and $C_3 = \{(x_1 = 0), (x_2 = 1)\}$.*

To find the MCSs of a PB formula we will use the state-of-art CLD algorithm [18]. The pseudo-code for this algorithm is presented in Algorithm 2.1. The algorithm starts by initializing the sets S and C, the sets of satisfied and unsatisfied constraints respectively (lines 1 - 2). Then it tries to satisfy at least

Algorithm 2.1: CLD Algorithm

Input: F

Output: C

1 $S \leftarrow \emptyset$
2 $C \leftarrow F$
3 $D \leftarrow (\bigvee_{c \in C} c)$
4 $\alpha \leftarrow \texttt{PBSolver}(S \cup D)$
5 **while** $\alpha \neq \emptyset$ **do**
6 $\quad S \leftarrow S \cup \bigcup_{c \in C, \alpha(c)=1} \{c\}$
7 $\quad C \leftarrow F \setminus S$
8 $\quad D \leftarrow (\bigvee_{c \in C} c)$
9 $\quad \alpha \leftarrow \texttt{PBSolver}(S \cup \{D\})$
10 **return** C

one of the constraints in C, that initially contains all of the constraints (lines 3 - 4). Next, the algorithm repeatedly updates S and C by removing the constraints from C that are satisfied and adding them to S (lines 6 - 7) and keeps checking if it is possible to satisfy all the constraints in S and at least one in C (lines 8 - 9). When this becomes impossible the cycle ends and C is the returned MCS (line 10).

A weighted PB constraint is a pair (c, ω) where c is a PB constraint and $\omega \in \mathbb{N}^+$ is the cost of not satisfying the constraint c. The Weighted Boolean Optimization (WBO) problem is defined by $W = (F_H, F_S)$, where F_H is a set of hard PB constraints, and F_S is a set of soft weighted PB constraints. The goal of WBO is to find a complete assignment that satisfies all the constraints in F_H and minimizes the sum of the weights of the constraints in F_S that are not satisfied.

The notion of MCS can be extended to WBO instances. Let $W = (F_H, F_S)$ be an WBO instance, where F_H denotes the set of hard constraints and F_S denotes the set of soft constraints. $C \subseteq F_S$ is a MCS of W if $F_H \cup (F_S \setminus C)$ is satisfiable and $F_H \cup (F_S \setminus C) \cup \{c\}$ is unsatisfiable for all $c \in C$. Algorithm 2.1 can be used to find MCSs in an WBO problem if S is initialized with F_H and C is initialized with F_S.

Finding an optimal solution of a PBO instance can be reduced to finding an MCS that minimizes its cost [4]. Consider a PBO instance with constraint set F and cost function f. Let $L(f)$ be the set of all literals in f and $L^{\neg}(f)$ the set of the negation of all literals in $L(f)$. Assuming that the cost function has the form $f = \omega_1 \cdot l_1 + \ldots + \omega_o \cdot l_o$, we can reduce the PBO instance to a WBO instance by setting $F_H = F$ and $F_S = \{(\neg l_1, \omega_1), \ldots, (\neg l_o, \omega_o)\}$. Then we apply an MCS algorithm to obtain an MCS of (F_H, F_S). The cost of that MCS C is calculated with the following formula:

$$f(C) = \sum_{(\neg l_i) \in C} w_i \tag{2.5}$$

A complete assignment α that satisfies all the constraints except the ones in the MCS provides an approximation to the optimum solution of PBO. Therefore the PBO problem can be reduced to finding an MCS $C \subseteq L^{\neg}(f)$ that minimizes $f(C)$.

Example 4. *Let* $F = \{(x_1 + x_2 \geq 1), (2x_1 + x_2 + x_3 \leq 2)\}$ *and* $f(X) = 3x_1 + x_2 + \neg x_3$. *An equivalent WBO instance would have* $F_H = \{(x_1 + x_2 \geq 1), (2x_1 + x_2 + x_3 \leq 2)\}$ *and* $F_S = \{(\neg x_1, 3), (\neg x_2, 1), (x_3, 1)\}$. F_S

Table 2.1: Satisfiable assignments and their costs for the instance in Example 5

x_1	x_2	x_3	f_1	f_2
1	1	1	3	3
0	**1**	**1**	**1**	**3**
1	**0**	**1**	**2**	**2**
1	**1**	**0**	**3**	**1**

has two MCSs: $C_1 = \{(\neg x_1), (x_3)\}$*, where* $f(C_1) = 4$*, and* $C_2 = \{(\neg x_2)\}$*, where* $f(C_2) = 1$*. As we can confirm,* C_2 *is the minimum cost MCS. Let* $\alpha = \{(x_1, 0), (x_2, 1), (x_3, 1)\}$ *be an assignment that satisfies all constraints except the ones in* C_2*.* α *is the optimal assignment for this problem and has cost* 1*.*

2.4 Multi-Objective Combinatorial Optimization

Let $X = \{x_1, \ldots, x_n\}$ be a set of n Boolean variables, $F = \{c_1, \ldots, c_m\}$ a set of m constraints and $O = \{f_1, \ldots, f_k\}$ a set of k cost functions that we want to minimize. A Multi-Objective Combinatorial Optimization (MOCO) problem is defined as $M = (F, O)$. In our case, we will consider that the set F is composed of PB constraints and that O is composed of PB expressions. Note that this special case of MOCO is very similar to the PBO problem defined in section 2.2, the main difference being that MOCO has multiple, possibly conflicting, cost functions whilst a PBO problem only has one.

Let $M = (F, O)$ be a MOCO instance and $\alpha, \alpha' : X \longrightarrow \{0, 1\}$ be two distinct models of F. For an assignment α, $O(\alpha)$ denotes the values of the cost functions in O for the assignment α. We say that α dominates α', denoted by $\alpha \prec \alpha'$, if $\forall_{f \in O} f(\alpha) \leq f(\alpha')$ and $\exists_{f \in O} f(\alpha) < f(\alpha')$. We can now define a Pareto-optimal solution as an assignment α such that $\alpha(F) = 1$ and there is no other complete assignment α' such that $\alpha'(F) = 1$ and $\alpha' \prec \alpha$. In this document, we will denote a Pareto-optimal solution by $\tilde{\alpha}$. The set of all Pareto-optimal solutions of a MOCO problem is called Pareto-front and it will be represented by $\tilde{A}$. Given a MOCO instance, the goal is to find its Pareto-front. However, since enumerating the complete Pareto-front cannot be done within a reasonable amount of time for most real-world cases, typically we are interested in finding just a good approximation.

Example 5. *Let* $M = (F, O)$ *be a MOCO instance where* $F = \{(x_1 + x_2 + x_3 \geq 2)\}$ *is the set of PB constraints and* $O = \{f_1, f_2\}$ *the set of cost functions that we want to minimize, where* $f_1 = (2x_1 + x_2)$ *and* $f_2 = (x_2 + 2x_3)$*. Table 1 shows all the satisfying assignments and the Pareto-optimal assignments are highlighted in bold. Note that three out of the four satisfiable assignments are Pareto-optimal. The first solution* $\alpha_1 = ((x_1, 1), (x_2, 1), (x_3, 1))$ *is not considered a Pareto-optimal solution because at least one other solution dominates this one, whereas the other solutions are non-dominated. So in this case,* $\tilde{A} = \{\tilde{\alpha_2}, \tilde{\alpha_3}, \tilde{\alpha_4}\}$*, where* $\tilde{\alpha_2} = \{(x_1, 0), (x_2, 1), (x_3, 1)\}, \tilde{\alpha_3} = \{(x_1, 1), (x_2, 0), (x_3, 1)\}$ *and* $\tilde{\alpha_4} = \{(x_1, 1),$-$(x_2, 1), (x_3, 0)\}$*. In fact, we can see that* α_1 *is dominated for example by* $\tilde{\alpha_2}$ *since* $O(\alpha_1) = (3, 3)$ *and* $O(\tilde{\alpha_2}) = (1, 3)$*.*

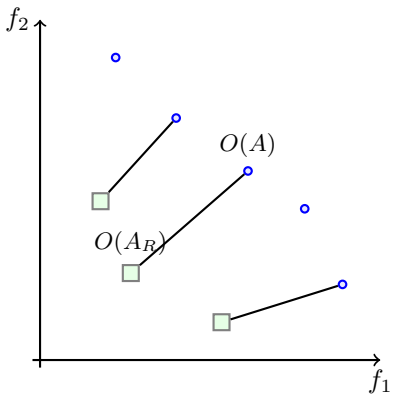


Figure 2.1: Example of the IGD of A

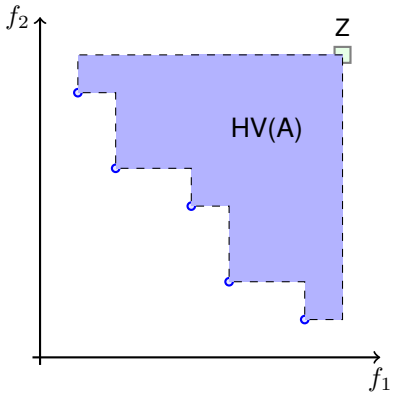


Figure 2.2: Example of the hypervolume of A

2.5 Performance Metrics

Finding which solution gives the best performance in a single-objective problem is simple: the solution with the smallest cost is the best one. However, in the multi-objective case, it is harder to find the Pareto-front within a time limit, and sometimes the algorithms can only produce an approximation of the Pareto-front. Let A_1 and A_2 be two approximations produced by different algorithms for the same MOCO instance. A_1 is said to dominate A_2 if all solutions in A_2 are dominated by some solution in A_1. However, usually, this does not happen. Most likely, both sets contain solutions that are not dominated by some solution in the other set. For this reason, several metrics have been proposed to compare the performance of multi-objective algorithms. There are three types of metrics:

1. Convergence-based metrics: measure how close the approximation is to the Pareto-front.
2. Diversity-based metrics: measure how well distributed the approximation is.
3. Combined metrics: provide a combined measure of convergence and diversity in a single value.

Let $M = (F, O)$ be a MOCO instance where A is a Pareto-front approximation. We define the set $O(A)$ as a set of points, where each point i is defined by the values that $\alpha_i \in A$ takes for each cost function $f \in O$. A_R is a reference set used to calculate the Inverted Generational Distance metric of a set A and ideally it should be the Pareto front if it is known.

The Inverted Generational Distance (IGD) [5] is a combined metric that measures the average distance from the cost vectors in a reference front $O(A_R)$ to the closest cost vectors in $O(A)$. IGD is illustrated in Figure 2.1. The IGD value of a set A can be obtained using the following equation:

$$IGD(A) = \frac{\sqrt[n]{\sum_{\alpha \in A_R} \min_{\alpha' \in A} \{\sqrt{\sum_{i=1}^{n} (f_i(\alpha) - f_i(\alpha'))^2}\}}}{|A_R|} \tag{2.6}$$

The hypervolume [33] is a combined metric that measures the volume of the cost space dominated by a set of solutions A up to a given reference point $Z = (z_1, \ldots, z_k)$. This reference point should be the worst possible cost vector. It can be approximated by setting each z_i to the highest possible value of f_i. An example with two objectives can be seen in Figure 2.2. Let C_α be the hypercube with corners $\alpha \in A$ and Z. The hypervolume value can be obtained using the following equation:

$$HV(A) = \texttt{Volume}(\bigcup_{\alpha \in A} C_\alpha) \tag{2.7}$$

Chapter 3

Related Work

In this section, various algorithms for MOCO are presented. First, stochastic algorithms are presented in section 3.1. Next, constraint based algorithms are presented in section 3.2. Finally, hybrid algorithms are presented in section 3.3.

3.1 Stochastic Algorithms

Multi-Objective Evolutionary Algorithms (MOEAs) start by generating a random population of assignments/solutions A, referred to as individuals. In each iteration, referred to as a generation, an offspring population B of size $|A|$ is created by applying mutation and crossover to the individuals of A. The standard mutation operator iterates over the variables of the individuals and flips them with a given probability. The standard crossover operator combines two individuals, by randomly replacing the assignment of a variable from one individual with the assignment from the other individual, creating two new individuals. Then, $|A|$ individuals among these two populations are selected to be in a new population for the next generation. This last step is where most MOEAs differ.

In this section, we will describe some of these MOEAs. First, the Nondominated Sorting Genetic Algorithm II (NSGA-II) is described in section 3.1.1. Then the Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D) is introduced in section 3.1.2. Finally, the Indicator-Based Evolutionary Algorithm (IBEA) is presented in section 3.1.3.

3.1.1 NSGA-II Algorithm

The NSGA-II algorithm [6] is an MOEA that distinguishes itself by using nondominated sorting to rank the individuals, using that rank to select the individuals that will be used on the next iteration of the algorithm. The pseudo-code for NSGA-II is presented in Algorithm 3.1. First we create a random population A with size N (line 1). Then, the main cycle begins. An offspring population A' is created using crossover and mutation operators (line 3). R is created from the union of A and A' (line 4), having size $2N$. Next, we sort the individuals from R into p sets $FN_1, \ldots, FN_p$ using a nondominated sorting approach (line 5). Each individual in FN_i is dominated by at least one individual from the previous set FN_{i-1} and cannot

Algorithm 3.1: NSGA-II

Input: $(F, O, stoppingcriteria, N)$
Output: A

```
1  A ← RandomPopulation(F, O, N)
2  while ¬stoppingcriteria do
3      A' ← OffspringPopulation(A)
4      R ← A ∪ A'
5      FN_1, ..., FN_p ← FastNondominatedSort(R)
6      A ← ∅
7      i ← 1
8      while |A| + |FN_i| ≤ N do
9          A ← A ∪ FN_i
10         i ← i + 1
11     FN_i ← SortByCrowdingDistance(FN_i)
12     A ← A ∪ FN_i[1 : (N − |A|)]
13 return A
```

be dominated by any of the individuals from its set FN_i to FN_p. Then A is emptied (line 6). Next, while the size of A does not exceed N, we keep adding the elements of the FN_i sets to A in order (lines 8 - 10). If $|A| < N$ and $|A| + |FN_i| > N$, then only some individuals from FN_i must be added to A. Since not all of the individuals can be added, we use the crowding distances, to measure how far each individual is from its neighbors. To promote diversity between the individuals, the ones with higher crowding distances have higher priority. Given this, each individual in FN_i is assigned the value of its crowding distance and then FN_i is sorted according to these (line 11). The first $N - |A|$ individuals of the sorted FN_i are added to A (line 12). This continues until the stopping criteria (line 2) is triggered. This condition is usually a time limit or a maximum number of iterations for the algorithm.

The nondominated sorting approach used in NSGA-II is what distinguishes it from the previous version, NSGA [23]. This approach is presented in Algorithm 3.2. First we calculate the set of solutions dominated by each individual $\alpha \in A$, S_α (lines 5 - 6), and the number of solutions that dominate α, n_α (lines 8 - 9). To find the solutions belonging to FN_1, for each α we simply need to check if $n_\alpha = 0$ (lines 10 - 12). Then, for each $\alpha \in FN_i$ where i starts at 1, $n_{\alpha'}$ is decremented if α' belongs to S_α (lines 17 - 18). If for any α' the dominated count reaches zero, we add it to the list FN_{i+1} (lines 19 - 21). This process is repeated until all solutions are assigned to a front. We save the rank (α_{rank}) of each solution, as it will be used later in the crowded-comparison operator. This operator gives priority to the individuals that are in less clustered areas, in order to improve diversity.

3.1.2 MOEA/D Algorithm

MOEA/D [32] decomposes the MOCO instance into N single-objective optimization sub-problems and solves them by evolving a population of solutions. In each generation, the population is composed of the best solutions found for each sub-problem. There are several scalarizations that could be used in this algorithm to decompose the multi-objective problem. In this work, we consider the Tchebycheff approach.

Algorithm 3.2: Fast Nondominated Sort

Input: (A)
Output: $(FN_1, \ldots, FN_p)$

```
1  foreach α ∈ A do
2      S_α ← ∅
3      n_α ← 0
4      foreach α' ∈ A \ {α} do
5          if α ≺ α' then
6              S_α ← S_α ∪ {α'}
7          else
8              if α' ≺ α then
9                  n_α ← n_α + 1
10     if n_α = 0 then
11         α_rank = 1
12         FN_1 ← FN_1 ∪ {α}
13 i ← 1
14 while FN_i ≠ ∅ do
15     FN_{i+1} ← ∅
16     foreach α ∈ FN_i do
17         foreach α' ∈ S_α do
18             n_α' ← n_α' − 1
19             if n'_α = 0 then
20                 β_rank ← i + 1
21                 FN_{i+1} ← FN_{i+1} ∪ {α'}
22     i ← i + 1
23 return FN_1, ..., FN_p
```

Let $\lambda = (\lambda_1, \ldots, \lambda_k)^T$ be a weight vector, i.e., $\forall i \in [1, \ldots, k], \lambda_i \geq 0$ and $\sum_{i=1}^{k} \lambda_i = 1$. Let $M = (F, O)$ be a MOCO instance and $z^* = (z_1^*, \ldots, z_k^*)^T$ be the reference point, where $\forall i \in [1, \ldots, k], z_i^* = min\{f_i(\alpha) | \alpha(F) = 1\}$. The respective Tchebycheff scalarization for M has the form:

$$\begin{aligned} \text{minimize } g(\alpha|\lambda, z^*) = max_{1 \leq i \leq N}\{\lambda_i \cdot |f_i(\alpha) - z_i^*|\} \\ \text{subject to } \alpha(F) = 1 \end{aligned} \tag{3.1}$$

For each Pareto optimal point x^* there exists a weight vector λ such that x^* is the optimal solution of (3.1). So it is possible to find different Pareto optimal solutions by altering the weight vector.

The pseudo-code for MOEA/D is shown in Algorithm 3.3. It starts by creating an initial random population $A = (\alpha_1, \ldots, \alpha_N)$ (line 2), an initial reference point $z = (z_1, \ldots, z_k)$ that will be used in the Tchebycheff approach (line 3) and N well distributed weight vectors (line 4). Then it computes the Euclidean distances between each weight vector λ^i with every other λ^j where $j \in \{1, \ldots, N\} \setminus \{i\}$, saving the index j of the T closest weight vectors to λ^i in B_i (line 6). Next, two random indexes l and m are chosen from each set B_i (line 9). The members of the population with the indexes l and m then go through mutation and crossover operators in order to get a new individual α (line 10). α then goes

Algorithm 3.3: MOEA/D Algorithm

Input: $(F, O, \mathit{stoppingcriteria}, N, T)$
Output: P

1 $P \leftarrow \emptyset$
2 $\{\alpha_1, \ldots, \alpha_N\} \leftarrow \texttt{RandomPopulation}(F, O, N)$
3 $z \leftarrow \texttt{ReferencePoint}(F, O)$
4 $\{\lambda^1, \ldots, \lambda^N\} \leftarrow \texttt{WeightVectors}(F, O, N)$
5 **for** $i = 1$ *to* N **do**
6 $B_i \leftarrow \texttt{ComputeNeighbours}(\lambda^i, \{\lambda^1, \ldots, \lambda^N\} \setminus \lambda^i, T)$
7 **while** $\neg \mathit{stoppingcriteria}$ **do**
8 **for** $i = 1$ *to* N **do**
9 $l, m \leftarrow \texttt{RandomIndex}(B_i)$
10 $\alpha \leftarrow \texttt{Reproduction}(\alpha_l, \alpha_m)$
11 $\alpha \leftarrow \texttt{Improvement}(\alpha)$
12 **for** $j = 1$ *to* k **do**
13 **if** $f_j(\alpha) < z_j$ **then**
14 $z_j \leftarrow f_j(\alpha)$
15 **foreach** $j \in B_i$ **do**
16 **if** $g(\alpha|\lambda^j, z) \leq g(\alpha_j|\lambda^j, z)$ **then**
17 $\alpha_j \leftarrow \alpha$
18 $P \leftarrow \texttt{UpdateNonDominated}(P \cup \{\alpha\})$
19 **return** P

through an optional improvement operator, that must be provided by the user (line 11). Next, if the value of $f_j(\alpha)$ is smaller than z_j, z_j is updated to $f_j(\alpha)$ (lines 12 - 14). After that, the neighboring solutions of α_i that are worse than α, according to the Tchebycheff approach, get replaced by α in the respective sub-problem (lines 15 - 17). Finally, the approximation P is updated, by removing all individuals in P that are dominated by α and adding α if it is not dominated by any of the individuals in P (line 18).

Another common scalarization that could be used is the weighted sum approach. It is defined as follows:

$$\begin{aligned} &\text{minimize } g(\alpha|\lambda) = \sum_{i=1}^{k} \{\lambda_i \cdot f_i(\alpha)\} \\ &\text{subject to } \alpha(F) = 1 \end{aligned} \tag{3.2}$$

The weighted sum approach has the disadvantage that there may exist Pareto-optimal solutions $\alpha*$ for which no weight vector exists such that $\alpha*$ is an optimal solution of the weighted sum. These solutions are known as unsupported solutions.

In the case where the Pareto-front is concave this approach could work well. However, when the Pareto-front is non-concave we can not obtain every Pareto optimal vector.

Algorithm 3.4: IBEA Algorithm

Input: (F, O), N, $stoppingcriteria$, κ
Output: A

```
1  A ← RandomPopulation(F, O, N)
2  while ¬stoppingcriteria do
3      foreach α1 ∈ A do
4          F(α1) = Σ_{α2 ∈ A\{α1}} −e^{−I({α2},{α1})/κ}
5      while |P| > N do
6          α* ← SmallestFitness(A)
7          A ← A \ {α*}
8          foreach α ∈ A do
9              F(α) = F(α) + e^{−I({α*},{α})/κ}
10     A' ← OffspringPopulation(A)
11     A ← A ∪ A'
12 return A
```

3.1.3 IBEA Algorithm

IBEA [34] uses indicators that measure the quality of the approximation in order to select the best individuals. These individuals are ranked based on their contribution to the quality of the approximation and the ones that contribute the most are selected for the next generation.

The pseudo-code for IBEA is presented in Algorithm 3.4. The algorithm starts by generating a random population of size N (line 1). Then, it calculates the fitness values of each individual, which ranks the individuals according to their contribution to the goal, using a fitness assignment function (line 4). Then, it iteratively removes the worst individual and updates the fitness value of the remaining ones (lines 6 - 9). Next, it goes through mutation and crossover procedures (line 11). These steps are repeated until the stopping_criteria is triggered.

A quality indicator is a function that maps k Pareto-front approximations to a real number. It represents a natural extension of the Pareto dominance relation, so it can be directly used to calculate the fitness. However, the considered indicator needs to preserve dominance, i.e. $\alpha_1 \prec \alpha_2 \rightarrow F(\alpha_1) > F(\alpha_2)$.

Various indicators could be used to compare the quality of two Pareto-front approximations [8]. In this work, we will focus on the I_{HD}-indicator.

The I_{HD}-indicator is based on the hypervolume metric presented in Section 2.5, and it is defined as follows:

$$I_{HD}(A, B) = \begin{cases} HV(B) - HV(A) & \text{if } \forall x_2 \in B\ \exists x_1 \in A : x_1 \prec x_2 \\ HV(A \cup B) - HV(A) & \text{otherwise} \end{cases} \tag{3.3}$$

$I_{HD}(A, B)$ measures the space dominated by B but not by A with respect to a reference point z.

Now, given an optimization problem and a binary quality indicator I, we can define the goal of the optimization process as minimizing $I(A, \tilde{A})$ where $\tilde{A}$ is the Pareto set. Note that $\tilde{A}$ is the formalization of the goal and is not required to be known.

Let A be a population. The fitness assignment ranks the population members according to their contribution to the goal. The approach that is used amplifies the influence of nondominated population members over dominated ones. It is defined as follows:

$$F(x_1) = \sum_{x_2 \in A \setminus \{x_1\}} -e^{-I(\{x_2\},\{x_1\})/\kappa} \tag{3.4}$$

The parameter κ is a fitness scaling factor that depends on I and needs to be greater than 0.

3.2 Constraint Based Algorithms

Constraint based algorithms rely on a constraint solving oracle, in order to find assignments that are guaranteed to satisfy all the constraints, and then blocking the assignments found, so that new solutions can be found.

Let $M = (F, O)$ be a MOCO instance, where F is a set of m constraints and O is a set of k cost functions. Let $\tilde{A}$ be the set of all Pareto-front solutions. The final set of solutions that a constraint-based MOCO solver returns (A) is:

- Sound: all solutions in A satisfy all the constraints, i.e. $\alpha \in A \rightarrow \alpha(F) = 1$
- Optimal: all solutions in A belong to the Pareto-front, i.e. $\alpha \in A \rightarrow \alpha \in \tilde{A}$
- Complete: all the solutions that belong to the Pareto-front are in A, i.e. $\alpha \in \tilde{A} \rightarrow \alpha \in A$;

In this section, some of these algorithms are presented. First, the Opportunistic Improvement Algorithm (OIA) is described in section 3.2.1. Then, the Guided Improvement Algorithm (GIA) is described in section 3.2.2. The P-minimal models algorithm is presented in section 3.2.3. Finally, the MCS-based algorithm is presented in section 3.2.4.

3.2.1 OIA Algorithm

Most of the times the user cannot find all the optimal solutions in useful time, due to the fact that the Pareto-front can be really large (in the worst case, the size of the Pareto-front is exponential). For this reason, we want the algorithms to yield solutions as they are being found, since the entire Pareto-front may take too long to obtain. The OIA algorithm [10] only guarantees that the solutions found before a possible timeout are sound since it is not possible to determine if a solution is also optimal before its execution ends.

The pseudo-code is presented in Algorithm 3.5 and it works as follows. After initializing the output set A (line 1) we call the PB solver on the set of constraints F in order to compute a solution α (line 2). Then the algorithm enters a loop where it adds the new solution α to A and filters the set, by removing all the solutions that are dominated by α (line 4). We then add constraints to F to ensure that the next solutions are not dominated by α (lines 5 - 6). The algorithm ends when F becomes unsatisfiable.

Algorithm 3.5: OIA Algorithm

Input: F, O
Output: A

```
1 A ← ∅
2 α ← PBSolver(F)
3 while α ≠ ∅ do
4     A ← UpdateNonDominated(A ∪ {α})
5     (v_1, ..., v_k) ← (f_1(α), ..., f_k(α))
6     F ← F ∪ {⋁_{f_i ∈ O}(f_i ≤ v_i − 1)}
7     α ← PBSolver(F)
8 return A
```

Algorithm 3.6: GIA Algorithm

Input: F, O
Output: A

```
1  A ← ∅
2  α ← PBSolver(F)
3  while α ≠ ∅ do
4      F' ← F
5      while α ≠ ∅ do
6          α' ← α
7          (v_1, ..., v_k) ← (f_1(α), ..., f_k(α))
8          F' ← F' ∪ ⋃_{f_i ∈ O}{(f_i ≤ v_i)} ∪ {⋁_{f_i ∈ O}(f_i ≤ v_i − 1)}
9          α ← PBSolver(F')
10     A ← A ∪ {α'}
11     F ← F ∪ {⋁_{f_i ∈ O}(f_i ≤ v_i − 1)}
12     α ← PBSolver(F)
13 return A
```

3.2.2 GIA Algorithm

The GIA algorithm [13] is similar to OIA. The main difference is the inner loop that GIA has which guarantees that all solutions remaining after the loop belong to the Pareto-front .

In Algorithm 3.6 we can see the pseudo-code for the GIA algorithm. The algorithm starts by calling the PB solver on the set of constraints F and saving the result in α (line 2). We then enter a loop where we augment the formula in order to search for a new solution that dominates α and call the solver again (lines 5 - 9). After leaving the loop we know that α' is a solution belonging to the Pareto-front since the loop only ends when the PB solver cannot find another solution that dominates α'. This way, GIA guarantees that all solutions found are optimal as well as sound. Hence, we can safely yield α' (line 10). Finally, we add constraints to F to ensure that the next solutions are not dominated by α' (line 11).

3.2.3 P-Minimal Models Algorithm

More recently, an algorithm based on P-minimal models [22] was proposed. This algorithm relies on using order encoding to translate the cost functions to a Conjunctive Normal Form (CNF) formula and searching for the minimal models of the encoding's variables since there exists a correspondence between P-minimal models and Pareto-optimal solutions [22].

Let α be a model of a CNF formula over the variables X. Let P be a set such that $P \subseteq X$. Let $\alpha|P$ denote the partial assignment that results from removing the mappings of variables that are not in P from α, i.e., $\alpha|P = \{(x, \alpha(x)) : x \in P)\}$. We use $P_{\alpha|1} = \{x : (x, 1) \in \alpha|P\}$ to denote the set of variables of P that are assigned 1 by α. Suppose that no model α' exists such that $P_{\alpha'|1} \subset P_{\alpha|1}$. Then $\alpha|P$ is a P-minimal model.

In the order encoding, given a cost function f_i of a MOCO problem with upper bound $ub(f_i)$, we introduce one auxiliary Boolean variable $p_{f_i,v}$ for each possible value v of f_i, where $0 < v \leq ub(f_i)$. If a variable $p_{f_i,v} = 1$ then this indicates that $f_i \geq v$. Since $f_i \geq 0$ is always true we do not need to use the variable $p_{f_i,0}$. Then for each one of these variables, auxiliary constraints are added to force each variable $p_{f_i,v}$ to be 1 if and only if $f_i \geq v$.

The pseudo-code for the P-minimal models algorithm is represented in Algorithm 3.7. First, the algorithm starts by building the order encodings of the cost functions in O (line 2), returning a formula F_P and a set P that contains the auxiliary variables. A model α is extracted from F_P (line 3). A clone F'_P of the formula F_P is created (line 5). Constraints are then added to F'_P forcing the $p \in P$ that were not satisfied in the model α to remain not satisfied and at least one variable $p_{f_i,f_i(\alpha)}$ to become satisfied (line 8). If the formula remains satisfiable, then α is not a P-minimal model and the inner loop continues, by adding more constrains and calling the solver again. Otherwise α is a P-minimal model and is added to A (line 10). Finally, constraints are added to F_P, so that new found solutions are not dominated by α (line 11). This is repeated until F_P becomes unsatisfiable. At that point all the P-minimal models, i.e. Pareto-optimal solutions, have been found.

3.2.4 MCS Enumeration Algorithm

Let $F_H = \{c_1, \ldots, c_m\}$ be a set of hard PB constraints and $O_S = \{F_{S_1}, \ldots, F_{S_k}\}$ be a set of soft weighted PB constraint sets. A Multi-Objective Weighted Boolean Optimization (MOWBO) instance is defined as $\mathbb{W} = (F_H, O_S)$. As in a MOCO problem, the goal is to find its set of Pareto-optimal solutions.

Let $\mathbb{W} = (F_H, O_S)$ be a MOWBO instance and $\mathbb{C} = (C_1, \ldots, C_k)$ be a tuple of sets such that $C_i \subseteq F_{S_i}$, $1 \leq i \leq k$. $\mathbb{C}$ is a Multi-MCS of $\mathbb{W}$ if and only if $F_H \cup \bigcup_{i=1}^{k}(F_{S_i} \setminus C_i)$ is satisfiable and $F_H \cup \bigcup_{i=1}^{k}(F_{S_i} \setminus C_i) \cup \{c\}$ is unsatisfiable for all $c \in \bigcup_{i=1}^{k} C_i$.

Let $\mathbb{C} = (C_1, \ldots, C_k)$ and $\mathbb{C}' = (C'_1, \ldots, C'_k)$ be two Multi-MCSs $\mathbb{W}$. $\mathbb{C}$ dominates $\mathbb{C}'$ ($\mathbb{C} \prec \mathbb{C}'$) if and only if $\forall_{1 \leq i \leq k} \sum_{(c,\omega) \in C_i} \omega \leq \sum_{(c',\omega') \in C'_i} \omega'$ and $\exists_{1 \leq i \leq k} \sum_{(c,\omega) \in C_i} \omega < \sum_{(c',\omega') \in C'_i} \omega'$. If no other Multi-MCS $\mathbb{C}'$ exists such that $\mathbb{C}' \prec \mathbb{C}$, then $\mathbb{C}$ is a Pareto-MCS.

Similar to the single-objective case (section 2.3), it is possible to reduce a MOCO instance $M = (F, O)$ to a MOWBO instance $\mathbb{W} = (F_H, O_S)$. We set $F_H = F$ and for each $f_i \in O$, where $f_i =$

Algorithm 3.7: P-Minimal Model Algorithm

Input: F, O
Output: A

1 $A \leftarrow \emptyset$
2 $(F_P, P) \leftarrow \texttt{OrderEncoding}(F, O)$
3 $\alpha \leftarrow \texttt{PBSolver}(F_P)$
4 **while** $\alpha \neq \emptyset$ **do**
5 $\quad F'_P \leftarrow F_P$
6 $\quad$ **while** $\alpha \neq \emptyset$ **do**
7 $\quad\quad \alpha' \leftarrow \alpha$
8 $\quad\quad F'_P \leftarrow F'_P \cup \bigcup_{p \in P_{\alpha|1}} \{(\neg p)\} \cup \{(\bigvee_{i=1}^{k} \neg p_{f_i, f_i(\alpha)})\}$
9 $\quad\quad \alpha \leftarrow \texttt{PBSolver}(F'_P)$
10 $\quad A \leftarrow A \cup \{\alpha'\}$
11 $\quad F_P \leftarrow F_P \cup \{(\bigvee_{i=1}^{k} \neg p_{f_i, f_i(\alpha')})\}$
12 $\quad \alpha \leftarrow \texttt{PBSolver}(F_P)$
13 **return** A

$\omega_1 \cdot l_1 + \ldots + \omega_o \cdot l_o$, we add a set of soft constraints $F_{S_i} = \{(\neg l_1, \omega_1), \ldots, (\neg l_o, \omega_o)\}$ to O_S.

Example 6. *Consider a MOCO instance that is constituted by a set of constraints* $F = \{(x_1 + x_2 \geq 1), (2x_1 + x_2 - x_3 \leq 1)\}$ *and a set of cost functions* $O = \{(5x_1 - 2x_2), (x_2 + \neg x_3)\}$*. An equivalent MOWBO instance would have* $F_H = \{(x_1 + x_2 \geq 1), (2x_1 + x_2 - x_3 \leq 1)\}$ *and* $O_S = \{\{(\neg x_1, 5), (\neg x_2, -2)\}, \{(\neg x_2, 1), (x_3, 1)\}\}$.

To identify all the Pareto-optimal solutions of a MOCO problem we just need to find all the MCSs of the MOWBO problem, since there exists an equivalence between Pareto-MCSs and Pareto-optimal solutions [24].

Let $\mathbb{W} = (F_H, O_S)$ be a MOWBO instance with $O_S = \{F_{S_1}, \ldots, F_{S_k}\}$, and $\mathbb{C} = (C_1, \ldots, C_k)$ be a Multi-MCS of $\mathbb{W}$. Then, $C = \bigcup_{i=1}^{k} C_i$ is an MCS of the WBO instance $W = (F_H, \bigcup_{i=1}^{k} F_{S_i})$. This implies that Pareto-MCS enumeration of a MOWBO instance $\mathbb{W} = (F_H, O_S)$ can be reduced to enumerating the MCSs of the WBO instance $W = (F_H, \cup_{F_S \in O_S} F_S)$ [24].

The pseudo-code for the Pareto-MCS enumeration algorithm [24] can be found in Algorithm 3.8. The algorithm receives a MOWBO instance. It starts by combining the soft constraint sets into a single set (line 1). Then it builds a clone F'_H of the hard constraints F_H (line 2). Then, while F'_H is satisfiable (line 5). While this is satisfiable, an MCS C is extracted for the WBO instance (F'_H, F'_S) and is stored in U (lines 5 - 9). To prevent the MCS C from being extracted again in the future, a blocking constraint is added to F'_H (line 7). In the end of the loop all the MCSs have been found. All the MCSs are then converted into their respective Multi-MCSs (line 10). Finally the Pareto-MCSs are filtered by removing the Multi-MCSs in U_{multi} that are dominated by other Multi-MCSs (line 11).

Stratification techniques that can be used to help finding lower cost MCSs faster, can be integrated into the MCS enumeration algorithm to improve the algorithm's performance [25]. The main idea of stratification is to focus on satisfying the literals with larger coefficients. In the single-objective scenario, this is done by partitioning F_S into k sets $A_1, \ldots, A_k$ such that all literals in A_i have higher coefficients

Algorithm 3.8: MCS Enumeration Algorithm

Input: F_H, O_S
Output: U_{pareto}

1 $F'_S \leftarrow \bigcup_{i=1}^{n} F_{Si}$
2 $F'_H \leftarrow F_H$
3 $U \leftarrow \emptyset$
4 $\alpha \leftarrow \texttt{PBSolver}(F'_H)$
5 **while** $\alpha \neq \emptyset$ **do**
6 $\quad C \leftarrow \texttt{MCS}(F'_H, F'_S)$
7 $\quad F'_H \leftarrow F'_H \cup \{(\bigvee_{(c,\omega)\in C} c)\}$
8 $\quad U \leftarrow U \cup \{C\}$
9 $\quad \alpha \leftarrow \texttt{PBSolver}(F'_H)$
10 $U_{multi} \leftarrow \{(C \cap F_{S1}, \ldots, C \cap F_{Sn}) : C \in U\}$
11 $U_{pareto} \leftarrow \{\mathbb{C} : \mathbb{C} \in U_{multi} \wedge \nexists_{\mathbb{C}' \in U_{multi}} \mathbb{C}' \prec \mathbb{C}\}$
12 **return** U_{pareto}

than the ones in A_{i+1}. Next, at each iteration $1 \leq i \leq k$, the MCS algorithm is used with A_i as the set of soft constraints while additional hard constraints are considered in order to ensure consistency with the MCSs computed in previous iterations.

The pseudo-code for a stratified MCS algorithm for MOWBO problem is presented in Algorithm 3.9. The algorithm starts by applying stratification on each soft constraint F_{Si}, splitting the set $L^{\neg}(F_{Si})$ into k_i partitions (line 2). Then, all partitions are combined in order to create a single sequence of partitions of size p where $p = \sum_{i=1}^{l} k_i$ (line 3). Assume there are three soft constraints F_{S1}, F_{S2} and F_{S3} with partitions $P_1^1 P_2^1$, $P_1^2 P_2^2$ and $P_1^3 P_2^3 P_3^3$, a possible combined partition would be $P_1^1 P_1^2 P_1^3 P_2^2 P_2^1 P_2^3 P_3^3$. A loop then begins where in each iteration the set S_i denotes the set of constraints to be satisfied while C denotes the set of unsatisfied constraints. In the first iteration, all hard constraints must be satisfied (line 4). For each partition P_i, the algorithm computes an MCS C_i of P_i with S_i as a set of hard constraints (line 6). S_{i+1} is then created by adding the constraints in $P_i \setminus C_i$, since these are satisfied, and also by adding the negation of the constraints in C_i, to restrict the search space in the next MCS extraction, in order to ensure consistency in future iterations with previous MCSs (line 7). C is also updated by adding the constraints in C_i (line 8). In the end, we have that C is an MCS of the MOWBO instance.

3.3 Hybrid Algorithms

There are several challenging real-world applications of MOCO where typical approaches fail to provide good solutions in a reasonable amount of time. Experimental results show that both IBEA and NSGA-II fail to provide a valid solution after 30 minutes in the Software Product Line Configuration (SPLC) problem [21]. Other studies show that it is unfeasible to compute exact solutions for instances with more than 45 variables [19]. Hybrid algorithms were created to solve some of these problems, using both evolutionary and constraint based techniques.

In this section we present some of these algorithms. SATIBEA is presented in section 3.3.1 and

Algorithm 3.9: Stratified MCS algorithm for MOWBO

Input: F_H, O_S
Output: C

for $i = 1$ *to* k **do**
  $(P_1^i, \ldots, P_{k_i}^i) = \texttt{Partition}\ (L^{\neg}(F_{Si}))$
$(P_1, \ldots, P_p) = \texttt{Combine}\ ((P_1^1, \ldots, P_{k_1}^1), \ldots, (P_1^k, \ldots, P_{k_k}^k))$
$(S_1, C) = (F_H, \emptyset)$
for $i = 1$ *to* p **do**
  $C_i = \texttt{MCS}\ (S_i, P_i)$
  $S_{i+1} = S_i \cup (P_i \setminus C_i) \cup \bigcup_{c \in C_i} \{\neg c\}$
  $C = C \cup C_i$
return C

SATVaEA is presented in section 3.3.2.

3.3.1 SATIBEA

SATIBEA [12] searches for a Pareto-front approximation by using a combination of SAT solving and the state-of-art IBEA algorithm. There are two key aspects to be considered: diversity promotion and the use of smart operators.

Smart operators are operators that use SAT solvers. There are two smart operators that SATIBEA uses:

- Smart mutation: finds variables that are involved in the violation of constraints and removes their assignments. Then a SAT solver is used to complete the resulting partial assignment, returning a satisfying one.
- Smart replacement: randomly picks an assignment from the solutions and replaces it with a valid one, improving the quality and diversity of the solutions.

The diversity promotion is ensured by introducing randomization in the smart mutation and smart replacement operators. Such randomization has an impact on the model that is returned by the solver. There are three parameters to randomly permute:

- The order in which constraints are added to the solver.
- The order in which the literals appear in each constraint.
- The order in which value assignments to variables ($\{true, false\}$) are instantiated.

This smart mutation operator was proposed for the SPLC, which can be expressed as a CNF formula. In this case, the operator considers that the variables in unsatisfied clauses are the ones responsible for constraint violation. However, this works for SPLC due to certain properties of the SAT encoding, not generalizing to other applications, especially when other types of constraints, such as PB constraints, are involved. For this reason, a smart mutation operator that uses unsat cores to determine which variable assignments are responsible for constraint violations was proposed [26].

3.3.2 SATVaEA

The main idea behind the SATVaEA [29] is to combine the Vector angle based Evolutionary Algorithm (VaEA) [28] with two types of SAT solvers: an SLS-style SAT solver and a CDCL-style SAT solver.

The pseudo-code for SATVaEA is presented in Algorithm 3.10. The algorithm starts by simplifying the MOCO instance M, by applying Boolean Constraint Propagation [31] (BCP) (line 1). This is done to prepare the MOCO instance for the application of both styles of SAT solvers. A random population A with size N is then created, and an offspring population B is created by applying mutation and crossover operators to A (lines 2 - 4). Next, a set R is created from the union of A and B (line 5). If there is an invalid solution in R, then, with probability θ, an SLS-style SAT solver is used to repair a random invalid solution of R (line 8), or, with probability $1 - \theta$, a CDCL-style SAT solver is used to generate a valid solution that will then replace a random invalid solution of R (line 10). Next, a new population A is created by performing environmental selection to R (line 11). The environmental selection is based on a two-level ranking method, that separates the individuals into layers. The first-level ranking is based on the number of violated constraints, where individuals with the same number of violated constraints are put in the same layer. The second-level ranking uses the nondominated sorting procedure from NSGA-II [6], which divides individuals in the same layer into different sub-layers according to their Pareto-dominance. The process is repeated until the stoppingcriteria is verified, and at that point the population A is returned (line 12).

Algorithm 3.10: SATVaEA Algorithm

Input: $F, O, N, \theta, stoppingcriteria$
Output: A

1 $F, O \leftarrow \texttt{Simplify}(F, O)$
2 $A \leftarrow \texttt{RandomPopulation}(F, O, N)$
3 **while** $\neg stoppingcriteria$ **do**
4 $\quad B \leftarrow \texttt{OffspringPopulation}(A)$
5 $\quad R \leftarrow A \cup B$
6 $\quad$ **if** $\texttt{HasInvalid}(R)$ **then**
7 $\quad\quad$ **if** $\texttt{Random}() \leq \theta$ **then**
8 $\quad\quad\quad R \leftarrow \texttt{ApplyRepairInvalid}(R)$
9 $\quad\quad$ **else**
10 $\quad\quad\quad R \leftarrow \texttt{ApplyReplaceInvalid}(R)$
11 $\quad A \leftarrow \texttt{EnvironmentalSelection}(R, N)$
12 **return** A

Chapter 4

Neon - an Hybrid Multi-Objective Combinatorial Optimization Software

Our new MOCO solver is developed on top of sat4j-moco, a solver for generic MOCO instances that implements the MCS based algorithm (see section 3.2.4 for details). To extend sat4j-moco, we developed a generic hybrid algorithm that combines stochastic search and constraint solving, in an attempt to generalize the algorithm proposed in [26].

In this chapter, we present Neon, a Hybrid MOCO solver. First, we cover the stochastic approach in Section 4.1. Then, in Section 4.2 we propose to integrate constraint-based techniques with stochastic algorithms for MOCO, thus creating an hybrid approach. Finally, we propose a technique that improves the performance of the stochastic algorithm by exploiting constraints in the MOCO instances in Section 4.3.

4.1 Stochastic Approach

The first objective of this project is to implement a fully functional stochastic algorithm. The stochastic algorithms used in Neon are the NSGAII and the MOEAD algorithms, introduced in sections 3.1.1 and 3.1.2, respectively. Both algorithms are already implemented in the MOEA framework [1].

Neon receives as input a file in an extended OPB format [2] that supports multiple objective functions. No additional information is provided about the problem to be solved since Neon was created to solve generic MOCO problems.

The stochastic algorithms both use the same operators, offered by the MOEA framework. For the initialization of the initial population, a random operator which initializes all variables uniformly at random is used. Next, mutation and crossover operators are used to evolve the population. For the mutation, the user can choose between two operators. As the default mutation operator, we implement the single point mutation operator (SPM). This operator mutates an individual with a probability of p. If it chooses

[1] http://moeaframework.org/
[2] http://www.cril.univ-artois.fr/PB16/format.pdf

to mutate the individual, it then selects one of its Boolean variable assignments, and with uniform probability, it changes it's value. The second mutation operator is the uniform mutation operator (UM) already implemented in the MOEA framework. The UM operator chooses for every variable if it is going to be mutated, while the SPM operator chooses for every individual if a random variable is mutated. This means that UM makes bigger mutations but takes longer to do so, when compared to SPM. For the crossover operator, we use the uniform crossover (UX), an operator that changes two individuals by iterating over all the variables and, with some probability, it exchanges the value of the variable i from one individual with the same variable i from the other individual, thus creating two new individuals in the end.

Example 7. *Consider the set of variables* $X = \{x_1, x_2, x_3\}$. *Let* $\alpha_1 = \{(x_1, 0), (x_2, 0), (x_3, 0)\}$ *be an assignment over the set* X. *Now imagine that we apply the SPM operator over this assignment, a new possible assignment, where the value in bold would be the value that the operator evolved, is* $\alpha_{1'} = \{(x_1, 0), (x_2, \mathbf{1}), (x_3, 0)\}$.

Example 8. *Consider the set of variables* $X = \{x_1, x_2, x_3\}$. *Let* $\alpha_1 = \{(x_1, 0), (x_2, 0), (x_3, 0)\}$ *be an assignment over the set* X. *If we apply the UM operator over this assignment, a new possible assignment can be* $\alpha_{1'} = \{(x_1, \mathbf{1}), (x_2, \mathbf{1}), (x_3, 0)\}$, *where both* x_1 *and* x_2 *are mutated.*

Example 9. *Consider the set of variables* $X = \{x_1, x_2, x_3, x_4, x_5\}$. *Let* $\alpha_1 = \{(x_1, 0), (x_2, 0), (x_3, 0), (x_4, 0), (x_5, 0)\}$ *and* $\alpha_2 = \{(x_1, 1), (x_2, 1), (x_3, 1), (x_4, 1), (x_5, 1)\}$ *be two assignments over the set* X. *After applying the UX operator over these two individuals, we could get as new possible assignments* $\alpha_{1'} = \{(x_1, 0), (x_2, 1), (x_3, 0), (x_4, 0), (x_5, 1)\}$ *and* $\alpha_{2'} = \{(x_1, 1), (x_2, 0), (x_3, 1), (x_4, 1), (x_5, 0)\}$, *where the assignments of the variables* x_2 *and* x_5 *swapped from one individual to the other.*

A feature added, which is enabled by default, is the unit propagation (UP), a standard feature in SAT and pseudo-Boolean solvers usually used to simplify a set of propositional clauses. The procedure is based on solving clauses that have a single literal l and since all clauses must be satisfied then we know the literal has to be true. All the other clauses that contain the literal l are removed since they are satisfied and the literal $\neg l$ is removed from all clauses remaining, leaving a simplified set of clauses equivalent to the initial. In the pseudo-Boolean case the idea is to find constraints that have a unique assignment to it's variables in order to become satisfiable and force those variables to keep the assignment found. All the other constraints that contain any of the forced variables can be simplified, by replacing the variable with the value assigned previously. All the deduced variables keep their values during the run of the stochastic algorithm, so therefore there is no need to encode these variables into the individual.

Example 10. *Let* F *be a set of four constraints where* $F = \{(x_1 >= 1), (-x_2 - x_3 >= 0), (x_4 + x_5 >= 1), (x_2 + x_6 >= 1)\}$. *By applying UP to this set of constraints we can infer, from the first constraint, that* $x_1 = 1$ *in all feasible assignments and that* $x_2 = 0, x_3 = 0$ *from the second constraint, since if* x_2 *or* x_3 *are set to* 1 *the constraint is violated. From the third and forth constraints, we cannot infer anything, since these constraints are satisfied if at least one of the variables is set to* 1. *However, we now know that* $x_2 = 0$, *which means that the forth constraint can now be reduced to* $(x_6 >= 1)$.

Algorithm 4.1: Smart mutation algorithm

Input: $OriginalIndividual$, V, F
Output: $FixedIndividual$

1 $A \leftarrow \cup_{x \in OriginalIndividual, x \notin V} \{(x)\}$
2 $IsSat \leftarrow \texttt{Solve}(F, A)$
3 **while** $\neg IsSat$ **and** $A \neq \emptyset$ **do**
4 $\quad C \leftarrow \texttt{GetUnsatCore}()$
5 $\quad A \leftarrow A \setminus (A \cap C)$
6 $\quad IsSat \leftarrow \texttt{Solve}(F, A)$
7 **if** $\neg IsSat$ **then**
8 $\quad \texttt{SetParetoTerminationCondition}()$
9 $\quad$ **return** $OriginalIndividual$
10 $FixedIndividual \leftarrow \texttt{GetModel}()$
11 $\texttt{AddBlockClause}(\sum_{x \in FixedIndividual} \{(\neg x)\} \geq 1)$
12 **return** $FixedIndividual$

We can now infer that $x_6 = 1$, just as it happened with the first constraint. Therefore, after applying the UP we find that a feasible assignment to this problem must have the partial assignment $\alpha = \{(x_1, 1), (x_2, 0), (x_3, 0), (x_4, \_), (x_5, \_), (x_6, 1)\}$ and all constraints but the third one can be removed, since they are always satisfied by the partial assignment α.

4.2 Hybrid Approach

In this section, we explore the smart operators added to the stochastic algorithms, to create a hybrid algorithm. Studies show that stochastic algorithms fail to provide valid solutions for big and tightly constrained instances in reasonable time [21, 19, 26]. Hence, Neon integrates smart operators that use constraint solvers in order to find feasible individuals. Neon uses two smart operators, the first is the smart mutation operator, which produces a feasible individual by fixing an unfeasible one. The other one is the smart improvement operator, which produces an improved version of an already feasible individual.

Before applying any of the operators, Neon selects an individual within the given population. After selecting an individual, it finds the set of literals that belong to constraints violated by the assignment. If the set is empty then the individual is feasible and smart improvement is applied, otherwise smart mutation is applied.

4.2.1 Smart Mutation

The pseudo-code for smart mutation is presented in Algorithm 4.1. Smart mutation receives the individual to be fixed, a set of violating variables V, which is the set of literals belonging to constraints violated by the assignment and the set of constraints F. Smart mutation starts by creating a set of assumptions (line 1), which is a partial assignment that the PB solver tries to satisfy along with the problem's

constraints. The set of assumptions is constituted by all literals except the ones from the set V.

Example 11. *Let F be a set of three constraints defined by $F = \{(x_1 + x_2 \geq 1), (x_2 + x_3 + x_4 \geq 2), (x_5 \geq 1)\}$ and let $\alpha = \{(x_1, 0), (x_2, 1), (x_3, 0), (x_4, 0), (x_5, 1)\}$ be an assignment for F. The set of assumptions would be $\{(x_1, 0), (x_5, 1)\}$, since the second constraint is violated by the assignment and x_2, x_3 and x_4 appear in that constraint, therefore the respective literals are not include in the assumptions.*

Afterwards, a SAT solver tries to find a new model for the problem that satisfies both the problem's constraints and the assumptions (line 2). If an assignment is not found, the operator tries to find where the conflict is, by getting an unsat core (line 4), which is a subset of assumption literals that make the formula unsatisfiable, then remove those literals from the assumptions (line 5) and call the solver again (line 6). This process is repeated until the formula becomes satisfiable, the set of assumptions becomes empty or a conflict budget is reached. The conflict budget is used to ensure that the algorithm does not get stuck in smart mutation, finishing unsuccessfully. If the solver proves unsatisfiability when the set of assumptions is empty (line 7), then a termination condition for the stochastic algorithm is activated (line 8). In both cases smart mutation returns the individual that was to be mutated (line 9), since no new individual was found. The termination condition exists due to the fact that every time a model is found (line 10), a blocking constraint is added to the formula (line 11) to guarantee that the same model is not found again by the SAT solver, promoting diversity in the population. Therefore, if the solver cannot find more models, it means that all the existing models have been found and blocked and any further search is unnecessary because the Pareto front was found. The pseudo-Boolean blocking constraint is constituted by the sum of the negation of the literals found in the assignment being greater or equal than 1, forcing future assignments to have at least one literal different from the assignments previously found.

Example 12. *Let $\alpha = \{(x_1, 0), (x_2, 1), (x_3, 1), (x_4, 0)\}$ be a model returned by the SAT solver. The blocking constraint for α is $x_1 + \neg x_2 + \neg x_3 + x_4 \geq 1$.*

If this point is reached, then this run of smart mutation is considered to be successful and the new individual is returned (line 12).

4.2.2 Smart improvement

The pseudo-code for smart improvement is presented in Algorithm 4.2. Smart improvement also receives the original individual. It starts by creating a set of assumptions (line 1), however, since this operator is applied to feasible individuals, there are no violated constraints. Therefore, to get the set of assumptions, the smart improvement operator starts by iterating over each equals-1 and at-most-1 constraints, which is the type of constraints that we know how to exploit in the structure improvements technique, and with some probability, referred to as relaxation rate, it chooses to either add all the literals belonging to each constraint to the assumptions or adds none of them. For the literals that do not belong to any of these constraints, the same is done but to each literal individually.

Example 13. *Let F be a set of three constraints defined by $F = \{(x_1 + x_2 \leq 1), (x_3 + x_4 = 2), (x_5 + x_6 \geq 1)\}$ and let $\alpha = \{(x_1, 0), (x_2, 1), (x_3, 1), (x_4, 1), (x_5, 1), (x_6, 0)\}$ be an assignment for F. The first*

Algorithm 4.2: Smart improvement algorithm

Input: $OriginalIndividual, F, O$
Output: $ImprovedIndividual$

```
1  A ← GetImprovementAssumptions(∪_{x∈OriginalIndividual}{(x)})
2  MCSOracle(F ∪ A, O)
3  ImprovedIndividual ← GetModel()
4  if ImprovedIndividual = ∅ then
5      IsUnsat ← Solve(A)
6      if IsUnsat then
7          SetParetoTerminationCondition()
8      return OriginalIndividual
9  MCS ← GetMCS()
10 AddBlockClause(∪_{x∈MCS}{(x)})
11 return ImprovedIndividual
```

two constraints are of the type at-most-1 and exactly-1, respectively, therefore, for each one of these constraints, we choose with random probability to either add all literals in the constraint to the assumption set, otherwise no literals are added . Since the literals x_5 and x_6 do not belong to an at-most-1 or exactly-1 constraint, we randomly choose to add them individually. A possible set of assumptions would be $\{(x_1, 0), (x_2, 1), (x_6, 0)\}$, if the operator chooses to add the literals from the the first constraint as well as x_6. The set of assumptions $\{(x_1, 0), (x_5, 1)\}$ is impossible to be obtained, because to add x_1 we also need to add x_2 to the assumptions.

Next, an MCS oracle is then called in order to find an MCS (line 2), using the state-of-art CLD algorithm with stratification [25]. In case the CLD algorithm finds an MCS, a model is then extracted, otherwise, the model is empty (line 3).

If no model is found, then a PB solver is called (line 5) to find if the formula is still satisfiable. If that is not the case, a termination condition for the stochastic algorithm is activated (line 7). The original individual is then returned (line 8).

If there is a model, the smart improvement operator gets the MCS found and adds it as a blocking clause to the formula so that CLD only finds each MCS once (lines 9 - 10) . The new individual is then returned and smart improvement is said to be successful (line 11).

Example 14. *Let $M = (F, O)$ be a MOCO problem with $F = \{(x_1 + x_2 \geq 1), (2x_1 + x_2 + x_3 \leq 2), (x_4 + x_5 = 1)\}$ is a set of constraints and $O = \{(3x_1 + x_2 + \neg x_3), (x_4 + 2\neg x_5)\}$ is a set of cost functions we want to minimize. An equivalent MOWBO instance would have $F_H = \{(x_1 + x_2 \geq 1), (2x_1 + x_2 + x_3 \leq 2), (x_4 + x_5 = 1)\}$ and $O_S = \{\{(\neg x_1, 3), (\neg x_2, 1), (x_3, 1)\}, \{(\neg x_4, 1), (x_5, 2)\}\}$. Let $\alpha = \{(x_1, 1), (x_2, 0), (x_3, 0), (x_4, 1), (x_5, 0)\}$ be an assignment for this problem with $O(\alpha) = (4, 3)$, where smart improvement is going to be applied on. Let a possible set of assumptions be $\{(x_4, 1), (x_5, 0)\}$. $\mathbb{C} = \{\{(\neg x_2, 1)\}, \{(\neg x_4, 1), (x_5, 2)\}\}$ be a Multi-MCS found by smart improvement. This Multi-MCS equals to the assignment $\alpha = \{(x_1, 0), (x_2, 1), (x_3, 1), (x_4, 1), (x_5, 0)\}$, where $O(\alpha) = (1, 3)$.*

4.3 Structure Improvements

In this section, the structure improvement (SI) technique is proposed. SI exploits the structure of the problem in order to improve the performance of the stochastic part of the hybrid algorithm. This is achieved by using integer variables to encode some constraints, instead of encoding each Boolean variable into the problem, since some constraints, such as the at-most-1 and exactly-1 constraints, could be satisfied by construction by using a different encoding. Let $(x_1 + \ldots + x_n = 1)$ be an exactly-1 constraint. By encoding these n variables into the individual, there are many possible assignments that do not satisfy the constraint, since to satisfy this constraint we need exactly one variable x_i where $1 \leq i \leq n$ to be set to 1 while all the other variables must be set to 0. A way to do this would be to encode the constraint as a variable $y \in \{1, \ldots, n\}$ where if y is set to i, then $x_i = 1$ and all the other variables are assigned to 0, which we denote as $(y, i) = \{(x_1, 0), \ldots, (x_i, 1), \ldots, (x_n, 0)\}$. If the constraint was an at-most-1 type, then 0 would also be part of the domain of y which would mean that all variables are set to 0.

Example 15. *Let $F = \{(x_1 + x_2 = 1), (x_3 + x_4 \leq 1), (x_5 + 2x_6 + x_7 = 2)\}$ be a set of PB constraints. If we encode $x_1, \ldots, x_7$ as Boolean variables, the stochastic algorithm can find many possible assignments, such as $\alpha_1 = \{(x_1, 1), (x_2, 1), (x_3, 0), (x_4, 0), (x_5, 1), (x_6, 1), (x_7, 0)\}$, which violates the first two constraints. Note that for a constraint $x_1 + \ldots + x_n = 1$ there are 2^n different assignments to these variables, however, only n different assignment satisfy the constraint. Using SI, an individual is encoded using only the variables y_1, y_2, x_5, x_6, x_7, where $y_1 \in \{1, 2\}$ encodes the first constraint and $y_2 \in \{0, 1, 2\}$ encodes the second constraint. Since the third constraint is not an at-most-1 or an exactly-1, we do not encode it using an integer variable. Let $\alpha = \{(y_1, 2), (y_2, 0), (x_5, 1), (x_6, 1), (x_7, 1)\}$ be an assignment for this encoding. We can see that α satisfies both the first and second constraint, since $(y_1, 2) = \{(x_1, 1), (x_2, 0)\}$ and $(y_2, 0) = \{(x_3, 0), (x_4, 0)\}$. As a matter of fact, all the possible assignments in the stochastic algorithm have to satisfy the two first constraints, only the third can be violated.*

The first step to implement the structure improvement technique is to find all the constraints that are of the type exactly-1 or at-most-1. Then, we choose which constraints to remove by prioritizing the ones with the most variables. Note that we must look at all the constraint's literals, since if a literal also belongs to a constraint that was already selected for the SI, then we cannot choose this constraint, since it may get different assignments from each constraint.

To encode a new individual for the stochastic algorithm there are two steps. First, the encoding of all the Boolean variables that do not have their assignment forced by the UP and do not belong to constraints that are being exploited by the SI. Then, there is the encoding of the exploited constraints. A new integer variable y that is exploiting a constraint c has a domain of $\cup_{x_i \in c, x_i \notin SI}\{x_i\}$, meaning that if a variable that is already being exploited by the UP then we do not encode it in the new variable. We call the set of variables that are not exploited by the UP as free variables. Note that if the constraint being exploited is a type at-most-1 then 0 must also belong to the domain of y.

Example 16. *Let $F = \{(x_1 = 0), (x_1 + x_2 + x_3 = 1)\}$ be a set of constraints. With unit propagation disabled, structure improvements will encode the second constraint as $y_1 \in \{1, 2, 3\}$. A possible assign-*

ment now could be $\alpha_1 = \{(y_1, 1)\}$*, meaning that the true assignment is* $\{(x_1, 1), (x_2, 0), (x_3, 0)\}$*. This assignment satisfies the second constraint, however, the first one is violated. With unit propagation enabled,* x_1 *will be set to* 0*, which can be concluded by analyzing the first constraint. This way, structure improvements will encode the second constraint as* $y_1 \in \{2, 3\}$*, where* $(y_1, 2) = \{(x_1, 0), (x_2, 1), (x_3, 0)\}$ *and* $(y_1, 3) = \{(x_1, 0), (x_2, 0), (x_3, 1)\}$*. This way all the possible assignments will not only satisfy the second constraint but also the first one.*

Now that the stochastic algorithm is now using mixed individuals while the smart operators use Boolean individuals. Therefore we must be able to translate from a mixed individual into a Boolean one and vice-versa.

A mixed individual is constituted by n_b Boolean variables and n_{si} integer variables that result from the structure improvements. To transform the individual into a full Boolean assignment, there are three types of variables that need to be taken into account: n_{up} variables fixed by the UP, n_{si} variables representing the constraints exploited by the SI and n_b Boolean variables. We now explain the three steps, one for each type of variable:

- We start by adding the n_{up} variables forced by the unit propagation to the Boolean assignment;
- Then we need to find out the position of each of the n_b variables in the Boolean assignment. To do this, a mapping that translates the positions is used.
- Finally there is the decoding of the n_{si} variables that represent the constraints exploited. We need to know which variable encodes each constraint. Then, for each value i of the assignment, we find the i^{th} non-forced variable and set it to **true** in the Boolean assignment while the other variables of the constraint are set to **false**.

Example 17. *Let* $F = \{(x_1 = 0), (x_1 + x_2 + x_4 = 1), (x_3 - x_5 \geq 0)\}$ *be a set of PB constraints. For this problem the UP would set* x_1 *to be* 0*, and the SI would encode the second constraint as* $y_1 \in \{1, 2\}$*, where* $y_1 = 1$ *means that* $x_2 = 1$ *and* $y_1 = 2$ *means* $x_4 = 1$*. Note that* x_1 *cannot be set to* 1 *in this encoding because it was already forced to be* 0 *by the UP. With this, our individual would be encoded as* $x'_1 x'_2 y_1$*.*

Now let $\alpha = \{(x'_1, 0), (x'_2, 1), (y_1, 2)\}$ *be an individual that we want to transform into a full Boolean assignment. From the UP we have that* $x_1 = 1$*, and from* $y_1 = 2$*, we have that* $x_4 = 1$ *and* $x_2 = 0$*. Finally from looking up the mapping created we know that* x'_1 *and* x'_2 *represent the variables* x_3 *and* x_5*, respectively, meaning that the assignment of these variables is* $x_3 = 0$ *and* $x_5 = 1$*.*

The procedure to transform the model returned from the SAT solver into an individual is similar to the previous one. All the variables are iterated and depending on the type of variable there are three different approaches:

- If the variable is one of the variables removed by the UP, it is ignored;
- If the variable is not being exploited by either UP or SI, then we must find its position in the individual. Once again, a mapping similar to the one created before is used.

- Finally, there is the encoding of the n_{si} variables that represent the constraints exploited by SI. We analyse all variables set to 1 that belong to a constraint exploited by SI. By knowing which constraint it belongs to, we can find the value of the exploited constraint in the mixed individual.

Example 18. *Let $F = \{(x_1 = 0), (x_2 + x_3 + x_4 = 1), (x_5 + x_6 \geq 1)\}$ be a set of PB constraints. For this problem the UP would set x_1 to be 0, and the SI would encode the second constraint as $y_1 \in \{1, 2, 3\}$. Let $\alpha = \{(x_1, 0), (x_2, 0), (x_3, 1), (x_4, 0), (x_5, 0), (x_6, 1)\}$ be an assignment found by a PB solver that we want to translate to an individual. x_1 can be ignored, since the variables forced by UP are not encoded into the individual. x_5 and x_6 are not affected by either UP or SI, therefore we just need to find their positions in the individual, which are x'_1 and x'_2 respectively. x_2 and x_4 are exploited by the SI, however, since they are assigned as 0, we can ignore them. x_3 is affected by the SI and is set to 1, so we search all constraints exploited by SI. A match is found in the second variable of the second constraint, which is encoded as y_1, so y_1 is set to 2. The assignment in the individual would then be $\{(x'_1, 0), (x'_2, 1), (y_1, 2)\}$.*

Chapter 5

Experimental Results

This chapter is split into two parts. First, section 5.1 provides a small description of the types of problems considered in the experimental analysis. Then, section 5.2 presents the experimental results for the different algorithms and variations for each benchmark set.

5.1 Benchmarks

In this section we briefly explain the problems that were used to test our software. First, we start by introducing the Set Covering Problem in section 5.1.1. Then, we introduce the Virtual Machine Consolidation problem in section 5.1.2. Next, in section 5.1.3, we describe the Development Assurance Level Allocation problem. Finally, in section 5.1.4, we define the Flying Tourist Problem.

5.1.1 Set Covering Problem

The Set Covering Problem (SCP) [7] is a model for many important applications, such as airline crew scheduling [14] and mass transit scheduling [27]. SCP can be either single or multi-objective, however, we only consider multi-objective in this work.

The SCP can be defined as follows. Consider m items, $n \leq 2^m - 1$ subsets of said items and p objectives for each subset. The goal of SCP is to determine which of the n subsets to select such that every m items must be included in at least one subset, while minimizing the p individual objective functions of each subset.

5.1.2 Virtual Machine Consolidation

Virtualization gives us the possibility to consolidate virtual machine instances running on underutilized computers into fewer host, which enables the computers to be turned off, saving energy. It is important in both data centers [20] and cloud computing [1].

[1] http://aws.amazon.com/ec2/

The goal of the Virtual Machine Consolidation (VMC) [15] problem is to place each virtual machine into a host. Each host has a maximum capacity corresponding to a specific resource, such as the CPU usage, memory usage or disk bandwidth. Similarly, each VM has a set of demands for each resource. The goal of this problem is to place all the virtual machines into as few hosts as possible while ensuring that the capacity of each host is not surpassed by the demand of the virtual machines, taking into account that the wastage of resources from each host and the costs of migrating a VM from one machine to another need to be minimized.

5.1.3 Development Assurance Level Allocation

The Development Assurance Level Allocation (DALA) [3] problem is used to find the level of rigor of the development of a software/hardware function of an aircraft. It guides the assurance activities that should be applied at each stage of development in order to eliminate the design and coding errors that impact the safety of the aircraft. The usage of DALA is related to development faults and it is applied not only in aeronautics, but also in other safety-critical domains such as space, nuclear, railway or automotive industries [2].

In the aeronautics industry, a level ranging from A to E is allocated to functions, software and hardware items. A higher value requires more assurance activities. A higher DALA level increases the development cost of software and hardware items. So the designers aim at allocating a DALA level to software and hardware as low as possible, while respecting the bounds imposed by the safety regulations.

5.1.4 Flying Tourist Problem

The Flying Tourist Problem (FTP) [17] consists of finding the best possible route, set of flights and schedule for a multi-city flight request. It has a large number of applications for routing problems that occur on a network and can be formulated as a graph [1].

The FTP is similar to another problem called Traveling Salesman Problem [16], where a person starting in a given city must visit $N - 1$ other different cities in the most efficient way. FTP is a specific case applied to commercial flights with some variations, such as cost changes depending on the time and direction of the trip and the fact that some cities must be visited within a specific time window.

This formulation leads to a much more realistic problem where the goal is to find the best route, schedule and set of flights for the trip, while also reflecting on the total cost and flight duration of the trip.

5.2 Analysis

In this section we analyse the performance of the smart operators and the structure improvements implemented on the set of benchmarks proposed. First, in section 5.2.1, we show the results on the SCP problem. Then, we analyse the results on the VMC problem in section 5.2.2. Next, in section 5.2.3, we analyse the DALA's results. Finally at section 5.2.4, we analyse the FTP.

All results were obtained on a dual socket Intel® Xeon® CPU E5-2630 v2 @ 2.60GHz, with a total of 12 cores and 24 threads, and 64GB of RAM. As for the instances, these are in OPB format [2].

Both MOEA/D and NSGA-II were tested with a population size of 100 and a timeout of 1800 seconds. The timeout of 1800 seconds was also used with the MCS Enumeration (MCSE) algorithm. The single point mutation and uniform crossover operators were tested with probabilities 0.05 and 0.8 respectively. As for the smart operators they were used with probability 0.01 and a conflict budget of 50000 for both smart mutation and smart improvement. To avoid confusion in the results, we denote the hybrid NSGA-II as H_NSGA-II and the hybrid MOEA/D as H_MOEA/D. If the algorithm has the structure improvements technique enabled, it is defined as NSGA-II_SI or MOEA/D_SI.

We evaluated the quality of the Pareto front approximations using the Inverted Generational Distance (IGD) [5] and Hypervolume [33] performance metrics, both presented in section 2.5. The IGD is a combined metric that measures the average distance from the cost vectors in a reference front $O(A_R)$ to the closest cost vectors in $O(A)$. Smaller values of IGD show that the population has solutions with higher quality. Hypervolume is another combined metric that measures the volume of the cost space dominated by population A up to a given reference point $Z = (z_1, \ldots, z_k)$, therefore, higher values are preferred.

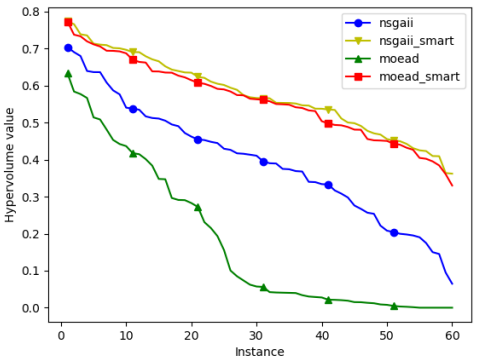
5.2.1 Set Covering Problem

Figure 5.1 shows the results of running both NSGA-II and MOEA/D with and without smart operators. Note that we do not show the results of running these algorithms with structure improvements as it would not change the results, since our tests for this problem do not have any constraints that can be exploited by the structure improvements technique.

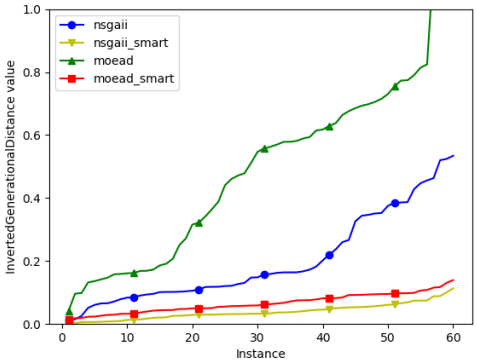
From figure 5.1a we can see that the addition of smart operators to both MOEA/D and NSGA-II greatly increases the performance of the algorithms, where both of them have an hypervolume that varies uniformly between around 0.8 to 0.4. The hypervolume of the stochastic algorithm NSGA-II goes from 0.7 to around 0.1 while MOEA/D starts a little over 0.6 and ends at 0, having a much faster decline in value than NSGA-II, and therefore having the worst performance. Similar results arise from figure 5.1b, where H_MOEA/D and H_NSGA-II achieve IGD values lower than 0.2 for all instances while MOEA/D has a fast increase of the IGD value, surpassing 1 for some instances, and NSGA-II almost reaches 0.6, although it has values lower than 0.2 in forty instances.

According to the results from figure 5.1 both MOEA/D and NSGA-II did not achieve great results while solving SCP, which should not be the case since stochastic algorithms typically perform better on loosely constrained problems [6], which is the case of SCP. Figures 5.2 and 5.3 compare the previous results from 5.1 with another execution of the algorithms but using uniform mutation instead of single point mutation. Both MOEA/D and NSGA-II with uniform mutation show very similar performance to the previous run of the algorithm with the smart operator. However, the addition of uniform mutation to the hybrid algorithms does not change their performance.

[2] http://www.cril.univ-artois.fr/PB16/format.pdf

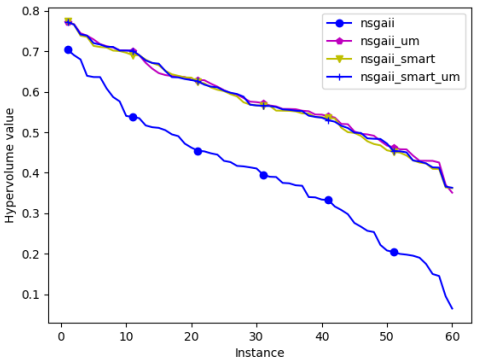


(a) Hypervolume of the stochastic algorithms.

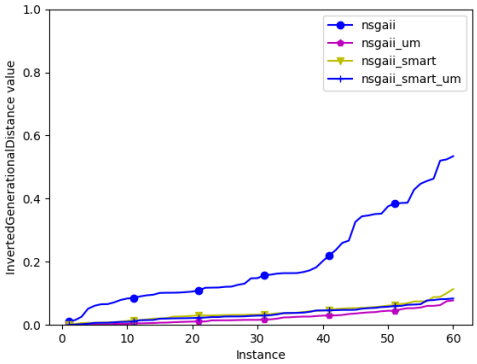


(b) IGD of the stochastic algorithms.

Figure 5.1: Performance of stochastic algorithms in SCP.



(a) Hypervolume of the smart operators in NSGA-II.

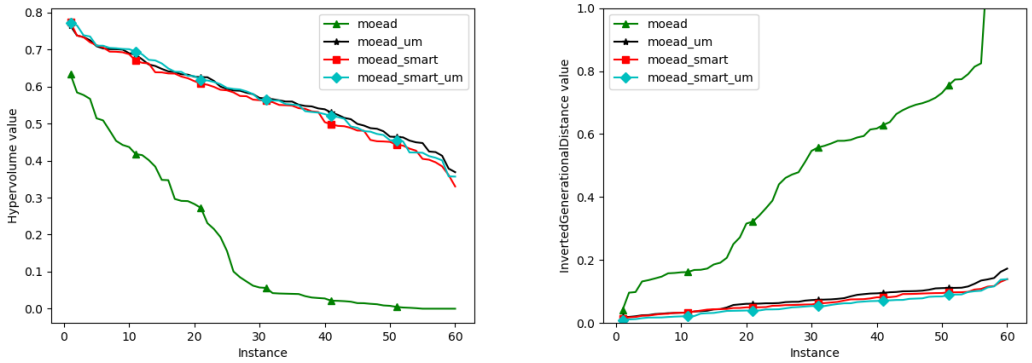


(b) IGD of the smart operators in NSGA-II.

Figure 5.2: Comparison of the performance of uniform mutation and single point mutation in NSGA-II.

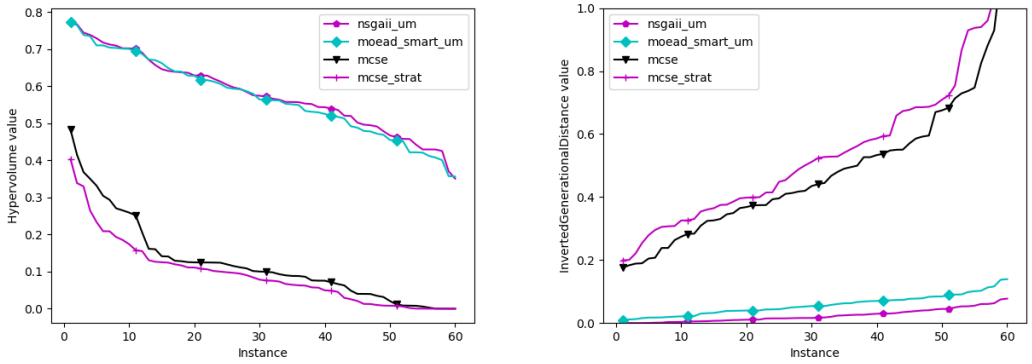
The fact that using uniform mutation improves the performance is to be expected, since the uniform mutation operator tries to mutate approximately five variable assignment in each the individual (since the mutation rate used is 0.05), while single point mutation only has 0.05 probability to mutate one variable assignment per individual. This means that single point mutation, while being faster than uniform mutation, returns individuals very similar to the original ones. On the other hand, uniform mutation performs a more aggressive search through the feasible space, finding a more diverse Pareto front approximation.

In figure 5.4 we compare the best results from MOEA/D (H_MOEA/D with uniform mutation) and NSGA-II (NSGA-II with uniform mutation) with the constraint based algorithm MCSE with and without stratification. By analysing figure 5.4a we can see that MCSE obtains to worst performance in hypervolume to this point, rapidly reaching values close to 0.15 and even degrading to 0 for some instances. Figure 5.4b shows similar results, with IGD values starting on 0.2 and surpassing 1 for some instances. These results come from the fact that constraint based algorithms do not achieve good results in problems with big feasible areas, since they exploit the structure of the constraints in order to prune big portions of the search space, and with problems loosely constrained there is not much to be exploited.



(a) Hypervolume of the smart operators in MOEA/D. (b) IGD of the smart operators in MOEA/D.

Figure 5.3: Comparison of the performance of uniform mutation and single point mutation in MOEA/D.



(a) Hypervolume of the MCSE algorithm. (b) IGD of the MCSE algorithm.

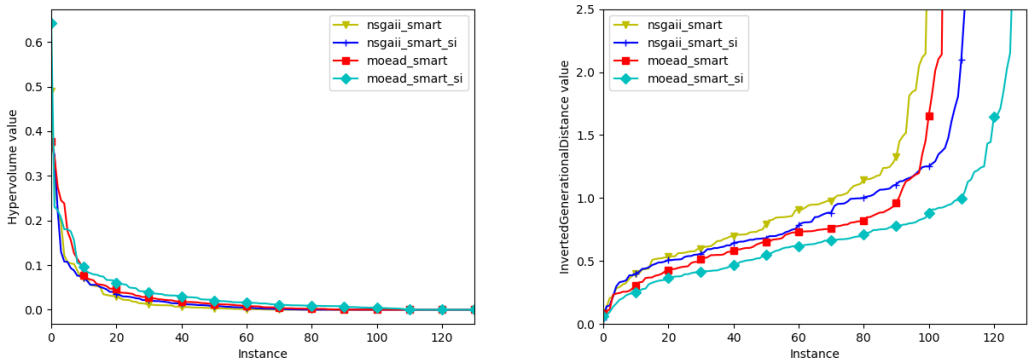
Figure 5.4: Comparison of the performance between stochastic and constraint based algorithms.

5.2.2 Virtual Machine Consolidation

In this section three types of the VMC problem are analysed. In section 5.2.2.1 the results of the full VMC problem with all the constraints and objectives are analysed. Then, in section 5.2.2.2, the results of a simpler version of VMC where there are no machines initially allocated are shown. Finally, in section 5.2.2.3, we present the results for another simpler version of VMC where the resource wastage objective is not considered.

5.2.2.1 Full VMC

The VMC problem is the hardest problem out of all the problems that are being tested here, therefore it is expected that stochastic algorithms cannot obtain results for any instance. We start by showing the results of the hybrid algorithms in figure 5.5. Figure 5.5a shows the hypervolume values of the hybrid algorithms. All the hybrid algorithms show similar performance, with the H_MOEA/D_SI being slightly better than the others, starting with an hypervolume of 0.62 and reaching 0 close to a hundred and ten instances, while the other algorithms reach this point in around ninety instances. Regarding the IGD values shown in figure 5.5b it is easier to see the difference in the performance of the algorithms,



(a) Hypervolume of the hybrid algorithms.

(b) IGD of the hybrid algorithms.

Figure 5.5: Performance of hybrid algorithms in VMC problems with and without structure improvements.

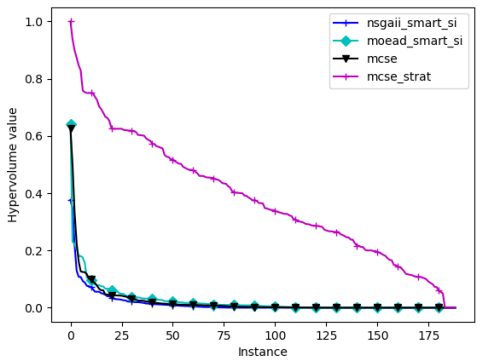
with H_MOEA/D_SI having the best performance. H_MOEA/D has the second best performance on almost one hundred instances and in the others it is surpassed by H_NSGA-II_SI. H_NSGA-II obtains the worst IGD value out of these four algorithms. This shows that structure improvements increase the performance of the algorithms, which is to be expected since the VMC instances have a lot of constraints that can be exploited by the structure improvements.

Now we compare the performance of the H_MOEA/D_SI and H_NSGA-II_SI algorithms with that of the MCSE algorithm on figure 5.6. On figure 5.6a we can see that the hypervolume of MCSE starts at 0.62 with a fast decline, showing similar performance to H_MOEA/D_SI. However, stratified MCSE shows a better performance, obtaining an hypervolume of 0.6 or more for thirty instances while the other algorithms are close to 0.3. This algorithm then shows an uniform decline, only having an hypervolume of 0 in five instances out of almost one hundred and ninety. As for the IGD value, as is shown in figure5.6b, MCSE has a similar value to the H_MOEA/D_SI, having a lower performance in one hundred instances, where it reaches a value of 0.8 at the same point as H_MOEA/D_SI, and then achieves a better performance than than this algorithm. As for the stratified MCSE algorithm, it shows an IGD value of 0 on seventy instances, then increasing and reaching a value of 1 after one hundred and eighty instances. For this type of problem the H_MOEA/D_SI can achieve performance close to MCSE, however, stratified MCSE shows the best performance out of all the algorithms.

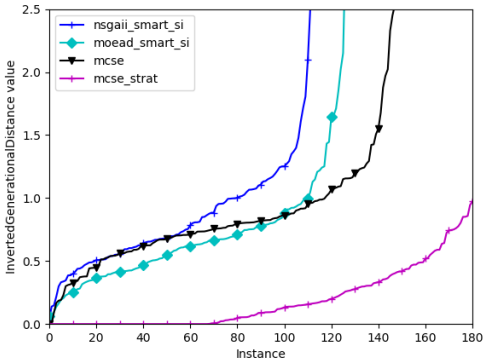
5.2.2.2 VMC without mapping

Now we analyse the results of the first simplified version of VMC, where there are no virtual machines initially allocated. Even if this is a simplified version of the original VMC problem, the stochastic algorithms still can not find any solutions. This is due to the Boolean encoding, which hinders the performance of the stochastic algorithms and is the reason why structure improvements were proposed in the first place.

Figure 5.7 shows the IGD and hypervolume values obtained with H_NSGA-II and H_MOEA/D. On figure 5.7a the hypervolume is shown. The H_MOEA/D_SI shows the best performance, starting with an hypervolume of 0.7 and obtaining 0 for twenty two instances, while the other algorithms reached a value of 0 at around thirteen, as it happens with the normal H_MOEA/D and H_NSGA-II_SI, or even less as

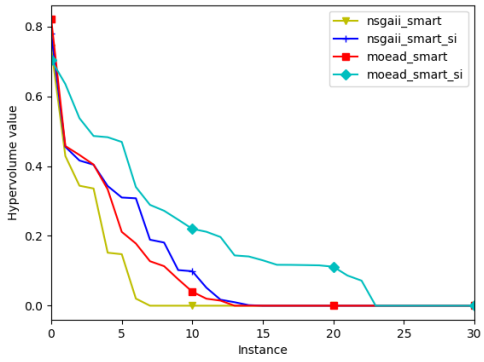


(a) Hypervolume of the MCSE algorithm.

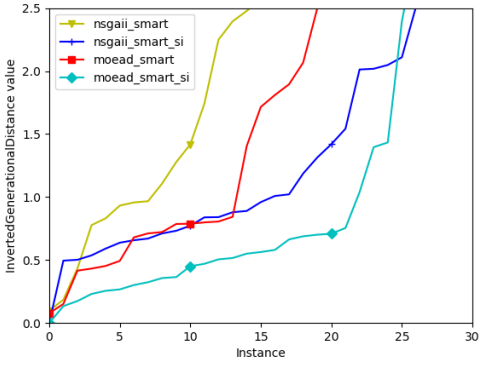


(b) IGD of the MCSE algorithm.

Figure 5.6: Comparison of the performance between hybrid and constraint based algorithms for VMC.



(a) Hypervolume of the hybrid algorithms.

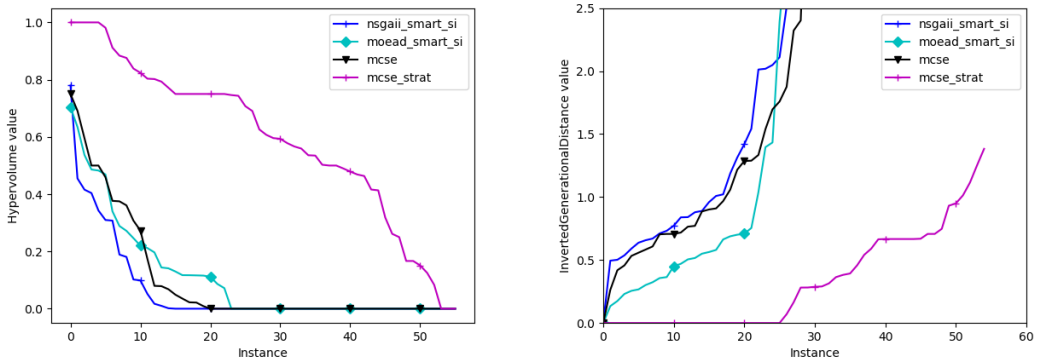


(b) IGD of the hybrid algorithms.

Figure 5.7: Performance of hybrid algorithms in VMC problems without VMs initially allocated.

happens with H_NSGA-II. For the IGD value we can see on figure 5.8b that H_MOEA/D_SI has a value of 0.5 after fifteen instances while all the other algorithms reach this point in one to five instances. After the twelfth instance point the H_NSGA-II_SI differentiates a lot from the other two algorithms, getting closer to the performance of MOEA/D using the same techniques.

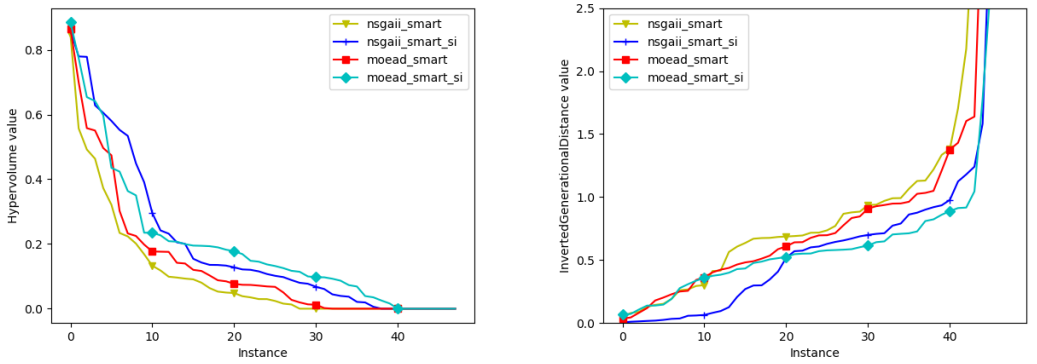
On figure 5.8 we compare MCSE with the hybrid algorithms both using the structure improvements technique. Figure 5.8a shows the hypervolume values for all algorithms. The MCSE algorithm shows a similar performance to H_MOEA/D_SI, reaching an hypervolume value of 0 in twenty instances, while stratified MCSE shows the best performance, starting with an hypervolume of 1 and only reaching 0 after fifty three instances. With the IGD values the performances are similar, as it is shown on figure 5.8b, where stratified MCSE obtains an IGD value of 0 in twenty five instances, reaching a value of 1 at the fiftieth instance. As for the normal MCSE it's performance is similar to H_NSGA-II_SI on the twenty best instances, then surpassing it and also H_MOEA/D_SI after twenty four instances at the point where the IGD value is 1.7.



(a) Hypervolume of the MCSE algorithm.

(b) IGD of the MCSE algorithm.

Figure 5.8: Comparison of the performance between hybrid and constraint based algorithms for VMC without VMs initially allocated.



(a) Hypervolume of the hybrid algorithms.

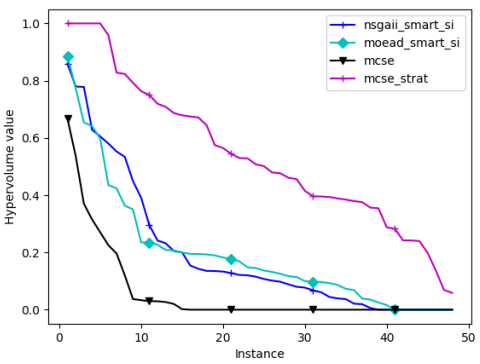
(b) IGD of the hybrid algorithms.

Figure 5.9: Performance of hybrid algorithms in VMC problems without wastage constraints with and without structure improvements.

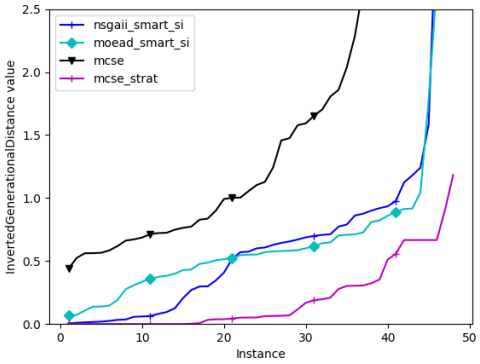
5.2.2.3 VMC without wastage

In this section we are analysing the results of Neon on a simplified version of VMC without resource wastage. Once again stochastic algorithms cannot find any feasible individuals for this problems, since this simplified version is still too constrained for this type of algorithms.

The performance of the stochastic algorithms with the application of smart operators is presented in figure 5.9. In this problem the addition of the structure improvement technique helps both hybrid algorithms achieve a better performance, as we can see on figure 5.9a where the H_NSGA-II_SI shows the best performance until the thirteenth instance point and reaches an hypervolume of 0 after thirty eight instances, while H_MOEA/D_SI has the best performance from the thirteenth instance point onwards, until it reaches 0 after forty instances. Both algorithms without structure improvements show similar performances, with H_MOEA/D being slightly better, and both reaching 0 in around thirty instances. Figure 5.9b shows similar results, where both algorithms achieve a better performance if the structure improvements technique is used. In this case H_NSGA-II_SI shows better performance until the twentieth instance mark and in the rest the H_MOEA/D_SI is ahead.



(a) Hypervolume of the MCSE algorithm.



(b) IGD of the MCSE algorithm.

Figure 5.10: Comparison of the performance between hybrid and constraint based algorithms for VMC without wastage constraints.

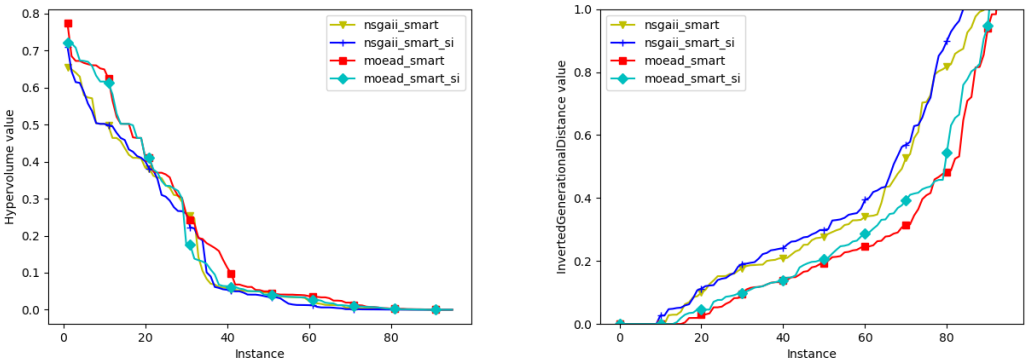
Next, on figure 5.10 we compare the performance of the MCSE algorithm with both hybrid algorithms using structure improvements. On figure 5.10a we can see that MCSE shows very poor performance, with an hypervolume value of 0 on almost thirty five instances. However, the stratified MCSE algorithm is able to obtain the best performance out of all the algorithms by a good margin, never reaching an hypervolume value of 0. As for the IGD values which can be seen on figure 5.10b, once again MCSE shows the worst performance, with IGD values over 2.5 on more than ten instances, while stratified MCSE achieves the best values, with an IGD of 0 on almost seventeen instances and at most 1 for the remaining ones.

We can conclude that for these instances the structure improvements technique is able to help the hybrid algorithms achieve better performance. All the hybrid algorithms perform better than the constraint based algorithm MCSE, however stratified MCSE outperforms all the other algorithms.

5.2.3 Development Assurance Level Allocation

For the DALA problem no stochastic algorithm with or without the structure improvement technique was able to find solutions for any instance, therefore there is no figure showing the performance of these algorithms, just as it happened in VMC.

On figure 5.11 we can see the results of running the stochastic algorithms with the aid of smart operators on the DALA instances. Figure 5.11a shows the hypervolume obtained by these hybrid algorithms. As we can see, both variants of hybrid MOEA/D are able to achieve a slightly better performance than both hybrid NSGA-II on the first twenty instances, however after that both algorithms show very similar results with no significant difference. Both algorithms start with an hypervolume of around 0.7 and reach 0.1 at the forty instances point, getting very low values on the rest of the instances. Note that the addition of structure improvements does not significantly affect the performance of the algorithms. As for the IGD values, figure 5.11b shows the results. MOEA/D outperforms NSGA-II throughout all the instances. NSGA-II reaches an IGD value of around 0.4 by the sixtieth instance, at which point it suddenly shows a bigger increase in the value, surpassing 1 for the worst instances, while MOEA/D only reaches an IGD



(a) Hypervolume of the hybrid algorithms.

(b) IGD of the hybrid algorithms.

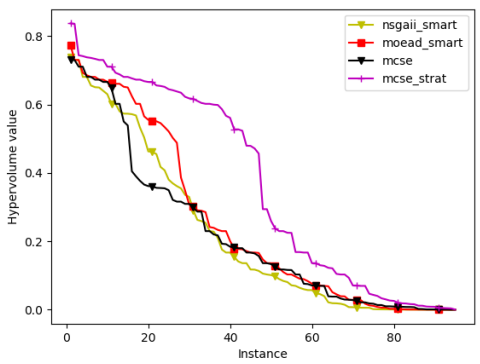
Figure 5.11: Performance of hybrid algorithms in DALA problems with and without structure improvements.

value of 0.4 around ten instances after NSGA-II, showing then a bigger increase in the IGD value as well. The addition of the structure improvements technique does not seem to improve the performance of the hybrid algorithms, as seen of figure 5.11b. This happens because in these instances only a small numbers of constraints with two variables each are eligible to be converted by the structure improvements technique. The fact that all these constraints only have two variables means that structure improvements only have a small effect on both the stochastic algorithm and the assumptions on the smart operators. On the other hand, enabling the structure improvements technique on the hybrid algorithm incurs additional overhead due to the conversion between Boolean assignments and individuals, which leads the algorithm to do less iterations, explaining the small variation in the performance.

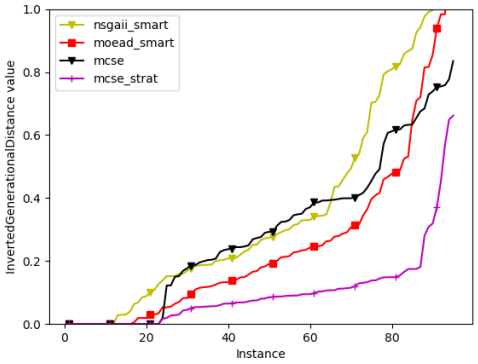
The comparison of the MCSE algorithm with the best among the hybrid algorithms is shown in figure 5.12. On figure 5.12a we see that stratified MCSE achieves the best distribution, only obtaining an hypervolume value similar to the others after the eightieth instance. All the other algorithms achieve similar performances, with H_MOEA/D showing a slightly better performance until the thirtieth instance mark while MCSE shows the worst performance, which can be seen from the tenth to the thirtieth instance. The IGD values from figure 5.12b show once again that stratified MCSE achieves the best performance on all instances, obtaining an IGD value of 0 on twenty instances and only surpassing 0.2 after the eighty fifth instance. MCSE without stratification obtains an IGD value of 0 on twenty instances and then, from instance thirty to sixty five, it shows the worst performance. However by the end, the performance of MOEA/D and NSGA-II greatly decreases and MCSE surpasses both of them, NSGA-II at the sixty fifth point and MOEA/D at the eighty fifth.

5.2.4 Flying Tourist Problem

The FTP problem is not as hard and constrained as the DALA or VMC problems, and therefore stochastic algorithms are able to find solutions for some of the instances, as can be seen in figure 5.13. NSGA-II and MOEA/D can only find feasible individuals on four and one instances, respectively, as we can see in figure 5.13a. The addition of structure improvements to the stochastic algorithms helps improve this

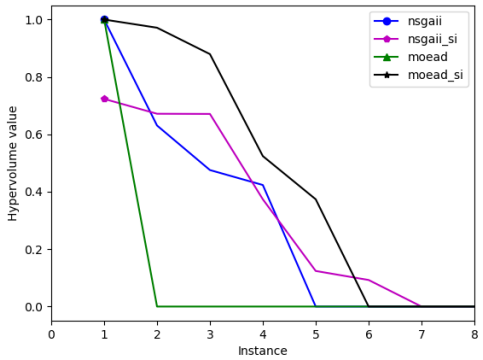


(a) Hypervolume of the MCSE algorithm.

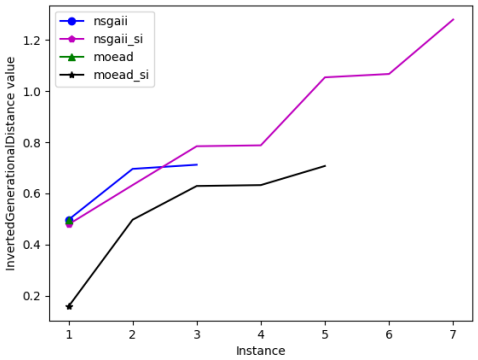


(b) IGD of the MCSE algorithm.

Figure 5.12: Comparison of the performance between hybrid and constraint based algorithms for DALA.



(a) Hypervolume of the stochastic algorithms.



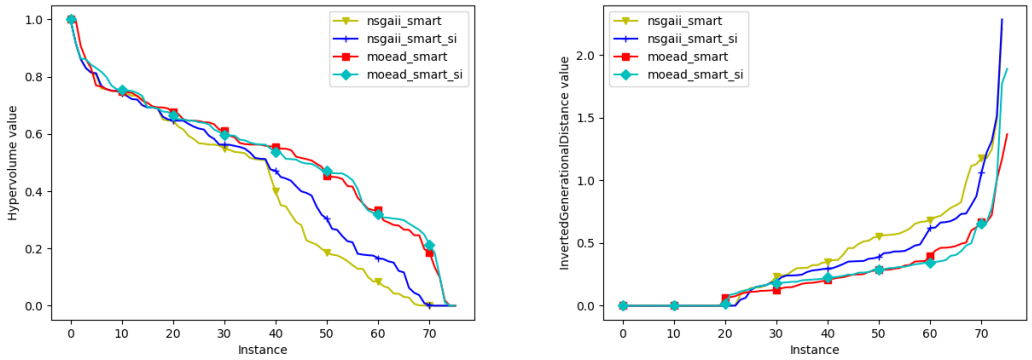
(b) IGD of the stochastic algorithms.

Figure 5.13: Performance of stochastic algorithms in FTP with and without structure improvements.

number by three for NSGA-II and four for MOEA/D, meaning that the algorithm with best performance is the NSGA-II with structure improvements, finding feasible individuals in seven out of the eighty instances, which is still a very poor performance.

The performance of the hybrid algorithms is presented in figure 5.14. Since stochastic algorithms only find feasible individuals for a small number of instances, we do not compare their results with the results from the hybrid algorithms. These algorithms obtain much better results, being able to find solutions in almost all instances. From figure 5.14a we see that all algorithms start with an hypervolume of 1 having similar values in forty instances and diverging after this point. After fifty instances both hybrid MOEA/D algorithms achieve an hypervolume of 0.5 while H_NSGA-II_SI shows an hypervolume of 0.3 and H_NSGA-II a value of 0.2. As for the IGD value, figure 5.14b shows that all these algorithms are able to get an IGD value of 0 for twenty instances. The hybrid MOEA/D results then reach a value of 0.5 at the sixty fifth instance while both hybrid NSGA-II algorithms reaches a value of around 0.7.

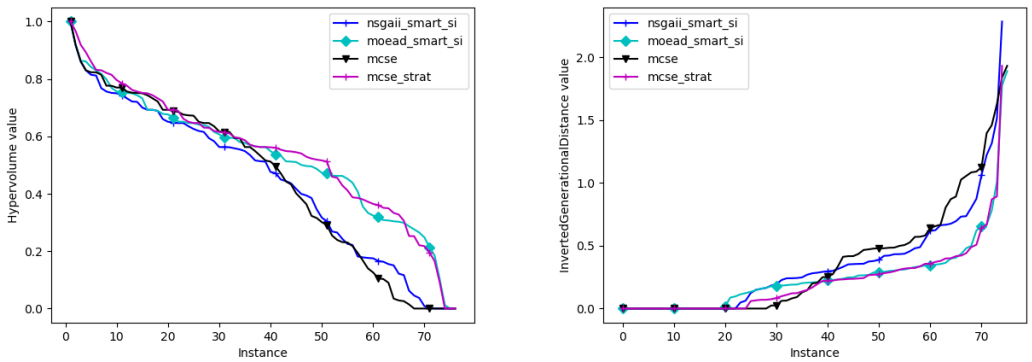
Finally for the FTP instances, figure 5.15 shows the comparison of the MCSE algorithm with the hybrid algorithms MOEA/D and NSGA-II both with structure improvement technique enabled. From figure 5.15a we can see that the MCSE algorithm achieves a performance very similar to H_NSGA-II_SI. MCSE with stratification enabled is able to obtain better results, having a similar hypervolume to



(a) Hypervolume of the hybrid algorithms.

(b) IGD of the hybrid algorithms.

Figure 5.14: Performance of hybrid algorithms in FTP with and without structure improvements.



(a) Hypervolume of the MCSE algorithm.

(b) IGD of the MCSE algorithm.

Figure 5.15: Comparison of the performance between hybrid and constraint based algorithms for FTP.

H_MOEA/D_SI throughout all instances. On figure 5.15b we see that the MCSE algorithm achieves an IGD value of 0 for almost thirty instances, but shortly after reaches the highest IGD value, showing the worst performance on half of the instances. As for the stratified MCSE, it achieves better performance, according to IGD, than H_MOEA/D_SI between the twentieth and fortieth point and then they both show identical IGD values for the rest of the instances.

Chapter 6

Conclusions and Future Work

In this thesis we explored algorithms for Multi-Objective Combinatorial Optimization problems. We introduced Pseudo-Boolean Optimization problems in order to explain what is Multi-Objective Combinatorial Optimization and then showed different state-of-art algorithms used to find the Pareto-front of the problem. We also proposed Neon, which uses hybrid algorithms to solve generic MOCO problems, and a technique called structure improvements.

To evaluate the results of the algorithms, the performance metrics hypervolume and inverted generational distance were used. The algorithms were tested using instances of Set Covering Problem, Flying Tourist, Development Assurance Level Allocation and Virtual Machine Consolidation. For problems very low constrained such as the Set Covering Problem, hybrid algorithms were able to obtain results as good as the stochastic algorithms, which have the best performance. For most of the remaining problems, the hybrid algorithms were able to obtain similar or better performance than the constraint based algorithm MCSE, however obtaining worse performance than stratified MCSE. The structure improvements technique increases the performance of hybrid algorithms for most cases, failing to improve the performance when testing instances of Set Covering Problem or Development Assurance Level Allocation, since these problems are not ideal for structure improvements to be applied.

Stratification greatly increases the performance of the MCSE algorithm to the point where no other algorithm performs similarly. As future work, an interesting idea would be to implement stratification in the smart mutation operator in order to improve the hybrid algorithms in Neon.

The structure improvements technique is effective in improving the performance of the hybrid algorithms MOEA/D and NSGA-II, so we could also improve this technique by extending the exploit of at-most-1 and exactly-1 constraints to at-most-k and exactly-k constraints. We could also improve it to new types of constraints, for example, constraints similar to $x_1 - x_2 \geq 0$ appeared many times in instances of Development Assurance Level Allocation where structure improvements was shown not to be effective in the current state, so extending it to exploit similar constraints could prove to be effective in solving these instances.

Bibliography

[1] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[2] P. Baufreton, J.-P. Blanquart, J. Boulanger, H. Delseny, J. Derrien, J. Gassino, G. Ladier, E. Ledinot, M. Leeman, P. Quéré, et al. Multi-domain comparison of safety standards. 2010.

[3] P. Bieber, R. Delmas, and C. Seguin. Dalculus–theory and tool for development assurance level allocation. In *International Conference on Computer Safety, Reliability, and Security*, pages 43–56. Springer, 2011.

[4] E. Birnbaum and E. L. Lozinskii. Consistent subsets of inconsistent systems: structure and behaviour. *Journal of Experimental & Theoretical Artificial Intelligence*, 15(1):25–46, 2003.

[5] C. A. C. 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. Springer, 2004.

[6] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[7] H. Farhangi, D. Konur, and C. H. Dagli. A separation method for solving multiobjective set covering problem. 2020.

[8] C. M. Fonseca and P. J. Fleming. Multiobjective optimization and multiple constraint handling with evolutionary algorithms. ii. application example. *IEEE Transactions on systems, man, and cybernetics-Part A: Systems and humans*, 28(1):38–47, 1998.

[9] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of Computer and System Sciences*, 79(8): 1230–1242, 2013.

[10] M. Gavanelli et al. An algorithm for multi-criteria optimization in csps. In *ECAI*, volume 2, pages 136–140, 2002.

[11] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. L. Traon. Multi-objective test generation for software product lines. In *Proceedings of the 17th International Software Product Line Conference*, pages 62–71. ACM, 2013.

[12] C. Henard, M. Papadakis, M. Harman, and Y. Le Traon. Combining multi-objective search and constraint solving for configuring large software product lines. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 517–528. IEEE Press, 2015.

[13] D. Jackson, H. Estler, D. Rayside, et al. The guided improvement algorithm for exact, general-purpose, many-objective combinatorial optimization. 2009.

[14] A. Jaszkiewicz. A comparative study of multiple-objective metaheuristics on the bi-objective set covering problem and the pareto memetic algorithm. *Annals of Operations Research*, 131(1-4): 135–158, 2004.

[15] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubramanian, K. Talwar, L. Uyeda, and U. Wieder. Validating heuristics for virtual machines consolidation. *Microsoft Research, MSR-TR-2011-9*, pages 1–14, 2011.

[16] J. D. Little, K. G. Murty, D. W. Sweeney, and C. Karel. An algorithm for the traveling salesman problem. *Operations research*, 11(6):972–989, 1963.

[17] R. Marques, L. Russo, and N. Roma. Flying tourist problem: Flight time and cost minimization in complex routes. *Expert Systems with Applications*, 130:172–187, 2019.

[18] J. Marques-Silva, F. Heras, M. Janota, A. Previti, and A. Belov. On computing minimal correction subsets. In F. Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 615–622. IJCAI/AAAI, 2013. ISBN 978-1-57735-633-2. URL `http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6922`.

[19] R. Olaechea, D. Rayside, J. Guo, and K. Czarnecki. Comparison of exact and approximate multi-objective optimization for software product lines. In *Proceedings of the 18th International Software Product Line Conference-Volume 1*, pages 92–101. ACM, 2014.

[20] M. Rosenblum and T. Garfinkel. Virtual machine monitors: Current technology and future trends. *Computer*, 38(5):39–47, 2005.

[21] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar. Scalable product line configuration: A straw to break the camel's back. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 465–474. IEEE, 2013.

[22] 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.

[23] N. Srinivas and K. Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248, 1994.

[24] 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. Springer, 2017.

[25] M. Terra-Neves, I. Lynce, and V. M. Manquinho. Stratification for constraint-based multi-objective combinatorial optimization. In *IJCAI*, pages 1376–1382, 2018.

[26] M. Terra-Neves, I. Lynce, and V. Manquinho. Integrating pseudo-boolean constraint reasoning in multi-objective evolutionary algorithms. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 1184–1190. AAAI Press, 2019.

[27] M. Upmanyu and R. R. Saxena. Linearization approach to multi objective set covering problem with imprecise nonlinear fractional objectives. *Opsearch*, 52(3):597–615, 2015.

[28] Y. Xiang, Y. Zhou, M. Li, and Z. Chen. A vector angle-based evolutionary algorithm for unconstrained many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 21(1):131–152, 2016.

[29] 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 (TOSEM)*, 26(4):14, 2018.

[30] Y. Yuan and W. Banzhaf. Arja: Automated repair of java programs via multi-objective genetic programming. *IEEE Transactions on Software Engineering*, 2018.

[31] R. Zabih and D. A. McAllester. A rearrangement search strategy for determining propositional satisfiability. In *AAAI*, volume 88, pages 155–160, 1988.

[32] Q. Zhang and H. Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6):712–731, 2007.

[33] E. Zitzler. *Evolutionary algorithms for multiobjective optimization: Methods and applications*, volume 63. Citeseer, 1999.

[34] E. Zitzler and S. Künzli. Indicator-based selection in multiobjective search. In *International Conference on Parallel Problem Solving from Nature*, pages 832–842. Springer, 2004.