# Stochastic Multi-objective Combinatorial Optimization Algorithms

**Miguel António dos Santos Machado Tavares**
Instituto Superior Técnico, Universidade de Lisboa, Portugal

## ABSTRACT

Multi-Objective Combinatorial Optimization (MOCO) has several real-world applications such as Virtual Machine Consolidation, Configuration of Software Product Lines, among others. Recently, 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.

In this work, we present Neon, a generic stochastic approach for solving MOCO problem instances formulated as Multi-Objective Boolean formulas. Moreover, Neon incorporates Logic-based techniques that would allow a faster convergence of the stochastic algorithm.

## INTRODUCTION

### Motivation

With single-objective problems, the goal is to find the optimal assignment that minimizes the cost function, that in this case, is a single point. With multi-objective problems, there are various cost functions that we want to minimize, so the number of optimal assignments can grow exponentially. There are many real-world problems that can be translated into multi-objective problems [3, 10, 4]. At the moment there are tools to find approximations of the set of optimal assignments, however, these tools are all focused on the specific real-world problems and to our knowledge, there is no tool to solve generic multi-objective problems. Our objective is to create a robust tool that can solve generic multi-objective problems.

There are many algorithms that have the purpose of solving multi-objective problems. These algorithms have been studied for many years, and are divided into two main categories: stochastic and constraint-based algorithms. Stochastic algorithms are good to find a good approximation of the set of optimal assignments when the problems are not highly constrained. Constraint-based algorithms can find the exact set with the best solutions, however, it is very time consuming.

Recently, hybrid algorithms have been gaining interest, using techniques from both stochastic and constraint-based algorithms. These algorithms can produce better results than stochastic or constraint-based algorithms on real-world problems. For this reason, our tool will use a hybrid algorithm.

### 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, 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.

### Organization of the Document

This document is structured as follows. First, some general concepts are presented in Section 2. Next, algorithms used to solve Multi-Objective Combinatorial Optimization problems are presented in Section 3. In Section 4 we propose a hybrid algorithm to solve generic MOCO instances. The evaluation of the results is present in Section 5. Finally, we conclude this work in Section 6.

## PRELIMINARIES

In this section, we will start by defining several terms and notations that will be used in the rest of the document.

### Multi-Objective Combinatorial 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 $c$ has the form:

$$\sum w_i \cdot l_i \bowtie k, \bowtie \in \{\leq, =, \geq\} \tag{1}$$

Let $f(X)$ be a PB expression known as cost function that has the form:

Table 1: Satisfiable assignments and their costs for the instance in Example 1

| $x_1$ | $x_2$ | $x_3$ | $f_1$ | $f_2$ |
|---|---|---|---|---|
| 1 | 1 | 1 | 3 | 3 |
| **0** | **1** | **1** | **1** | **3** |
| **1** | **0** | **1** | **2** | **2** |
| **1** | **1** | **0** | **3** | **1** |

$$\sum w_i \cdot l_i \tag{2}$$

For an assignment $\alpha : X \longrightarrow \{0,1\}$, $\alpha(c) = 1$ denotes a constraint $c$ is satisfied by $\alpha$ and $f(\alpha)$ denotes the value of $f(X)$ for the assignment $\alpha$.

Now let $F = \{c_1, \ldots, c_m\}$ a set of $m$ PB constraints and $O = \{f_1, \ldots, f_k\}$ a set of $k$ PB cost functions that we want to minimize. A Multi-Objective Combinatorial Optimization (MOCO) problem is defined as $M = (F, O)$.

Let $M = (F, O)$ be a MOCO instance and $\alpha, \alpha' : X \longrightarrow \{0,1\}$ be two distinct models of $F$. For an assignment $\alpha$, $O(\alpha)$ denotes the values of the cost functions in $O$ for the assignment $\alpha$. We say that $\alpha$ dominates $\alpha'$, 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 $\alpha$ such that $\alpha(F) = 1$ and there is no other complete assignment $\alpha'$ 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 1. *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 $\alpha_1$ is dominated for example by $\tilde{\alpha_2}$ since $O(\alpha_1) = (3,3)$ and $O(\tilde{\alpha_2}) = (1,3)$.*

### 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) [1] 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 $O(A)$. The IGD value of a set $A$ can be obtained using the following equation:

$$IGD(A) = \frac{\sqrt[n]{\sum_{\alpha \in A_R} \min_{\alpha' \in A}\{\sqrt{\sum_{i=1}^{n}(f_i(\alpha) - f_i(\alpha'))^2}\}}}{|A_R|} \tag{3}$$

The hypervolume [12] 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 $f_i$. Let $C_\alpha$ be the hypercube with corners $\alpha \in A$ and $Z$. The hypervolume value can be obtained using the following equation:

$$HV(A) = \texttt{Volume}(\bigcup_{\alpha \in A} C_\alpha) \tag{4}$$

## RELATED WORK

In this section, various algorithms to solve MOCO instances are presented. First, stochastic algorithms are presented in section 3.1. Next, constraint based algorithms are presented in section 3.2.

### 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

**Algorithm 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
```

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.

#### 3.1.1 NSGA-II Algorithm

The NSGA-II algorithm [2] 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 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 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.

#### 3.1.2 MOEA/D Algorithm

MOEA/D [11] 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 2:** MOEA/D Algorithm

**Input:** ($F, O$, $stoppingcriteria$, $N$, $T$)
**Output:** $P$

```
1  P ← ∅
2  {α_1, ..., α_N} ← RandomPopulation(F, O, N)
3  z ← ReferencePoint(F, O)
4  {λ^1, ..., λ^N} ← WeightVectors(F, O, N)
5  for i = 1 to N do
6      B_i ← ComputeNeighbours(λ^i, {λ^1, ..., λ^N} \ λ^i, T)
7  while ¬stoppingcriteria do
8      for i = 1 to N do
9          l, m ← RandomIndex(B_i)
10         α ← Reproduction(α_l, α_m)
11         α ← Improvement(α)
12         for j = 1 to k do
13             if f_j(α) < z_j then
14                 z_j ← f_j(α)
15         foreach j ∈ B_i do
16             if g(α|λ^j, z) ≤ g(α_j|λ^j, z) then
17                 α_j ← α
18         P ← UpdateNonDominated(P ∪ {α})
19 return 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:

$$\begin{aligned} &\text{minimize } g(\alpha|\lambda, z^*) = max_{1 \leq i \leq N}\{\lambda_i \cdot |f_i(\alpha) - z_i^*|\} \\ &\text{subject to } \alpha(F) = 1 \end{aligned} \tag{5}$$

For each Pareto optimal point $x^*$ there exists a weight vector $\lambda$ such that $x^*$ is the optimal solution of (5). 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 2. 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 $\lambda^i$ with every other $\lambda^j$ where $j \in \{1, \ldots, N\} \setminus \{i\}$, saving the index $j$ of the $T$ closest weight vectors to $\lambda^i$ 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 $\alpha$ (line 10). $\alpha$ then goes 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 $\alpha_i$ that are worse than $\alpha$, according to the Tchebycheff approach, get replaced by $\alpha$ in the respective sub-problem (lines 15 - 17). Finally, the approximation $P$ is updated, by removing all individuals in $P$ that are dominated by $\alpha$ and adding $\alpha$ if it is not dominated by any of the individuals in $P$ (line 18).

### 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.

#### 3.2.1 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.

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 $f_i = \omega_1 \cdot l_1 + \ldots + \omega_o \cdot l_o$, 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 2. *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 [7].

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 $W = (F_H, \cup_{F_S \in O_S} F_S)$ [7].

The pseudo-code for the Pareto-MCS enumeration algorithm [7] can be found in Algorithm 3. 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).

**Algorithm 3:** 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** $U_{pareto}$

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 [9]. 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 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.

$(P^i_1, \ldots, P^i_{k_i}) = \texttt{Partition}\,(L^{\neg}(F_{Si}))$

## 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 3.2.1 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 [8].

In this Section, 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].

[1] http://moeaframework.org/

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 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 3. *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 4. *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 5. *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 6. *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* $(x_6 >= 1)$. *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* $\alpha$.

### 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 [6, 5, 8]. 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. 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 7. *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*

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

**Algorithm 4:** 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** $\neg IsSat$ & $A \neq \emptyset$ **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** $FixedIndividual$

**Algorithm 5:** Smart improvement algorithm

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

1 $A \leftarrow$ $\texttt{GetImprovementAssumptions}(\cup_{x \in OriginalIndividual}\{(x)\})$
2 $\texttt{MCSOracle}(F \cup A, O)$
3 $ImprovedIndividual \leftarrow \texttt{GetModel}()$
4 **if** $ImprovedIndividual = \emptyset$ **then**
5 $\quad IsUnsat \leftarrow \texttt{Solve}(A)$
6 $\quad$ **if** $IsUnsat$ **then**
7 $\quad\quad \texttt{SetParetoTerminationCondition}()$
8 $\quad$ **return** $OriginalIndividual$
9 $MCS \leftarrow \texttt{GetMCS}()$
10 $\texttt{AddBlockClause}(\cup_{x \in MCS}\{(x)\})$
11 **return** $ImprovedIndividual$

*$\alpha = \{(x_1,0),(x_2,1),(x_3,0),(x_4,0),(x_5,1)\}$ 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 8. *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 $\alpha$ 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 5. 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 9. *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 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 [9]. 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 10. *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 11. *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 $\alpha$ 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 12. *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 assignment 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 13. *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:

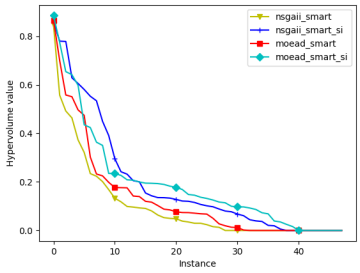
- If the variable is one of the variables removed by the UP, it is ignored;
- If the variable is not being exploited by either UP or SI, then we must find its position in the individual. Once again, a mapping similar to the one created before is used.
- Finally, there is the encoding of the $n_{si}$ 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 14. *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* $\{(x'_1, 0), (x'_2, 1), (y_1, 2)\}$*.*

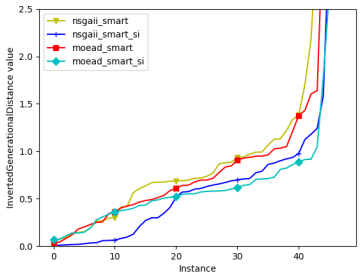
## 5. EXPERIMENTAL RESULTS

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 [3].

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




(a) Hypervolume of the hybrid algorithms.




(b) IGD of the hybrid algorithms.

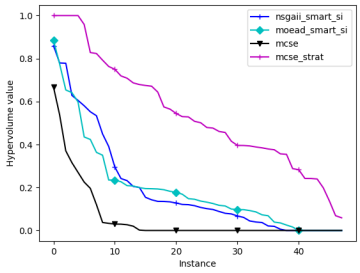
Figure 1: Performance of hybrid algorithms in VMC problems without wastage constraints with and without structure improvements.

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.

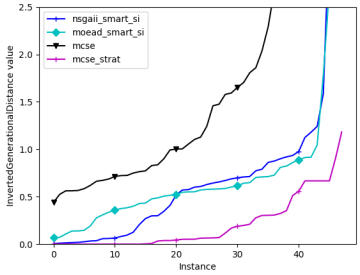
We evaluated the quality of the Pareto front approximations using the Inverted Generational Distance (IGD) [1] and Hypervolume [12] performance metrics, both presented in section 2.2. The IGD is a combined metric that measures the average distance from the cost vectors in a reference front $O(A_R)$ to 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.

### 5.1 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.

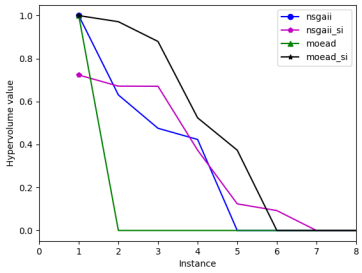



(a) Hypervolume of the MCSE algorithm.

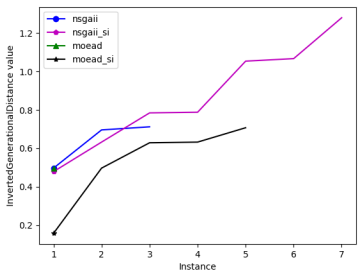



(b) IGD of the MCSE algorithm.

Figure 2: Comparison of the performance between hybrid and constraint based algorithms for VMC without wastage constraints.




(a) Hypervolume of the stochastic algorithms.




(b) IGD of the stochastic algorithms.

Figure 3: Performance of stochastic algorithms in FTP with and without structure improvements.

The performance of the stochastic algorithms with the application of smart operators is presented in figure 1. In this problem the addition of the structure improvement technique helps both hybrid algorithms achieve a better performance, as we can see on figure 1a 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 1b 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.

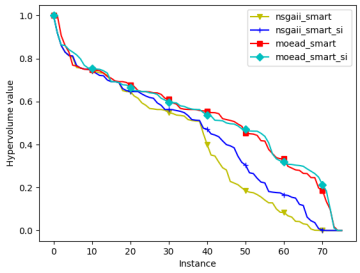
Next, on figure 2 we compare the performance of the MCSE algorithm with both hybrid algorithms using structure improvements. On figure 2a we can see that MCSE shows very poor performance, with an hypervolume value of 0 on almost thirty five instances. However, the stratified MCSE algorithm is able to 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 2b, 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.

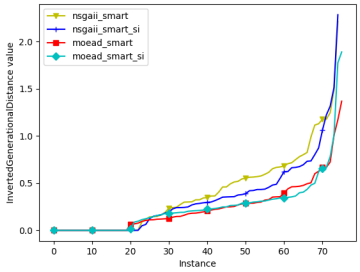
### 5.2 Flying Tourist Problem

The FTP problem is not as hard and constrained as the VMC problem, and therefore stochastic algorithms are able to find solutions for some of the instances, as can be seen in figure 3. NSGA-II and MOEA/D can only find feasible individuals on four and one instances, respectively, as we can see in figure 3a. The addition of structure improvements to the stochastic algorithms helps improve this 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 4. 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 4a 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 4b shows that all these algorithms are able to get an IGD value

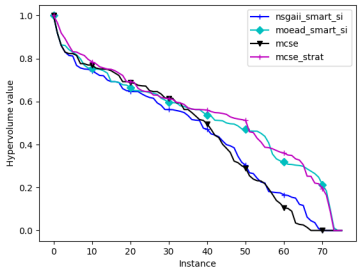



(a) Hypervolume of the hybrid algorithms.

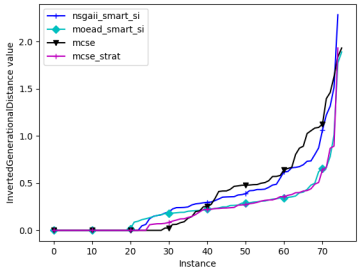



(b) IGD of the hybrid algorithms.

Figure 4: 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: Comparison of the performance between hybrid and constraint based algorithms for FTP.

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 shows the comparison of the MCSE algorithm with the hybrid algorithms MOEA/D and NSGA-II both with structure improvement technique enabled. From figure 5a 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 H_MOEA/D_SI throughout all instances. On figure 5b 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.

## 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, for some cases. 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, or even to new types of constraints.

## 7. REFERENCES

[1] Carlos A Coello Coello and Margarita Reyes Sierra. 2004. A study of the parallelization of a coevolutionary

multi-objective evolutionary algorithm. In *Mexican International Conference on Artificial Intelligence*. Springer, 688–697.

[2] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[3] Yongqiang Gao, Haibing Guan, Zhengwei Qi, Yang Hou, and Liang Liu. 2013. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *J. Comput. System Sci.* 79, 8 (2013), 1230–1242.

[4] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, and Yves Le Traon. 2013. Multi-objective test generation for software product lines. In *Proceedings of the 17th International Software Product Line Conference*. ACM, 62–71.

[5] Rafael Olaechea, Derek Rayside, Jianmei Guo, and Krzysztof Czarnecki. 2014. Comparison of exact and approximate multi-objective optimization for software product lines. In *Proceedings of the 18th International Software Product Line Conference-Volume 1*. ACM, 92–101.

[6] Abdel Salam Sayyad, Joseph Ingram, Tim Menzies, and Hany Ammar. 2013. Scalable product line configuration: A straw to break the camel's back. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 465–474.

[7] Miguel Terra-Neves, Inês Lynce, and Vasco Manquinho. 2017. Introducing Pareto minimal correction subsets. In *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 195–211.

[8] Miguel Terra-Neves, Inês Lynce, and Vasco Manquinho. 2019. Integrating Pseudo-Boolean constraint reasoning in multi-objective evolutionary algorithms. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 1184–1190.

[9] Miguel Terra-Neves, Inês Lynce, and Vasco M Manquinho. 2018. Stratification for Constraint-Based Multi-Objective Combinatorial Optimization.. In *IJCAI*. 1376–1382.

[10] Yuan Yuan and Wolfgang Banzhaf. 2018. ARJA: Automated repair of java programs via multi-objective genetic programming. *IEEE Transactions on Software Engineering* (2018).

[11] Qingfu Zhang and Hui Li. 2007. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation* 11, 6 (2007), 712–731.

[12] Eckart Zitzler. 1999. *Evolutionary algorithms for multiobjective optimization: Methods and applications*. Vol. 63. Citeseer.