Graph-based solution batch management for Multi-Objective Evolutionary Algorithms

P.M. Mateo[a,*], I. Alberto[b]

[a]*Depto. Métodos Estadísticos, Facultad de Ciencias, Universidad de Zaragoza, Pedro Cerbuna 12, 50009 Zaragoza, Spain.*
[b]*Depto. Métodos Estadísticos, Escuela de Ingeniería y Arquitectura, Universidad de Zaragoza, María de Luna 3, 50018 Zaragoza, Spain.*

Abstract

In Alberto and Mateo [2], 2004, a graph-based structure used for manipulating populations of Multi-Objective Evolutionary Algorithms in a more efficient way than the structures existing at that point was defined. In this paper, an improvement of such tool is presented. It consists of the simultaneous insertion of a set of solutions (solution batch), instead of a single one, into the created graph structure. Furthermore, two experiments devoted to comparing the behavior of the new algorithms with the original version from Alberto and Mateo [2] and with a well-known non-dominated sorting algorithm are carried out. The first shows how the new version outperforms the original one in time and number of Pareto comparisons. The second experiment shows a reduction in the time needed in all the cases and an important reduction in the number of Pareto comparisons when inserting chains of dominated solutions. From these experiments it is verified that, in general, the new proposals save computational time and, in the majority of the cases, the number of Pareto comparisons carried out for the insertion. In addition, when the new proposals outperform the others, they increase their gain over them as the size of the population and/or the size of the batch increases. The new tool can also be used, for example, in parallel genetic algorithms such as the ones based on islands, to carry out the migrations of the

[*]Corresponding Author
Email addresses: mateo@unizar.es (P.M. Mateo), isolina@unizar.es (I. Alberto)

solutions.

Keywords: Evolutionary Computation, Metaheuristic, Multi-Objective Optimization, Graph, Partially Ordered Set

1. Introduction

In Alberto and Mateo [2] a tool for managing the population of Multi-Objective Evolutionary Algorithms (MOEAs), significantly different from the proposals available in the literature at that time [15], was presented. The proposed method allowed the management of the whole population of a MOEA, efficient and non-efficient solutions, instead of only the efficient ones, as the available methods did. The use of this tool for managing the individuals of the populations, instead of storing them using linear lists or arrays, provided us with important improvements with respect to the time needed for executing, for instance, a simplified version of the well-known Non-dominated Sorting Genetic Algorithm-II, NSGA-II, [9], especially when the size of the population grows.

MOEAs are highly time consuming algorithms so researchers are focused on developing algorithms reaching "better" solutions in a "faster" way. Therefore, the improvement of any aspect of a MOEA is an extremely important task due to the great relevance that these kinds of algorithms exhibit nowadays; for instance, in Coello [7], more than 10,000 references related to MOEAs are collected by the well-known researcher C.A. Coello. The elements that researchers try to improve are, for example, representation and management of solutions, design of specific variation and selection operators, techniques for improving the coverage of the Pareto front, and so on. However, as far as we know, the number of papers devoted to the representation and management of solutions that bear in mind the specific issues of MOEAs is not significant enough.

The originally proposed tool was based on what the authors called *Irreducible Domination Graph*, IDG. A *Domination Graph*, DG, is a graph in which the individuals of the population of the MOEA are represented by means of nodes, and the domination relations among these individuals by means of arcs. In particular, the IDG is the simplest DG (i.e. the domination graph with the fewest number of arcs) which represents all the domination relations among the individuals of the population. The interest in the use of IDGs is to reduce the computational time needed for updating the population (i.e. for updating the IDG).

It should also be noticed that this structure and the tools built around it enable us to efficiently obtain some of the different elements associated with MOEAs, especially for Pareto Dominance-Based Multi-Objective Evolutionary Algorithms (PDMOEAs). For example, to accomplish a non-dominated sorting (NDS) of the solutions in the population, which is used in a great deal of algorithms based on NSGA-II; or to obtain the raw fitness assignment of algorithms as SPEA2 [28], which, for each individual, takes into account how many individuals it dominates and it is dominated by; or even for the rank of Fonseca and Fleming [12] based on the number of individuals dominating each solution. All these calculations can be carried out by using efficient existing search network algorithms.

The DG is an acyclic directed graph and the IDG is, in fact, the transitive closure of any of the different DGs that represent the population. Due to this, it could be used in other fields different from the Multi-Objective Optimization, in which a huge set of objects are ordered according to a certain partial order and where it is important to rapidly establish the relationship between every pair of objects in the set. Examples that could be considered are the ones related to Formal Concept Analysis [4] in which the different formal concepts are ordered according to an appropriate partial order and then, a concept lattice is built (a Direct Acyclic Graph, DAG). They are also important in the database field, the computation of the transitive closure of large database relations [1] allows us to answer, in quite a fast way, reachability requests; almost all important database engines incorporate commands to calculate the transitive closure of databases. A final example in which it could be useful is as a tool in the process of inferring cellular networks in which some specialized versions of transitive closures are calculated, [17]. However, the development of this paper is centered in the field of the Multi-Objective Optimization as in Alberto and Mateo [2].

So far, we have not found algorithms that manage the whole population and/or use the solution batch processing. The ones most used consist of storing the solutions in arrays. An important issue for them is reducing of the time needed to obtain the efficient solutions or the Pareto layers of a population: the first NDS was proposed by [14] with complexity $O(mN^3)$, (m no. objective functions, N population size); which was reduced to $O(mN^2)$ in [9]. With the same complexity but more efficient in practice, are [19, 21, 25, 26].

Other tools for storing and managing solutions are: quad-trees [15], dominated and non-dominated trees [11], dominance decision trees [22], Pareto optimal trees [6] and fast incremental binary space partitioning (BSP) trees [13]. What they have in common is that they only manage the efficient solutions, which makes them appropriate alternatives for storing the elitist external archives some algorithms use. The efficient solutions are stored in different kinds of trees. For instance, the first one uses quad-trees, a special structure previously proposed to represent multidimensional graphical point data in which each node corresponds to one solution and the children are defined by a successorship relation based on the values of the objective functions of the compared solutions. On the other hand, dominated trees [23], and trie-trees [20] are devoted to managing the whole population in a similar but not equivalent way to how IDGs do. Dominated trees define a modification of a Binary Search Tree in which the dominated solutions of a node are incorporated into its left-subtree and non-comparable solutions into its right-subtree. The difference, with respect to our proposal, is that with these methodologies some relations between non-efficient solutions could be lost, or the information of Pareto layers which are different from the first one might even not be directly accessible. Finally, the tool based on trie-trees is a hybrid methodology since it maintains two trees: One devoted to storing efficient solutions and the other for the remaining ones. In addition, information about efficiency relations is not stored in the population tree. Although it is a new storing structure, it is hardly related to our proposal. A brief but complete description of the majority of these algorithms can be read in Altwaijry and El Bachir Menai [3].

The aim of this paper is to show an improvement of the original algorithm for incorporating solutions into the population, i.e., for inserting individuals into the IDG. The main contribution of this paper is the development of two new algorithms for the insertion of nodes into the IDG. The novelty consists of the solution batch processing, i.e., the simultaneous insertion of a set of non-comparable solutions, or a chain of dominated solutions (the first solution dominates the second one, which dominates the third and so on). For both alternatives, the specific characteristics of these batches of solutions will be considered and taken into account, the theoretical results needed will be presented and the specific algorithms will be developed. After showing the algorithms, a first experiment to show how the new proposal outperforms the original one will be conducted. A second experiment will compare the proposals with a well-known NDS algorithm [19]. For these experiments, the execution time and the number of accomplished comparisons between solutions in order to determine the Pareto relations among them (see Section 6) will be compared. With the experiments carried out it will be shown that the solution batch processing improves the speed of the algorithm with respect to the sequential processing of our original version and the NDS; hence, it could accelerate the performance of classical PDMOEAs, as for instance NSGA-II, SPEA2, etc., and their variants.

This paper is organized as follows: In the next section, the definitions and notations which will be used throughout the article are presented. Next, Section 3 examines the case of the simultaneous insertion of k non-comparable solutions. The case of the simultaneous insertion of a sequence of k dominated solutions is presented in Section 4. The next section presents the study of the theoretical complexities of the algorithms introduced in the article and Section 6 analyzes the practical behavior of the algorithms when applying them to a classical collection of test problems. The final conclusions and future research lines are detailed in Section 7.

2. Previous definitions and notations

2.1. Multi-Objective Optimization Problems

Multi-Objective Optimization Problems (MOOPs) tend to be characterized by a family of alternatives that must be considered equivalent in the absence of information concerning the relevance of each objective with respect to the others. A MOOP can be defined as follows:

$$\begin{aligned} \text{Minimize} \quad & \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_q(\mathbf{x})), \\ \text{subject to:} \quad & \mathbf{x} = (x_1, \ldots, x_p) \in D \subset \mathbb{R}^p, \end{aligned} \tag{1}$$

and the solutions to one of such problems are formed with all the feasible solutions of the search space such that the components of the objective function, $\mathbf{f}(\mathbf{x})$, cannot all be simultaneously improved. These solutions are called *Pareto optimal* or *efficient*.

Definition 1. A solution $\mathbf{x} \in D$ is said to be *Pareto optimal* or *efficient* if and only if $\nexists \mathbf{y} \in D$ such that $\forall k = 1, \ldots, q, f_k(\mathbf{y}) \leq f_k(\mathbf{x})$ and $\exists k \in \{1, \ldots, q\}$ such that $f_k(\mathbf{y}) < f_k(\mathbf{x})$. Given $\mathbf{x}, \mathbf{y} \in D$, solution $\mathbf{x}$ *dominates* solution $\mathbf{y}$, denoted by $\mathbf{x} \prec \mathbf{y}$, if $\forall k = 1, \ldots, q, f_k(\mathbf{x}) \leq f_k(\mathbf{y})$ and $\exists k \in \{1, \ldots, q\}$ such that $f_k(\mathbf{x}) < f_k(\mathbf{y})$. In this case, solution $\mathbf{y}$ is called a *non-efficient* or *dominated solution*. If $\mathbf{x}, \mathbf{y} \in D$ such that $\mathbf{x} \nprec \mathbf{y}$ and $\mathbf{x} \nsucc \mathbf{y}$, the solutions are called *non-comparable solutions*, denoted by $\mathbf{x} \not\sim \mathbf{y}$.

2.2. Evolutionary Algorithms

Evolutionary Algorithms (EAs) are search and optimization methods based on the principles of natural evolution and genetics whose most important difference with classical techniques is that, in EAs, a population of solutions is processed in every generation. This is a tremendous advantage for its use in solving multi-objective optimization problems, since they seek the Pareto optimal set. An extensive review on this matter can be obtained from Coello et al. [8], Zhou et al. [27], Talbi et al. [24] and an introduction can be obtained from Eiben and Smith [10].

The population of a Multi-Objective Evolutionary Algorithm in its t-th iteration is denoted by $\mathcal{P}_t$, and by n its size. The individuals of the population, feasible solutions of the MOOP, are denoted by x_i, $i = 1, \dots, n$. In the selection process of an EA, the individuals of the population need to be ordered. In general, orders based on the Pareto relation are used, as in the well-known algorithm NSGA-II (Deb et al. [9]), in which, from the joint population of parents and children, the fittest individuals are selected according to the partial order derived from the domination relation established by Definition 1. Reviewing bibliography on MOEAs, there are many algorithms that maintain a similar scheme to NSGA-II.

2.3. Graphs

A graph $\mathcal{G}$ is a pair $(\mathcal{N}, \mathcal{A})$ where the set $\mathcal{N} = \{x_1, \dots, x_n\}$ is called the *node set* of $\mathcal{G}$ and its elements are called *nodes*; and the set $\mathcal{A} = \{(x_i, x_j) | x_i, x_j \in \mathcal{N}\} \subseteq \mathcal{N} \times \mathcal{N}$ is called the *arc set* and its elements are called *arcs*.

In this work *directed graphs* are considered, that is to say, if the arcs (x_i, x_j) and (x_j, x_i) exist in $\mathcal{A}$, then they are different. A *directed path in $\mathcal{G}$ from node x_i to node x_j* is a sequence of distinct arcs, $v_1, v_2, \dots, v_p$, $p \geq 1$, such that a corresponding sequence of nodes exists $x_i = x_{s_0}, x_{s_1}, \dots, x_{s_p} = x_j$ satisfying $v_h = (x_{s_{h-1}}, x_{s_h}) \in \mathcal{A}$, for $1 \leq h \leq p$. A directed path is a *simple directed path* if all its nodes are different. A *cycle* is a directed path where the first and last nodes coincide. A graph $\mathcal{G}$ with no cycles is called an *acyclic graph*. The notation $x_i \hookrightarrow x_j$ represents that a simple directed path from x_i to x_j exists in $\mathcal{G}$. In this case, x_i is an *ancestor* of x_j and x_j is a *descendant* of x_i, and A_j denotes the set of ancestors of x_j and D_i the set of descendants of x_i. If the directed path only consists of the arc (x_i, x_j), the notation $x_i \to x_j$ can be used, if it is explicitly necessary to emphasize it. In this case, x_i is a *parent* of x_j and x_j is a *child* of x_i.

Finally, let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ be a directed graph. The *transitive closure* of $\mathcal{G}$ is the graph $\bar{\mathcal{G}} = (\mathcal{N}, \mathcal{A}^*)$ where $\mathcal{A}^* = \{(x_i, x_j) | x_i, x_j \in \mathcal{N}, x_i \hookrightarrow x_j \text{ in } \mathcal{G}\}$. The *transitive reduction* of $\mathcal{G}$ is graph $\mathcal{G}^- = (\mathcal{N}, \mathcal{A}^-)$ with a minimal number of arcs satisfying $\bar{\mathcal{G}} = (\overline{\mathcal{G}^-})$.

2.4. Domination graphs

For managing the populations of a MOEA, Domination Graphs, DGs, are used. A DG is a graph whose nodes have associated individuals x_i of a population and in which the arcs represent the domination relations between individuals. An arc $x_i \rightarrow x_j$ means that the solution associated with x_i dominates the solution associated with x_j ($x_i \prec x_j$). A directed path $x_i \hookrightarrow x_j$ also means, by the transitivity of the domination relation, that the solution associated with x_i dominates the solution associated with x_j.

Definition 2. Let $\mathcal{N}$ be the set of nodes associated with all the feasible solutions to a problem, one node per solution. Let $\mathcal{N}_t \subseteq \mathcal{N}$ be the set of all the nodes associated with the t-th population, $\mathcal{P}_t$, in a run of the MOEA. Let $\mathcal{A}_t \subseteq \mathcal{N}_t \times \mathcal{N}_t$ be the arc set such that it reflects the domination relations of the elements in the current population, i.e., given x_i and x_j in $\mathcal{P}_t$ such that $x_i \prec x_j$, then a directed path from x_i to x_j has to exist. The pair $(\mathcal{N}_t, \mathcal{A}_t)$ is called *Domination Graph*, DG, associated with the population $\mathcal{P}_t$, and it is denoted by $\mathcal{G}_t = (\mathcal{N}_t, \mathcal{A}_t)$.

To make the management as efficient as possible, among all the possible DGs associated with one population, the one with the fewest number of arcs is selected. This is called *Irreducible Domination Graph*, IDG, which is actually the transitive reduction of any of the DGs of our population. Lemma 1 and Theorem 1 establish a characterization and the uniqueness of the IDG (both results have been taken from [2]).

Lemma 1. *Given a population $\mathcal{P}_t$ and a DG associated, $\mathcal{G}_t$, then $\mathcal{G}_t$ is irreducible if and only if $\forall x_i, x_j \in \mathcal{N}_t$ such that $x_i \prec x_j$ only one of the next statements holds:*

1. *The only directed path from x_i to x_j is the arc $x_i \rightarrow x_j$.*
2. *Every directed path from x_i to x_j, $x_i \hookrightarrow x_j$, has two or more arcs.*

Theorem 1. *Given a population $\mathcal{P}_t$, the irreducible domination graph associated with it is unique.*

The algorithm for adding a solution y_j to an IDG (a new solution in the population) can be seen in Algorithm 1 and it works with the following sets, which will also be used in the simultaneous insertion algorithms that will be presented later:

- $B_j = \{x_i \in \mathcal{N}_t | x_i \prec y_j\}$, B_j will be y_j's ancestors.
- $B_j^* = \{x_i \in B_j | \nexists x_k \in B_j$ such that $x_i \prec x_k\}$, B_j^* will be y_j's parents.
- $b_j = \{x_i \in \mathcal{N}_t | y_j \prec x_i\}$, b_j will be y_j's descendants.
- $b_j^* = \{x_i \in b_j | \nexists x_k \in b_j$ such that $x_k \prec x_i\}$, b_j^* will be y_j's children.

As shown in Algorithm 1, the procedure for inserting y_j is the following: If $x_i \prec y_j$ then x_i's ancestors set, A_i, is determined (line 2); and if $x_i \succ y_j$, x_i's descendants set, D_i, is determined (line 3). A labeling procedure lets us identify sets B_j^* and b_j^* (lines 4 and 5). Then, all the arcs between nodes in B_j^* and nodes in b_j^* are removed since they will be redundant in the new graph (line 6). Finally, the new node y_j is added (line 7) and the domination relations are re-established by adding all the arcs $B_j^* \to y_j$ and $y_j \to b_j^*$ to the IDG (lines 8 to 11). The complete description of this algorithm can be seen in [2].

Algorithm 1: Insertion Algorithm

```
   Data: IDG, $y_j$ to be added to the IDG
   Result: IDG
1  foreach $x_i \in \mathcal{N}_t$ and $x_i$ unmarked do
2      if $x_i \prec y_j$ then mark all unmarked nodes of $A_i$ and $B_j = B_j \cup \{x_i\}$;
3      if $x_i \succ y_j$ then mark all unmarked nodes of $D_i$ and $b_j = b_j \cup \{x_i\}$;
4  Determine the set $B_j^*$ of unmarked nodes of $B_j$;
5  Determine the set $b_j^*$ of unmarked nodes of $b_j$;
6  $\mathcal{A}_t = \mathcal{A}_t \setminus \{(x_i, x_k) | x_i \in B_j^*, x_k \in b_j^*\}$;
7  $\mathcal{N}_t = \mathcal{N}_t \cup \{y_j\}$;
8  foreach $x_i \in B_j^*$ do
9      $\mathcal{A}_t = \mathcal{A}_t \cup \{(x_i, y_j)\}$;
10 foreach $x_i \in b_j^*$ do
11     $\mathcal{A}_t = \mathcal{A}_t \cup \{(y_j, x_i)\}$;
```

2.5. Example

Let $\{a, b, c, d, e, f, g, h, i, j\}$ be the population of solutions of a bi-objective minimization problem, (f_1, f_2). The values of the objective functions and the resulting IDG after applying Algorithm 1 are shown in Fig. 1.

	(f_1, f_2)
a	$(0,3)$
b	$(3,0)$
c	$(1,7)$
d	$(4,3)$
e	$(5,1)$
f	$(2,9)$
g	$(6,5)$
h	$(7,2)$
i	$(8,8)$
j	$(9,6)$

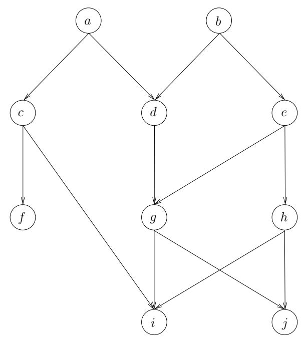


Figure 1: Population and IDG obtained after successively applying Algorithm 1.

3. Simultaneous insertion of k non-comparable solutions

The first new proposal consists of the insertion of a set of non-comparable solutions, it generalizes Algorithm 1, and it follows a similar schema as can be seen in Algorithm 2.

Let $\{y_1, \ldots, y_k\}$ such that $y_i \not\sim y_j$ $i, j = 1, \ldots, k$, be a set of solutions to be inserted. For inserting every new solution y_j into the IDG it is essential to identify sets B_j^* and b_j^*, $\forall j$, and to investigate the relationships among them. These tasks are carried out by means of Function LabelingProcessNC (line 1). Next, the following processes to be accomplished are the removal of the arcs between nodes of B_j^* and nodes of b_j^* (lines 2 and 3), the insertion of the new nodes y_j (line 4), and finally the re-establishment of the domination relations by adding the arcs $B_j^* \to y_j$ and $y_j \to b_j^*$, $\forall j$ (lines 5 and 6).

```
Algorithm 2: Insertion Algorithm
  Data: Node list, $\mathcal{N}_t$; Arc list, $\mathcal{A}_t$;
        Solutions to be inserted, $y_1, \ldots, y_k$, $y_i \not\sim y_j$, $i \neq j$
  Result: Updated node list, $\mathcal{N}_t$; Updated arc list $\mathcal{A}_t$
1 LabelingProcessNC($\mathcal{N}_t$);
2 for $j = 1$ to $k$ do
3     $\mathcal{A}_t = \mathcal{A}_t \setminus \{(x, x') | \exists j \in \{1, \ldots, k\}$ such that $l_j(x) = 2, l_j(x') = -2\}$;
4 $\mathcal{N}_t = \mathcal{N}_t \cup \{y_1, \ldots, y_k\}$;
5 for $j = 1$ to $k$ do
6     $\mathcal{A}_t = \mathcal{A}_t \cup \{(x, y_j) | l_j(x) = 2\} \cup \{(y_j, x) | l_j(x) = -2\}$;
```

The following lemma lets us simplify the labeling process carried out by Function LabelingProcessNC since it justifies that only a certain kind of labels will be used. The proof of this lemma can be found in the Appendix.

Lemma 2. *Given y_i and y_j such that $y_i \not\sim y_j$, then $B_i \cap b_j = \emptyset$. As a consequence, $B_i^* \cap b_j^* = \emptyset$.*

3.1. Obtaining B_j^* and b_j^*

The labeling procedure proposed in Function LabelingProcessNC lets us identify sets B_j^* and b_j^*, $\forall j$. The main idea of this function is to update the labels in parallel, i.e., instead of labeling each node in relation to one of the new solutions, each node will be labeled, if it is possible, in relation to several of the new solutions at the same time.

Let $y_1, \ldots, y_k$ be the solutions to be inserted. Every node x in the current IDG will receive one label, $L(x) = (l_1(x), \ldots, l_k(x))$, where component j is associated to solution y_j. If $l_j(x)$ is positive and equal to 1 or 2, it means that node x has been labeled in an ascending trajectory (construction of B_j^*); if $l_j(x)$ is negative and equal to -1 or -2 it means that node x has been labeled in a descending trajectory (construction of b_j^*); and if $l_j(x)$ it is equal to 3, it means that node x is not comparable with y_j. Initially, $\forall x \in \mathcal{N}_t$ will have $L(x) = \mathbf{0}$ (line 2) and it means that node x is still not labeled in any component.

As a consequence of Lemma 2, all the components of $L(x)$ different from 3 will have the same sign (positive or negative), i.e. no node x will have positive and negative values different from 3 in its label (this fact would imply that $B_i \cap b_j \neq \emptyset$).

Given a node $x \in \mathcal{N}_t$ and its label $L(x) = (l_1(x), \ldots, l_k(x))$, the following elements are defined:

$$z(x) = \{j | l_j(x) = 0\}_{j \in \{1,\ldots,k\}}$$
$$p(x) = \#\{j | l_j(x) \in \{0, 2, -2\}\}_{j \in \{1,\ldots,k\}}$$
$$s(x) = sign\{l_j(x) | l_j(x) \notin \{0, 3\}\}_{j \in \{1,\ldots,k\}}$$

Function LabelingProcessNC($\mathcal{N}_t$)

```
Input: Node list, $\mathcal{N}_t$
Result: Set of labels, $L = \{L(x) = (l_1(x), \ldots, l_k(x)) | x \in \mathcal{N}_t\}$
$p(x) = k, \forall x \in \mathcal{N}_t$;
$L(x) = (0, \ldots, 0), \forall x \in \mathcal{N}_t$;
foreach $x \in \mathcal{N}_t$ do
    $P = \emptyset$;
    if $z(x) \neq \emptyset$ then
        foreach $j \in z(x)$ do
            if $x \prec y_j$ then
                $l_j(x) = 2$;
                $P = P \cup \{j\}$;
            else if $y_j \prec x$ then
                $l_j(x) = -2$;
                $P = P \cup \{j\}$;
            else
                $l_j(x) = 3$;
    $p(x) = |P|$;
    if $p(x) > 0$ then
        switch $s(x)$ do
            case $(+)$ do
                $L \leftarrow AscLabelNC(x, P, L, \mathbf{p})$
            case $(-)$ do
                $L \leftarrow DescLabelNC(x, P, L, \mathbf{p})$
```

i.e. $z(x)$ keeps the components of $L(x)$ which are not labeled; $p(x)$ is equal to the number of pending components of $L(x)$, i.e. components to be examined; and $s(x)$ contains the sign of the labels, '$+$' for an ascendant labeled node (Function AscLabelNC) or '$-$' for a descendant labeled node (Function DescLabelNC[1]).

Briefly, Function LabelingProcessNC works as follows. First, the number of pending components is set to k and labels are set to 0 (lines 1 and 2). Then, for each node x partially labeled ($z(x) \neq \emptyset$) (line 5), it is completely labeled and the set of current components P is established (lines 6 to 14). If P is not empty (line 16) a labeling process which depends on the sign $s(x)$ is carried out (lines 18-19 or 20-21). Functions AscLabelNC/DescLabelNC accomplish a standard ascendant/descendant labeling process for all the components of the set of current indexes, P. Finally, Function LabelingProcessNC provides us with the nodes of B_j^* (those with label 2 in component j) and the nodes of b_j^* (those with label -2 in component j). Furthermore, if a node has several labels equal to 2 (-2), $l_{j_1}(x) = l_{j_2}(x) = \cdots = l_{j_q}(x) = 2$ (-2), then it belongs to the intersection $B_{j_1}^* \cap B_{j_2}^* \cdots \cap B_{j_q}^*$, ($b_{j_1}^* \cap b_{j_2}^* \cdots \cap b_{j_q}^*$).

3.2. Insertion procedure

As it has been previously established, Function LabelingProcessNC lets us obtain the labels for all the nodes in the IDG and to identify sets B_j^* and b_j^*, $j = 1 \ldots, k$. Following the insertion procedure of Algorithm 2, the next step is the removal of arcs between B_j^* and b_j^* (lines 2 and 3), i.e. all the arcs (x, x') for which there exists at least one $j \in \{1, \ldots, k\}$ with $l_j(x) = 2$ and $l_j(x') = -2$. Then, the new y_j nodes are incorporated into the graph (line 4) and all the relations between the current and the new nodes are re-established (lines 5 and 6). All these processes are the sequential application of the corresponding steps of the original insertion algorithm (Algorithm 1), which can be applied in this

[1]The description of Function DescLabelNC is omitted because it is analogous to Function AscLabelNC in which *start*(a) must be changed to *end*(a), 1 to -1 and *incident in* x to *salient from* x

```
Function AscLabelNC($x_0, P, L, \mathbf{p}$)
   Input: Initial node, $x_0$; Current components, $P$; Set of labels, $L$; Set of
          No. of pending components, $\mathbf{p}$
   Result: Updated set of labels, $L$ and pending components, $\mathbf{p}$
 1 $NodeList = \{x_0\}$;
 2 while $NodeList \neq \emptyset$ do
 3     Extract $x$ from $NodeList$;
 4     foreach arc $a$ incident in $x$ do
 5         if $p(start(a)) > 0$ then
 6             $aux =$"F";
 7             foreach $j \in P$ do
 8                 if $l_j(start(a)) \neq 1$ then
 9                     $l_j(start(a)) = 1$;
10                     $p(start(a)) = p(start(a)) - 1$;
11                 if $l_j(start(a)) == 0$ then
12                     $aux =$"T";
13             if $aux ==$"T" then
14                 $NodeList = NodeList \cup \{start(a)\}$;
```

way since the non-comparability of the new solutions causes the arcs to be removed and so the new ones to be inserted become independent.

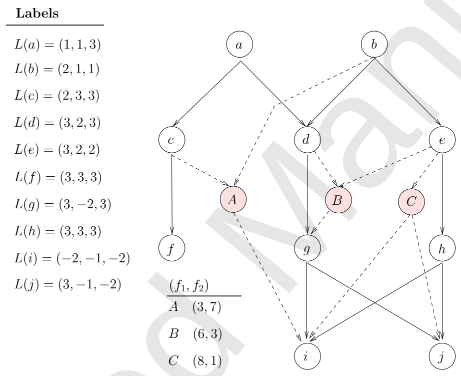
To finish this section, the following theorem states the correctness of the algorithm.

Theorem 2. *Let $\mathcal{G}_t$ be the initial IDG associated with a population $\mathcal{P}_t$ and $\mathcal{G}'_t$ the graph obtained by applying the former procedure for inserting the solutions $y_1, \ldots, y_k$ such that $y_i \nsim y_j$, $\forall i \neq j$. Then $\mathcal{G}'_t$ is the corresponding IDG.*

Proof. By construction, Function LabelingProcessNC ensures the correct identification of sets B^*_j and b^*_j, $\forall j$. Then, given that $B^*_i \cap b^*_j = \emptyset$ (by Lemma 2) and since the procedure for inserting $y_1, \ldots, y_k$ such that $y_i \nsim y_j$, $\forall i \neq j$, is the natural extension of the one proposed in Alberto and Mateo [2], the graph obtained after applying the procedure is an IDG. $\square$

3.3. Example

Let A, B and C be the images in the objective space of three non-comparable solutions, see Fig. 2, to be inserted into the IDG of Fig. 1. After applying the labeling process of Function LabelingProcessNC, the labels obtained for the nodes of the IDG are also shown in Fig. 2. Then, lines 2 and 3 of Algorithm 2 determine that the arcs to be removed are (e, g) and (c, i). Once the new solutions are inserted (line 4), lines 5 and 6 determine the new arcs to be added, which appear in dashed line in Fig. 2.



Labels

$L(a) = (1, 1, 3)$
$L(b) = (2, 1, 1)$
$L(c) = (2, 3, 3)$
$L(d) = (3, 2, 3)$
$L(e) = (3, 2, 2)$
$L(f) = (3, 3, 3)$
$L(g) = (3, -2, 3)$
$L(h) = (3, 3, 3)$
$L(i) = (-2, -1, -2)$
$L(j) = (3, -1, -2)$

	(f_1, f_2)
A	$(3, 7)$
B	$(6, 3)$
C	$(8, 1)$

Figure 2: A, B, C non-comparable solutions to be inserted, set of labels and new added arcs (in dashed line) after applying Algorithm 2.

4. Simultaneous insertion of a sequence of k dominated solutions

In this section a sequence of k solutions such that each solution is dominated by the previous one, i.e. k solutions $y_1, y_2, \ldots, y_k$ such that $y_1 \prec y_2 \prec \cdots \prec y_k$ is considered. The objective of this section is to develop an algorithm for the simultaneous insertion of the whole set.

As in the previous section, sets B_j^* and b_j^*, $\forall j$, have to be identified as well as all the existing relations among them. But unlike what happened in the case of non-comparable solutions, after constructing the sets and removing the corresponding arcs, the insertion must be made taking into account the order in which the new solutions are added. This is so because the insertion of previous solutions can alter the composition of the remaining B_j^* and b_j^*. For example, Fig. 3.a) shows the simultaneous insertion of two solutions with sets B_j^* and $b_j^*, j = 1, 2$, and Fig. 3.b) shows how B_2^* and b_2^* will be if solution y_1 has already been inserted into the IDG.

Algorithm 3: Insertion Algorithm

```
   Data: Node list, $\mathcal{N}_t$; Arc list, $\mathcal{A}_t$; Solutions to be inserted, $y_1 \prec \cdots \prec y_k$
   Result: Updated node list, $\mathcal{N}_t$; Updated arc list, $\mathcal{A}_t$
1  LabelingProcessC($\mathcal{N}_t$);
2  for $j = 1$ to $k$, $i \geq j$ do
3      $\mathcal{A}_t = \mathcal{A}_t \setminus \{(x, x') | l_p(x) \in \{2j, 2j+1\}$ and $l_n(x') \in \{-2i-2, -2i-3\}\}$;
4  $\mathcal{N}_t = \mathcal{N}_t \cup \{y_1, \ldots, y_k\}$;
5  for $j = 1$ to $k$ do
6      $\mathcal{A}_t = \mathcal{A}_t \cup \{(x, y_j) | l_p(x) \in \{2j, 2j+1\}\} \cup \{(y_j, x) | l_n(x') \in \{-2i-2, -2i-3\}\}$;
7      if $\nexists x$ such that $l_n(x) = -2j - 3$ then
8          $\mathcal{A}_t = \mathcal{A}_t \cup (y_j, y_{j+1})$;
```

The new proposal can be seen in Algorithm 3. Its general steps are the same as in Algorithm 2. First, Function LabelingProcessC accomplishes a labeling process in order to identify some sets related with B_j^* and b_j^* (line 1). After this, the removal of the corresponding arcs between nodes of these sets (lines 2 and 3) and the insertion of the new nodes (line 4) are carried out. Then, a new arc addition process is proposed, which implicitly considers the necessary updating of sets B_j^* and b_j^* as the successive new solutions y_j are added (lines 5 to 8).

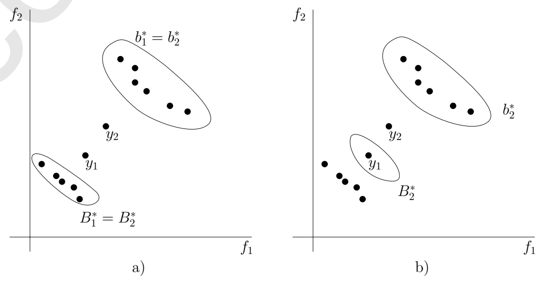


Figure 3: Example of insertion of y_1 and y_2 with the construction of sets B_j^* and b_j^* in the function space. a) Simultaneous insertion. b) Successive insertion.

A very important issue in further development of our algorithm is the following decomposition of sets B_j^* and b_j^*, $j = 1, 2, \dots, k$:

$$\begin{aligned} B_j^* = & \quad B_j^{*1} \dot{\cup} B_j^{*2} \dot{\cup} B_j^{*3}, \\ b_j^* = & \quad b_j^{*1} \dot{\cup} b_j^{*2} \dot{\cup} b_j^{*3}, \end{aligned} \tag{2}$$

where:

- B_j^{*1} = nodes in B_j^* which dominate y_{j-1}, i.e., belonging to B_{j-1}.
- B_j^{*2} = nodes in B_j^* non-comparable with y_{j-1}.
- B_j^{*3} = nodes in B_j^* dominated by y_{j-1}, i.e., belonging to b_{j-1}.

That is to say, set B_j^* is divided into three disjoint subsets according to the situation of its nodes with respect to y_{j-1}, i.e., the "previous" solution.
And:

- b_j^{*1} = nodes in b_j^* dominated by y_{j+1}, i.e, belonging to b_{j+1}.
- b_j^{*2} = nodes in b_j^* non-comparable with y_{j+1}.
- b_j^{*3} = nodes in b_j^* which dominate y_{j+1}, i.e., belonging to B_{j+1}.

In this case, set b_j^* is divided into three disjoint subsets according to the situation of its nodes with respect to y_{j+1}, i.e., the "next" solution. For convenience with notation, if $j = 1$, $B_1^* = B_1^{*3}$ and $B_1^{*1} = B_1^{*2} = \emptyset$. Similarly, if $j = k$, $b_k^* = b_k^{*3}$ and $b_k^{*1} = b_k^{*2} = \emptyset$. In Fig. 4, for three solutions $y_{j-1} \prec y_j \prec y_{j+1}$, the distribution of the different subsets can be observed in an example in $\mathbb{R}^2$.

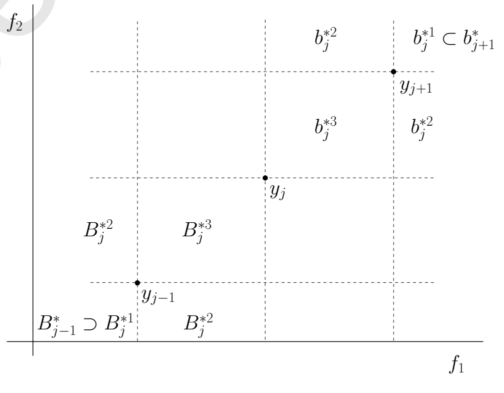


Figure 4: Distribution of sets B_j^* and b_j^* for $y_{j-1} \prec y_j \prec y_{j+1}$ in the function space (f_1, f_2).

4.1. Obtaining B_j^* and b_j^*

In order to identify sets B_j^* and b_j^* and their appropriate decomposition according to Eq.(2), it is necessary to define the following node lists:

$$^aL_c^b = \{x \in \mathcal{N}_t | x \not\sim a \text{ and } b \prec x \prec c\}, \tag{3}$$

i.e., $^aL_c^b$ includes all those nodes in the IDG which are non-comparable with node "a", which are dominated by node "b" and which dominate node "c".

In this case, since $y_1 \prec \cdots \prec y_k$, and in order to classify the nodes of the IDG, the following lists will be used, $j = 2, \ldots, k$.

- L_j^{j-1}: set of nodes dominated by y_{j-1} which dominate y_j (candidates to $B_j^{*3} \cup b_{j-1}^{*3}$).
- $^jL^{j-1}$: set of nodes non-comparable with y_j and which are dominated by y_{j-1} (candidates to b_{j-1}^{*2}).
- $^{j-1}L_j$: set of nodes non-comparable with y_{j-1} which dominate y_j (candidates to B_j^{*2}).

Fig. 5 shows an example for three new solutions, y_1, y_2 and y_3, to be inserted. There, the position of the different lists is depicted in relation with the new solutions y_j, as well as the subset of Eq.(2) associated with that node list. By construction, the following pairs of lists are always disjoint: L_j^{j-1} and L_i^{i-1}; $^jL^{j-1}$ and $^iL^{i-1}$; $^{j-1}L_j$ and $^{i-1}L_i$; L_j^{j-1} and $^{i-1}L_i$, $\forall i \neq j$; and $^{j-1}L_j$ and $^jL^{j-1}$, $\forall j$ (see Fig. 6 for an intuitive example in $\mathbb{R}^2$). However, $^{j-1}L_j$ and $^iL^{i-1}$, $j > i$, $i = 2, \ldots, k$ may not be disjoint (these sets are those on the left of Fig. 5 with the ones on the right at the same height and lower). But, as it will be explained later in the labeling process, nodes in $^{j-1}L_j$ will receive positive labels; and nodes in $^iL^{i-1}$ will receive negative labels. Then, if $x \in {}^{j-1}L_j \cap {}^iL^{i-1}$, x will receive both, positive and negative.

In Function ListAssignmentC the lists L_1^0, $^1L^0$, $^0L_1$, L_{k+1}^k and $^{k+1}L^k$ appear; these have not been previously defined. They correspond to cases $j = 1$ and

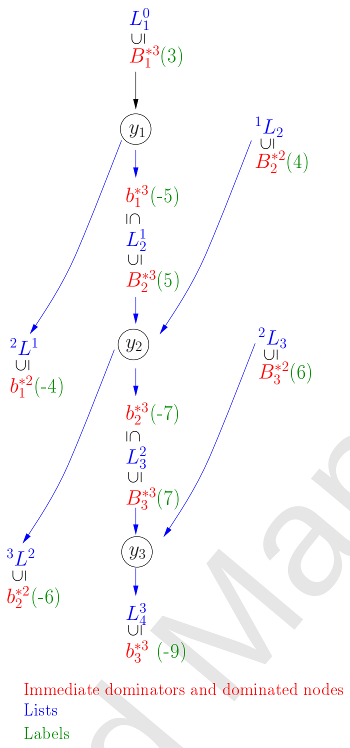


Figure 5: Node lists created for the insertion process of 3 solutions $y_1 \prec y_2 \prec y_3$.

$j = k$: L_1^0 contains the candidates to B_1^*; L_{k+1}^k the ones to b_k^* and ${}^1L^0$, ${}^0L_1$ and ${}^{k+1}L^k$ are used for an easier description of the algorithm.

The development continues showing the process for building the sets of Eq.(2). It starts in Function LabelingProcessC by executing Function ListAssignmentC for comparing the nodes x of the IDG with the successive y_j and for including them in the appropriate node list (lines 1 to 3). Function ListAssignmentC goes over the tree of Fig. 7, in order to add these nodes to their appropriate lists (${}^{j-1}L_j$, L_j^{j-1} or ${}^jL^{j-1}$). Given an x node from the IDG, x is compared to y_1. If $x \prec y_1$, x ends in Ds_1; if $x \not\sim y_1$, x goes to the second level from Nc_1; and if $y_1 \prec x$, then x also goes to the second level from Db_1. If x is on the second level, x is compared to y_2. If $x \prec y_2$, x ends in Ds_2; if $x \not\sim y_2$, then x goes to the next level from Nc_2; and if $y_2 \prec x$, x goes to the next level from Db_2. The process continues until x reaches a terminal vertex, and it then takes another

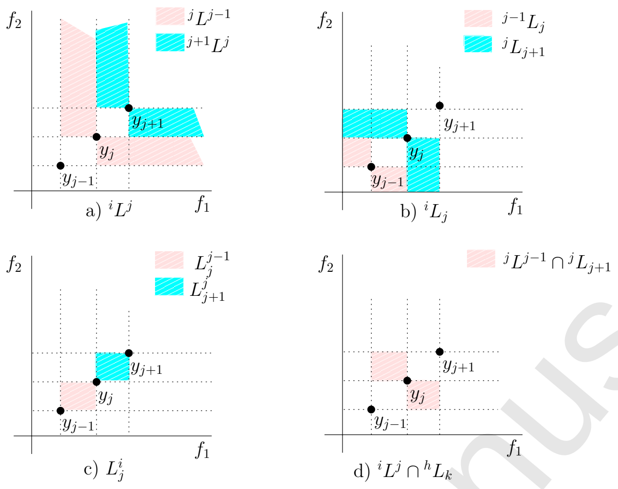


Figure 6: Description and relations among the different node lists in the function space (f_1, f_2).

node from the IDG to repeat the process. Once the nodes have reached a terminal vertex they are assigned to a list (L_j^{j-1}, ${}^{j}L^{j-1}$ or ${}^{j-1}L_j$). For example, for $j = 2, \ldots, k-1$, if $x \prec y_j$, the process ends in Ds_j and x joins L_j^{j-1} if it comes from Db_{j-1}, or ${}^{j-1}L_j$ if it comes from Nc_{j-1} in level $j-1$; and if $x \not\prec y_j$ and in level $j-1$ it comes from Db_{j-1}, then x joins ${}^{j}L^{j-1}$. Lines 1 to 5 of this function are included in order to avoid superfluous Pareto comparisons when the parent of the solution to be classified has been previously classified.

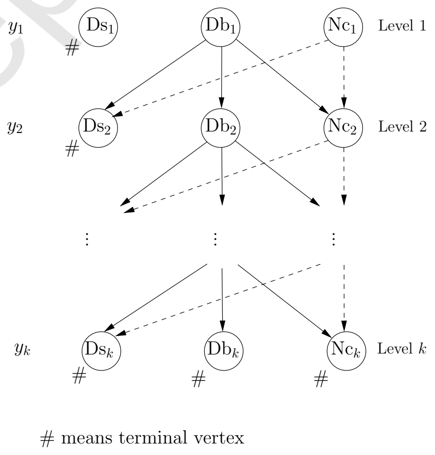


Figure 7: List construction tree (k solutions to be inserted, $y_1 \prec \cdots \prec y_k$).

```
Function LabelingProcessC($\mathcal{N}_t$)
    Input: Node list, $\mathcal{N}_t$
    Result: Set of labels, $L = \{L(x) = (l_p(x), l_n(x)) | x \in \mathcal{N}_t\}$
 1  foreach $x \in \mathcal{N}_t$ do
 2      ListAssignmentC($x$);
 3      $L(x) = (0, 0)$;
 4  for $j = 1$ to $k$ do
 5      foreach $x \in {}^{j-1}L_j \cup L_j^{j-1}$ with $l_p(x) = 0$ do
 6          if $x \in {}^{j-1}L_j$ then
 7              $l_p(x) = 2j$;
 8          else
 9              $l_p(x) = 2j + 1$;
10          AscLabelC($x, j, L$);
11  for $j = k$ to $1$ do
12      foreach $x \in {}^{j+1}L^j \cup L_{j+1}^j$ with $l_n(x) = 0$ do
13          if $x \in {}^{j+1}L^j$ then
14              $l_n(x) = -2j - 2$;
15          else
16              $l_n(x) = -2j - 3$;
17          DescLabelC($x, j + 1, L$);
```

Once the lists have been constructed, Function LabelingProcessC continues with a labeling process in stages performed from $j = 1$ to k in the case of Function AscLabelC (lines 4 to 10), and from $j = k$ to 1 in the case of Function DescLabelC[2] (lines 11 to 17). This means that every node x of the IDG will receive one label, $L(x) = (l_p(x), l_n(x))$, which will let us identify the sets of Eq.(2) and so, they will help us insert solutions $y_1, \ldots, y_k$ into the IDG. In the case of Function AscLabelC, it considers one by one the elements in ${}^{j-1}L_j$ and L_j^{j-1} not positively labeled. If the node belongs to ${}^{j-1}L_j$ the function labels it with $2j$; and if the node belongs to L_j^{j-1}, with $2j + 1$. Then, the function proceeds by labeling all its non-labeled ascendants with 2. If a node labeled with $2j$ or $2j + 1$ is found it is switched to 2; but if any other positive label is found, they are kept and from then on its ancestors are not examined because they have already been considered in a previous stage. This process guarantees that the nodes of the previous lists are not relabeled.

At the end of the labeling process, the following labels have been assigned:

- $B_1^* = B_1^{*3}$ nodes with $l_p(x) = 3$; and for $j = 2, \ldots, k$, B_j^{*2} nodes with $l_p(x) = 2j$ and B_j^{*3} nodes with $l_p(x) = 2j + 1$.
- For $j = 1, \ldots, k-1$, b_j^{*2} nodes with $l_n(x) = -2j - 2$ and b_j^{*3} nodes with $l_n(x) = -2j - 3$; and $b_k^* = b_k^{*3}$ nodes with $l_n(x) = -2k - 3$.

4.2. Insertion procedure

In case that y_j, $j = 1, \ldots, k$, were to be added to the initial IDG in a sequential way by using iteratively the original algorithm (Algorithm 1), sets B_j^* and b_j^* could not be the same as the ones identified with the simultaneous labeling procedure. This is so since as $y_1 \prec \cdots \prec y_k$, some of the y_j could be in sets B_i^* for $i > j$, if the sequential insertion started with y_1; or some of the y_j could be in sets b_i^* for $i < j$, if the sequential insertion started with y_k (see Fig. 3).

[2]Function DescLabelC is symmetrical to Function AscLabelC changing *incident in x* to *salient from x*, *start(a)* to *end(a)*, l_p to l_n, and multiplying the labels by -1.

```
Function ListAssignmentC($x_0$)
    Input: Node to be classified, $x_0$
    Result: Lists to which $x_0$ belongs
1 if $\exists x_p$ parent of $x_0$ in the IDG already classified then
2     $posLevel = j$ such that $x_p \in L_j^{j-1} \cup {}^{j-1}L_j$;
3     $negLevel = j$ such that $x_p \in L_j^{j-1} \cup {}^{j}L^{j-1}$;
4 else
5     $posLevel = negLevel = 1$;
6 $preState = state = Db$;
7 $j = negLevel$;
8 while $j \leq k$ & $state \neq Ds$ do
9     if $x_0 \prec y_j$ then
10        $state = Ds$;
11        if $preState == Db$ then
12            Incorporate $x_0$ to $L_j^{j-1}$;
13        else
14            Incorporate $x_0$ to ${}^{j-1}L_j$;
15    else if $x_0 \not\prec y_j$ then
16        if $preState == Db$ then
17            Incorporate $x_0$ to ${}^{j}L^{j-1}$;
18            $j = \max\{j+1, posLevel\}$;
19        else
20            $j = j + 1$;
21        $preState = Nc$;
22    else //$x_0 \succ y_j$
23        $preState = Db$;
24        if $j$=$k$ then
25            Incorporate $x_0$ to $L_{k+1}^{k}$;
26        $j = j + 1$;
```

Function AscLabelC(x_0, j, L)

```
Input: Initial node, $x_0$; level, $j$; set of labels, L;
Result: Updated set of labels, L
$NodeList = \{x_0\}$;
while $NodeList \neq \emptyset$ do
    Extract $x$ from $NodeList$;
    foreach arc $a$ incident in $x$ do
        if $l_p(start(a)) = 2j$ or $l_p(start(a)) = 2j + 1$ then
            $l_p(start(a)) = 2$;
        else if $l_p(start(a)) = 0$ then
            $l_p(start(a)) = 2$;
            $NodeList = NodeList \cup \{start(a)\}$;
```

In the simultaneous insertion procedure, as set B_j^* has been decomposed depending on the relation with solution y_{j-1}, for the insertion of y_j it is necessary to identify only the nodes of B_j^* which will have a direct arc with y_j, which are those nodes non-comparable with (B_j^{*2}) and dominated by (B_j^{*3}) y_{j-1}; for those nodes of B_j^* that dominate y_{j-1} (B_j^{*1}), the domination relation with y_j will be reflected with a trajectory that goes through y_{j-1}. The case for b_j^* is symmetrical.

In order to insert the new nodes, the same operations as in the previous section need to be carried out. All the arcs between nodes in B_j^* and nodes in b_j^*, $\forall j$ must be removed. Then, the new nodes must be inserted and finally, the relations between nodes in B_j^* and nodes in b_j^* should be expressed through the newly inserted nodes. To carry out these processes the following lemma, whose proof can be found in the Appendix, is enunciated.

Lemma 3. *Once sets B_j^{*h}, b_j^{*l}, $h, l = 2, 3$, $j = 1, \dots, k$, have been identified, in order to insert $y_1, \dots, y_k$, such that $y_1 \prec \cdots \prec y_k$, the following steps need to be followed:*

1. *Remove the arcs: $B_j^{*h} \to b_i^{*l}$ for $i, j = 1, \dots, k$, $j \leq i$ and $h, l = 2, 3$.*
2. *Add the nodes: y_j, $j = 1, \dots, k$.*
3. *Add the arcs:*

- *$B_j^{*h} \to y_j$, for $j = 1, \ldots, k$, $h = 2, 3$.*
- *$y_j \to b_j^{*l}$ for $j = 1, \ldots, k$ and $l = 2, 3$.*
- *And if $\exists j$, $j = 1, \ldots, k$, such that $b_j^{*3} = \emptyset$ ($B_{j+1}^{*3} = \emptyset$) the arc $y_j \to y_{j+1}$ is incorporated.*

After this result, the following theorem that guarantees that the process works correctly can be stated.

Theorem 3. *Let $\mathcal{G}_t$ be an IDG and $\mathcal{G}'_t$ the graph obtained by applying the former procedure for inserting the solutions $y_1, \ldots, y_k$ such that $y_1 \prec \cdots \prec y_k$. Then $\mathcal{G}'_t$ is an IDG (it preserves all the existing relations, adds the new ones and does not reflect fictitious relations).*

Proof. By construction, Function LabelingProcessC ensures the correct identification of sets B_j^{*h} and b_j^{*l}, $h, l = 2, 3$, $j = 1, \ldots, k$. Then, since the procedure for inserting $y_1 \prec \cdots \prec y_k$ is the natural extension of the one proposed in Alberto and Mateo [2], the graph obtained after applying the procedure is an IDG. $\square$

4.3. Example

Let A, B and C, see Fig. 8, be the images in the objective space of three solutions to be inserted into the IDG of Fig. 1. The labels obtained after applying Function LabelingProcessC are also shown in Fig. 8. So, lines 2 and 3 of Algorithm 3 determine the arcs to be removed, (d, g) and (e, g). Then, in line 4, A, B and C are inserted and lines 5 to 8 determine the new arcs to be added, which appear in dashed line in Fig. 8.

5. Theoretical study of the complexity of the proposed algorithms

In this section the computational complexity corresponding to the insertion algorithms proposed is going to be calculated. After that, these complexities will be compared with the ones derived from successively applying Algorithm 1. According to Section 2, let n be the population size; m, the number of arcs of

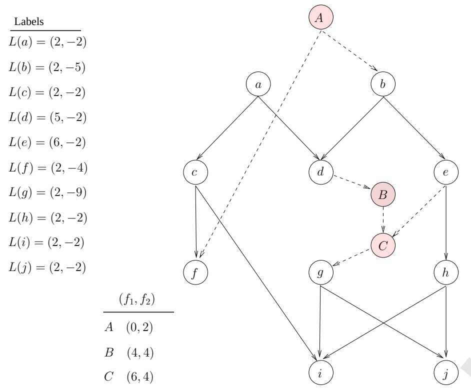


Figure 8: A, B, C dominated solutions to be inserted, set of labels and new added arcs (in dashed line) after applying Algorithm 3.

the initial IDG, which is bounded by n^2; q, the number of objective functions; and k, the number of new solutions to be inserted.

Theorem 4. *Algorithm 2 for inserting k non-comparable solutions has complexity $O(n^2k + nkq)$.*

Proof. In order to demonstrate the result, the different steps of Algorithm 2 are considered. First, in Function LabelingProcessNC, the comparison of each solution of the current population with the set of non-comparable solutions is made at most once, so it reaches a complexity $O(nkq)$. Functions AscLabelNC and DescLabelNC are called at most n times. Inside each of these functions the needed calculations are the following: The list of incident or salient arcs of each node are examined at most k times, since, in order to be included in $NodeList$, a component of the label equal to 0 or 2 must be changed to 1 and $p(x)$ is reduced by, at least, one. Then, the complexity for visiting all the arcs is $O(mk)$. In addition, in each execution of AscLabelNC or DescLabelNC the label vector of any node is examined at most once and these functions are executed at most n times so a complexity of $O(n^2k)$ is added. Summarizing, the labeling process has a complexity of $O(n^2k + nkq)$, since $O(mk)$ is included in $O(n^2k)$ since $m < n^2$.

To determine sets B_j^* and b_j^* the algorithm needs to examine the list of labels of every node and it needs $O(nk)$. For removing the arcs between B_j^* and b_j^*, the list of all salient arcs from nodes of B_j^* are examined. Then, all their ends are tested looking for negative labels in the same components as the start ones, so, at most m arcs are examined and for each one at most k components of the label vector are examined, hence a global complexity of $O(mk)$ is obtained, which again is included in the term of n^2k.

Finally, to add the k new solutions, the algorithm needs $O(k)$ operations and to add the new arcs between B_j^* and y_j and y_j and b_j^* $O(nk)$ since each pair B_j^* and b_j^* contains at most n nodes and the algorithm has to repeat the operation at most for each of the k new solutions.

Adding up, the whole complexity is $O(n^2k + nkq)$. □

Theorem 5. *Algorithm 3 for inserting a sequence of k dominated solutions has complexity $O(n \log n + m + nkq)$.*

Proof. In order to demonstrate the result, the different steps of Algorithm 3 are considered. First, in Function LabelingProcessC, its first step is Function ListAssignmentC, in which each solution of the current population is compared to each y_j, which takes $O(nkq)$. Then, each loop examines, at most once, the arc set of the graph since the solutions are taken from the successive disjoint sets ${}^{j-1}L_j$ and ${}^{j}L^{j-1}$ in order, so it requires $O(m)$ and hence, the whole process is $O(m + nkq)$.

To obtain sets B_j^{*h} and b_j^{*h}, $\forall j$ and $h = 1, 2$, first the labels are examined and each node with a positive or negative label is assigned the level to which it belongs, then the sets are sorted. The examination of the labels takes n steps and the sorting $n \log n$, so this subprocess takes $O(n \log n)$.

To remove all the arcs, since B_j^{*h} and $b_j^{*h}, h = 1, 2$ are disjoint, the salient arcs from nodes in B_j^{*h} are examined only once in order to determine if the end of the arc reaches a node with an appropriate negative label. So the process takes $O(m)$.

Finally k new solutions are added, $O(k)$, and the arcs to reflect the new relationships are added too. Since sets B_j^{*h} and $b_j^{*h}, h = 1, 2$ are disjoint, at most $2n + k$ arcs are added, so the complexity is $O(n + k)$.

Adding up, the complexity becomes $O(n \log n + m + nkq)$. □

The next step is to calculate the complexity associated with repeating k times the process proposed in Alberto and Mateo [2]. The complexity for the insertion of one new solution in a graph with m arcs and n nodes is $O(m+nq)$. To obtain this value it is taken into account that the process of comparing each existing solution with the new one was $O(nq)$, the labeling process was $O(m)$, the determination of the sets B^* and b^* was $O(n)$, the deletion of the arcs between those sets was $O(m)$, the insertion of the new solution was $O(1)$, and the insertion of the new arcs was $O(n)$, resulting in the mentioned complexity $O(m + nq)$.

Theorem 6. *The process of inserting k non-comparable solutions by iteratively applying Algorithm 1 from Alberto and Mateo [2] reaches complexity $O(n^2k + nk^2 + k^3 + nqk)$.*

Proof. First, comparing the nodes with the successive new solutions, assuming that their relation is known, it is only necessary to compare the initial solutions with each of the new ones, which gives a complexity of $O(nqk)$. Although the labeling process depends on the number of arcs and it is going to vary as the new solutions are added, the complexity of this part is $O(mk)$, since none of the new added solutions will be labeled and then, only the original arcs of the graph will participate in this process.

To obtain successive sets B_j^* and b_j^* will consume $O(nk)$, k pairs of sets that only involve the original solutions (the new ones will never belong to those sets).

For the arc deletion process it has to be taken into account that the complexity is of the order of the arc number, since each time the algorithm has to visit all the arcs with the origin in B_j^* to see if they reach a node in the corresponding b_j^*, and there, the new arcs added will also appear. However, the number of arcs varies when a new node is added, so it is difficult to find a value. Therefore, instead of considering the arc number, its maximum will be considered, which is equal to the square of the number of nodes ($m < n^2$), which is equal to $\sum_{j=1}^{k}(n+j)^2 = kn^2 + (2n(k+1)k/2) + (k(k+1)(2k+1)/6)$ which leads $O(k(n^2 + nk + k^2))$.

Finally, the insertion of the new arcs requires (in a trivial way) $O(nk)$, since again no arcs between new nodes are added.

Hence, and taking into account that $O(mk)$ is included in $O(n^2k)$, the global complexity is equal to $O(n^2k + nk^2 + k^3 + nqk)$. □

Theorem 7. *The process of inserting a sequence of k dominated solutions by iteratively applying Algorithm 1 from Alberto and Mateo [2] reaches complexity $O(n^2k + nk^2 + k^3 + nqk)$.*

Proof. Similarly to Theorem 6, the complexity is fixed by the initial comparisons of the new solutions against the current ones, $O(nqk)$, and the operations involving the examination of all the arcs of the graph. As the number of arcs varies with the addition of the new solutions, this number is bound by using the square of the number of nodes and the earlier complexity $O(k(nk+n^2+k^2))$ is obtained. So, globally the process reaches a complexity of $O(n^2k+nk^2+k^3+nqk)$.
□

From the above theorems it can be seen that, in the case of non-comparable solutions, the complexity with the new methodology proposed is equal to or improves the computational complexity of the successive application of the original algorithm; and in the case of dominated solutions the complexity with new methodology improves the one obtained with the original algorithm.

Finally, on the complexity associated with the storage of the different structures used, all of them have the same complexity: Those derived from the storage of the arc lists of the largest graph, which, in theory, would correspond to the graph with $n + k$ nodes.

6. Experiments

In order to test the behavior of the proposed insertion algorithms, firstly, a computational analysis, in which the new proposal is compared with the original one, is carried out. Secondly, a specialized NDS algorithm, ENLU [19], which uses arrays to store the population, is compared to Algorithms 2 and 3. ENLU has been proved by their authors to be more efficient than well-known NDS's in the literature as Deductive Sort [21], Corner Sort [25] and ENS-BS/SS [26]. The ENLU complexity for inserting one newly created offspring into the population is $O(n^2q)$; then, for successively inserting k individuals, operating as in Theorem 6, the complexity will be $O(n^2qk + nqk^2 + qk^3)$. So, compared to Algorithms 2 and 3, the theoretical complexity of the selected NDS algorithm is higher. For instance, assuming that the size of the batch k is a proportion of the population size n, and then $O(k) \equiv O(n)$, the complexity of ENLU reaches $O(n^3q)$ while Algorithm 2 reaches $O(n^3+n^2q)$ and Algorithm 3, $O(n \log n+m+n^2q)$. Therefore, ENLU has a higher complexity by a factor of q when compared to Algorithm 2, and n when compared to Algorithm 3. This fact will be confirmed in the practical experiment, in which ENLU computational time will be the highest.

The performance of the algorithms is measured in terms of execution time and number of Pareto comparisons (comparisons between solutions in order to know their Pareto relation). All the experiments have been executed in an Intel(R) Core(TM) i7 870/2.93GHz, and the authors' programs have been coded in GNU C under Linux (Ubuntu 14.04 Trusty). The Java code of ENLU has been obtained from [18] and literally rewritten in C. The codification has been accomplished assuming that the size of the subpopulations are not known in advance, so, there can be no previous optimization.

To guarantee that the proposed algorithms will confront efficient solution sets of different characteristics, a wide set of test problems proposed in [16] has been used. A short description of the test suite used is shown in Table 1, which has been obtained from the work mentioned. For each problem, a set of random populations with different sizes are built. Then, subpopulations of non-comparable and dominated solutions with different sizes are built for each problem. The summary of these sets can be seen in Table 2.

For each combination of problem and population size, 25 replicas for each

Test problem	Objective functions	No. of variables	Total no. of variables	Separability and modality	Geometry
1. S-ZDT1	f_1	1	30	S, U	convex
	f_2	29		S, U	
2. S-ZDT2	f_1	1	30	S, U	concave
	f_2	29		S, U	
3. S-ZDT4	f_1	1	30	S, U	convex
	f_2	29		S, M	
4. R-ZDT4	$f_{1:2}$	10	10	NS, M	convex
5. S-ZDT6	f_1	1	30	S, M	concave
	f_2	29		S, M	
6. S-DTLZ2	$f_{1:3}$	10	10	S, U	concave
7. R-DTLZ2	$f_{1:3}$	10	10	NS, M	concave
8. S-DTLZ3	$f_{1:3}$	10	10	S, M	concave
9. SYMPART	$f_{1:2}$	30	30	NS, M	concave
10. WFG1	$f_{1:3}$	24	24	S, U	mixed
11. WFG8	$f_{1:3}$	24	24	NS, U	concave
12. WFG9	$f_{1:3}$	24	24	NS, M, D	concave

Table 1: Properties of the test functions. S: Separable; NS: Nonseparable; U: Unimodal; M: Multimodal; D: Deceptive.

PS	300, 500, 1000 and 1500 individuals
SPS	2%, 5%, 10%, 20%, 30%, 40%, 50% of the PS
No. subpopulations	25 for each PS, SPS and problem

Table 2: Characteristics of the populations and subpopulations (set of non-comparable solutions and sequence of dominated solutions) built. PS = population size, SPS = subpopulation size.

subpopulation size are incorporated into the population (and later removed to recover the initial population using the removal algorithm presented in [2]) by using the new algorithms and by repeatedly applying (SPS times) the original algorithm. The insertion execution time (in nanoseconds, Ns.) and the number of Pareto comparisons are recorded in order to compare the algorithms.

For the first experiment, Tables 3 and 5 show the ratio between the time required for inserting the solutions with the new proposal and with the original algorithm. In addition, Tables 4 and 6 show the ratio between the number of Pareto comparisons carried out for the new proposal and with the original algorithm. The rows show the population size (PS); and the columns, the subpopulation size (SPS, number of new solutions) in terms of a percentage of the population size.

For contrasting the performance of both algorithms, a statistical two-sided Wilcoxon signed rank test was performed, with a significance level equal to 0.05. When the value of the table is in *italics* it means that the original algorithm outperforms the new methodology, if the value is in **boldface** it means that both methodologies can be considered equivalent, and finally, when the value is in standard text, it means that the new methodology outperforms the original one.

		SPS								SPS						
PS		2%	5%	10%	20%	30%	40%	50%		2%	5%	10%	20%	30%	40%	50%
300	S_ZDT1	*1.28*	*1.05*	0.92	0.78	0.73	0.65	0.63	S_ZDT2	*1.10*	0.86	0.72	0.61	0.55	0.51	0.49
500		*1.18*	0.94	0.81	0.71	0.67	0.62	0.59		**0.98**	0.74	0.62	0.52	0.50	0.47	0.44
1000		0.95	0.80	0.71	0.59	0.57	0.52	0.48		0.71	0.58	0.51	0.43	0.39	0.35	0.33
1500		0.85	0.69	0.60	0.52	0.48	0.46	0.40		0.61	0.51	0.44	0.37	0.32	0.30	0.27
300	S_ZDT4	**1.03**	0.85	0.70	0.60	0.53	0.50	0.46	R_ZDT4	*1.20*	0.91	0.82	0.69	0.62	0.58	0.54
500		0.87	0.71	0.61	0.53	0.48	0.48	0.43		**1.02**	0.80	0.71	0.63	0.55	0.53	0.51
1000		0.76	0.60	0.50	0.43	0.40	0.36	0.33		0.84	0.71	0.64	0.52	0.48	0.43	0.40
1500		0.61	0.51	0.42	0.37	0.33	0.29	0.29		0.76	0.60	0.52	0.45	0.36	0.34	0.28
300	S_ZDT6	*1.08*	0.84	0.69	0.58	0.51	0.46	0.46	S_DTLZ2	**1.06**	**0.99**	0.92	0.83	0.77	0.74	0.67
500		0.90	0.68	0.57	0.51	0.46	0.44	0.41		**1.03**	**0.96**	0.83	0.81	0.76	0.74	0.64
1000		0.70	0.56	0.47	0.39	0.38	0.33	0.31		0.94	0.81	0.78	0.71	0.63	0.56	0.57
1500		0.61	0.49	0.41	0.35	0.30	0.29	0.25		0.89	0.77	0.69	0.63	0.58	0.57	0.49
300	R_DTLZ2	*1.12*	*1.05*	**0.96**	0.88	0.84	0.78	0.79	S_DTLZ3	**1.03**	0.93	0.94	0.81	0.75	0.73	0.71
500		**1.04**	**0.94**	0.95	0.87	0.82	0.73	0.70		**1.02**	0.96	0.85	0.81	0.78	0.77	0.71
1000		**0.99**	0.92	0.85	0.74	0.70	0.65	0.61		**0.97**	0.84	0.77	0.71	0.66	0.61	0.59
1500		0.92	0.85	0.79	0.71	0.63	0.61	0.59		0.90	0.79	0.72	0.67	0.62	0.58	0.51
300	SYMP.	0.56	0.38	0.31	0.23	0.23	0.22	0.20	WFG1	0.93	0.82	0.74	0.64	0.62	0.54	0.54
500		0.38	0.29	0.23	0.20	0.19	0.18	0.17		0.82	0.76	0.68	0.59	0.58	0.52	0.54
1000		0.29	0.20	0.18	0.16	0.14	0.14	0.14		0.78	0.67	0.57	0.53	0.49	0.47	0.44
1500		0.23	0.18	0.15	0.13	0.13	0.13	0.11		0.67	0.60	0.55	0.48	0.45	0.43	0.39
300	WFG8	0.93	0.84	0.77	0.75	0.69	0.65	0.63	WFG9	0.92	0.77	0.79	0.77	0.75	0.76	0.66
500		0.88	0.82	0.78	0.73	0.70	0.67	0.62		0.87	0.84	0.76	0.72	0.72	0.71	0.70
1000		0.84	0.81	0.76	0.71	0.68	0.64	0.59		0.83	0.75	0.77	0.74	0.74	0.70	0.68
1500		0.81	0.75	0.74	0.69	0.67	0.64	0.59		0.80	0.77	0.72	0.77	0.74	0.68	0.67

Table 3: Ratio of time, in nanoseconds, for the new and original algorithms, for k non-comparable solutions.

In general, and as might be expected, for small values of PS and SPS, the laboriousness required to implement the new algorithm produces little decrease in execution time. For example, in Table 3, situations in which Algorithm 2 is not competitive with respect to Algorithm 1 appear only on 8 occasions, most of which are with PS= 300 and SPS= 2%. It can be observed that when SPS> 5%, Algorithm 2 is never surpassed by Algorithm 1. It is also observed that as PS and SPS increase, the gain obtained when applying Algorithm 2 increases progressively. For example, the percentage of times in which Algorithm 2 reduces the execution time by half or more is 26.79% (90 out of 336 in which the entries are less than or equal to 0.5 in Table 3); being 16.07% when PS= 300 or 500 and 37.5% when PS= 1000 or 1500. This behavior can also be observed graphically, as an example, in the first column of Fig. 9, for problem S_ZDT2[3], where the difference in execution time between Algorithm 2 (in blue) and Algorithm 1 (in red) increases progressively as PS and SPS increase.

		SPS								SPS						
PS		2%	5%	10%	20%	30%	40%	50%		2%	5%	10%	20%	30%	40%	50%
300	S_ZDT1	0.99	0.97	0.94	0.89	0.85	0.80	0.77	S_ZDT2	0.99	0.96	0.93	0.87	0.81	0.76	0.72
500		0.99	0.97	0.94	0.89	0.84	0.80	0.76		0.98	0.96	0.92	0.86	0.81	0.76	0.71
1000		0.99	0.97	0.94	0.88	0.84	0.79	0.76		0.98	0.96	0.92	0.86	0.80	0.75	0.71
1500		0.99	0.97	0.94	0.88	0.84	0.80	0.74		0.98	0.96	0.92	0.85	0.80	0.75	0.68
300	S_ZDT4	0.99	0.96	0.93	0.87	0.81	0.76	0.71	R_ZDT4	0.99	0.96	0.93	0.87	0.81	0.76	0.72
500		0.98	0.96	0.93	0.86	0.80	0.76	0.71		0.98	0.96	0.93	0.87	0.80	0.76	0.73
1000		0.98	0.96	0.92	0.86	0.80	0.75	0.71		0.98	0.96	0.93	0.86	0.81	0.75	0.71
1500		0.98	0.96	0.92	0.85	0.80	0.74	0.70		0.98	0.96	0.92	0.86	0.78	0.74	0.66
300	S_ZDT6	0.99	0.96	0.92	0.86	0.80	0.75	0.71	S_DTLZ2	0.99	0.98	0.95	0.91	0.87	0.83	0.80
500		0.98	0.96	0.92	0.85	0.80	0.74	0.70		0.99	0.98	0.95	0.91	0.87	0.83	0.79
1000		0.98	0.96	0.92	0.85	0.79	0.74	0.69		0.99	0.97	0.95	0.91	0.87	0.83	0.80
1500		0.98	0.96	0.92	0.85	0.79	0.74	0.68		0.99	0.98	0.95	0.91	0.86	0.83	0.79
300	R_DTLZ2	0.99	0.98	0.95	0.91	0.87	0.83	0.80	S_DTLZ3	0.99	0.98	0.95	0.91	0.87	0.84	0.80
500		0.99	0.98	0.95	0.91	0.87	0.83	0.80		0.99	0.98	0.95	0.91	0.87	0.83	0.80
1000		0.99	0.98	0.95	0.91	0.87	0.83	0.80		0.99	0.98	0.95	0.91	0.87	0.83	0.80
1500		0.99	0.98	0.95	0.91	0.87	0.83	0.80		0.99	0.98	0.95	0.91	0.87	0.83	0.80
300	SYMP.	0.94	0.87	0.78	0.57	0.54	0.48	0.42	WFG1	0.99	0.97	0.95	0.90	0.86	0.82	0.79
500		0.91	0.84	0.73	0.60	0.51	0.44	0.40		0.99	0.97	0.95	0.90	0.86	0.82	0.79
1000		0.92	0.81	0.70	0.57	0.44	0.39	0.35		0.99	0.97	0.95	0.90	0.86	0.82	0.79
1500		0.93	0.83	0.70	0.51	0.46	0.42	0.34		0.99	0.97	0.95	0.90	0.86	0.82	0.78
300	WFG8	0.99	0.98	0.96	0.91	0.88	0.84	0.81	WFG9	0.99	0.98	0.96	0.91	0.88	0.84	0.81
500		0.99	0.98	0.96	0.91	0.88	0.84	0.81		0.99	0.98	0.95	0.91	0.87	0.84	0.81
1000		0.99	0.98	0.95	0.91	0.87	0.84	0.81		0.99	0.98	0.95	0.91	0.87	0.84	0.81
1500		0.99	0.98	0.95	0.91	0.87	0.84	0.81		0.99	0.98	0.95	0.91	0.87	0.84	0.81

Table 4: Ratio of number of Pareto comparisons, for the new and original algorithms, for k non-comparable solutions.

[3]For the rest of problems, the behavior is quite similar to S_ZDT2, so, those figures are not shown

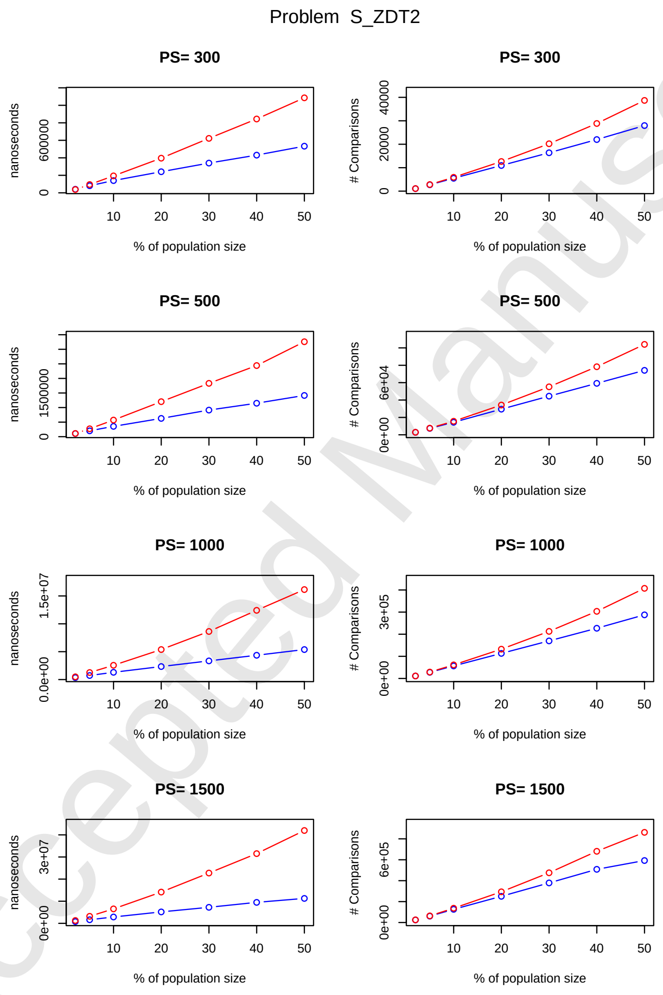


Figure 9: Simultaneous insertion of k non-comparable solutions. Evolution of the time, in nanoseconds (left) and the number of Pareto comparisons (right), for problem S_ZDT2, in function of the SPS (expressed in terms of % of PS), for PS=300, 500, 1000 and 1500. Algorithm 1 in red, Algorithm 2 in blue.

As for the number of Pareto comparisons made, in Table 4 it can be observed that Algorithm 2 is significantly better than Algorithm 1 because in no case does Algorithm 1 surpass or equal the new one. Although the gain in the number of Pareto comparisons is not as great as the gain obtained in time, more than 50% of the time (concretely 53.57%) Algorithm 2 produces an improvement of at least 10%. It is also observed that the difference between the two algorithms increases progressively as PS and SPS grow. We should also note that the reduction in the number of Pareto comparisons would be greater since, in the calculation of these values with Algorithm 1, we have subtracted the quantity $k(k-1)/2$ that are the comparisons resulting from comparing the new solutions $y_1, \dots, y_k$ with each other. In this way, both methodologies use the same amount of information, that is, both use the fact that the new solutions are non-comparable. If these comparisons were included in the case of the original algorithm, a quadratic behavior could be observed in the second column of Fig. 9 instead of a quasi-linear behavior as a function of SPS.

In the case of inserting a sequence of k dominated solutions, the results in Tables 5 and 6 show that the improvements when using Algorithm 3 instead of Algorithm 1 are even better than when inserting k non-comparable solutions, except when considering PS moderate and SPS equal to 2%. Similar to what happened in the previous case, for moderate values of PS and SPS, the complexity of Algorithm 3 makes it impossible to outperform Algorithm 1 with respect to the time needed (Table 5). However, from there, for incorporations of 5% or more solutions, in almost all situations (except in 2 out of 288) Algorithm 3 is equal to or better than Algorithm 1 (in 5 out of 288 times both algorithms are equivalent, and in 281 of 288 Algorithm 3 is better than Algorithm 1).

Also, in the first column of Fig. 10 for problem S_ZDT2, it can be observed that, in this case, Algorithm 3 has a steeper improvement with respect to Algorithm 1 than in the previous case.

As for the number of Pareto comparisons, Table 6, we can see that, except for 2 occasions in which both algorithms are equivalent, in all the others Algorithm 3 is significantly better than Algorithm 1. Moreover, in this case the improvement

		SPS								SPS						
PS		2%	5%	10%	20%	30%	40%	50%		2%	5%	10%	20%	30%	40%	50%
300	S_ZDT1	*2.61*	**1.09**	0.56	0.33	0.24	0.20	0.16	S_ZDT2	*2.18*	0.80	0.49	0.29	0.19	0.16	0.14
500		*1.52*	0.67	0.35	0.23	0.16	0.14	0.12		*1.43*	0.52	0.30	0.18	0.14	0.11	0.10
1000		0.90	0.34	0.23	0.14	0.11	0.09	0.08		0.65	0.29	0.17	0.12	0.09	0.08	0.08
1500		0.52	0.24	0.14	0.09	0.08	0.07	0.06		0.43	0.20	0.13	0.09	0.08	0.07	0.06
300	S_ZDT4	*1.99*	0.92	0.48	0.30	0.19	0.17	0.15	R_ZDT4	*1.86*	0.85	0.50	0.26	0.20	0.17	0.14
500		*1.20*	0.58	0.35	0.19	0.15	0.12	0.09		*1.24*	0.53	0.31	0.18	0.13	0.11	0.09
1000		0.71	0.34	0.18	0.12	0.09	0.07	0.06		0.66	0.30	0.18	0.11	0.09	0.07	0.07
1500		0.44	0.24	0.14	0.08	0.07	0.05	0.05		0.46	0.20	0.11	0.08	0.06	0.06	0.05
300	S_ZDT6	*2.02*	0.86	0.48	0.27	0.22	0.16	0.14	S_DTLZ2	*2.65*	**1.07**	0.58	0.32	0.26	0.20	0.17
500		*1.19*	0.64	0.33	0.21	0.14	0.12	0.12		*1.67*	0.70	0.38	0.25	0.18	0.14	0.13
1000		0.67	0.30	0.19	0.12	0.10	0.08	0.07		0.95	0.41	0.26	0.15	0.13	0.11	0.10
1500		0.40	0.19	0.11	0.09	0.07	0.07	0.06		0.67	0.29	0.18	0.12	0.09	0.09	0.08
300	R_DTIZ2	*2.54*	**1.04**	0.55	0.32	0.24	0.18	0.16	S_DTLZ3	*2.51*	*1.09*	0.56	0.34	0.24	0.20	0.17
500		*1.53*	0.61	0.36	0.22	0.15	0.13	0.12		*1.69*	0.69	0.37	0.22	0.17	0.14	0.13
1000		0.82	0.36	0.21	0.12	0.11	0.08	0.08		**0.98**	0.38	0.22	0.14	0.11	0.09	0.08
1500		0.54	0.25	0.12	0.09	0.07	0.07	0.06		0.60	0.29	0.16	0.10	0.09	0.07	0.06
300	SYMP.	*1.47*	0.61	0.33	0.18	0.12	0.10	0.08	WFG1	*2.27*	**1.10**	0.62	0.35	0.27	0.22	0.20
500		0.82	0.34	0.19	0.10	0.08	0.06	0.06		*1.52*	0.70	0.41	0.27	0.20	0.16	0.14
1000		0.39	0.18	0.10	0.06	0.04	0.04	0.03		0.88	0.40	0.27	0.15	0.13	0.12	0.10
1500		0.25	0.11	0.07	0.04	0.03	0.03	0.02		0.68	0.26	0.18	0.12	0.10	0.09	0.08
300	WFG8	*2.51*	*1.19*	0.70	0.46	0.36	0.32	0.28	WFG9	*2.44*	**1.05**	0.59	0.36	0.27	0.24	0.21
500		*1.62*	0.78	0.49	0.34	0.28	0.25	0.22		*1.57*	0.73	0.42	0.25	0.21	0.18	0.16
1000		**0.99**	0.47	0.32	0.23	0.19	0.16	0.15		0.87	0.42	0.24	0.16	0.12	0.12	0.10
1500		0.72	0.34	0.22	0.17	0.13	0.12	0.11		0.64	0.31	0.18	0.12	0.09	0.08	0.08

Table 5: Ratio of time, in nanoseconds, for the new and original algorithms, for a sequence of k dominated solutions.

is greater than in the case of non-comparable solutions. For example, in 88.39% of the occasions it reduces to less than half the number of Pareto comparisons made; and a reduction of 75% occurs in 22.62% of the occasions.

In the second column of Fig. 10 we can see the evolution of the difference in number of Pareto comparisons with each algorithm (Algorithm 3 in blue and Algorithm 1 in red) as a function of PS and SPS. In this case, it can be seen that the separation between the two lines is more pronounced than in the case of non-comparable solutions.

As we have said before, in the second experiment, Algorithms 2 and 3 are compared to ENLU. With respect to the first objective, time needed for the insertions, ENLU always presents worse behavior than the proposed algorithms, except in 3 out of 672 cases (12 test problems $\times$ 4 PS $\times$ 7 SPS and 2 algorithms)

		SPS								SPS						
PS		2%	5%	10%	20%	30%	40%	50%		2%	5%	10%	20%	30%	40%	50%
300	S_ZDT1	0.51	0.40	0.35	0.31	0.29	0.27	0.28	S_ZDT2	0.58	0.55	0.45	0.33	0.38	0.35	0.32
500		0.39	0.31	0.31	0.26	0.25	0.24	0.22		0.49	0.49	0.43	0.35	0.33	0.29	0.29
1000		0.28	0.31	0.26	0.26	0.24	0.22	0.21		0.54	0.51	0.42	0.38	0.43	0.33	0.36
1500		0.32	0.28	0.28	0.25	0.25	0.22	0.22		0.51	0.51	0.47	0.49	0.40	0.38	0.34
300	S_ZDT4	0.71	0.55	0.44	0.36	0.31	0.30	0.30	R_ZDT4	0.83	0.62	0.54	0.48	0.40	0.42	0.42
500		0.62	0.49	0.33	0.33	0.32	0.25	0.24		0.65	0.56	0.47	0.42	0.41	0.42	0.39
1000		0.45	0.33	0.34	0.28	0.27	0.25	0.28		0.60	0.47	0.45	0.45	0.33	0.36	0.37
1500		0.48	0.37	0.38	0.32	0.30	0.25	0.30		0.52	0.48	0.45	0.34	0.39	0.35	0.31
300	S_ZDT6	0.68	0.67	0.48	0.50	0.46	0.35	0.37	S_DTLZ2	0.36	0.31	0.27	0.26	0.22	0.21	0.22
500		0.68	0.46	0.49	0.40	0.43	0.36	0.31		0.30	0.28	0.23	0.24	0.22	0.20	0.21
1000		0.56	0.47	0.39	0.41	0.36	0.33	0.32		0.31	0.25	0.23	0.23	0.23	0.21	0.20
1500		0.53	0.50	0.46	0.37	0.40	0.39	0.34		0.26	0.27	0.24	0.23	0.20	0.21	0.19
300	R_DTLZ2	0.47	0.38	0.37	0.35	0.32	0.34	0.35	S_DTLZ3	0.42	0.32	0.30	0.30	0.27	0.29	0.26
500		0.39	0.38	0.36	0.36	0.34	0.34	0.30		0.32	0.28	0.29	0.26	0.24	0.25	0.23
1000		0.34	0.34	0.35	0.34	0.29	0.30	0.23		0.27	0.28	0.26	0.25	0.25	0.22	0.21
1500		0.30	0.28	0.31	0.27	0.26	0.26	0.29		0.24	0.22	0.22	0.24	0.23	0.20	0.21
300	SYMP.	**1.08**	0.72	0.58	0.48	0.39	0.35	0.31	WFG1	0.58	0.39	0.37	0.38	0.33	0.34	0.36
500		**0.95**	0.74	0.62	0.46	0.42	0.39	0.36		0.47	0.39	0.37	0.34	0.37	0.35	0.27
1000		0.88	0.73	0.59	0.52	0.48	0.44	0.35		0.35	0.34	0.32	0.34	0.31	0.28	0.32
1500		0.85	0.67	0.62	0.53	0.46	0.39	0.35		0.33	0.35	0.37	0.32	0.32	0.25	0.30
300	WFG8	0.57	0.51	0.48	0.45	0.43	0.41	0.40	WFG9	0.44	0.36	0.33	0.30	0.27	0.27	0.27
500		0.47	0.42	0.41	0.38	0.36	0.35	0.33		0.32	0.27	0.25	0.24	0.23	0.23	0.22
1000		0.32	0.30	0.28	0.27	0.26	0.25	0.23		0.22	0.19	0.18	0.18	0.16	0.16	0.16
1500		0.26	0.24	0.23	0.22	0.21	0.20	0.20		0.18	0.16	0.15	0.14	0.14	0.13	0.13

Table 6: Ratio of number of Pareto comparisons, for the new and original algorithms, for a sequence of k dominated solutions.

in which ENLU behaves slightly better, and 2 cases in which both can be considered equivalent[4]. In Figs. 11 and 12 left we show the results for S_ZDT2, which is representative of the behavior of all the problems. On the other hand, with regard to the number of Pareto comparisons accomplished, the results depend on the type of inserted solutions. For the case of non-comparable solutions, ENLU behaves significantly better than Algorithm 2 in all the problems except for SYMPART, where Algorithm 2 exhibits a slightly better performance. The general behavior of ENLU versus Algorithm 2 can be observed in Fig. 11 right. When the case of dominated solutions is considered, the behavior is the oppo-

[4]As the summary of the experiment is reduced to stating that in only 5 cases the behavior of Algorithms 2 and 3 do not outperform ENLU, the equivalent tables to Tables 3 and 5 are not shown.

site: Algorithm 3 outperforms ENLU in all the cases except in 9 out of the 336 situations, in which both algorithms can be considered equivalent. In Fig. 12 this general behavior, which can be extrapolated to the rest of problems, can be observed in the particular case of problem S_ZDT2.[5]

Hence, taking into account the results obtained with these experiments and, with regard to the time, we have verified the improvement in the practical behavior of the new proposed algorithms in comparison with Algorithm 1 and ENLU, in a wide battery of test problems.

With respect to the number of Pareto comparisons both, Algorithms 2 and 3, progressively improve Algorithm 1, when PS and/or SPS increase. When comparing to ENLU and considering sets of non-comparable solutions, ENLU shows a better performance which gets higher as PS and/or SPS increases. However when considering sets of dominated solutions the behavior drastically changes, being Algorithm 3 significantly better than ENLU.

7. Conclusions and future research lines

In this work two new algorithms for the solution batch insertion into a population of a MOEA have been presented. Their theoretical complexities have been calculated and they have also been compared with the original proposal (Alberto and Mateo [2]) and a well-known NDS algorithm (Li et al. [19]). In these comparisons, the new proposals generally outperform the others when the execution time is considered; only when the population and batch sizes are really small, the newly proposed algorithms could not be that competitive due to the laboriousness needed for the implementation. When taking into account the number of Pareto comparisons, the new proposal shows, in general, the best behavior, except for non-comparable solutions for which the new algorithm is better than the original one but shows a worse performance than ENLU. In general, the higher the population and batch sizes, the higher the outperformance of the new proposal.

[5]The corresponding tables are omitted for the same reason as before.

The next step to carry out to study and corroborate the advantages in the use of these new proposals will be to implement these tools in some of the existing algorithms. A first interesting idea could be to test it in parallel genetic algorithms [5] as, for instance, the ones based on island models, in which the different demes can be stored by using particular IDGs. Then, in any of the IDGs, classical search network algorithms can be used to obtain paths (batches of dominated solutions) or to obtain non-connected nodes (batches of non-comparable solutions). Once these batches are identified, the different migration schemas that these kinds of algorithms carry out could be accomplished in a more efficient way by using the proposed algorithms.

A second idea could be to use the new methodology in standard PDMOEAs (such as NSGA-II) in which, once the population of descendants has been obtained, batches of non-comparable or dominated solutions extracted from this set would be identified for their insertion into the current population (before the selection process). These batches can be built by using simple rules, for instance, two descendants are taken; if they are non-comparable (or if one dominates the other) a third one is taken and compared. If the three solutions are non-comparable (or if they are a sequence of dominated solutions), a forth one is considered. The process is considered finished when one solution is found which dominates or is dominated by at least one of the previous solutions (or if a solution breaks the sequence of dominated solutions). Subsequently, Algorithm 2 (or Algorithm 3) is applied and the process is repeated with the remaining descendants. Another possibility, instead of using simple rules, would be to build a new IDG in which to store the descendants and extract from it sets of dominated/non-comparable solutions to be inserted, as we have commented for parallel algorithms. In the first possibility, unlike what we have done in this work, the new algorithms could be programmed taking into account that the size of the batches would be smaller, and therefore, the algorithms could be optimized taking into consideration this fact and the improvement could become greater than the one reached in the experiments presented.

Appendix. Theoretical results

Lemma 2. Given y_i and y_j such that $y_i \not\prec y_j$, then $B_i \cap b_j = \emptyset$. As a consequence, $B_i^* \cap b_j^* = \emptyset$.

Proof. By contradiction, let's assume that $B_i \cap b_j \neq \emptyset$, i.e., $\exists x \in B_i \cap b_j$. So, since $x \in b_j$, it holds that $y_j \prec x$; and since $x \in B_i$, it holds that $x \prec y_i$. Then, by the transitivity of the domination relation, it holds that $y_j \prec y_i$, but, however, $y_i \not\prec y_j$, which is a contradiction. $\square$

Lemma 3. Once sets B_j^{*h}, b_j^{*l}, $h, l = 2, 3$, $j = 1, \dots, k$, have been identified, in order to insert $y_1, \dots, y_k$, such that $y_1 \prec \cdots \prec y_k$, the following steps need to be followed[6]:

1. Remove the arcs: $B_j^{*h} \to b_i^{*l}$ for $i, j = 1, \dots, k$, $j \leq i$ and $h, l = 2, 3$.
2. Add the nodes: $y_j, j = 1, \dots, k$.
3. Add the arcs:
   - $B_j^{*h} \to y_j$, for $j = 1, \dots, k$, $h = 2, 3$.
   - $y_j \to b_j^{*l}$ for $j = 1, \dots, k$ and $l = 2, 3$.
   - And if $\exists j$, $j = 1, \dots, k$, such that $b_j^{*3} = \emptyset$ (and, by Lemma 6, $B_{j+1}^{*3} = \emptyset$) the arc $y_j \to y_{j+1}$ is incorporated.

Proof. 1. As stated in Alberto and Mateo [2], to insert y_j into an IDG, all the arcs $B_j^* \to b_j^*$, $j = 1, \dots, k$ need to be removed. By Lemma 5, it holds that $B_j^* \subset \bigcup_{i=1}^{j} \left(B_i^{*2} \dot\cup B_i^{*3}\right)$ and $b_j^* \subset \bigcup_{i=j}^{k} \left(b_i^{*2} \dot\cup b_i^{*3}\right)$, and sets B_i^{*2}, B_i^{*3}, b_i^{*2} and b_i^{*3} were identified in the labeling process with labels $l_p(x) = 2i$, $l_p(x) = 2i + 1$, $l_n(x) = -2j - 2$ and $l_n(x) = -2j - 3$, respectively. So, if all arcs $B_j^{*h} \to b_i^{*l}$ for $i, j = 1, \dots, k$, $j \leq i$ and $h, l = 2, 3$ are removed, arcs $B_j^* \to b_j^*$, $j = 1, \dots, k$ have also certainly been removed.

But now however, there is the doubt of whether too many arcs have been removed, since the arcs between larger sets of nodes have been removed.

[6]For the proof of Lemma 3, Lemmas 4 to 6 need to be stated.

To see that it is not so, suppose that $\exists x_1 \rightarrow x_2$ in the initial IDG with $x_1 \in \bigcup_{h \leq j} \left(B_h^{*2} \dot{\cup} B_h^{*3}\right) \setminus B_j^*$ and $x_2 \in \bigcup_{h \geq i} \left(b_h^{*2} \dot{\cup} b_h^{*3}\right) \setminus b_i^*$, $j \leq i$, and then, $x_1 \in B_j$ and $x_2 \in b_i$. This means that x_1 is an ancestor but not a parent of y_j and x_2 is a descendant but not a child of y_i. Then $\exists x', x''$ such that $x_1 \prec \cdots \prec x' \prec y_j \prec y_i \prec x'' \prec \cdots \prec x_2$, and then, the domination relation $x_1 \prec \cdots \prec x' \prec x'' \prec \cdots \prec x_2$ has to be reflected in the initial IDG by means of a trajectory $x_1 \hookrightarrow x' \hookrightarrow x'' \hookrightarrow x_2$.

Therefore, in the initial IDG there exist both the arc $x_1 \rightarrow x_2$ and the trajectory $x_1 \hookrightarrow x_2$, which is a contradiction with Lemma 1. Hence, the arc $x_1 \rightarrow x_2$ does not exist in the initial IDG and so it is not possible to remove it.

2. Add the nodes y_j, $j = 1, \ldots, k$, to the IDG.
3. Once the previous arcs are removed and the new nodes y_j, $j = 1, \ldots, k$, added, the domination relations broken must be reestablished, i.e., all the arcs $B_j^* \rightarrow y_j$ and $y_j \rightarrow b_j^*$, $j = 1, \ldots, k$ must be incorporated. However, taking into account the decomposition of sets B_j^* and b_j^*, only the following steps need to be carried out:
   - Add all arcs $B_j^{*h} \rightarrow y_j$ and $y_j \rightarrow b_j^{*l}$ for $j = 1, \ldots, k$ and $h, l = 2, 3$.
   - For $j = 1, \ldots, k$, there is no need to add arcs $B_j^{*1} \rightarrow y_j$ (or the arcs $y_j \rightarrow b_j^{*1}$) since all the domination relations between nodes in B_j^{*1} (b_j^{*1}) and y_j must go through y_{i-1} (y_{j+1}) since nodes in B_j^{*1} dominate y_{j-1} by definition (nodes in b_j^{*1} are dominated by y_{j+1} by definition), and those arcs have been added in the previous step. The addition of the arcs $B_j^{*1} \rightarrow y_j$ ($y_j \rightarrow b_j^{*1}$) would cause them to be redundant.
   - Finally, if $\exists j$, $j = 1, \ldots, k-1$, such that $b_j^{*3} = \emptyset$, and then, by Lemma 6, $B_{j+1}^{*3} = \emptyset$, the arc $y_j \rightarrow y_{j+1}$ must be added. This is so because there will be no parent of y_{j+1} in the IDG but y_j (and there will be no child for y_j in the IDG but y_{j+1}), and for reflecting the domination relation $y_j \prec y_{j+1}$ the arc $y_j \rightarrow y_{j+1}$ is added. □

Lemma 4. *If $B_j^{*1} \neq \emptyset$ then $B_j^{*1} \subseteq B_{j-1}^*$. Also, if $b_j^{*1} \neq \emptyset$ then $b_j^{*1} \subseteq b_{j+1}^*$.*

Proof. Let's assume that there is a $x_h \in B_j^{*1} \subseteq B_{j-1}$ node with $x_h \notin B_{j-1}^*$. The fact that $x_h \in B_{j-1}$ together with $x_h \notin B_{j-1}^*$, implies that there exists at least one $x_l \in B_{j-1}^*$ such that $x_h \prec x_l \prec y_{j-1} \prec y_j$, which is a contradiction because $x_h \in B_j^*$. □

Lemma 5. $B_j^{*1} \subset \bigcup_{i<j} \left(B_i^{*2} \dot\cup B_i^{*3}\right)$. *Symmetrically,* $b_j^{*1} \subset \bigcup_{i>j} \left(b_i^{*2} \dot\cup b_i^{*3}\right)$.

Proof. Set B_j^* was decomposed as $B_j^* = B_j^{*1} \dot\cup B_j^{*2} \dot\cup B_j^{*3}$. By Lemma 4 it holds that $B_j^{*1} \subset B_{j-1}^*$, and also $B_{j-1}^* = B_{j-1}^{*1} \dot\cup B_{j-1}^{*2} \dot\cup B_{j-1}^{*3}$. So, repeating the process it holds that $B_j^* = B_j^{*1} \dot\cup B_j^{*2} \dot\cup B_j^{*3} \subset \bigcup_{i=1}^{j} \left(B_i^{*2} \dot\cup B_i^{*3}\right)$, and since the sets are disjoint, $B_j^{*1} \subset \bigcup_{i=1}^{j-1} \left(B_i^{*2} \dot\cup B_i^{*3}\right)$. □

Lemma 6. $b_j^{*3} = \emptyset \Leftrightarrow B_{j+1}^{*3} = \emptyset$.

Proof. ($\Rightarrow$) Let's imagine that $b_j^{*3} = \emptyset$ and assume that $\exists x_h \in B_{j+1}^{*3}$. By definition of B_{j+1}^{*3} it holds that $y_j \prec x_h \prec y_{j+1}$. As $b_j^{*3} = \emptyset$, x_h cannot be a child of y_j so, a trajectory $y_j \to u_1 \hookrightarrow x_h$ must exist with node u_1 being a child of y_j, which dominates y_{j+1}. Then, $u_1 \in b_j^{*3}$ and a contradiction appears since $b_j^{*3} = \emptyset$.

($\Leftarrow$) This situation is symmetrical. □

Acknowledgments

This research has been supported by the Research Group of Gobierno de Aragón E58 and by the spanish Ministerio de Economía, Industria y Competitividad under grant MTM2016-77015-R (AEI/FEDER, UE).

References

[1] Agrawal, R., Jagadish, H. V., 1987. Direct algorithms for computing the transitive closure of database relations. Proc. 13th Int. Conf. Very Large Data Bases, 255–266.

[2] Alberto, I., Mateo, P., 2004. Representation and management of MOEA populations based on graphs. Eur. J. Oper. Res. 159 (1), 52–65.

[3] Altwaijry, N., El Bachir Menai, M., nov 2012. Data Structures in Multi-Objective Evolutionary Algorithms. J. Comput. Sci. Technol. 27 (6), 1197–1210.

[4] Belohlavek, R., 2008. Introduction to formal concept analysis. Tech. rep., Department of Computer Science, Faculty of Science, Palacký University, Olomuc.

[5] Cantu-Paz, E., 2000. Efficient and Accurate Parallel Genetic Algorithms. Kluwer Academic Publishers, Norwell, MA, USA.

[6] Chen, X., 2001. Pareto Tree Searching Genetic Algorithm. Approaching Pareto Optimal Front by Searching Pareto Optimal Tree. Tech. Rep. 1999, Department of Computer Science, Nankai University, Tianjin, China.

[7] Coello, C., 2017. List of references on evolutionary multiobjective optimization.
URL `http://www.lania.mx/ccoello/EMOO/EMOObib.html`

[8] Coello, C., Lamont, G., Van Veldhuizen, D., 2007. Evolutionary algorithms for solving multi-objective problems, 2nd Edition. Springer, Berlin.

[9] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. 6 (2), 182–197.

[10] Eiben, A., Smith, J., 2007. Introduction to evolutionary computing. Springer, Berlin.

[11] Fieldsend, J., Everson, R., Singh, S., 2003. Using Unconstrained Elite Archives for Multiobjective Optimization. IEEE Trans. Evol. Comput. 7 (3), 305–323.

[12] Fonseca, C., Fleming, P., 1993. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In: Proc. fifth Int. Conf. Genet. algorithms. Vol. 423. Citeseer, pp. 416–423.

[13] Glasmachers, T., 2017. A Fast Incremental BSP Tree Archive for Non-dominated Points. In: Trautmann, H., Rudolph, G., Klamroth, K., Schütze, O., Wiecek, M., Jin, Y., Grimme, C. (Eds.), 9th Int. Conf. Evol. Multi-Criterion Optim. EMO 2017. Lect. Notes Comput. Sci. Vol. 10173. Springer, Cham, pp. 252–266.

[14] Goldberg, D., 1989. Genetic algorithms in search, optimization and machine learning. Addison Wesley, Reading, Massachusetts.

[15] Habenicht, W., 1983. Quad Trees, a Datastructure for Discrete Vector Optimization Problems. Lect. Notes Econ. Math. Syst. 209, 136–145.

[16] Huang, V., Qin, A., Deb, K., Zitzler, E., Suganthan, P., Liang, J., Preuss, M., Huband, S., 2007. Problem definitions for performance assessment of multi-objective optimization algorithms. Tech. rep., Nanyang Technological University, Singapore.

[17] Klamt, S., Flassig, R. J., Sundmacher, K., 2010. TRANSWESD: Inferring cellular networks with transitive reduction. Bioinformatics 26 (17), 2160–2168.

[18] Li, K., 2016. ENLU Codes.
URL `https://github.com/JerryI00/publication_codes/tree/master/ENLU`

[19] Li, K., Deb, K., Zhang, Q., Zhang, Q., 2016. Efficient Nondomination Level Update Method for Steady-State Evolutionary Multiobjective Optimization. IEEE Trans. Cybern. (Accepted for publication).

[20] Li, X., Du, G., 2013. BSTBGA: A hybrid genetic algorithm for constrained multi-objective optimization problems. Comput. Oper. Res. 40 (1), 282–302.

[21] McClymont, K., Keedwell, E., 2012. Deductive sort and climbing sort: new methods for non-dominated sorting. Evol. Comput. 20 (1), 1–26.

[22] Schütze, O., 2003. A new data structure for the nondominance problem in multi-objective optimization. In: 2nd EMO. pp. 509–518.

[23] Shi, C., Yan, Z., Shi, Z., Zhang, L., 2010. A fast multi-objective evolutionary algorithm based on a tree structure. Appl. Soft Comput. 10 (2), 468–480.

[24] Talbi, E., Basseur, M., Nebro, A., Alba, E., jan 2012. Multi-objective optimization using metaheuristics: non-standard algorithms. Int. Trans. Oper. Res. 19 (1-2), 283–305.

[25] Wang, H., Yao, X., 2014. Corner sort for pareto-based many-objective optimization. IEEE Trans. Cybern. 44 (1), 92–102.

[26] Zhang, X., Tian, Y., Cheng, R., Jin, Y., 2015. An Efficient Approach to Non-dominated Sorting for Evolutionary Multi-objective. IEEE Trans. Evol. Comput. 19 (2), 1–15.

[27] Zhou, A., Qu, B.-Y., Li, H., Zhao, S.-Z., Suganthan, P., Zhang, Q., mar 2011. Multiobjective evolutionary algorithms: A survey of the state of the art. Swarm Evol. Comput. 1 (1), 32–49.

[28] Zitzler, E., Laumanns, M., Thiele, L., 2001. SPEA2: Improving the strength Pareto evolutionary algorithm. Tech. rep., Computer Engineering and Networks Laboratory, ETH Zentrum, Zurich.

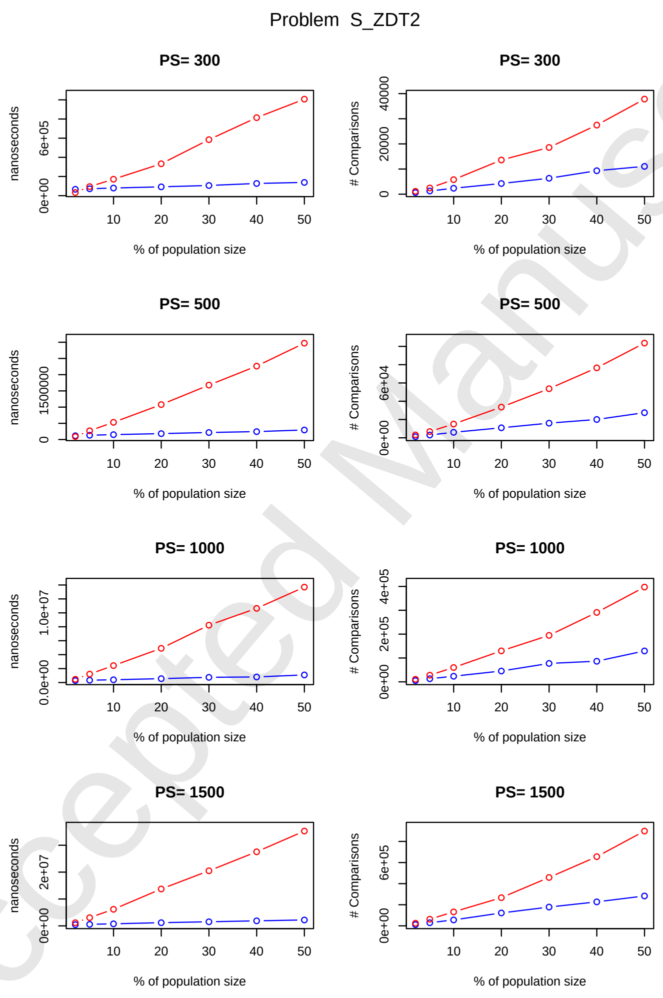


Figure 10: Simultaneous insertion of a sequence of k dominated solutions. Evolution of the time, in nanoseconds (left) and the number of Pareto comparisons (right), for problem S_ZDT2, in function of the SPS (expressed in terms of % of PS), for PS=300, 500, 1000 and 1500. Algorithm 1 in red, Algorithm 3 in blue.

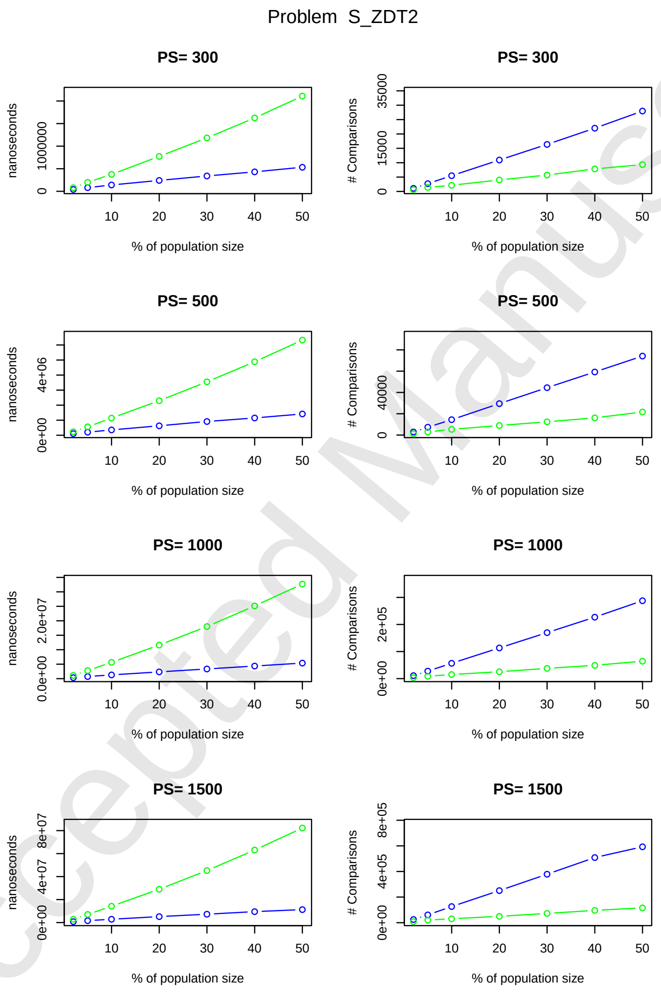


Figure 11: Simultaneous insertion of a sequence of k non-comparable solutions. Evolution of the time, in nanoseconds (left) and the number of Pareto comparisons (right), for problem S_ZDT2, in function of the SPS (expressed in terms of % of PS), for PS=300, 500, 1000 and 1500. ENLU in green, Algorithm 2 in blue.

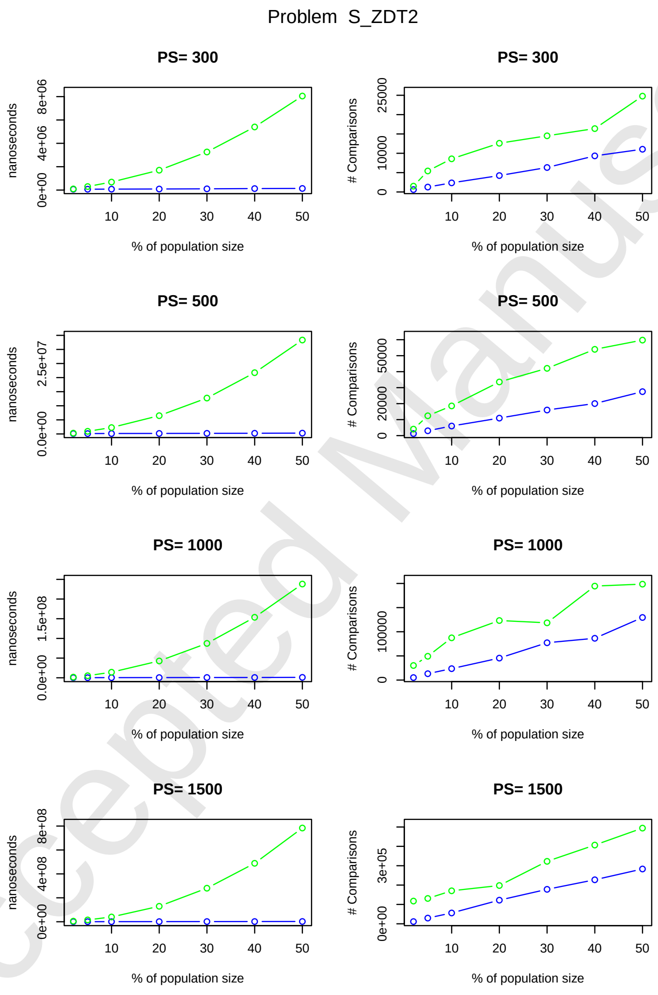


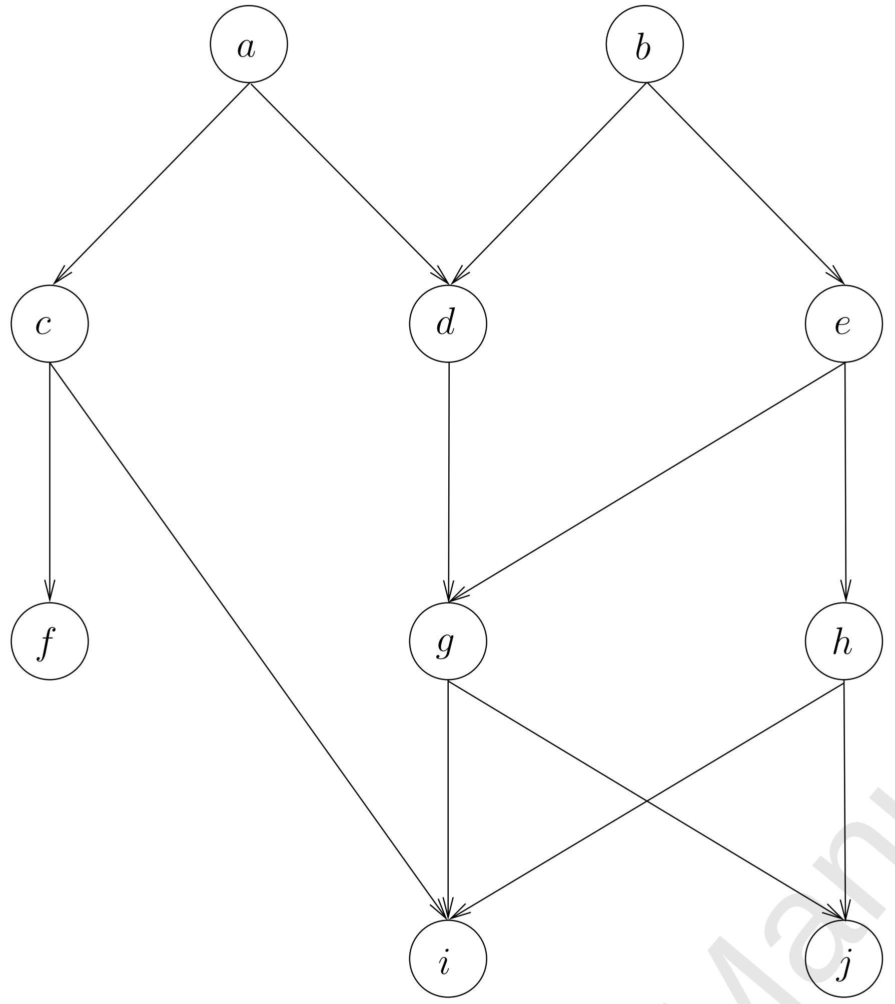
Figure 12: Simultaneous insertion of a sequence of k dominated solutions. Evolution of the time, in nanoseconds (left) and the number of Pareto comparisons (right), for problem S_ZDT2, in function of the SPS (expressed in terms of % of PS), for PS=300, 500, 1000 and 1500. ENLU in green, Algorithm 3 in blue.

Highlights

An improved version of the graph-based structure used for manipulating populations of Multi-Objective Evolutionary Algorithms is presented.
The improvement consists in the simultaneous insertion of solutions (instead of a single one) into the created graph structure.
The experiments carried out show that the new proposal saves computational time and number of Pareto comparisons carried out for the insertion.

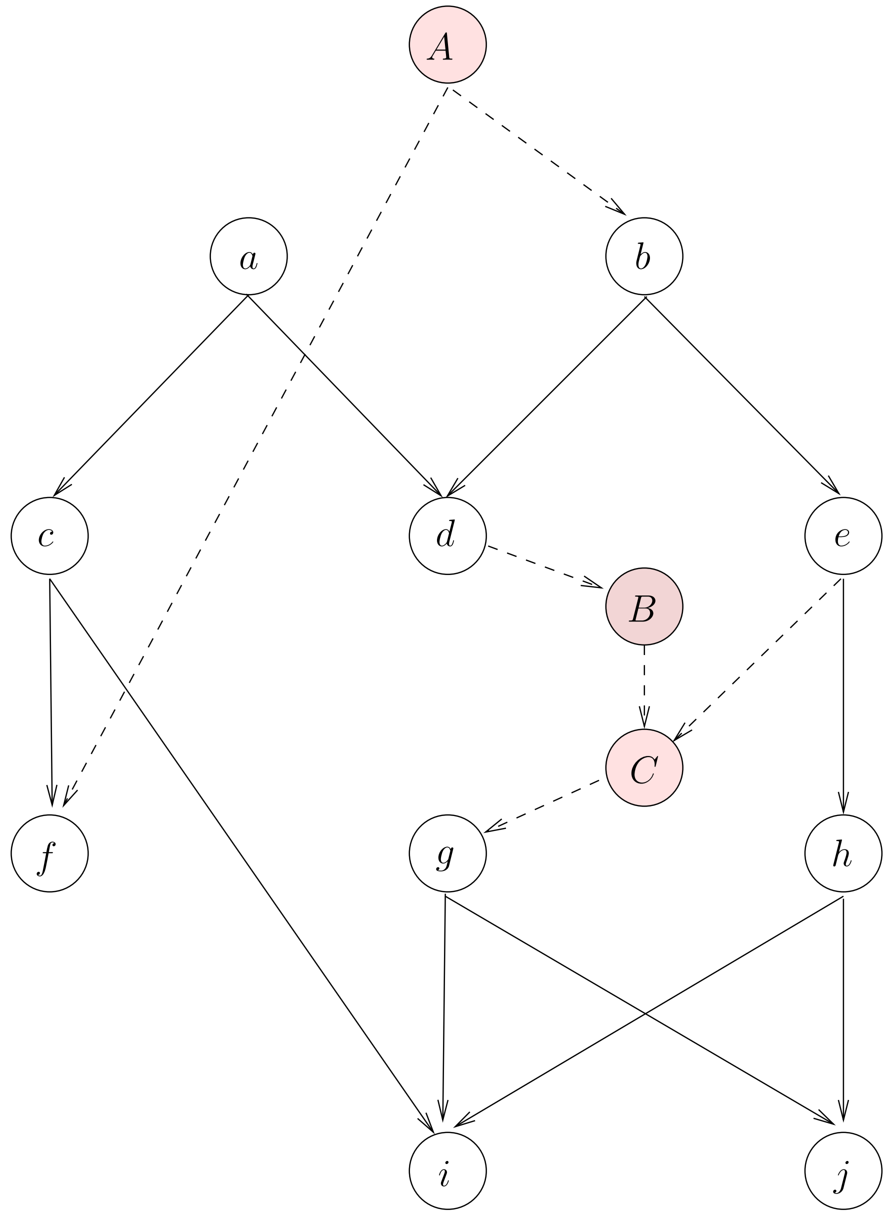
Initial IDG

	(f_1, f_2)
a	$(0, 3)$
b	$(3, 0)$
c	$(1, 7)$
d	$(4, 3)$
e	$(5, 1)$
f	$(2, 9)$
g	$(6, 5)$
h	$(7, 2)$
i	$(8, 8)$
j	$(9, 6)$



Simultaneous insertion of three dominated solutions

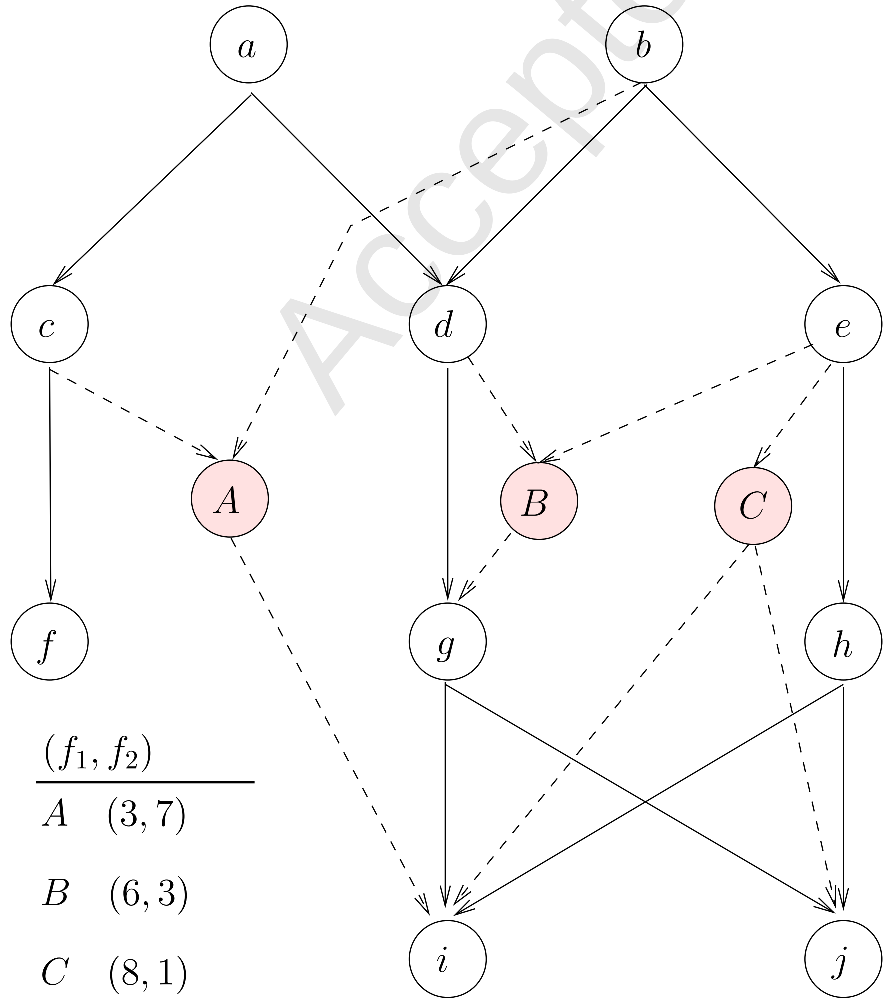
Labels
$L(a) = (2, -2)$
$L(b) = (2, -5)$
$L(c) = (2, -2)$
$L(d) = (5, -2)$
$L(e) = (6, -2)$
$L(f) = (2, -4)$
$L(g) = (2, -9)$
$L(h) = (2, -2)$
$L(i) = (2, -2)$
$L(j) = (2, -2)$



	(f_1, f_2)
A	$(0, 2)$
B	$(4, 4)$
C	$(6, 4)$

Simultaneous insertion of three non−comparable solutions

Labels
$L(a) = (1, 1, 3)$
$L(b) = (2, 1, 1)$
$L(c) = (2, 3, 3)$
$L(d) = (3, 2, 3)$
$L(e) = (3, 2, 2)$
$L(f) = (3, 3, 3)$
$L(g) = (3, -2, 3)$
$L(h) = (3, 3, 3)$
$L(i) = (-2, -1, -2)$
$L(j) = (3, -1, -2)$



	(f_1, f_2)
A	$(3, 7)$
B	$(6, 3)$
C	$(8, 1)$

PII:	S1568-4946(17)30656-7
DOI:	https://doi.org/doi:10.1016/j.asoc.2017.10.042
Reference:	ASOC 4537
To appear in:	*Applied Soft Computing*
Received date:	10-3-2017
Revised date:	21-9-2017
Accepted date:	26-10-2017