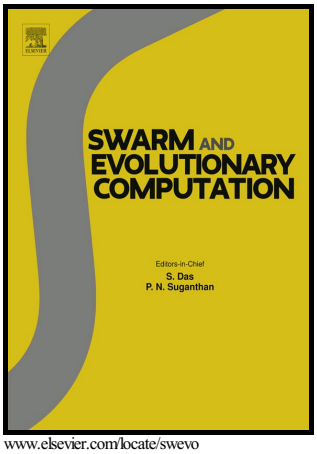
LONSA: A Labeling-Oriented Non-dominated Sorting Algorithm for Evolutionary Many-Objective Optimization

R.F. Alexandre, C.H.N.R. Barbosa, J.A. Vasconcelos



www.elsevier.com/locate/swevo

PII: S2210-6502(17)30680-6
DOI: http://dx.doi.org/10.1016/j.swevo.2017.08.003
Reference: SWEVO301

To appear in: *Swarm and Evolutionary Computation*

Received date: 3 December 2014
Revised date: 23 July 2017
Accepted date: 12 August 2017

Cite this article as: R.F. Alexandre, C.H.N.R. Barbosa and J.A. Vasconcelos, LONSA: A Labeling-Oriented Non-dominated Sorting Algorithm for Evolutionary Many-Objective Optimization, *Swarm and Evolutionary Computation,* http://dx.doi.org/10.1016/j.swevo.2017.08.003

LONSA: A Labeling-Oriented Non-dominated Sorting Algorithm for Evolutionary Many-Objective Optimization

R.F. Alexandre[a,b,*], C.H.N.R. Barbosa[b], J.A. Vasconcelos[a]

[a] *Federal University of Minas Gerais, Evolutionary Computation Laboratory and Postgraduate Program in Electrical Engineering*
Av. Antônio Carlos, 6627 Pampulha, CEP: 31270-901 Belo Horizonte, MG, Brazil - Tel.: +55-31-34093426
[b] *Federal University of Ouro Preto, Institute of Exact and Applied Sciences*
Rua 37, no 115, Loanda, CEP: 35931-026 João Monlevade, MG, Brazil

Abstract

Multiobjective algorithms are powerful in tackling complex optmization problems mathematically represented by two or more conflicting objective functions and their constraints. Sorting a set of current solutions across non-dominated fronts is the key step for the searching process to finally identify which ones are the best solutions. To perform that step, a high computational effort is demanded, especially if the size of the solution set is huge or the mathematical model corresponds to a many-objective problem. In order to overcome this, a new labeling-oriented algorithm is proposed in this paper to speed up the solution-to-front assignment by avoiding usual dominance tests. Along with this algorithm, called Labeling-Oriented Non-dominated Sorting Algorithm (LONSA), the associated methodology is carefully detailed to clearly explain how the classification of the solution set is successfully achieved. This work presents a comparison between LONSA and other well-known algorithms usually found in the literature. The simulation results have shown a better performance of the proposed algorithm against nine chosen strategies in terms of computational time as well as number of comparisons.

Keywords: Multiobjective Optimization, Non-Dominance, Many-Objective, Solution Labeling.

1. Introduction

Over the last decade, an increasing number of real-world multidimensional optimization problems have been addressed by researchers with the aid of several multiobjective evolutionary algorithms (MOEAs) [1–3] enabled by data processing power of today's computers. According to the literature, those classes of optimization problems, known as many-objective problems (MOPs) [4–6] in short, have been dealt by many MOEAs [7–10] designed to be resilient in the task of filtering optima among solutions scattered throughout a highly challenging n-dimensional search space. Most of those MOEAs were based on Pareto-dominance concept such as MOGA [1], NSGA-II [7], NSGA-III [4, 5], SPEA2 [8], PAES [11], and MODEA [12]. All of them are able of achieving the same result, considering a set of solutions as input and the non-dominated fronts as output. However, each algorithm has its own computational cost to bound the non-dominated fronts. According to Zhou et al. [2], the multipurpose framework based on Pareto dominance [3] was applied to the NSGA-II [7] as well as other existing MOEAs with different levels of adaption of the underlying principle. Nevertheless, the computing time of those algorithms have been the bottleneck which is severely degraded by the efficency of how sorting is performed on the solutions distributed across a finite number of fronts. This burden is responsible for the major quota of the time spent during the whole optimization process, straightly influenced by the model complexity features (number of objectives, number and type of the decision variables) and algorithm parameters (population size, number of generations). Therefore, a sorting procedure applied on identifying such fronts which has low-computational cost and is free of redundant operations regardless the landscape of the search space is worthy and it is still an open challenge. Recently, additional strategies based on niching, archiving, tree structures, or lists were proposed in the literature to overcome or mitigate that lim-

*Corresponding author
Email addresses: rfalexandre@decea.ufop.br (R.F. Alexandre), cbarbosa@deelt.ufop.br (C.H.N.R. Barbosa), jvasconcelos@ufmg.br (J.A. Vasconcelos)

List of symbols.

Symbols	Definition	Symbols	Definition
$\mathcal{F}$	Set of non-dominated fronts.	p	Number of non-dominated fronts.
$\mathcal{F}^*$	The estimated Pareto front.	n	Cardinality of the solution set.
Ω	Feasible region of the search space.	m	Number of objectives.
$\mathcal{S}$	A given set of solutions.	γ	Number of the solutions per front.
β	Subset with index of objectives.	τ	Front iterator.
$\mathcal{M}$	Matrix $\in \mathbb{R}^{m \times n}$.	i, j	General purpose iterators.
x, u, v, s	Vector in objective space (solution).	$\prec$	Dominance operator.
x^*, α	Non-dominated solution.		

itation. Corner Sort [13], Efficient Non-dominated Sort [14], Fast Non-dominated Sort [7], Get Dominance Tree [15], Deductive Sort [16], and Arenas Principle [17] are examples of the newest approaches, just to name a few.

The computational complexity of the first algorithms has achieved $O(m.n^3)$ [18] wherein m denotes the number of objectives and n equals the cardinality of the set to be sorted. Newer algorithms are less demanding, with their computational complexity varying from $O(m.n^2)$ to $O(m.n.log(n))$ [14]. The main features of the latest strategies which led to complexity reduction are briefly discussed in Section 2.2.

In this work, we propose a new sorting method applicable to MOPs aiming at the identification of the non-dominated fronts: Labeling-Oriented Non-dominated Sorting Algorithm (LONSA). LONSA consists of two distinct sub-procedures tailored for sorting in two-objective or in a m-objective problems, respectively. Like other well-known methods, LONSA can be used to categorize/classify solution sets into non-dominated fronts when coupled with an evolutionary algorithm based on the concept of Pareto dominance. The proposed algorithm is simple but effective in separating n solutions in a set of p non-dominated fronts correctly, regardless their distribution over a n-dimensional space.

This work is organized as follows. In Section 2, important definitions are provided for the sake of clarity such as solution dominance and Pareto front. Additionally, relevant features of the existing methods are discussed. The conceived algorithm LONSA is detailed in Section 3. The principle of the sorting scheme is discriminated in that section and it is demonstrated for two and many-objective cases. In Section 4, the test bed applied to validate our proposition is presented as well as numeric comparisons including performances of LONSA and nine other methods found in the literature. The work conclusions are provided in the last section.

2. Background

Identifying non-dominated fronts from a arbitrary set of solutions consists in sorting that set by means of a dominance relation, previously defined. It may be better understood if we consider n solutions belonging to a given original set $\mathcal{S}$ to be sorted on p non-dominated fronts, where each is denoted as $\mathcal{F}_\tau$, with $\tau = \{1, ..., p\}$. According to the assumed non-dominance criteria, every non-dominated solution s^j existing in the current set $\mathcal{S}'$ must be assigned to any front $\mathcal{F}_\tau$ ($\mathcal{F}_\tau = \mathcal{F}_\tau \cup s^j$) and then removed from $\mathcal{S}'$ to obtain $\mathcal{S}''$, written as $\mathcal{S}'' = \mathcal{S}' - s^j$. Those sequential assignments continue until all solutions in $\mathcal{S}$ have been designated to some front and that set becomes empty. In Figure 1, a set of 21 solutions was sorted into three non-dominated fronts by using the Pareto dominance. In that same figure, the solution α dominates any solution found in the hatched region labeled as Ω.

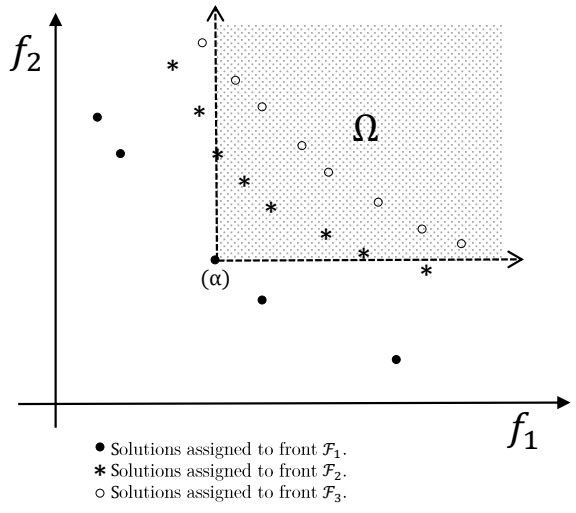


Figure 1: Example of a solution set containing 21 solutions ($\|\mathcal{S}\|$=21) associated to a two-objective minimization problem, sorted into three non-dominated fronts.

2.1. Basic definitions

In order to be able to compare a pair of solutions during the selection process, the proposed algorithm depends on the non-dominance definition. Here, we mathematically provide some concepts in multiobjective optimization that are useful to this work. These definitions were already presented by many works [3, 18].

Definition 1. *Dominance relation: A vector* $\boldsymbol{u} = \{u_1, ..., u_m\}$*, for all* $\boldsymbol{u} \in \mathbb{R}^m$*, is said to dominate other vector* $\boldsymbol{v} = \{v_1, ..., v_m\}$*, for all* $\boldsymbol{v} \in \mathbb{R}^m$*, mathematically denoted by* $\boldsymbol{u} \prec \boldsymbol{v}$*, if* $\boldsymbol{u}$ *is not worse than* $\boldsymbol{v}$ *for all the m objectives but strictly better at least in one of them, such as*

$$\forall\, i \in \{1, ..., m\},\ u_i \leq v_i \wedge \exists\, j \in \{1, ..., m\} \mid u_j < v_j \tag{1}$$

Definition 2. *Pareto-optimal Solution: A given feasible vector* $\boldsymbol{x}^*$ *is considered to be Pareto-optimal if there is no other feasible vector* $\boldsymbol{x}$ *such that* $\boldsymbol{x} \prec \boldsymbol{x}^*$*. In this case, the vector* $\boldsymbol{x}^*$ *is also said to be a non-dominated or efficient solution.*

Definition 3. *Pareto-optimal Set* $\mathcal{F}^*$ *: Let* $\boldsymbol{\Omega}$ *be the representative set of the feasible region located inside the objective space. The Pareto-optimal set* $\mathcal{F}^*$ *can be defined as:*

$$\mathcal{F}^* = \{x^* \in \mathcal{S} \mid \neg\exists\, x \in \mathcal{S}, \mathcal{S} \subseteq \boldsymbol{\Omega}, x \prec x^*\} \tag{2}$$

The three definitions just seen are incorporated by most of the MOEAs which are based on Pareto dominance. By adopting dominance criteria, algorithms for sorting solutions in non-dominated fronts are needed to be part of such MOEAs. The main goal of those algorithms is the determining of well-dispersed solutions as close as possible to the real Pareto-optimal set $\mathcal{F}^*$. Therefore, this paper presents a new strategy for sorting solutions in non-dominated fronts that also uses compatible dominance criteria with the existent ones.

2.2. Non-dominated Solution Sorting Algorithms

The procedure of classifying solutions into non-dominated fronts is one of the most time-consuming tasks involved in multiobjective optimization [10, 14, 15]. Designing an efficient strategy to accomplish such task is too relevant to the overall performance of MOEAs, especially when dealing with MOPs. In this section, we briefly discuss the main features of the algorithms applied for sorting efficient solutions into non-dominated fronts. Nine of them were chosen as basis of performance comparison to our proposal.

Many schemes have already been proposed in the literature to mitigate this problem, since the efficiency of a classification method depends on the data structure used by MOEA. Niching, hashing, archiving, stacking and listing are types of such structures used to develop efficient sorting strategies. In addition, several methods have adopted specific approaches that consider sorting of solutions wherein the values of objectives are taken as the sorting criterion [13, 14, 19]. Other works rely on some properties, such as symmetry and transitivity, to speed up comparisons among solutions and avoid repetitive operations [10, 15, 16, 20]. Consequently, the use of those properties allows the algorithms to avoid unnecessary comparisons. For example, given a solution set $\mathcal{S} = \{a, b, c\}$, we can say that if $\boldsymbol{a} \prec \boldsymbol{b}$, $\boldsymbol{b} \prec \boldsymbol{c}$, then $\boldsymbol{a} \prec \boldsymbol{c}$.

Recent studies have presented new methods for sorting m-objective solutions in non-dominated fronts by using approaches which require the ordering according to the values of the respective objectives. The first work which applied this approach was presented by Jensen [19]. It has used a simple comparison proper to two-objective problems and the divide-and-conquer paradigm to problems with three or more objectives. It is worth to mention that this method does not yield correct results if there are solutions with identical objective values in the solution set $\mathcal{S}$. Furthermore, the performance worsens significantly as the number of objectives increases. Following this way, another work is proposed by Zhang et al. [14] presenting two algorithms: i) Efficient Non-dominated Sort Sequential Search (ENS-SS), and ii) Efficient Non-dominated Sort - Binary Search (ENS-BS). The ENS determines to which front a solution s $(s \in \mathcal{S})$ is belonged, one by one. If s is dominated by any solution of front $\mathcal{F}_\tau$, where $\tau = \{1, ..., p\}$, the value of τ will be incremented and the procedure will be repeated while τ is lower than p. If there is no solution in $\mathcal{F}_\tau$ that dominates s, then it will be included in $\mathcal{F}_\tau$. If τ equals p, a new front will be created to add s to $\mathcal{F}_{\tau+1}$, and the value of p will be incremented. The two algorithms were compared to other three known as Deductive Sort [16], Arena's Principle [17] and Fast Non-dominated Sort (FNDS) [7]. Results have shown that ENS-SS and ENS-BS have achieved superior performance. However, that perceived performance is considerably degraded once the number of objectives has increased leading to computational cost around $O(m.n^2)$ for the worst case.

Handing et al. [13] have proposed an algorithm called Corner Sort (CS) that sorts solutions by their objectives values. It starts its procedure taking extreme solutions, i.e., those that are at the beginning of the sorted arrays.

Table 1: Related works with proposals to solution sorting.

Method	Complexity (Worst Case)	Remark
Dominance Depth + Dominance Rank [8]	$O(m.n^2)$	Most used method, but with high computational cost.
Fast Nondominated Sorting [7]	$O(m.n^2)$	Also often used but with high computational cost.
Non-Dominated-Sort + ND-Helper [19]	$O(m.\log^{m-1}(n))$	Uses recursive procedure.
Better-nondominated-sort [10]	$O(m.n^2)$	Uses symmetry and transitivity properties.
Get Dominance Tree [15]	$O(m.n.\log(n))$	Adopted the divide-and-conquer mechanism.
Non-Dominated Tree [20]	–	Uses a Binary tree.
Mak Tree [21]	$O(n.\log(m))$	Uses a Binary tree. Restricted to two-objective problems.
Arena's Principle [17]	$O(m.n^2)$	Performs a tournament between solutions.
Deductive and Climbing Sort [16]	$O(m.n^2)$	Stores information of comparisons made.
Corner Sort [13]	$O(m.n^2)$	Sorts solutions and starts by comparing extreme solutions.
ENS [14]	$O(m.n^2)$	Sorts solutions using sequential or binary search strategies.

Note: m is the number of objectives and n is the cardinality of the solution set.

Considering one certain objective, this method assumes that extreme solutions tend to dominate more solutions. Then, solutions dominated by extreme solutions are neglected in next iterations so that possible unnecessary comparisons are avoided. According to the authors, the method requires $m.(n-1)$ comparisons considering each objective at a time. However, the computing time spent for sorting solutions in MOPs badly affects the performance of the CS even if dominance comparisons are avoided.

Other methods found in the literature have arranged solutions into data structures known as trees [22]. Among these methods we can mention the Non-Dominated Tree (NDT) [20], Mak Tree (MT) [21] and the Get Dominance Tree (GDT) [15]. Generally, the idea is to orderly fill a tree with solutions so that the algorithm is able to find all the dominated solutions related to a given node belonging to that tree. Thus, when inserting a new solution in that tree, dominance operations among solutions can be avoided. For example, the GDT method associates to the same node of the tree the solutions that do not have any dominance relationship. Only the following property must be guaranteed: the solutions placed in the right subtree of a node r dominate that ones belonging to r and, still, the solutions placed in the left subtree of r are dominated by the ones belonging to r. Even being a complex structure, the GDT must keep tree properties whenever new solutions are added. The authors did not present the computational cost of GDT. But results have shown better performance of it when compared to FNDS.

A summary of the algorithms found in the literature is shown in Table 1. Among those methods, we highlight Fast Non-Dominated Sorting (FNDS) [7] and Better Non-dominated Sorting (BNS) [8] which are used by most of the cited MOEAs like NSGA-II [7] and SPEA2 [8].

3. Description of the proposed algorithm

In this section, a novel non-dominated solution sorting method named Labeling-Oriented Non-dominated Sorting Algorithm (LONSA) will be presented. LONSA has a distinct reasoning to sort a given set of solutions by taking two objectives at a time to compare and evaluate partially or totally the dominance among solutions. To accomplish this, two arrays are built ($array_x$ and $array_y$) in a ascending order such that each position of these arrays contains one solution plus a standard identifying label "not evaluated - **#1**" (see Figure 2). After the initial step, LONSA sweeps both arrays in search of label exchanges along solutions, if necessary, to "to be inserted in F - **#2**" or "to be removed from array - **#3**" in order to identify whether each solution belongs to the current front or not. The criteria used to choose one of the three available labels for each solution are carefully detailed in Sections 3.1 and 3.2 where some didactic examples are also provided. In the end of each iteration, solutions labeled as **#2** are moved away to the current front and the algorithm starts over a new iteration unless all solutions have already be included into some front. The computational gain of LONSA is achieved during sweeps in the following situations: i) when solutions labeled as **#2** remain in the arrays and, therefore, they do not belong to the current front or; ii) when solutions with label **#3** are found indicating that

they do not belong to the current front. Thus, dominance tests are performed only for the solutions with label **#1**.

For a two-objective problem, LONSA uses LONSA2Obj(.) - its sub-procedure tailored for the simplest case of multiobjective optimization. Otherwise, when dealing with MOPs ($\boldsymbol{m > 2}$), LONSA uses a sub-procedure called LONSAMany-Obj(.) and it can be easily explained after the basics of the conceived methodology are understood, essentially coded into LONSA2Obj(.). LONSA is outlined in the Algorithm 1 which consists of creating two sorted arrays[1] in an ascending order for the values of the first and second objectives of the solutions. Firstly, label **#1** is assigned to all solutions and a hash table [22] is created aiming at the efficiency of any search for elements in those arrays. LONSA checks up the number of objectives $\boldsymbol{m}$ and it runs the proper sub-procedure: LONSA2Obj(.) for $\boldsymbol{m = 2}$ or LONSAManyObj(.) for $\boldsymbol{m > 2}$. The sub-procedures are responsible of identifying all the non-dominated solutions, iterating over $\boldsymbol{array_x}$ and $\boldsymbol{array_y}$, removing them from $\boldsymbol{\mathcal{S}}$. The non-dominated solutions are returned to the callee function and they are assigned to $\mathcal{F}_\tau$. The procedure continues until all solutions have been removed from $\boldsymbol{\mathcal{S}}$.

Algorithm 1: Pseudocode for sorting fronts of non-dominated solutions. (LONSA)

Input: $\mathcal{S}$: solution set / m: number of objectives
Output: $\mathcal{F}$: non-dominated fronts

1 **Begin**
2 $array_x \leftarrow sort(\mathcal{S}, 1)$ `/*sort using the objective 1.*/`
3 $array_y \leftarrow sort(\mathcal{S}, 2)$ `/*sort using the objective 2.*/`
4 $hashTable \leftarrow makeHash(\mathcal{S})$
5 $\tau \leftarrow 1$
6 **while** $\mathcal{S} \neq \emptyset$ **do**
7 **if** $m = 2$ **then**
8 $\mathcal{F}_\tau \leftarrow$ LONSA2Obj($\mathcal{S}, array_x, array_y, hashTable$)
9 **else**
10 $\mathcal{F}_\tau \leftarrow$ LONSAManyObj($\mathcal{S}, array_x, array_y, hashTable$)
11 **end**
12 $\tau \leftarrow \tau + 1$
13 **end**
14 **return** $\mathcal{F}$
15 **End**

In Sections 3.1 and 3.2, details are provided about the sub-procedures LONSA2Obj(.) and LONSAMany-Obj(.), respectively. The features of each sub-procedure are presented as well as didactic examples in order to aid the overall understanding.

[1]Due to performance issues, the Quicksort algorithm, proposed by Hoare [23], is encouraged since its computational complexity is $\boldsymbol{O(n.log(n))}$. Thus, the computational time is doubled if the sorting procedure is applied for both objectives.

3.1. Sub-procedure for two-objective optimization

Given a set of solutions $\boldsymbol{\mathcal{S}}$ associated to a two-objective search space, we propose a specific algorithm (see Algorithm 2) to discriminate $\boldsymbol{\mathcal{S}}$ in non-dominated fronts. This procedure consists on labeling the solutions into two possible groups to differ: (i) those inserted in the current non-dominated front (label **#2**); and (ii) those excluded from the arrays (label **#3**). If previously labeled as **#1**, each solution $\boldsymbol{s}$ in the $\boldsymbol{array_x}$ has its label value changed to **#2** (check out line 4 of the Algorithm 2) and its correspondent position in the $\boldsymbol{array_y}$ determined (see line 5 of the Algorithm 2). Any solution to the right of $\boldsymbol{s}$ in $\boldsymbol{array_y}$ is labeled as **#3** in both arrays while a solution labeled as **#2** or **#3** is not found in that array or its last element has not been reached yet (see Algorithm 2, lines 6 to 11). According to the pseudocode represented in Algorithm 2, if testing conditions at the beginning of the loop in line 6 fails due to a solution not labeled as **#1** have been found ($\boldsymbol{hasLabel(array_y[pos_y]; \#1) = false}$), a dominance test is performed by code lines 12 through 23 and the solutions have their labels updated. After iterating over the $\boldsymbol{array_x}$, all solutions labeled as **#2** (non-dominated solutions) are removed from $\boldsymbol{\mathcal{S}}$, $\boldsymbol{array_x}$ (being in $\boldsymbol{array_x}$ and $\boldsymbol{array_y}$), included in $\mathcal{F}_\tau$, and returned by the algorithm (Algorithm 2, lines 26 through 37) to the callee function. Again, the typical dominance tests are avoided for those solutions located to the left of $\boldsymbol{s}$ in $\boldsymbol{array_y}$ or whenever a solution labeled as **#2** or **#3** is found in that same array. In Figure 2, an example of a solution set $\boldsymbol{\mathcal{S}}$ with 13 elements is given. To make distinction possible among solutions, identifying letters (from $\boldsymbol{A}$ to $\boldsymbol{M}$) are assigned to each one of them. As long as any solution in $\boldsymbol{array_x}$ remains not yet evaluated (labeled as **#1**), a new round is started up. During Round 1 of LONSA2Obj(.) (Figure 2), $\boldsymbol{array_x}$ and $\boldsymbol{array_y}$ are filled up with sorted solutions of the type **#1**, and the corresponding instructions are coded in lines 2-3 of the Algorithm 1. In Round 2, the small black arrow on the top left of the $\boldsymbol{array_x}$ indicates the current solution under consideration (solution $\boldsymbol{C}$, in this case). Additionally, a thin horizontal arrow drawn just below $\boldsymbol{array_y}$ points out to the initial position (solution $\boldsymbol{D}$, in this case) of this array indicating that an evaluation must be done leading to an interruption inside the loop comprised by lines 6-11 of the Algorithm 2. For that algorithm, $\boldsymbol{\mathcal{S}}$ denotes the solutions of the type **#2**, better said, the ones that belong to the current front being built at a certain iteration. Still in Round 2, solutions $\boldsymbol{L}$ and

M located to the right of the reference solution ***C*** in $array_y$ are correspondingly identified back in $array_x$ and they have their labels changed to **#3** (see lines 8-9, Algorithm 2). In the next iteration of the algorithm, solution ***M*** would be the reference solution, but it is of type **#3**, then a false boolean is returned by a conditional test and no operation is executed (line 3, Algorithm 2). This same iteration is not shown in Figure 2 because no label has changed at this time. Next, during Round 3, solution ***A*** is assumed to be the current reference solution in the $array_x$. In the $array_y$, we find solutions ***B*** and ***J*** (type **#1**), ***C*** (type **#2**), and solutions ***L*** and ***M*** (type **#3**) to the right of solution ***A***, according to theirs statuses showed on the previous round. After, solutions ***B*** and ***J*** are marked up (loop in line 6, Algorithm 2) and their labels are switched to **#3**. By finding solution ***C***, that loop is interrupted, and this solution has its label also modified to **#3** (lines 12 to 23, Algorithm 2). In Round 4, solution ***D*** is considered to be the reference. In the $array_y$, the first solution to the right of ***D*** is solution ***A***, whose label is **#2**. Now, that loop in line 6 of the Algorithm 2 is not executed, and solution ***D*** is labeled as **#3** in accordance to the dominance test (lines 12 to 23, Algorithm 2). Solution ***K*** is the reference in $array_x$ during Round 5. In this round, $array_y$ is swept (loop comprised by lines 6 to 11, Algorithm 2) until solution ***A*** is found. Those operations allow the labels of solutions ***I***, ***G*** and ***D*** to have their values changed to **#3**. What follows is the test performed in line 12 which indicates a true value for the logical condition. Two additional tests (lines 13 and 17, Algorithm 2) are executed resulting in a false value to each case and no further modification applies to the label of solution ***K***. Therefore, there is no dominance relationship between solutions ***A*** and ***K***. After this round, solutions ***A*** and ***K*** are labeled as **#2**.

It should be noted that in the next two repetitions of the instruction ***foreach*** in line 2 (Algorithm 2), solutions ***B*** and ***G*** have their labels modified to **#3**. Once the test performed in the next line of the that algorithm returns a false boolean, it is needless to perform any subsequent processing involving those solutions. These last steps are not included in Figure 2. In Round 6, solution ***F*** is taken as the reference. To the right of solution ***F*** in $array_y$, we found solution ***H*** labeled as **#1**. Once again, the loop in line 6 is repeated until solution ***K*** (labeled as **#2**) is found. From that point on, test in line 12 succeeds returning a true value and tests implemented by lines 13 and 17 fail for both cases, so that the previous label of solution ***F*** is kept the same. At the end of this iteration, solutions ***A***, ***K***, and ***F*** are labeled as **#2**. In the next round, solution ***E*** is the reference. The loop declared in line 6 is executed until solution ***F*** is found. Then, lines 12 to 23 are executed forcing the label of solution ***E*** to change to **#3**. The labels of the remaining solutions in $array_x$ (***G***, ***H***, ***I***, and ***J***) are different from **#1** and they do not demand further computer processing. For practical purposes, those iterations are not shown in Figure 2. After solution ***J*** in $array_x$ has been taken, the loop in line 2 (Algorithm 2) is interrupted. Now, all solutions in $array_x$ or $array_y$, labeled as **#2**, belong to the front $\mathcal{F}$.

In Figure 2, two aspects should be highlighted. First, whenever any two solutions, say solutions ***A*** and ***C***,

Algorithm 2: Pseudocode for sorting fronts of non-dominated solutions in two-objective space. (LONSA2Obj)

Input: $\mathcal{S}$: solution set
Input: $array_x$: solutions ordered by the first objective
Input: $array_y$: solutions ordered by the second objective
Input: *hash*: hash table used to search for solutions
Output: $\mathcal{F}$: Pareto front found

```
1  Begin
2  foreach s in array_x do
3      if hasLabel(s, #1) then
4          label(s, #2)
5          pos_y ← find_y(s, hash) + 1
6          while (pos_y < length(array_y)) and
             (hasLabel(array_y[pos_y], #1)) do
7              pos_x ← find_x(array_y[pos_y], hash)
8              label(array_x[pos_x], #3)
9              label(array_y[pos_y], #3)
10             pos_y ← pos_y + 1
11         end
12         if pos_y < length(array_y) then
13             if s < array_y[pos_y] then
14                 label(array_y[pos_y], #3)
15                 pos_x ← find_x(array_y[pos_y], hash)
16                 label(array_x[pos_x], #3)
17             else if array_y[pos_y] < s then
18                 pos_y ← find_y(s, hash)
19                 label(array_y[pos_y], #3)
20                 pos_x ← find_x(s, hash)
21                 label(array_x[pos_x], #3)
22             end
23         end
24     end
25 end
26 F ← ∅
27 foreach s in array_x do
28     if hasLabel(s, #2) then
29         S ← S − s
30         F ← F ∪ s
31         remove(s, array_x)
32         remove(s, array_y)
33     else
34         label(s, #1)
35     end
36 end
37 return F
38 End
```

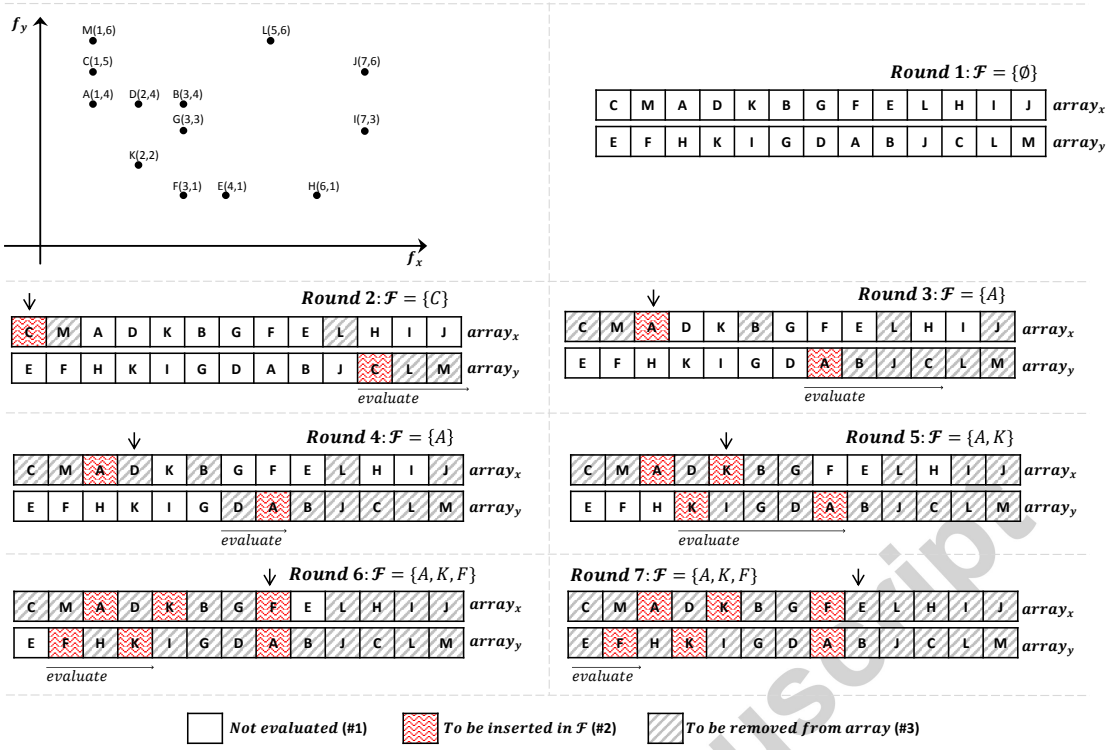


Figure 2: Applying LONSA to a set of solutions in a two-objective space.

have identical values for one of the objectives (first objective, for example), it will demand a dominance-checking. While solution $\boldsymbol{A}$ is being evaluated in Round 3, solution $\boldsymbol{C}$ is removed, although it was previously considered a member of the current front, labeled as **#2**. By performing the comparison operation ($\boldsymbol{A} \prec \boldsymbol{C}$) (line 17, Algorithm 2), solution $\boldsymbol{C}$ is definitely removed from the current front (line 18 through 21, Algorithm 2). Second, a solution may be early considered belonging to the current front (label **#2**) as we can see during the evaluation of solution $\boldsymbol{D}$, in Round 4. Although being labeled as **#2** (line 4, Algorithm 2), this solution does not really belong to the current front. In this case, the label of this solution will be changed after the comparison operation ($\boldsymbol{A} \prec \boldsymbol{D}$) returns a true boolean (line 17, Algorithm 2) and lines 18 to 21 of the Algorithm 2 have been already executed.

3.2. Sub-procedure for many-objective optimization

A sorting scheme must be also capable of tackling a m-objective search space ($\boldsymbol{m \geq 3}$). In this sense, some adaptions were designed to the Algorithm 2 of the previous section. For the sake of clarity, those modifications are now presented in the LONSAManyObj(.) procedure (Algorithm 3). First, it checks the label of $\boldsymbol{s}$ in the $\boldsymbol{array_x}$ (line 3, Algorithm 3). If that label is **#1**, it is changed to **#2** (line 4, Algorithm 3) and the position $\boldsymbol{pos_y}$ of the first solution in $\boldsymbol{array_y}$ with $\boldsymbol{f_2}$ (second objective) equals to $\boldsymbol{f_2}$ of $\boldsymbol{s}$ is returned by the function $\boldsymbol{findFirst_y}(.)$ (line 5, Algorithm 3). After that, a sweep along the $\boldsymbol{array_y}$ is done to ensure the dominance tests are only performed to the solutions whose labels are **#1** or **#2** and to avoid unnecessary dominance tests. Then, those solutions are labeled according to the result of the dominance testing (lines 8-18, Algorithm 3). If the solution located in position $\boldsymbol{pos_y}$ dominates solution $\boldsymbol{s}$, the inner loop (line 6) is interrupted (line 17) and dominance tests are avoided. After iterating over $\boldsymbol{array_x}$, Algorithm 3 performs the very same operations defined across lines 26 to 37 of the Algorithm 2.

In Figure 3, an example demonstrating the use of Algorithm 3 is shown. The objective values highlighted in red, $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$, were used to sort solutions of the set $\boldsymbol{\mathcal{S}}$. In that algorithm, it is important to note that any already evaluated solution is considered to belong to the current non-dominated front (labeled **#2**) and they may become dominated in the next iteration, as we can see in Round 4. At this moment, solution $\boldsymbol{A}$ is labeled as **#3**. The improvements in performance of the suggested scheme is notorious since solutions labeled as **#3** are

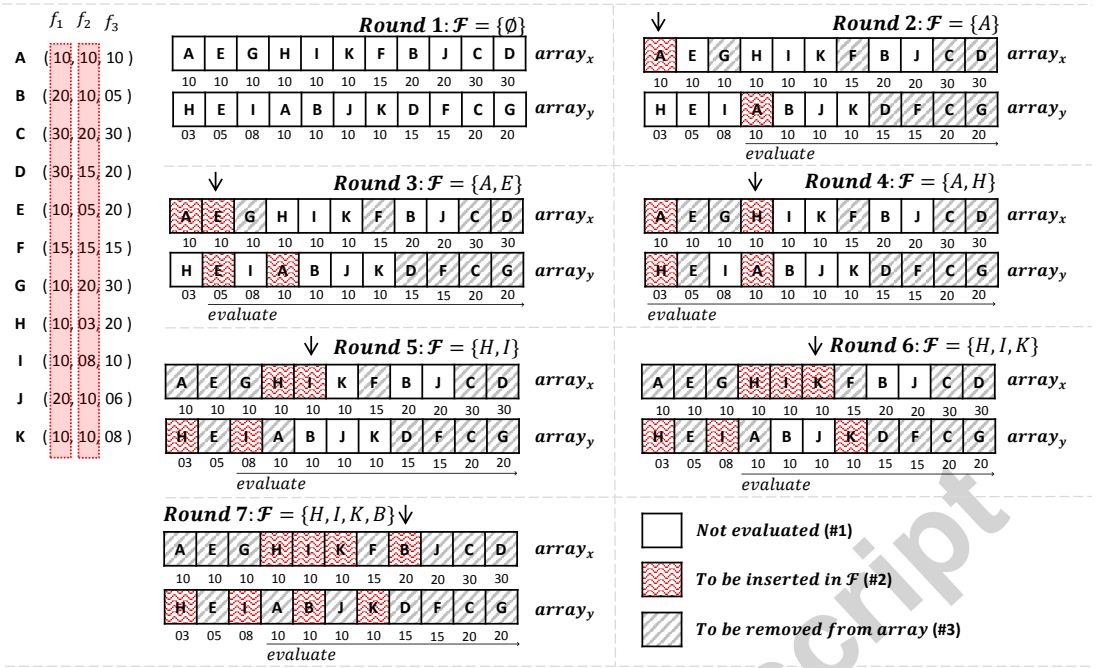


Figure 3: Applying LONSA to a set of solutions in a 3-objective space.

found in $\boldsymbol{array_x}$ and $\boldsymbol{array_y}$ and, so, dominance tests are avoided. It is also important to take a look at the evaluation of the solution $\boldsymbol{K}$ while Round 6 takes place. The beginning of that evaluation occurs in $\boldsymbol{array_y}$ for solution $\boldsymbol{A}$ rather than solution $\boldsymbol{K}$. This particular situation happens due to the output parameters of the function $\boldsymbol{findFirst_y}$ (line 5, Algorithm 3) which in turn returns the first solution whose $\boldsymbol{f_2}$ value equals to 10 (solution $\boldsymbol{A}$) in $\boldsymbol{array_y}$.

3.3. Time-complexity analysis

This section presents a discussion about the computational cost corresponding to the two-objective and many-objective cases dealt by the two versions of LONSA. Initially, LONSA applies its sorting scheme to solution set $\boldsymbol{\mathcal{S}}$ based on the values of two objectives. This ordering is performed by an efficient strategy known as Quicksort [22]. Its computational cost is $\boldsymbol{O(n.\log(n))}$, where $\boldsymbol{n}$ is the number of solutions contained by $\boldsymbol{\mathcal{S}}$. Thus, the cost of this same procedure is $\boldsymbol{2*O(n.\log(n))}$. Furthermore, LONSA employs a structure known as Hash Table [22] to allow any query for solution to be performed in a faster way. The cost to build such special data structure is $\boldsymbol{O(n)}$, and each searching in it has a fixed cost of $\boldsymbol{O(1)}$. The quantity of dominance tests depends on the sequence these solutions are found in the arrays. The best case occurs when a given solution is placed in the first position in both arrays. For two-objective problems, it can be said that that solution dominates all the others. Thus, the solutions of the i-th position in $\boldsymbol{array_x}$ and $\boldsymbol{array_y}$ have their labels modified to **#3**, where $\boldsymbol{i = \{2, ..., ||array_x||\}}$. In this case, the number of non-dominated fronts equals the cardinality of $\boldsymbol{\mathcal{S}}$ ($\boldsymbol{||\mathcal{S}||}$). On the other hand, the worst case occurs whenever the solutions of the i-th position in the $\boldsymbol{array_x}$ are placed into position $\boldsymbol{||array_x|| - i}$ of the $\boldsymbol{array_y}$. Finally, we have $\boldsymbol{(n-1).n}$ dominance tests to be executed and one single non-dominated front. Therefore, the algorithm complexity is $\boldsymbol{O(n^2)}$ for the worst scenario.

In the next section, we investigate the performance of LONSA to compare it with other nine methods found in the literature. A number of experiments were carried out to assess its overall performance and the correspondent computational cost of the algorithms. Those trials have taken into consideration some issues raised by Zhang [14] and Ishibuchi [6].

4. Computational experiments

For validation purposes, LONSA performance was compared to other nine classification methods: Fast Non-dominated Sorting (FNDS) [7], Get Dominance Tree (GDT) [15], Non-Dominated Tree (NDT) [20], Arena's Principle (ARENA) [17], Climbing Sort

Algorithm 3: Pseudocode tailored for a many-objective optimization problem. (LONSAManyObj)

```
   Input: S: solution set
   Input: array_x: solutions ordered by the first objective
   Input: array_y: solutions ordered by the second objective
   Input: hash: hash table used to search for solutions
   Output: F: Pareto front found
1  Begin
2  foreach s in array_x do
3      if hasLabel(s, #1) then
4          label(s, #2)
5          pos_y ← findFirst_y(s, hash)
6          while pos_y < length(array_y) do
7              if not hasLabel(array_y[pos_y], #3) then
8                  if s < array_y[pos_y] then
9                      label(array_y[pos_y], #3)
10                     pos_x ← find_x(array_y[pos_y], hash)
11                     label(array_x[pos_x], #3)
12                 else if array_y[pos_y] < s then
13                     pos_y ← find_y(s, hash)
14                     label(array_y[pos_y], #3)
15                     pos_x ← find_x(s, hash)
16                     label(array_x[pos_x], #3)
17                     break;          /*Exit the loop.*/
18                 end
19             end
20             pos_y ← pos_y + 1
21         end
22     end
23 end
24 ..... lines 26–37 of Algorithm 2 .....
25 End
```

(CLIMBING) and Deductive Sort (DEDUCTIVE) [16], Corner Sort (CORNER) [13], Efficient Non-dominated Sort with Sequential Search Strategy (ENS-SS) and with Binary Search Strategy (ENS-BS) [14]. All those methods were evaluated with the same testbed sets. They were also implemented by the same programming language (Java) and the respective code compiled using the JDK 1.7 compiler. The computational simulations were performed on the same computer containing a processor Pentium Core i7 2.2GHz 8GB RAM.

In order to allow a fair comparison among the sorting methods, two quality indicators were envisioned: (i) the number of dominance tests and (ii) runtime needed to sort the entire solution set into non-dominated fronts. The simulations were organized into three configurations: (i) synthetic sets with predefined m were especially created ensuring a well-controlled number of solutions per front, (ii) sets with a given m and predefined cardinality containing randomly generated solutions, and (iii) sets evaluated by embedding the sorting method to the NSGA-II (in case of two-objective problem) or to the NSGA-III (in case of optimization problems with $m = 5$ and $m = 10$).

Using the Algorithm 4, we provide the five necessary steps to create a solution set $\mathcal{S}$ for the first configuration according to [14]. For this, the following parameters are needed: set size or its cardinality (n); number of objectives (m); number of fronts (p), lower limit for objective values ($minVal$) and, upper limit for objective values ($maxVal$). The five basic steps consist in: (i) to create matrix $\mathcal{M}$ ($\mathcal{M} \in \mathbb{R}^{m \times n}$) wherein each position of it must be a random value in the range $[minVal, maxVal]$; (ii) sort the lines of $\mathcal{M}$ in an ascending order resulting in $\mathcal{M}'$; (iii) create solutions based on the values of $\mathcal{M}'$ and assign them to a subset of solutions $\mathcal{F}$; (iv) define a subset β of objective indexes to be dynamically changed. The solutions belonging to $\mathcal{F}$ have their objective values shuffled to be then assigned to $\mathcal{S}$; and, (v) shuffle the set $\mathcal{S}$. These five steps are schematically illustrated in Figure 4 for a solution set with $n = 8$ ($||\mathcal{S}|| = 8$), three objectives ($m = 3$) and two fronts ($p = 2$). Restricted to the range $[1, 9]$, integer values were generated for the objectives.

The example in Figure 4 considers the subset of objectives $\beta = \{3, 1\}$ to be used in the fourth step (line 17, Algorithm 4). At first, the matrix $\mathcal{M}$ is created by means of the function $makeMatrix(.)$ (line 2, Algorithm 4) where each value $\mathcal{M}_{i,j}$ follows an uniform distribution defined in the interval $[minVal, maxVal]$ with $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$. Next, the values of each row of $\mathcal{M}$ are sorted in ascending sequence by function $sort(.)$. In the loop defined through lines 8-16, solutions are created and distributed over p sets such that γ solutions are found in each one of them ($\gamma = \lfloor n/m \rfloor$).

The value of the i-th objective of a j-th solution receives the value of $\mathcal{M}'_{i,j}$, such that $i = \{1, \ldots m\}$, from the function $newSolution(.)$ (line 11, Algorithm 4). After the loop (lines 8 to 16), a subset β of length b, where b is an integer in the range $[1, m[$. Each position of β, say β_i, is filled with a distinct random integer which belongs to the same interval $\beta_i \in \{1, \ldots, m\}$, provided by the function $randObjectiveIndex(.)$ (line 17, Algorithm 4). After that, the loop defined by lines 20 to 24 implements a shuffle of the objective values associated to the solutions of $\mathcal{F}_\tau$. For it, the function $randObjectives(.)$ (line 17, Algorithm 4) provides the value of each β_i objective of the j-th solution belonging to $\mathcal{F}_\tau$ and then swaps its value with the objective value of index i of the $\gamma + 1 - j$ for all $j \in \{1, \ldots, \gamma/2\}$. The following step consists on assignment of the solutions belonging to $\mathcal{F}_\tau$ to the set $\mathcal{S}$ (line 22, Algorithm 4). After the loop, the elements of $\mathcal{S}$ are shuffled (line 25, Algorithm 4) and returned by the procedure (line 26,

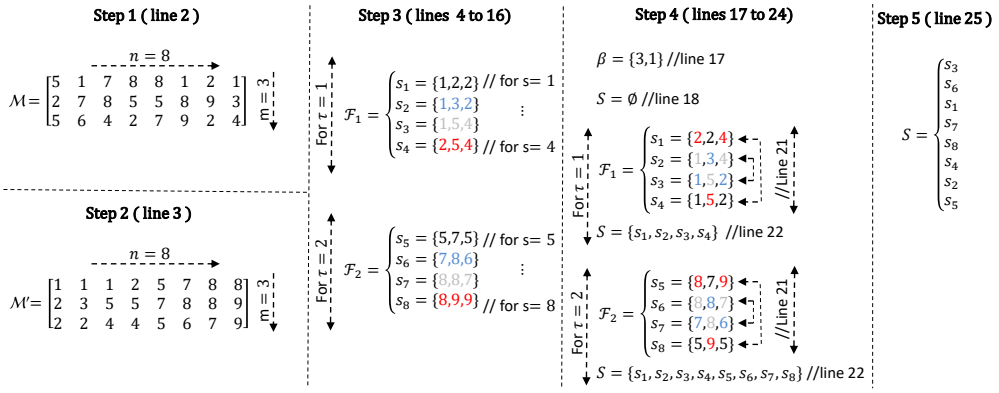


Figure 4: Discrimination of solutions considering $n = 8$, $m = 3$ and $p = 2$. The objective integer values were generated using the range **[1, 9]**. The code lines refer to the Algorithm 4.

Algorithm 4: Pseudocode for construction of synthetic solution sets.

Input: n: cardinality of the set
Input: m: number of objectives
Input: p: number of fronts
Input: $minVal$: minimum objective value
Input: $maxVal$: maximum objective value
Output: $\mathcal{S}$: solution set

1 **Begin**
2 $\mathcal{M} \leftarrow makeMatrix(m, n, minVal, maxVal)$
3 $\mathcal{M}' \leftarrow sort(\mathcal{M})$
4 $\mathcal{F} \leftarrow \emptyset$
5 $\gamma \leftarrow \lfloor n/p \rfloor$ /*number of the solutions per front*/
6 $\tau \leftarrow 1$ /*counter of the fronts */
7 $j \leftarrow 1$ /*counter of the solutions */
8 **while** $\tau \leq p$ **do**
9 $i \leftarrow 1$ /*counter of the sol. per fronts*/
10 **while** $i \leq \gamma$ **do**
11 $\mathcal{F}_\tau \leftarrow \mathcal{F}_\tau \cup newSolution(\mathcal{M}', m, j)$
12 $j \leftarrow j + 1$
13 $i \leftarrow i + 1$
14 **end**
15 $\tau \leftarrow \tau + 1$
16 **end**
17 $\beta \leftarrow randObjectivesIndex(m)$
18 $\mathcal{S} \leftarrow \emptyset$
19 $\tau \leftarrow 1$ /*counter of the fronts */
20 **while** $\tau \leq p$ **do**
21 $randObjectives(\mathcal{F}_\tau, \beta, \gamma)$
22 $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{F}_\tau$
23 $\tau \leftarrow \tau + 1$
24 **end**
25 $\mathcal{S} \leftarrow shuffles(\mathcal{S})$
26 **return** $\mathcal{S}$
27 **End**

Algorithm 4).

4.1. Tests on sets with predefined fronts

This section presents some tests with sets assuming a fixed number of fronts. Thus, we have generated three patterns for sets considering the variation of the number of objectives $\boldsymbol{m = \{2, 5, 10\}}$. Each set has a number of fronts defined by $\boldsymbol{p = \{2, 3, 4, \ldots, 70\}}$. Thus, 207 scenarios ($\boldsymbol{||m|| \times ||p||}$) were produced to evaluate the implemented algorithms. This dataset considers sets whose cardinality is predefined and, therefore, previously known ($\boldsymbol{n = 2000}$). To have a statistically meaningful result, each set was assessed 33 times.

Figures 5 and 6 show the results of LONSA and other nine methods reported in the literature. From Figure 5, we can infer the FNDS algorithm with its worst performance if compared to the other algorithms. Its number of comparisons (dominance evaluations) is given by $\boldsymbol{n.(n-1)}$. Once the number of comparisons and the runtime of the FNDS is significantly greater than the other methods, the $\boldsymbol{Y}$ axis is converted to a logarithmic scale. For solution sets with two objectives, LONSA had the lowest number of comparisons.

When comparing LONSA to ENS-BS for a two-objective case, a significant reduction in the number of dominance evaluations is noticed regardless of the number of fronts associated to the sets. Tests performed to sets with 5 and 10 objectives have demonstrated that ENS-BS needs the lowest number of comparisons. That behavior becomes more evident as the number of fronts $\boldsymbol{p}$ grows, especially when $\boldsymbol{p > 10}$. On the other hand, if the number of fronts $\boldsymbol{p}$ is lower than 10, the algorithms have exhibited similar number of comparisons, excluding NDT, GDT and FNDS with the worst performances.

Figure 6 shows the average runtime profiles for all the tested sorting methods obtained from 33 trials when they were applied to the same solution set of the first configuration. For sets with two objectives, LONSA and the methods based on the ENS strategy have shown lower execution times than the others. For sets with

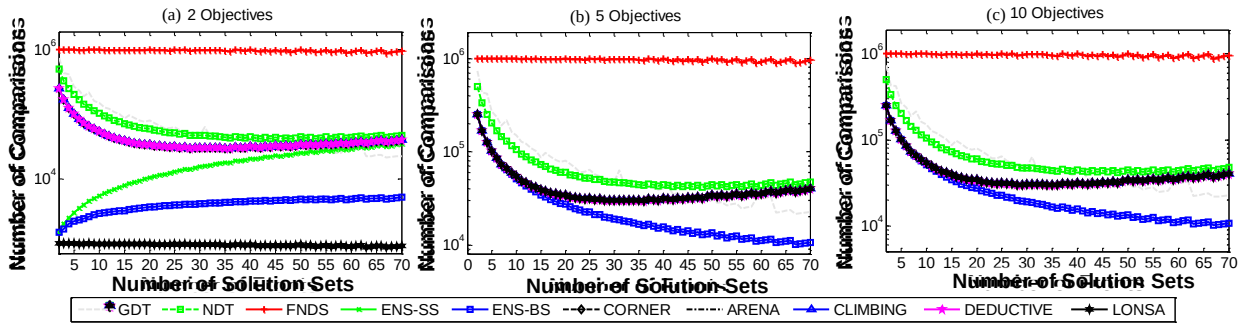


Figure 5: Numbers of comparisons of ten non-dominated sorting approaches for solution sets with predefined fronts.

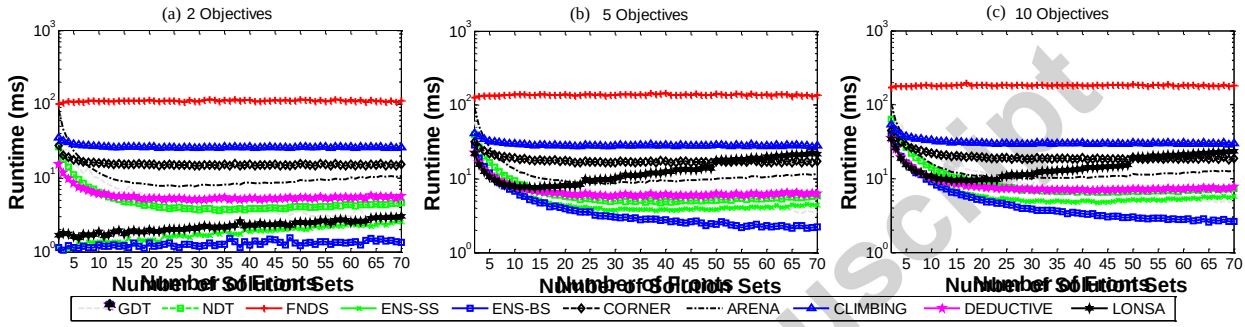


Figure 6: Runtimes of ten non-dominated sorting approaches for solution sets with predefined fronts.

5 and 10 objectives, the ENS-BS algorithm required lower runtime. This behavior can be explained by the considerable smaller number of comparisons performed with ENS-BS. In those cases, LONSA have shown similar performance to the other methods being evaluated when sets of solutions with ten fronts ($p = 10$) or more were tried. However, the runtime is sensibly increased as p grows (see Figure 6 again). The reduction in computing performance can be explained by the need to update the array of solutions whenever a new front is found. LONSA executes $O(n)$ operations for every front found. However, Koppens Theory [24] shows that the size of a given non-dominated solution set grows significantly whenever the number of objectives is increased as well. In this sense, the set of non-dominated solutions becomes equal to the cardinality of S, better said, all solutions will be located on the first front as shown in [13, 14].

4.2. Performing tests on random sets

In this this part of the work, simulations were conceived to evaluate the general behavior of the methods when cardinality of the sets had enlarged considerably. To accomplish this, solution sets with two, five and ten objectives ($m = \{2, 5, 10\}$) were considered here and the cardinality of the sets ranged from 50 to 1000 in steps of 50 ($n = \{50, 100, \ldots, 1000\}$). Thus, we have produced 20 possibilities for set sizes along with two, five or ten objectives, compounding 60 scenarios altogether. Each combination was rated through 33 trials.

Figure 7 shows the simulation results which were obtained when the number of paired comparisons was considered as one of the performance indexes. If the number of paired comparisons was used to identify the best method to tackling solution sets related to two-objective problems, LONSA have proved to be better adapted than other methods, mainly for sets with 500 solutions or more (see Figure 7(a)). For solution sets with more than 500 solutions, LONSA has achieved comparable efficiency to the best method known so far: the ENS-BS. For sets associated with optimization problems defined by 5 and 10 objectives (Figures 7(b) and 7(c), respectively), LONSA have performed as good as other methods available in the specialized literature. For those cases, NDT and FNDS have presented the worst results. Figure 8 gathers runtime information achieved by the ten chosen methods for problems with 2, 5 or 10 objectives. In terms of computational time, Figure 8(a) re-

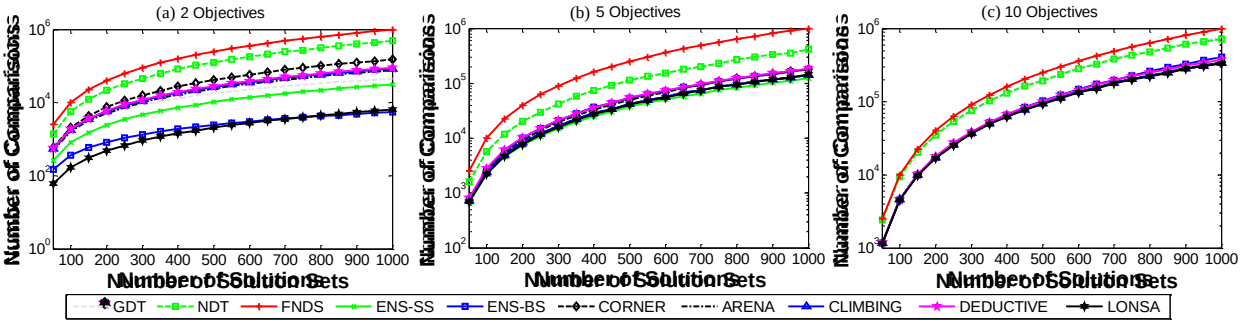


Figure 7: Numbers of comparisons of ten non-dominated sorting approaches for randomly generated solution sets.

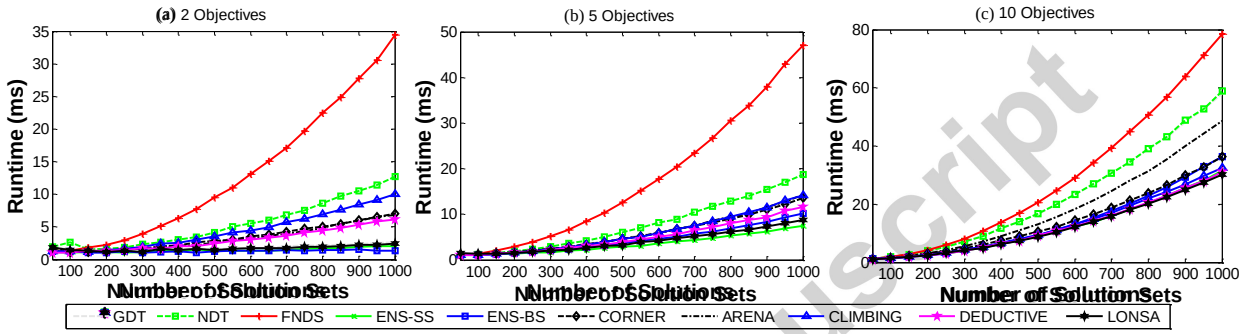


Figure 8: Runtime of ten non-dominated sorting approaches for randomly generated solution sets.

Table 2: The means and standard deviations of runtime and numbers of comparisons over 33 randomly generated solution sets having 300, 500 and 1,000 solutions. Best performance is shown in bold.

Method	Objectives	Number of Solutions: 300	500	1000	300	500	1000
		Number of Comparisons (Standard Deviation)			Runtime (ms) (Standard Deviation)		
GDT	2	9.00×10^3 (3.77×10^2)	1.75×10^4 (5.11×10^2)	4.50×10^4 (1.11×10^3)	1.72 (0.67)	1.97 (0.68)	2.94 (0.50)
	5	4.11×10^4 (2.46×10^3)	1.05×10^5 (4.03×10^3)	3.62×10^5 (1.15×10^4)	2.54 (0.51)	4.78 (0.64)	13.85 (0.62)
	10	7.81×10^4 (2.82×10^3)	2.04×10^5 (9.34×10^3)	7.63×10^5 (2.11×10^4)	7.60 (0.50)	17.81 (0.84)	62.36 (1.85)
NDT	2	4.69×10^4 (7.07×10^3)	1.26×10^5 (1.94×10^4)	4.79×10^5 (4.97×10^4)	2.75 (0.14)	4.30 (0.91)	12.27 (1.28)
	5	4.26×10^4 (5.33×10^3)	1.06×10^5 (9.64×10^3)	3.90×10^5 (2.96×10^4)	2.91 (0.46)	5.66 (0.47)	17.94 (1.37)
	10	7.68×10^4 (3.61×10^4)	1.97×10^5 (1.13×10^4)	7.34×10^5 (2.67×10^4)	6.85 (0.44)	16.06 (0.96)	55.72 (1.92)
FNDS	2	9.00×10^4 (0)	2.50×10^5 (0)	1.00×10^6 (0)	5.18 (0.55)	9.42 (0.83)	34.15 (2.81)
	5	9.00×10^4 (0)	2.50×10^5 (0)	1.00×10^6 (0)	5.27 (0.45)	12.66 (0.59)	46.91 (0.77)
	10	9.00×10^4 (0)	2.50×10^5 (0)	1.00×10^6 (0)	8.21 (0.41)	20.69 (0.91)	78.09 (1.33)
ENS-SS	2	4.43×10^3 (2.84×10^2)	1.07×10^4 (5.98×10^2)	3.15×10^4 (1.20×10^3)	**1.27 (0.45)**	1.55 (0.50)	**1.64 (0.49)**
	5	**1.50×10^4 (7.76×10^2)**	**3.69×10^4 (1.35×10^3)**	1.23×10^5 (4.61×10^3)	**1.81 (0.39)**	**2.96 (0.30)**	**7.42 (0, 50)**
	10	3.73×10^4 (1.55×10^3)	9.59×10^4 (5.37×10^3)	3.49×10^5 (1.18×10^4)	4.09 (0.29)	9.00 (0.61)	31.72 (1.37)
ENS-BS	2	1.33×10^3 (2.74×10^1)	2.46×10^3 (3.13×10^1)	5.48×10^3 (4.20×10^1)	1.30 (0.46)	**1.39 (0.55)**	1.97 (0.17)
	5	2.11×10^4 (1.51×10^3)	5.27×10^4 (3.23×10^3)	1.81×10^5 (8.34×10^3)	2.03 (0.17)	3.72 (0.51)	10.00 (0.56)
	10	3.78×10^4 (1.59×10^3)	1.04×10^5 (5.31×10^3)	4.03×10^5 (1.41×10^4)	4.09 (0.29)	9.54 (0.56)	35.96 (1.59)
CORNER	2	1.60×10^4 (1.03×10^3)	4.19×10^4 (2.12×10^3)	1.53×10^5 (6.86×10^3)	3.88 (0.77)	5.67 (1.53)	10.58 (14.62)
	5	1.92×10^4 (1.78×10^3)	5.01×10^4 (3.01×10^3)	1.84×10^5 (1.01×10^4)	2.44 (0.50)	4.63 (0.48)	13.36 (0.59)
	10	3.72×10^4 (1.66×10^3)	9.57×10^4 (6.44×10^3)	3.52×10^5 (1.98×10^4)	4.91 (0.28)	10.52 (0.60)	35.33 (1.63)
ARENA	2	1.02×10^4 (5.02×10^2)	2.40×10^4 (8.37×10^4)	7.68×10^4 (2.14×10^3)	1.69 (0.46)	2.73 (0.62)	6.76 (0.66)
	5	1.65×10^4 (1.44×10^3)	4.21×10^4 (2.49×10^3)	1.45×10^5 (7.54×10^3)	2.54 (0.50)	4.60 (0.49)	14.09 (0.91)
	10	3.75×10^4 (1.78×10^3)	9.40×10^4 (7.78×10^3)	3.47×10^5 (1.69×10^4)	5.70 (0.58)	12.81 (1.3)	50.54 (2.91)
CLIMBING	2	1.12×10^4 (5.89×10^2)	2.70×10^4 (1.19×10^3)	8.51×10^4 (3.17×10^3)	2.12 (0.33)	3.76 (0.43)	11.39 (0.50)
	5	1.66×10^4 (1.25×10^3)	4.16×10^4 (2.82×10^3)	1.44×10^5 (6.95×10^3)	2.48 (0.51)	4.66 (0.54)	14.82 (0.46)
	10	3.73×10^4 (1.81×10^3)	9.39×10^4 (7.44×10^3)	3.45×10^5 (1.74×10^4)	4.21 (0.41)	9.18 (0.58)	33.03 (1.13)
DEDUTIVE	2	1.19×10^4 (7.17×10^2)	2.85×10^4 (1.39×10^3)	9.00×10^4 (3.26×10^3)	1.69 (0.46)	2.61 (0.49)	6.06 (0.24)
	5	2.10×10^4 (1.57×10^3)	5.31×10^4 (3.11×10^3)	1.85×10^5 (7.67×10^3)	2.06 (0.24)	3.93 (0.24)	11.30 (0.59)
	10	3.89×10^4 (1.66×10^3)	1.02×10^5 (5.16×10^3)	3.81×10^5 (1.39×10^4)	**4.00 (0.35)**	8.87 (0.54)	29.82 (1.06)
LONSA	2	**2.63×10^2 (2.49)**	**4.49×10^2 (3.05)**	**9.24×10^2 (2.98)**	2.03 (0.76)	2.12 (0.85)	2.94 (0.61)
	5	1.60×10^4 (1.16×10^3)	4.16×10^4 (2.51×10^3)	**1.43×10^5 (7.42×10^3)**	2.12 (0.33)	3.33 (0.47)	8.33 (0.54)
	10	**3.71×10^4 (1.80×10^3)**	**9.37×10^4 (7.18×10^3)**	**3.45×10^5 (1.59×10^4)**	4.12 (0.33)	**8.69 (0.52)**	**29.78 (1.45)**

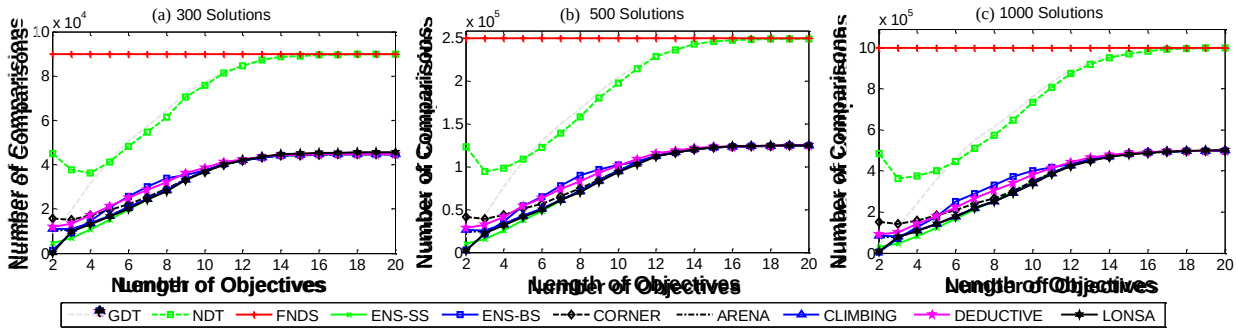


Figure 9: Numbers of comparisons of ten non-dominated sorting approaches for randomly generated solution sets with 300, 500 and 1000 solutions and different number of objectives.

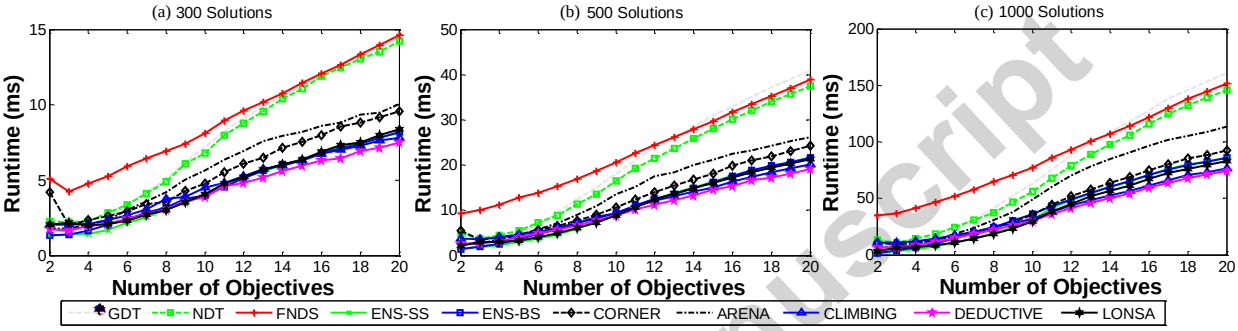


Figure 10: Runtimes of ten non-dominated sorting approaches for randomly generated solution sets with 300, 500 and 1000 solutions and different number of objectives.

veals that LONSA has exhibited a performance close to the best methods when dealing with solution sets of size equals to 500 solutions. For solution sets ranged from 500 to 1000 solutions, LONSA has stood out along with other three methods, namely GDT, ENS-SS and ENS-BS. For sets with five objectives, ENS-SS and LONSA have presented the best ever results. For sets with 10 objectives, LONSA have featured slightly shorter runtime if it is compared to the other methods, which can be explained by fewer paired comparisons needed by it. In Table 2, a comparison among the ten methods was made considering sets with 300, 500 and 1000 solutions. In addition, it also accounts for solutions with 2, 5 and 10 objectives. In that table, the best results have had their values highlighted in bold. It is noticed that LONSA is the sorting method which presents fewer comparisons in most cases. Further, computational times outlined by LONSA were often shorter than other evaluated methods for sets with 500 and 1000 solutions, when considering problems with 10 objectives.

The second part of this section presents simulations which were fulfilled to evaluate the behavior of the methods as the number of objectives increases. Figures 9 and 10 have shown the simulation results for solution sets whose size is 300, 500 and 1000 solutions, respectively. Taking those cardinalities for the solution set, an interval of variation in number of objectives was considered from 2 to 20 in steps of 1 for testing purposes, totalizing 57 configurations. Each configuration was evaluated 33 times. In general, results have showed the efficiency of the methods does not straightly depend on cardinalities. In case with up to 10 objectives, LONSA have presented smaller number of comparisons among all the evaluated methods (see Figure 9). For solution sets with more than 10 objectives, all the methods have outlined a similar behavior except GDT, NDT and FNDS, which have had the worst performances. It is noteworthy that LONSA has had the shorter runtime when compared to the other methods for sets with number of objectives ranging from 6 to 10. When the number of objectives considered was larger than 10, the DEDUCTIVE method has provided the best results (see Figure 10).

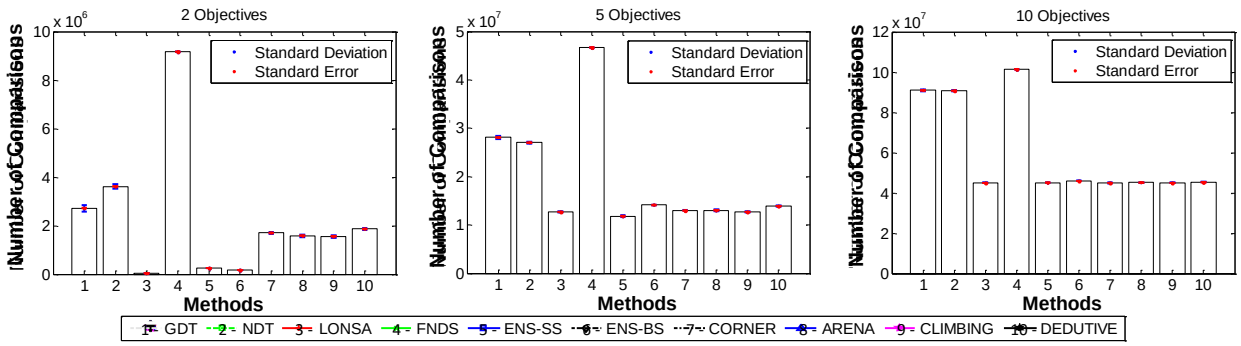


Figure 11: The mean, standard deviation, and standard error of comparisons of the NSGA-II (2 objectives) and NSGA-III (5 and 10 objectives) framework with ten non-dominated sorting approaches on DTLZ1 problem.

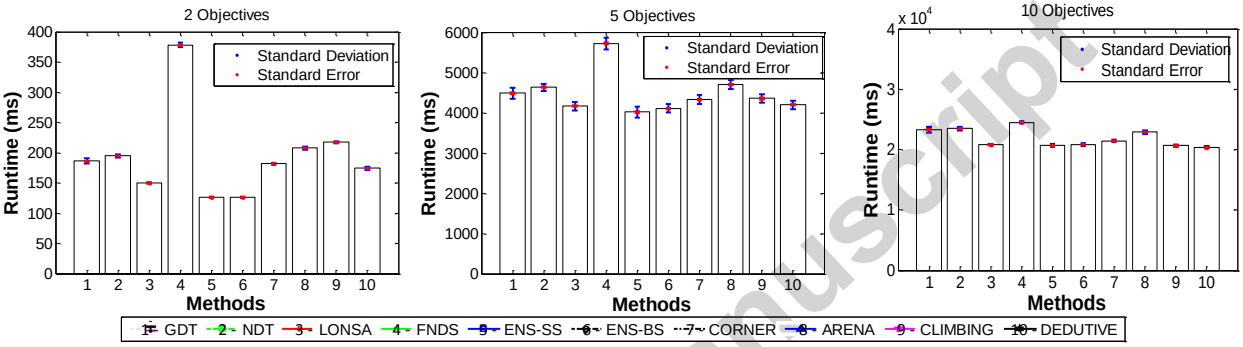


Figure 12: The mean, standard deviation, and standard error of runtime of the NSGA-II (2 objectives) and NSGA-III (5 and 10 objectives) framework with ten non-dominated sorting approaches on DTLZ1 problem.

4.3. Simulations with sorting algorithms embedded in NSGA-II and NSGA-III

The sorting methods assayed in this work were embedded into the multiobjective evolutionary algorithms known as NSGA-II [7] and NSGA-III [4] replacing their respective original sorting procedure FNDS. By doing so, the improvements in the performance of those optimization algorithms could be properly analyzed. During this evaluation process, we have used the benchmark problem known as DTLZ1 [25] defined for 2, 5 and 10 objectives. For the first problem ($m = 2$), the NSGA-II have dealt with a set of 100 solutions for a maximum number of generations set up to 250. Other parameters were specified as recommended in [7]. On the other hand, the NSGA-III was applied to the problems with 5 and 10 objectives since it has proved to get along with MOPs [4, 5]. The parameters of the NSGA-III were defined as recommended in [4]. Each configuration employed to the algorithms was verified by running them 33 times.

The results achieved with the two algorithms using the sorting methods are shown in Figures 11 and 12. The comparisons were conducted by taking the number of comparisons and runtime as the performance indexes. The NSGA-II plus LONSA has had the fewest number of paired comparisons if it is opposed to the other methods. Besides that, NSGA-II using algorithms based on ENS has had the lowest computational time. For DTLZ1 with 5 objectives, the NSGA-III plus LONSA and the ENS-SS has showed the best performance according to the performance indexes mentioned above. For DTLZ1 with 10 objectives, all the methods have showed similar behaviors in terms of the number of comparisons except GDT, NDT and FNDS. The runtime for all the methods were also similar but GDT, NDT, FNDS, and Arena which have exhibited the worst results.

5. Conclusion

In this work, a new method LONSA is devised to speed up the sorting procedure generally employed by multiobjective optimization algorithms to identify non-dominated fronts from a given solution set. Our

proposal differs from other known strategies found in the literature. Additionally, our study have compared LONSA to other nine extensively used methods. The performance indexes chosen to make evaluation possible among all methods implemented were the number of paired comparisons required in dominance test and the runtime needed to carry out the sorting task.

Solution sets with distinct properties were generated in order to challenge the sorting methods in different ways. Simulations have assumed variation in terms of number of objectives, number of solutions and number of fronts in synthetic sets, intentionally designed to hamper the classification process. Moreover, the algorithms were embedded into two well-known optimization algorithms: NSGA-II and NSGA-III. The NSGA-II was tested with DTLZ1 problem configured for two objectives and NSGA-III was tested with the DTLZ1 set up for five and ten objectives.

In most simulation cases, LONSA has outperformed other nine methods considering the two performance indexes - number of comparisons and runtime. In addition, results have showed that LONSA has achieved the shortest runtime among all the methods analyzed when problems with 5 to 10 objectives were taken as the size of the sets have continually increased. It is recommended that future studies should evaluate other data structures to reduce solution array updating and, consequently, the computational cost to sort the solutions in the arrays by their objective values.

Acknowledgements

The authors would like to thank the UFOP and Brazilian agencies CAPES and CNPq for providing financial support.

[1] C. M. Fonseca, P. J. Fleming, Genetic algorithms for multiobjective optimization: Formulationdiscussion and generalization, in: Proceedings of the 5th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993, pp. 416–423.

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

[3] C. Coello, G. Lamont, D. Veldhuizen, Evolutionary Algorithms for Solving Multi-Objective Problem, 2nd Edition, Springer, 2007.

[4] K. Deb, H. Jain, An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints, Evolutionary Computation, IEEE Transactions on 18 (4) (2014) 577–601.

[5] H. Jain, K. Deb, An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part ii: Handling constraints and extending to an adaptive approach, Evolutionary Computation, IEEE Transactions on 18 (4) (2014) 602–622.

[6] H. Ishibuchi, N. Akedo, H. Ohyanagi, Y. Nojima, Behavior of EMO algorithms on many-objective optimization problems with correlated objectives, in: Evolutionary Computation (CEC), 2011 IEEE Congress on, 2011, pp. 1465–1472.

[7] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, Evolutionary Computation, IEEE 6 (2) (2002) 182–187.

[8] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization, in: K. Giannakoglou, et al. (Eds.), Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001), International Center for Numerical Methods in Engineering (CIMNE), 2002, pp. 95–100.

[9] D. Ahangaran, A. Yasrebi, A. Wetherelt, P. Foster, Real-time dispatching modelling for trucks with different capacities in open pit mines, Archives of Mining Sciences 57 (1) (2012) 39–52.

[10] C. Shi, M. Chen, Z. Shi, A fast nondominated sorting algorithm, in: Neural Networks and Brain, 2005. ICNN B '05. International Conference on, Vol. 3, 2005, pp. 1605–1610.

[11] J. Knowles, D. Corne, The pareto archived evolution strategy: a new baseline algorithm for pareto multiobjective optimisation, in: Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on, Vol. 1, 1999, pp. 98–105.

[12] M. Ali, P. Siarry, M. Pant, An efficient differential evolution based algorithm for solving multi-objective optimization problems, European Journal of Operational Research 217 (2) (2012) 404–416.

[13] H. Wang, X. Yao, Corner sort for pareto-based many-objective optimization, Cybernetics, IEEE Transactions on 44 (1) (2014) 92–102.

[14] X. Zhang, Y. Tian, R. Cheng, Y. Jin, An efficient approach to nondominated sorting for evolutionary multiobjective optimization, Evolutionary Computation, IEEE Transactions on 19 (2) (2015) 201–213.

[15] H. Fang, Q. Wang, T. Y., M. Horstemeyer, An efficient non-dominated sorting method for evolutionary algorithms, Evolutionary Computation 16 (3) (2008) 355–384.

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

[17] S. Tang, Z. Cai, J. Zheng, A fast method of constructing the non-dominated set: Arena's principle, 2013 International Conference on Computing, Networking and Communications (ICNC) 1 (2008) 391–395.

[18] D. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, 1st Edition, Addison-Wesley, 1989.

[19] M. Jensen, Reducing the run-time complexity of multiobjective eas: The nsga-ii and other algorithms, Evolutionary Computation, IEEE Transactions on 7 (5) (2003) 503–515.

[20] J. B. Mendes, J. A. de Vasconcelos, Using an adaptation of a binary search tree to improve the nsga-ii nondominated sorting procedure, in: Simulated Evolution and Learning, Vol. 6457 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2010, pp. 558–562.

[21] A. Berry, P. Vamplew, An efficient approach to unbounded bi-objective archives -: Introducing the mak_tree algorithm, in: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO '06, New York, NY, USA, 2006, pp. 619–626.

[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, Third Edition, 3rd Edition, The MIT Press,

2009.
[23] C. Hoare, Quicksort, The Computer Journal 5 (1) (1962) 10–16.
[24] M. Köppen, R. Vicente-Garcia, B. Nickolay, Fuzzy-pareto-dominance and its application in evolutionary multi-objective optimization, in: C. Coello Coello, A. Hernández Aguirre, E. Zitzler (Eds.), Evolutionary Multi-Criterion Optimization, Vol. 3410 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2005, pp. 399–412.
[25] K. Deb, L. Thiele, M. Laumanns, E. Zitzler, Scalable multi-objective optimization test problems, in: Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on, Vol. 1, 2002, pp. 825–830.