Merge Non-Dominated Sorting Algorithm for Many-Objective Optimization

Javier Moreno, Daniel Rodriguez, *Member, IEEE,* Antonio J. Nebro, and Jose A. Lozano *Senior Member, IEEE*

***Abstract*—Many Pareto-based multi-objective evolutionary algorithms require to rank the solutions of the population in each iteration according to the dominance principle, what can become a costly operation particularly in the case of dealing with many-objective optimization problems. In this paper, we present a new efficient algorithm for computing the non-dominated sorting procedure, called Merge Non-Dominated Sorting (MNDS), which has a best computational complexity of $O(NlogN)$ and a worst computational complexity of $O(MN^2)$, being N the population size and M the number of objectives. Our approach is based on the computation of the *dominance set*, i.e. for each solution, the set of solutions that dominate it, by taking advantage of the characteristics of the merge sort algorithm. We compare MNDS against six well-known techniques that can be considered as the state-of-the-art. The results indicate that the MNDS algorithm outperforms the other techniques in terms of number of comparisons as well as the total running time.**

***Index Terms*— Multi-objective optimization, non-dominated sorting, many objective problems, evolutionary algorithms.**

I. Introduction

Evolutionary algorithms (EAs) have been successfully applied in the solution of multi-objective optimization problems (MOPs) in the last two decades. These approaches can be mainly classified into Pareto-based, indicator-based and decomposition-based EAs. Most of algorithms belonging to the first group, which includes NSGA-II [1], SPEA2 [2] and many others [3], typically require to rank the population in the selection and replacement phases according to the dominance principle [4].

The non-dominated ranking procedure can be computationally significant in the total computing time of a multi-objective evolutionary algorithm (MOEA), particularly when dealing with many-objective problems, and large populations.

In this paper, we present the Merge Non-Dominated Sorting (MNDS) algorithm aimed at efficiently performing the non-dominated ranking. MNDS takes advantage of the characteristics of the merge sort algorithm to calculate the *dominance set*, i.e. the set of solutions that dominate other solution, for each solution. MNDS achieves a best computational complexity of $O(NlogN)$, while the worst case is $O(MN^2)$, where N corresponds to the population size and M is the number of objectives. As it usually happens with these kinds of algorithms, there is a time vs. memory trade-off. In our algorithm the storage of the *dominance set* of each solution allows a reduction of the computational time.

The rest of the paper is organized as follows. Section II briefly presents current works aiming to reduce the computational cost of the non-dominated sorting problem. Section III describes our proposal in detail. Experimental work and results are provided in Section IV. Finally, Section V highlights the conclusions and outlines future work.

II. Background and Related Work

Non-dominated sorting is based on the concept of Pareto-dominance between vectors (or solutions, in the context of EAs). Let P be a population of N solutions, $\{s_1, \dots, s_N\} \in P$, where each solution contains a vector of M objectives to minimize, $(f_1(s_i), \dots, f_M(s_i)), \forall i \in \{1, \dots, N\}$. A solution s_i dominates a solution s_j, denoted by $s_i \preceq s_j$, if the vector of objectives of s_i is partially less than the vector of objectives of s_j, i.e., $\forall m \in \{1, \dots, M\}, f_m(s_i) \leq f_m(s_j) \land \exists m' \in \{1, \dots, M\}$ $s.t.$ $f_{m'}(s_i) < f_{m'}(s_j)$ (we assume minimization without loss of generality). Given a set of solutions, those solutions which are non-dominated by any other are assigned rank 1. If these solutions are removed, then those solutions which are non-dominated by any other are assigned rank 2, and so on. Finding these ranks is called non-dominated sorting. Kung et al. [5] were the first to propose a method based on the divide-and-conquer idea to find maximal elements of a set of vectors, paving the way for further studies.

Javier Moreno and Daniel Rodriguez are with the University of Alcala, Spain. javier.morenom@edu.uah.es, daniel.rodriguezg@uah.es

Antonio Nebro is with University of Malaga, Spain. ajnebro@uma.es

Jose A. Lozano is with the University of the Basque Country and BCAM (Basque Centre for Applied Mathematics), Spain. ja.lozano@ehu.eus, jlozano@bcamath.org

Manuscript received ...; revised ...

TABLE I
COMPLEXITY OF NON-DOMINATED SORTING ALGORITHMS REPRESENTATIVE OF THE STATE-OF-THE-ART.

Algorithm	Complexity		
	Best Case	Worst Case	Space
FNDS [1]	MN^2	MN^2	N^2
Dominance Tree [6]	$MNlogN$	MN^2	MN
Deductive Sort [7]	$MN\sqrt{N}$	MN^2	N
Corner Sort [8]	$MN\sqrt{N}$	MN^2	N
ENS-SS [9]	$MN\sqrt{N}$	MN^2	1
ENS-BS [9]	$MNlogN$	MN^2	1
ENS-NDT [10]	$MNlogN$, *if* $M > logN$ $Nlog^2N$, *rest of cases*	MN^2	$NlogN$
M-Front [11]	MN	MN^2	MN^2
DDA-NS [12]	MN^2	MN^2	N^2
HNDS [13]	$MN\sqrt{N}$	MN^2	N
BOS [3]	$MNlogN$	MN^2	MN
MNDS	$NlogN$	MN^2	N^2

Reducing the complexity of non-dominated sorting is a matter of active research. The original implementation of NSGA (Non-dominated Sorting Genetic Algorithm) [4] had a complexity of $O(MN^3)$. A later version, in NSGA-II [1], the *Fast Non-dominated Sorting* reduced the cost to $O(MN^2)$. Table I shows both computational and spatial costs of the most representative algorithms for non-dominated sorting and how they compare against the two variants of our current proposal (MNDS).

We briefly summarize next the different strategies used by each of these algorithms (they are fully described in the provided references):

- *Fast Non-dominated Sorting (FNDS)* [1] compares each solution with the rest of the solutions of the population to obtain their dominance relationship. While carrying out this comparison, each solution stores those solutions that it dominates in a list. Once the comparisons are done, the lists of dominated solutions are traversed to rank them.
- *Dominance Tree* [6] uses a divide-and-conquer strategy to obtain the dominance relationships among the population solutions. These relationships are stored in a tree-like data structure called dominance tree.
- *Deductive Sort* [7] iterates through the population repeatedly, comparing the solutions one by one. Non-dominated solutions are assigned to the corresponding rank and eliminated from the population.
- *Corner Sort* [8] reduces the number of comparisons using two strategies: (i) as *Deductive Sort*, it avoids comparing solutions marked as dominated; the second strategy (ii) shows a preference for comparing corner solutions when determining the dominance between solutions.
- *Efficient Non-dominated Sort (ENS)* [9] calculates the rank of each solution at a time. To do so, it sorts the first objective using the lexicographical comparison[1]. Then, it looks for the rank of each solution using a sequential search strategy (version ENS-SS) or a binary search (version ENS-BS).
- *M-Front* [11] proposes to modify the typical MOEA's structure to improve their performance. In order to reduce the number of comparisons among solutions, the M-Front algorithm applies the geometric and algebraic properties of the Pareto dominance to perform interval queries using a nearest neighbor search. M-Front defines a special data structure named *archive* where all non-dominated individuals are stored. Additionally, M-Front stores all solutions in lists and uses a K-d tree for nearest neighbor search.
- *Hierarchical Non-Dominated Sorting (HNDS)* [13] minimizes the number of comparisons of objectives by ordering the population by the first objective and then by comparing the first solution with the rest of the solutions. These solutions are moved to an auxiliary list if they are not dominated by the first solution or a list of dominated solutions otherwise. The first solution is assigned to its corresponding rank and then the algorithm iterates until all the solutions are assigned their corresponding rank.
- *Dominance Degree Approach for Non-dominated Sorting (DDA-NS)* [12] is based on the concept of dominance degree matrix, which is a $N \times N$ square matrix where each column and row represents a solution $s_i : i \in \{1, \ldots, N\}$ and the cells contains the number of objectives in which each solution s_i dominates other solution s_j, $\forall s_i \neq s_j$ with $s_i, s_j \in P$. Once the dominance degree matrix

[1]The lexicographical comparison between two solutions compares the value of the objectives of both solutions starting from the first one. If the values are the same, then the second objectives are considered. This is carried out iteratively until the values are different or their objectives are exactly the same.

is obtained, DDA-NS traverses the matrix to find the maximum values of each column, gets their corresponding solution and assigns a rank to it.

- *Best Order Sort (BOS)* [3] sorts the population by each objective, resolving ties by means of lexicographical comparison. For each objective and solution s_i, it searches those solutions that are not worse than s_i. These solutions are stored in a set T associated with s_i. BOS will look at T for the s_j solution with the worst rank r. The rank of s_i will be $r + 1$.
- *Efficient Non-Dominated Sort with Non-Dominated Tree (ENS-NDT)* [10] extends the ENS-BS [9] algorithm using a new data structure, a variant of a bucket k-d tree, named *Non-Dominated Tree (NDTree)*. ENS-NDT is similar to ENS-BS but in the binary search it uses a NDTree instead of an array to store the fronts, speeding up the domination checking.

We must note that there are also some approaches related to the efficient computation of non-dominated sorting in steady-state multiobjective evolutionary algorithms [14]. However, our focus here is on the most general case of generational algorithms such as the standard NSGA-II. Those techniques should probably need to be adapted to work properly in this context, so we have not considered them in this work.

As we can observe in Table I, all algorithms have a computational complexity, in the worst case of $O(MN^2)$. The only algorithm that improves that complexity is Kung et al. [5] algorithm, that reaches a $O(MNlogN)$ computational complexity but it applies only to two objectives. Thus, the difference among the rest of algorithms lies in the average case and not in the worst case. In this sense, different algorithms apply different strategies to reduce the computational time.

Since to determine the dominance between two solutions it is necessary to compare the values of their objectives, a common quality indicator is the number of comparisons performed by an algorithm. For example, FNDS [1] compares all solutions among them and stores the result of each comparison to obtain the rank of each solution. HNDS [13] and ENS [9] [10] versions sort the population by the first objective value and use different data structures to differentiate a solution from the rest, and finally assign it a rank. BOS [3] sorts the population by each objective and assigns, to each solution s, M sets where it stores those solutions that are not worse than s in each objective. DDA-NS [12] sorts the population by each objective, and store in their dominance degree matrix the number of objectives in which one solution dominates the rest.

TABLE II
EXAMPLE OF A POPULATION WITH THEIR DOMINANCE SETS AND RANK

Population	Solution DS	Rank
s_1 = {34,30,41}	$s_1.ds = \{s_5\}$	2
s_2 = {33,34,30}	$s_2.ds = \emptyset$	1
s_3 = {32,32,31}	$s_3.ds = \emptyset$	1
s_4 = {31,34,34}	$s_4.ds = \emptyset$	1
s_5 = {34,30,40}	$s_5.ds = \emptyset$	1
s_6 = {36,33,32}	$s_6.ds = \{s_3\}$	2
s_7 = {35,31,43}	$s_7.ds = \{s_1, s_5\}$	3
s_8 = {37,36,39}	$s_8.ds = \{s_2,s_3,s_4,s_6,s_9\}$	3
s_9 = {35,34,38}	$s_9.ds = \{s_2,s_3,s_4\}$	2
s_{10} = {38,38,37}	$s_{10}.ds = \{s_2,s_3,s_4,s_6\}$	3
s_{11} = {39,37,31}	$s_{11}.ds = \{s_2,s_3\}$	2
s_{12} = {37,36,39}	$s_{12}.ds = \{s_2,s_3,s_4,s_6,s_9\}$	3

The MNDS strategy is quite straightforward. MNDS associates each solution to a total ordered set named *dominance set* ($s.ds$ in algorithms) containing the solutions that dominate it. More formally, given a solution $s_i \in P$, its *dominance set,* $s_i.ds$ contains all solutions that dominate s_i, i.e., each $s_j \in P$ such that $s_j \preceq s_i$. Table II shows a set of solutions along their dominance set and rank. For example, solution s_{10} is dominated by solutions s_2, s_3, s_4 and s_6, therefore, s_{10} dominance set is represented as $s_{10}.ds = \{s_2, s_3, s_4, s_6\}$. Once the dominance set of all solutions is computed, their rank is obtained as follows. Those solutions with an empty dominance set belong to the first rank. The rank of a solution with one or more elements in its dominance set will be calculated adding one to the largest rank of the solutions that compose the dominance set. Following our example, solutions s_2, s_3, s_4, and s_5 are not dominated and correspond to rank 1. The solution s_6 is dominated by s_3, i.e., $s_6.ds = \{3\}$, and as a result belongs to the second rank. Finally, the rank of s_{10} is $Max(R(s_2), R(s_3), R(s_4), R(s_6)) + 1 = 3$. Figure 1 shows a 3D representation of the example population in Table II. The three planes correspond to the three ranks. The higher the plane the higher the rank, and the lowest plane corresponds to the first rank.

To obtain the dominance set of each solution, MNDS sorts the population by each objective. The

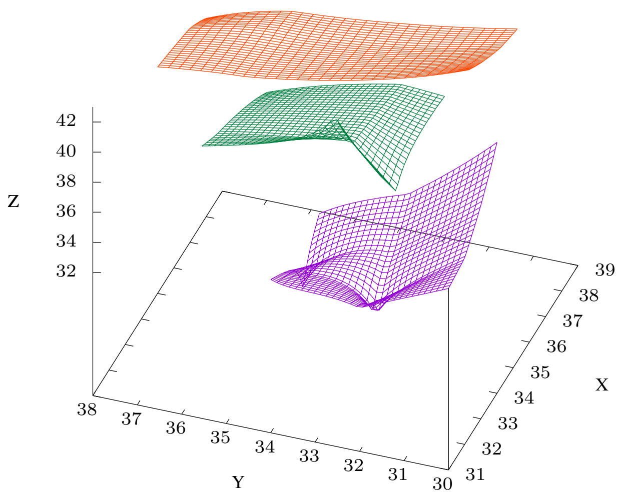


Fig. 1. 3D representation of the example population

TABLE III
EXAMPLE POPULATION SORTED BY OBJECTIVE 1

Population	ods	Solution DS
s_4 = {**31**,34,34}	$\emptyset$	$\emptyset$
s_3 = {**32**,32,31}	$\{s_4\}$	$\{s_4\}$
s_2 = {**33**,34,30}	$\{s_3,s_4\}$	$\{s_3,s_4\}$
${s_5}^2$ = {**34**,30,40}	$\{s_2,...,s_4\}$	$\{s_2,...,s_4\}$
${s_1}^2$ = {**34**,30,41}	$\{s_2,...,s_5\}$	$\{s_2,...,s_5\}$
${s_7}^2$ = {**35**,31,43}	$\{s_1,...,s_5\}$	$\{s_1,...,s_5\}$
${s_9}^2$ = {**35**,34,38}	$\{s_1,...,s_5,\ s_7\}$	$\{s_1,...,s_5,\ s_7\}$
s_6 = {**36**,33,32}	$\{s_1,...,s_5,\ s_7,\ s_9\}$	$\{s_1,...,s_5,\ s_7,\ s_9\}$
${s_8}^2$ = {**37**,36,39}	$\{s_1,...,s_7,\ s_9\}$	$\{s_1,...,s_7,\ s_9\}$
s_{10} = {**38**,38,37}	$\{s_1,...,s_9\}$	$\{s_1,...,s_9\}$
s_{11} = {**39**,37,31}	$\{s_1,...,s_{10}\}$	$\{s_1,...,s_{10}\}$

TABLE IV
EXAMPLE POPULATION SORTED BY OBJECTIVE 2

Population	ods	Solution DS
${s_5}^3$ = {34,**30**,40}	$\emptyset$	$\emptyset$
${s_1}^3$ = {34,**30**,41}	$\{s_5\}$	$\{s_5\}$
s_7 = {35,**31**,43}	$\{s_1,\ s_5\}$	$\{s_1,\ s_5\}$
s_3 = {32,**32**,31}	$\{s_1,\ s_5,\ s_7\}$	$\emptyset$
s_6 = {36,**33**,32}	$\{s_1,\ s_3,\ s_5,\ s_7\}$	$\{s_1,\ s_3,\ s_5,\ s_7\}$
${s_4}^3$ = {31,**34**,34}	$\{s_1,\ s_3,\ s_5,..,s_7\}$	$\emptyset$
${s_2}^3$ = {33,**34**,30}	$\{s_1,\ s_3,..,s_7\}$	$\{s_3,\ s_4\}$
${s_9}^3$ = {35,**34**,38}	$\{s_1,..,s_7\}$	$\{s_1,..,s_5,\ s_7\}$
s_8 = {37,**36**,39}	$\{s_1,..,s_7,\ s_9\}$	$\{s_1,..,s_7,\ s_9\}$
s_{11} = {39,**37**,31}	$\{s_1,..,s_9\}$	$\{s_1,..,s_9\}$
s_{10} = {38,**38**,37}	$\{s_1,..,s_9,\ s_{11}\}$	$\{s_1,..,s_9\}$

order of the solutions corresponds to their dominance relationship for that objective, i.e., the first solution is not dominated, the second is dominated by the first and so on. To do so, MNDS creates the *objective dominance set*, (ods). Once the population has been sorted by an objective, the ods is constructed traversing the ordered solutions and adding a solution to ods in each iteration. Therefore, we could define a function $ods(solution, objective)$ that returns the content of the ods considering the position of a solution sorted for an objective. The dominance set of a solution s will be $s.ds = ods(s,1) \cap ods(s,2) \cap ... \cap ods(s,M)$. Tables III, IV and V show the results of sorting the population on objectives 1, 2 and 3 respectively. Each table also shows the objective dominance set (ods) and the dominance set ($s.ds$) for each solution. It is worth noting that solution s_{12} does not appear in Tables III, IV and V as this solution is a duplicate of s_8 and MNDS removes duplicate solutions when sorting objective 1. In this way, the size of the population is reduced avoiding unnecessary operations. At the end those duplicate solutions are added back to the population with their rank (already calculated for the similar solution kept in the population). It can be observed that the dominance set in Table V corresponds to the actual one shown in Table II.

This strategy has two important advantages: (i) it minimizes the number of comparisons among objective values of the solutions, and (ii) it performs an early detection of non-dominance among the solutions of the population. When the dominance sets of all the solutions are empty, there is no dominance and the algorithm ends. This property is particularly advantageous when MNDS is used within multi-objective evolutionary algorithms such as NSGA, where the dominance among solutions decreases as the number of generations of the algorithm increases.

III. MERGE NON-DOMINATED SORTING

As already stated in the previous section, the overall idea behind our proposal is to obtain the dominance set of each solution in the population,

TABLE V
EXAMPLE POPULATION SORTED BY OBJECTIVE 3

Solution	ods	Solution DS
s_2 = {33,34,**30**}	$\emptyset$	$\emptyset$
s_3 = {32,32,**31**}	$\{s_2\}$	$\emptyset$
s_{11} = {39,37,**31**}	$\{s_2,\ s_3\}$	$\{s_2,\ s_3\}$
s_6 = {36,33,**32**}	$\{s_2,\ s_3,\ s_{11}\}$	$\{s_3\}$
s_4 = {31,34,**34**}	$\{s_2,\ s_3,\ s_6\ s_{11}\}$	$\emptyset$
s_{10} = {38,38,**37**}	$\{s_2,..,s_4,\ s_6,\ s_{11}\}$	$\{s_2,..,s_4,\ s_6\}$
s_9 = {35,34,**38**}	$\{s_2,..,s_4,\ s_6,\ s_{10},\ s_{11}\}$	$\{s_2,..,s_4\}$
s_8 = {37,36,**39**}	$\{s_2,..,s_4,\ s_6,\ s_9,..,s_{11}\}$	$\{s_2,..,s_4,\ s_6,\ s_9\}$
s_5 = {34,30,**40**}	$\{s_2,..,s_4,\ s_6,\ s_8,..,s_{11}\}$	$\emptyset$
s_1 = {34,30,**41**}	$\{s_2,..,s_6,\ s_8,..,s_{11}\}$	$\{s_5\}$
s_7 = {35,31,**43**}	$\{s_1,..,s_6,\ s_8,..,s_{11}\}$	$\{s_1,\ s_5\}$

[2]In case of ties, and only for the first objective, the lexicographic ordering is applied, as explained in the section III

[3]To equal values, merge sort keeps the order obtained from the previous objective. This is explained in detain in Section III

and then calculate their rank based on their corresponding dominance set. To obtain the dominance set for each solution s_i, it is necessary to sequentially sort the population by each of the objectives. The output obtained during the sorting of the m-th objective is the input for objective $m+1$-th. A key point of our approach is the treatment of ties. In the case of the first objective, a lexicographical[1] comparison is used to break ties. If there are ties for all objective values, the second solution is considered a duplicate. For the rest of objectives, ties are broken using the output of the previous iteration, i.e., the order obtained with the previous objective. This is automatically done by the merge sort[4] algorithm.

We now formalise this approach and provide an step-by-step example to illustrate our algorithm. In a population P, where each solution contains a vector of M objective values, the dominance set of solutions $s_i \in P$ can be obtained by sorting P iteratively by each objective as follows:

- For the first objective ($m = 1$), the individuals are sorted taking into account the objective function value of the first objective. When there are ties, a lexicographical order is used to rank the individuals. Once the population is sorted by the first objective, each solution keeps its ordinal position in a variable $s_i.index$. The index is used to identify each solution and to create the dominance set in each algorithm. It is worth noting that we do not need to create ods for the first objective.[5] Finally, duplicate solutions are removed from the population.
- For the second objective ($m = 2$), individuals are sorted by the objective function value of the second objective. In case of a tie, both solutions maintain the order obtained during the sorting of the first objective. Next, the dominance set of all solutions is initialised. The dominance set of each solution, $s.ds$, is composed of the indices contained in the set $ods(s, 2)$ whose values are lower than the solution index, $s.index$, i.e., $s.ds = \{u|u.index < s.index$ and $u \in ods(s, 2)\}$. Finally, dominance between solutions is checked and when there is no dominance, MNDS stops as all solutions belong the first rank.
- For the remaining objectives ($1 < m \leq M$), we sort the population (previously sorted by objective $m-1$) by each objective m. In case of a tie, both solutions maintain the order from the previous ($m - 1$ objective) sorting. The dominance set of the i-th solution s_i, in this order, is obtained by the $ods(s, m)$ intersected with the previous dominance set of s_i, i.e., $s_i.ds = ods(s_i, m) \cap s_i.ds$. As before, dominance is checked to decide whether to stop MNDS (all solutions belong to the first rank).

[4]Merge sort is a stable sorting algorithm, i.e., when it rearranges the population and there is a tie between two solutions, the relative position of both solutions in the population is maintained.

[5]For the first objective, the index of a solution $s \in P$ corresponds with its ordinal and therefore $s.ds = \{u|u.index < s.index$ and $u \in P\}$.

TABLE VI
EXAMPLE POPULATION. SORTFIRSTOBJECTIVE()

	Population	Index
Algorithm steps:	s_4 = {**31**,34,34}	0
- Sort population by first	s_3 = {**32**,32,31}	1
objective	s_2 = {**33**,34,30}	2
- Remove duplicated solutions	s_5 = {**34**,30,40}	3
- Assign a solution index	s_1 = {**34**,30,41}	4
	s_7 = {**35**,31,43}	5
	s_9 = {**35**,34,38}	6
	s_6 = {**36**,33,32}	7
	s_8 = {**37**,36,39}	8
	s_{12} = {**37**,36,39}	duplicated
	s_{10} = {**38**,38,37}	9
	s_{11} = {**39**,37,31}	10

After sorting by the last objective, the dominance set of each solution, $s_i.ds$, contains all the indices of the solutions that dominate s_i. The rank of a solution $s_i \in P$ will be the next rank to the highest rank of all the solutions $s_j \in s_i.ds$. In case that $s_i.ds$ is empty, s_i is assigned rank 1. Tables VI, VII and VIII show the result of applying the previous steps to the example population of Table II. It is worth noting that when sorting by the first objective, the solutions s_1 and s_5 have the same objective function value. Due to merge sort being a stable sorting algorithm, when comparing s_1 and s_5 lexicographically, we obtain that $s_5 \preceq s_1$ and the relationship between s_1 and s_5 will be maintained in case of ties when sorting by the next objectives. The treatment of duplicated is illustrated with individual s_{12}. In our example, solution s_{12} is a duplicate of solution s_8 and as a result, it is removed from the population while carrying out the sorting but added back again to the population after obtaining the ranking (due to most MOEAs need to keep their population size fixed).

A summary of the steps performed by MNDS with the sample population is shown in Table IX. The first two columns show the solutions sorted by

TABLE VII
EXAMPLE POPULATION SORTSECONDOBJECTIVE()

	Population	Idx	ods	Sol. DS
Algorithm steps:	s_5 = {34,**30**,40}	3	∅	∅
- Sort population by	s_1 = {34,**30**,41}	4	{3}	{3}
second objective	s_7 = {35,**31**,43}	5	{3,4}	{3,4}
- For each solution:	s_3 = {32,**32**,31}	1	{3,..,5}	∅
- Compute	s_6 = {36,**33**,32}	7	{1,3,..,5}	{1,3,..,5}
dominance set	s_4 = {31,**34**,34}	0	{1,3,..,5,7}	∅
- Check global	s_2 = {33,**34**,30}	2	{0,1,3,..,5,7}	{0,1}
dominance	s_9 = {35,**34**,38}	6	{0,..,5,7}	{0,...,5}
	s_8 = {37,**36**,39}	8	{0,..7}	{0,...,7}
	s_{11} = {39,**37**,31}	10	{0,..,8}	{0,...,8}
	s_{10} = {38,**38**,37}	9	{0,..,8,10}	{0,...,8}

TABLE VIII
EXAMPLE POPULATION SORTRESTOFOBJECTIVES()

	Population	Idx	ods	Sol. DS
- For obj = 3 to 3:	s_2 = {33,34,**30**}	2	∅	∅
- Sort population	s_3 = {32,32,**31**}	1	{1,2}	∅
- For each sol.:	s_{11} = {39,37,**31**}	10	{1,2,10}	{1,2}
- Compute	s_6 = {36,33,**32**}	7	{1,2,10}	{1}
dominance set	s_4 = {31,34,**34**}	0	{1,2,7,10}	∅
- Check global	s_{10} = {38,38,**37**}	9	{0,..,2,7,10}	{0,..,2,7}
dominance	s_9 = {35,34,**38**}	6	{0,..2,7,9,10}	{0,..,2}
	s_8 = {37,36,**39**}	8	{0,..2,6,7,9,10}	{0,..,2,6,7}
	s_5 = {34,30,**40**}	3	{0,..2,6,..,10}	∅
	s_1 = {34,30,**41**}	4	{0,..3,6,..,10}	{3}
	s_7 = {35,31,**43**}	5	{0,..4,6,..,10}	{3,4}

the first objective and their associated index. The next four columns show the value of each solution dominance set after sorting by objectives 2 and 3. The last two columns show the index and rank of each solution respectively. To further describe the example and following the steps previously described, the dominance set of solution s_7, for example, is obtained as follows:

- Objective 1: The population is sorted (from lowest to highest) using this first objective. After sorting the population, a index is assigned to each solution. In our example, s_7 appears ordered in the fifth position, $s_7.index = 5$. This means that all solutions with an index value less than 5 dominate s_7 in the first objective. These solutions are $s_4.index = 0$, $s_3.index = 1$, $s_2.index = 2$, $s_5.index = 3$ and $s_1.index = 4$.
- Objective 2: After sorting the population by objective 2, solutions with indices 3 and 4 dominate s_7, as a result, the dominance set of s_7 is initializated as $s_7.ds = \{3, 4\}$
- Objective 3: After sorting the population by objective 3, solution s_7 is dominated by the rest of the solutions. Therefore, $s_7.ds = s_7.ds \cap ods(s_7, 3) = \{3, 4\} \cap \{0, .., 4, 6, .., 10\} = \{3, 4\}$.

Finally, the ranks of the solutions are obtained based on the dominance sets and duplicates are inserted again with their corresponding rank. In our example, solution s_7 is dominated by solutions with indices 3 and 4, which are s_5 and s_1, respectively. The rank of $s_7 = Max\{R(s_5), R(s_1)\} + 1 = Max\{1, 2\} + 1 = 3$.

A. Formalization of the MNDS Algorithm

As it can be observed in Algorithm 1, MNDS receives the population to sort as the only parameter. The process followed by MNDS can be divided into the following four phases:

1) Sort the population by the *first objective* and assign the solution ordinal to the index variable ($s.index = ord(s)$). Ties are broken using lexicographical comparison and the duplicated solutions are moved to a list of $duplicates$ solutions (Algorithm 1, line 2). This list is composed of tuples *(duplicate solution, original solution)*. Note that although this requires more memory than just keeping the original solution with a list of duplicates, our solution is faster because it avoids searching through such list. The rank of duplicate solutions is assigned at the end of the Algorithm 1 (line 6).
2) Sort the population by the *second objective* and initialize the dominance set of each solution. If there is an iteration where all the dominance sets are empty, there is no dominance and MNDS ends since all solutions belong to the first rank.
3) Iteratively sort the population by the rest of the objectives $2 < m \leq M$. In case that in any iteration all the dominance sets are empty, i.e. there is no dominance, MNDS ends since all solutions belong to the first rank.
4) Calculate the rank of each solution.

TABLE IX
SUMMARY EXAMPLE

Order obj. 1→		Order obj. 2→		Order obj. 3→		Rank	
	Index	Index	Sol. DS	Index	Sol. DS	Index	Rank
s_4	0	3	∅	2	∅	2	1
s_3	1	4	{3}	1	∅	1	1
s_2	2	5	{3,4}	10	{1,2}	10	2
s_5	3	1	∅	7	{1}	7	2
s_1	4	7	{1,3,..,5}	0	∅	0	1
s_7	5	0	∅	9	{0,..,2,7}	9	3
s_9	6	2	{0,1}	6	{0,..,2}	6	2
s_6	7	6	{0,...,5}	8	{0,..2,6,7}	8	3
s_8	8	8	{0,...,7}	3	∅	3	1
s_{10}	9	10	{0,...,8}	4	{3}	4	2
s_{11}	10	9	{0,...,8}	5	{3,4}	5	3

Lines 2, 3 and 4 in Algorithm 1 correspond to the first three phases respectively. These phases are in turn further described in Algorithms 2, 3 and 4, respectively. The calculation of the ranking of each solution (phase 4) corresponds to lines 5 and 6.

Algorithm 1 Merge Non-Dominated Sorting(P)

```
Input: population $P$
Output: ranking for each solution $R$
1: $R \leftarrow \emptyset$
2: $duplicates \leftarrow$ SortFirstObjective($P$)
3: if SortSecondObjective($P$) then
4:     if SortRestOfObjectives($P$) then
5:         $R \leftarrow$ GetRanking($P$)
6:         Update the rank of each $duplicates$ solution with
           the rank of its original solution
7:     end if
8: end if
9: return $R$
```

Algorithms 2, 3 and 4 sort the population P by the objective O using Algorithm 6, $MergeSort(P, O)$. As previously stated, this algorithm is based on the merge sort algorithm. When sorting by the first objective ($O = 1$), in case of ties, the lexicographical comparison is applied (see Algorithm 6 line 2).

The method $SortFirstObjective(P)$ shown in Algorithm 2 implements the sorting by the first objective (phase 1). Line 3 sorts the population P by its first objective using the lexicographic rule in case of ties. Next, the loop (from lines 7 to 16) calculates the $index$ of each solution (lines 8, 10) and moves the duplicate solutions (see lines 12, 13) to the $duplicates$ list.

Algorithm 2 SortFirstObjective(P)

```
Input: population $P$
Output: population $P$, duplicate solutions $duplicates$
1: $ods \leftarrow \emptyset$          ▷ Dominance set for this objective. ods is
   implemented with a bitset
2: $duplicates \leftarrow \emptyset$
3: MergeSort($P, 1$)
4: $u \leftarrow P[1]$                              ▷ auxiliary solution $u$
5: $ordinal \leftarrow 1$
6: $u.index \leftarrow ordinal$
7: for $s : P$ do                       ▷ $s_i \in P, \forall i \in \{2, \ldots, |P|\}$
8:     $ordinal \leftarrow ordinal + 1$
9:     if $s \neq u$ then
10:        $s.index \leftarrow ordinal$
11:    else
12:        $duplicates \leftarrow duplicates \cup s$
13:        $P \leftarrow P - s$
14:    end if
15:    $u \leftarrow s$
16: end for
17: return $P, duplicates$
```

The method $SortSecondObjective(P)$ shown in Algorithm 3 implements the sorting by the second objective (phase 2). The loop (from lines 6 to 8) initialize the solution dominance set ($s.ds$) with solutions in ods with an index lower than $s.index$. Note that, at each iteration i, the dominance set ods contains the solutions that dominate the s_i solution for this objective.

Algorithm 3 SortSecondObjective(P)

```
Input: population $P$
Output: population $P$, hasDominance Boolean with whether
   there is dominance
1: $ods \leftarrow \emptyset$                ▷ Dominance set for this objective.
2: $hasDominance \leftarrow$ false
3: MergeSort($P, 2$)
4: for $s : P$ do
5:     $s.ds \leftarrow \emptyset$
6:     if subSet($ods, 1, s.index - 1$) $\neq \emptyset$ then
7:         $top \leftarrow Min(s.index - 1, ods.max)$
8:         $s.ds \leftarrow ods.subSet(ods.min, top)$
9:         $hasDominance \leftarrow$ true
10:    end if
11:    $ods \leftarrow ods \cup s.index$
12: end for
13: return $P, hasDominance$
```

The method $SortRestOfObjectives(P)$ shown in Algorithm 4 implements the third phase. The first loop iterates through all objectives except the first two. The calculation of the dominance sets is carried out by the internal loop (lines 7 to 13), which also evaluates if there is dominance among the solutions (line 10). When there is no further dominance, the method ends.

Algorithm 4 SortRestOfObjectives(P)

```
Input: population $P$
Output: population $P$, $hasDominance$ Boolean with whether there is dominance
1: $hasDominance \leftarrow$ true
2: $Obj \leftarrow 3$
3: while $Obj \leq M \wedge hasDominance$ do
4:     if MergeSort($P, Obj$) then
5:         $hasDominance \leftarrow$ false
6:         $ods \leftarrow \emptyset$        ▷ Dominance set for this objective
7:         for $s : P$ do
8:             $s.ds \leftarrow s.ds \cap ods$
9:             $ods \leftarrow ods \cup s.index$
10:            if $s.ds \neq \emptyset$ then
11:                $hasDominance \leftarrow$ true
12:            end if
13:        end for
14:    end if
15:    $Obj \leftarrow Obj + 1$
16: end while
17: return $P, hasDominance$
```

The last phase, the calculation of the population ranking, is implemented by the method $GetRanking(P)$. In this method, the variable $maxRank$ always contains the highest rank value of all evaluated solutions. Note that the rank of a solution s is always in the range $[1, maxRank + 1]$. The internal loop (lines 6 to 16) traverses the dominance set $s.ds$, obtaining the rank ($iR[i]$, line 7) of each solution in the current dominance set. If that value is greater than current $rank$, the $rank$ value is increased to $iR[i] + 1$ (line 8). Likewise, if the value of the $rank$ variable is greater than $maxRank$ (line 10), the $rank$ value is assigned to $maxRank$ and the search ends. Note that all dominance sets contain indices to solutions and the rank $iR[]$ is calculated taking into account these indices. The ranking of all solutions is stored in R in line 13.

B. Implementation considerations

We make use of *bitsets* to deal with sets operations. The motivation behind using bitsets to represent sets is their capability of maintaining the set sorted to facilitate the insertion of elements with a complexity of $O(1)$ while in other implementations, such as lists, their cost is $O(logN)$. It is worth noting that we use *sorted* sets to speed up the intersection operation between sets. Furthermore, our implementation of bitsets considers the range of the values in the set $[min, max]$, the intersection between two sets a and b is only applied within the range $[Max(a.min, b.min), Min(a.max, b.max)]$. Therefore, the intersection in Algorithm 4, line 8 will not be calculated if the solutions in ods do not dominate the solution s.

Algorithm 5 GetRanking(P)

```
Input: Population $P$
Output: Population Ranking $R$
1: $R \leftarrow \emptyset$
2: $iR \leftarrow \emptyset$        ▷ Ranking considering solution indices
3: $maxRank \leftarrow 0$
4: for $s : P$ do
5:     $rank \leftarrow 0$        ▷ Ranking of population solutions
6:     for $i : s.ds$ do        ▷ for each solution index $i$ in $s.ds$
7:         if $iR$[i] $\geq rank$ then
8:             $rank = iR$[i] $+ 1$
9:         end if
10:        if $rank > maxRank$ then
11:            $maxRank \leftarrow rank$
12:            $iR[i] \leftarrow rank$
13:            $R[s] \leftarrow rank$
14:            break
15:        end if
16:    end for
17: end for
18: return $R$
```

Algorithm 6 MergeSort(P, O)

```
Input: population $P$, objective $O$
Output: population $P$ sorted by objective $O$,
    $isSorted$ Boolean with whether input $P$ is already ordered
1: if O = 1 then
2:     $isSorted \leftarrow$ Sort $P$ by objective $O$. In case of ties,
       apply lexicographical order for objectives $O + 1$ to $M$
3: else
4:     $isSorted \leftarrow$ Sort $P$ by objective $O$.
5: end if
6: return $P$, $isSorted$
```

C. Computational and Spatial Complexity

MNDS is based on the merge sort algorithm (see Algorithm 6) which has a best and worst computational complexity of $O(NlogN)$. The computational complexity of MNDS (Algorithm 1), in the worst case scenario, is the sum of the complexities of the methods shown in Algorithms 2, 4 and 5, which are calculated as follows:

- Algorithm 2: The worst case belongs to the sorting which has a complexity of $O(NlogN)$.

- Algorithm 3: Sorting by objective 2 has a complexity $O(NlogN)$. The loop initializes the dominance set ($s.ds$) of all solutions, so its complexity is $O(N)$. Each $s.ds$ is initializated with the indices of those solutions in ods whose index is less than $s.index$. The worst case occurs when the first solution dominates the second, the second to the third and so on. In that case, the complexity is $O(N)$, so the complexity of Algorithm 3, in the worst case, is $O(N^2)$. The best case occurs when there is no dominance between solutions, or each solution is dominated only by another. In that case, the initialization of every $s.ds$ is $O(1)$, and the best case of Algorithm 3 is $O(NlogN)$.
- Algorithm 4: The inner loop calculates the dominance of all solutions in P ($O(N)$), which computes the intersection $s.ds \cap ods$ ($O(N)$) so this loop has a worst complexity of ($O(N^2)$). The external loop sorts ($O(NlogN)$) the population P for each objective except the first two, i.e., objectives 3 to M. Therefore, the complexity for this algorithm is $O((M-2)(NlogN+N^2))$. The best case occurs when there is no dominance between solutions, or each solution is dominated only by another. In that case, the calculation of the intersection $s.ds \cap ods$ has a complexity ($O(1)$) and the best case of Algorithm 4 is $O(NlogN)$. Please, note that Algorithm 4 also performs the early detection of non-dominance, in order to minimize the calculations as much as possible.
- Algorithm 5 is composed of two nested loops, so its worst computational complexity is $O(N^2)$. The best case of computational complexity occurs, once again, when each solution is dominated by another solution. In that case, the complexity is $O(N)$.

Therefore, the worst complexity of MNDS is the sum of $O(NlogN)$, $O(N^2)$, $O((M-2)(NlogN+N^2))$, and $O(N^2)$ which equals to $O(MN^2)$. The best case happens when there is no dominance among the solutions. In this case, at the end of the Algorithm 3 all the dominance sets of the solutions are empty. In this case MNDS ends and its complexity is $O(NlogN)$.

It is important to emphasize a difference between our proposal and the state-of-the-art algorithms. In our case, when the number of fronts decreases the algorithm tends to approximate to the behaviour of the best case. The spatial complexity is determined by the size of the *dominance sets* of each solution ($|s_i.ds| = |P|$) which corresponds to $O(N^2)$.

IV. Experimental Work

A. Implementation details

To validate the performance of MNDS[6], we compare the computational time of MNDS with six state-of-the-art algorithms: BOS [3], HNDS [13], ENS-SS [9], ENS-BS [9], ENS-NDT [10] and DDA-NS [12]. To do so, we use the BOS implementation provided by the authors[7]. For the ENS-SS and ENS-BS algorithms, the implementations provided by Buzdalov[8] were used with minimal modifications. ENS-NDT was implemented in Java from the C# source code provided by the authors. In addition, we implemented the HNDS and DDA-NS algorithms from scratch. In addition to computational time, in some experiments we also count the number of comparisons of objective function values, as it has been done in similar studies. We have to point out that most of the computing time in MNDS is not invested in comparing the objective function values of the solutions, so a good performance of our algorithm is expected.

The implementation of all the algorithms was done in Java without using multithreading nor specific CPU/GPU features as SIMD[9] or similar.

B. Experimental settings

In order to compare the algorithms previously discussed, four types of experiments were carried out:

1) Varying the number of objectives for a fixed number of solutions, using the BOS dataset[7].
2) Varying the population size for a fixed number of objectives, using again the BOS dataset.
3) Varying the number of objectives for a fixed population size, using datasets generated by NSGA-II. In this case, we have additionally obtained the number of comparisons made by each algorithm.

[6]MNDS is integrated into the jMetal framework: `https://github.com/jMetal/jMetal`

[7]`https://github.com/Proteek/Best-Order-Sort`

[8]`https://github.com/mbuzdalov/non-dominated-sorting`

[9]Single Instruction, Multiple Data.

4) Executing the algorithms within the NSGA-II algorithm. To do so, we have replaced the original FNDS algorithm in NSGA-II for each of the evaluated algorithms.

The original BOS dataset contains 10,000 solutions with up to 10 objectives; we extended it to 20 objectives, generating the new values randomly.

In experiment 1), the algorithms were executed varying the number of objectives between 3 and 20, with population sizes of 500, 1,000, 5,000. In experiment 2), the size of the population ranged between 500 and 10,000 with an increase of 1,000 for 5, 10, 15 and 20 objectives. For experiment 3), the NSGA-II [1] implementation from jMetal [15] was used to generate 16 datasets obtained after 500 generations for the DTLZ1 [16], DTLZ2 [16], WFG1 [17] and WFG2 [17] problems with 5, 10, 15 and 20 objectives. The population size used was 1000 solutions. All the algorithms were executed 5,000 times under the same conditions using the execution time as performance measure. The final execution time was calculated averaging those 5,000[10] executions. It is worth noting that the NSGA-II was used with the same problems and applying the same configuration as the one defined in the paper describing BOS. In this way, when comparing MNDS against BOS, we are also comparing MNDS, indirectly, with the other algorithms that were also compared with the BOS algorithm, i.e., fast non-dominated sorting, deductive sort, corner sort and divide-and-conquer sort. Finally, in experiment 4, NSGA-II was configured to run for 2,000 generations with a population size of 1,000 solutions. The crossover operator used was the simulated binary crossover, with a distribution index value of 20 and a crossing probability of 90%. As a mutation operator we used the polynomial mutation. All the executions used the same random-seed, and therefore, with the same initial population. The DTLZ and WFG algorithms were configured for 5 objectives.

The computer and software versions used have the following features:

- Debian GNU/Linux 9.0. 64 bits architecture.
- 4 x Intel® Core$^{\text{TM}}$ i5 CPU M-460. 2.53GHz.
- 8GB of RAM Memory.
- Java version: 1.8.0-121, 64 bits.

[10]This number of executions mitigates possible differences in runtime values due to the behavior of the just-in-time compiler and the garbage collector of Java.

C. Results

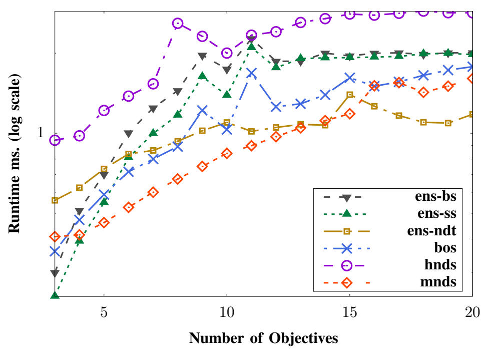
The execution times obtained in the experiments, by all the algorithms, but DDA-NS, are shown in Figures 2, 3, 4 and 5. We have excluded DDA-NS from the figures due to the large differences in performance with the rest of the algorithms. The MNDS algorithm was designed to work efficiently with large population sizes as well as with a large number of objectives. As a result, the algorithm maintains a very high performance even if we increase the number of objectives or the size of the population. With few objectives (≤ 5) and/or a small number of solutions, the behavior of the compared techniques is similar, except for HNDS and DDA-NS which perform worse than the rest. However, it can be observed in the figures that (i) as the number of objectives increases or the size of the population increases, the rest of the algorithms suffer a performance degradation; (ii) only BOS and ENS-NDT algorithms present a performance close to MNDS when using the BOS dataset.

The computing times obtained with the datasets generated by NSGA-II in experiment 3, indicate that the differences with the other algorithms are noticeable. The results of experiment 3 can be observed in Figure 4. As stated in Section III-C, MNDS tends to approximate to the behaviour of the best case when the number of fronts decreases. Such behaviour can be observed in Figure 5 where MNDS outperforms the rest of the algorithms when they are running within NSGA-II. We have to note that in this experiment we are reporting the execution time of each NSGA-II iteration as performance measure; we made 10 independent runs but the differences in the running times were negligible (the standard deviations were very small) to be observed in the graphs, so we decided to plot a single value.

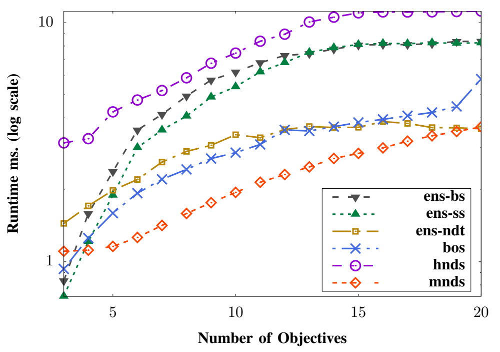
The number of comparisons made by each algorithm in experiment 3) is shown in Table X, where we can observe that MNDS requires a number of comparisons that is at least one order of magnitude lower than the best of the other algorithms compared. This result confirms the expectations we claimed in Section IV-A

V. Conclusions and Future Work

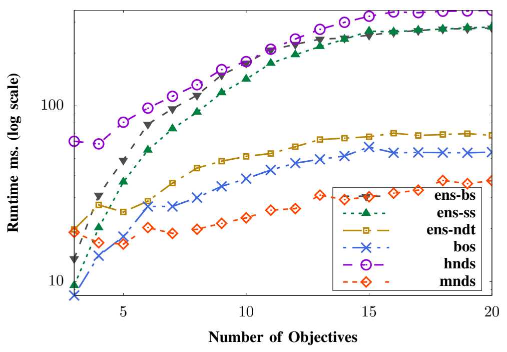
In this paper, we have presented a new and efficient algorithm for computing the non-dominated sorting called Merge Non-Dominated



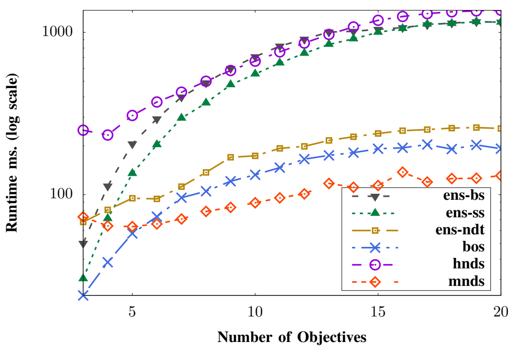
(a) 500 Solutions



(b) 1,000 Solutions

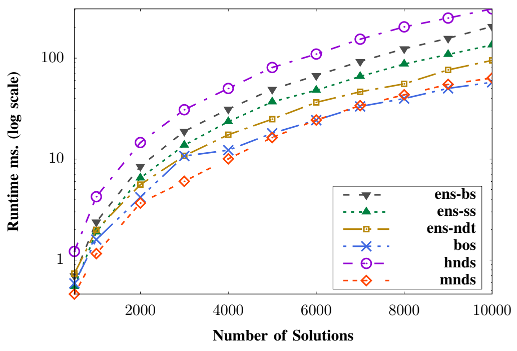


(c) 5,000 Solutions

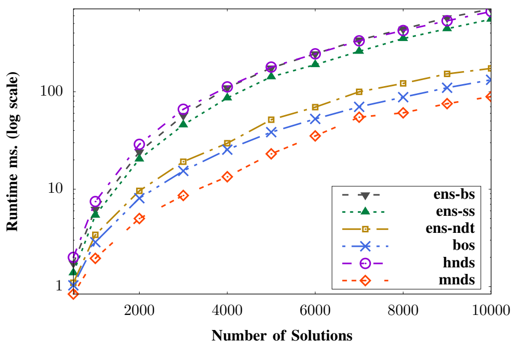


(d) 10,000 Solutions

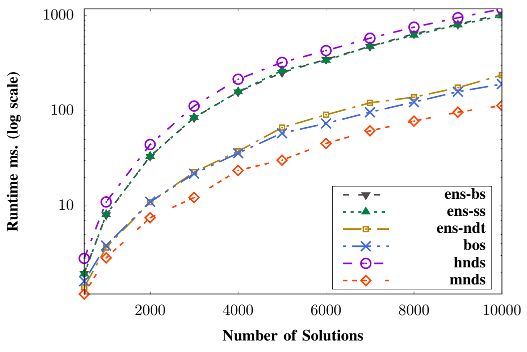
Fig. 2. Experiment 1. Results with a fixed number of solutions, and increasing the number of objectives, using the BOS dataset.



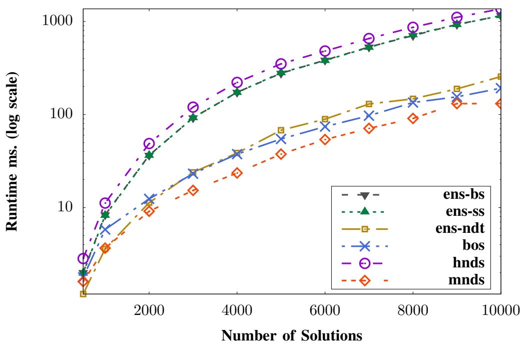
(a) 5 Objectives



(b) 10 Objectives

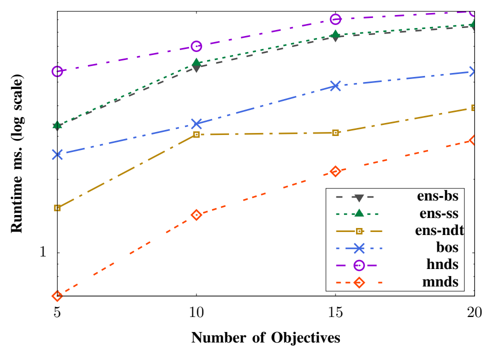


(c) 15 Objectives

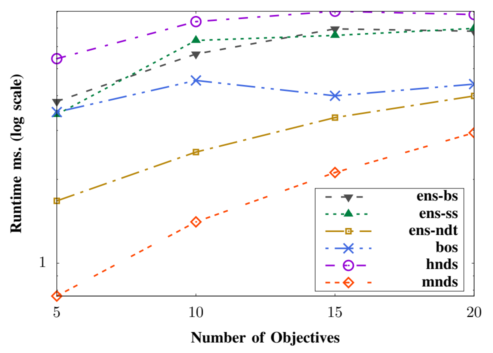


(d) 20 Objectives

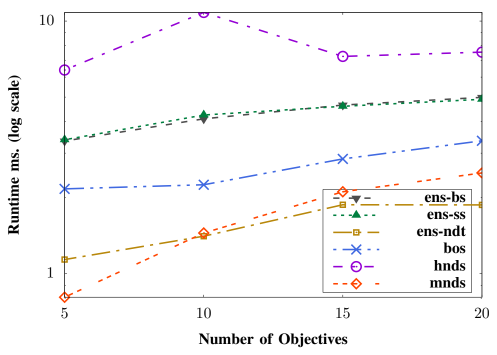
Fig. 3. Experiment 2. Results with a fixed number of objectives, and increasing the number of solutions, using the BOS dataset.



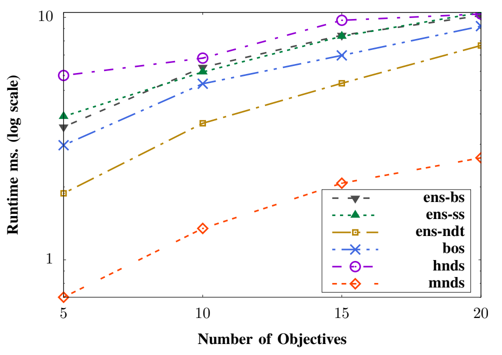
(a) DTLZ1



(b) DTLZ2



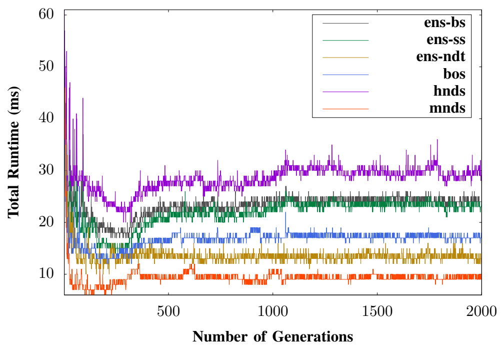
(c) WFG1



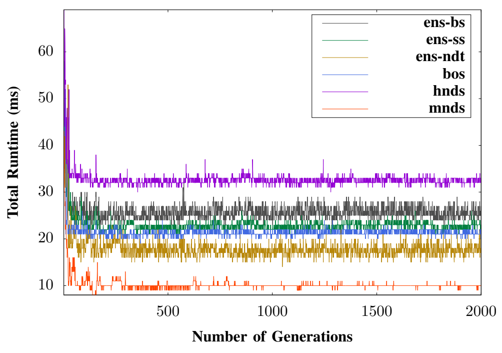
(d) WFG2

Fig. 4. Experiment 3. Results with the Dataset Generated by NSGA-II with a population of 1,000 solutions after 500 generations

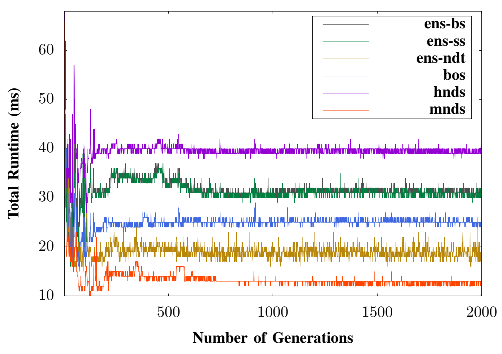
.



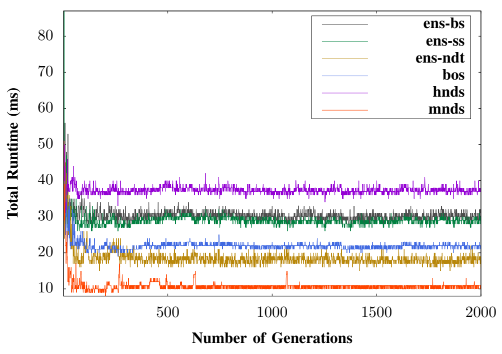
(a) DTLZ1



(b) DTLZ2



(c) WFG1



(d) WFG2

Fig. 5. Experiment 4. Results executing the algorithms within NSGA-II for 2,000 generations with a population of 1,000 solutions.

TABLE X
EXPERIMENT 3. NUMBER OF COMPARISONS MADE BY THE ALGORITHMS WITH THE DATASET GENERATED BY NSGA-II WITH A POPULATION OF 1,000 SOLUTIONS AFTER 500 GENERATIONS.

	Obj	BOS	ENS-SS	ENS-BS	ENS-NDT	HNDS	DDA-NS	MNDS
DTLZ1	5	5.37e+08	2.40e+06	2.40e+06	4.20e+08	1.97e+06	5.79e+07	**6.84e+04**
	10	5.08e+08	3.83e+06	3.83e+06	1.68e+09	2.68e+06	1.20e+08	**1.34e+05**
	15	7.10e+08	2.40e+06	2.40e+06	6.53e+08	1.97e+06	5.83e+07	**6.88e+04**
	20	6.13e+08	6.78e+06	6.78e+06	2.34e+09	4.15e+06	2.41e+08	**2.66e+05**
DTLZ2	5	6.65e+08	2.50e+06	2.50e+06	6.65e+08	2.01e+06	5.96e+07	**6.75e+04**
	10	5.31e+08	3.70e+06	3.70e+06	9.71e+08	2.61e+06	1.21e+08	**1.34e+05**
	15	4.88e+08	4.26e+06	4.26e+06	1.35e+09	2.89e+06	1.81e+08	**2.00e+05**
	20	5.31e+08	5.39e+06	5.39e+06	2.10e+09	3.46e+06	2.41e+08	**2.67e+05**
WFG1	5	5.05e+08	2.26e+06	2.26e+06	7.60e+08	1.94e+06	5.46e+07	**7.04e+04**
	10	3.56e+08	2.86e+06	2.86e+06	6.87e+08	2.19e+06	1.18e+08	**1.35e+05**
	15	3.50e+08	3.39e+06	3.39e+06	6.51e+08	2.46e+06	1.77e+08	**2.02e+05**
	20	3.75e+08	2.88e+06	2.88e+06	5.30e+08	2.20e+06	2.38e+08	**2.57e+05**
WFG2	5	4.97e+08	2.12e+06	2.12e+06	3.21e+08	1.82e+06	6.00e+07	**6.82e+04**
	10	7.45e+08	3.80e+06	3.80e+06	1.50e+09	2.66e+06	1.20e+08	**1.35e+05**
	15	9.20e+08	5.48e+06	5.48e+06	2.77e+09	3.50e+06	1.80e+08	**2.01e+05**
	20	1.18e+09	9.07e+06	9.07e+06	5.28e+09	5.30e+06	2.40e+08	**2.54e+05**

Sorting (MNDS) based on the merge sort algorithm. The experimental work showed that MNDS strongly outperforms the current state of the art algorithms in terms of running time and number of comparisons carried out.

As future work we plan to enhance our approach in several ways. Particularly, we think that the algorithm used to calculate the ranking of each solution from the domination sets could be improved by the use of different search methods and data structures. For example, we could use the Timsort algorithm instead of merge sort, and the sequential search used in finding the ranking of each solution could be replaced by a binary search or a k-d tree. We will also explore issues such as that the performance of the MOEAs using MNDS will increase as the number of non-dominated solutions also increases. This circumstance occurs in all MOEA algorithms, i.e., as the generations advance, the number of non-dominated solutions also increases. Finally, we will also consider to adapt our algorithm to be used in steady-steady evolutionary algorithms.

ACKNOWLEDGMENTS

Javier Moreno and Daniel Rodriguez thank the University of Alcala, Daniel Rodriguez is also supported by projects TIN2016-76956-C3-3-R, uDEVOPS 823992 and RED2018-102472-T. Jose A. Lozano is supported by the Basque Government through the BERC 2018-2021 program and research group grant IT1244-19 and by the Spanish Ministry of Science, Innovation and Universities: BCAM Severo Ochoa accreditation SEV-2017-0718 and TIN2016-78365-R. Antonio J. Nebro is supported by grant TIN2017-86049-R (Spanish Ministry of Education and Science).

REFERENCES

[1] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr 2002.

[2] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm," Tech. Rep., 2001.

[3] P. Roy, M. Islam, and K. Deb, *Best order sort: A new algorithm to non-dominated sorting for evolutionary multi-objective optimization*. Association for Computing Machinery, Inc, 7 2016, pp. 1113–1120.

[4] N. Srinivas and K. Deb, "Multiobjective optimization using nondominated sorting in genetic algorithms," *Evol. Comput.*, vol. 2, no. 3, pp. 221–248, Sep. 1994. [Online]. Available: http://dx.doi.org/10.1162/evco.1994.2.3.221

[5] H. T. Kung, F. Luccio, and F. P. Preparata, "On finding the maxima of a set of vectors," *J. ACM*, vol. 22, no. 4, pp. 469–476, Oct. 1975. [Online]. Available: http://doi.acm.org/10.1145/321906.321910

[6] H. Fang, Q. Wang, Y.-C. Tu, and M. F. Horstemeyer, "An efficient non-dominated sorting method for evolutionary algorithms," *Evol. Comput.*, vol. 16, no. 3, pp. 355–384, 9 2008. [Online]. Available: http://dx.doi.org/10.1162/evco.2008.16.3.355

[7] K. McClymont and E. Keedwell, "Deductive sort and climbing sort: New methods for non-dominated sorting," *Evol. Comput.*, vol. 20, no. 1, pp. 1–26, Mar. 2012. [Online]. Available: http://dx.doi.org/10.1162/EVCO_a_00041

[8] H. Wang and X. Yao, "Corner sort for pareto-based many-objective optimization," *IEEE Trans. Cybern.*, vol. 44, no. 1, pp. 92–102, Jan 2014.

[9] X. Zhang, Y. Tian, R. Cheng, and Y. Jin, "An efficient approach to nondominated sorting for evolutionary multiobjective optimization," *IEEE Trans. Evol. Comput.*, vol. 19, no. 2, pp. 201–213, April 2015.

[10] P. Gustavsson and A. Syberfeldt, "A new algorithm using the non-dominated tree to improve non-dominated sorting," *Evol. Comput.*, vol. 26, no. 1, pp. 89–116, Mar. 2018. [Online]. Available: https://doi.org/10.1162/evco_a_00204

[11] M. Drozdík, Y. Akimoto, H. Aguirre, and K. Tanaka, "Computational cost reduction of nondominated sorting using the m-front," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 5, pp. 659–678, Oct 2015.

[12] Y. Zhou, Z. Chen, and J. Zhang, "Ranking vectors by means of the dominance degree matrix," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 1, pp. 34–51, Feb 2017.

[13] C. Bao, L. Xu, E. D. Goodman, and L. Cao, "A novel non-dominated sorting algorithm for evolutionary multi-objective optimization," *Journal of Computational Science*, vol. 23, pp. 31 – 43, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877750317310530

[14] K. Li, K. Deb, Q. Zhang, and Q. Zhang, "Efficient nondomination level update method for steady-state evolutionary multiobjective optimization," *IEEE Transactions on Cybernetics*, vol. 47, no. 9, pp. 2838–2849, Sep. 2017.

[15] J. J. Durillo and A. J. Nebro, "jMetal: A java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, no. 10, pp. 760–771, Oct. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.advengsoft.2011.05.014

[16] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable multi-objective optimization test problems," in *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, vol. 1, May 2002, pp. 825–830.

[17] S. Huband, P. Hingston, L. Barone, and L. While, "A review of multiobjective test problems and a scalable test problem toolkit," *IEEE Trans. Evol. Comput.*, vol. 10, no. 5, pp. 477–506, Oct 2006.