An Efficient Approach to Non-dominated Sorting for Evolutionary Multi-objective Optimization

Xingyi Zhang, Ye Tian, Ran Cheng, and Yaochu Jin, *Senior Member, IEEE*

***Abstract*—Evolutionary algorithms have been shown to be powerful for solving multi-objective optimization problems, where non-dominated sorting is a widely adopted technique in selection. This technique, however, can be computationally expensive, especially when the number of individuals in the population becomes large. This is mainly due to the fact that in most existing non-dominated sorting algorithms, a solution needs to be compared with all other solutions before it can be assigned to a front. In this work, we propose a novel, computationally efficient approach to non-dominated sorting, termed efficient non-dominated sort (ENS). In ENS, a solution to be assigned to a front needs to be compared only with those that have already been assigned to a front, thereby avoiding many unnecessary dominance comparisons. Based on this new approach, two non-dominated sorting algorithms have been suggested. Both theoretical analysis and empirical results show that the ENS-based sorting algorithms are computationally more efficient than the state-of-the-art non-dominated sorting methods.**

***Index Terms*—Evolutionary multi-objective optimization, non-dominated sorting, Pareto-optimality, computational complexity**

I. Introduction

MOST REAL-WORLD optimization problems are characterized by multiple objectives which often conflict with each other. For solving such multi-objective optimization problems (MOPs), a set of optimal solutions, known as Pareto-optimal solutions, instead of a single optimal solution, are to be achieved. Most classical optimization methods are inefficient in solving MOPs, since they can typically find only one Pareto-optimal solution in one run, which means that this kind of method has to be applied multiple times to achieve a Pareto-optimal solution set.

Over the past 20 years, a variety of evolutionary algorithms have been developed to tackle MOPs, e.g., PESA-II [1], NSGA-II [2], SPEA2 [3], and M-PAES [4], to name just a few. These multi-objective evolutionary algorithms (MOEAs) are able to find a set of Pareto-optimal solutions in one single run.

Although various approaches have been adopted for selection [5], most MOEAs adopt the Pareto-based approach, i.e., the qualities of the candidate solutions are compared using Pareto dominance. Among various dominance comparison mechanisms, non-dominated sorting [2] has been shown to be very effective for finding Pareto-optimal solutions. Much work has also been done to efficiently store non-dominated solutions found during search in an archive [6], [7]. Non-dominated sorting is a procedure where solutions in the population are assigned to different fronts based on their dominance relationships. Without loss of generality, we assume that the individuals in population P can be categorized into K Pareto fronts, denoted as F_i, $i = 1, \ldots, K$. According to non-dominated sorting, all non-dominated solutions in population P are assigned to front F_1; then the non-dominated solutions in $P - F_1$, which is the set of solutions by removing the solutions assigned to front F_1, are assigned to front F_2. This procedure repeats until all solutions in P are assigned to a front F_i, $i = 1, \ldots, K$. Note that the solutions belonging to front F_j are dominated by at least one solution belonging to front F_i, if $i < j$, $i, j = 1, 2, \ldots, K$. Fig. 1 provides an illustrative example of a population of 13 solutions composed of four fronts, where both objectives are to be minimized.

Non-dominated sorting is computationally intensive, in particular when the population size increases. To address this problem, much research work has been dedicated to the improvement of the computational efficiency of this procedure. The idea of non-dominated sorting was first suggested in [8] as a selection strategy for evolutionary multi-objective optimization, which was implemented in a multi-objective genetic algorithm, termed

This work was supported by grants from National Natural Science Foundation of China, Projects 61272152, 61033003, 91130034, 61373066, 61073116, 61003131 and 61202011, by a grant from Ph.D. Programs Foundation of Ministry of Education of China, Project 20100142110072, by a grant from Fundamental Research Funds for the Central Universities, Project 2010ZD001, by grants from Natural Science Foundation of Anhui Higher Education Institutions of China, Projects KJ2012A010 and KJ2013A007, and by a grant from Scientific Research Foundation for Doctor of Anhui University, Project 02203104.

X. Zhang and Y. Tian are with the Key Lab of Intelligent Computing and Signal Processing of Ministry of Education, School of Computer Science and Technology, Anhui University, Hefei 230039, China (email: xyzhanghust@gmail.com; field910921@gmail.com)

R. Cheng and Y. Jin are with the Department of Computing, University of Surrey, Guildford, Surrey, GU2 7XH, United Kingdom (r.cheng@surrey.ac.uk; yaochu.jin@surrey.ac.uk)

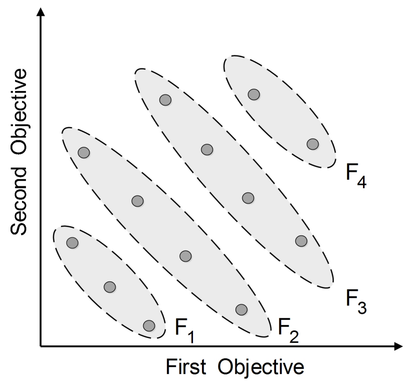


Fig. 1. A population with 13 solutions of a bi-objective minimization problem. The individuals can be divided into four fronts.

non-dominated sorting GA (NSGA) [9]. Non-dominated sorting in NSGA has a time complexity of $O(MN^3)$ and a space complexity of $O(N)$, where M is the number of objectives and N is the number of solutions in the population. A faster version of non-dominated sorting, termed fast non-dominated sort was proposed in [2], where the time complexity is reduced to $O(MN^2)$. The fast non-dominated sort, however, requires a larger storage space than the non-dominated sorting in NSGA, which is increased to $O(N^2)$. Jensen [10] adopted a divide-and-conquer strategy for non-dominated sorting, the time complexity of which is $O(N \log^{M-1} N)$. Tang *et al.* [11] proposed a novel non-dominated sorting approach based on arena's principle, i.e., each winner will be the next "arena host" to be challenged. This approach has been proved to have the same time complexity as the fast non-dominated sort, while empirical results show that it outperforms the fast non-dominated sort in terms of computational efficiency, since it can achieve a time complexity $O(MN\sqrt{N})$ in some best cases. Clymont and Keedwell [12] proposed two improved approaches to non-dominated sorting, called climbing sort and deductive sort, where some dominance relationships between solutions can be inferred based on recorded comparison results.

In this work, we propose a new, computationally efficient approach to non-dominated sorting, called efficient non-dominated sort (ENS). ENS adopts an idea different from those used in the above-mentioned methods. The main difference lies in the fact that existing non-dominated sorting approaches usually compare a solution with all other solutions in the population before assigning it to a front, whilst ENS compares it only with those that have already been assigned to a front. This is made possible by the fact that in ENS, the population is sorted in one objective before ENS is applied. Thus, a solution added to the fronts cannot dominate any solutions that are added before. As a result, ENS can avoid a large number of redundant dominance comparisons, which significantly improves the computational efficiency. Theoretical analysis shows that the ENS approach has a space complexity of $O(1)$, which is smaller than all existing non-dominated sorting methods. Meanwhile, the time complexity of ENS will be $O(MN \log N)$ in good cases, which is much lower than that of all existing algorithms. Even in the worst case, ENS has a complexity of $O(MN^2)$, which is the same as the fast non-dominated sort. Experimental results confirm that ENS has better computational efficiency than the state-of-the-art.

The rest of this paper is organized as follows. In Section II, we briefly review a few widely used non-dominated sorting approaches and analyze their computational complexity. In Section III, we propose a new approach to non-dominated sorting, ENS, based on which two non-dominated sorting algorithms are developed. The computational complexities of the two algorithms are then analyzed. Simulation results are presented in Section IV to empirically compare the two ENS-based non-dominated sorting algorithms with three state-of-the-art methods. Finally, conclusions and remarks are given in Section V.

II. Related Work

In this section, we review a few popular non-dominated sorting approaches together with an analysis of their computational complexities.

A. Non-dominated Sorting Methods

Since Goldberg [8] suggested the use of non-dominated sorting for selection in MOEAs, a number of non-dominated sorting methods have been reported in the literature over the past years. Below we review a few non-dominated sorting approaches widely used in MOEAs.

The non-dominated sorting strategy was first adopted for selecting parents from offspring in NSGA for multi-objective optimization [9]. The non-dominated sorting in NSGA is carried out as follows. Each solution is compared with all other solutions in the population, and solutions which are not dominated by any other solutions are assigned to front F_1. All solutions assigned to F_1 are temporarily removed from the population. Then each solution in the remaining population is compared with others, and all non-dominated solutions are assigned to front F_2. This operation is repeated until all solutions have been assigned to a front. This approach contains many redundant comparisons in the sense that the comparison between two solutions may be performed more than once. The time complexity of this approach is $O(MN^3)$, which makes NSGA highly time-consuming and computationally inefficient for large populations. As an improved version of NSGA, Deb *et al.* [2] proposed a computationally more efficient non-dominated sorting approach, called fast non-dominated sort, where the comparison between any two solutions is performed only once. Fast non-dominated sort has a time complexity of $O(MN^2)$, albeit at the cost of an increased space complexity from $O(N)$ to $O(N^2)$.

A recursive non-dominated sorting approach [10], usually called Jensen's sort, was suggested based on the divide-and-conquer mechanism, which reduces the time complexity to $O(MN \log N)$ for bi-objective MOPs, and to $O(N \log^{M-1} N)$ for MOPs having more than two objectives. The space complexity of this approach is $O(1)$ and $O(N)$ for MOPs with two objectives and more than two objectives, respectively. Just as shown in $O(N \log^{M-1} N)$, the time complexity of Jensen's sort will grow exponentially with the increment of number of objectives. This means that Jensen's sorting method will not work efficiently for MOPs with a large number of objectives. Actually, this sorting method will likely consume more runtime in simulation due to its recursive nature. In addition, as Clymont and Keedwell [12] and Fang *et al.* [13] pointed out, Jensen's sorting algorithm is not applicable in many cases, for instance, when strong-dominance [14] or ϵ-dominance [15] is used in comparison, or when the population contains duplicate solutions.

Tang *et al.* [11] used arena's principle to assign solutions to a front, which has been shown to have a better computational efficiency than the fast non-dominated sort and Jensen's sort in empirical evaluations. This approach randomly selects one solution from the population, regarded as an "arena host", and all the remaining solutions in the population are compared with the "arena host". The solution which dominates the "arena host" will become the new "arena host" to replace the current one. The time complexity and space complexity of this approach are $O(MN^2)$ and $O(N)$, respectively.

Clymont and Keedwell [12] proposed two non-dominated sorting approaches: climbing sort and deductive sort. As shown in [12], deductive sort often performs better than climbing sort. Deductive sort infers the dominance relationship between solutions by recording the results of comparisons, thereby avoiding some unnecessary comparisons. Deductive sort holds a time complexity of $O(MN^2)$ and a space complexity of $O(N)$, which outperforms other approaches, for instance, the fast non-dominated sort.

There are a few other non-dominated sorting approaches inspired by different ideas, such as the non-dominated rank sort of the omni-optimizer [16], better non-dominated sort [17], immune recognition based algorithm [18], quick sort [19], sorting based algorithm [20], and divide-and-conquer based non-dominated sorting algorithm [13]. Most of these approaches are effective in dealing with MOPs that have a small number of objectives, however, their efficiency often seriously degrades as the number of objectives increases.

B. Analysis of Existing Methods

Although existing non-dominated sorting approaches perform front assignments based on various ideas, most of them can be described in a generic framework as shown in Fig. 2. In this framework, solutions in different fronts are assigned front by front. For example for a population P containing K fronts F_i, $1 \leq i \leq K$, all non-dominated solutions in P are first assigned to front F_1. Once this is done, the non-dominated solutions in $P - F_1$ (the remaining population with all solutions assigned to F_1 being removed) can then be assigned to F_2. In other words, solutions belonging to front F_{i+1} cannot be assigned until all solutions belonging to F_i have been assigned.

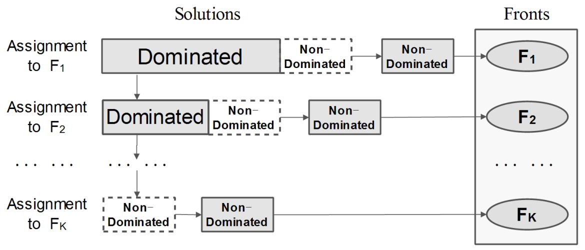


Fig. 2. An illustration of the commonly used strategy for non-dominated sorting in most existing non-dominated sorting approaches, which determines the front number of all solutions on the same front all at once, and solutions on different fronts sequentially.

Dominance comparisons between the solutions are the main operation in non-dominated sorting, i.e., the number of needed comparisons determines the efficiency of a non-dominated sorting approach. Most existing non-dominated sorting methods focus on the reduction of the number of comparisons to improve their computational efficiencies. The reason is that some dominance comparisons between solutions are unnecessary and can be spared. Taking a closer look, we find that the result of one dominance comparison can be categorized into the following four cases, assuming that solution p_m is compared with solution p_n:

- Case 1: p_m is dominated by p_n, or p_n is dominated by p_m.
- Case 2: p_m and p_n are non-dominated, and they belong to the same front F_i, where F_i is the current front (i.e., the front which the solutions are being assigned to).
- Case 3: p_m and p_n are non-dominated, and they belong to the same front F_i, where F_i is not the current front.
- Case 4: p_m and p_n are non-dominated, but they belong to different fronts.

Recall that a solution is assigned to the current front if it is not dominated by any other solutions in the current population. In case 1, if solution p_m dominates solution p_n, then p_n does not belong to the current front and we no longer need to perform additional comparisons between p_n and all other solutions in the current population, which means that such comparisons, if performed, are redundant. In case 2, both p_m and p_n belong to the current front, so there does not exist any solution which can dominate p_m or p_n, and the comparison between p_m and p_n should be done to verify whether one dominates the other. In fact, all solutions belonging to the current

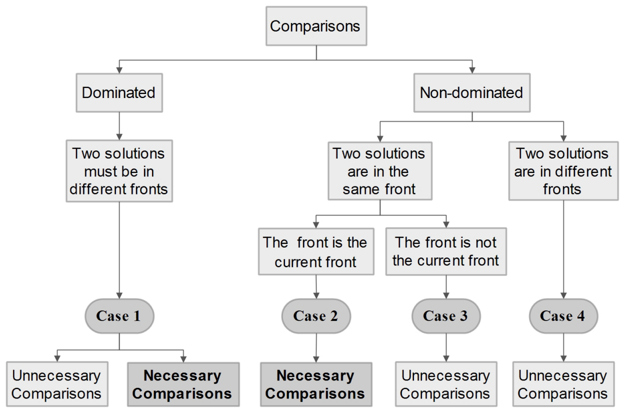

Comparisons
Dominated
Non-dominated
Two solutions must be in different fronts
Two solutions are in the same front
Two solutions are in different fronts
The front is the current front
The front is not the current front
Case 1
Case 2
Case 3
Case 4
Unnecessary Comparisons
Necessary Comparisons
Necessary Comparisons
Unnecessary Comparisons
Unnecessary Comparisons

Fig. 3. A categorization of dominance comparison results between two solutions in non-dominated sorting.

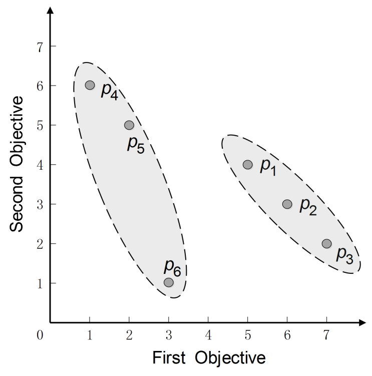


Fig. 4. A population containing six solutions for a bi-objective minimization problem.

front should be compared with each other to ensure that they are all non-dominated with each other. In case 3, neither p_m nor p_n belongs to the current front, which means that there exists at least one solution dominating p_m and a solution dominating p_n, and the comparison between p_m and p_n can be skipped. In case 4, since there exists at least one solution dominating p_m or p_n, a comparison between p_m and p_n is unnecessary. The four cases of possible comparisons are illustrated in Fig. 3.

As shown in Fig. 3, for non-dominated sorting, comparisons in case 1 and comparisons in case 2 cannot be avoided, which are termed necessary comparisons. The necessary comparisons in case 1 refer to the comparisons between solutions in different fronts, while the necessary comparisons in case 2 refer to comparisons between solutions in the same front. The number of necessary comparisons in case 1 and case 2 is the theoretical minimum number of needed dominance comparisons for any non-dominated sorting algorithm. If a non-dominated sorting approach determines the front of a solution by starting to check from the first front to the last one, e.g., from F_1 to F_4 in Fig. 1, the number of necessary comparisons in case 1 can be calculated in the following way. Given a population consisting of N solutions that can be divided into K fronts, assume front F_i contains N_i solutions, where $1 \leq i \leq K$. So, we have $N_1 + N_2 + \ldots + N_K = N$. If a solution p_n belongs to front F_i, then at least one solution dominating p_n will be found in each of the preceding $i-1$ fronts. This means that $i-1$ comparisons in case 1 are needed for solution p_n. Since there are N_i solutions belonging to F_i, a total of $(i-1)N_i$ comparisons in case 1 are needed for front F_i. Therefore, the total number of necessary dominance comparisons between solutions in different fronts is:

$$Num\_Comp_1 = \sum_{i=1}^{K}(i-1)N_i \tag{1}$$

Regarding the minimum number of needed comparisons in case 2, since each of the N_i solutions in front F_i should be compared with the other solutions in front F_i, which needs a total of $N_i(N_i-1)/2$ comparisons, the total number of comparisons between solutions in the same front is:

$$Num\_Comp_2 = \sum_{i=1}^{K}\frac{N_i(N_i-1)}{2} \tag{2}$$

Unfortunately, most popular non-dominated sorting approaches have a total number of dominance comparisons much higher than the sum of Num_Comp_1 and Num_Comp_2. From the above discussions, we can find that further improvements of the computational efficiencies of non-dominated sorting approaches should concentrate on the reduction of unnecessary comparisons in cases 1, 3 and 4. In fact, most existing improved non-dominated sorting approaches have successfully removed unnecessary dominance comparisons in case 1, however, still perform many unnecessary comparisons belonging to cases 3 and 4.

As an example, we consider the comparisons performed in deductive sort over a population shown in Fig. 4. This population contains six candidate solutions of a bi-objective minimization problem, each being denoted by $p_i(f_1, f_2), i = 1, 2, \ldots, 6$, where f_1 and f_2 are the values of the two objectives of solution p_i, respectively. In this example, the six candidate solutions are $p_1(5,4), p_2(6,3), p_3(7,2), p_4(1,6), p_5(2,5), p_6(3,1)$.

As shown in Fig. 4, there are two fronts in this population, where solutions p_4, p_5 and p_6 belong to the first front F_1, and solutions p_1, p_2 and p_3 belong to F_2. Deductive sort performs the following comparisons to assign the solutions to one of the two fronts. It begins with comparing solution p_1 with all other solutions in the population one by one. Solution p_1 is first compared with solution p_2. Since p_1 is not dominated by p_2, deductive sort continues to perform the comparison between p_1 and p_3. The comparison result indicates that solution p_1 is not dominated by p_3 either. Similarly, p_1 will be further compared with p_4 and p_5, and it is concluded that p_1 is dominated neither by p_4 nor by p_5. Then, p_1 is compared with p_6, and it will be found that p_1 is dominated by p_6, which means that p_1 does not belong to the current

TABLE I
COMPARISONS PERFORMED BY DEDUCTIVE SORT FOR THE POPULATION SHOWN IN FIG. 4.

Front	Comparison	Operation	Comparison Result
F_1	(p_1, p_2)		Case 3
	(p_1, p_3)		Case 3
	(p_1, p_4)		Case 4
	(p_1, p_5)		Case 4
	(p_1, p_6)	p_1 is ignored	Case 1
	(p_2, p_3)		Case 3
	(p_2, p_4)		Case 4
	(p_2, p_5)		Case 4
	(p_2, p_6)	p_2 is ignored	Case 1
	(p_3, p_4)		Case 4
	(p_3, p_5)		Case 4
	(p_3, p_6)	p_3 is ignored	Case 1
	(p_4, p_5)		Case 2
	(p_4, p_6)	p_4 is assigned into F_1	Case 2
	(p_5, p_6)	p_5 and p_6 are assigned into F_1	Case 2
F_2	(p_1, p_2)		Case 2
	(p_1, p_3)	p_1 is assigned into F_2	Case 2
	(p_2, p_3)	p_2 and p_3 are assigned into F_2	Case 2
Case 1: 3	Case 2: 6	Case 3: 3	Case 4: 6
	Total: 18		

front F_1. By then, solution p_1 will no longer be involved in further comparisons of front F_1 in deductive sort.

In the above procedure, the following dominance comparisons have been made: two comparisons of case 3 (comparisons between p_1 and p_2, or p_3), two comparisons of case 4 (comparisons between p_1 and p_4, or p_5), and one comparison of case 1 (a comparison between p_1 and p_6). After p_1 is compared with all the other solutions, deductive sort starts to consider solution p_2. Dominance comparison will continue until all solutions are assigned to a front. Table I lists all comparisons performed by deductive sort for the population shown in Fig. 4.

From Table I, it is not difficult to see that there exist many unnecessary comparisons performed by deductive sort, which belong to cases 3 and 4. Among these unnecessary comparisons, several of them are duplicate comparisons, such as the comparisons between p_1 and p_2. In fact, all duplicate comparisons in deductive sort belong to case 3. In this work, we propose a new non-dominated sorting algorithm using a strategy different from the one illustrated in Fig. 2, which aims to avoid duplicate comparisons, thereby considerably reducing the number of unnecessary comparisons.

III. AN EFFICIENT NON-DOMINATED SORT FRAMEWORK

Here, we present a new efficient non-dominated sorting strategy, termed ENS, which is conceptually different from most existing non-dominated sorting methods. The main idea of the ENS approach is shown in Fig. 5. By comparing Figs. 2 and 5, we can see that the ENS approach determines the front each solution belongs to one by one, while most existing non-dominated sorting approaches determine the front of all solutions on the same front as a whole. The main merit of determining the front to which each solution belongs separately is that it can avoid duplicate comparisons, since in this approach, a solution to be assigned only needs to be

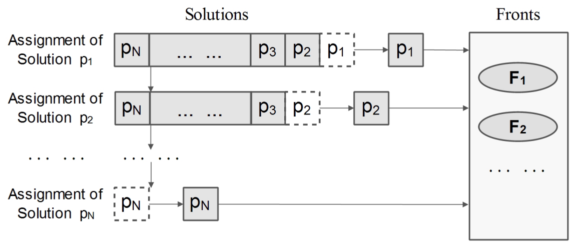

Solutions
Fronts
Assignment of Solution p1
pN
… …
p3
p2
p1
p1
F1
Assignment of Solution p2
pN
… …
p3
p2
p2
F2
… …
… …
… …
Assignment of Solution pN
pN
pN

Fig. 5. An illustration of the proposed dominance comparison strategy, where solutions in the population can be assigned to the fronts one by one.

Algorithm 1 The main steps of ENS for non-dominated sorting.

```
Input: population $P$
Output: the set of fronts $F$
1: $F = empty$;
2: sort $P$ in an ascending order of the first objective value;
3: for all $P[n] \in$ sorted $P$ do
4:     assign solution $P[n]$ into $F$ by Algorithm 2 or Algorithm 3;
5: end for
6: return $F$;
```

compared with solutions that have already been assigned to a front.

The details of ENS are given in Algorithm 1. For a minimization problem, this approach first sorts the N solutions in population P in an ascending order according to the first objective value, where N is the population size. If the first objective values of two solutions are the same, then they are sorted according to the second objective value. This procedure continues until all individuals in the population are sorted. If solutions have the same value in all objectives, their order can be arbitrary. For this sorted population P, a solution p_m will never be dominated by a solution p_n, if $m < n$, since there exists at least one objective in p_m whose value is smaller than that of the same objective in p_n. This means that there exist only two possible relationships between the two solutions: either p_m dominates p_n, or p_m and p_n are not comparable.

After finishing sorting the individuals in population P, ENS begins to assign solutions to fronts in the sorted population P one by one, starting from the first solution p_1 and ending with the last one p_N. As we know, if a solution is assigned to a front, it is dominated by at least one solution in the preceding front. As pointed out above, a solution can never be dominated by any succeeding solution in the sorted population P. Therefore, it is sufficient to compare a solution with those that have already been assigned to a front to determine the front of this solution. The possible relationships between a solution to be assigned and those that have been assigned to a front are shown in Fig. 6. Actually, if a solution p_n is assigned to front F_i, F_i must satisfy the following two conditions:

1) There exists at least one solution in each front F_j that has been assigned and dominates p_n, for $1 \leq j \leq i-1$;
2) There exists no solution in any of the assigned fronts F_k that dominates p_n, for $k \geq i$.

In this way, the front to which a solution belongs can be determined by finding out the front which satisfies the above two conditions. In what follows, we present two strategies for searching for the front satisfying the above two conditions within the ENS frmework, one using a sequential search strategy (termed ENS-SS), and the other using a binary search strategy (ENS-BS).

A. A Sequential Search Strategy

The pseudocode of the sequential search is presented in Algorithm 2. The idea in this search strategy is quite straightforward. For solution p_n, the algorithm checks at first whether there exists a solution that has been assigned to the first front F_1 and dominates p_n. If such a solution does not exist, assign p_n to front F_1. If p_n is dominated by any solution in F_1, start comparing p_n with the solutions assigned to F_2. If no solution in front F_2 dominates p_n, assign p_n to front F_2. If p_n is not assigned to any of the existing fronts, create a new front and assign p_n to this new front.

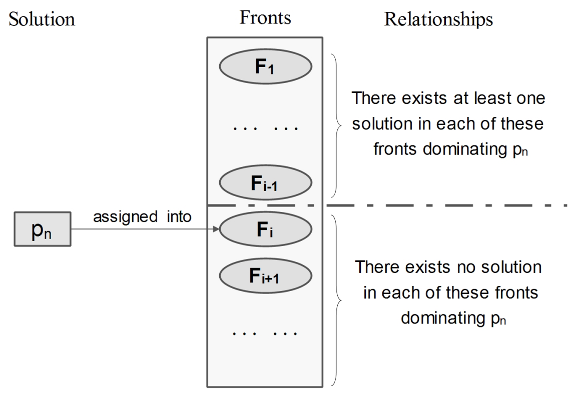

Solution
Fronts
Relationships
F1
... ...
Fi-1
There exists at least one solution in each of these fronts dominating pn
pn
assigned into
Fi
Fi+1
... ...
There exists no solution in each of these fronts dominating pn

Fig. 6. The relationships between p_n and the solutions having been assigned to a front.

Algorithm 2 The Sequential Search Strategy for Finding the Front of a Solution

```
Input: solution $P[n]$, the set of fronts $F$
Output: the front number of solution $P[n]$
1: $x$ = size($F$); {the number of fronts having been found}
2: $k = 1$; {the front now checked}
3: while true do
4:   compare $P[n]$ with the solutions in $F[k]$ starting from the last one and ending with the first one;
5:   if $F[k]$ contains no solution dominating $P[n]$ then
6:     return $k$; {move $P[n]$ to $F[k]$}
7:     break;
8:   else
9:     $k++$;
10:    if $k > x$ then
11:      return $x + 1$; {move $P[n]$ to a new front}
12:      break;
13:    end if
14:  end if
15: end while
```

There is a little trick in checking whether a front has a solution dominating p_n. Recall that solutions assigned to an existing front are also sorted in the same order as the population. Therefore, the comparisons between p_n and the solutions assigned to the front should start with the last one in the front and ends with the first one. This trick often leads to fewer comparisons if a solution assigned to this front dominates p_n, since solutions in the end of the sorted front are more likely to dominate p_n. As a result, unnecessary comparisons can be avoided.

For bi-objective optimization problems, the idea presented here is computationally more efficient than existing non-dominated sorting methods. In fact, as shown in Algorithm 2, only one comparison is sufficient for determining whether a solution to be assigned belongs to an existing front. The reason is as follows. In the sorting,

TABLE II
COMPARISONS PERFORMED BY ENS-SS FOR THE POPULATION SHOWN IN FIG. 4

Assigned Solution	Comparison	Operation	Comparison Result
p_4		p_4 is assigned into F_1	
p_5	(p_5, p_4)	p_5 is assigned into F_1	Case 2
p_6	(p_6, p_5)		Case 2
	(p_6, p_4)	p_6 is assigned into F_1	Case 2
p_1	(p_1, p_6)	p_1 is assigned into F_2	Case 1
p_2	(p_2, p_6)		Case 1
	(p_2, p_1)	p_2 is assigned into F_2	Case 2
p_3	(p_3, p_6)		Case 1
	(p_3, p_2)		Case 2
	(p_3, p_1)	p_3 is assigned into F_2	Case 2
Case 1: 3	Case 2: 6	Case 3: 0	Case 4: 0
Total: 9			

solutions assigned to a front are sorted in the ascending order of the first objective, which means that the second objectives of these solutions are in the descending order, since these solutions are non-dominated with each other. This means that, if p_n, a solution to be assigned, is dominated by a solution in an existing front, it should be dominated by the last solution in the front, since the last solution has the smallest value in the second objective among all solutions in the front. Therefore, for bi-objective optimization problems, this method can determine the front number of a solution by performing only the comparisons between this solution and the last solution in each front.

In Table II, we list the comparisons performed by ENS using the sequential search strategy (ENS-SS) for non-dominated sorting of the population given in Fig. 4. As shown in the table, ENS-SS needs only nine comparisons in total, which is much smaller than the number of comparisons needed by deductive sort, refer to Table I. It is not difficult for the reader to check that there does not exist any comparison belonging to case 3 in ENS-SS. This means that ENS-SS does not perform any duplicate comparisons, which can be attributed to the ENS strategy shown in Fig. 5. It should be noted that, although ENS-SS does not perform any comparison belonging to case 4 for the population shown in Fig. 4, such comparisons may occur for other populations.

B. A Binary Search Strategy

The pseudocode of the binary search strategy is presented in Algorithm 3. Different from sequential search, the binary search strategy starts with checking the intermediate front $F_{\lfloor L/2 \rfloor}$ instead of the first front F_1, where L is the number of fronts which have been created thus far, i.e., before solution p_n is assigned. If solution p_n is not dominated by any solution in front $F_{\lfloor L/2 \rfloor}$, then solution p_n will be compared with the solutions in front $F_{\lfloor L/4 \rfloor}$. Otherwise, p_n is compared with the solutions in front $F_{\lfloor 3L/4 \rfloor}$. In this way, the binary search can determine the front to which solution p_n belongs after checking $\lceil \log(L+1) \rceil$ fronts. If the last existing front F_L has been checked and p_n doses not belong to this front, a new front F_{L+1} will be created and solution p_n is assigned to this new front.

Algorithm 3 The Binary Search Strategy for Finding the Front of a Solution

```
Input: solution $P[n]$, the set of fronts $F$
Output: the front number of solution $P[n]$
1: $x$ = size($F$); {the number of fronts having been found}
2: $kmin = 0$; {the lower bound for checking}
3: $kmax = x$; {the upper bound for checking}
4: $k = \lfloor (kmax + kmin)/2 + 1/2 \rfloor$; {the front now checked}
5: while true do
6:   compare $P[n]$ with the solutions in $F[k]$ starting from the last one and ending with the first one;
7:   if $F[k]$ has no solution dominating $P[n]$ then
8:     if $k == kmin + 1$ then
9:       return $k$; {move $P[n]$ to $F[k]$}
10:      break;
11:    else
12:      $kmax = k$;
13:      $k = \lfloor (kmax + kmin)/2 + 1/2 \rfloor$;
14:    end if
15:  else
16:    $kmin = k$;
17:    if ($kmax == kmin + 1$) and ($kmax < x$) then
18:      return $kmax$; {move $P[n]$ to $F[kmax]$}
19:      break;
20:    else if $kmin == x$ then
21:      return $x + 1$; {move $P[n]$ to a new front}
22:      break;
23:    else
24:      $k = \lfloor (kmax + kmin)/2 + 1/2 \rfloor$;
25:    end if
26:  end if
27: end while
```

The binary search strategy adopted here usually outperforms the sequential search strategy in that it requires to check fewer fronts in a population. But, this does not mean that the binary search strategy can always perform fewer comparisons than the sequential search strategy in front assignment. In binary search, it can happen that more than one front that does not have any solution dominating p_n needs to be checked. All solutions in these checked fronts have to be compared with p_n. In sequential search, at most one front containing no solution dominating p_n needs to be checked. Therefore, the binary search strategy may not always outperform the sequential search strategy in terms of the number of comparisons, especially for the populations consisting of a small number of fronts.

The trick in comparing solutions in the same front used in the sequential search strategy can also be adopted here. This means that the comparison starts with the last solution assigned to the front to be checked.

Since the population shown in Fig. 4 contains only two fronts, the binary search strategy will perform the same number of comparisons as the sequential search strategy shown in Table II, which should be straightforward for the reader to verify.

C. Analysis of Computational Complexity

In this section, we briefly analyze the computational complexities of the proposed two ENS-based algorithms. In what follows, we assume that non-dominated sorting is to be performed on a population containing N solutions with M objectives.

As described in the previous section, the ENS approach consists of two main steps. First, sort the population in an ascending order according to the objectives. Second, assign solutions in the sorted population to fronts. We can adopt the heapsort [21] in the first step, which is of a time complexity of $O(N \log N)$ and a space complexity of $O(1)$.

If the sequential search is adopted to assign solutions in the population to a front, the second step has a time complexity of $O(MN^2)$ in the worst case, when all N individuals in the population are non-dominated with each other. Consequently, all solutions in the population will be assigned to one front. As indicated by Equation (1), when all solutions are non-dominated, the number of comparisons between solutions in different fronts in the sequential search strategy will be

$$Num\_Comp_1 = \sum_{i=1}^{1}(i-1)N = 0.$$

For the N solutions in the same front, the number of comparisons needed in the sequential search strategy, as shown in Equation (2), is

$$Num\_Comp_2 = \sum_{i=1}^{1} \frac{N(N-1)}{2} = \frac{N(N-1)}{2}.$$

The sequential search strategy will not perform any unnecessary comparison in the worst case, since each pair of two solutions in the population has been compared. Thus, the total number of comparisons performed by the sequential search strategy is

$$\begin{aligned} Num\_Comp^s_{worst} &= Num\_Comp_1 + Num\_Comp_2 \\ &= \frac{N(N-1)}{2}. \end{aligned}$$

Since each solution has M objectives, the sequential search strategy has a worst case time complexity as follows

$$\begin{aligned} T^s_{worst}(N) &= M \cdot Num\_Comp^s_{worst} \\ &= M \cdot \frac{N(N-1)}{2} = O(MN^2). \end{aligned}$$

In the best case, ENS-SS has a time complexity of $O(MN\sqrt{N})$. This happens when the N solutions belong to $\lceil\sqrt{N}\rceil$ fronts, each of which roughly has $\lceil\sqrt{N}\rceil$ solutions, and each solution in a front is dominated by all solutions in the preceding front. According to Equation (1), the number of comparisons between solutions in different fronts in ENS-SS will be

$$Num\_Comp_1 = \sum_{i=1}^{\lceil\sqrt{N}\rceil} (i-1)\lceil\sqrt{N}\rceil = \frac{\lceil\sqrt{N}\rceil^2(\lceil\sqrt{N}\rceil - 1)}{2}.$$

In each of the $\lceil\sqrt{N}\rceil$ fronts, any two solutions of the $\lceil\sqrt{N}\rceil$ solutions in the same front need to be compared. As indicated by Equation (2), the number of comparisons between solutions in the same front needed by ENS-SS is

$$\begin{aligned} Num\_Comp_2 &= \sum_{i=1}^{\lceil\sqrt{N}\rceil} \frac{\lceil\sqrt{N}\rceil(\lceil\sqrt{N}\rceil - 1)}{2} \\ &= \frac{\lceil\sqrt{N}\rceil^2(\lceil\sqrt{N}\rceil - 1)}{2}. \end{aligned}$$

Since each solution in a front is dominated by all solutions in the preceding front, the sequential search strategy will not perform any unnecessary comparison in the best case. Therefore, in the best case, the total number of comparisons performed by ENS-SS for non-dominated sorting will amount to

$$\begin{aligned} Num\_Comp^s_{best} &= Num\_Comp_1 + Num\_Comp_2 \\ &= \lceil\sqrt{N}\rceil^2(\lceil\sqrt{N}\rceil - 1). \end{aligned}$$

Thus, in the best case, the time complexity of ENS-SS is

$$\begin{aligned} T^s_{best}(N) &= M \cdot Num\_Comp^s_{best} \\ &= M \cdot \lceil\sqrt{N}\rceil^2(\lceil\sqrt{N}\rceil - 1) = O(MN\sqrt{N}). \end{aligned}$$

The worst case time complexity of ENS-BS occurs in the same situation as the worst case of ENS-SS, in which both ENS-SS and ENS-BS need to perform the same number of comparisons for a given population. So, the worst case time complexity of ENS-BS is

$$T^b_{worst}(N) = T^s_{worst}(N) = O(MN^2).$$

The best case of ENS-BS occurs when the N solutions in the population belong to N different fronts. In this case, no solution in any of the N fronts needs to be compared with the solution to be assigned. By Equation (2), we can see that the number of comparisons between solutions in the same front in ENS-BS is

$$Num\_Comp_2 = \sum_{i=1}^{N} \frac{1 \cdot (1-1)}{2} = 0.$$

In ENS-BS, if a solution is to be assigned to a front, this solution may need to be compared with the solutions in many existing fronts, but not all. This means that the number of comparisons between the solution to be assigned and the solutions in existing fronts cannot be calculated simply using Equation (1) if the binary search

TABLE III
TIME AND SPACE COMPLEXITIES OF FIVE NON-DOMINATED SORTING METHODS

Approach	Space Complexity	Time Complexity	
		Best Case	Worst Case
ENS-BS	$O(1)$	$O(MN \log N)$	$O(MN^2)$
ENS-SS	$O(1)$	$O(MN\sqrt{N})$	$O(MN^2)$
Deductive Sort	$O(N)$	$O(MN\sqrt{N})$	$O(MN^2)$
Arena's Principle	$O(N)$	$O(MN\sqrt{N})$	$O(MN^2)$
Fast Non-dominated Sort	$O(N^2)$	$O(MN^2)$	$O(MN^2)$

strategy is used. In the best case, each front contains only one solution, so a solution p_n, $1 \leq n \leq N$, should be assigned to front F_n. To achieve this, $\lceil \log n \rceil$ fronts need to be checked and $\lceil \log n \rceil$ comparisons must be performed. The number of comparisons between solutions in different fronts using the binary search strategy is

$$Num\_Comp_1 = \sum_{i=1}^{N} \lceil \log i \rceil.$$

The binary search strategy will not perform any unnecessary comparison in this case, since each front only contains one solution. Consequently, the total number of comparisons performed by ENS-BS will be

$$\begin{aligned} Num\_Comp^b_{best} &= Num\_Comp_1 + Num\_Comp_2 \\ &\approx \lceil \log(N!) \rceil, \end{aligned}$$

and its best case time complexity is

$$\begin{aligned} T^b_{best}(N) &= M \cdot Num\_Comp^b_{best} \\ &\approx M \lceil \log(N!) \rceil = O(MN \log N). \end{aligned}$$

To summarize, the ENS-SS has the worst case time complexity of

$$O(N \log N) + T^s_{worst}(N) = O(MN^2),$$

and the best case time complexity of

$$O(N \log N) + T^s_{best}(N) = O(MN\sqrt{N}).$$

By contrast, ENS-BS has the worst case time complexity of

$$O(N \log N) + T^b_{worst}(N) = O(MN^2),$$

and the best case time complexity of

$$O(N \log N) + T^b_{best}(N) = O(MN \log N).$$

We can see from Algorithms 1, 2 and 3 that ENS and the two search strategies involve scalar variables only during the non-dominated sorting, except for the given population P and resulting fronts F_i, $1 \leq i \leq K$, where K is the number of fronts of a population. Therefore, ENS has a space complexity of $O(1)$, no matter whether the sequential search strategy or the binary search strategy is employed.

Table III presents the time and space complexities of the ENS-based non-dominated sorting algorithms (ENS-SS and ENS-BS), together with those of three existing popular non-dominated sorting algorithms, namely, deductive sort, arena's principle and fast non-dominated sort. As seen from Table III, the ENS-based algorithms have the same worst case time complexity of $O(MN^2)$ as the three existing non-dominated sorting methods. Note that the best case time complexity and worst case time complexity of the fast non-dominated sort are the same, while the best case time complexities of deductive sort and arena's principle can be reduced to $O(MN\sqrt{N})$. ENS-SS has the same best case time complexity as deductive sort and arena's principle, whereas ENS-BS achieves an improved efficiency on the best case time complexity, which is $O(MN \log N)$. We should stress that, this does not mean that ENS-BS has a better efficiency than ENS-SS on average. On the other hand, both ENS-SS and ENS-BS can achieve a lower space complexity than the three compared existing approaches.

To summarize, the proposed ENS algorithm has an overall lower time and space complexity than existing approaches. In the following, we will verify the theoretical analysis using empirical results.

IV. SIMULATION RESULTS

In this section, we will compare ENS-SS and ENS-BS with three popular non-dominated sorting algorithms, namely, deductive sort, arena's principle and the fast non-dominated sort for two different setups. In the first setup, we use two performance indicators, i.e., the number of dominance comparisons and the amount of runtime to evaluate the compared approaches using synthetic test populations. In the second setup, we compare these non-dominated sorting approaches within the framework of NSGA-II to test the performance of the proposed ENS approach in optimization. All simulations reported in this work are conducted on a PC with a 2.30GHz Intel Core i7-3610QM CPU and the Windows 7 SP1 64 bit operating system.

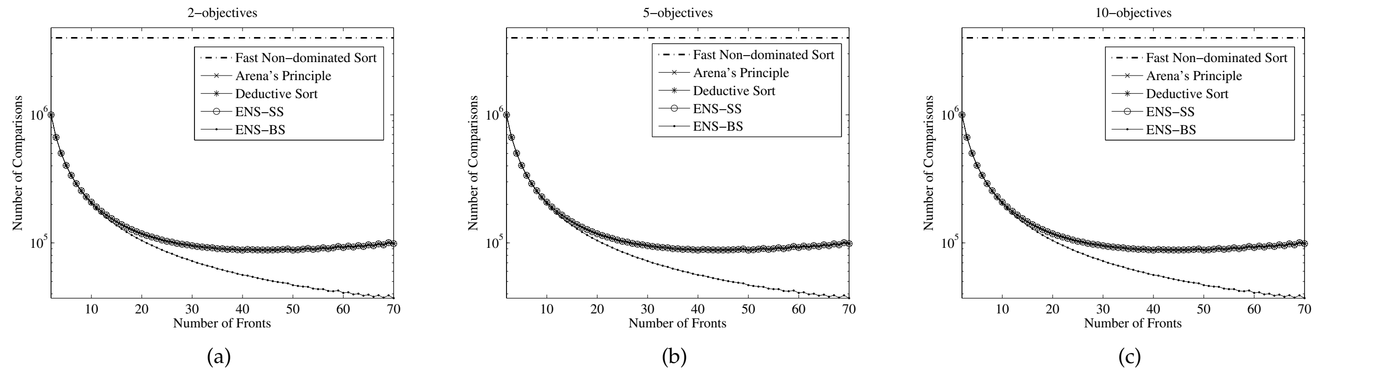


Fig. 7. Numbers of comparisons of five non-dominated sorting approaches for populations with predefined fronts.

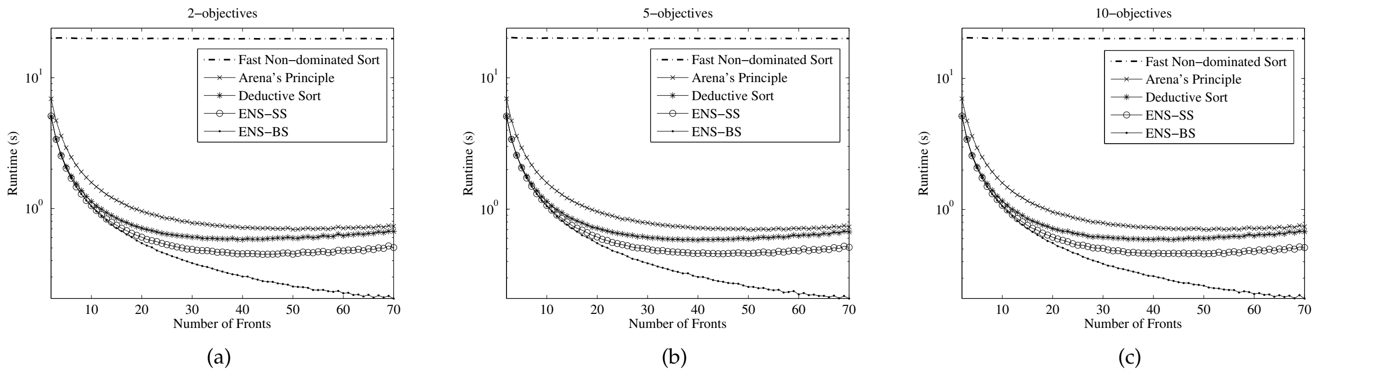


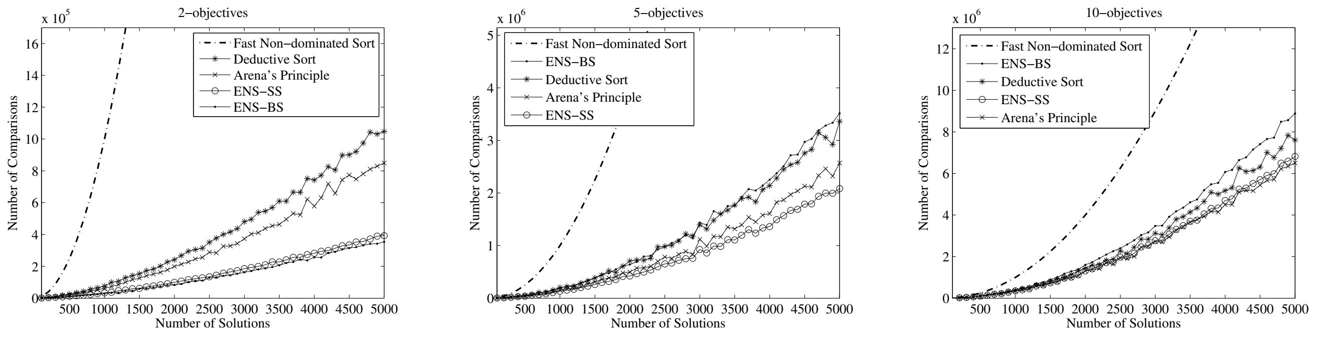
Fig. 8. Runtimes of five non-dominated sorting approaches for populations with predefined fronts.

In the first setup, random populations and populations with predefined fronts [12] are used. For a random population, each objective value of the candidate solutions (individuals) in the population is randomly sampled from a uniform distribution on the interval $[0, 1]$. For populations with predefined fronts, each front contains the same number of solutions, and each solution in a front is dominated by all solutions in the preceding front. Such populations can be generated in the following way: First, generate a random population with N solutions of M objectives, read out all the objective values and sort them in an ascending order for each objective, which results in M sorted vectors of a size N, denoted by $(p_{1,j}, p_{2,j}, ..., p_{N,j})$, where $j = 1, 2, ..., M$. Then, N new solutions can be generated by assigning $p_{i,j}$ to be the j-th objective of the i-individual, thereby creating N solutions of M objectives. The new population is then split into K sets of solutions, where the first $\lfloor N/K \rfloor$ solutions are assigned to the first set, the second $\lfloor N/K \rfloor$ solutions to the second set and the last $N - \lfloor N/K \rfloor \cdot (K-1)$ solutions to the K-th set. Randomly choose some objectives of the solutions in each set and replace these objective values of the i-th solution with the ones of $(S-i)$-th solution, where S is the number of solutions in this set, and $1 \leq i \leq \lfloor S/2 \rfloor$. The population with predefined fronts consists of all solutions in the K sets arranged in a random order.

A. Experiments on Populations with Predefined Fronts

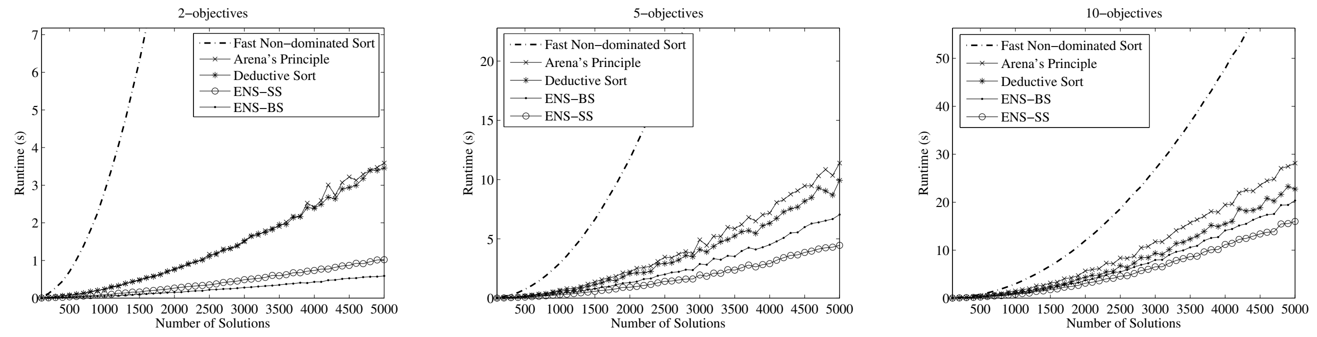
We use three types of populations in the experiments on populations with predefined fronts to evaluate the proposed ENS approach: populations with two objectives, populations with five objectives and populations with ten objectives. For each type of population, the number of fronts a population contains varies from two to 70 with an increment of one, resulting in 69 populations in total. In the experiments, each population contains 2,000 solutions.

Figs. 7 and 8 present the simulation results of ENS-SS and ENS-BS, together with those of deductive sort, arena's principle and fast non-dominated sort on the given populations. From Figs. 7 and 8, it is easy to find that the performance curves of the fast non-dominated sort remains unchanged over different setups. The reason is that the number of comparisons the fast non-dominated sort needs depends only on the number of solutions in the population, which is $N(N-1)$, where N is the number of solutions in the population, regardless of the number of fronts the population contains. The fast non-dominated sort performs much more comparisons than other approaches. This leads to a longer runtime for the fast non-dominated sort than other non-dominated sorting approaches under comparison. Note that the Y-axes in Figs. 7 and 8 are labelled in log-scale, since the number of comparisons and runtime of the fast non-



(a) (b) (c)

Fig. 9. Numbers of comparisons of five non-dominated sorting approaches for random populations.



(a) (b) (c)

Fig. 10. Runtimes of five non-dominated sorting approaches for random populations.

dominated sort are much larger than those of the other sorting methods.

Compared with the fast non-dominated sort, a dramatic decrease in the number of comparisons has been achieved by the deductive sort and arena's principle. ENS-SS needs almost the same number of comparisons as the deductive sort and arena's principle, while ENS-BS requires a slightly smaller number of comparisons than other approaches. Particularly, ENS-BS outperforms other non-dominated sorting approaches in terms of the number of dominance comparisons when the number of fronts becomes larger. As far as the runtime is concerned, both ENS-SS and ENS-BS consistently need less runtime than deductive sort and arena's principle. Since it has a lower space complexity of $O(1)$, the ENS approach consumes less runtime than deductive sort and arena's principle in case they require the same number of comparisons. This can also be seen from Fig. 7: Although ESN-SS performs almost the same number of comparisons as deductive sort and arena's principle on each population, it takes less runtime than others.

Different from the fast non-dominated sort, the number of comparisons needed by the two ENS-based algorithms, deductive sort and arena's principle will decrease as the number of fronts in the population increases. This can be attributed to the fact that the population with predefined fronts is close to the best cases of these approaches as the number of fronts becomes larger. For ENS-SS, deductive sort and arena's principle, the best case occurs when the number of fronts $K = \sqrt{N} = \sqrt{2000} \approx 45$. Therefore, the number of comparisons will increase when the number of fronts is larger than 45, just as shown in Fig. 7. The number of comparisons performed by the ENS-BS always decreases, since the best case of this approach occurs when the number of fronts $K = N = 2000$.

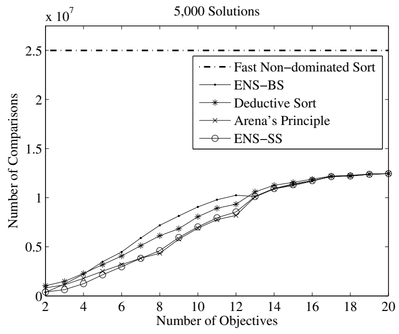
B. *Experiments on Random Populations*

Similar to the experiments on populations with predefined fronts, random populations with two, five and ten objectives are adopted to evaluate computational efficiency of the proposed ENS approach. For each type of population, the population size varies from 100 to 5,000 with an increment of 100. Consequently, 50 random populations with two, five and ten objectives are used in the simulations.

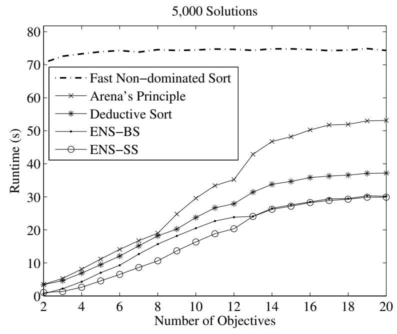
The simulation results from the random populations are shown in Figs. 9 and 10. We can see that the ENS approach clearly outperforms other non-dominated sorting approaches on the random populations for bi-objective optimization problems. Furthermore, ENS-BS is more efficient than ENS-SS on the random populations when the number of objectives is two. For random populations having five objectives, ENS-BS clearly outperforms the fast non-dominated sort, while it underperforms

TABLE IV
THE MEANS AND STANDARD DEVIATIONS OF RUNTIMES AND NUMBERS OF COMPARISONS OVER 50 RANDOM POPULATIONS HAVING 5,000 SOLUTIONS. BEST PERFORMANCE IS SHOWN IN BOLD.

Approach	Objectives	Run Time (s) (Standard Deviation)	Number of Comparisons (Standard Deviation)
ENS-BS	2	**0.60** (0.01)	**358,910** (9,764)
	5	7.06 (0.27)	3,551,188 (121,927)
	10	20.14 (0.45)	8,963,163 (175,492)
ENS-SS	2	1.02 (0.02)	397,642 (5,649)
	5	**4.52** (0.15)	**2,113,312** (67,034)
	10	**15.93** (0.32)	6,875,560 (131,477)
Deductive Sort	2	3.54 (0.06)	1,069,973 (23,245)
	5	9.82 (0.29)	3,278,099 (103,396)
	10	23.31 (0.74)	7,833,565 (264,666)
Arena's Principle	2	3.66 (0.10)	854,582 (25,183)
	5	11.46 (0.41)	2,530,377 (99,463)
	10	28.93 (1.01)	**6,562,012** (208,166)
Fast Non-dominated Sort	2	70.56 (0.13)	24,995,000 (0)
	5	74.21 (0.16)	24,995,000 (0)
	10	74.69 (0.22)	24,995,000 (0)



(a)



(b)

Fig. 11. Numbers of comparisons and runtimes of five non-dominated sorting approaches for random populations with different objectives.

deductive sort and arena's principle in terms of the number of needed comparisons. The ENS-SS requires a smaller number of comparisons than deductive sort and arena's principle for random populations having five objectives. However, both ENS-SS and ENS-BS are more efficient than deductive sort and arena's principle in terms of runtime. Similar conclusions can be made for the random populations having ten objectives.

To further demonstrate the computational efficiency of the ENS-based algorithms, we consider populations having two, five and ten objectives, where for each category of populations 50 different random populations of a size of 5,000 are used. Table IV shows the mean and standard deviation of runtimes and of numbers of comparisons the different approaches need to perform over 50 random populations. It can be seen that both ENS-SS and ENS-BS outperform the fast non-dominated sort, deductive sort and arena's principle in dealing with populations having two and five objectives. For populations having ten objectives, the ENS-based algorithms can still work more efficiently than other three popular non-dominated sorting approaches, although the superiority in performance becomes less significant.

As shown in Figs. 9, 10 and Table IV, the computational efficiency of the proposed ENS-based algorithms is affected by the number of objectives for random populations. To further investigate the influences of the number of objectives, we conducted additional experiments on the ENS-based algorithms for random populations having two to 20 objectives, respectively, where the population size is set to 5,000. The experimental results of the ENS-SS, ENS-BS, fast non-dominated sort, deductive sort and arena's principle on these populations are shown in Fig. 11. We can observe that ENS, deductive sort and arena's principle are expected to perform more comparisons when the number of objectives increases. By

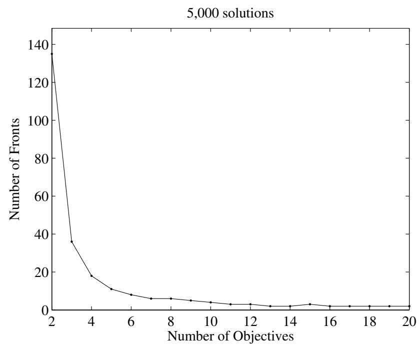


Fig. 12. The relationships between the number of objectives and the number of fronts for the populations used in Fig. 11.

contrast, the number of comparisons needed by fast non-dominated sort remains constant. The increasing number of required comparisons in the ENS-based algorithms, deductive sort and arena's principle is caused by a decrease in the number of fronts in the population, which can also be seen from Fig. 7 in Section IV-A. Actually, for a population with a given size, the number of fronts decreases dramatically with the increase in the number of objectives. Fig. 12 illustrates the relationship between the number of objectives and the number of fronts for the populations used in the experiment of Fig. 11.

As shown in Fig. 11, the runtimes of ENS-SS, ENS-BS, deductive sort and arena's principle will increase as the number of objectives becomes larger. By contrast, the runtime of fast non-dominated sort slightly increases with an increase in the number of objectives. The reason for this is that the number of comparisons between objectives of two solutions will increase when the number of objectives increases, even if the number of dominance comparisons remains the same. As a result, the runtime consumed by one comparison between two solutions will increase as the number of objectives increase. It can be found that both ENS-SS and ENS-BS are more efficient than the fast non-dominated sort, deductive sort and arena's principle in terms of runtimes, as the number of objectives increases.

C. Experiments on the Sorting Algorithms Embeded in NSGA-II

To assess the performance of the ENS-based algorithms in evolutionary multi-objective optimization, we compare them with fast non-dominated sort, deductive sort and arena's principle when these methods are embeded in NSGA-II. This means that the compared optimization algorithms are completely the same except for the non-dominated sorting method used. The benchmark problem DTLZ1 [22] having two, five and ten objectives are used to test the performance. In the experiments, the maximum number of generations is set to 250, other parameters are specified as recommended in [2]. Note that the deductive sort, arena's principle and fast non-dominated sort do not necessarily need to sort the entire population when they are applied to an MOEA. By contrast, the ENS-based sorting algorithms do need to sort the entire population. This has been taken into account in the comparative experiments here.

Tables V and VI present the mean and standard deviation of runtimes of the NSGA-II using ENS-SS, ENS-BS, fast non-dominated sort, deductive sort and arena's principle for non-dominated sorting, respectively on DTLZ1. The computational efficiency is studied for a population size of 200 and 800, respectively, averaged over ten independent runs. From Tables V and VI, we find that both ENS-SS and ENS-BS outperform fast non-dominated sort, deductive sort and arena's principle in the evolutionary optimization. We can also find that ENS-SS is more efficient than ENS-BS in dealing with DTLZ1 of five and ten objectives, while the latter can work more efficiently than the former on DTLZ1 of two objectives. Note that, the runtimes of the NSGA-II using the five compared non-dominated sorting approaches all increase as the number of objectives increases, however, the increase in runtimes of the NSGA-II using the fast non-dominated sort is relatively slow. These results are consistent with those shown in Fig. 11.

V. Conclusions and Remarks

In this paper, a novel non-dominated sorting approach, called ENS, is proposed, which adopts a sorting strategy that is different from those in the existing non-dominated sorting approaches. In ENS, each pair of solutions will be compared at most once, thereby avoiding many unnecessary comparisons. ENS has a time complexity of $O(MN \log N)$ in the best case, and a space complexity of $O(1)$. Comparative experimental results confirm that the proposed ENS approach outperforms three popular non-dominated sorting approaches, namely, the fast non-dominated sort, deductive sort and arena's principle, most significantly for optimization problems having a small number of objectives.

Two non-dominated sorting algorithms based on the ENS approach have been implemented, one using the sequential search strategy and the other using the binary search strategy to locate the front to which a solution belongs. The two ENS-based non-dominated sorting algorithms have been shown to be efficient for dealing with populations having a relatively small number of objectives. However, the efficiency of the two sorting algorithms will decrease as the number of objectives increases. Therefore, an important topic for further research is to find a new search strategy for the ENS approach that can work more efficiently in dealing with a larger number of objectives.

References

[1] D. W. Corne, N. R. Jerram, J. D. Knowles, and M. J. Oates, "PESA-II: region-based selection in evolutionary multi-objective optimization," in *2001 Genetic and Evolutionary Computation Conference*, 2001, pp. 283–290.

TABLE V
THE MEAN AND STANDARD DEVIATION OF RUNTIMES OF THE NSGA-II FRAMEWORK WITH FIVE NON-DOMINATED SORTING APPROACHES ON DTLZ1 WITH A POPULATION SIZE OF 200, AVERAGED OVER 10 RUNS. BEST PERFORMANCE IS SHOWN IN BOLD.

Approach	Runtime (s) (Standard Deviation)		
	2-objectives	5-objectives	10-objectives
ENS-BS	**6.28** (0.95)	28.94 (1.12)	45.36 (1.46)
ENS-SS	6.36 (0.63)	**24.49** (1.10)	**42.35** (1.68)
Deductive Sort	12.35 (1.14)	32.76 (2.73)	52.88 (1.63)
Arena's Principle	17.16 (1.76)	47.39 (4.05)	70.03 (2.21)
Fast Non-dominated Sort	102.92 (1.16)	110.81 (1.09)	124.15 (1.57)

TABLE VI
THE MEAN AND STANDARD DEVIATION OF RUNTIMES OF THE NSGA-II FRAMEWORK WITH FIVE NON-DOMINATED SORTING APPROACHES ON DTLZ1 WITH A POPULATION SIZE OF 800, AVERAGED OVER 10 RUNS. BEST PERFORMANCE IS SHOWN IN BOLD.

Approach	Runtime (s) (Standard Deviation)		
	2-objectives	5-objectives	10-objectives
ENS-BS	**34.06** (0.44)	258.20 (11.58)	627.21 (13.16)
ENS-SS	43.48 (0.63)	**192.21** (13.01)	**499.30** (14.16)
Deductive Sort	127.91 (4.55)	234.17 (10.13)	649.16 (15.82)
Arena's Principle	186.09 (6.42)	337.47 (14.29)	797.13 (19.71)
Fast Non-dominated Sort	1674.62 (5.08)	1843.45 (6.38)	1858.24 (19.01)

[2] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multi-objective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[3] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: improving the strength pareto evolutionary algorithm for multiobjective optimization," in *Fifth Conference on Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*, 2001, pp. 95–100.

[4] J. D. Knowles and D. W. Corne, "M-PAES: a memetic algorithm for multiobjective optimization," in *2000 IEEE Congress on Evolutionary Computation*, 2000, pp. 325–332.

[5] A. Berry and P. Vamplew, "The combative accretion model c-multiobjective optimisation without explicit pareto ranking," in *Third International Conference on Evolutionary Multi-Criterion Optimization*, 2005, pp. 77–91.

[6] J. E. Fieldsend, R. M. Everson, and S. Singh, "Using unconstrained elite archives for multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 3, pp. 305–323, 2003.

[7] A. Berry and P. Vamplew, "An efficient approach to unbounded bi-objective archives – introducing the mak_tree algorithm," in *8th Conference on Genetic and Evolutionary Computation*, 2006, pp. 619–626.

[8] D. E. Goldberg, *Geneticdd Algorithms in Search, Optimization, and Machine Learning*. Boston, USA: Addison-Wesley, 1989.

[9] N. Srinivas and K. Deb, "Multiobjective optimization using non-dominated sorting in genetic algorithms," *Evolutionary Computation*, vol. 2, no. 3, pp. 221–248, 1995.

[10] M. T. Jensen, "Reducing the run-time complexity of multiobjective eas: the NSGA-II and other algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 5, pp. 503–515, 2003.

[11] S. Tang, Z. Cai, and J. Zheng, "A fast method of constructing the non-dominated set: arena's principle," in *Fourth International Conference on Natural Computation*, 2008, pp. 391–395.

[12] K. M. Clymont and E. Keedwell, "Deductive sort and climbing sort: new methods for non-dominated sorting," *Evolutionary Computation*, vol. 20, no. 1, pp. 1–26, 2012.

[13] H. Fang, Q. Wang, Y. Tu, and M. F. Horstemeyer, "An efficient non-dominated sorting method for evolutionary algorithms," *Evolutionary Computation*, vol. 16, no. 3, pp. 355–384, 2008.

[14] K. Deb, "An efficient constraint handling method for genetic algorithms," *Computer Methods in Applied Mechanics and Engineering*, vol. 186, no. 2-4, pp. 311–338, 2000.

[15] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler, "Combining convergence and diversity in evolutionary multiobjective optimization," *Evolutionary Computation*, vol. 10, no. 3, pp. 263–282, 2002.

[16] K. Deb and S. Tiwari, "Omni-optimizer: a procedure for single and multi-objective optimization," in *Third International Conference on Evolutionary Multi-Criterion Optimization*, 2005, pp. 47–61.

[17] C. Shi, M. Chen, and Z. Shi, "A fast nondominated sorting algorithm," in *2005 International Conference on Neural Networks and Brain*, 2005, pp. 1605–1610.

[18] X. Zhou, J. Shen, and J. Shen, "An immune recognition based algorithm for finding non-dominated set in multi-objective optimization," in *2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, 2008, pp. 305–310.

[19] J. Zheng, C. X. Ling, Z. Shi, and Y. Xie, "Some discussions about MOGAs: individual relations, non-dominated set, and application on automatic negotiation," in *2004 IEEE Congress on Evolutionary Computation*, 2004, pp. 706–712.

[20] J. Du, Z. Cai, and Y. Chen, "A sorting based algorithm for finding non-dominated set in multi-objective optimization," in

Third International Conference on Natural Computation, 2007, pp. 436–440.

[21] J. W. J. Williams, “Algorithm 232 – heapsort,” *Communications of the ACM*, vol. 7, no. 6, pp. 347–348, 1964.

[22] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, “Scalable multi-objective optimization test problems,” in *2002 IEEE Congress on Evolutionary Computation*, 2002, pp. 825–830.