Journal of Computational Science 23 (2017) 31–43

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

A novel non-dominated sorting algorithm for evolutionary multi-objective optimization

Chunteng Bao[a,b], Lihong Xu[a,*], Erik D. Goodman[b], Leilei Cao[a]

[a] College of Electronics and Information Engineering, Tongji University, Shanghai 201804, China
[b] BEACON Center for the study of Evolution in Action, Michigan State University, East Lansing, MI 48824, USA

ARTICLE INFO

Article history:
Received 6 April 2017
Received in revised form 3 August 2017
Accepted 24 September 2017
Available online 25 September 2017

Keywords:
Multi-objective evolutionary algorithm
Non-dominated sort
Pareto front
Computational complexity

ABSTRACT

Evolutionary computation has shown great performance in solving many multi-objective optimization problems; in many such algorithms, non-dominated sorting plays an important role in determining the relative quality of solutions in a population. However, the implementation of non-dominated sorting can be computationally expensive, especially for populations with a large number of solutions and many objectives. The main reason is that most existing non-dominated sorting algorithms need to compare one solution with almost all others to determine its front, and many of these comparisons are redundant or unnecessary. Another reason is that as the number of objectives increases, more and more candidate solutions become non-dominated solutions, and most existing time-saving approaches cannot work effectively. In this paper, we present a novel non-dominated sorting strategy, called Hierarchical Non-Dominated Sorting (HNDS). HNDS first sorts all candidate solutions in ascending order by their first objective. Then it compares the first solution with all others one by one to make a rapid distinction between different quality solutions, thereby avoiding many unnecessary comparisons. Experiments on populations with different numbers of solutions, different numbers of objectives and different problems have been done. The results show that HNDS has better computational efficiency than fast non-dominated sort, Arena's principle and deductive sort.

1. Introduction

During the past two decades, there has been a surge in research on evolutionary multi-/many-objective optimization (EMO) [1,2], especially for optimization problems with more than three objectives. The interest in multi-objective evolutionary algorithms (MOEAs) [3] is mainly inspired by the massive emergence of recognized real-world multi-objective optimization problems (MOPs)[4–7]. In fact, there are many scientific and engineering problems involving multiple conflicting objectives in the real world—e.g., greenhouse microclimate control [8,9], control system design [10,11], and search-based software engineering [12,13]. These objectives must be optimized simultaneously to obtain a tradeoff among the multiple objectives. To solve such problems, some effective MOEAs have been developed, such as genetic algorithm (GA) [14,15], differential evolution (DE) [16,17], indicator-based evolutionary algorithm (IBEA) [1,18], particle swarm optimization (PSO) [19,20], ant colony optimization (ACO) [21], decomposition-based algorithm (MOEA/D) [22] and so on.

It is worth noting that among these solution strategies, many MOEAs based on Pareto dominance [23,24] have shown excellent performance in solving MOPs. A large number of MOEAs based on this dominance mechanism have been developed, such as NSGA [25], SPEA [4], PESA [26], TDEA [27], MOGA [28], PAES [29], MOGA [25], NPGA [30], MOPSO [20], NSGA-II [14], SPEA2 [15], PESA-II [31] and so on. Recently, there has been increasing interest in addressing many-objective evolutionary algorithms—for example, Deb et al.'s NSGA-III [32,33], Yang et al.'s Grid-based strategy [34] and in several others [35–37].

Although various strategies have been proposed to solve the original problems faced by Pareto-based MOEAs, a common problem most existing Pareto-based MOEAs face is that Pareto dominance needs to conduct a large number of objective comparisons, which is computationally expensive, especially for populations with large number of solutions and many objectives. As an example, we implemented a multi-objective evolutionary algorithm called MODE [38] on a WFG6 [39] problem with five objectives and 800 initial individuals for 400 generations. We noted

* Corresponding author.
E-mail addresses: jiuzhe321@163.com (C. Bao), xulhk@163.com (L. Xu), goodman@egr.msu.edu (E.D. Goodman), mcaoleilei@sina.com (L. Cao).

https://doi.org/10.1016/j.jocs.2017.09.015
1877-7503/© 2017 Elsevier B.V. All rights reserved.

that nearly 78% of the computer instructions and 84% of the runtime were spent on non-dominated sorting. For different MOEAs and MOPs, this ratio may change, but the runtime spent on non-dominated sorting occupies a large proportion of the total time when populations are large and more than two objectives are involved.

In the past decade, there has been much research on improving the computational efficiency of non-dominated sorting in multi-/many-objective evolutionary algorithms. Originally, when N. Srinivas et al. [25] applied non-dominated sorting to multi-objective evolutionary algorithms, it had a computational complexity of $O(MN^3)$ and a space complexity of $O(N)$, where M is the number of objectives and N is the population size. In 2002, Deb et al. [14] put forward an improved version of non-dominated sorting called fast non-dominated sorting; the idea of this method is to avoid duplicated comparisons in determining the dominance relationships among solutions, and it reduced the computational complexity to $O(MN^2)$, at the cost of increasing its space complexity to $O(N^2)$. In 2014, Deb and Jain proposed an improved version of NSGA-II called NSGA-III [32,33], which uses a reference-point-based non-dominated sorting approach to solve many-objective optimization problems: two parts of the algorithm use the usual dominance principle [40].

In 2003, Jensen [41] applied the divide-and-conquer strategy [42] to non-dominated sorting, and reduced the run-time complexity to $O(GNlog^{M-1}N)$. It is worth noting that Jensen's approach assumed that no individuals share the same value for any objective, and it cannot work efficiently for some cases. Luckily, in 2013, Fortin et al. [43] put forward an improved version of Jensen's algorithm and compensated for this incompleteness without changing the original time complexity. In 2003, Ding et al. [44] proposed an improved non-dominated sorting approach called MLNFC, which adopts two techniques, getting an integer rank set and locking the non-dominated solutions as soon as possible. The computational complexity of this approach is O $(MN \log N)$ $+O$ (NL), where$L = \sum_{i=0}^{N-1} \frac{C^N_{NM-iM}}{C^N_{NM}}$. In 2008, Tang et al. [45] applied Arena's principle to non-dominated sorting—that is, each winner is preserved as the host and continues to take part in the next round of comparisons. Experiments showed that this approach outperformed Deb's fast non-dominated sort, especially for populations with a large proportion of dominated solutions. The time complexity of this approach can achieve$O(M\bar{N}N)$, among which$\bar{N}$is the number of non-dominated solutions in the population. In 2012, McClymont and Keedwell [46] proposed two notable algorithms for non-dominated sorting, called climbing sort and deductive sort, which use the existing results of comparisons to infer certain dominance relationships between solutions.

Recently, in 2015, Drozdik [47] et al. introduced a data structure, called an M-front, to reduce the time complexity of non-dominated sorting. This data structure holds the non-dominated solutions and updates this information as the population changes. The average time complexity of this dynamic approach can achieve$O(M^2N^{2-(1/M-1)})$. Experimental results show that this approach outperforms Jensen-Fortin's algorithm, especially for the many-objective case. In addition, there are also many other novel non-dominated sorting methods [44,48,49] or algorithms that can efficiently determine the non-dominated set [50–52]. All of these methods produce identical Pareto fronts; they simply aim to reduce the time complexity of non-dominated sorting as far as possible.

As mentioned above, the non-dominated sorting in most existing Pareto-based MOEAs is computationally expensive, especially for the case of many-objective optimization algorithms. This mechanism needs to spend a large amount of time in determining the dominance relationships between solutions in population; that is, most of the running time is consumed by objective comparisons. This can be attributed to two factors: first, non-dominated sorting is a complex process and must be implemented serially, so it costs a lot of time to determine the dominance relationships between solutions. Second, as the number of objectives increases, the number of non-dominated solutions increases very quickly, so the existing time-saving MOEAs do not work very effectively. Thus, how to avoid unnecessary or duplicative comparisons between solutions in many-objective evolutionary algorithms is an urgent problem to solve. In view of this, we perform this study, mainly engaging in the further improvement of non-dominated sorting in terms of time complexity.

In this paper, we present a new non-dominated sorting algorithm, termed Hierarchical Non-Dominated Sorting (HNDS). Unlike other non-dominated sort algorithms, HNDS does not need to compare each solution with all others before determining its Pareto rank; it first sorts all solutions by their first objective values in ascending order, which means that the first individual is a non-dominated solution and can never be dominated by any solutions further down the list. Then, HNDS compares the first solution with all other solutions one by one; the solutions dominated by the first solution will be discarded and will not be compared again in the determination of the current front. Thus, this method can save a large number of duplicative and unnecessary dominance comparisons, which improves the efficiency of the algorithm significantly. The running time complexity of HNDS can achieve O $(MN\sqrt{N})$ in some best cases. However, in the worst case, HNDS has a time complexity of O (MN^2), which is the same as that of Deb's fast non-dominated sort. However, average time complexity of HNDS is much lower. In addition, it is worth noting that HNDS has a space complexity of O (N). Theoretical analysis and experimental results both demonstrate that HNDS is more computationally efficient than the state-of-the-art non-dominated sorting algorithms to which we compare it, especially for large population sizes and many objectives.

The remainder of this paper is organized as follows. Section 2 gives a brief review of non-dominated sorting methods and a few well-known existing improved approaches. Section 3 describes the concrete implementation of our proposed non-dominated sorting algorithm and theoretically analyzes its time complexity. Extensive experiments are carried out in Section 4, and the results are presented to empirically compare with three state-of-the-art non-dominated sorting algorithms. Finally, in Section 5, conclusions of this study are drawn.

2. Preliminaries and background

In this section, we will give a simple description of non-dominated sorting, and then give a brief review of current widely used non-dominated sorting algorithms together with an analysis of their computational complexities.

2.1. Non-dominated sorting

Non-dominated sorting is mainly used to sort the solutions in population according to the Pareto dominance principle, which plays a very important role in the selection operation of many multi-objective evolutionary algorithms. Although different non-dominated sorting algorithms have been proposed based on different ideas, most of them follow a unified framework as shown in Fig. 1, and all of them achieve the same result, as shown in Fig. 2. In non-dominated sorting, an individual A is said to dominate another individual B, if and only if there is no objective of A worse than that objective of B and there is at least one objective of A better than that objective of B. Taking a minimization problem without loss of generality, we assume that the solutions of a population P

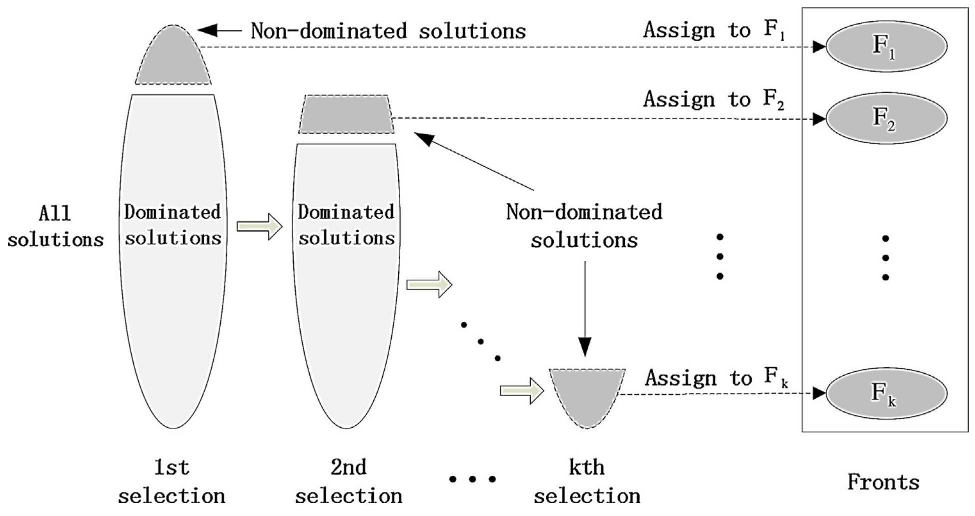


Fig. 1. Non-dominated sorting process of the solutions in a population.

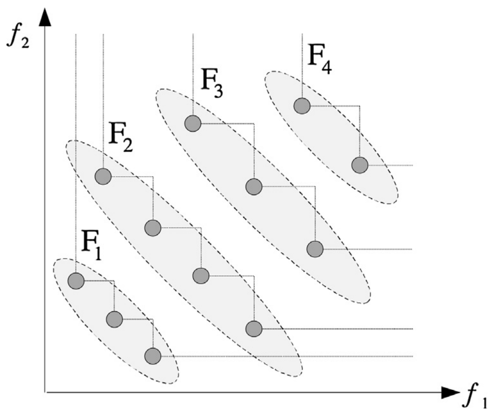


Fig. 2. An example of a Population with 12 solutions and 4 Pareto fronts.

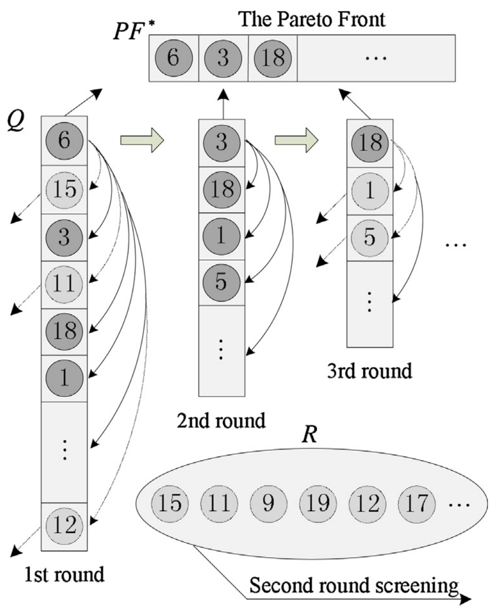


Fig. 3. Illustration of the hierarchical search strategy.

can be assigned to K Pareto fronts F_i, $i = 1, 2, \ldots, K$. Non-dominated sorting first selects all the non-dominated solutions from population P and assigns them to F_1 (the rank 1 front); it then selects all the non-dominated solutions from the remaining solutions and assigns them to F_2 (the rank 2 front); it repeats the above process until all individuals have been assigned to a front. (Note that these fronts are relative to the given population, without consideration of the optima of any process that may be generating them.)

2.2. Some existing non-dominated sorting methods

Since the non-dominated sorting algorithm [53] was first applied to the selection operation of multi-objective evolutionary algorithm, there have been many improved versions of the original approach, all of which try to reduce the number of redundant objective comparisons required to obtain the right dominance relationships among solutions. Here, we review a few of the widely used non-dominated sorting methods.

2.2.1. Fast non-dominated sorting

Fast non-dominated sorting [14] is the first improved version of the original non-dominated sorting algorithm. It compares each solution with each other and stores the results so as to avoid duplicate comparisons between every pair of solutions. For a population with N solutions, this method needs to conduct $MN(N\text{-}1)$ objective comparisons, which means that it has a time complexity of $O(MN^2)$. In addition, for each solution p, it needs a set S_P to store the individuals that are dominated by p and a counter n_p to record the number of individuals that dominate p, so this method has a space complexity of $O(N^2)$. As the number of solutions increases, both the time complexity and the space complexity increase quickly.

2.2.2. Arena's principle

Arena's principle [45] is an efficient strategy to reduce the runtime of non-dominated sorting. This method first selects a solution x from the population randomly, then it compares x with each other solution (for example y) in the population one by one; if x dominates y, y is removed from the population; if x and y are mutually non-dominating, y is removed to a set $\boldsymbol{R}$; otherwise, x is replaced by y and the comparisons continue until all solutions in the population have been compared. Finally, x will be a non-dominated solution, and it is put into the non-dominated solution set $\boldsymbol{Q}$. If there are any other solutions in set $\boldsymbol{R}$, all of them are treated as new candidate solutions and the above process is repeated until set $\boldsymbol{R}$ is empty. The time complexity of this approach can achieve $O(M\bar{N}N)$, where $\bar{N}$ is the number of the non-dominated solutions in the population.

Experimental results show that this approach outperforms Deb's fast non-dominated sort.

2.2.3. Deductive sort

Deductive sort [46] is another novel approach to reduce the unnecessary objective comparisons in non-dominated sorting. It assesses each solution in a fixed natural order and compares each solution with all following solutions. In this process, any solution that is dominated by the current solution will be marked and ignored. At the same time, if the current solution is found to be dominated by a certain solution, it will also be marked and the algorithm will skip over it to assess the next solution. By recording the results of dominance relationship between solutions, deductive sort can avoid some duplicate comparisons. It has a time complexity of $O(MN\sqrt{N})$ in the best case and a time complexity of $O(MN^2)$ in the worst case. It has a space complexity of $O(N)$.

Algorithm 1: The main program of HNDS for hierarchical non-dominated sorting

```
    Input: Initial population $\boldsymbol{P}$, $\boldsymbol{P} = \{p_1, p_2, \ldots, p_N\}$
    Output: The set of fronts $\boldsymbol{F}$, $\boldsymbol{F} = \{\boldsymbol{F}_1, \boldsymbol{F}_2, \ldots\}$
1:  $\boldsymbol{Q}=\boldsymbol{P}$; {assign all solutions to a set $\boldsymbol{Q}$}
2:  $k=0$; {initialize the front number }
3:  $\boldsymbol{R}$=empty; {used to store the solutions dominated by the first one in each round of comparison}
    $\boldsymbol{ND}$=empty; {used to store the solutions which are mutually non-dominated with the first one in each round of comparison}
4:  while set $\boldsymbol{Q}$ is not empty do
5:      $k=k+1$; {the number of current front }
6:      $\boldsymbol{F}_k$= empty; {Construct an empty set $\boldsymbol{F}_k$ to store the solutions belonging to the current front}
7:      Sort all solutions in set $\boldsymbol{Q}$ in ascending order by their first objective values;
8:      for all solutions in set $\boldsymbol{Q}$ do
9:          Assign each solution to a certain front $\boldsymbol{F}_k$ ( $k$ =1,2,...) according to Algorithm 2;
10:     end for
11: end while
12: return $\boldsymbol{F}$
```

3. A new strategy for non-dominated sorting

3.1. HNDS algorithm framework

Here, we propose a new search strategy for non-dominated sorting, termed Hierarchical Non-Dominated Sort (HNDS), which differs from current non-dominated sorting approaches by having an improved time complexity. The main idea of HNDS is shown in Fig. 3. It utilizes a hierarchical approach to determine the fronts one by one. The main advantages of this approach lie in its ability to quickly obtain a non-dominated individual at each round of comparison and its technology to quickly distinguish between solutions that belong to different fronts, which can avoid a number of duplicate comparisons.

The main program of HNDS is given in Algorithm 1. Taking the minimization problem as an example, it first sorts all solutions in set $\boldsymbol{Q}$ in ascending order by their first objective values, from which we can get the following conclusions: 1) according to the definition of non-dominated solution, the first solution is a non-dominated solution. 2) No matter what the number of objective is, this approach can obtain a non-dominated solution by comparing only the solutions' first objective values. 3) There are only two possible relationships between two solutions: either the preceding solution dominates the succeeding one, or they are mutual non-dominated. 4) A preceding solution can never be dominated by any succeeding solution. 5) Since all solutions have been sorted by their first objective values, it is necessary to compare only the remaining (M-1) objective values when determining their dominance relationships. This comparison strategy can continue until all solutions in a population have been sorted.

Based on the sorted solutions, HNDS begins to determine the front of a given rank one rank at a time, starting from F_1 and ending with F_{end}. As pointed out earlier, a solution may dominate any one of the succeeding solutions but it can never be dominated by any one of them, which means that the earlier a solution appears in the sorted list, the more solutions it may dominate. With this in mind, we can further infer that if we compare the first solution with all succeeding solutions one by one and discard the ones dominated by the first solution, it can reduce the number of objective comparisons needed as much as possible in this context. That is because once a solution is discarded, it will not again be compared with the first solution and also not with any solutions which are non-dominated with the first solution.

In order to make this comparison mechanism throughout the whole algorithm, as shown in Fig. 4, the solutions which are non-dominated with the first one should be transferred to the next round according to their current order. For example, in the first round of comparison, solution 3 is the first solution that is non-dominated with solution 6, so it will be ranked first in the second round of comparisons; solution 18 is the second one that is non-dominated with solution 6, so it will be ranked second in the second round of comparison. If it is also the first solution non-dominated with solution 3 in the second round of comparison, it will be ranked first in the third round of comparisons. It is worth noting that the solutions dominated by the first one in each round of comparisons do not necessarily follow this pattern, because a solution may be non-dominated with the first solution in the current round of comparisons, but may be dominated by any one of the first solutions in succeeding rounds; if this occurs, the latter part of list R may not be sorted according to the first objective.

In this way, the first solution in each round of comparison is always a non-dominated solution, because it is always non-dominated with the solutions that have been assigned to the current front. It also means that, from the second round of comparison, HNDS can obtain a non-dominated solution without any dominance comparison, which is different from other non-dominated sorting algorithms.

3.2. The implementation of hierarchical search strategy

Within the framework of HNDS, the fronts of the solutions will be determined from small to large one by one, with respect to their first objective. The pseudo code of the implementation process is given in Algorithm 2. At the beginning of each iteration, HNDS first assigns the first solution of $\boldsymbol{Q}$ to the current front, in what follows, it compares the first individual with the succeeding solutions one by one; if a succeeding solution is non-dominated with the first one, it will be moved to set $\boldsymbol{ND}$ for the next round of comparison; if it is dominated by the first one, it will be moved to set $\boldsymbol{R}$ for the determination of the next front. In this process, the solutions which are moved to set $\boldsymbol{R}$ do not involve sort operations, while the solutions that are non-dominated with the first one should be transferred to set $\boldsymbol{ND}$ according to their original order as shown in lines 5–10.

It is worth noting that the while loop in Algorithm 2 only applies to the case that set $\boldsymbol{Q}$ contains more than one solutions, which is different from the while loop in Algorithm 1. The while loop in Algorithm 2 is mainly used to generate a non-dominated solution in each round of comparison and to exclude dominated solutions as soon as possible. However, the while loop in Algorithm 1 is mainly used to ensure that each solution in set $\boldsymbol{Q}$ can be assigned to a front. The premise of the while loop in Algorithm 1 is that the set $\boldsymbol{Q}$ is not empty, which includes three possible cases: 1) if set $\boldsymbol{Q}$ is empty, the program will break. 2) If there is exactly one solution in set $\boldsymbol{Q}$, it will be assigned to the next front in line 15 of Algorithm 2 and the program will stop at the end of this loop. 3) If there is more than one solution in set $\boldsymbol{Q}$, the program will jump to the while loop in Algorithm 2 until the number of solutions in set $\boldsymbol{Q}$ is no more than one, then execute the while loop in Algorithm 1.

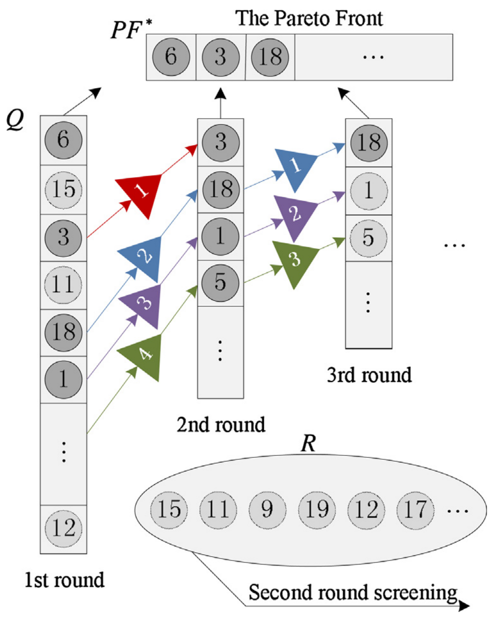


Fig. 4. Transfer rule of the solutions non-dominated with the first one in each round of comparison.

```
Algorithm 2: The implementation of Hierarchical Non-Dominated Sorting
    Input: Solution set $\boldsymbol{P}=\{P_1, P_2, \ldots, P_N\}$ sorted by their first objective values
    Output: Fronts with their corresponding solutions $\boldsymbol{P}[n]$
1:  while the number of solutions in set $\boldsymbol{Q}$ is more than one do
2:      Assign the first solution of $\boldsymbol{Q}$ to current front $\boldsymbol{F}_k$;
3:      $i$=1; { the current position in set $ND$}
4:      for $j$=2 to length($\boldsymbol{Q}$) do
5:          if $\boldsymbol{Q}_1$ is non-dominated with $\boldsymbol{Q}_j$ then
6:              Move $\boldsymbol{Q}_j$ to position i in $\boldsymbol{ND}$;
7:              $i$=$i$+1;
8:          else
9:              Move $\boldsymbol{Q}_j$ to set $\boldsymbol{R}$;
10:         end if
11:     end for
12:     Move the solutions of $\boldsymbol{ND}$ to set $\boldsymbol{Q}$; {for the next round of comparison}
13:     clear $\boldsymbol{ND}$; {for the next round of comparison}
14: end while
15: $\boldsymbol{F}_k$=[$\boldsymbol{F}_k$, $\boldsymbol{Q}$]; {Assign the last solution of $\boldsymbol{Q}$ to $\boldsymbol{F}_k$}
16: Move the solutions of $\boldsymbol{R}$ to set $\boldsymbol{Q}$; {for the determination of next front}
17: clear $\boldsymbol{R}$; {for the determination of next front}
```

Another detail in Algorithm 2 exists at the end of the while loop in line 15. The idea of hierarchical non-dominated sorting is to separate the non-dominated solutions from the dominated ones as soon as possible. At each round of comparison, the loop will generate a non-dominated solution. Accordingly, the last round of comparison is no exception: there will be at least one solution left in the last round. If more than one solution is left, it means that except for the first solution, all the other solutions are dominated and they will be moved to set $\boldsymbol{R}$; the set $\boldsymbol{Q}$ in line 15 will be empty, and front $\boldsymbol{F}_k$ will also be empty. If there is only one solution left, it will be assigned to front $\boldsymbol{F}_k$, which means that there is one and only one solution in front $\boldsymbol{F}_k$.

As an example, we use the necessity analysis method [54] to compare the number of comparisons conducted by HNDS with that of deductive sort on a population shown in Fig. 5. According to [55], the necessity of comparison between two solutions A and B can be divided into four cases: 1) solution A dominates solution B, which means that they belong to different fronts, if the front which solution B belongs to is next to the current front, it only needs to compare solution A with solution B, the other comparisons are redundant. 2) Solutions A and B are mutual non-dominated, if they belong to the same front and the front is the current front, the comparison is necessary. 3) Solutions A and B are mutual non-dominated, if they belong to the same front and the front is not the current front, the comparison is unnecessary. 4) Solutions A and B

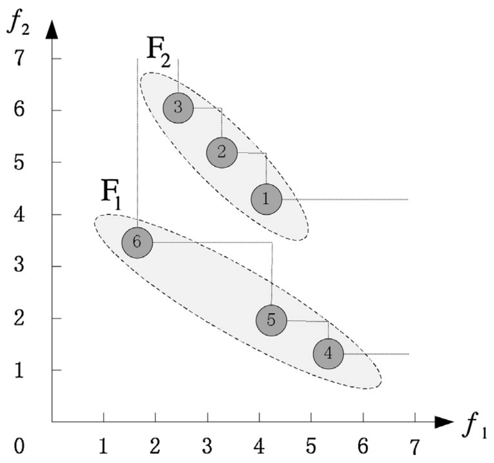


Fig. 5. A population with two fronts and six solutions.

Table 1
The number of dominance comparisons conducted by deductive sort for the population shown in Fig. 5.

Front	Comparison	Operation	Comparison Result
F1	(p1, p2)		Case3
	(p1, p3)		Case3
	(p1, p4)		Case4
	(p1, p5)		Case4
	(p1, p6)	P1 is ignored	Case1
	(p2, p3)		Case3
	(p2, p4)		Case4
	(p2, p5)		Case4
	(p2, p6)	P2 is ignored	Case1
	(p3, p4)		Case4
	(p3, p5)		Case4
	(p3, p6)	P3 is ignored	Case1
	(p4, p5)		Case2
	(p4, p6)	P4 is assigned to F1	Case2
	(p5, p6)	P5 and p6 are assigned to F1	Case2
F2	(p1, p2)		Case2
	(p1, p3)	P1 is assigned to F2	Case2
	(p2, p3)	P2 and p3 are assigned to F2	Case2

Case1: 3; Case2: 6; Case3: 3; Case4: 6; Total: 18.

are mutual non-dominated, if they belong to different fronts, the comparison is unnecessary.

Deductive sort determines the dominance relationships among solutions in a natural order. It must compare each solution with all the following solutions one by one. For the population shown in Fig. 5, it first compares p1 with p2; because p1 is not dominated by p2, the algorithm continues to compare p1 with p3; p1 is also not dominated by p3. Then p1 is compared with p4, p5 and so on, until it is dominated by p6. At this point, p1 is ignored and will no longer be compared in the determination of the current front. Then the algorithm continues to compare p2 with the following solutions and so on. In the comparison of p4, the algorithm first compares it with p5; because it is not dominated by p5, the algorithm continues to compare it with p6; it is also not dominated by p6, which means that it is a non-dominated solution and should be assigned to F1. Solutions p5 and p6 will be assigned to a corresponding front similarly to the assignment process for p4.

The detail of comparisons conducted by deductive sort for the population in Fig. 5 is listed in Table 1. In general, a total of 18 comparisons have been made. More concretely, there are three instances of Case 1, six instances of Case 2, three instances of Case 3 and six instances of Case 4. The comparisons of Case 3 and Case 4 account for half of the total comparisons, from which we know that there are quite a few redundant comparisons in deductive sort. Table 2 lists the number of comparisons conducted by Hierarchical

Table 2
The number of dominance comparisons conducted by HNDS for the population shown in Fig. 5.

Front	Comparison	Operation	Comparison Result
F1	(p6, p3)		Case1
	(p6, p2)		Case1
	(p6, p1)		Case1
	(p6, p5)		Case2
	(p6, p4)	p6 is assigned to F1	Case2
	(p5, p4)	p5 is assigned to F1	Case2
		p4 is assigned to F1	
F2	(p3, p2)		Case2
	(p3, p1)	p3 is assigned to F2	Case2
	(p2, p1)	p2 is assigned to F2	Case2
		p1 is assigned to F2	

Case1: 3; Case2: 6; Case3: 0; Case4: 0; Total: 9.

Non-Dominated Sorting, from which we can easily find that there are three instances of Case 1, six instances of Case 2, and no comparisons resulting in Case 3 or Case 4. It only conducts the necessary comparisons to accomplish non-dominated sorting for the population in Fig. 5, which are many fewer than those of deductive sort. It is worth noting that although HNDS does not conduct any comparison of Case 4 for the population in Fig. 5, this is not always the case for all populations. But from the idea of Hierarchical Non-Dominated Sorting, we can further infer that HNDS will never encounter an instance of Case 3 for any form of population.

3.3. *Analysis of the computational complexity*

According to the implementation of HNDS, it must first sort all candidate solutions in ascending order according to their first objective values. Then it continues using a hierarchical strategy to assign each solution to a certain front, one by one. Its computational complexity can also be divided into two parts: the computational complexity of the sorting and the computational complexity of the dominance comparisons.

Let N be the number of solutions in the population, and let M be the number of objectives. Let F be the number of rank i fronts F_i that result from the completion of the HNDS algorithm, and denote by N_i the number of individuals in each front F_i, where $1 \le i \le F$. Before the determination of each front F_i, the number of current candidate solutions P_i is N-N_1-N_2-. . .-N_{i-1}. The computational complexity of sorting these candidate solutions (using quick-sort) is $O(P_i \log P_i)$. Then the total computational complexity of sorting candidate solutions that belong to different fronts is

$$T_1 = O\left(\sum_{i=1}^{F}(P_i \log P_i)\right) \tag{1}$$

From the idea of Hierarchical Non-Dominated Sorting, we know that it can determine at least one non-dominated solution at each round of comparison, from which we can infer that it must conduct at most N_i rounds of comparisons to determine each front. In the first round of comparisons, HNDS must to compare the first solution with all following solutions one by one, which requires (N-1) dominance comparisons. At the same time, HNDS moves the first solution to the current front and moves the solutions dominated by the first one to a set $\boldsymbol{R}$ for the determination of next front, the remaining solutions are still sorted by their original order. Based on these remaining solutions, HNDS begins to conduct the second round of comparisons; we know that the first solution is also a non-dominated solution. It is compared with the following solutions one by one, which requires (N-2-d_1) dominance comparisons, where d_1 is the number of solutions dominated by the first one in the first round of comparisons. During this comparison process, HNDS

Table 3
Execution time of hnds on nine types of populations compared with three well-known non-dominated sorting algorithms. The best performance is highlighted with bold font and gray shading.

N	M	Time Complexity (s) *Mean+std*			
		Fast Non-dominated Sort	Arena's principle	Deductive Sort	HNDS
	2	0.445+0.002	0.256+0.002	0.132+0.001	**0.131+0.001**
200	5	0.471+0.001	0.259+0.005	0.135+0.012	**0.133+0.002**
	10	0.500+0.000	0.578+0.013	0.143+0.003	**0.142+0.002**
	2	2.717+0.008	1.093+0.004	0.572+0.003	**0.500+0.003**
500	5	2.880+0.009	1.175+0.006	0.648+0.004	**0.639+0.003**
	10	3.055+0.004	1.564+0.032	0.875+0.003	**0.696+0.021**
	2	7.322+0.047	1.766+0.007	0.950+0.005	**0.784+0.003**
800	5	7.761+0.051	2.122+0.004	**0.734+0.004**	0.903+0.004
	10	8.233+0.025	3.037+0.017	1.561+0.004	**1.320+0.004**

obtains the candidate solutions for the third round of comparisons. In what follows, HNDS continues the third round of comparisons; it must conduct (N-3-d_1-d_2) dominance comparisons, where d_2 is the number of solutions dominated by the first one in the second round of comparison. Similarly, for the *ith* round of comparison, HNDS must conduct (N-i-d_1-d_2-. . .-d_{i-1}) dominance comparisons. HNDS repeats this process until all solutions belonging to the current front have been determined. The number of dominance comparisons in determining front F_i can be calculated by adding up the comparisons conducted in each round of comparison; it can achieve

$$Num\_Comp_F i = N_i * N - (N_i + 1)N_i/2 - \sum_{i=1}^{N_i-1} (N_i - i) * d_i$$

The computational complexity of determining all fronts can be

$$T_2 = O\left(M \sum_{i=1}^{F} (Num\_Comp_{F_i})\right) \tag{2}$$

The worst case of HNDS occurs in two cases: 1) all solutions in the population are non-dominated ones, which also means that all solutions in the population can be assigned to one and only one front; then $N_i = N_1 = N$ and $d_i = 0$. Alternatively, 2) all solutions in the population can be assigned to different fronts and each solution is dominated by all solutions belonging to preceding fronts. In these two cases, all solutions must be compared with each other, and the computational complexity of determining all fronts is:

$$T_{2\_worst} = M * N(N - 1)/2 = O(MN^2)$$

In the best case, N solutions in the population are evenly assigned to $\sqrt{N}$ fronts, and each solution is dominated by all other solutions that belong to preceding fronts. According to HNDS, the number of comparisons for determining all fronts is $N(\sqrt{N}$-1) +1, and the computational complexity is:

$$T_{2\_best} = M * [N(\sqrt{N} - 1) + 1] = O(MN\sqrt{N})$$

Through the above analysis, we can get that the total computational complexity of HNDS in the worst case is:

$$T_{total\_worst} = T_1 + T_{2\_worst} = O(MN^2) \tag{3}$$

The total computational complexity of HNDS in the best case is:

$$T_{total\_best} = T_1 + T_{2\_best} = O(MN\sqrt{N}) \tag{4}$$

In the implementation of HNDS, all solutions in the population should be sorted according to their first objective. At the same time, the solutions which are dominated by the first one at each round of comparison should be moved to the next round according to their current order. Thus, HNDS has a space complexity of $O(N)$.

4. Experimental results

In order to verify the efficiency of our approach, we conduct four groups of experiments and compare HNDS with three famous non-dominated sorting approaches, including the fast non-dominated sort [14], Arena's principle [45] and deductive sort [46]. Each non-dominated sorting approach is embedded in the naive NSGA-II, and each approach produces exactly the same output; the only difference is that they use different non-dominated sorting mechanisms and consume different amounts of runtime.

The experiments are implemented on different kinds of populations, including randomly generated populations with different number of individuals, random populations with different numbers of objectives and random populations generated in different benchmark problems. The performance indicators of the non-dominated sorting algorithms are presented by the number of dominance comparisons and the execution times they consume. All experiments are conducted on a PC under a Windows 7 SP1 64-bit operation system and 1.60 GHz AMD E-350 CPU with 6.00G memory. Each algorithm is implemented in MATLAB R2014a.

4.1. Experiments on random populations

In the first group of experiments, we explore the relationship between the computational complexity and the number of solutions (i.e., population size). The experiments are conducted on three types of populations: a population with two objectives, a population with five objectives and a population with ten objectives. Each type of population is generated randomly—in other words, each objective value of all solutions in a population is randomly generated from a uniform distribution over the interval [0,1]. The number of solutions in each type of population is set from 100 to 5000 with an increment of 100.

For each non-dominated sorting algorithm, Figs. 6 and 7 present the numbers of comparisons of individual objectives and the execution times versus the number of solutions, respectively. In these

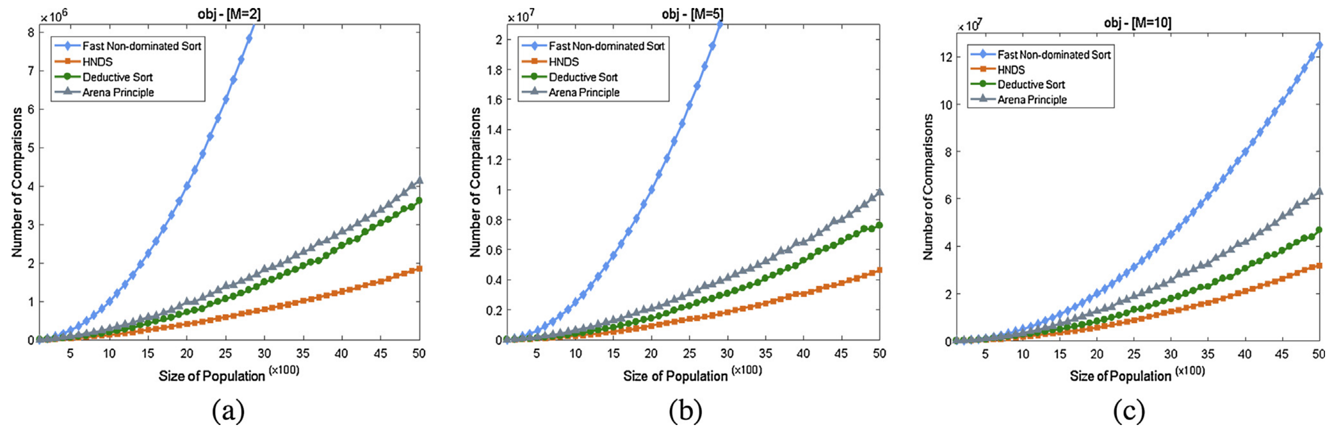


Fig. 6. Number of objective comparisons conducted by four non-dominated sorting approaches on three types of random populations ($M=2, 5, 10$).

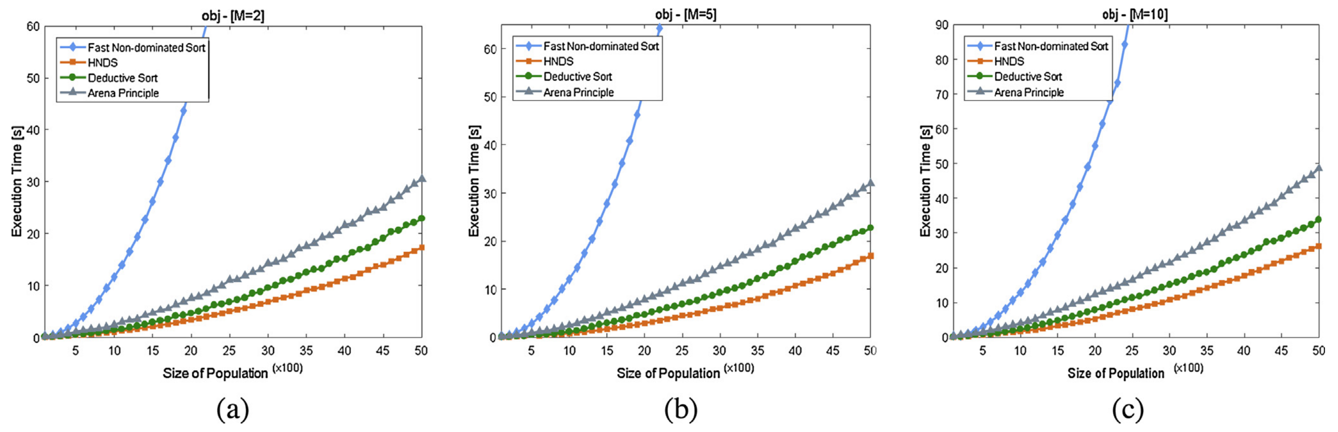


Fig. 7. Running time consumed by four non-dominated sorting approaches on three types of random populations ($M=2, 5, 10$).

figures, each algorithm has been run 10 times independently and the average values have been recorded.

From Figs. 6 and 7, we can see that for each non-dominated sorting approach, both the number of dominance comparisons and the execution time increase with a certain exponent as the number of solutions increases. For each type of population, the naïve NDS from NSGA-II conducts the most comparisons and consumes the most execution time, Arena's principle ranks second. For the case of two objectives, HNDS clearly outperforms the other three non-dominated sorting algorithms. For the case of five objectives, when the population size is no more than 700, HNDS does about the same number or even a few more objective comparisons than deductive sort, but as the population size continues to increase, HNDS clearly outperforms deductive sort. The case of ten objectives has similar results. Also, as the number of objectives increases, the difference between HNDS and deductive sort decreases slightly, while the difference between deductive sort and Arena's principle increases slightly.

In order to further explore the relationship between the number of objectives and the execution times of different non-dominated sorting algorithms, we consider random populations containing 200, 500 or 800 solutions, and for each size, having 2, 5 or 10 objectives. Table 3 shows the mean values and standard deviations of execution times of each algorithm. The best performance in each case is highlighted with bold font and gray shaded. From Table 3, we can see that HNDS outperforms the naïve NSGA-II, Arena's principle and deductive sort in all but one case. The exception is for the population with 800 solutions and 5 objectives, for which case deductive sort consumes less time than HNDS.

4.2. Explanation regarding populations with fixed (Artificially generated) fronts

Many papers [46,55] have used a very artificial procedure (given in Algorithm 3) to enable them to generate problem instances with a specified number of fronts. It is our contention that the extremely uncharacteristic distribution of the solutions generated by this algorithm means that it provides no useful insight into the relative performance of NDS algorithms in sorting sets of instances that arise either from random generation or from populations in a multi-objective optimization problem. That is, the performance of NDS algorithms on a set with a particular number of fronts should be done either by 1) generating random populations and counting the number of fronts, then tabulating the sorting performance for THIS number of fronts, or 2) analyzing the behavior of an NDS algorithm embedded in a particular multi-objective optimization algorithm, once again tabulating the performance versus the number of fronts encountered at any particular generation. This type of analysis will provide far greater insight into the expected relative performance of the NDS algorithm in actual usage.

Let's examine briefly the problem with the commonly used algorithm for generating problem instances with a pre-specified number of fronts. Fig. 8 shows the effect of generating an instance with a population of 9 solutions and with 3 fronts. As you will observe, the generation algorithm (Algorithm 3) generates a problem instance with a very special property: every individual in any front dominates ALL individuals in any higher-numbered front. This is FAR from the usual distribution of individuals in a set of fronts, and changes significantly the ease of sorting such individuals. Thus, it is a very poor benchmark. It is a *convenient* benchmark, in that it allows, for example, generating a population of 50 individuals in 50 fronts, but that is clearly very far from a typical distribution of fronts in a population with 50 individuals.

Therefore, results on this benchmark will not be analyzed in this paper, although they again proved favorable to HNDS.

conducted by fast non-dominated sort remains the same as the number of objectives increases. The numbers of comparisons conducted by the other three non-dominated sorting algorithms tend to be similar to the numbers of fast non-dominated sort when the number of fronts exceeds 15–20; for lower numbers of fronts, HNDS performs the fewest comparisons. For populations with 1000 and 3000 solutions, fast non-dominated sort consumes the most execution time, while Arena's principle and deductive sort rank second and third, respectively, and HNDS consumes the least. In addition, for the population with 5000 solutions, fast non-dominated sort also consumes the most execution time, HNDS consumes the least. The execution time consumed by Arena's principle tends to be similar to or even less than that of deductive sort as the number of objectives increases.

Algorithm 3: Creation of populations with fixed numbers of fronts

```
    Input: Random Population P={s_1, s_2, …, s_N}
    Output: Populations with fixed numbers of fronts PF(t) , where t=1,2, …, 50
1:  P=k*rand(N, M);  { generate a random population P}
2:      for  i=1 to M do  {sort all objectives in ascending order}
3:          b=P( :, i );
4:          P (:, i)= sort(b);
5:      end for
6:  f= P (:, 2) ;  {select the second objectives of all solutions}
7:  S=[ ] ;  {used to store the second objectives of all fixed fronts}
8:  for  m=1: 50 do  {the number of fronts is set from 1 to 50 with an increment of one}
9:  N=2000;  {the number of solutions}
10:         while  (mod(N, m) ~= 0)  do  {make sure that each front has the same
                                          number of individuals}
11:             N=N-1;
12:         end while
13:     n=N/m;  {the number of solutions in each front}
14:     s=[ ] ;  {used to store the second objectives of the solutions in current fixed front}
15:     for  i=0:( m-1)  {create the second objectives of the solutions in current fixed
                          front}
16:         j=i*n;
17:         k=f(j+1: j+n);
18:         k= sort(k, 'descend');
19:         s=[s; k];
20:     end for
21:     if  ( mod(2000, m)~= 0)  do {if all solutions cannot be equally assigned to front m,
                                    move the remaining solutions to the last front}
22:         i= m-1;
23:         j= i*n;
24:         k= f(j+1: end);
25:         k= sort(k, 'descend');
26:         s=s(1: n*(m-1));
27:         s=[s; k];
28:     end if
29:     S=[S, s];  {store the second objectives of each fixed front to set S}
30: end for
31: PF(t)= [P (:,1), S(:, t), P (:,3), P (:,4), P (:,5), P (:,6), P (:,7), P (:,8), P (:,9), P (:,10)] ;
```

4.3. *Experiments on populations with different numbers of objectives*

In order to further explore the influence of the number of objectives on random populations, we conduct a group of experiments on populations with 1000, 3000 and 5000 solutions respectively. The number of objectives is set from 2 to 30 with an increment of one for each type of population and each objective value of the solution is randomly generated from a uniform distribution over the interval [0,1]. The relationship between the number of fronts and the number of objectives for random population is shown in Fig. 9, from which we can see that the number of fronts decreases very quickly as the number of objectives increases.

Figs. 10 and 11 present the results of the four non-dominated sorting algorithms on populations used in this group of experiments. We can easily see that the number of comparisons

4.4. *Experiments on the various non-dominated sorting algorithms imbedded in the NSGA-II framework*

To demonstrate the performance of HNDS in multi-objective evolutionary algorithms, we embed HNDS and the other three non-dominated sorting algorithms into the naïve NSGA-II respectively.

Table 4
Time ratio of nsga-ii framework embedded with different non-dominated sorting algorithms on tnk and dtlz2 problems. the best performances are shown in bold font and gray shading.

Test problem	Number of solutions	Number of objectives	Time Ratio (**TR**)			
			NSGA-II	Arena's Principle	Deductive Sort	HNDS
TNK	200	2	1.000	0.623	0.412	**0.409**
	300		1.000	0.492	0.319	**0.278**
	500		1.000	0.465	0.321	**0.292**
DTLZ2	200	2	1.000	0.329	0.197	**0.134**
		3	1.000	0.302	0.201	**0.103**
		5	1.000	0.351	0.231	**0.162**
		8	1.000	0.416	0.326	**0.273**
		10	1.000	0.468	0.386	**0.341**

NSGA-II framework.

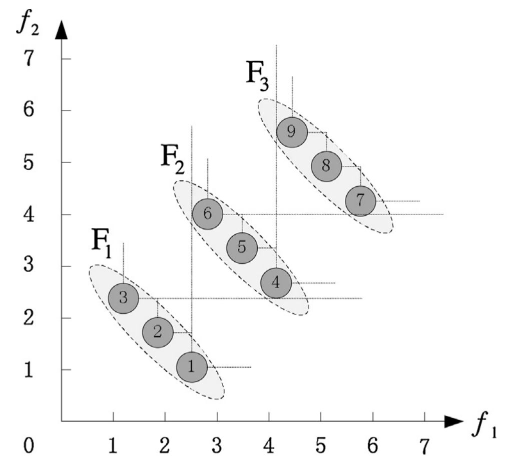


Fig. 8. A population with 9 individuals and three fixed fronts.

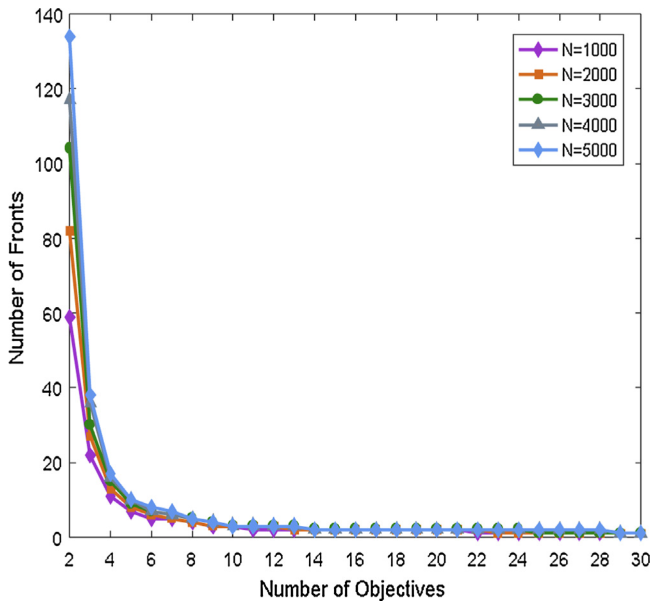


Fig. 9. Relationship between the Number of Fronts and the Number of Objectives for Random Populations (N = 1000, 2000, 3000, 4000 and 5000).

In the experiments, we choose the TNK problem and DTLZ2 problem [56] as the test problems on which to compare the algorithms' performances. The TNK problem has a Pareto front of several discontinuous curves, while the Pareto front of the DTLZ2 problem is composed of the first quadrant circle. The number of generation is set at 200, the simulated binary crossover (SBX) is used, the other parameters are the same as in [57]. For the TNK problem, 2 objectives are used, and populations contain 200, 300 or 500 individuals. For the DTLZ2 problem, which has a concave, continuous Pareto front, 2, 3, 5, 8 and 10 objectives are used, and population size is 200 individuals. In addition, to ensure that each non-dominated sorting algorithm is tested on exactly the same problems, matching random seeds are employed across algorithms.

Table 4 shows the time ratio (TR) [55] of the NSGA-II framework embedded with different non-dominated sorting approaches. The best performances are shown in bold font and gray shading. The definition of time ratio is shown in Formula (3). The NSGA-II framework embedded with each non-dominated sorting algorithm is run ten times independently and each execution time is recorded. The time ratio is the ratio of the total execution time consumed by the NSGA-II framework embedded with a certain non-dominated sorting algorithm and the total execution time consumed by the naive

$$TR = \frac{\sum_{i=1}^{10} T^i}{\sum_{i=1}^{10} T^i_{naive}} \quad (3)$$

From Table 4, we can easily see that the NSGA-II frameworks embedded with HNDS consume the least execution time for all cases, which also means that HNDS can effectively reduce the computational intensity of multi-objective evolutionary algorithms in terms of execution time.

5. Conclusion

In this paper, we propose a novel non-dominated sorting algorithm, called HNDS, to speed up the non-dominated sorting of a population of solutions. This approach adopts a hierarchical search strategy to reduce the number of dominance comparisons, so as to reduce the computational complexity. HNDS has a best-case time complexity of $O(MN\sqrt{N})$ and a space complexity of $O(N)$, although its worst-case time complexity is $O(MN^2)$, similar to several others. But experimental results show that in terms of dominance comparisons and objective comparisons performed and execution time, HNDS outperforms fast non-dominated sort, Arena's principle and deductive sort across a wide range of situations.

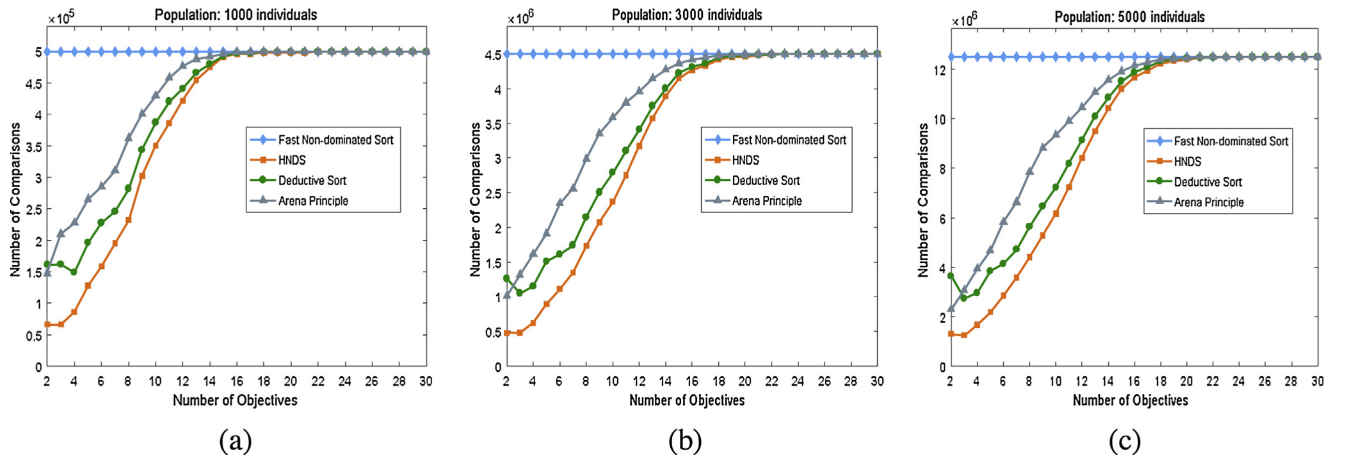


Fig. 10. Number of dominance comparisons conducted by four non-dominated sorting algorithms on random populations with different numbers of objectives ($N = 1000$, 3000 and 5000).

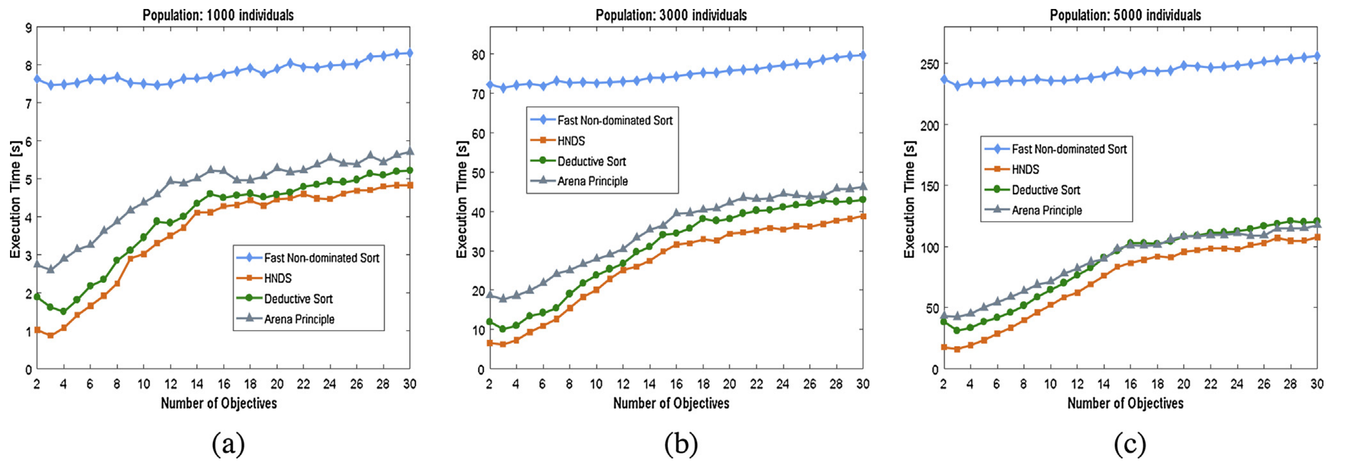


Fig. 11. Execution time consumed by four non-dominated sorting algorithms on random populations with different numbers of objectives ($N = 1000$, 3000 and 5000).

HNDS has played an effective role in non-dominated sorting, especially for populations with large numbers of solutions and many objectives. It is worth noting that in the process of creating candidates for the second front through last fronts, the sorting of all these candidate solutions by their first objectives may take advantage of their already partially-sorted order. Moving these solutions according to their initial order in the first objective can save many comparisons and reduce runtime. We plan to explore sorting that takes advantage of the special partially-sorted nature of the solutions internal to the HNDS process. This could prove to be beneficial, especially for populations with large numbers of solutions and many objectives.

Acknowledgments

This work was supported in part by the National High-Tech R&D Program of China (863 Program) under grant 2013AA102305, in part by the National Natural Science Foundation of China under grant 61573258, and in part by the U.S. National Science Foundation's BEACON Center for the Study of Evolution in Action, under cooperative Agreement DBI-0939454.

References

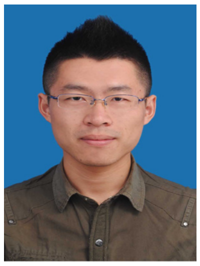
[1] T. Wagner, N. Beume, B. Naujoks, Pareto-, aggregation-, and indicator-based methods in many-objective optimization, Evolutionary Multi-Criterion Optimization, Springer, Berlin/Heidelberg, 2007, pp. 742–756.
[2] H. Ishibuchi, N. Tsukamoto, Y. Nojima, Evolutionary many-objective optimization: a short review, CEC (Conference on Evolutionary Computation),2008 in IEEE World Congress on Computational Intelligence (2008) 2419–2426.
[3] A.M. Zhou, B.Y. Qu, H. Li, S.Z. Zhao, P.N. Suganthan, Q.F. Zhang, Multiobjective evolutionary algorithms: a survey of the state of the art, Swarm Evol. Comput. 1 (1) (2011) 32–49.
[4] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach, IEEE Trans. Evol. Comput. 3 (4) (1999) 257–271.
[5] Y. Yuan, H. Xu, Multiobjective flexible job shop scheduling using memetic algorithms, IEEE Trans. Autom. Sci. Eng. 12 (1) (2015) 336–353.
[6] V. Khare, X. Yao, K. Deb, Performance Scaling of Multi-objective Evolutionary Algorithms, Springer, Berlin, Heidelberg, 2003, 376–90 p.
[7] C.A.C. Coello, G.T. Pulido, M.S. Lechuga, Handling multiple objectives with particle swarm optimization, IEEE Trans. Evol. Comput. 8 (3) (2004) 256–279.
[8] D.N. Gerasimov, M.V. Lyzlova, Adaptive control of microclimate in greenhouses, J. Comput. Syst. Sci. Int. 53 (6) (2014) 896–907.
[9] M. Ramdani, A. Hamza, M. Boughamsa, Multiscale fuzzy model-based short term predictive control of greenhouse microclimate, 2015 IEEE 13th International Conference on Industrial Informatics (INDIN). IEEE (2015) 1348–1353.
[10] J.G. Herrero, A. Berlanga, J.M.M. Lopez, Effective evolutionary algorithms for many-Specifications attainment: application to air traffic control tracking filters, IEEE Trans. Evol. Comput. 13 (1) (2009) 151–168.
[11] P.J. Fleming, R.C. Purshouse, R.J. Lygoe, Many-objective optimization: an engineering design perspective, Evolut. Multi-criterion Optim. 5 (2005) 14–32.
[12] A.S. Sayyad, T. Menzies, H. Ammar, On the value of user preferences in search-based software engineering: a case study in software product lines, Proceedings of the 2013 International Conference on Software Engineering, IEEE Press (2013) 492–501.

[13] K. Praditwong, M. Harman, X. Yao, Software module clustering as a multi-objective search problem, IEEE Trans. Software Eng. 37 (2) (2011) 264–282.
[14] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.
[15] E. Ziztler, M. Laumanns, L. Thiele, Spea2: Improving the strength pareto evolutionary algorithm for multiobjective optimization, Evol. Methods Design Optim. Control (2002) 95–100.
[16] R. Storn, K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optim. 11 (4) (1997) 341–359.
[17] T. Robic, B. Filipic, Differential evolution for multiobjective optimization, Evolutionary Multi-Criterion Optimization, Springer, Berlin/Heidelberg, 2005, pp. 520–533.
[18] H. Ishibuchi, T. Doi, Y. Nojima, Incorporation of scalarizing fitness functions into evolutionary multiobjective optimization algorithms, Parallel Problem Solving from Nature – PPSN IX (2006) 493–502.
[19] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, Proceedings of the Sixth International Symposium on Micro Machine and Human Science, 1995. MHS'95. IEEE (1995) 39–43.
[20] C.A.C. Coello, M.S. Lechuga, MOPSO: a proposal for multiple objective particle swarm optimization, Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002) 2 (2002) 1051–1056.
[21] S.K. Chaharsooghi, A.H.M. Kermani, An intelligent multi-colony multi-objective ant colony optimization (ACO) for the 0-1 knapsack problem, IEEE Conference on Evolutionary Computation, 2008 (CEC 2008) in IEEE World Congress on Computational Intelligence (2008) 1195–1202.
[22] Q. Zhang, H. Li, MOEA/D: AMultiobjective evolutionary algorithm based on decomposition, IEEE Trans. Evol. Comput. 11 (6) (2007) 712–731.
[23] D.A. Van Veldhuizen, G.B. Lamont, Multiobjective evolutionary algorithms: analyzing the state-of-the-art, Evol. Comput. 8 (2) (2000) 125–147.
[24] M.R. Chen, J. Weng, X. Li, X. Zhang, Handling multiple objectives with integration of particle swarm optimization and extremal optimization, in: Foundations of Intelligent Systems, Springer, Berlin, Heidelberg, 2014, pp. 287–297.
[25] N. Siinivas, K. Deb, Multiobjective optimization using nondominated sorting in genetic algorithms, Evol. Comput. 2 (3) (1994) 221–248.
[26] D.W. Corne, J.D. Knowles, M.J. Oates, The pareto envelope-based selection algorithm for multiobjective optimization. International Conference on Parallel Problem Solving From Nature, Springer, Berlin, Heidelberg, 2000, pp. 839–848.
[27] I. Karahan, M. Koksalan, A territory defining multiobjective evolutionary algorithms and preference incorporation, IEEE Trans. Evol. Comput. 14 (4) (2010) 636–664.
[28] Genetic algorithms for multiobjective optimization: formulation, discussion and generalization, in: C.M. Fonseca, P.J. Fleming (Eds.), Proceedings of ICGA-93: Fifth International Conference on Genetic Algorithms, 17–22 July 1993, San Mateo, CA, USA : Morgan Kaufmann, 1993.
[29] J. Knowles, D. Corne, The pareto archived evolution strategy: a new baseline algorithm for pareto multiobjective optimisation, Proceedings of the 1999 Congress on Evolutionary Computation (CEC 1999) 1 (1999) 98–105.
[30] J. Horn, N. Nafpliotis, D.E. Goldberg, A niched Pareto genetic algorithm for multiobjective optimization. Evolutionary Computation, 1994 IEEE World Congress on Computational Intelligence, IEEE (1994) 82–87.
[31] PESA-II: Region-based selection in evolutionary multiobjective optimization, in: D.W. Corne, N.R. Jerram, J.D. Knowles, et al. (Eds.), Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation, Morgan Kaufmann Publishers Inc., 2001, pp. 283–290.
[32] K. Deb, H. Jain, An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints, IEEE Trans. Evol. Comput. 18 (4) (2014) 577–601.
[33] H. Jain, K. Deb, An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part II: handling constraints and extending to an adaptive approach, IEEE Trans. Evol. Comput. 18 (4) (2014) 602–622.
[34] S.X. Yang, M.Q. Li, X.H. Liu, J.H. Zheng, A grid-based evolutionary algorithm for many-objective optimization, IEEE Trans. Evol. Comput. 17 (5) (2013) 721–736.
[35] X.Y. Zhang, Y. Tian, Y.C. Jin, A knee point-driven evolutionary algorithm for many-objective optimization, IEEE Trans. Evol. Comput. 19 (6) (2015) 761–776.
[36] K. Li, K. Deb, Q.F. Zhang, S. Kwong, An evolutionary many-objective optimization algorithm based on dominance and decomposition, IEEE Trans. Evol. Comput. 19 (5) (2015) 694–716.
[37] Y. Yuan, H. Xu, B. Wang, X. Yao, A new dominance relation-based evolutionary algorithm for many-Objective optimization, IEEE Trans. Evol. Comput. 20 (1) (2016) 16–37.
[38] F. Xue, A.C. Sanderson, R.J. Graves, Ieee: Pareto-based Multi-objective Differential Evolution, 2003, 862–9 p.
[39] S. Huband, P. Hingston, L. Barone, L. While, A review of multiobjective test problems and a scalable test problem toolkit, IEEE Trans. Evol. Comput. 10 (5) (2006) 477–506.
[40] C. Vira, Y.Y. Haimes, Multiobjective Decision Making: Theory and Methodology: North-Holland, 1983.
[41] M.T. Jensen, Reducing the run-time complexity of multiobjective EAs: the NSGA-II and other algorithms, IEEE Trans. Evol. Comput. 7 (5) (2003) 503–515.
[42] H.T. Kung, F.P. Preparata, On finding the maxima of a set of vectors, J. ACM 22 (4) (1975) 469–476.
[43] F.A. Fortin, S. Grenier, M. Parizeau, Generalizing the improved run-time complexity algorithm for non-Dominated sorting, Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, ACM (2013) 615–622.
[44] L.X. Ding, S.Y. Zeng, L.S. Kang, A fast algorithm on finding the non-dominated set in multi-objective optimization, IEEE Congress on Evolutionary Computation, 2003 (CEC2003) 4 (2003) 2565–2571.
[45] S. Tang, Z. Cai, J. Zheng, A fast method of constructing the Non-Dominated set: Arena's principle, ICNC'08, IEEE, in: Fourth International Conference on Natural Computation, 1, 2008, pp. 391–395.
[46] K. McClymont, E. Keedwell, Deductive sort and climbing sort new methods for non-dominated sorting, Evol. Comput. 20 (1) (2012) 1–26.
[47] M. Drozd, Y. Ed Akimoto, H. Aguirre, K. Tanaka, Computational cost reduction of nondominated sorting using the M-front, IEEE Trans. Evol. Comput. 19 (5) (2015) 659–678.
[48] S. Chuan, C. Ming, S. Zhongzhi (Eds.), A Fast Nondominated Sorting Algorithm. 2005 International Conference on Neural Networks and Brain (2005), 13–15 Oct. 2005.
[49] H. Fang, Q. Wang, Y.C. Tu, M.F. Horstemeyer, An efficient non-dominated sorting method for evolutionary algorithms, Evol. Comput. 16 (3) (2008) 355–384.
[50] X. Zhou, J. Shen, J.X. Shen, in: Y. Zhang, H. Tan, Q. Luo (Eds.), An Immune Recognition Based Algorithm for Finding Non-dominated Set in Multi-objective Optimization, 2008 (291-6 p.).
[51] J.H. Zheng, Z.Z. Shi, C.X. Ling, Y. Xie, A new method to construct the non-dominated set in multi-objective genetic algorithms, in: Z. Shi, Q. He (Eds.), Intelligent Information Processing Ii. International Federation for Information Processing., 2017, 1632005. p. 457–70.
[52] J.E. Fieldsend, R.M. Everson, S. Singh, Using unconstrained elite archives for multiobjective optimization, IEEE Trans. Evol. Comput. 7 (3) (2003) 305–323.
[53] J.H. Holland, GDE: Genetic Algorithms in Search Optimization and Machine Learning, 1989.
[54] X.Y. Zhang, Y. Tian, R. Cheng, Y.C. Jin, An efficient approach to nondominated sorting for evolutionary multiobjective optimization, IEEE Trans. Evol. Comput. 19 (2) (2015) 201–213.
[55] Y. Zhou, Z. Chen, J. Zhang, Ranking vectors by means of the dominance degree matrix, IEEE Trans. Evol. Comput. (2016).
[56] K. Deb, L. Thiele, M. Laumanns, E. Zitzler, Scalable test problems for evolutionary multiobjective optimization, in: Evolutionary Multiobjective Optimization Theoretical Advances and Applications, 2005, pp. 105–145.
[57] K. Deb, J. Sundar, in: M. Keijzer (Ed.), Reference Point Based Multi-objective Optimization Using Evolutionary Algorithms, Assoc Computing Machinery, New York, 2006, 635–42 p.

Chunteng Bao received the B.S. degree in Mechanical design manufacturing and automation from Henan University of Technology, Zhengzhou, China, in 2010, and the M.S. degree in control theory and control engineering from Shanghai Maritime University, Shanghai, China, in 2013. He is currently pursuing the Ph.D degree in Control Science and Engineering at the College of Electronics and Information Engineering, Tongji University, Shanghai, China. His research interests include intelligent control, evolutionary computation, and multi-objective optimization.

Lihong Xu received the Ph.D. degree from the Department of Automatic Control, Southeast University in Nanjing, China in 1991. He became a member of IEEE in 1989. In 1994, he was specially appointed as a professor in Southeast University. He transferred to Tongji University in August 1997, and has been a professor in Tongji University since then. His research fields include control theory, computational intelligence, and optimization theory. He got the first prize of Science and Technology Advancement Award of Ministry of Education of China in 2005 and the second prize of National Science and Technology Advancement Award of China in 2007, respectively. Dr. Xu is a member of ACM, and the president of IEEE CIS's Shanghai Chapter. He was the co-Chair of the 2009 GEC Summit in Shanghai. He is now doing joint research work as a visiting professor and advisor with the greenhouse research team of BEACON, USA.

Erik D. Goodman received the B.S. degree in mathematics and the M.S. degree in system science from Michigan State University (MSU), East Lansing, MI, USA, and the Ph.D. degree in computer and communication sciences from the University of Michigan, Ann Arbor, MI, USA, in 1972. He is currently the Director of the BEACON Center for the Study of Evolution in Action, an NSF Science and Technology Center, MSU. He has been a Professor with the Department of Electrical and Computer Engineering, MSU, since 1984, where he also co-directs the Genetic Algorithms Research and Applications Group. He is the Co-Founder in 1999 and the former Vice President for Technology of Red Cedar Technology, Inc. He is an Advisory Professor with Tongji University, Shanghai, China, and East China Normal University, Shanghai. Dr. Goodman is a Senior Fellow of the International Society for Genetic and Evolutionary Computation. He was the Chair of the Executive Board of that society from 2001 to 2004. From 2005–2007, he was the Founding Chair of the ACM SIG for Genetic and Evolutionary Computation.



Leilei Cao received the B.S. degree in vehicle engineering from Yangzhou University, Yangzhou, China, in 2011, and the M.S. degree in vehicle engineering from Jiangsu University, Zhenjiang, China, in 2014. He is currently pursuing the Ph.D degree in Control Science and Engineering at the College of Electronics and Information Engineering, Tongji University, Shanghai, China. His research interests include evolutionary computation, dynamic optimization, and multi-objective optimization.