Efficient Non-domination Level Update Approach for Steady-State Evolutionary Multiobjective Optimization

Ke Li[1], Kalyanmoy Deb[1], Qingfu Zhang[2], and Sam Kwong[2]

[1]Department of Electrical and Computer Engineering, Michigan State University
[2]Department of Computer Science, City University of Hong Kong

COIN Report Number: 2014014

Abstract

Non-dominated sorting, dividing the population into several non-domination levels, is a basic step for many Pareto-based evolutionary multiobjective optimization (EMO) algorithms. Different from the generational scheme, where the selection of next parents is conducted after the generation of a population of offspring, the steady-state scheme updates the parent population whenever a new candidate solution is reproduced. As a consequence, non-dominated sorting can be a computationally expensive part in steady-state EMO algorithm when the population size or number of objectives becomes large. In fact, before introducing a new candidate solution, the non-domination level structure of the parent population is already known from the last environmental selection. Thus, conducting non-dominated sorting from scratch, each time, obviously does not take advantages of the existing knowledge and results in a severe waste of computational resources. In view of this, we propose an efficient non-domination level update (ENLU) mechanism for steady-state EMO algorithm. By extracting the knowledge of the non-domination level structure of the parent population, ENLU mechanism only updates the non-domination levels of necessary solutions during the update procedure, including the addition of a new candidate solution into the parent popualtion and the elimination of an inferior solution. Theoretical analysis show the time complexity of ENLU mechanism is $O(mN\sqrt{N})$ in the worst-case and $O(m)$ in the best case. Extensive experiments empirically demonstrate that ENLU mechanism is more computationally efficient than the fast non-dominated sorting procedure, as it saves 10^3 to 10^4 magnitude of unnecessary dominance comparisons.

1 Introduction

A multiobjective optimization problem (MOP) can be stated as follows:

$$\begin{array}{ll} \text{minimize} & \mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \cdots, f_m(\mathbf{x}))^T \\ \text{subject to} & \mathbf{x} \in \Omega \end{array} \tag{1}$$

where $\Omega = \prod_{i=1}^{n}[a_i, b_i] \subseteq \mathbb{R}^n$ is the decision (variable) space, $\mathbf{x} = (x_1, \ldots, x_n)^T \in \Omega$ is a candidate solution. $\mathbf{F} : \Omega \rightarrow \mathbb{R}^m$ constitutes m conflicting objective functions, and $\mathbb{R}^m$ is called the objective space. The attainable objective set is defined as the set $\Theta = \{\mathbf{F}(\mathbf{x}) | \mathbf{x} \in \Omega\}$.

$\mathbf{x}^1$ is said to dominate $\mathbf{x}^2$ (denoted as $\mathbf{x}^1 \preceq \mathbf{x}^2$) if and only if $f_i(\mathbf{x}^1) \leq f_i(\mathbf{x}^2)$ for every $i \in \{1, \cdots, m\}$ and $f_j(\mathbf{x}^1) < f_j(\mathbf{x}^2)$ for at least one index $j \in \{1, \cdots, m\}$. A solution $\mathbf{x}^*$ is Pareto-optimal to (1) if there is no other solution $\mathbf{x} \in \Omega$ such that $\mathbf{x} \preceq \mathbf{x}^*$. $\mathbf{F}(\mathbf{x}^*)$ is then called a Pareto-optimal (objective) vector. In other words, any improvement of a Pareto-optimal vector in one objective must lead to deterioration in at least one other objective. The set of all Pareto-optimal solutions is called the Pareto-optimal set (PS).

Accordingly, the set of all Pareto-optimal vectors, $PF = \{\mathbf{F}(\mathbf{x}) \in \mathbb{R}^m | \mathbf{x} \in PS\}$, is called the Pareto-optimal front (PF) [1].

Due to the use of population, evolutionary algorithms (EAs) are able to approximate the PF in a single run [2]. They are widely recognized as a main approach for MOPs. Over the past three decades, much effort has been devoted to developing evolutionary multiobjective optimization (EMO) algorithms to tackle MOPs. According to selection strategies used, the existing EMO algorithms can be divided into three categories: Pareto-based methods, e.g., NSGA-II [3], indicator-based methods, e.g., IBEA [4] and decomposition-based methods, e.g., MOEA/D [5]. Among them, the Pareto-based method, which employs Pareto dominance relation as the major criterion to compare solutions, is the origin of modern EMO techniques and it has been widely studied [6]. Non-dominated sorting, which divides solutions into different non-domination levels according to their dominance relationships, is a basic operation in most Pareto-based methods. Without loss of generality, we assume that solutions in a population P can be divided into l $(l \geq 1)$ non-domination levels, denoted as $F_1, F_2, \cdots, F_l$. According to the principle non-dominated sorting, all non-dominated solutions are first assigned to F_1. Afterwards, solutions assigned to F_1 are temporarily removed from P and the non-dominated solutions in $P - F_1$ are assigned to F_2, and so on. It is worth noting that each solution in F_j should dominate at least one solution in F_k, where $j < k$ and $k, j \in \{1, \cdots, l\}$. Fig. 1 provides a simple example of three non-domination levels formed by a population of 9 solutions.

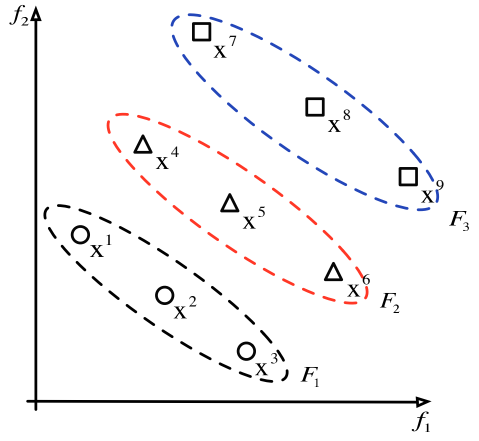


Figure 1: An illustration of non-domination levels of a population with 9 solutions.

Nevertheless, with the increase of number of objectives and population size, the computational costs of non-dominated sorting become increasingly heavy. The first non-dominated sorting procedure proposed in [2] has a time complexity of $O(mN^3)$, where N is the population size. Later, a fast non-dominated sorting procedure was developed in [3], where the time complexity is reduced to $O(mN^2)$. By using a divide-and-conquer strategy, [7] presented a non-dominated sorting procedure with a time complexity of $O(Nlog^{m-1}N)$. By inferring some dominance relationships based on the recorded comparisons, [8] suggested two approaches, called climbing sort and deductive sort, to improve the computational efficiency of the non-dominated sorting procedure. Although these two approaches hold the same worst-case complexity of $O(mN^2)$ as the fast non-dominated sorting procedure, empirical studies have shown that both of them outperform the fast non-dominated sorting procedure in terms of both CPU time and number of dominance comparisons.

Different from the generational EMO algorithm, in which all offspring solutions are generated before comparing with their parents, a steady-state EMO algorithm updates the parent population once a new candidate solution has been generated. As discussed in [9], the steady-state scheme possesses several advantages over the generational scheme, such as the better chances for creating promising offspring, applicability for parallel implementation in distributed computing environments and feasibility for the-

Algorithm 1: Steady-state NSGA-II

```
  Input: algorithm parameters
  Output: population $P$
1 Initialize a population $P \leftarrow \{\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^N\}$;
2 while termination criterion is not met do
3     Mating selection and generate an offspring $\mathbf{x}^c$;
4     Use non-dominated sorting to divide $P' \leftarrow P \bigcup \{\mathbf{x}^c\}$ into several non-domination levels
      $F_1, F_2, \cdots, F_l$;
5     Identify the worst solution $\mathbf{x}' \in F_l$ and set $P \leftarrow P' \setminus \{\mathbf{x}'\}$;
6 end
7 return $P$;
```

oretical analysis. The most famous steady-state EMO algorithms are ϵ-MOEA [9] and SMS-EMOA [10]. The former one is computationally efficient since it does not need non-dominated sorting, while the latter one uses a hypervolume indicator, the calculation of which is much more computationally expensive than non-dominated sorting, in selection. Unfortunately, not many other steady-state EMO algorithms have been proposed and studied in the literature. One major reason is the computational burden caused by non-dominated sorting. In fact, before generating a new candidate solution, the non-domination level structure of the parent population is already known from the last update procedure. Introducing a new candidate solution usually does not refresh the entire non-domination level structure. Instead, only a few population members need to change their non-domination levels. Therefore, it is unnecessary to conduct the non-dominated sorting from scratch each time. Furthermore, the solution, which should change its non-domination level, only need to move forward or backward one non-domination level. Considering these two properties, this paper proposes an efficient non-domination level update (ENLU) approach for steady-state EMO algorithms. Before the update procedure, ENLU approach locates the non-domination level which the new offspring belongs to. Then, it identifies the solutions that need to change their non-domination levels and move them backward to the next non-domination levels. After the update procedure, where an inferior solution has been removed from the parent population, ENLU approach identifies those solutions that need to change their non-domination levels and move them forward to the prior non-domination levels. Theoretical analysis show that the time complexity of ENLU approach is $O(m)$ in the best case and $O(mN\sqrt{N})$ in the worst case, which are better than the fast non-dominated sorting procedure. Extensive experiments confirm the computational efficiency achieved by the proposed ENLU approach, which avoid a significant number of unnecessary dominance comparisons.

In the rest of this paper, we first discuss the motivations of this work in Section 2. Then, the implementations of our proposed ENLU approach are described step by step in Section 3. Afterwards, the time complexity of ENLU approach is theoretically analyzed in Section 4. Next, Section 5 empirically investigates performance of ENLU approach on several synthetic test cases. Finally, Section 6 concludes this paper and provides some future directions.

2 Motivations

Without loss of generality, here we use NSGA-II as a baseline to illustrate the principle of steady-state scheme. Algorithm 1 presents the pseudo-code of steady-state NSGA-II. First of all, a population $P = \{\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^N\}$ is initialized via uniform sampling over Ω (line 1). During the main while loop, P is updated as soon as the generation of each offspring $\mathbf{x}^c$. In particular, the update procedure involves two steps: one is the use of non-dominated sorting to divide the hybrid population P', formed by the combination of P and $\mathbf{x}^c$, into several non-domination levels, $F_1, F_2, \cdots, F_l$ (line 4); the other is the elimination of the worst solution $\mathbf{x}' \in F_l$, in terms of crowding distance, from P' to form a new P (line 5). As

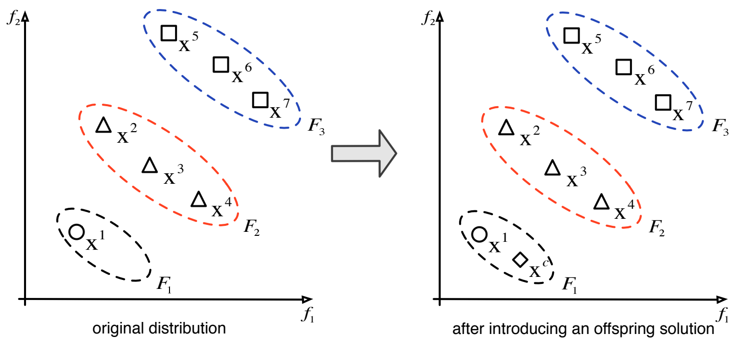


Figure 2: None of the solutions in P to change their non-domination levels after introducing $\mathbf{x}^c$.

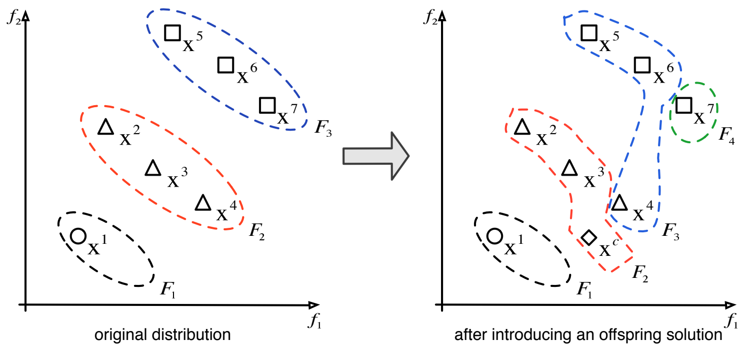


Figure 3: Only $\mathbf{x}^4$ and $\mathbf{x}^7$ move to the next non-domination level after introducing $\mathbf{x}^c$.

the non-dominated sorting requires pair-wise dominance comparisons among solutions, most of the computational costs of NSGA-II come from this operation. For the original NSGA-II, a generational scheme, each generation at most costs $2N \times (2N-1) = O(N^2)$ dominance comparisons in non-dominated sorting, while for the steady-state NSGA-II, generating N offspring at most costs $N^2 \times (N+1) = O(N^3)$ dominance comparisons. With the increase of number of objectives and population size, non-dominated sorting definitely aggravates the computational efficiency of steady-state NSGA-II. However, in case the non-domination level structure of a population has been obtained after the first non-dominated sorting, it can be observed that adding or eliminating a solution usually does not refresh the entire non-domination level structure. On the contrary, only some population members need to change their non-domination levels. Therefore, one question naturally comes to our mind:

- Is it really necessary to conduct non-dominated sorting from scratch, each time, in the update procedure of steady-state NSGA-II?

Let us consider a simple example presented in Fig. 2. There are seven solutions in P, which can be divided into three non-domination levels, i.e., $F_1 = \{\mathbf{x}^1\}, F_2 = \{\mathbf{x}^2, \mathbf{x}^3, \mathbf{x}^4\}, F_3 = \{\mathbf{x}^5, \mathbf{x}^6, \mathbf{x}^7\}$. After introducing $\mathbf{x}^c$, none of the solutions in P need to change their non-domination levels and we only need to insert $\mathbf{x}^c$ into F_1. For another example in Fig. 3, when introducing $\mathbf{x}^c$ to P, only $\mathbf{x}^4$ and $\mathbf{x}^7$ need to move themselves backward to the next non-domination levels. Analogously, when eliminating a solution from P' after the update procedure, the non-domination level structure of the newly formed P might be different from that of P'. For simplicity, let us consider the same examples discussed in Fig. 2 and Fig. 3, and we assume that just $\mathbf{x}^c$ is eliminated from P' after the update procedure. For the example in Fig. 2, none of the remaining solutions need to change their non-domination levels, while for the example in Fig. 3, only $\mathbf{x}^4$ and $\mathbf{x}^7$ need to move themselves forward to the prior non-domination levels.

Based on the above discussions, we confirm that it is unnecessary to conduct the non-dominated sorting from scratch during the update procedure of steady-state NSGA-II. Instead, we only need to figure out the following three questions, after generating an offspring $\mathbf{x}^c$.

- Which non-domination level $\mathbf{x}^c$ belongs to?
- Is there any solution in P needs to change its non-domination level?
- If yes, what is the new non-domination level such solution belongs to?

In addition, when removing a solution from P' after the update, we need to figure out the following two questions:

- Is there any solution in the newly formed P needs to change its non-domination level?
- If yes, what is the new non-domination level such solution belongs to?

In the following paragraphs, we will present an efficient non-domination level update approach to address these above considerations.

3 An Efficient Non-domination Level Update Framework

Instead of conducting the non-dominated sorting from scratch, ENLU takes advantage of the existing knowledge of the non-domination level structure to update the solutions that are required to change their non-domination levels. As discussed in Section 2, the non-domination level structure might be changed both when introducing a new candidate solution after the offspring reproduction and eliminating an inferior solution after the update procedure. In the following paragraphs, we will describe the technical details of these two circumstances, respectively, step by step.

3.1 Change of Non-domination Levels When Introducing An Offspring Solution

According to the discussion in Section 2, we have to figure out the following three issues.

3.1.1 Which non-domination level $\mathbf{x}^c$ belongs to

Here we suggest a top-down approach to find out the non-domination level that $\mathbf{x}^c$ belongs to. To be specific, starting from F_1, we check the dominance relation between $\mathbf{x}^c$ with all solutions in F_1, one by one. If $\mathbf{x}^c$ is non-dominated with all solutions in F_1 or it dominates some solutions in F_1, $\mathbf{x}^c$ is added into F_1. Otherwise, $\mathbf{x}^c$ does not belong to F_1 in case it is dominated by a solution in F_1. As soon as such dominating solution is found, we do not compare the dominance relation with the remaining solutions in F_1 any longer and turn to investigate the solutions in F_2. Analogously, $\mathbf{x}^c$ is added into F_2 in case it is non-dominated with all solutions in F_2 or it dominates some solutions in F_2. Otherwise, we turn to F_3 for investigation, so on and so forth. Note that if $\mathbf{x}^c$ does not belong to any existing non-domination level, $\mathbf{x}^c$ is added into a newly created non-domination level F_{l+1}.

3.1.2 Is there any solution needs to change its non-domination level

Note that all solutions in the same non-domination level are non-dominated with each other. Let us assume that $\mathbf{x}^t$ is going to be added into F_i, where $i \in \{1, 2, \cdots, l\}$. Based on the above discussion, the solution, in F_i, dominated by $\mathbf{x}^t$ needs to move to another non-domination level.

Algorithm 2: ENLU after offspring generation

```
Input: non-domination level structure F=$\{F_1, F_2, \cdots, F_l\}$, offspring solution $\mathbf{x}^c$
Output: updated non-domination level structure F
$flag1 \leftarrow flag2 \leftarrow flag3 \leftarrow 0$;
for $i \leftarrow 1$ to $l$ do
    $S \leftarrow \emptyset$;
    for $j \leftarrow 1$ to $|F_i|$ do
        if $\mathbf{x}^c$ is non-dominated with $F_i(j)$ then
            $flag1 \leftarrow 1$;
        else if $\mathbf{x}^c$ is dominated by $F_i(j)$ then
            $flag2 \leftarrow 1$;
            break;
        else
            $flag3 \leftarrow 1$;
            $S \leftarrow S \bigcup \{F_i(j)\}$;
        end
    end
    if $flag2 = 1$ then
        break;
    else if $flag1 = 0$ and $flag3 = 1$ then
        Move all solutions originally in $F_i$ to $F_{i+1}$;
        break;
    else
        Update($S, i+1$);
        break;
    end
end
return F
```

3.1.3 What is the new non-domination level such solution belongs to

Let us assume that $\mathbf{x}'$ originally belongs to F_i, where $i \in \{1, 2, \cdots, l\}$, and $\mathbf{x}^t \preceq \mathbf{x}'$. In this case, adding $\mathbf{x}^t$ into F_i will incurs the movement of $\mathbf{x}'$ from F_i to F_{i+1}. According to the property of non-domination level, none of the solutions in F_j can dominate any solution in F_i, and each solution in F_i dominates at least one solution in F_j, where $i < j$ and $i, j \in \{1, 2, \cdots, l\}$. Since $\mathbf{x}'$ originally belongs to F_i, it dominates at least one solution in F_{i+1}. If we move $\mathbf{x}'$ to F_k, where $k > i+1$, it contradicts the property of non-domination level. Consequently, $\mathbf{x}'$ can only be moved from F_i to F_{i+1}.

Based on the above discussions, Algorithm 2 presents the pseudo-code of the non-domination level update procedure after the generation of $\mathbf{x}^c$. Note that the non-domination level structure of P is assumed to be known after using non-dominated sorting during the initialization procedure. To start with, the algorithm first checks whether there exists a solution in F_1 that dominates $\mathbf{x}^c$. If such solution exists, we start comparing $\mathbf{x}^c$ with the solutions in F_2, so on and so forth. In summary, we might meet one of the following four cases when checking with the solutions in $F_i (1 \leq i \leq l)$.

1. $\mathbf{x}^c$ is dominated by a solution in F_i. We stop checking with the remaining solutions in F_i, and move to check with solutions in F_{i+1}, in case $1 \leq i < l$. Otherwise, i.e., when $i = l$, $\mathbf{x}^c$ is directly assigned to a newly created non-domination level F_{l+1}.

2. $\mathbf{x}^c$ is non-dominated with all solutions in F_i, then $\mathbf{x}^c$ is assigned to F_i. Fig. 4 presents a simple example to illustrate this scenario. $\mathbf{x}^c$ is dominated by $\mathbf{x}^2$ and $\mathbf{x}^3$ in F_1, thus it does not belong to F_1 and we

Algorithm 3: Update(S, i)

```
Input: solution set $S$, non-domination level index $i$
Output: updated non-domination structure F
if $i = l + 1$ then
    Move all solutions in $S$ to $F_i$;
else
    Move all solutions in $S$ to $F_i$;
    $T \leftarrow \emptyset$;
    for $i \leftarrow 1$ to $|S|$ do
        for $j \leftarrow 1$ to $|F_i|$ do
            if $S(i) \preceq F_i(j)$ then
                $T \leftarrow T \bigcup \{F_i(j)\}$;
            end
        end
    end
    if $T \neq \emptyset$ then
        Update($T, i + 1$);
    end
end
return F
```

move to check with solutions in F_2. Since $\mathbf{x}^c$ is non-dominated with all solutions in F_2, it should be added into F_2. In this case, we can stop checking with the remaining solutions, and none of the solutions in the population need to update their non-domination levels.

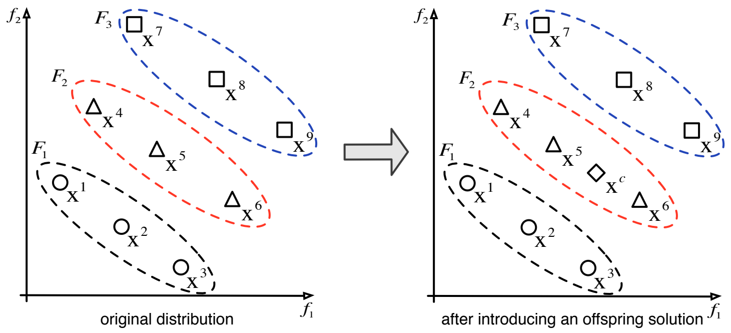


Figure 4: A simple illustration of case 2. $F_1 = \{\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3\}$, $F_2 = \{\mathbf{x}^4, \mathbf{x}^5, \mathbf{x}^6\}$ and $F_3 = \{\mathbf{x}^7, \mathbf{x}^8, \mathbf{x}^9\}$. Since $\mathbf{x}^c$ is dominated by $\mathbf{x}^2$ and $\mathbf{x}^3$, it does not belong to F_1. On the other hand, $\mathbf{x}^c$ is non-dominated with all solutions in F_2, it is assigned to F_2. None of the other solutions need to update the non-domination levels.

3. $\mathbf{x}^c$ dominates all solutions in F_i, then the non-domination level of each solution in F_k, where $k \in \{i, i+1, \cdots, l\}$, is increased by one, and $\mathbf{x}^c$ is added into F_i. Fig. 5 presents a simple example to illustrate this scenario. $\mathbf{x}^c$ is dominated by $\mathbf{x}^2$ and $\mathbf{x}^3$ in F_1, thus we move to check with solutions in F_2. Since $\mathbf{x}^c$ dominates all solutions in F_2, it is assigned to this non-domination level. In addition, solutions originally belonged to F_2 and F_3 are, respectively, moved to F_3 and F_4.

4. $\mathbf{x}^c$ dominates some solutions in F_i, but it is non-dominated with the remaining ones there, then $\mathbf{x}^c$ is added into F_i. We store the solutions originally in F_i and are dominated by $\mathbf{x}^c$ into a temporary archive S, and move them to F_{i+1}. Then, we find out the solutions in F_{i+1} that are dominated by solutions in S. If there does not exist such solutions, no more operation is required. Otherwise,

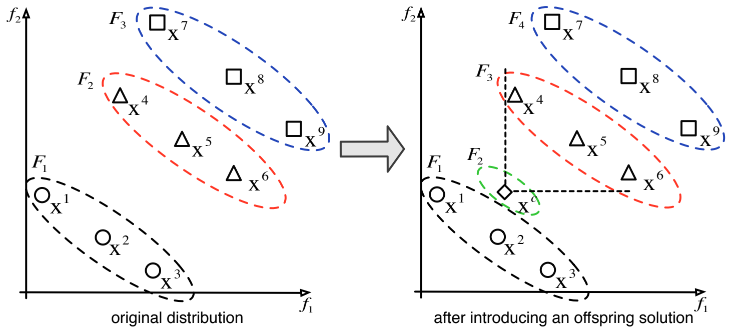


Figure 5: A simple illustration of case 3. $F_1 = \{\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3\}$, $F_2 = \{\mathbf{x}^4, \mathbf{x}^5, \mathbf{x}^6\}$ and $F_3 = \{\mathbf{x}^7, \mathbf{x}^8, \mathbf{x}^9\}$. Since $\mathbf{x}^c$ is dominated by $\mathbf{x}^2$ and $\mathbf{x}^3$, it does not belong to F_1. Since $\mathbf{x}^c$ dominates all solutions in F_2, it is assigned to F_2. In addition, $\mathbf{x}^4$, $\mathbf{x}^5$ and $\mathbf{x}^6$ are moved to F_3, while $\mathbf{x}^7$, $\mathbf{x}^8$ and $\mathbf{x}^9$ are moved to a newly created non-domination level F_4.

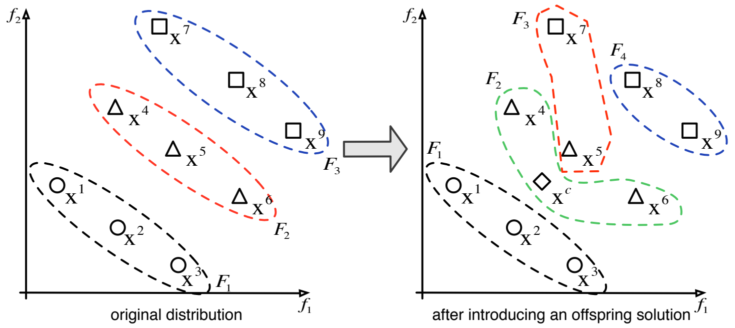


Figure 6: A simple illustration of case 4. $F_1 = \{\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3\}$, $F_2 = \{\mathbf{x}^4, \mathbf{x}^5, \mathbf{x}^6\}$, $F_3 = \{\mathbf{x}^7, \mathbf{x}^8, \mathbf{x}^9\}$. Since $\mathbf{x}^c$ is dominated by $\mathbf{x}^2$ and $\mathbf{x}^3$, it does not belong to F_1. On the other hand, $\mathbf{x}^c$ dominates $\mathbf{x}^5$ and is non-dominated with $\mathbf{x}^4$ and $\mathbf{x}^6$. Thus, $\mathbf{x}^c$ is assigned to F_2 and $\mathbf{x}^5$ is moved to F_3. Moreover, since $\mathbf{x}^5$ dominates $\mathbf{x}^8$ and $\mathbf{x}^9$ and is non-dominated with $\mathbf{x}^7$, $\mathbf{x}^8$ and $\mathbf{x}^9$ are moved to a newly created non-domination level F_4.

similar operations are conducted to move the dominated solutions to the next non-domination level, so on and so forth. Note that if $i = l$, solutions in S are assigned to a newly created non-domination level F_{l+1}. Fig. 6 presents a simple example to illustrate this scenario. $\mathbf{x}^c$ is dominated by $\mathbf{x}^2$ and $\mathbf{x}^3$ in F_1, thus we move to check with solutions in F_2. Since $\mathbf{x}^c \preceq \mathbf{x}^5$, $\mathbf{x}^c$ is assigned to F_2 and $\mathbf{x}^5$ is moved to check with solutions in F_3. As $\mathbf{x}^5 \preceq \mathbf{x}^8$ and $\mathbf{x}^5 \preceq \mathbf{x}^9$, $\mathbf{x}^5$ is assigned to F_3 and $\mathbf{x}^8$, $\mathbf{x}^9$ are moved to a newly created non-domination level F_4.

3.2 Change of Non-domination Levels After Update Porcedure

According to the discussion in Section 2, we have to figure out the following two issues.

3.2.1 Is there any solution needs to change its non-domination level

Let us assume that the solution, denoted as $\mathbf{x}^e$, eliminated by the update procedure, belongs to F_i, where $i \in \{1, 2, \cdots, l\}$. The elimination of $\mathbf{x}^e$ cannot influence the assignment of non-domination levels of solutions in F_j, where $1 \leq j \leq i$. This is because such solutions either dominate $\mathbf{x}^e$ or are non-dominated with $\mathbf{x}^e$, while only those solutions, dominated by $\mathbf{x}^e$, might change their non-domination levels.

Algorithm 4: ENLU after update procedure

```
Input: non-domination level structure F={$F_1, F_2, \cdots, F_l$}, solution $\mathbf{x}^e$
Output: updated non-domination structure F
1  Locate the non-domination level $F_i$ that $\mathbf{x}^e$ belongs to;
2  while $i < l$ do
3      $S \leftarrow \emptyset$;
4      foreach $\mathbf{x} \in F_{i+1}$ do
5          if $\mathbf{x}^e \preceq \mathbf{x}$ then
6              $S \leftarrow S \bigcup \{\mathbf{x}\}$;
7          end
8      end
9      if $S \neq \emptyset$ then
10         foreach $\mathbf{x}' \in S$ do
11             $flag \leftarrow 0$;
12             foreach $\mathbf{x}'' \in F_i$ do
13                 if $\mathbf{x}'' \preceq \mathbf{x}'$ then
14                     $flag \leftarrow 1$;
15                     break;
16                 end
17             end
18             if $flag = 0$ then
19                 $F_{i+1} \leftarrow F_{i+1} \backslash \{\mathbf{x}'\}$;
20                 $F_i \leftarrow F_i \bigcup \{\mathbf{x}'\}$;
21             end
22         end
23     else
24         break;
25     end
26     $i$++;
27 end
28 return F
```

3.2.2 What is the new non-domination level such solution belongs to

Similar to the discussion in Section 3.1, a solution can at most move forward one non-domination level. According to the property of non-domination level, $\exists \mathbf{x}^* \in F_{i+1}$ and $\exists \mathbf{x}' \in F_{i-1}$ that $\mathbf{x}^e \preceq \mathbf{x}^*$ and $\mathbf{x}' \preceq \mathbf{x}^e$. According to the transitivity of Pareto dominance relation, it can be inferred that $\mathbf{x}' \preceq \mathbf{x}^*$. Therefore, the elimination of $\mathbf{x}^e$ might result in the movement of $\mathbf{x}^*$ to F_i, in case $\nexists \mathbf{x}'' \in F_i$ that $\mathbf{x}'' \preceq \mathbf{x}^*$, but not F_{i-1}.

Algorithm 4 presents the pseudo-code of the non-domination level update procedure when eliminating an inferior solution from P' after the update procedure. First of all, we locate the non-domination level F_i that $\mathbf{x}^e$ belongs to. Starting from F_{i+1}, we find out solutions dominated by $\mathbf{x}^e$. If there does not exist such solutions, the non-domination level update procedure terminates and no solution needs to change its non-domination level. Otherwise, we store the dominated solutions in a temporary archive S. For each solution $\mathbf{x}$ in S, we check its dominance relation with solutions in F_i. If $\nexists \mathbf{x}'' \in F_i$ that $\mathbf{x}'' \preceq \mathbf{x}$, $\mathbf{x}$ is added into F_i. Otherwise, the non-domination level of $\mathbf{x}$ does not need to change. If none of the solutions in S need to change their non-domination levels, it means that the elimination of $\mathbf{x}^e$ does not change the non-domination levels of the other solutions. Otherwise, we use a similar method to investigate the movement of solutions dominated by the ones in S. Note that if $\mathbf{x}^e \in F_l$, no more operation is required.

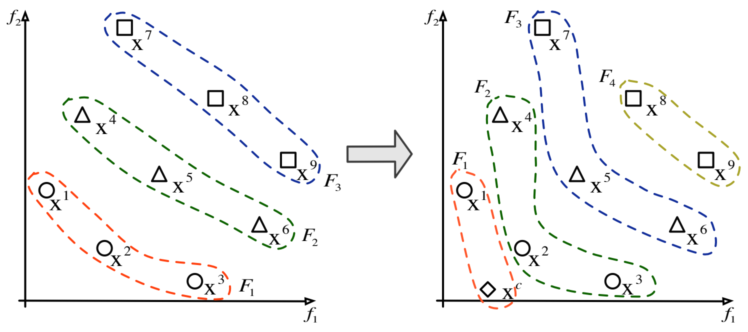


Figure 7: An example of worst case complexity.

4 Computational Complexity Analysis

In this section, we briefly analyze the computational complexity of the proposed ENLU approach. Let us first consider the worst case complexity. As a prerequisite, the non-domination level structure of the parent population P has already been known. When introducing a newly generated offspring solution $\mathbf{x}^c$ to P, the worst case happens when each solution in F_i $(i > 1)$ is dominated by only one solution in F_{i-1}. Fig. 7 presents a simple example to illustrate this scenario, e.g., $\mathbf{x}^6 \in F_2$ is merely dominated by $\mathbf{x}^3 \in F_1$. After generating an offspring solution $\mathbf{x}^c$, denoted as a pentagram, in F_1, $\mathbf{x}^c \preceq \mathbf{x}^2$ and $\mathbf{x}^3$, and is non-dominated with $\mathbf{x}^1$. According to ENLU approach introduced in Section 3.1, $\mathbf{x}^c$ is added into F_1, and $\mathbf{x}^2$, $\mathbf{x}^3$ should be moved from F_1 to F_2. Now, we cost 3 dominance comparisons. Then, since, in F_2, $\mathbf{x}^2 \preceq \mathbf{x}^5$ and $\mathbf{x}^3 \preceq \mathbf{x}^6$, $\mathbf{x}^5$ and $\mathbf{x}^6$ should be moved from F_2 to F_3. These operations cost $2 \times 3 = 6$ dominance comparisons. Analogously, as $\mathbf{x}^5$ and $\mathbf{x}^6$, respectively, dominates $\mathbf{x}^8$ and $\mathbf{x}^9$ in F_3, $\mathbf{x}^8$ and $\mathbf{x}^9$ should be moved from F_3 to F_4, which is a newly created non-domination level. These operations cost $2 \times 3 = 6$ dominance comparisons. In total, for the example shown in Fig. 7, the ENLU approach costs $3 + 6 + 6 = 15$ dominance comparisons.

More generally, suppose P contains N solutions, which form $\lceil\sqrt{N}\rceil$ non-domination levels. Each non-domination level roughly has $\lceil\sqrt{N}\rceil$ solutions. Suppose the newly generated offspring solution $\mathbf{x}^c$ dominates $\lceil\sqrt{N}\rceil - 1$ solutions in F_1. ENLU approach makes all those $\lceil\sqrt{N}\rceil - 1$ dominated solutions move from F_1 to F_2. Since each of these $\lceil\sqrt{N}\rceil - 1$ solutions dominates one solution in F_2, there are $\lceil\sqrt{N}\rceil - 1$ solutions in F_2 that should be moved from F_2 to F_3, so on and so forth. In total, the ENLU approach costs $N\lceil\sqrt{N}\rceil - 2N + 2\lceil\sqrt{N}\rceil$ [1] dominance comparisons. Considering m objective functions, the worst case complexity of ENLU approach when introducing a new offspring solution is

$$m \cdot (N\lceil\sqrt{N}\rceil - 2N + 2\lceil\sqrt{N}\rceil) = O(mN\lceil\sqrt{N}\rceil) \tag{2}$$

After eliminating an inferior solution from $P' = P\bigcup\{\mathbf{x}^c\}$ in the update procedure, the non-domination level structure of the newly formed P also might be updated. Considering the same example discussed above, the worst case happens when $\mathbf{x}^c$ is removed from P'. Starting from F_2, we find out $\mathbf{x}^2$, $\mathbf{x}^3$ and $\mathbf{x}^4$ are dominated by $\mathbf{x}^c$. This indicates that they have potential to move from F_2 to F_1. By comparing the Pareto dominance relation between $\mathbf{x}^1$ and all these three solutions, only $\mathbf{x}^2$ and $\mathbf{x}^3$ need to update their non-domination levels. These operations cost $3 + 3 = 6$ dominance comparisons. Afterwards, we

[1]

$$\begin{aligned}
&\underbrace{\lceil\sqrt{N}\rceil + \lceil\sqrt{N}\rceil \times (\lceil\sqrt{N}\rceil - 1) + \cdots + \lceil\sqrt{N}\rceil \times (\lceil\sqrt{N}\rceil - 1)}_{\lceil\sqrt{N}\rceil} \\
&= \lceil\sqrt{N}\rceil + \lceil\sqrt{N}\rceil \times (\lceil\sqrt{N}\rceil - 1) \times (\lceil\sqrt{N}\rceil - 1) \\
&= N\lceil\sqrt{N}\rceil - 2N + 2\lceil\sqrt{N}\rceil
\end{aligned}$$

find out $\mathbf{x}^5$, $\mathbf{x}^6$ and $\mathbf{x}^7$, in F_3, are dominated by $\mathbf{x}^2$ and $\mathbf{x}^3$. Similar dominance comparisons are conducted to check whether these three solutions can be moved from F_3 to F_2. This procedure costs $2 \times 3 + 3 = 9$ dominance comparisons. Analogously, the movement of $\mathbf{x}^8$ and $\mathbf{x}^9$ from F_4 to F_3 costs $2 \times 3 + 3 = 9$ dominance comparisons. In total, after removing $\mathbf{x}^c$ from P', ENLU approach costs $6 + 9 + 9 = 24$ dominance comparisons to update the non-domination level structure of P.

More generally, suppose P' conotains N solutions, which form $\lceil \sqrt{N} \rceil$ non-domination levels. Each non-domination level roughly has $\lceil \sqrt{N} \rceil$ solutions. The worst case happens when eliminating a solution from F_1 and its elimination incurs the movement of $\lceil \sqrt{N} \rceil - 1$ solutions from F_2 to F_1. These operations cost $\lceil \sqrt{N} \rceil \times (\lceil \sqrt{N} \rceil - 1)$ dominance comparisons. Analogously, from F_2 to $F_{\sqrt{N}}$, each step at most costs $\lceil \sqrt{N} \rceil \times (\lceil \sqrt{N} \rceil - 1) + \sqrt{N}$ dominance comparisons. Thus, in total, the ENLU approach costs $N\lceil \sqrt{N} \rceil - N$ [2] dominance comparisons. Considering m objective functions, the worst case complexity of ENU mechanism when removing a solution after the update procedure is

$$m(N\lceil \sqrt{N} \rceil - N) = O(mN\sqrt{N}) \tag{3}$$

In summary, the worst case complexity of ENLU approach is $O(mN\sqrt{N}) + O(mN\sqrt{N}) = O(mN\sqrt{N})$. In contrast, the computational complexity of fast non-dominated sorting is $O(mN^2)$. As for the example discussed in Fig. 7, the ENU mechanism totally costs $15 + 24 = 39$ dominance comparisons, while the fast non-dominated sorting procedure should spend 100 dominance comparisons.

On the other hand, the best case complexity analysis is relatively simple. When introducing a newly generated offspring solution $\mathbf{x}^c$ to P, the best case happens when F_1 only contains a single solution, and it is non-dominated with $\mathbf{x}^c$. In this case, according to the ENLU approach, it only requires one dominance comparion. As for the update of non-domination level structure after eliminating an inferior solution from P' in the update procedure, the best case happens when this eliminated solution is in the last non-domination level. In this case, no dominance comparion is required. In summary, considering m objective funcitons, the best case complexity of ENLU approach is $O(m)$.

5 Empirical Results and Discussions

In this section, we empirically compare the computational efficiency of ENLU approach with four popular non-dominated sorting algorithms, namely, fast non-dominated sort [3], deductive sort [8], two variants of efficient non-dominated sort, i.e., ENS-SS and ENS-BS [11], for two different scenarios. In the first scenario, two different indicators, i.e., the number of dominance comparisons and the amount of CPU time, are employed to evaluate the computational efficiencies of these approaches on several synthetic data sets. In the second scenario, we compare these approaches when they are embedded in a steady-state NSGA-II for several benchmark problems. All simulation results reported in this paper are conducted on a Macintosh with a 2.30GHz Intel Core i7 CPU and 16GB RAM.

First of all, we introduce the two different kinds of synthetic data sets used in the empirical studies.

1. *Cloud Data Set*: In this data set, the objective values of each solution are randomly sampled from a uniform distribution within the interval $[0, 1]$. This randomly sampled population is unstructured, and it consists of solutions arranged in a random order. Moreover, the randomly sampled population contains a varying number of non-domination levels, and each solution dominates a random

[2]
$$\begin{aligned}
&\underbrace{(\lceil \sqrt{N} \rceil \times (\lceil \sqrt{N} \rceil - 1) + \lceil \sqrt{N} \rceil) + (\lceil \sqrt{N} \rceil \times (\lceil \sqrt{N} \rceil - 1) + \lceil \sqrt{N} \rceil) + \cdots + (\lceil \sqrt{N} \rceil \times (\lceil \sqrt{N} \rceil - 1) + \lceil \sqrt{N} \rceil)}_{\lceil \sqrt{N} - 1 \rceil} \\
&= (\lceil \sqrt{N} \rceil - 1) \times N \\
&= N\lceil \sqrt{N} \rceil - N
\end{aligned}$$

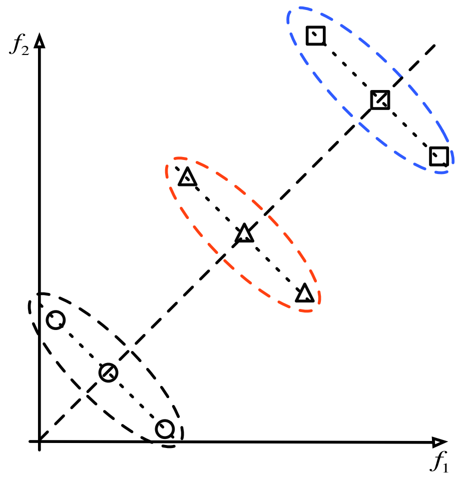


Figure 8: A simple example of a population containing three discrete non-domination levels in a two-dimensional space.

number of solutions in the population. The cloud data set intends to mimic the population structure in the early stages of evolution, and it investigates the general ability to identify the non-domination levels in a mixed population.

2. *Fixed Fronts Data Set*: This data set specifies a population with a controllable number of non-domination levels. In particular, solutions are divided into a fixed number of non-domination levels with almost the same size. And each solution in a given non-domination level dominates all solutions in subsequent non-domination levels. Every non-domination level is distributed on a line or a hyper-plane that is orthogonal to the vector $\mathbf{1} = (1, 1, \cdots, 1)^T$. Fig. 8 shows a simple example that 9 solutions are sampled along three lines perpendicular to the vector $\mathbf{1} = (1, 1)^T$. This data set is able to investigate the change of computational efficiency as the number of non-domination levels diminish. Note that the number of non-domination levels diminishes with the progress of evolution. This issue will be further investigated in Section 5.3.

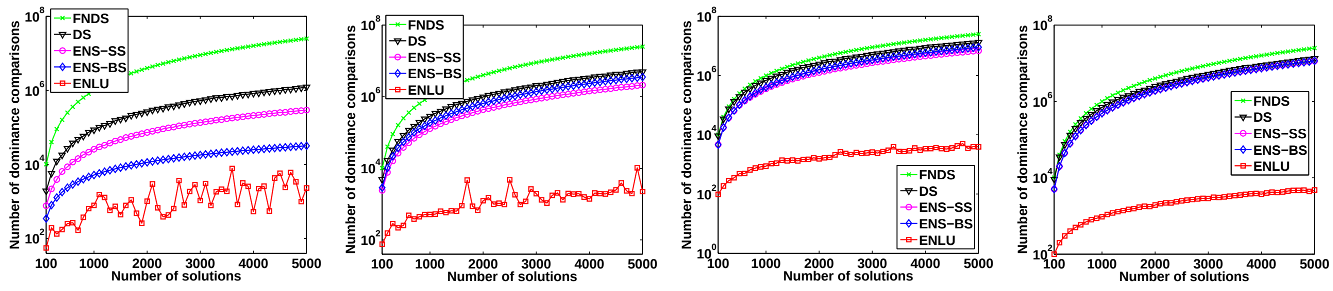
In order to mimic the steady-state scheme, a point, randomly sampled from $[0, 1]^m$, is added to the synthetic data set before conducting non-dominated sorting. In addition, each non-dominated sorting algorithm is launched 20 independent times for each data set. And only the mean indicator values are used in presentation.

5.1 Experiments on Cloud Data Set

Here we compare the computational efficiency of ENLU approach with other four non-dominated sorting algorithms on cloud data sets with various number of solutions, in two-, five-, ten- and fifteen-objective cases, respectively. In particular, the size of the data set ranges from 100 to 5,000 with an increment of 100. Consequently, for a given number of objectives, there are in total 50 randomly generated populations used in the empirical studies.

Fig. 9 plots the result of numbers of dominance comparisons with various number of solutions. For clarity, the Y-axes of all figures are labeled in log-scale, since the fast non-dominated sort costs much more computational resources than the others. As expected, fast non-dominated sort (denoted as FNDS) requires the most dominance comparisons. Its number of dominance comparisons increases with the growth of the data set size. However, there is little change for the trajectories of fast non-dominated sort in different number of objectives. This can be explained as the computational complexity of fast non-dominated sort mainly depends on the population size. Since deductive sort (denoted as DS) ignores some dominated solutions in sorting, it requires fewer dominance comparisons than fast non-dominated sort. As discussed in [12], in ENS-SS and ENS-BS, a solution to be assigned to a non-domination level only needs to compare with those that have already been assigned a non-domination level. Therefore, they save many unnecessary dominance comparisons in sorting. The empirical results in Fig. 9 validate this argument. Especially for 2-objective data sets, ENS-BS requires much fewer dominance comparisons than the other three non-dominated sorting algorithms. However, in 5- and 10-objective data sets, ENS-SS performs slightly better than ENS-BS. In addition, it can be observed that the amount of dominance comparisons also increases with the growth of the dimensionality of the objective space. And the performance of ENS-SS and ENS-BS become similar to deductive sort and even fast non-dominated sort in 10- and 15-objective cases. Comparing to the other four non-dominated sorting algorithms, ENLU approach shows constantly better performance. Its superiority becomes even more evident with the increase of number of objectives. It is also worth noting that the trajectories of ENLU approach fluctuate significantly for 2- and 5-objective data sets, and become stable later on. This can be explained as the non-domination level structure of a randomly generated population is rather complicated (i.e., the number of non-domination levels is huge) in case the number of objectives is small. Since the number of dominance comparisons incurred by ENLU approach largely depends on the population distribution, as discussed in Section 4, adding a new solution into the population might largely influence its non-domination level structure in the low-dimensional space. On the other hand, the number of non-domination levels diminishes with the increase of number of objectives. In this case, the non-domination level structure of the population becomes simpler, and thus makes the ENLU approach become stable. The issue of non-domination level structure will be further explored in Section 5.3.

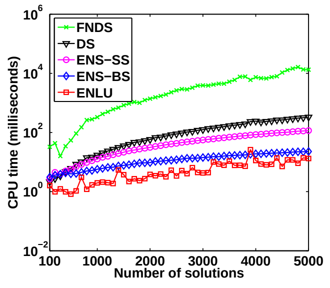
Fig. 10 presents the result of CPU time comparisons of five non-dominated sorting algorithms on cloud data sets. Consistent with the observations in Fig. 9, ENLU approach is the most efficient method to identify the non-domination level structure of the population. By contrast, fast non-dominated sort costs the most CPU time in all cloud data sets. As for the other three algorithms, the differences of their performance are relatively small.



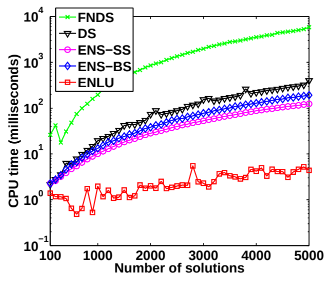
(a) 2-objective case (b) 5-objective case (c) 10-objective case (d) 15-objective case

Figure 9: Number of dominance comparisons of ENLU approach and other four non-dominated sorting methods for cloud data sets.

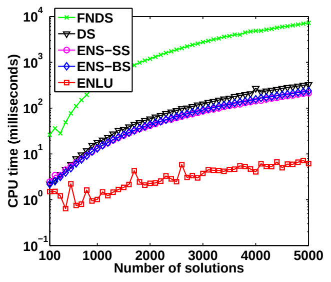
We conduct another experiment to investigate the variation of computational efficiency in different number of objectives when tackling the cloud data sets with a fixed size (here we set $N = 5,000$). Fig. 11 shows the performance comparisons of ENLU approach and the other four non-dominated sorting algorithms, regarding the dominance comparisons and CPU time. In contrast with the most inefficient fast non-dominated sort, ENLU approach requires the least dominance comparisons and CPU time in all cases. Deductive sort requires less dominance comparisons than the fast non-dominated sort in low-dimensional cases, but the number of dominance comparisons increases with the number of objectives. It is worth noting that the number of dominance comparisons required by deductive sort becomes the same as fast non-dominated sorting in case more than 15 objectives have been considered. This can be explained as the cloud data sets with more than 15 objectives have only one non-domination level, thus



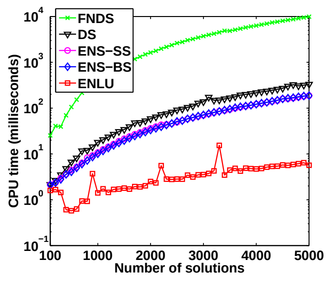
(a) 2-objective case



(b) 5-objective case



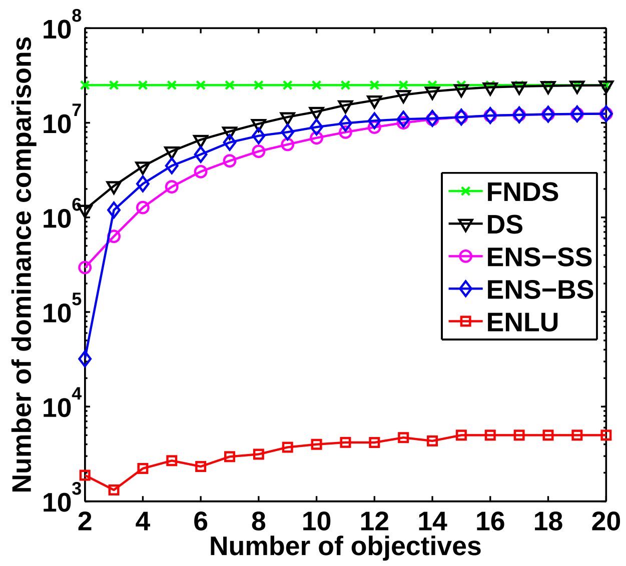
(c) 10-objective case



(d) 15-objective case

Figure 10: Comparisons of CPU time of ENLU approach and other four non-dominated sorting methods for cloud data sets.

no dominated solutions can be ignored by the deductive sort. Similar trends can be found for ENLU approach, ENS-SS and ENS-BS. This issue will be further validated in Section 5.3.



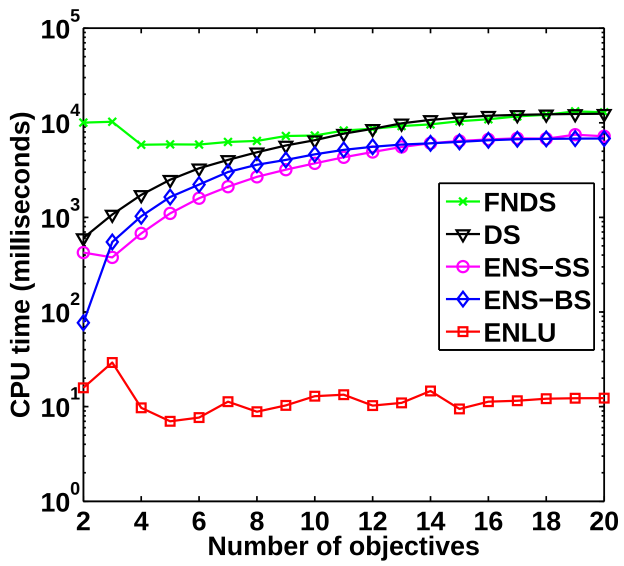


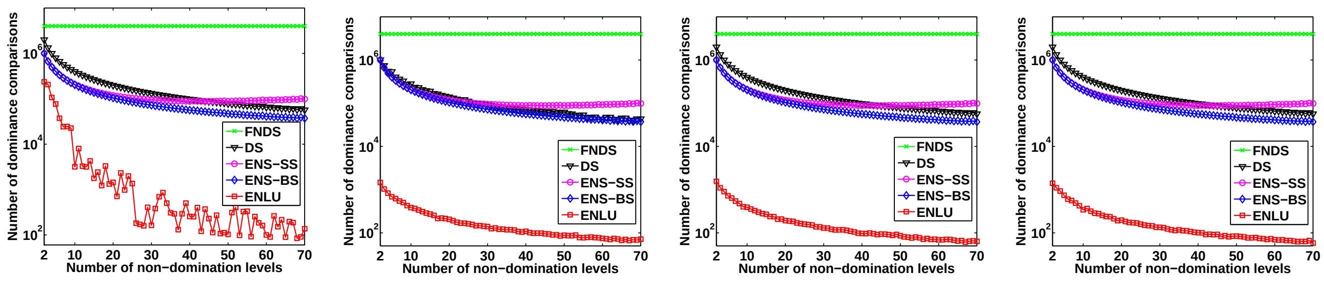
Figure 11: Results of ENLU approach and other four non-dominated sorting algorithm on 2- to 20-objective cloud data sets of 5,000 solutions.

5.2 Experiments on Fixed Fronts Data Sets

Similar to the experiments on cloud data sets, this section investigates the computational efficiency of ENLU approach on populations with fixed fronts. In particular, the following empirical studies consider two factors: one is the number of non-domination levels, the other is the number of objectives. Since the influence of the population size has been studied in Section 5.1 for cloud data sets, here the fixed fronts data sets use a constant size.

The first experiments investigate the performance of these five algorithms on fixed fronts data sets with two, five, ten and fifteen objectives. The population size is fixed to 2,000, and the number of non-domination levels varies from 2 to 70 with an increment of one, resulting in 69 populations in total for each test case. Fig. 12 and Fig. 13, respectively, presents the simulation results of ENLU approach, together with those of fast non-dominated sort, deductive sort, ENS-SS and ENS-BS, regarding number of dominance comparisons and CPU time. Similar to the observations for cloud data sets, as shown in Fig. 12, fast non-dominated sort costs the most dominance comparisons among all five algorithms. It is interesting to note that the trajectories of the fast non-dominated sort keep stable over different number of objectives. This can be explained as the number of dominance comparisons incurred by the fast non-dominated sort only depends on the population size. As discussed in [8], this number is $N^2 - N$, regardless of the existing number of non-domination levels. Comparing to the fast non-dominated sort, a significant decrease in the number of dominance comparisons have been observed by the deductive sort, ENS-SS and ENS-BS. In particular, ENS-SS performs better than deductive sort only when the number of non-domination levels is less than 40. However, since the ENS-SS has a lower space complexity of $O(1)$ than the deductive sort of $O(N)$, it consumes less CPU time than the deductive sort. In contrast, the number of dominance comparisons and CPU time cost by ENLU approach is always the least among all five algorithms. Particularly, the number of dominance comparisons cost by ENLU approach decreases with the increase of the number of non-domination levels. This is attributed to the fact that the less non-domination levels a population has, the larger chance for ENLU approach to make dominance comparisons with all solutions in a population.

In the second experiments, we investigate the performance of these five algorithms on data sets with 1 and 20 non-domination levels, respectively, in different number of objectives. This time the population size is constantly set as 5,000. Fig. 14 shows the results of ENLU approach and the other four non-dominated sorting algorithms, regarding number of dominance comparisons and CPU time. From these results we find that non-dominated sort still requires the most dominance comparisons and CPU time, while ENLU approach cost the least. ENS-SS and ENS-BS require as few dominance comparisons as ENLU approach in 2-objective case, but suddenly increase to a high level later on. The number of dominance comparisons cost by deductive sort keeps stable all the time. It is worth noting that ENS-SS, ENS-BS and deductive sort require less dominance comparisons when there are more non-domination levels in the data set.

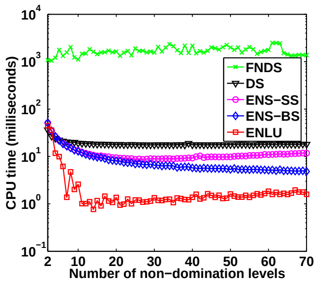


(a) 2-objective case (b) 5-objective case (c) 10-objective case (d) 15-objective case

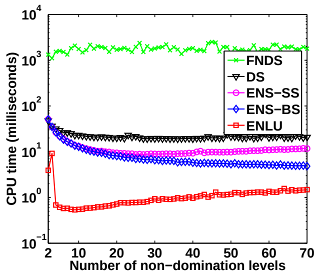
Figure 12: Number of dominance comparisons of ENLU approach and other four non-dominated sorting methods for fixed fronts data sets.

5.3 Further Investigations of Non-domination Level Structure

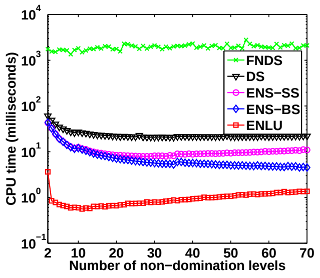
In this section, we investigate several aspects of the non-domination level structure of a population. As discussed in [13], for a randomly generated population with a given size, the number of non-domination levels significantly decreases with the increase of the number of objectives. In order to validate this consideration, we conduct an experiment on three randomly generated population with different popualtion sizes $N = 200$, $N = 800$ and $N = 2,000$, where each objective value of an individual in the population is sampled from a uniform distribution within the interval $[0, 1]$. The number of objectives varies from 2 to 15 with an increment of one. Fig. 15 shows the variation of the number of non-domination levels with different number of objectives. All these three trajectories drop rapidly with the increase of number of objectives, which fully confirm the observations in [13].



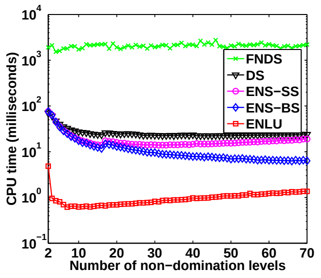
(a) 2-objective case



(b) 5-objective case

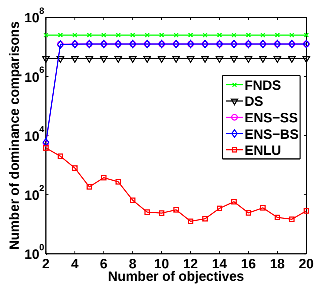


(c) 10-objective case

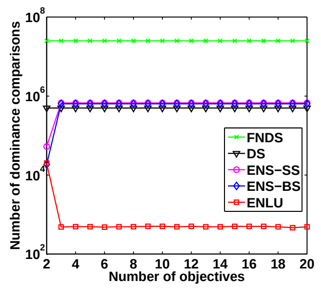


(d) 15-objective case

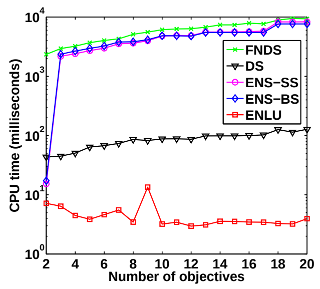
Figure 13: Comparisons of CPU time of ENLU approach and other four non-dominated sorting methods for fixed fronts data sets.



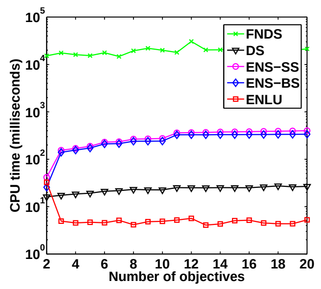
(a) 1 non-domination level



(b) 20 non-domination levels



(c) 1 non-domination level



(d) 20 non-domination levels

Figure 14: Comparisons of ENLU approach and other four non-dominated sorting methods for fixed fronts data sets with different number of objectives.

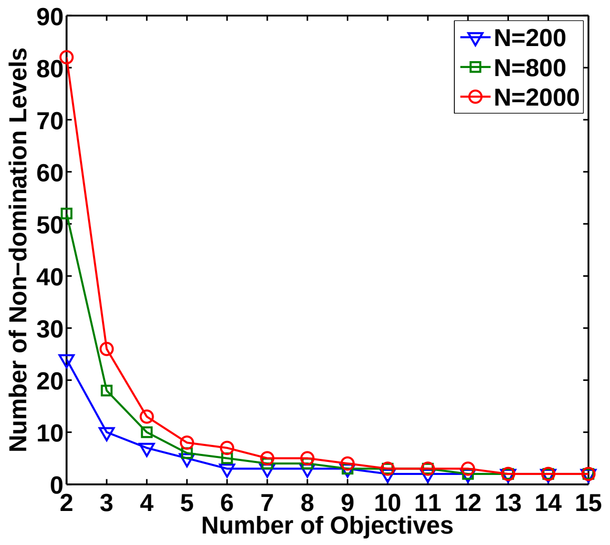
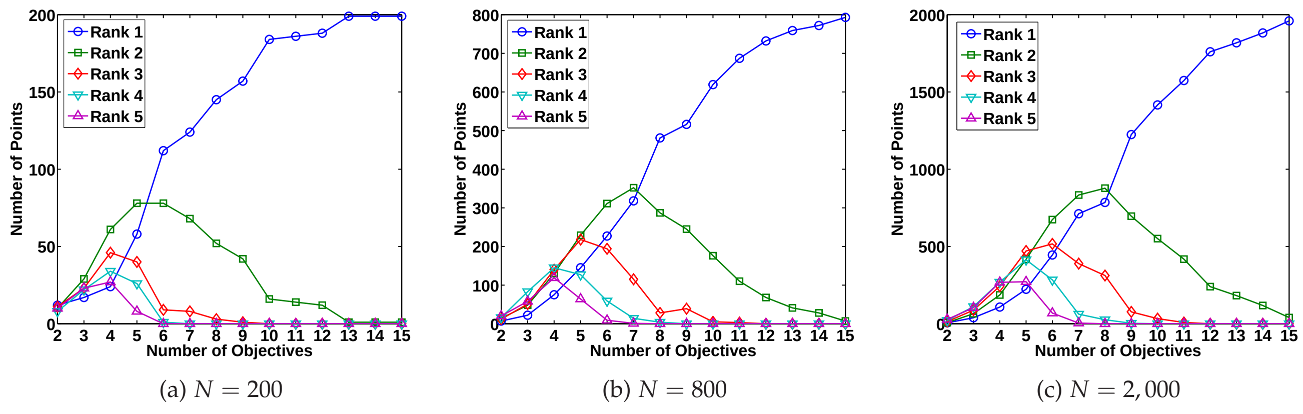


Figure 15: Number of non-domination levels formed in a random population, with population size 200, 800 and 2,000, respectively, for different number of objectives.

In real optimization scenarios, people might be more interested in the amount of solutions in the first few non-domination levels. Here, we conduct an experiment to investigate the variation of the number of solutions in the first five non-domination levels for different number of objectives. Fig. 16 presents the corresponding trajectories for $N = 200$, 800 and 2,000. From these three subfigures, we find that the variations of the number of solutions in different non-domination levels share a similar trend. In particular, the amount of solutions in the first non-domination level increases with the number of objectives, while for the other four non-domination levels, the number of solutions slightly increases at first with the number of objectives and decreases later on. This observation also confirms the previous experiment that the number of non-domination levels decreases with the increase of the number of objectives.



(a) $N = 200$ (b) $N = 800$ (c) $N = 2,000$

Figure 16: Number of points in the first five non-domination levels for different number of objectives.

The empirical studies on randomly generated populations uncover several issues in the early stages of evolution. In order to investigate the scenarios in the late stage of evolution, we design another synthetic data set developed from DTLZ2 [14]. In particular, the mathematical description of DTLZ2 is as follows:

$$\begin{aligned}
\min f_1(\mathbf{x}) &= T\cos(\frac{x_1\pi}{2})\cdots\cos(\frac{x_{m-2}\pi}{2})\cos(\frac{x_{m-1}\pi}{2})\\
\min f_2(\mathbf{x}) &= T\cos(\frac{x_1\pi}{2})\cdots\cos(\frac{x_{m-2}\pi}{2})\sin(\frac{x_{m-1}\pi}{2})\\
\min f_3(\mathbf{x}) &= T\cos(\frac{x_1\pi}{2})\cdots\sin(\frac{x_{m-2}\pi}{2})\\
&\cdots\\
\min f_m(\mathbf{x}) &= T\sin(\frac{x_1\pi}{2})
\end{aligned}$$

where $T = 1 + g(\mathbf{x}_m)$ and $g(\mathbf{x}_m) = \sum_{x_i\in\mathbf{x}_m}(x_i - 0.5)^2$, $x_i \in [0,1]$ for each $i \in \{1,2,\cdots,n\}$. We set x_i, where $i \in \{1,2,\cdots,m-1\}$, be sampled from a uniform distribution within the interval $[0,1]$ and x_j, where $j \in \{m,m+1,\cdots,n\}$, be sampled from the interval $[0.5,0.6]$. By evaluating the objective values of each sampled solution, we obtain a synthetic data set beneath the PF of DTLZ2. Fig. 17 presents an illustration in a three-dimensional space. Similar to the previous experiments on the randomly generated population, we also investigate two aspects: the variation of the number of non-domination levels for different number of objectives and the variation of the number of solutions in the first five non-domination levels. The empirical results are presented in Fig. 18 and Fig. 19. The trajectories in these figures share similar trends as those observed in Fig. 15 and Fig. 16. However, it is also worth noting that the number of non-domination levels formed in such synthetic data set is much smaller than that in the randomly generated population. This observation also implies that the number of non-domination level structure becomes relatively simpler, in the late stage of evolution, when the population almost converges to the PF.

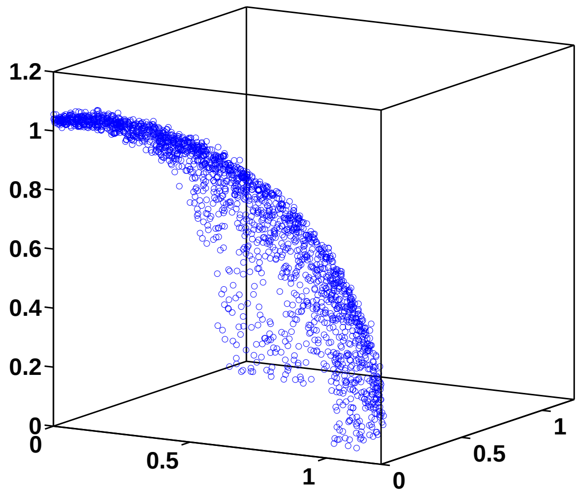


Figure 17: Illustration of population distribution in a band area for three-dimensional case.

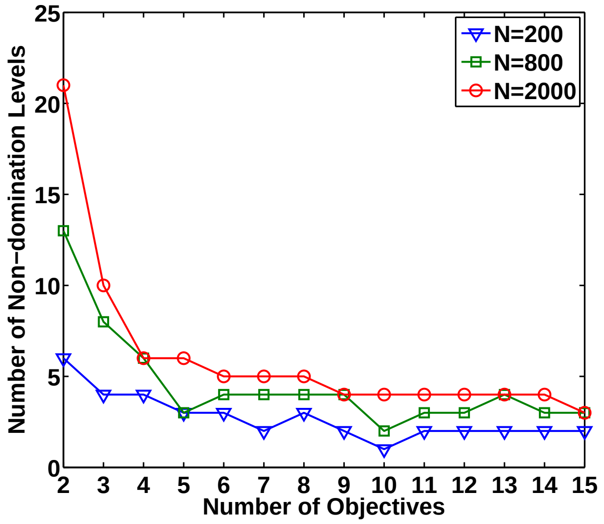


Figure 18: Number of non-domination levels formed in the synthetic data sets with population size 200, 800 and 2,000, respectively, for different number of objectives.

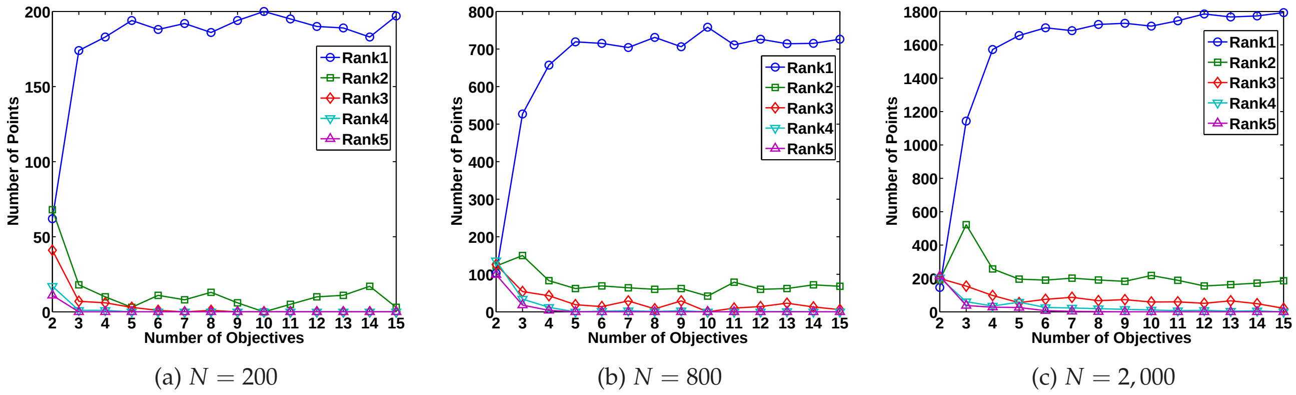


Figure 19: Number of points in the first five non-domination levels for different number of objectives.

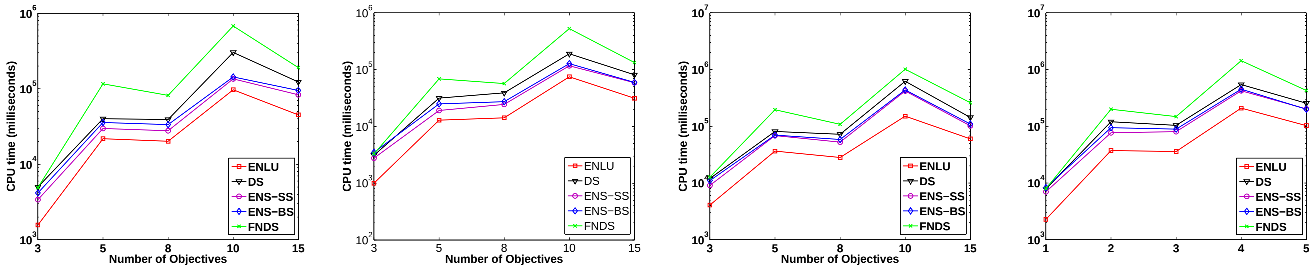
5.4 Performance Investigations on Embedded NSGA-II

In addition to the empirical studies on synthetic data sets, it is also interesting to investigate the performance of ENLU approach in EMO algorithms. To this end, we compare ENLU approach with fast non-dominated sort when they are embedded in a steady-state NSGA-II. This can be implemented by replacing line 4 of Algorithm 1 with each of these methods. As in [15], the benchmark problems DTLZ1 to DTLZ4 [14] having three, five, eight, ten and fifteen objectives are used to test the performance. In the experiments, the steady-state NSGA-II uses simulated binary crossover [16] and polynomial mutation [17] for offspring generation. The parameters are set the same as recommended in [3]. The number of generations and population size for different number of objectives are given in Table 1.

Fig. 20 presents the CPU time comparisons of the steady-state NSGA-II, using ENLU approach, fast non-dominated sort, respectively, for different benchmark problems. From these figures, it is clear that the proposed ENLU approach outperforms the fast non-dominated sort algorithms in evolutionary optimization. Consistent to the studies in previous section, fast non-dominated sort consumes the longest CPU time when embedded in the steady-state NSGA-II. Moreover, it is worth noting that the trajectories in these four figures share similar trend, where the amount of CPU time depends on both population size and number of objectives.

Table 1: Number of generations for different test instances.

Test Instance	$m = 3$	$m = 5$	$m = 8$	$m = 10$	$m = 15$
DTLZ1	400	600	750	1,000	1,500
DTLZ2	250	350	500	750	1,000
DTLZ3	1,000	1,000	1,000	1,500	2,000
DTLZ4	600	1,000	1,250	2,000	3,000
Population Size	92	212	156	276	136



(a) DTLZ1 (b) DTLZ2 (c) DTLZ3 (d) DTLZ4

Figure 20: Comparisons of CPU time of the steady-state NSGA-II with ENLU mechanism and Fast Non-dominated Sorting Procedure on DTLZ1 to DTLZ4.

6 Conclusions

In this paper, a novel non-domination level update approach, termed ENLU, is proposed for steady-state EMO algorithms. Instead of conducting the non-dominated sorting from scratch, ENLU approach takes advantages of the existing knowledge of the population to update its non-domination level structure. In steady-state EMO algorithms, the non-domination level structure might be varied in two circumstances: one is the addition of a new candidate solution into the parent population, the other is the elimination of an inferior solution from the hybrid population of parents and offspring after the update procedure. By carefully analyzing the population structure, ENLU approach only updates some population members which are necessary to change their non-domination levels. This largely saves unnecessary dominance comparisons. Theoretically, ENLU approach has a time complexity of $O(mN\sqrt{N})$ in the worst case, and a time complexity of $O(m)$ in the best case. Extensive empirical studies fully demonstrate the computational efficiency achieved by the proposed ENLU approach, comparing to the other four popular non-dominated sorting algorithms.

References

[1] K. Miettinen, *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, 1999, vol. 12.

[2] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

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

[4] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *PPSN'04: Proc. of the 8th International Conference on Parallel Problem Solving from Nature*, 2004, pp. 832–842.

[5] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, pp. 712–731, Dec. 2007.

[6] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA: John Wiley & Sons, Inc., 2001.

[7] 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.

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

[9] K. Deb, M. Mohan, and S. Mishra, "Evaluating the ϵ-domination based multi-objective evolutionary algorithm for a quick computation of pareto-optimal solutions," *Evolutionary Computation*, vol. 13, no. 4, pp. 501–525, 2005.

[10] N. Beume, B. Naujoks, and M. Emmerich, "SMS-EMOA: Multiobjective selection based on dominated hypervolume," *European Journal of Operational Research*, vol. 181, no. 3, pp. 1653–1669, 2007.

[11] S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Decomposition-based multiobjective evolutionary algorithm with an ensemble of neighborhood sizes," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 3, pp. 442–446, 2012.

[12] X. Zhang, Y. Tian, R. Cheng, and Y. Jin, "An efficient approach to non-dominated sorting for evolutionary multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, 2014, accepted for publication.

[13] H. Ishibuchi, N. Tsukamoto, and Y. Nojima, "Evolutionary many-objective optimization: A short review," in *CEC'08: Proc. of the 2008 IEEE Congress on Evolutionary Computation*, 2008, pp. 2419–2426.

[14] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable test problems for evolutionary multiobjective optimization," in *Evolutionary Multiobjective Optimization*, ser. Advanced Information and Knowledge Processing, A. Abraham, L. Jain, and R. Goldberg, Eds. Springer London, 2005, pp. 105–145.

[15] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point based non-dominated sorting approach, part I: Solving problems with box constraints," *IEEE Transactions on Evolutionary Computation*, 2013, accepted for publication.

[16] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Systems*, vol. 9, pp. 1–34, 1994.

[17] K. Deb and M. Goyal, "A combined genetic adaptive search (GeneAS) for engineering design," *Computer Science and Informatics*, vol. 26, pp. 30–45, 1996.