

Analysis of Merge Non-dominated Sorting Algorithm

Sumit Mishra¹, Ved Prakash², and Carlos A. Coello Coello³

¹ Dept. of CSE, RGIPT, Amethi, India

² Dept. of CSE, IIIT Guwahati, Guwahati, India

³ Departamento de Computación, CINVESTAV-IPN, Mexico City, Mexico

sumitm@rgipt.ac.in, ved.prakash@iiitg.ac.in,

carlos.coellocoello@cinvestav.mx

Abstract. Regardless of the recent trends regarding the design of multi-objective evolutionary algorithms (MOEAs), non-dominated sorting remains as an important mechanism to classify solutions to a multi-objective optimization problem. Non-dominated sorting is defined for a set of points in M -dimensional space (where M is the number of objectives of a multi-objective problem). These points are the objective vectors associated with the solutions of the optimization problem. Many approaches have been proposed in the last two decades to produce more computationally efficient non-dominated sorting procedures. A recent approach is Merge Non-Dominated Sorting (MNDS) which uses the concept of dominance set of points and exploits the efficient implementation of bitset in the Java programming language. In the original paper proposing MNDS, it was claimed that for the best case to happen, there should be no dominance among the points, *i.e.*, all points should be in a single front. However, in this paper, we show that, in this particular scenario, the worst-case can also occur. So, when there is no dominance among the points, *i.e.*, all the points are in the same front, the best and the worst cases can occur. We also provide here a detailed complexity analysis of this algorithm.

Keywords: Non-dominated sorting · Bitset · Dominance set.

1 Introduction

Non-dominated sorting was introduced as a key component of Pareto-based multi-objective evolutionary algorithms (MOEAs) in the 1990s, but it has been used also with other types of MOEAs (*e.g.*, the SMS-EMOA [1], which is an indicator-based approach). Non-dominated sorting is defined for the set of points in M -dimensional space (where M is the number of objectives of a multi-objective optimization problem (MOP)). Generally, these M -dimensional points are the objective vectors that are associated with the solutions of the corresponding MOP.

Here, we assume⁴ that all objectives that correspond to the coordinates of the points need to be minimized. Under this assumption, we define *dominance* as follows: a point $P_i = \langle P_{i_1}, P_{i_2}, \dots, P_{i_M} \rangle$ dominates another point $P_j = \langle P_{j_1}, P_{j_2}, \dots, P_{j_M} \rangle$, denoted as $P_i \prec P_j$, in case the following conditions are satisfied:

- $\forall m \in \{1, 2, \dots, M\} P_{i_m} \leq P_{j_m}$
- $\exists m \in \{1, 2, \dots, M\} P_{i_m} < P_{j_m}$.

When neither $P_i \prec P_j$ nor $P_j \prec P_i$, then P_i and P_j are said to be *non-dominated*. When point P_i does not dominate another point P_j , it is denoted as $P_i \not\prec P_j$. Let $\mathbb{P} = \{P_1, P_2, \dots, P_N\}$ be a set of N points which we denote as population of size N . A point $P_i \in \mathbb{P}$ in \mathbb{R}^M is represented using a vector of size M . Non-dominated sorting produces a set of *fronts* $\mathbb{F} = \{F_1, F_2, \dots\}$ such that the following conditions hold:

- All the points in the population are present in one of the fronts
 - $\bigcup_{k \geq 1} F_k = \mathbb{P}$
- No two fronts can share any point
 - $\forall i \neq j F_i \cap F_j = \emptyset$
- All the points in a front are non-dominated
 - $\forall k \geq 1 \forall P_i, P_j \in F_k : P_i \not\prec P_j \wedge P_j \not\prec P_i$
- No point in the first front is dominated by any other point
 - $\forall P_j \in F_1 \nexists P_i \in \mathbb{P} : P_i \prec P_j$
- All the points in a front (except for the first front) are dominated by at least one point in its preceding front
 - $\forall P_j \in F_k, k > 1 \exists P_i \in F_{k-1} : P_i \prec P_j$.

Authors have developed various approaches in the last 25 years for the non-dominated sorting problem. Recently, an approach named Merge Non-Dominated Sorting (MNDS) was proposed in [19] utilizing the concept of `bitset` available in programming languages such as `Java`.

As per the original paper [19], “The best case happens when there is no dominance among the solutions”. However, here, we show that, in this particular scenario, the worst-case can also occur. So, when there is no dominance among the solutions, *i.e.*, all the solutions are in the same front, then the best and the worst cases can occur.

We have also discussed that to achieve $\mathcal{O}(N \log N)$ best case time complexity as reported in the original paper [19], the `SUBSET()` procedure mentioned in line no. 6 of Algorithm 3 of the paper, should be implemented in an efficient manner so that it takes $\mathcal{O}(1)$ time. For this purpose, we maintain an additional information which is the first `set` bit in the `bitset` and this information can be maintained in constant time whenever we `set` a bit. The alternative to this is to use the `Java` language `nextSetBit(0)` method but the time complexity of this method is not provided at the Oracle Documentation⁵. We have implemented the

⁴ without loss of generality

⁵ <https://docs.oracle.com/en/java/javase/22/docs/api/java.base/java/util/BitSet.html>

algorithm in the manner that takes the time provided in the original paper [19]. In this paper, our motivation is not to compare the performance of the MNDS with other approaches as that has been provided by the authors of the MNDS paper. Here we focus on the analysis part. The main contributions of this paper are the following:

- We provide a more detailed explanation of the algorithms presented in [19].
- The best and worst-case scenarios are discussed in detail.
- We also discuss a scenario where both the best and the worst cases can occur depending on the objective values of the points. This scenario occurs when all the points are in the same front.
 - **Worst-case:** When the initial $M - 2$ objective values are the same for all the points and the last two objectives are able to decide that all the points are non-dominated. In this case, MNDS takes $\mathcal{O}(MN^2)$ time.
 - **Best-case:** When the initial two objectives are able to decide that all the points are non-dominated. In this case, the MNDS takes $\mathcal{O}(N \log N)$ time.
- We provide a detailed explanation of each part of the algorithm so that the mentioned time complexity can be achieved. We have implemented the algorithm and the source code can be found at Github⁶.

2 Previous Related Work

In the last two decades, numerous algorithms have been proposed for non-dominated sorting. Each of them has their own advantages and can be beneficial for different sets of points.

One of the early approaches was proposed by Srinivas *et al.* [21] where a point can be compared with other points multiple times. To restrict these multiple comparisons, Deb *et al.* [4] proposed a fast non-dominated sort where all the points are compared only once. After this paper, many approaches have been developed. An approach known as Deductive sort [12] was proposed using a transitivity relationship between the dominance relation of the points to reduce the dominance comparisons between the points. Some of the approaches have been proposed based on the idea of pre-sorting the points, before actually assigning them to their respective fronts.

A framework known as ENS (Efficient Non-dominated Sort) [23] was developed based on the idea of pre-sorting where the points are initially sorted based on the first objective. After pre-sorting, the points are assigned to their respective fronts by comparing a point with the already assigned points. Based on the search technique, two approaches – ENS with sequential search and ENS with binary search were proposed. By extending the idea of ENS-BS, another approach known as ENS-NDT [9] was proposed. Roy *et al.* [20] proposed an approach known as Best Order Sort where the pre-sorting is performed based on all the objectives unlike ENS [23], ENS-NDT [9], and others. An approach based

⁶ https://github.com/sumitiitp/MNDS/blob/main/MNDS_Sumit.java

on the concept of dominance degree was proposed by Zhou *et al.* [24] where the authors reduced the floating point comparisons. However, in the worst case, this approach takes $\mathcal{O}(MN^2 + N^3)$ time [15].

Jensen [10] developed a divide-and-conquer-based approach motivated by the work of Kung *et al.* [11]. Jensen’s approach is unsuitable for a population where the points share the same value for any objective. This limitation is addressed by Fortin *et al.* [8]. Buzdalov *et al.* [3] proposed an improved version of Fortin’s algorithm. Using van Emde Boas tree [5, 6], a better algorithm was proposed by Buzdalov [2] which takes $\mathcal{O}(N \log^{M-2} N \log \log N)$ time. Other approaches have exploited the divide-and-conquer strategy [7, 16–18] where the population is divided unlike [3, 8, 10] where the objective space is also divided.

As per [10], their limitation of not handling the points with the same objective value can be easily overcome, but as per Fortin *et al.* [8], doing this will increase the time to $\mathcal{O}(MN^2)$ from $\mathcal{O}(N \log^{M-1} N)$. In [14], it has been shown that Deductive Sort takes $\Theta(MN^3)$ time in the worst case. The complexity analysis of Filter Sort [22] has not been performed. Later on, it was shown in [13] that this approach takes $\Omega(N^3)$ time in the worst-case.

3 Merge Non-Dominated Sort

This section discusses the Merge Non-Dominated Sort (MNDS) algorithm which uses the concept of dominance set of points in the population. The dominance set of a point $P \in \mathbb{P}$ is the set of points in the population that dominates P and is denoted as $P.\text{ds}$. Formally, $P.\text{ds} = \{P' | P' \in \mathbb{P} \wedge P' \prec P\}$. MNDS first obtains the dominance set of all the points and after that, it ranks these points. For this purpose, the points with an empty dominance set are assigned a rank 1 because they are not dominated by any point in the population. The rank of other point (say P) is one more than the largest rank of the points in the dominance set $P.\text{ds}$ of point P . Formally, $P.\text{rank} = 1 + \max(P'.\text{rank} | P' \in P.\text{ds})$.

A simple solution to obtain the dominance set of points is to compare all the points with each other. However, this will take $\mathcal{O}(MN^2)$ time. To efficiently compute the dominance set of points, MNDS uses a different technique. MNDS uses the concept of objective dominance set $ods(P, m)$ for each point $P \in \mathbb{P}$ and for each objective $1 \leq m \leq M$. MNDS sorts the population based on every objective one by one. The sorted order of the points based on a particular objective determines the dominance relationship for that objective, *i.e.*, the first point is not dominated, the second point is dominated by the first, the third point is dominated by the first and the second, and so on. To obtain the objective dominance set of a point P based on the m^{th} objective, *i.e.*, to obtain $ods(P, m)$, MNDS sorts the points based on the m^{th} objective. After sorting the population based on the m^{th} objective, the $ods(P, m)$ is constructed by traversing the ordered points until P in the sorted list and adding those points to $ods(P, m)$. After obtaining the objective dominance set of each point based on all the objectives, the dominance set of a point P is calculated as follows. $P.\text{ds} = ods(P, 1) \cap ods(P, 2) \cap \dots \cap ods(P, M)$.

Algorithm 1 MNDS

Input: $\mathbb{P} = \{P_1, P_2, \dots, P_N\}$: Points in M -dimensional space
Output: $\text{ranks}[1, 2, \dots, N]$: Rank of all the points in the population

```

1:  $\text{ranks}[1, 2, \dots, N] \leftarrow \emptyset$  ▷ An array to store the rank of each point
2: for  $i \leftarrow 1$  to  $N$  do
3:    $\text{ranks}[i] \leftarrow 1$ 
4:  $\text{isDominance} \leftarrow \text{FALSE}$  ▷ Initially, there is no dominance
5:  $\mathbb{P} \leftarrow \text{SORT-1}(\mathbb{P})$  ▷ Sort the points based on the first objective
6:  $\langle \mathbb{P}, \text{isDominance} \rangle \leftarrow \text{SORT-2}(\mathbb{P})$  ▷ Sort the points based on the second objective
7: if  $\text{isDominance}$  then
8:    $\langle \mathbb{P}, \text{isDominance} \rangle \leftarrow \text{SORT-REST}(\mathbb{P})$  ▷ Sort based on remaining objectives
9:   if  $\text{isDominance}$  then
10:     $\text{ranks} \leftarrow \text{RANKING}(\mathbb{P})$ 
11: return  $\text{ranks}$  ▷ Return the rank of all the points in the population

```

MNDS is discussed in Algorithm 1. In this algorithm, points are first sorted based on the first objective. Ties are resolved using lexicographical sorting like [23]. In this process, MNDS maintains an index $P.\text{index}$ of each point $P \in \mathbb{P}$ in the sorted list based on the first objective. This index helps in calculating the objective dominance set based on the first objective. The objective dominance set of a point $P \in \mathbb{P}$ based on the first objective $ods(P, 1)$ is the set of points with indices less than $P.\text{index}$, i.e., $ods(P, 1) = \{P' | P'.\text{index} < P.\text{index}\}$. So, MNDS does not store $ods(P, 1)$ for $P \in \mathbb{P}$ explicitly, rather the index of the point $P.\text{index}$ is sufficient to tell the objective dominance set. Sorting the points based on the first objective and calculating the index of the points is summarized in Algorithm 2.

Algorithm 2 SORT-1

Input: $\mathbb{P} = \{P_1, P_2, \dots, P_N\}$: Points in M -dimensional space
Output: \mathbb{P} : Sorted population based on the first objective

```

1: Sort the population  $\mathbb{P}$  considering the first objective
2: for  $i \leftarrow 1$  to  $N$  do
3:    $P \leftarrow \mathbb{P}[i]$  ▷  $i^{\text{th}}$  point of the sorted population
4:    $P.\text{index} \leftarrow i$ 
5: return  $\mathbb{P}$  ▷ Return the sorted population

```

MNDS then sorts the points based on the second objective and obtains the dominance set considering the sorted order of the points based on the first two objectives. To solve the tie, the ordering based on the first objective is considered. Sorting the points based on the second objective is summarized in Algorithm 3. In this algorithm, the dominance set $P.\text{ds}$ of point $P \in \mathbb{P}$ is the set of points in the objective dominance set $ods(P, 2)$ with an index lower than $P.\text{index}$. This is basically same as $P.\text{ds} = ods(P, 1) \cap ods(P, 2)$. In this algorithm, the subset procedure has been used in line 8 and 13 of Algorithm 3. In the subset procedure of line 8, we have to only check whether the subset is empty or not. We are not

interested in the actual contents of the subset. In this case, if $P.\text{index}$ is less than $ods.\text{min}$ then it means that the subset within the range 1 and $P.\text{index}$ is empty. This condition can be checked in $\mathcal{O}(1)$ time. However, in line 13, we are checking the subset of ods within a particular range. For the subset of line 13, the intersection between two sets a and b is only applied within the range $[\max(a.\text{min}, b.\text{min}), \min(a.\text{max}, b.\text{max})]$ as discussed in the original paper [19].

Objective dominance set ods is implemented using bitset. $ods.\text{min}$ provides the index of the first set bit in the ods and $ods.\text{max}$ provides the index of the last set bit in the ods .

Algorithm 3 SORT-2

Input: $\mathbb{P} = \{P_1, P_2, \dots, P_N\}$: Points in M -dimensional space
Output: \mathbb{P} : Sorted population based on the second objective, **isDominance**: A boolean variable (TRUE when there is dominance and FALSE when there is no dominance)

```

1:  $ods \leftarrow \emptyset$  ▷ Objective dominance set for the second objective. It is implemented using bitset
2:  $ods.\text{min} \leftarrow N$  ▷  $ods.\text{min}$  is the index of the first set bit
3: Sort the population  $\mathbb{P}$  based on the second objective
4: isDominance  $\leftarrow$  FALSE ▷ Initially, there is no dominance
5: for  $i \leftarrow 1$  to  $N$  do
6:    $P \leftarrow \mathbb{P}[i]$  ▷  $i^{\text{th}}$  point of the sorted population
7:    $P.\text{ds} \leftarrow \emptyset$  ▷ Dominance set of point  $P$ 
8:   if  $ods.\text{SUBSET}(1, P.\text{index}) = \emptyset$  then ▷ Subset of  $ods$  within the range 1 and  $P.\text{index}$  (both inclusive) is not empty
9:     if  $P.\text{index} < ods.\text{min}$  then
10:        $ods.\text{min} \leftarrow P.\text{index}$ 
11:   else
12:      $\text{high} \leftarrow \min(ods.\text{max}, P.\text{index} - 1)$ 
13:      $P.\text{ds} \leftarrow ods.\text{SUBSET}(ods.\text{min}, \text{high})$  ▷ Subset of  $ods$  within the range  $ods.\text{min}$  and  $\text{high}$  (both inclusive)
14:     isDominance  $\leftarrow$  TRUE
15:      $ods \leftarrow ods \cup \{P.\text{index}\}$ 
16:     if  $P.\text{index} > ods.\text{max}$  then
17:        $ods.\text{max} \leftarrow P.\text{index}$ 
18: return  $\mathbb{P}, \text{isDominance}$ 

```

The dominance set $P.\text{ds}$ of a point P is the intersection of the objective dominance set $ods(P, m)$ of point P based on all the objectives. So, in case the dominance set of point P is empty after sorting the points based on the second objective, then there is no need to compute the objective dominance set based on other objectives as there will not be any further change. MNDS exploits this fact and checks if the dominance set of all the points is empty. If that's the case, then there is no need to compute the objective dominance set any further.

In case the dominance set of all the points is not empty after obtaining the dominance set based on the first two objectives, we sort the points based on other

objectives. In this process of sorting the points based on the m^{th} ($3 \leq m \leq M$) objective, whenever the dominance set of all the points is empty, we stop the sorting process. In case of tie while sorting the points, the sorted order based on the first objective is considered. While sorting the points based on the m^{th} objective, we compute the dominance set by obtaining the intersection of the objective dominance set and the current dominance set of point P . Sorting the points based on other objectives is summarized in Algorithm 4.

After obtaining the dominance set of all the points, we rank the points. Initially, the rank of all the points is initialized with 1. The first point is having rank 1 as its dominance set is empty. Now, we traverse the sorted list and rank the points. To assign rank to a point P , we check the dominance set of P . The rank of P is equal to one more than the highest rank of the points in the dominance set of P . This ranking procedure is summarized in Algorithm 5.

Algorithm 4 SORT-OTHERS

Input: $\mathbb{P} = \{P_1, P_2, \dots, P_N\}$: Points in M -dimensional space

Output: \mathbb{P} : Sorted population based on the second objective, **isDominance**: A Boolean variable (TRUE when there is dominance and FALSE when there is no dominance)

```

1: isDominance  $\leftarrow$  TRUE
2:  $m \leftarrow 3$ 
3: while isDominance  $\wedge m \leq M$  do
4:   if SORT( $\mathbb{P}, m$ ) then ▷ Sort the population based on
     the  $m^{th}$  objective
5:     isDominance  $\leftarrow$  FALSE
6:      $ods \leftarrow \emptyset$  ▷ Objective dominance set for the  $m^{th}$  objective
7:     for  $i \leftarrow 1$  to  $|\mathbb{P}|$  do
8:        $P \leftarrow \mathbb{P}[i]$ 
9:        $P.ds \leftarrow P.ds \cap ods$ 
10:       $ods \leftarrow ods \cup \{P.index\}$ 
11:      if  $P.ds \neq \emptyset$  then
12:        isDominance  $\leftarrow$  TRUE
13:       $m \leftarrow m + 1$ 
14: return  $\mathbb{P}, isDominance$ 

```

4 Complexity Analysis

This section discusses the complexity analysis of MNDS. For an efficient implementation of MNDS, it is important to efficiently check the subset procedure in line no. 8 of Algorithm 3. In line no. 8 of Algorithm 3, the subset procedure finds the subset of ods where the index of the points lies within the range 1 to $P.index$. As the ods is implemented using **Bitset**, so to obtain the subset we have to traverse the **Bitset** from index 1 to $P.index$ which may take linear

Algorithm 5 RANKING**Input:** $\mathbb{P} = \{P_1, P_2, \dots, P_N\}$: Points in M -dimensional space**Output:** $\text{ranks}[1, 2, \dots, N]$: Rank of all the points in the population

```

1:  $\text{ranks}[1, 2, \dots, N] \leftarrow 1$  ▷ Initialize the array with 1
2:  $\text{ranks}[1] \leftarrow 1$ 
3: for  $i \leftarrow 2$  to  $|\mathbb{P}|$  do
4:    $P \leftarrow \mathbb{P}[i]$ 
5:    $\text{myRank} \leftarrow 1$ 
6:   for each  $q \in P.\text{ds}$  do
7:      $\text{thatRank} \leftarrow \text{ranks}[q]$ 
8:     if  $\text{thatRank} \geq \text{myRank}$  then
9:        $\text{myRank} \leftarrow \text{thatRank} + 1$ 
10:   $\text{ranks}[P] \leftarrow \text{myRank}$ 
11: return  $\text{ranks}$ 

```

time in the worst case. However, in this subset procedure, we are not interested in the actual contents of the subset, but instead, we are interested in knowing whether the result of the subset is empty or not. *ods* is implemented using Bitset. To make this process efficient, we can maintain a variable that keeps the lowest index in the Bitset which is set to TRUE. This can also be done using `nextSetBit(0)` in the Java programming language. We denote this index with the lowest set index as *ods.min*. If $P.\text{index}$ is less than *ods.min*, then the subset is empty and this can be done in constant time.

In line 13 of Algorithm 3, we are interested in the contents of the subset. For the subset of line 13, the intersection between two sets is performed and this is done using the AND operation in Java's bitset. Now we discuss the complexity of several algorithms.

- **Algorithm 2:** In the worst case, this algorithm takes $\mathcal{O}(MN \log N)$ time when the initial $M - 1$ objective values of the points share the same value so that all the objectives are considered while sorting. However, in the best case, the same algorithm takes $\mathcal{O}(N \log N)$ when points have different values for the first objective so that only one objective is considered while sorting.
- **Algorithm 3:** Sorting the points based on the second objective takes $\mathcal{O}(N \log N)$ time as ties are solved considering the sorted order of the points based on the first objective. The loop in this algorithm runs for N times. This loop initializes the dominance set of each point which will take $\mathcal{O}(N)$ time for all the points.
 - In case there is no dominance, *i.e.*, all the points are in the same front, the condition in line no. 8 is always TRUE and the dominance set $P.\text{ds}$ of each point $P \in \mathbb{P}$ is empty. Thus, the best-case time taken by Algorithm 3 is $\mathcal{O}(N \log N)$.
 - In the presence of dominance, *i.e.*, at least one point is dominated, the dominance set $P.\text{ds}$ of a point $P \in \mathbb{P}$ is initialized with the indices of those points in objective dominance set *ods* with index less than $P.\text{index}$.

In the worst-case scenario, a point at the i^{th} index is dominated by all the previous $i - 1$ points. In this case, the worst-case time complexity of line no. 13 is $1 + 2 + 3 + \dots + N - 1 = \mathcal{O}(N^2)$. Thus, the worst-case time taken by Algorithm 3 is $\mathcal{O}(N^2)$.

- **Algorithm 4:** Here, the points are sorted based on the last $M - 2$ objectives. Sorting the points based on a particular objective takes $\mathcal{O}(N \log N)$ time. Thus, the time to sort the points based on $M - 2$ objectives is $(M - 2)\mathcal{O}(N \log N)$ which is $\mathcal{O}(MN \log N)$. The loop in lines 7 – 12 computes the dominance set of each point using the Intersection operation in line no. 9 which can take $\mathcal{O}(N)$ time for each point. There are N points so the loop will take $\mathcal{O}(N^2)$ time. As the loop can run a maximum of $M - 2$ times, the worst-case time complexity of the loop is $(M - 2)\mathcal{O}(N^2) = \mathcal{O}(MN^2)$. The worst case time complexity of this algorithm is $\mathcal{O}(MN \log N) + \mathcal{O}(MN^2)$ that is $\mathcal{O}(MN^2)$.

4.1 Best Case

In this case, the dominance relationship among the points can be established based on only the first two objectives and there is no need to sort the points based on the remaining objectives. Algorithms 2 and 3 take $\mathcal{O}(N \log N)$ time when no two points share the same value for the first and the second objective. In Algorithm 3, while sorting the points based on the second objective, we need to check whether the subset of *ods* is empty or not in line no. 8. This can be done in constant time. In the best case, all the points are in the same front. So, the subset is always empty and this algorithm will take $\mathcal{O}(N \log N)$ time. Algorithm 4 will not be called in the best case. As there is no dominance, the dominance set of all the points is empty and the ranking using Algorithm 5 will take $\mathcal{O}(N)$ time. Thus, the best case time complexity is $\mathcal{O}(N \log N)$. Such a scenario is shown in Table 1.

4.2 Worst Case

The worst case occurs in some of the scenarios which are as follows.

1. **All the points are in different fronts:** Such scenario is shown in Table 2. In this case, Algorithm 2 will take $\mathcal{O}(N \log N)$ time. Algorithm 3 will take $\mathcal{O}(N^2)$ time. Here, we need to sort the points based on all the remaining objectives, so Algorithm 4 requires $\mathcal{O}(MN^2)$ time. The ranking algorithm takes $\mathcal{O}(N^2)$ time. Thus, the worst-case time complexity is $\mathcal{O}(MN^2)$.
2. **All the points are in the same front where the last two objectives decide the dominance and the initial $M - 2$ objective values of all the points are the same:** Such scenario is shown in Table 3. In this case, Algorithm 2 will take $\mathcal{O}(MN \log N)$ time. Algorithm 3 will take $\mathcal{O}(N^2)$ time. Here, we need to sort the points based on all the remaining objectives so Algorithm 4 requires $\mathcal{O}(MN^2)$ time. The ranking algorithm takes $\mathcal{O}(N^2)$ time. Hence, the time complexity of MNDS in the worst-case is $\mathcal{O}(MN^2)$.

Table 1. Best Case scenario: All the points in the same front. Here, the last $M - 2$ objective values of a point do not matter as the dominance has been already decided based on the first two objectives. So, the last $M - 2$ objectives can take any value (represented as X).

P_1	$\langle 1, 8, X, X, \dots, X \rangle$
P_2	$\langle 2, 7, X, X, \dots, X \rangle$
P_3	$\langle 3, 6, X, X, \dots, X \rangle$
P_4	$\langle 4, 5, X, X, \dots, X \rangle$
P_5	$\langle 5, 4, X, X, \dots, X \rangle$
P_6	$\langle 6, 3, X, X, \dots, X \rangle$
P_7	$\langle 7, 2, X, X, \dots, X \rangle$
P_8	$\langle 8, 1, X, X, \dots, X \rangle$

Table 2. Worst Case scenario: All the points in different front.

P_1	$\langle 1, 1, \dots, 1 \rangle$
P_2	$\langle 2, 2, \dots, 2 \rangle$
P_3	$\langle 3, 3, \dots, 3 \rangle$
P_4	$\langle 4, 4, \dots, 4 \rangle$
P_5	$\langle 5, 5, \dots, 5 \rangle$
P_6	$\langle 6, 6, \dots, 6 \rangle$
P_7	$\langle 7, 7, \dots, 7 \rangle$
P_8	$\langle 8, 8, \dots, 8 \rangle$

3. **All the points are in the same front where the last two objectives decide the dominance and the initial $M - 2$ objective values of the i^{th} point are greater than the initial $M - 2$ objective values of all the previous points:** Such scenario is shown in Table 4. In this case, Algorithm 2 will take $\mathcal{O}(N \log N)$ time. Algorithm 3 will take $\mathcal{O}(N^2)$ time. Here, we need to sort the points based on all the remaining objectives so Algorithm 4 requires $\mathcal{O}(MN^2)$ time. The ranking algorithm takes $\mathcal{O}(N^2)$ time. Hence, the time complexity of MNDS in the worst-case is $\mathcal{O}(MN^2)$.

Table 3. Worst Case scenario: All the points are in the same front.

P_1	$\langle 1, 1, \dots, 1, 1, 8 \rangle$
P_2	$\langle 1, 1, \dots, 1, 2, 7 \rangle$
P_3	$\langle 1, 1, \dots, 1, 3, 6 \rangle$
P_4	$\langle 1, 1, \dots, 1, 4, 5 \rangle$
P_5	$\langle 1, 1, \dots, 1, 5, 4 \rangle$
P_6	$\langle 1, 1, \dots, 1, 6, 3 \rangle$
P_7	$\langle 1, 1, \dots, 1, 7, 2 \rangle$
P_8	$\langle 1, 1, \dots, 1, 8, 1 \rangle$

Table 4. Worst Case scenario: All the points are in the same front.

P_1	$\langle 1, 1, \dots, 1, 1, 8 \rangle$
P_2	$\langle 2, 2, \dots, 2, 2, 7 \rangle$
P_3	$\langle 3, 3, \dots, 3, 3, 6 \rangle$
P_4	$\langle 4, 4, \dots, 4, 4, 5 \rangle$
P_5	$\langle 5, 4, \dots, 5, 5, 4 \rangle$
P_6	$\langle 6, 6, \dots, 6, 6, 3 \rangle$
P_7	$\langle 7, 7, \dots, 7, 7, 2 \rangle$
P_8	$\langle 8, 8, \dots, 8, 8, 1 \rangle$

5 Conclusions

In this paper, we have shown that for MNDS, when all the points are in a single front, then the best as well as the worst-case scenario can occur. This is due to the objective values of the points. We have also discussed how to efficiently

implement the SUBSET() procedure mentioned in line no. 6 of Algorithm 3 so that it can be done in constant time. The best and the worst-case scenarios were also discussed in detail. A detailed explanation of each part of the algorithm was provided so that the previously indicated time complexity could be achieved.

References

1. Beume, N., Naujoks, B., Emmerich, M.: SMS-EMOA: Multiobjective Selection Based on Dominated Hypervolume. *European Journal of Operational Research* **181**(3), 1653–1669 (2007)
2. Buzdalov, M.: Make Evolutionary Multiobjective Algorithms Scale Better with Advanced Data Structures: Van Emde Boas Tree for Non-Dominated Sorting. In: *International Conference on Evolutionary Multi-Criterion Optimization (EMO'2019)*. pp. 66–77. Springer. Lecture Notes in Computer Science Vol. 11411, East Lansing, MI, USA (March 10-13 2019), ISBN: 978-3-030-12597-4
3. Buzdalov, M., Shalyto, A.: A Provably Asymptotically Fast Version of the Generalized Jensen Algorithm for Non-Dominated Sorting. In: *International Conference on Parallel Problem Solving from Nature – PPSN XIII*. pp. 528–537. Springer. Lecture Notes in Computer Science Vol. 8672, Ljubljana, Slovenia (September 13-17 2014)
4. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* **6**(2), 182–197 (April 2002)
5. van Emde Boas, P.: Preserving Order in a Forest in Less Than Logarithmic Time. In: *Annual Symposium on Foundations of Computer Science (SFCS'1975)*. pp. 75–84. IEEE (1975)
6. van Emde Boas, P., Kaas, R., Zijlstra, E.: Design and Implementation of an Efficient Priority Queue. *Mathematical Systems Theory* **10**(1), 99–127 (1976)
7. Fang, H., Wang, Q., Tu, Y.C., Horstemeyer, M.F.: An Efficient Non-Dominated Sorting Method for Evolutionary Algorithms. *Evolutionary Computation* **16**(3), 355–384 (Fall 2008)
8. Fortin, F.A., Greiner, S., Parizéau, M.: Generalizing the Improved Run-Time Complexity Algorithm for Non-Dominated Sorting. In: *Proceedings of Genetic and Evolutionary Computation Conference (GECCO'2013)*. pp. 615–622. ACM Press, New York, USA (July 2013), ISBN: 978-1-4503-1963-8
9. Gustavsson, P., Syberfeldt, A.: A New Algorithm Using the Non-Dominated Tree to Improve Non-Dominated Sorting. *Evolutionary Computation* **26**(1), 89–116 (September 2018)
10. Jensen, M.T.: Reducing the Run-Time Complexity of Multiobjective EAs: The NSGA-II and Other Algorithms. *IEEE Transactions on Evolutionary Computation* **7**(5), 503–515 (October 2003)
11. Kung, H.T., Luccio, F., Preparata, F.P.: On Finding the Maxima of a Set of Vectors. *Journal of the ACM* **22**(4), 469–476 (1975)
12. McClymont, K., Keedwell, E.: Deductive Sort and Climbing Sort: New Methods for Non-Dominated Sorting. *Evolutionary Computation* **20**(1), 1–26 (Spring 2012)
13. Mishra, S., Buzdalov, M.: Filter Sort is $\Omega(N^3)$ in the Worst Case. In: *International Conference on Parallel Problem Solving from Nature – PPSN XVI*. pp. 675–685. Springer (2020)

14. Mishra, S., Buzdalov, M.: If Unsure, Shuffle: Deductive Sort is $\Theta(MN^3)$, but $\mathcal{O}(MN^2)$ in Expectation over Input Permutations. In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO'2020). pp. 516–523 (2020)
15. Mishra, S., Buzdalov, M., Senwar, R.: Time Complexity Analysis of the Dominance Degree Approach for Non-Dominated Sorting. In: Proceedings of Genetic and Evolutionary Computation Conference Companion (GECCO'2020). pp. 169–170 (2020)
16. Mishra, S., Saha, S., Mondal, S.: Divide and Conquer Based Non-Dominated Sorting for Parallel Environment. In: IEEE Congress on Evolutionary Computation (CEC'2016). pp. 4297–4304. IEEE Press, Vancouver, Canada (July 24-29 2016), ISBN: 978-1-5090-0623-6
17. Mishra, S., Saha, S., Mondal, S., Coello Coello, C.A.: A Divide-And-Conquer Based Efficient Non-Dominated Sorting Approach. *Swarm and Evolutionary Computation* **44**, 748–773 (February 2019)
18. Mishra, S., Saha, S., Mondal, S., Coello Coello, C.A.: Divide-and-conquer Based Non-dominated Sorting with Reduced Comparisons. *Swarm and Evolutionary Computation* **51** (December 2019), article Number: UNSP 100580
19. Moreno, J., Rodriguez, D., Nebro, A.J., Lozano, J.A.: Merge Nondominated Sorting Algorithm for Many-objective Optimization. *IEEE Transactions on Cybernetics* **51**(12), 6154–6164 (2020)
20. Roy, P.C., Islam, M.M., Deb, K.: Best Order Sort: A New Algorithm to Non-Dominated Sorting for Evolutionary Multi-Objective Optimization. In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO'2016). pp. 1113–1120. ACM Press, Denver, Colorado, USA (July 20-24 2016), ISBN: 978-1-4503-4323-7
21. Srinivas, N., Deb, K.: Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation* **2**(3), 221–248 (Fall 1994)
22. Wang, J., Li, C., Diao, Y., Zeng, S., Wang, H.: An Efficient Nondominated Sorting Algorithm. In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO'2018). pp. 203–204. ACM (2018)
23. Zhang, X., Tian, Y., Cheng, R., Yaochu, J.: An Efficient Approach to Nondominated Sorting for Evolutionary Multiobjective Optimization. *IEEE Transactions on Evolutionary Computation* **19**(2), 201–213 (April 2015)
24. Zhou, Y., Chen, Z., Zhang, J.: Ranking Vectors by Means of the Dominance Degree Matrix. *IEEE Transactions on Evolutionary Computation* **21**(1), 34–51 (2017)