GBOS: Generalized Best Order Sort Algorithm for Non-dominated Sorting

Sumit Mishra[a,*], Samrat Mondal[b], Sriparna Saha[b], Carlos A. Coello Coello[a]

[a] *Departamento de Computacion, CINVESTAV-IPN, Mexico City, D.F. 07360, Mexico*
[b] *Department of Computer Science & Engineering*
Indian Institute of Technology Patna, Patna, Bihar – 801106, India

Abstract

Non-dominated sorting is one of the prominent steps in developing any Pareto-dominance based multi-objective evolutionary algorithm. The computational complexity of any Pareto-dominance based multi-objective evolutionary algorithm primarily depends on this step. Thus, researchers are working on reducing the complexity of this step. Recently, an efficient approach for non-dominated sorting known as Best Order Sort (BOS) has been proposed. This approach is very efficient in terms of the number of comparisons between the solutions. Another advantage of this approach is that while comparing two solutions, the number of objectives which are compared is reduced from the actual number of objectives associated with each solution. However, in spite of these two advantages, this approach is not suitable in its current form for cases in which we have duplicate solutions. This paper generalizes BOS to handle duplicate solutions while retaining both of its advantages. We call this generalized version *Generalized Best Order Sort* (GBOS). The present work shows that BOS can be generalized to handle its limitation without compromising its time and space complexity.

Keywords: Non-dominated sorting, dominance relation, computational complexity

1. Introduction

Non-dominated sorting is one of the most critical steps of an important number of Pareto-dominance based multi- and many-objective evolutionary algorithms [1], [2]. Additionally, non-dominated sorting has applications in several domains including economics, databases, game theory and computational geometry [3].

*Corresponding author
Email addresses: smishra@computacion.cs.cinvestav.mx (Sumit Mishra), samrat@iitp.ac.in (Samrat Mondal), sriparna@iitp.ac.in (Sriparna Saha), ccoello@cs.cinvestav.mx (Carlos A. Coello Coello)

Consider $\mathbb{P} = \{s_1, s_2, \ldots, s_N\}$ to be a population of N solutions in an M-dimensional space. In case of multi- and many-objective evolutionary algorithms, M is basically the number of objective functions considered. A solution s in an M-dimensional space is represented as $s = \{f_1(s), f_2(s), \ldots, f_M(s)\}$ where $f_m(s), 1 \leq m \leq M$ is the value of s for the mth objective. We assume the minimization of all the objective functions. In this case, a solution s_i is said to dominate another solution s_j represented as $s_i \prec s_j$ when the two following conditions are satisfied:

i. $f_m(s_i) \leq f_m(s_j), \ \forall m \in \{1, 2, \ldots, M\}$
ii. $f_m(s_i) < f_m(s_j), \ \exists m \in \{1, 2, \ldots, M\}$

Two solutions s_i and s_j are said to be non-dominated when neither $s_i \nprec s_j$ nor $s_j \nprec s_i$. Now, we formally define the problem of non-dominated sorting.

Definition 1 (Non-dominated Sorting). *Given a population $\mathbb{P} = \{s_1, s_2, \ldots, s_N\}$ of size N in an M-dimensional space. Non-dominated sorting divides the solutions in the population in $K (1 \leq K \leq N)$ different fronts $\{F_1, F_2, \ldots, F_K\}$ which are arranged in decreasing order of their dominance. These non-dominated fronts are such that the two following conditions hold:*

i. $\forall s_i, s_j \in F_k$: $s_i \nprec s_j$ and $s_j \nprec s_i$ $(1 \leq k \leq K)$. This represents that all the solutions in a particular front are non-dominated to each other.
ii. $\forall s \in F_k$, $\exists s' \in F_{k-1}$: $s' \prec s$ $(2 \leq k \leq K)$. This represents that all the solutions in the kth front are dominated by at least one of the solutions in the $k-1^{th}$ front.

The recent past has seen various approaches for non-dominated sorting. In the naive approach proposed by Srinivas *et al.* [4], each solution is compared with all other solutions. The solutions which are not dominated by any other solution are assigned to the first front. At that point, only those solutions which have not been assigned to the first front are considered. These solutions are compared with each other and the solutions which are not dominated by any other solution are assigned to the second front. The process of assigning solutions to the fronts is repeated until no solution is left. The worst case time complexity of the naive approach is $\mathcal{O}(MN^3)$ when all the solutions are in different fronts, whereas the best case time complexity is $\mathcal{O}(MN^2)$ when all the solutions are in a single front. The space complexity of this approach is $\mathcal{O}(N)$.

Deb *et al.* [1] developed a fast approach for non-dominated sorting known as Fast Non-dominated Sort (FNDS) with a better worst case time complexity which is $\mathcal{O}(MN^2)$. However, the space complexity has been increased to $\mathcal{O}(N^2)$. Jensen *et al.* [5] presented a recursive approach for non-dominated sorting with time complexity of $\mathcal{O}(N \log^{M-1} N)$. The space complexity of Jensen's approach is $\mathcal{O}(MN)$. For two objectives, this approach has the time complexity of $\mathcal{O}(N \log N)$. However, this approach is not suitable when the solutions have the same value for an objective. This limitation of Jensen's approach is removed by Fortin *et al.* [6]. However, this limitation is removed at the expense of an

increased worst case time complexity to $\mathcal{O}(MN^2)$. The average case time complexity remains the same as Jensen's approach. The space complexity of Fortin's approach is the same as in Jensen's approach which is $\mathcal{O}(MN)$. An efficient non-dominated sorting method for evolutionary algorithms based on divide-and-conquer strategy is proposed by Fang *et al.* [7]. The space complexity of this algorithm is $\mathcal{O}(MN)$. The worst case time complexity of this algorithm is $\mathcal{O}(MN^2)$ when all the solutions are non-dominated. The lower bound time complexity of this algorithm is $\mathcal{O}(MN \log N)$. This algorithm considers one solution as dominated by another if both are identical. Tang *et al.* [8] proposed a fast method for constructing the non-dominated set based on arena's principle. In some cases this approach can achieve a time complexity of $\mathcal{O}(MN\sqrt{N})$ [9].

McClymont *et al.* [10] proposed two approaches: Climbing Sort and Deductive Sort (DS). Deductive sort performs better than climbing sort. The best case time complexity of Deductive sort is $\mathcal{O}(MN\sqrt{N})$. However, the worst case time complexity remains $\mathcal{O}(MN^2)$.

An Efficient Non-dominated Sort (ENS) approach is presented by Zhang *et al.* [9]. This is a two-phased approach. In the first phase, the solutions are sorted based on the first objective. In the second phase, the solutions are assigned to their respective fronts. The solutions are taken from the sorted solutions in the first phase and assigned to the fronts. Based on the search technique for assigning solutions to the fronts, there are two variants – ENS-SS (ENS with sequential search) and ENS-BS (ENS with binary search). The worst case time complexity of ENS-SS and ENS-BS is $\mathcal{O}(MN^2)$. However, the best case time complexity of both approaches differ. ENS-SS has a best case time complexity of $\mathcal{O}(MN\sqrt{N})$ and ENS-BS has a best case time complexity of $\mathcal{O}(MN \log N)$. The space complexity of both ENS-SS and ENS-BS is $\mathcal{O}(1)$. A divide-and-conquer based non-dominated sorting algorithm was developed by Mishra *et al.* [11]. This approach also works in two phases as ENS. The worst case time complexity of this approach is $\mathcal{O}(MN^2)$ and the best case time complexity is $\mathcal{O}(MN \log N)$. The time complexity of non-dominated sorting was proved to be $\mathcal{O}(N \log^{M-1} N)$ by Buzdalov *et al.* [12].

In general, for a solution to be inserted in a particular front, it is compared with all the solutions in the front. Some recent approaches have shown that there is no need to compare a solution with all the solutions in a front for its insertion. In general, a subset of solutions that have been assigned to a front are compared with the solution which needs to be inserted. This idea has been exploited by some recent approaches [13, 14, 3, 15].

Tree based Efficient Non-dominated Sort (T-ENS) was developed by Zhang *et al.* [13, 14]. In T-ENS, the solutions are sorted based on the first objective as done in ENS [9]. T-ENS considers a front as a tree which saves many unnecessary dominance comparisons. The best case time complexity of this approach is $\mathcal{O}(MN \log N / \log M)$ and the worst case time complexity is $\mathcal{O}(MN^2)$. The space complexity of T-ENS is $\mathcal{O}(MN)$. This approach is very efficient in terms of the number of dominance comparisons. However, T-ENS is not suitable when the solutions share identical values for any of the objectives [15]. Inspired by ENS-BS [9], the Efficient Non-dominated Sort with Non-dominated Tree (ENS-

NDT) [15] approach was recently proposed. Like in ENS [9] and T-ENS [13], in ENS-NDT [15] the solutions are sorted based on the first objective. The best case time complexity of this approach is $\mathcal{O}(MN \log N)$ and the worst case time complexity is $\mathcal{O}(MN^2)$. The space complexity of this approach depends on the situation. In the worst case, it is $\mathcal{O}(N \log N)$. The average case space complexity is $\mathcal{O}(N)$ and the best case space complexity is $\mathcal{O}(\log N)$.

Roy *et al.* [3] developed Best Order Sort (BOS) for non-dominated sorting. This approach works in two phases. In the first phase, the solutions are sorted based on each objective separately unlike ENS [9], T-ENS [13] and ENS-NDT [15] where solutions are sorted based on the first objective. In the second phase (rank assignment phase), rank is assigned to the solutions. The worst case time complexity of this approach is $\mathcal{O}(MN^2)$ and the best case time complexity is $\mathcal{O}(MN \log N)$. The space complexity is $\mathcal{O}(MN)$. There are two main advantages of BOS: (i) The number of dominance comparisons between the solutions is reduced to a great extent and (ii) two solutions are compared without considering all the objectives. The second advantage is because of the comparison set concept [3]. If all the solutions are in a single front, then the minimum number of dominance comparisons can be 0 in case $M \geq N$. However, in its current form, BOS is not suitable when there are duplicate solutions (in terms of objective vectors) in the population. This limitation is because of the comparison set concept. In case of duplicate solutions, one solution is considered as dominated by another in BOS. Recently, BOS has been modified to handle its limitation by removing the concept of comparison set.[1] However, removing the concept of comparison set will increase the time to compare two solutions as all the objective values of the solutions will be considered then. Hence, the second advantage of BOS is lost. We call this modified version of BOS which can handle duplicate solutions as BOS*. There are some approaches proposed for steady-state evolutionary algorithms where a solution needs to be inserted into a set of fronts (see for example [16, 17, 18, 19, 20, 21]).

The contributions in this paper are as follows.

- We have generalized BOS to remove its limitation by retaining both of its advantages. The generalized version of BOS is termed as Generalized Best Order Sort (GBOS)
- It has been theoretically shown that BOS can be generalized without compromising its worst case time and space complexities.
- The number of dominance comparisons performed by GBOS in three different scenarios in its worst and best case are also theoretically obtained.
- We have theoretically obtained the actual number of comparisons between the objectives of the solutions in three different scenarios in its worst and best case.

The rest of the paper is organized as follows. Best Order Sort (BOS) is briefly described in Section 2. The proposed generalized version of BOS (GBOS)

[1] https://github.com/Proteek/Best-Order-Sort/blob/master/BestOrderSort.java

is presented in Section 3. The complexity analysis of GBOS is provided in Section 4. Section 5 provides a detailed experimental evaluation of our proposed approach. Finally, Section 6 concludes the paper and provides some possible future research directions.

2. Brief Description of BOS

In this section we briefly discuss BOS [3] which works in two phases. The solutions are sorted in the first phase based on each objective individually. In the second phase, the actual sorting is performed. BOS uses a concept of comparison set C_s for each solution s and this comparison set is the set of objectives. Before rank assignment, the comparison set of each solution contains all the objectives. In BOS, the solutions are sorted based on each objective individually and the sorted solutions are stored in a list Q_j which stores the sorted solutions based on j^{th} objective. Initially, the first solution in each of the sorted lists is ranked, then the second solution in each of the sorted lists is ranked and so on. When a solution is explored in the sorted lists for rank assignment purposes, then its comparison set is reduced. When a solution is compared with previously ranked solutions for rank assignment, then only the objectives in the comparison set of the previously ranked solutions are compared. BOS does not provide the correct set of fronts when there are duplicate solutions in the population. This limitation is due to the use of *Comparison Set*.

2.1. Influence of Duplicate solutions on BOS

Here, we discuss the influence of duplicate solutions on BOS using an example.

Example 1. *Consider $\mathbb{P} = \{s_1, s_2, s_3\}$ be a population of three solutions in 2-dimensional space. Let $s_1 = \{1, 2\}$, $s_2 = \{2, 1\}$ and $s_3 = \{2, 1\}$. Thus, solutions s_2 and s_3 are identical (in terms of objective vectors). The sorted solutions based on each objective is given in Table 1. Here, when the solutions are sorted based on the first objective, then both objective values of solutions s_2 and s_3 are considered. When the solutions are sorted based on the second objective, then the sorted order of the solutions based on the first objective is used to decide the order of the solutions s_2 and s_3, because the second objective value of these two solutions is the same. Initially, the comparison set of all the three solutions are as follows: $C_{s_1} = \{1, 2\}$, $C_{s_2} = \{1, 2\}$ and $C_{s_3} = \{1, 2\}$.*

Q_1	Q_2
s_1	s_2
s_2	s_3
s_3	s_1

Table 1: Sorted solutions in population $\mathbb{P}$ based on each objective after lexicographic sorting.

Initially, solution s_1 is assigned a rank when it is found in Q_1 and objective 1 is removed from the comparison set C_{s_1}. So, we have an updated $C_{s_1} = \{2\}$.

Rank 1 *is assigned to* s_1*. After this, solution* s_2 *is ranked as it is the first solution in* Q_2*. The rank of* s_2 *is also* 1 *as there is no solution which had been ranked based on the second objective. Objective* 2 *is removed from the comparison set* C_{s_2}*. So, we have an updated* $C_{s_2} = \{1\}$*. Next,* s_2 *is found in* Q_1*. As* s_2 *has already been ranked, so objective* 1 *is removed from the comparison set of* s_2 *and we have an updated* $C_{s_2} = \{\}$*.*

When solution s_3 *is found in* Q_2*, then objective* 2 *is removed from* C_{s_3}*. So, we have an updated* $C_{s_3} = \{1\}$*. As* s_2 *has already ranked based on objective* 2*, so* s_3 *is compared with* s_2 *using the* DominationCheck(s_3, s_2) *procedure [3] which is adopted to obtain the dominance relation between two solutions. In this procedure, as the comparison set of* s_2 *is empty, i.e.,* $C_{s_2} = \emptyset$*, so the loop in lines* $1 - 5$ *of this procedure is not traversed and '*True*' is returned. This '*True*' means that solution* s_3 *is dominated by* s_2 *and hence* s_3 *has a rank of* 3*. Two identical solutions are considered as non-dominated, but here* s_3 *is considered as dominated by* s_2*. So, in case of duplicate solutions, BOS does not provide the correct set of non-dominated fronts.*

*2.2. Influence of Duplicate solutions on BOS**

The original BOS is not able to handle duplicate solutions. However, BOS* has the ability to handle them. So, now we discuss the influence of duplicate solutions in BOS*. In the first phase of BOS*, the solutions are sorted based on each of the M objectives. In this phase, when the solutions are sorted based on the first objective, then, in the presence of duplicate solutions, all the objective values of these solutions are considered when they are compared with respect to each other.

Duplicate solutions have the same order in each of the M sorted lists after the first phase. In the second phase, the solutions are assigned a rank. In BOS*, a duplicate solution is compared with respect to all the previously ranked duplicate solutions. Let all the solutions in the population be duplicate. In this case, after sorting the solutions in the first phase, all the solutions have the same order in all the M sorted lists. Thus, a solution is compared with respect to all the previously ranked solution before being assigned a rank. Therefore, the total number of dominance comparisons will be $\sum_{i=1}^{N}(i-1) = \frac{1}{2}N(N-1)$ which is the maximum number of dominance comparisons performed by BOS*. Thus, when all the solutions are duplicate, then BOS* performs the maximum number of dominance comparisons. However, duplicate solutions can be handled efficiently as in ENS-NDT [15].

3. Proposed Non-dominated Sorting Approach (GBOS)

This section describes our proposed generalized BOS. We call this generalized BOS as GBOS (Generalized Best Order Sort).

3.1. Generalization of BOS

In the process of generalizing BOS, we have retained all the steps of BOS. To handle duplicate solutions and also keeping both advantages of BOS, we have identified duplicate solutions before assigning rank to the solutions. Identifying duplicate solutions helps in handling them in an efficient manner without losing any of the advantages of BOS. To further improve the time complexity of generalized BOS, we have considered a binary search based technique to identify the rank of a solution. The binary search based technique is also discussed in other approaches like ENS-BS [9], DCNS-BS [11] and ENS-NDT [15]. GBOS is similar to BOS and, therefore, the notations used to define the algorithms are the same as in BOS. The changes to BOS are highlighted in the algorithms.

3.2. Proposed Approach

The steps of GBOS are summarized in Algorithm 1. Same as BOS [3], here, we also have $N \times M$ empty sets which are represented by L and are initialized to $\emptyset$. These $N \times M$ empty sets can be considered as a matrix of size $N \times M$ where each cell represents an empty set. A cell in this matrix which is in the ith row and the jth column is denoted by L^i_j. L^r_j represents the set of solutions which have rank r and have been ranked based on the jth objective. For each solution s in the population, a set of objectives known as comparison set C_s is associated. The goal of the comparison set is to reduce the number of comparisons between the objectives when two solutions are compared. If a solution s is checked whether it is dominated by solution t or not, then only the objectives present in C_t need to be compared. A variable *isRanked*(s) is associated with each solution s to keep track of whether solution s is ranked or not.

Algorithm 1 INITIALIZATION OF GBOS

Input: Population $\mathbb{P}$ of size N and objective M
Output: Ranked solutions

// global variables
$L^i_j \leftarrow \emptyset, \forall j = 1, 2, \ldots, M, \forall i = 1, 2, \ldots, N$ *// Stores the rank i solutions which have been ranked based on j^{th} objective*
$C_s \leftarrow \{1, 2, \ldots, M\}, \forall s \in \mathbb{P}$ *// Comparison set*
isRanked$(s) \leftarrow$ FALSE$, \forall s \in \mathbb{P}$ *// Solutions ranked or not*
sameAs$(s) \leftarrow \Phi, \forall s \in \mathbb{P}$ *// Previous solution if it is same as s*
$SC \leftarrow 0$ *// Number of solutions already ranked*
$RC \leftarrow 1$ *// Number of fronts discovered so far*
$R(s) \leftarrow 0, \forall s \in \mathbb{P}$ *// Rank of solutions*
for $i \leftarrow 1$ to M **do**
  $Q_j \leftarrow$ Sort the population $\mathbb{P}$ based on j^{th} objective

A variable *sameAs*(s) is associated with each solution s to keep track of its previous solution if it is the same as s. This helps in removing the limitation of BOS and in reducing the number of dominance comparisons in the presence of duplicate solutions in the population. The number of ranked solutions is stored

in variable SC. Initially, no solution is ranked, so variable SC is initialized to zero. Variable RC is used to store the number of already obtained fronts. The rank of a solution s is stored in variable $R(s)$.

GBOS works in two phases. In the first phase, the solutions are sorted based on each objective, considered separately. Solutions are sorted based on a lexicographic order. The sorted solutions based on the jth objective are stored in a list Q_j. The ith solution in the sorted list Q_j is represented as $Q_j(i)$. When solutions are sorted based on the first objective, if the value of the first objective is the same for two solutions, then the value of the second objective is considered to find the sorted order. If the value of the second objective is the same for two solutions, then the value of the third objective is considered to find the sorted order. In general, if the value of the jth objective is the same for two solutions, then the value of the $j+1^{th}$ objective is considered to find the sorted order. If two solutions are identical, *i.e.*, if they have the same value for all the objectives, then any order can be considered. However, when the solutions are sorted based on other objectives, except the first and if the value of two solutions is the same for a particular objective, then sorted order between the solutions is decided depending on the sorted order based on the first objective. In this manner, we get M sorted lists of solutions based on each of the M objectives. The size of each sorted list is N. The solutions are sorted based on all the objectives in lines $8-9$ of Algorithm 1.

Algorithm 2 MAIN LOOP OF THE GBOS ALGORITHM

Input: List of sorted solutions, $Q_1, Q_2, \ldots, Q_M$
Output: Rank of each solution, R

```
for i ← 2 to N do
    if IsSame(Q_1(i), Q_1(i − 1)) then
        sameAs(Q_1(i)) ← Q_1(i − 1)
for i ← 1 to N do
    for j ← 1 to M do
        s ← Q_j(i)                          // Take i^th solution from Q_j
        C_s ← C_s − {j}                     // Reduce comparison set of s
        if isRanked(s) = True then          // s is already ranked
            L_j^R(s) ← L_j^R(s) ∪ {s}       // Include s to L_j^R(s)
        else
            FindRank(s, j)                  // Find the rank of s
            isRanked(s) ← True              // Rank is assigned to s
            SC ← SC + 1                     // Number of ranked solutions
    if SC = N then                          // Ranks are assigned to all solutions
        Break                               // Sorting completed
```

Once the sorted order of the solutions based on each objective is identified, then the actual rank assignment is performed. The steps of the ranked assignment are summarized in Algorithm 2. The input to this algorithm is the sorted solutions based on each objective. For rank assignment purpose, M sorted lists

based on each objective are considered as a matrix of size $N \times M$ where the jth column corresponds to the sorted solutions based on the jth objective, Q_j. We call this matrix *sorted matrix*. For rank assignment purposes, *sorted matrix* is traversed in a row-wise manner. In each row, the solutions are traversed from left to right. In this matrix, each solution occurs M times. Initially, the solutions in the first row are assigned a rank, then the solutions in the second row are assigned a rank and so on. If a solution is already ranked when it is explored in the *sorted matrix* traversal, then the solution is not ranked again. As soon as the rank is assigned to all the solutions, the *sorted matrix* traversal stops and the process of non-dominated sorting is completed.

Algorithm 3 IsSame

```
// Not Considered in BOS
Input: Solution s and t
Output: True if s and t are duplicate solutions, False otherwise
for j ← 1 to M do                          // for each objective
    if s(j) ≠ t(j) then
        return False                       // s and t are different solutions
return True                                // s and t are duplicate solutions
```

When the solutions are sorted based on all the objectives, then identical solutions have the same order in all the objectives lists. So, before traversing the *sorted matrix* row-wise for rank assignment, for each solution s in the objective list Q_1, we keep the information of whether the current solution is the same as the previous solution. If s is the same as its previous solution, then the previous solution of s is stored in variable *isSame* associated with s. To obtain this information, the objective list Q_1 is traversed and two consecutive solutions are checked to see if they are same or not. Here, two solutions are compared using IsSame() procedure which is summarized in Algorithm 3. The process is shown in lines $1-3$ of Algorithm 2.

When a solution s is explored in the jth column of the *sorted matrix*, *i.e.*, a solution is explored in Q_j, then its comparison set C_s is reduced (line 7). After this, the algorithm checks whether s is already ranked or not (line 8). In case s is already ranked (line 8), then s is added to $L_j^{R(s)}$ (line 9). Otherwise, the rank is assigned to s using Algorithm 4. Once the rank is assigned to s, it is marked as ranked (line 12).

3.3. Illustration of the FindRank() *procedure*

This procedure assigns a rank to a solution s which is first explored in the jth column of the *sorted matrix*, *i.e.*, in objective list Q_j. Initially, we check whether the current solution (say s) is the same as its previous solution (line 1). If it is the same, then the rank of the current solution is the same as the rank of its previous solution and the solution is added to $L_j^{R(s)}$. If the current solution s is not the same as the previous solution, then solution s is compared with respect to all the solutions in the existing fronts in a sequential manner. Basically,

Algorithm 4 FindRank

Input: Solution s and List number j
Output: Rank of s

```
if sameAs(s) ≠ Φ then                    // Same solution as s is previously ranked
    R(s) ← R(sameAs(s))
    L_j^{R(s)} ← L_j^{R(s)} ∪ {s}                                  // Add s to L_j^{R(s)}
else                          // Current solution is different than previous solution
    done ← False                                        // Rank is not yet assigned to s
    for k ← 1 to RC do                                          // for all discovered ranks
        check ← False                                                   // s is non-dominated
        for t ∈ L_j^k do                                          // for all solutions in L_j^k
            check ← DominationCheck(s, t)
            if check = True then                                          // s is dominated
                Break
        if check = False then                                           // s is non-dominated
            R(s) ← k                                                      // Assign rank to s
            done ← True                                              // Rank is assigned to s
            L_j^{R(s)} ← L_j^{R(s)} ∪ {s}                              // Include s to L_j^{R(s)}
            Break
    if done = False then                                // Rank is not yet assigned to s
        RC ← RC + 1                                                   // Update rank count
        R(s) ← RC                                                       // Assign rank to s
        L_j^{R(s)} ← L_j^{R(s)} ∪ {s}                                // Include s to L_j^{R(s)}
```

solution s is compared with respect to those solutions which have been assigned the rank based on the jth objective. This means that s is compared with respect to $L_j^k (1 \leq k \leq RC)$, sequentially. Solution s is compared with respect to the solutions of the existing fronts using the DominationCheck() procedure which is illustrated in Algorithm 5. As soon as s becomes non-dominated with respect to all the solutions in L_j^k, the rank of s becomes k. Once s is dominated by any of the solutions in L_j^k, solution s is compared with respect to the solutions of the next front which had been ranked based on the jth objective, *i.e.*, s is compared with respect to L_j^{k+1}. If s is compared with respect to all $L_j^k (1 \leq k \leq RC)$ and it is dominated by at least one of the solutions in all $L_j^k (1 \leq k \leq RC)$, then the rank count RC is incremented and the updated rank is assigned to s. Once the rank assignment is done, s is added to $L_j^{R(s)}$.

3.4. Illustration of DominationCheck() *procedure*

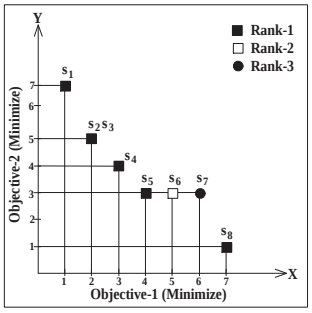
The dominance relation between two solutions s and t is obtained using the DominationCheck(s, t) procedure which is illustrated in Algorithm 5. This procedure does not compare all the objective values between the solutions. Only the objectives which are present in the comparison set of solution t are used for comparison.

Algorithm 5 DOMINATIONCHECK

Input: Solution s and t

Output: TRUE if t dominates s, FALSE otherwise

```
for each objective j ∈ C_t do                    // for all objectives in C_t
    if s(j) < t(j) then                          // s is better than t in objective j
        return FALSE                             // t cannot dominate s
return TRUE                                      // t dominates s
```



(a) Eight solutions

Q_1	Q_2
s_1	s_8
s_2	s_5
s_3	s_6
s_4	s_7
s_5	s_4
s_6	s_2
s_7	s_3
s_8	s_1

(b) Sorted list

s_1	s_8
s_2	s_5
s_3	s_6
s_4	s_7
s_5	s_4
s_6	s_2
s_7	s_3
s_8	s_1

8×2

(c) Sorted matrix

s_1	s_8	s_2	s_5	s_3	s_6	s_4	s_7
$Q_1(1)$	$Q_2(1)$	$Q_1(2)$	$Q_2(2)$	$Q_1(3)$	$Q_2(3)$	$Q_1(4)$	$Q_2(4)$

(d) Order of solutions in which they are assigned ranks.

Explored Solution	**Objective List**	**Comparison Set**	**Rank of solution**	$\boldsymbol{L_1^r}$	$\boldsymbol{L_2^r}$
				$\mathbf{1 \leq r \leq 3}$	
s_1	1	$C_{s_1}=\{2\}$	1	$L_1^1=\{s_1\}$	
s_8	2	$C_{s_8}=\{1\}$	1	$L_1^1=\{s_1\}$	$L_2^1=\{s_8\}$
s_2	1	$C_{s_2}=\{2\}$	1	$L_1^1=\{s_1, s_2\}$	$L_2^1=\{s_8\}$
s_5	2	$C_{s_5}=\{1\}$	1	$L_1^1=\{s_1, s_2\}$	$L_2^1=\{s_8, s_5\}$
s_3	1	$C_{s_3}=\{2\}$	1	$L_1^1=\{s_1, s_2, s_3\}$	$L_2^1=\{s_8, s_5\}$
s_6	2	$C_{s_6}=\{1\}$	2	$L_1^1=\{s_1, s_2, s_3\}$	$L_2^1=\{s_8, s_5\}$
					$L_2^2=\{s_6\}$
s_4	1	$C_{s_4}=\{2\}$	1	$L_1^1=\{s_1, s_2, s_3, s_4\}$	$L_2^1=\{s_8, s_5\}$
					$L_2^2=\{s_6\}$
s_7	2	$C_{s_7}=\{1\}$	3	$L_1^1=\{s_1, s_2, s_3, s_4\}$	$L_2^1=\{s_8, s_5\}$
					$L_2^2=\{s_6\}$
					$L_2^3=\{s_7\}$

(e) Procedure for sorting.

Figure 1: Example showing the GBOS procedure.

Example 2. *Let $\mathbb{P} = \{s_1, s_2, \ldots, s_8\}$ be a population of size eight in 2-dimensional space as shown in Figure 1(a). In this population, two solutions s_2 and s_3 are*

identical in terms of their objective values. Let's consider a problem such that both objectives need to be minimized. In the first phase of GBOS, the solutions are sorted based on each objective individually. Figure 1(b) shows the sorted solutions based on each of the objectives. The size of both sorted lists is eight. These two sorted lists of size eight can be considered as a matrix of size 8×2 *which we call* sorted matrix. *The sorted solutions in the form of a matrix are shown in Figure 1(c). The solutions are ranked considering each solution in a row-wise manner in this matrix. The order in which solutions are ranked is given in Figure 1(d).*

Initially, the first solution in row-1 (i.e., s_1*) is ranked, then the second solution in row-1 (i.e.,* s_8*) is ranked. As these two solutions are the first in their respective columns, they are not compared with any other solutions and the rank of these two solutions is* 1*. As solution* s_1 *is found in the first column, so the first objective is removed from its comparison set and we have an updated* $C_{s_1} = \{2\}$*. Once the rank is assigned to* s_1*,* s_1 *is added to* L_1^1 *as* s_1 *is ranked based on the first objective and its rank is* 1*. Similarly, as solution* s_8 *is found in the second column, so the second objective is removed from its comparison set and we have an updated* $C_{s_8} = \{1\}$*. Once the rank is assigned to* s_8*, it is added to* L_2^1 *as* s_8 *is ranked based on the second objective and its rank is* 1*. Now, if any other solution is compared with respect to these two solutions, then those solutions will be compared only with respect to the objectives which are present in the comparison set of these two solutions.*

Next, the solutions in row-2 are ranked if they have not already been ranked. Here, solution s_2 *is ranked and then* s_5 *is ranked. As* s_2 *is found in column-1, so* s_2 *is compared with respect to* L_1^1 *(i.e., with respect to* s_1*). When* s_2 *is compared with respect to* s_1*, then* s_2 *is compared based on only the second objective of* s_1 *because the comparison set of* s_1 *has only the second objective. As* s_2 *is non-dominated with respect to* s_1*, so the rank of* s_2 *is also* 1*. After performing the rank assignment to* s_2*, it is added to* L_1^1 *as* s_2 *is ranked based on the first objective and its rank is* 1*. Similarly, as* s_5 *is found in column-2, so* s_5 *is compared with respect to* L_2^1 *(i.e., with respect to* s_8*). When* s_5 *is compared with respect to* s_8*, then* s_5 *is compared based on only the first objective of* s_8 *because the comparison set of* s_8 *has only the first objective. As* s_5 *is non-dominated with respect to* s_8 *so the rank of* s_5 *is also* 1*. Once the rank is assigned to* s_5*, it is added to* L_2^1 *as* s_5 *is ranked based on the second objective and its rank is* 1*.*

All these four solutions $\{s_1, s_8, s_2, s_5\}$ *are ranked using the* FINDRANK() *procedure explained in Algorithm 4. Next, the solutions in row-3 are ranked if they have not already been ranked. First, solution* s_3 *is ranked. When solution* s_3 *needs to be ranked, then it is not compared with respect to any other solution. As* s_3 *is the same as* s_2*, so the rank of* s_2 *is directly assigned to* s_3*. Thus, the rank of* s_3 *is also* 1*. Once the rank is assigned to* s_3*,* s_3 *is added to* L_1^1 *as it is ranked based on the first objective and its rank is* 1*. Now, solution* s_6 *is ranked. As* s_6 *is found in column-2, so* s_6 *is compared with respect to* L_2^1 *(i.e., with respect to* $\{s_8, s_5\}$*). When* s_6 *is compared with respect to* $\{s_8, s_5\}$*, then* s_6 *is compared based on only the first objective of* $\{s_8, s_5\}$ *because the comparison sets of these two solutions have only the first objective. As* s_6 *is dominated*

by s_5, so s_6 cannot have the same rank as $\{s_8, s_5\}$. There is no solution with rank two which had been ranked based on the second objective, so the rank of s_6 becomes 2. Once the rank is assigned to s_6, it is added to L_2^2 as s_6 is ranked based on the second objective and its rank is 2.

In the same manner, solutions s_4, s_7 are ranked using the FindRank() *procedure. The complete process of assigning rank to all the eight solutions is shown in Figure 1(e).*

Algorithm 6 FindRankBinary

// **New in GBOS: Not considered in BOS**

Input: Solution s and List number j

Output: Rank of s

```
if sameAs(s) ≠ Φ then                          // Same solution as s is previously ranked
    R(s) ← R(sameAs(s))
    L_j^{R(s)} ← L_j^{R(s)} ∪ {s}                                   // Include s to L_j^{R(s)}
else                                   // Current solution is different than previous solution
    min ← 1, max ← RC, mid ← ⌊(min+max)/2⌋
    while True do                                          // Position of s is not identified
        check ← False                                                      // s is non-dominated
        for t ∈ L_j^{mid} do                                        // for all solutions in L_j^{mid}
            check ← Domination-Check(s, t)
            if check = True then                                              // s is dominated
                Break
        if check = False then                                               // s is non-dominated
            if mid = min then                                   // The front at leaf is explored
                R(s) ← mid                                                      // Assign rank to s
                L_j^{R(s)} ← L_j^{R(s)} ∪ {s}                                  // Include s to L_j^{R(s)}
                Break
            else
                max ← mid − 1, mid ← ⌊(min+max)/2⌋                          // Explore left sub-tree
        else
            if min = RC then                                        // Right most leaf is explored
                RC ← RC + 1                                                   // Update rank count
                R(s) ← RC                                                       // Assign rank to s
                L_j^{R(s)} ← L_j^{R(s)} ∪ {s}                                  // Include s to L_j^{R(s)}
                Break
            else if mid = max then
                R(s) ← max + 1                                                  // Assign rank to s
                L_j^{R(s)} ← L_j^{R(s)} ∪ {s}                                  // Include s to L_j^{R(s)}
                Break
            else
                min ← mid + 1, mid ← ⌊(min+max)/2⌋                          // Explore right sub-tree
```

In the FindRank() procedure, the rank of a solution is obtained by com-

paring it with respect to previously ranked solutions in a sequential manner. As all the ranked solutions are arranged in decreasing order of their dominance in L, so the binary search based technique can be used to find the rank of a solution. The binary search based technique is also used in some of the previously proposed approaches like [9], [11], [15]. We have used a binary search based technique because the time complexity can be reduced in some of the cases using this technique [9]. Although this technique is a bit more complicated than the FindRank() procedure, its computational efficiency led us to adopt it, as explained in the following subsection.

3.5. Illustration of FindRankBinary() *procedure*

As the ranked solutions are arranged in decreasing order of dominance, if a solution which needs to be ranked (say s) based on a particular objective (say the jth) is dominated by the solution (which have been ranked based on the jth objective) of a particular front, then there is no need to compare s with respect to the solutions of higher dominance fronts. This is because if a solution is dominated by the solution of a particular front, then the same solution will also be dominated by at least one of the solutions in all the higher dominance fronts. Similarly, if a solution which needs to be ranked (say s) based on a particular objective (say the jth) is non-dominated with respect to all the solutions of a particular front which have been ranked based on the jth objective, then there is no need to compare s with respect to the solutions of the lower dominance fronts. This is because if a solution is non-dominated with respect to all the solutions of a particular front, then the same solution can also be non-dominated with respect to the solutions in the higher dominance fronts. In this way, as a solution is compared with respect to the solutions of a particular front, we discard half of the fronts. So, the solution s is compared with respect to only $\lceil \log(RC+1) \rceil$ fronts. The procedure to assign the rank to a solution using the binary search based technique is summarized in Algorithm 6.

The procedure to find the rank of a solution is based on binary search and therefore, a tree structure is used. Here, the tree is not explicitly created, but instead the set of ranked solutions which are inserted in L are visualized as a tree. As the tree structure is used, we are considering two variables *min* and *max* to follow the tree structure. Initially, *min* is set to 1 and *max* is set to RC as the obtained number of fronts is RC. The solution which needs to be ranked is first compared with respect to L_j^{mid} where $mid = \lfloor \frac{min+max}{2} \rfloor$. If the solution s is not dominated by any other solution of L_j^{mid} that means that s cannot have a rank worse than *mid*. However, it can have a better rank than *mid*. As s is non-dominated with respect to all the solutions of L_j^{mid} so it can also be non-dominated by better ranked solutions. So, if s is non-dominated with respect to all the solutions of L_j^{mid}, then there are two possibilities:

- If leaf of the tree is reached (*i.e.*, $mid = min$), then the rank of s is *mid* and s is inserted into L_j^{mid}.
- Otherwise, the root of the left sub-tree is checked.

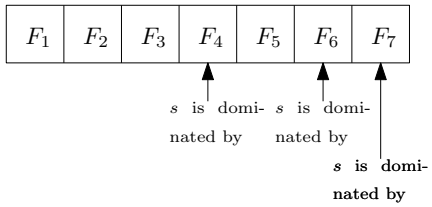


Figure 2: Example showing the binary search based technique to find the rank of a solution.

If s is dominated by any of the solutions of L_j^{mid}, *i.e.*, s cannot have a better rank than *mid*, then there are three possibilities:

- If the rightmost node of the tree is reached (*i.e.*, $min = RC$), then s is dominated by all the existing ranked solutions and the rank of s is $RC+1$.
- If s is dominated by the leaf node, (*i.e.*, $mid = max$), then the rank of s is $max + 1$.
- Otherwise, the root of the right sub-tree is checked.

Example 3. *Let's assume that we have seven non-dominated fronts and the solutions in these fronts are ranked based on a particular objective (say the jth). Let's assume a new solution s needs to be ranked based on the jth objective. In the* FindRank() *procedure, s needs to be compared with respect to the solutions of all the seven fronts sequentially. However, in the* FindRankBinary() *procedure, the tree-based structure is followed.*

Here, s is first compared with respect to the solutions of F_4. Let s be dominated by one of the solutions of F_4. So, now s is compared with respect to the solutions of F_6. Let s be dominated by one of the solutions of F_6. So, now s is compared with respect to the solutions of F_7. Let's consider that s is dominated by one of the solutions of F_7. Hence, s is dominated by the solutions of all the fronts so, s will make a new front F_8. This process is shown in Figure 2.

Depending on the search technique used for assigning rank to a solution, there are two variants of GBOS. The first one is GBOS with sequential search (GBOS-SS) and the second one is GBOS with binary search (GBOS-BS).

4. Complexity Analysis

In the first phase of GBOS, solutions are sorted based on each objective using heap sort [22] as in [3]. The auxiliary space required by heap sort is $\mathcal{O}(1)$. In GBOS, M sorted lists, each of size N, are used to store the sorted solutions based on each of the M objectives. This requires $\mathcal{O}(MN)$ space. Also, $N \times M$ sets are considered which also require $\mathcal{O}(MN)$ space. Thus, the space complexity of GBOS is $\mathcal{O}(MN)$.

The time complexity of GBOS, $T(N, M)$ depends on the time complexities of its three main steps: (1) sorting of the solutions based on each objective individually, $T_{\text{presort}}(N, M)$, (2) checking for duplicate solutions (lines $1 - 3$ of

Algorithm 2), $T_{\text{duplicate}}(N, M)$ and (3) assigning rank to the solutions (rank assignment), $T_{\text{assign}}(N, M)$.

$$T(N, M) = T_{\text{presort}}(N, M) + T_{\text{duplicate}}(N, M) + T_{\text{assign}}(N, M) \tag{1}$$

Let's assume that heap sort [22] be used in the first phase to sort the solutions based on each objective. Heap sort is also used in [9], [11] to sort the solutions based on the first objective. The worst case time complexity to sort the solutions based on the first objective is $\mathcal{O}(MN \log N)$ because in this case all the objective values of the solutions can be considered for lexicographic sorting. The time complexity to sort the solutions based on other objectives except the first is $\mathcal{O}(N \log N)$ because in this case the sorted order based on the first objective is considered when the solutions have the same value for a particular objective. Thus, the time complexity of the first phase of the algorithm, *i.e.*, $T_{\text{presort}}(N, M)$, is given by Eq. (2).

$$T_{\text{presort}}(N, M) = \mathcal{O}(MN \log N) + (M-1)\mathcal{O}(N \log N) = \mathcal{O}(MN \log N) \tag{2}$$

In GBOS, after sorting the solutions based on all the objectives, duplicate solutions are checked in lines $1-3$ of Algorithm 2. The number of dominance comparisons required to check for duplicate solutions is given by Eq. (3) as the ith solution is compared with respect to only the $i-1^{th}$ solution in the sorted list Q_1 where $2 \leq i \leq N$. When duplicate solutions are checked, then in the worst case all the objective values between the solutions can be considered. Hence, the maximum number of objective value comparisons required to check for duplicate solutions is given by Eq. (4). So, the worst case time complexity of checking for duplicate solutions, *i.e.*, $T_{\text{duplicate}}(N, M)$, is given by Eq. (5).

$$\#\text{dcmp}_{\text{duplicate}} = N - 1 \tag{3}$$

$$\#\text{cmp}_{\text{duplicate}} = M(N-1) \tag{4}$$

$$T_{\text{duplicate}}(N, M) = \mathcal{O}(MN) \tag{5}$$

In the first phase of GBOS, solutions are sorted based on each objective individually. Two solutions are compared in the first phase based on one of the objectives. However, when the solutions are sorted based on the first objective, then all the objective values of the solutions can be considered when the solutions are identical. The number of objective value comparisons performed in the first phase by ENS-SS [9], ENS-BS [9], BOS, BOS* and GBOS depends on the sorting algorithm used and also on the objective values of the solutions. In the first phase, BOS, BOS* and GBOS sort the solutions based on each of the M objectives. So, the first phase of these three approaches is the same. Thus, if the same sorting algorithm is used in the first phase of BOS, BOS* and GBOS to sort the solutions, then the number of objective value comparisons performed by these three algorithms in the first phase will also be the same. So, we theoretically calculate the number of objective value comparisons performed by GBOS in checking duplicate solutions and in assigning rank to the

solutions. The number of objective value comparisons performed by BOS* is also calculated in the rank assignment phase.

The total number of dominance comparisons performed by GBOS corresponds to the sum of the number of dominance comparisons required to check the duplicate solutions and the number of dominance comparisons required for rank assignment. Similarly, the number of objective value comparisons is the sum of the number of objective value comparisons required to check for duplicate solutions and the number of objective value comparisons needed for rank assignment. Let $\#\text{dcmp}_{\text{assign}}$ be the number of dominance comparisons for rank assignment and $\#\text{cmp}_{\text{assign}}$ be the number of objective value comparisons for rank assignment. Thus, the total number of dominance comparisons is given by Eq. (6) and the total number of objective value comparisons is given by Eq. (7).

$$\#\text{dcmp} = \#\text{dcmp}_{\text{duplicate}} + \#\text{dcmp}_{\text{assign}} \tag{6}$$

$$\#\text{cmp} = \#\text{cmp}_{\text{duplicate}} + \#\text{cmp}_{\text{assign}} \tag{7}$$

In subsequent sections, we discuss the time complexities of GBOS-SS and GBOS-BS in different scenarios. Some of the examples of the behavior of GBOS-SS and GBOS-BS in different scenarios are illustrated in the Appendix.

4.1. All solutions are in a single front

The time complexity of GBOS when all the solutions are non-dominated with respect to each other, is discussed in this section. In case of having a single front, GBOS-SS and GBOS-BS perform the same.

4.1.1. All the solutions are duplicate

In this case, the order of the solutions in all Q_j after pre-sorting is the same. So, each column of a particular row of the *sorted matrix* has the same solution. Thus, the rank is assigned to a solution when it is explored in the first column of the *sorted matrix*, *i.e.*, in Q_1. When a solution is explored in other columns of the *sorted matrix*, then its comparison set is reduced and is added to the set L.

Initially, solution $Q_1(1)$ is assigned rank 1 without comparing with any other solution. When other solutions are ranked, then, in the FindRank() procedure, only lines $1-3$ are used because the kth solution which needs to be ranked is always the same as the $k-1^{th}$ ranked solution in Q_1. These three lines take constant time for rank assignment. Thus, there is no dominance comparison between the solutions in the FindRank() procedure. Hence, the total number of dominance comparisons is the same as the number of dominance comparisons required to check the duplicate solutions and it is obtained by Eq. (3). Hence, the number of objective value comparisons is obtained by Eq. (4).

The FindRank() procedure takes constant time when it is used to assign rank to a solution. When an already ranked solution is explored in the *sorted matrix*, then the solution is added to L which also takes constant time. Thus,

the time complexity of the rank assignment step is $\mathcal{O}(MN)$. Hence, the time complexity of GBOS-SS and GBOS-BS is

$$\begin{aligned} T(N,M) &= T_{\text{presort}}(N,M) + T_{\text{duplicate}}(N,M) + T_{\text{assign}}(N,M) \\ &= \mathcal{O}(MN\log N) + \mathcal{O}(MN) + \mathcal{O}(MN) = \mathcal{O}(MN\log N) \end{aligned} \tag{8}$$

ENS-NDT [15] also performs $N-1$ number of dominance comparisons in this case. However, the time complexity of ENS-NDT in this case is $\mathcal{O}(MN\log N)+\mathcal{O}(N\log^2 N)+\mathcal{O}(MN) = \mathcal{O}(MN\log N + N\log^2 N)$ because of pre-sorting, split phase and the rank assignment phase. BOS* performs $\sum_{i=1}^{N}(i-1) = \frac{1}{2}N(N-1)$ dominance comparisons in this case and the number of objective value comparisons is $\frac{1}{2}MN(N-1)$. ENS-SS [9] and ENS-BS [9] also perform $\frac{1}{2}N(N-1)$ dominance comparisons when this type of case arises. T-ENS [13] is not able to handle this case.

4.1.2. Worst case

The dominance nature between two solutions can be decided with the help of only two objectives also. If all the solutions are in a single front, then the maximum number of dominance comparisons occurs when the 1st to the $M-2^{th}$ objective values of each of the solutions are the same. The last two objective values of each of the solutions should be such that they are able to declare all the solutions as non-dominated.

In this case, the order of the solutions in the initial $M-1$ objective lists are the same and in the last list, it is just reverse. This means that for a particular row of the *sorted matrix*, the initial $M-1$ columns have the same solution and the last column has a different solution. Thus, a solution is ranked when it is explored either in the first column (objective list Q_1) or in the last column (objective list Q_M) of the *sorted matrix*. When a solution is explored in the 2nd to the $M-1^{th}$ column, then it will not be ranked because the same solution has already been ranked when it was explored in the first column. In each row of the *sorted matrix*, only two solutions are ranked (if not already ranked): one from the first column and another from the last column and there are N solutions. Thus, all the solutions are explored for rank assignment in the initial $N/2$ rows of the *sorted matrix*. Let a solution s be explored the first time in the ith row and the $j^{th}(j \in \{1, M\})$ column of the *sorted matrix*. For rank assignment, this solution is compared with respect to the solutions of the previous $i-1$ rows which have been ranked based on the jth objective. Thus, the number of dominance comparisons is given by Eq. (9).

When a solution is explored in the first objective list, then it is ranked and the first objective is removed from its comparison set. When the same solution is explored in the 2nd to $M-1^{th}$ objective lists, then the 2nd to the $M-1^{th}$ objectives are removed from the comparison set. So, next time, when a solution is ranked based on the first objective, then the solution is compared based only on the last objective as only the last objective remains in the comparison set of the solutions which have already been ranked based on the first objective. Similarly, when a solution is explored in the last objective, then it is ranked

and the last objective is removed from its comparison set. Therefore, next time, when a solution is ranked based on the last objective, then the solution is compared based on the 1st to the $M-1^{th}$ objectives as only the last objective is removed from the comparison set of solutions which have been ranked based on the last objective. Thus, the number of objective value comparisons is given by Eq. (10).

$$\begin{aligned}\#\text{dcmp} &= \#\text{dcmp}_{\text{duplicate}} + \#\text{dcmp}_{\text{assign}} \\ &= N-1+\sum\nolimits_{i=1}^{N/2}(i-1)+(i-1) = N-1+\frac{1}{4}N(N-2) \qquad (9)\\ \#\text{cmp} &= \#\text{cmp}_{\text{duplicate}} + \#\text{cmp}_{\text{assign}} \\ &= M(N-1)+\sum\nolimits_{i=1}^{N/2}(i-1)(1)+(i-1)(M-1) \\ &= M(N-1)+\frac{1}{8}MN(N-2) \qquad (10)\end{aligned}$$

Thus, the time complexity of rank assignment, *i.e.*, $T_{\text{assign}}(N,M)$, is $\mathcal{O}(MN^2)$. Hence, the time complexity of GBOS-SS and GBOS-BS in this case is

$$\begin{aligned}T(N,M) &= T_{\text{presort}}(N,M)+T_{\text{duplicate}}(N,M)+T_{\text{assign}}(N,M) \\ &= \mathcal{O}(MN\log N)+\mathcal{O}(MN)+\mathcal{O}(MN^2) = \mathcal{O}(MN^2) \qquad (11)\end{aligned}$$

ENS-NDT performs $N-1+\frac{1}{2}N(N-1)$ dominance comparisons in this case with a bucket size of one and $M \geq \log N+2$ [15]. The number of objective value comparisons performed by ENS-NDT in the rank assignment phase in this case is $\frac{1}{2}(M-1)N(N-1)$. So, the time complexity of ENS-NDT in this case is $\mathcal{O}(MN\log N)+\mathcal{O}(N\log^2 N)+\mathcal{O}(MN^2) = \mathcal{O}(MN^2)$ because of pre-sorting, split phase and the rank assignment phase. As explained in [15], this is the worst case for ENS-NDT. BOS* performs $\sum_{i=1}^{N/2}(i-1)+(i-1) = \frac{1}{4}N(N-2)$ dominance comparisons in this case and the number of objective value comparisons is $\frac{1}{4}MN(N-2)$. In this case, the number of dominance comparisons performed by GBOS-SS and GBOS-BS is $N-1$ more than those performed by BOS* because of the checking of duplicate solutions in GBOS-SS and GBOS-BS. However, the number of objective value comparisons performed by GBOS-SS and GBOS-BS is less than those performed by BOS* because of the use of the comparison set concept in GBOS-SS and GBOS-BS.

4.1.3. Best case

When all the solutions are in a single front, then best case of GBOS occurs if the row-wise traversal of the solutions in the *sorted matrix* follows a specific pattern. The traversal should be such that before the second occurrence of a solution, all the solutions must be traversed at least once. Without loss of generality, let us assume $N = 2^a$ and $M = 2^b$ where $a, b \geq 1$.

Here, before the second occurrence of a solution, all the solutions should had been traversed at least once. So, in each row of the *sorted matrix*, M solutions are ranked and there are N solutions. So, the solutions are explored for rank

assignment in the initial N/M rows (in case $M > N$, only the solutions in the first row are explored). The solution which is explored in the ith row and the jth column of the *sorted matrix* is compared with the solutions in the previous $i-1$ rows of the *sorted matrix* which have been ranked based on the jth objective. Thus, the number of dominance comparisons is given by Eq. (12). In this case, as the number of objectives increases, the number of dominance comparisons decreases. When $M \geq N$, then the number of dominance comparisons becomes zero, because of the rank assignment.

In the best case, before the second occurrence of a solution, all the solutions should had been traversed at least once. So, each time a solution is ranked, its comparison set is reduced by one objective and before a solution occurs a second time, the process of rank assignment completes. So, whenever a solution is compared with respect to other solutions, a maximum of $M-1$ objectives can be considered regardless of M. Thus, the maximum number of objective value comparisons is given by Eq. (13).

$$\begin{aligned}
\#\text{dcmp} &= \#\text{dcmp}_{\text{duplicate}} + \#\text{dcmp}_{\text{assign}} \\
&= N - 1 + \sum\nolimits_{i=1}^{N/M} M(i-1) = N - 1 + \frac{1}{2M} N\,(N-M) \qquad (12) \\
\#\text{cmp} &= \#\text{cmp}_{\text{duplicate}} + \#\text{cmp}_{\text{assign}} \\
&= M(N-1) + \sum\nolimits_{i=1}^{N/M} M(i-1)(M-1) \\
&= M(N-1) + \frac{1}{2M} N(M-1)(N-M) \qquad (13)
\end{aligned}$$

When $M \geq N$, the solutions in the first row of the *sorted matrix* are explored and ranked. Thus, the time complexity of the rank assignment when $M \geq N$, *i.e.*, $T_{\text{assign}}(N, M)$ is $\mathcal{O}(N)$. Hence, the time complexity of GBOS-SS and GBOS-BS in this case is

$$\begin{aligned}
T(N, M) &= T_{\text{presort}}(N, M) + T_{\text{duplicate}}(N, M) + T_{\text{assign}}(N, M) \\
&= \mathcal{O}(MN \log N) + \mathcal{O}(MN) + \mathcal{O}(N) = \mathcal{O}(MN \log N) \qquad (14)
\end{aligned}$$

BOS* performs $\sum_{i=1}^{N/M} M(i-1) = \frac{1}{2M} N(N-M)$ dominance comparisons in this case and the number of objective value comparisons is $\frac{1}{2} N(N-M)$. In this case, the number of dominance comparisons performed by GBOS is $N-1$ more than BOS* because of the checking of duplicate solutions in GBOS.

4.2. All solutions are in separate fronts

The time complexity when all the solutions are in separate fronts is analyzed here. GBOS-SS and GBOS-BS perform differently for this scenario.

In this scenario, the order of the solutions in all Q_j after pre-sorting is the same, *i.e.*, each column in a particular row of the *sorted matrix* has the same solution. So, each of the solutions is ranked when it is explored in the first column of a particular row. When the same solution is explored in other columns, its comparison set reduces and finally it becomes empty after exploring

the solution in all the columns. As the comparison set of ranked solutions is empty, when the solutions are compared for rank assignment, there is no objective value comparison. So, the solutions are compared for rank assignment in $\mathcal{O}(1)$ time.

4.2.1. GBOS-SS

The rank of the ith solution is obtained by comparing it with respect to the $i-1$ previously ranked solutions which have been assigned ranks based on the first objective. However, two solutions are compared for rank assignment in $\mathcal{O}(1)$ time because of the empty comparison set of the already ranked solutions. Thus, the total number of dominance comparisons is the same as the number of dominance comparisons required to check for duplicate solutions which is given by Eq. (3). The number of objective value comparisons is given by Eq. (4).

In the rank assignment process, there is no dominance comparison. However, for rank assignment, the ith solution is compared with respect to all the previously ranked $i-1$ solutions. So, the time taken by the rank assignment process is obtained by Eq. (15).

$$T_{\text{assign}}(N,M) = \sum\nolimits_{i=1}^{N}(i-1) = \frac{1}{2}N(N-1) = \mathcal{O}(N^2) \tag{15}$$

The time complexity of GBOS-SS is

$$\begin{aligned} T(N,M) &= T_{\text{presort}}(N,M) + T_{\text{duplicate}}(N,M) + T_{\text{assign}}(N,M) \\ &= \mathcal{O}(MN\log N) + \mathcal{O}(MN) + \mathcal{O}(N^2) = \mathcal{O}(MN\log N + N^2) \end{aligned} \tag{16}$$

4.2.2. GBOS-BS

The rank of the ith solution is obtained by comparing it with respect to the $\lceil \log i \rceil$ previously ranked solutions which have been ranked based on the first objective. However, two solutions are compared for rank assignment in $\mathcal{O}(1)$ time because of the empty comparison sets of the already ranked solutions. Thus, the total number of dominance comparisons is the same as the number of dominance comparisons required to check for duplicate solutions. The corresponding expression is given by Eq. (3). The number of objective value comparisons is given by Eq. (4).

In the rank assignment process, there is no dominance comparison. However, for rank assignment, the ith solution is compared with respect to the $\lceil \log i \rceil$ previously ranked solutions. So, the time taken by the rank assignment process is obtained by Eq. (17).

$$T_{\text{assign}}(N,M) = \sum\nolimits_{i=1}^{N}\lceil \log i \rceil = N\log N - (N-1) = \mathcal{O}(N\log N) \tag{17}$$

The time complexity of GBOS-BS is

$$\begin{aligned} T(N,M) &= T_{\text{presort}}(N,M) + T_{\text{duplicate}}(N,M) + T_{\text{assign}}(N,M) \\ &= \mathcal{O}(MN\log N) + \mathcal{O}(MN) + \mathcal{O}(N\log N) = \mathcal{O}(MN\log N) \end{aligned} \tag{18}$$

The time complexity of ENS-BS is $\mathcal{O}(MN \log N)$ and for ENS-SS is $\mathcal{O}(MN^2)$ in this case. ENS-SS [9] performs $\frac{1}{2}N(N-1)$ dominance comparisons and ENS-BS performs $N \log N - (N-1)$ dominance comparisons. T-ENS [13] requires $\frac{1}{2}N(N-1)$ dominance comparisons in this case. ENS-NDT performs $N \log N$ dominance comparisons and its time complexity is $\mathcal{O}(MN \log N)$ if $M \geq \log N$; otherwise, it is $\mathcal{O}(N \log^2 N)$. BOS* performs $\frac{1}{2}N(N-1)$ dominance comparisons in this case and the number of objective value comparisons is $\frac{1}{2}MN(N-1)$. The time complexity of BOS* in this case is $\mathcal{O}(MN^2)$.

4.3. N solutions are equally divided into $\sqrt{N}$ fronts

We analyze the time complexity when there is an equal division of N solutions into $\sqrt{N}$ fronts such that each solution in a front dominates all the solutions in its succeeding fronts. Here, GBOS-SS and GBOS-BS perform differently as the number of fronts is $\sqrt{N}$. When the *sorted matrix* is traversed in a row-wise manner, then before the occurrence of a solution in the kth front, all the solutions of the $(k-1)^{th}$ front have been traversed in all the columns. This is because each solution in a front dominates all the solutions in its succeeding front. Thus, before exploring the solutions of the kth front, the comparison set of all the solutions of the $(k-1)^{th}$ front is empty. Thus, when a solution of the kth front is compared with the solutions of the previous fronts, then there is no objective value comparison and the comparison between the solutions of different fronts takes $\mathcal{O}(1)$ time.

Let us assume that a solution s be first explored in the mth column (m^{th} objective list) of the *sorted matrix*. This solution is assigned a rank k in two steps – step (a) and step (b) which are discussed as follows.

(a) In case of GBOS-SS, s is dominated by one of the solutions in each of the previous $k-1$ fronts which have been ranked based on the mth objective. In the case of GBOS-BS, it is dominated by one of the solutions in only the $\lceil \log k \rceil$ previous fronts which have been ranked based on the mth objective. s is compared with only one solution in its previous fronts and s is also dominated by that particular solution. This is because, each solution in a front is dominated by all the solutions in its preceding fronts.

(b) s is non-dominated with respect to all the previous solutions in the kth front which have been ranked based on the mth objective for GBOS-SS and GBOS-BS.

In this situation, BOS and BOS* also assign the rank to a solution in the aforementioned two steps. Without loss of generality, let us assume $N = 2^{2a}$ and $M = 2^b$ where $a, b \geq 1$.

4.3.1. Solutions in the fronts are the same

Here, all the solutions in a particular front are the same. In this case, the order of the solutions in all Q_j after pre-sorting is the same. So, each column of a particular row of the *sorted matrix* has the same solution. Thus, the rank is assigned to a solution when it is explored in the first column of the *sorted*

matrix, *i.e.*, in Q_1. When a solution in explored in other columns of the *sorted matrix*, then its comparison set is reduced and the solution is added to the set, L.

In this case, as all the solutions in a particular front are the same, only the first solution of each front is compared with respect to one of the solutions of the previous fronts. The rest of the solutions of that particular front are not compared with respect to the solutions of the previous fronts. This is because, in the FindRank() or FindRankBinary() procedures when the rest of the solutions of a front are considered for rank assignment, only lines $1-3$ are used to check for duplicate solutions.

There are $\sqrt{N}$ fronts, so the first solution of the kth front is compared with one solution of all the previous $k-1$ fronts in case of GBOS-SS. In the case of GBOS-BS, the first solution of the kth front is compared with respect to one solution of the previous $\lceil \log k \rceil$ fronts. The comparison between the solutions of two different fronts takes $\mathcal{O}(1)$ time. Thus, the time complexity of GBOS-SS and GBOS-BS, because of step (a), is given by Eqs. (19) and (20), respectively.

$$T_{a_{\text{SS}}} = \sum_{k=1}^{\sqrt{N}} (k-1) = \frac{1}{2}\sqrt{N}(\sqrt{N}-1) = \mathcal{O}(N) \tag{19}$$

$$T_{a_{\text{BS}}} = \sum_{k=1}^{\sqrt{N}} \lceil \log k \rceil = \sqrt{N}\log\sqrt{N} - (\sqrt{N}-1) = \mathcal{O}(\sqrt{N}\log N) \tag{20}$$

When a solution, which is explored in the first column of the *sorted matrix* is assigned to the kth front, then it should be compared with respect to all the previous solutions of the kth front which have been ranked based on the first objective. As all the solutions in a front are the same, so when the FindRank() or FindRankBinary() procedures are adopted, then lines $1-3$ are used. Thus, the FindRank() or FindRankBinary() procedures take constant time. Hence, the time complexity of GBOS-SS and GBOS-BS, because of step (b), is given by Eq. (21).

$$T_b = \sum_{k=1}^{\sqrt{N}} \left[\sum_{i=2}^{\sqrt{N}} 1 \right] = \sqrt{N}(\sqrt{N}-1) = \mathcal{O}(N) \tag{21}$$

Thus, the time taken by the rank assignment process using GBOS-SS is obtained by Eq. (22).

$$T_{\text{assign}}(N,M) = T_{a_{\text{SS}}} + T_b = \mathcal{O}(N) + \mathcal{O}(N) = \mathcal{O}(N) \tag{22}$$

Hence, the time complexity of GBOS-SS is

$$\begin{aligned} T(N,M) &= T_{\text{presort}}(N,M) + T_{\text{duplicate}}(N,M) + T_{\text{assign}}(N,M) \\ &= \mathcal{O}(MN\log N) + \mathcal{O}(MN) + \mathcal{O}(N) = \mathcal{O}(MN\log N) \end{aligned} \tag{23}$$

The time taken by the rank assignment process using GBOS-BS is obtained by Eq. (24).

$$T_{\text{assign}}(N,M) = T_{a_{\text{BS}}} + T_b = \mathcal{O}(\sqrt{N}\log N) + \mathcal{O}(N) = \mathcal{O}(N) \tag{24}$$

Hence, the time complexity of GBOS-BS is

$$\begin{aligned} T(N,M) &= T_{\text{presort}}(N,M) + T_{\text{duplicate}}(N,M) + T_{\text{assign}}(N,M) \\ &= \mathcal{O}(MN\log N) + \mathcal{O}(MN) + \mathcal{O}(N) = \mathcal{O}(MN\log N) \end{aligned} \tag{25}$$

ENS-NDT requires $N-1+\frac{1}{2}\sqrt{N}\log N-(\sqrt{N}-1)$ dominance comparisons in this case. However, the number of dominance comparisons performed by GBOS-SS and GBOS-BS is $N-1$ which is used to check for duplicate solutions.

When all the solutions in a particular front are the same, then the order of the solutions in all the sorted lists is the same. Thus, the rank is assigned to a solution when it is explored in the first column of the *sorted matrix*. Here, in step (a), a solution which needs to be assigned a rank k is compared and dominated by one solution of each of the previous $k-1$ fronts, so the number of dominance comparisons performed by BOS*, because of step (a), is $\sum_{i=1}^{\sqrt{N}}\sqrt{N}(i-1) = \frac{1}{2}N(\sqrt{N}-1)$. The solution is compared and dominated by only one solution, because each solution in the kth front is dominated by all the solutions in its preceding front. In step (b), the solution is compared and non-dominated with respect to all the solutions which have been assigned the rank k. So, the number of dominance comparisons performed by BOS*, because of step (b), is $\sqrt{N}\sum_{i=1}^{\sqrt{N}}(i-1) = \frac{1}{2}N(\sqrt{N}-1)$. Thus, the number of dominance comparisons performed by BOS* is $N(\sqrt{N}-1)$ and the number of objective value comparisons is $MN(\sqrt{N}-1)$.

The number of dominance comparisons, because of step (a), remains the same regardless of the objective value of the solutions, because each solution in a front is dominated by all the solutions in its preceding front. However, in step (b), as all the solutions need to be ranked based on the first objective, a solution which needs to be assigned a rank k, is compared with respect to all the solutions which have been already assigned the rank k. Thus, when all the solutions in a particular front are the same, then BOS* performs the maximum number of dominance comparisons in this particuar situation. The number of dominance comparions performed by ENS-SS in the same situation is also $N(\sqrt{N}-1)$ [9] so, the number of dominance comparisons in this case, is the same as those performed by ESN-SS.

4.3.2. Worst case

The maximum number of dominance comparisons occurs when the 1st to the $(M-2)^{th}$ objective values of all the solutions in a particular front are the same. The last two objective values of the solutions in a particular front should be such that they are able to declare all the solutions in a particular front as non-dominated.

In this case, the order of the solutions in the initial $M-1$ objective lists for each of the fronts is the same and in the last list it is just the opposite. This means that the initial $M-1$ columns of a particular row of the *sorted matrix* have the same solution and the last column has a different solution. Thus, a solution is ranked when it is explored either in the first column or in the last

column of the *sorted matrix*. When a solution is explored in the 2nd to the $(M-1)^{th}$ columns, then it will not be ranked because the same solution has already been ranked when it was explored in the first column. Thus, from each row, only two solutions are ranked (if not already ranked): one from the first column and another one from the last column.

The initial $\sqrt{N}$ rows of the *sorted matrix* have the solutions of the first front. So, all the solutions of the first front are explored for rank assignment in the initial $\sqrt{N}/2$ rows. The $(\sqrt{N}+1)^{th}$ to the $(2\sqrt{N}^{th})$ rows of the *sorted matrix* have the solutions of the second front. So, all the solutions of the second front are explored for rank assignment in the initial $\sqrt{N}/2$ rows after $\sqrt{N}$ rows where the solutions of the first front are present. In the same manner, all the solutions of the third front are explored for rank assignment in the initial $\sqrt{N}/2$ rows after $2\sqrt{N}$ rows where the solutions of the first and second fronts are present. At last, all the solutions of the $\sqrt{N}^{th}$ front are explored for rank assignment in the initial $\sqrt{N}/2$ rows after $(\sqrt{N}-1)\sqrt{N}$ rows where the solutions of the first to the $(\sqrt{N}-1)^{th}$ fronts are present. In general, the solutions of the kth front are explored for rank assignment in the initial $\sqrt{N}/2$ rows after the $(k-1)\sqrt{N}$ rows where the solutions of the first to the $(k-1)^{th}$ fronts are present. Thus, the number of dominance comparisons performed by GBOS-SS and GBOS-BS is given by Eq. (26).

When a solution is explored in the first objective list, then it is ranked and the first objective is removed from its comparison set. When the same solution is explored in the 2nd to the $(M-1)^{th}$ objective lists, then the 2nd to the $(M-1)^{th}$ objectives are removed from the comparison set. So, next time, when a solution is ranked based on the first objective, then the solution is compared based only on the last objective, as only the last objective remains in the comparison set of the solutions which have already been ranked based on the first objective. Similarly, when a solution is explored in the last objective, then it is ranked and the last objective is removed from its comparison set. So, next time, when a solution is ranked based on the last objective, then the solution is compared based on the 1st to the $(M-1)^{th}$ objectives as only the last objective is removed from the comparison set of solutions which have been ranked based on the last objective. Thus, the actual number of objective value comparisons performed by GBOS-SS and GBOS-BS is given by Eq. (27).

$$\begin{aligned}
\#\text{dcmp} &= \#\text{dcmp}_{\text{duplicate}} + \#\text{dcmp}_{\text{assign}} \\
&= N - 1 + \sqrt{N}\left[\sum\nolimits_{i=1}^{\sqrt{N}/2}(i-1) + (i-1)\right] \\
&= N - 1 + \frac{1}{4}N(\sqrt{N}-2) \qquad (26) \\
\#\text{cmp} &= \#\text{cmp}_{\text{duplicate}} + \#\text{dcmp}_{\text{assign}} \\
&= M(N-1) + \sqrt{N}\left[\sum\nolimits_{i=1}^{\sqrt{N}/2}(i-1)(1) + (i-1)(M-1)\right] \\
&= M(N-1) + \frac{1}{8}MN(\sqrt{N}-2) \qquad (27)
\end{aligned}$$

So, the time complexity of GBOS-SS and GBOS-BS, because of step (b), is given by Eq. (28).

$$T_b = \frac{1}{8}MN(\sqrt{N}-2) = \mathcal{O}(MN\sqrt{N}) \tag{28}$$

When a solution needs to be assigned a rank k, then the solution is compared with respect to only one solution of all the previous $k-1$ fronts in the case of GBOS-SS and dominated by all of them. There are $\sqrt{N}$ solutions in each front so all the solutions of the kth front are compared with respect to one solution of all the previous $k-1$ fronts. Thus, the time complexity of GBOS-SS, because of step (a), is given by Eq. (29). Similarly, all the solutions of the kth front are compared with respect to one solution of the previous $\lceil \log k \rceil$ fronts in the case of GBOS-BS. Specifically, the first solution of the kth front is compared with the one solution of the previous $\lceil \log k \rceil$ fronts and dominated by each of them. After the insertion of the first solution in the kth front, the structure of the tree changes, *i.e.*, the position of the nodes inside the tree changes. So, the remaining solution of the kth front is compared with respect to one solution of the previous $\lceil \log k \rceil - 1$ fronts and dominated by each of them. However, this is not the case with all the fronts. The remaining solutions of the $k^{th}(k \in \{2, 4, \ldots, \sqrt{N}\})$ front is also compared with respect to one solution of the previous $\lceil \log k \rceil$ fronts and dominated by each of them. The reason for this is that when the first solution of these fronts is inserted into the tree, then the position of the existing nodes does not change. Thus, the time complexity of GBOS-BS, because of step (a), is given by Eq. (30).

$$\begin{aligned} T_{a_{\mathrm{SS}}} &= \sum\nolimits_{k=1}^{\sqrt{N}} \sqrt{N}(k-1) = \sqrt{N}\sum\nolimits_{k=1}^{\sqrt{N}}(k-1) \\ &= \frac{1}{2}N(\sqrt{N}-1) = \mathcal{O}(N\sqrt{N}) \end{aligned} \tag{29}$$

$$\begin{aligned} T_{a_{\mathrm{BS}}} &= \left[\sum\nolimits_{k=2}^{\sqrt{N}} \lceil \log k \rceil + \left(\sqrt{N}-1\right)\left(\lceil \log k \rceil - 1\right)\right] + \left(\sqrt{N}-1\right)\log\sqrt{N} \\ &= \frac{1}{2}\left(N+\sqrt{N}-1\right)\log N - 2N + 3\sqrt{N} - 1 = \mathcal{O}(N\log N) \end{aligned} \tag{30}$$

The time taken by the rank assignment process using GBOS-SS is obtained by Eq. (31).

$$T_{\mathrm{assign}}(N,M) = T_{a_{\mathrm{SS}}} + T_b = \mathcal{O}(N\sqrt{N}) + \mathcal{O}(MN\sqrt{N}) = \mathcal{O}(MN\sqrt{N}) \tag{31}$$

Hence, the time complexity of GBOS-SS is

$$\begin{aligned} T(N,M) &= T_{\mathrm{presort}}(N,M) + T_{\mathrm{duplicate}}(N,M) + T_{\mathrm{assign}}(N,M) \\ &= \mathcal{O}(MN\log N) + \mathcal{O}(MN) + \mathcal{O}(MN\sqrt{N}) = \mathcal{O}(MN\sqrt{N}) \end{aligned} \tag{32}$$

The time taken by the rank assignment process using GBOS-BS is obtained by Eq. (33).

$$\begin{aligned} T_{\mathrm{assign}}(N,M) &= T_{a_{\mathrm{BS}}} + T_b = \mathcal{O}(N\log N) + \mathcal{O}(MN\sqrt{N}) \\ &= \mathcal{O}(MN\sqrt{N}) \end{aligned} \tag{33}$$

Hence, the time complexity of GBOS-BS is

$$\begin{aligned} T(N,M) &= T_{\text{presort}}(N,M) + T_{\text{duplicate}}(N,M) + T_{\text{assign}}(N,M) \\ &= \mathcal{O}(MN\log N) + \mathcal{O}(MN) + \mathcal{O}(MN\sqrt{N}) = \mathcal{O}(MN\sqrt{N}) \end{aligned} \tag{34}$$

ENS-NDT requires $N-1+\frac{1}{2}N(\sqrt{N}-1)+\frac{1}{2}(N+\sqrt{N}-1)\log N-2N+3\sqrt{N}-1$ dominance comparisons. The number of dominance comparisons performed by BOS* because of step (a) is $\sum_{i=1}^{\sqrt{N}}\sqrt{N}(i-1) = \frac{1}{2}N(\sqrt{N}-1)$ and because of step (b) is $\sqrt{N}\left[\sum_{i=1}^{\sqrt{N}/2}(i-1)+(i-1)\right] = \frac{1}{4}N(\sqrt{N}-2)$. Thus, the number of dominance comparisons performed by BOS* is $\frac{1}{2}N(\sqrt{N}-1)+\frac{1}{4}N(\sqrt{N}-2) = \frac{3}{4}N\sqrt{N}-N$ and the number of objective value comparisons is $\frac{3}{4}MN\sqrt{N}-MN$.

4.3.3. Best case

When N solutions are equally divided into $\sqrt{N}$ fronts, then the best case of GBOS occurs if the traversal of the solutions in the *sorted matrix* follows a specific pattern. The traversal should be such that before the second occurrence of a solution in a front, all the solutions of that particular front must be traversed at least once.

Here, while traversing the *sorted matrix*, before the second occurrence of a solution in a front, all the solutions of that particular front must be traversed at least once. So, in each row of the *sorted matrix*, M solutions are ranked and there are $\sqrt{N}$ solutions in each front. So, the solutions of the first front are ranked in the initial $\sqrt{N}/M$ rows. The solutions of the second front are ranked in the initial $\sqrt{N}/M$ rows after $\sqrt{N}$ rows. The solutions of the third front are ranked in the initial $\sqrt{N}/M$ rows after $2\sqrt{N}$ rows. At the end, the solutions of the last front are ranked in the initial $\sqrt{N}/M$ rows after $(\sqrt{N}-1)\sqrt{N}$ rows. In general, the solutions of the kth front are ranked in the initial $\sqrt{N}/M$ rows after $(k-1)\sqrt{N}$ rows. Thus, the number of dominance comparisons performed by GBOS-SS and GBOS-BS is given by Eq. (35). From this equation, it is clear that as the number of objective increases, the value of $\sqrt{N}/M$ decreases and the number of dominance comparisons decreases. When $M \geq \sqrt{N}$, then the number of dominance comparisons is fixed to $N-1$ which is required for checking for duplicate solutions.

Here, each time a solution of a particular front is ranked, its comparison set is reduced by one objective and before the same solution occurs a second time, all the solutions of that particular front are ranked. So, whenever a solution is compared with respect to other solutions of the same front, a maximum $M-1$ number of objectives is considered. Thus, the maximum number of objective value comparisons performed by GBOS-SS and GBOS-BS is given by Eq. (36).

$$\begin{aligned} \#\text{dcmp} &= \#\text{dcmp}_{\text{duplicate}} + \#\text{dcmp}_{\text{assign}} \\ &= N-1+\sqrt{N}\left[\sum_{k=1}^{\sqrt{N}/M} M(k-1)\right] \\ &= N-1+\frac{1}{2M}N(\sqrt{N}-M) \end{aligned} \tag{35}$$

$$
\begin{aligned}
\#\text{dcmp} &= \#\text{dcmp}_{\text{duplicate}} + \#\text{dcmp}_{\text{assign}} \\
&= M(N-1) + \sqrt{N}\left[\sum\nolimits_{k=1}^{\sqrt{N}/M} M(k-1)(M-1)\right] \\
&= M(N-1) + \frac{1}{2M}N(M-1)(\sqrt{N}-M) \qquad (36)
\end{aligned}
$$

When $M \geq \sqrt{N}$, then the solutions in the $\sqrt{N}(\sqrt{N}-1)+1$ rows of the *sorted matrix* are explored and ranked if they had not been already ranked. Also, there is no dominance comparison between the solutions of the same front except for checking for duplicate solutions when $M \geq \sqrt{N}$. So, only the *sorted matrix* is traversed once until all the solutions are ranked. Thus, the time complexity of GBOS-SS and GBOS-BS, because of step (b), is given by Eq. (37).

$$
T_b = M\left[\sqrt{N}(\sqrt{N}-1)+1\right] = \mathcal{O}(MN) \qquad (37)
$$

In this case, the time complexity of GBOS-SS, because of step (a), is given by Eq. (29) and the time complexity of GBOS-BS, because of step (a), is given by Eq. (30). The time taken by the rank assignment process using GBOS-SS is obtained by Eq. (38).

$$
T_{\text{assign}}(N,M) = T_{a_{\text{SS}}} + T_b = \mathcal{O}(N\sqrt{N}) + \mathcal{O}(MN) = \mathcal{O}(MN + N\sqrt{N}) \qquad (38)
$$

Hence, the time complexity of GBOS-SS is

$$
\begin{aligned}
T(N,M) &= T_{\text{presort}}(N,M) + T_{\text{duplicate}}(N,M) + T_{\text{assign}}(N,M) \\
&= \mathcal{O}(MN\log N) + \mathcal{O}(MN) + \mathcal{O}(MN + N\sqrt{N}) \\
&= \mathcal{O}(MN\log N) \text{ as } M \geq \sqrt{N} \qquad (39)
\end{aligned}
$$

The time taken by the rank assignment process using GBOS-BS is obtained by Eq. (40).

$$
\begin{aligned}
T_{\text{assign}}(N,M) &= T_{a_{\text{BS}}} + T_b = \mathcal{O}(N\log N) + \mathcal{O}(MN) \\
&= \mathcal{O}(N\log N + MN) \qquad (40)
\end{aligned}
$$

Hence, the time complexity of GBOS-BS is

$$
\begin{aligned}
T(N,M) &= T_{\text{presort}}(N,M) + T_{\text{duplicate}}(N,M) + T_{\text{assign}}(N,M) \\
&= \mathcal{O}(MN\log N) + \mathcal{O}(MN) + \mathcal{O}(N\log N + MN) \\
&= \mathcal{O}(MN\log N) \qquad (41)
\end{aligned}
$$

When N solutions are equally divided into $\sqrt{N}$ fronts, then the best case of BOS* occurs if the traversal of the solutions in the *sorted matrix* follows a specific pattern. The traversal should be such that before the second occurrence of a solution in a front, all the solutions of that particular front must be traversed at least once.

Here, while traversing the *sorted matrix*, before the second occurrence of a solution in a front, all the solutions of that particular front must be traversed at least once. So, in each row of the *sorted matrix*, M solutions are ranked and there are $\sqrt{N}$ solutions in each front. So, the solutions of the first front are ranked in the initial $\sqrt{N}/M$ rows. The solutions of the second front are ranked in the initial $\sqrt{N}/M$ rows after $\sqrt{N}$ rows. The solutions of the third front are ranked in the initial $\sqrt{N}/M$ rows after $2\sqrt{N}$ rows. At the end, the solutions of the last front are ranked in the initial $\sqrt{N}/M$ rows after $(\sqrt{N}-1)\sqrt{N}$ rows. In general, the solutions of the kth front are ranked in the initial $\sqrt{N}/M$ rows after $(k-1)\sqrt{N}$ rows.

The number of dominance comparisons performed by BOS* because of step (a) is $\sum_{i=1}^{\sqrt{N}} \sqrt{N}(i-1) = \frac{1}{2}N(\sqrt{N}-1)$ and because of step (b) is $\sqrt{N}[\sum_{i=1}^{\sqrt{N}/M} M(i-1)] = \frac{1}{2M}N(\sqrt{N}-M)$. Thus, the number of dominance comparisons performed by BOS* is $\frac{1}{2}N(\sqrt{N}-1) + \frac{1}{2M}N(\sqrt{N}-M)$ and the number of objective value comparisons is $\frac{1}{2}MN(\sqrt{N}-1) + \frac{1}{2}N(\sqrt{N}-M)$. Hence, as the number of objectives increases, the number of dominance comparisons performed by BOS* decreases. When $M \geq \sqrt{N}$, the number of dominance comparisons, because of step (b), becomes 0 and the total number of dominance comparisons is equal to the number of dominance comparisons because of step (a). Thus, when $M \geq \sqrt{N}$, the number of dominance comparisons performed by BOS* remains fixed and is equal to $\frac{1}{2}N(\sqrt{N}-1)$.

Table 2 shows the number of dominance comparisons performed by some of the non-dominated sorting approaches in different scenarios. From this table it is clear that the number of dominance comparisons performed by GBOS-SS and GBOS-BS is less as compared to other approaches. The time and space complexities of several non-dominated sorting approaches are given in Table 3.

4.4. Number of Dominance Comparisons

Let $N = 2^{16}$ solutions be non-dominated. Consider the number of objectives $M = 2^i (1 \leq i \leq 15)$. The number of dominance comparisons performed by ENS-SS, ENS-BS, BOS* and GBOS (GBOS-SS and GBOS-BS) are shown in Figure 3(a). The number of dominance comparisons performed by GBOS when all the solutions are duplicated is shown in this figure. The number of dominance comparisons performed by GBOS in this situation, in the best and the worst case, are also shown in the figure. From this figure, it is clear that the number of dominance comparisons when all the solutions are duplicated remains fixed irrespectively of the number of objectives and it is $N-1$ for N solutions. The number of dominance comparisons in the worst case also remains fixed, irrespectively of the number of objectives. However, in the best case, as the number of objectives increases, the number of dominance comparisons decreases. The worst case of BOS* occurs when all the solutions in a front are the same. The number of dominance comparisons performed in this case by BOS* is $\frac{1}{2}N(N-1)$ which is the same as the number of dominance comparisons performed by ENS. In the best case, the number of dominance comparisons performed by BOS* is $N-1$ less than those performed by GBOS.

Table 2: Number of dominance comparisons performed by several non-dominated sorting approaches in different scenarios.

Approach	**Number of Dominance Comparisons**		
	N solutions in single front	**N solutions in N fronts**	**Equal division of N solutions in $\sqrt{N}$ fronts**
FNDS [1]	$\frac{N(N-1)}{2}$	$\frac{N(N-1)}{2}$	$\frac{N(N-1)}{2}$
Deductive Sort [10]	$\frac{N(N-1)}{2}$	$\frac{N(N-1)}{2}$	$\frac{1}{2}(N-1)(\sqrt{N}+1)^a$
ENS-SS [9]	$\frac{N(N-1)}{2}$	$\frac{N(N-1)}{2}$	$N(\sqrt{N}-1)$
ENS-BS [9]	$\frac{N(N-1)}{2}$	$N\log N-(N-1)$	$\frac{N(\sqrt{N}-1)}{2}+\frac{(N+\sqrt{N}-1)}{2}\log N-$ $2N+3\sqrt{N}-1$
ENS-NDT [15]	$N-1^b$ $N-1+\frac{N(N-1)}{2}^c$	$N\log N$	$N-1+\frac{1}{2}\sqrt{N}\log N-(\sqrt{N}-1)^b$ $\frac{N(\sqrt{N}-1)}{2}+\frac{(N+\sqrt{N}-1)}{2}\log N-$ $N+3\sqrt{N}-2^c$
BOS*	$\frac{N(N-1)}{2}^b$ $\frac{N(N-2)}{4}^c$ $\frac{N(N-M)}{2M}^d$	$\frac{N(N-1)}{2}$	$N(\sqrt{N}-1)^b$ $\frac{N(\sqrt{N}-1)}{2}+\frac{N(\sqrt{N}-2)}{4}^c$ $\frac{N(\sqrt{N}-1)}{2}+\frac{N(\sqrt{N}-M)}{2M}^d$
GBOS-SS GBOS-BS	$N-1^b$ $N-1+\frac{N(N-2)}{4}^c$ $N-1+\frac{N(N-M)}{2M}^d$	$N-1$	$N-1^b$ $N-1+\frac{N(\sqrt{N}-2)}{4}^c$ $N-1+\frac{N(\sqrt{N}-M)}{2M}^d$

[a]Assumption: First solution selected in each iteration is in the current front [10].

[b]All the solutions in a front are the same.

[c]Worst case of GBOS-SS and GBOS-BS.

[d]Best case of GBOS-SS and GBOS-BS.

Let $N = 2^i (1 \leq i \leq 16)$ solutions be in different fronts. The number of dominance comparisons performed by ENS-SS, ENS-BS, BOS* and GBOS (GBOS-SS and GBOS-BS) are shown in Figure 3(b). The number of dominance comparisons performed by GBOS is less than those performed by ENS-SS, ENS-BS and BOS*. ENS-SS and BOS* perform the same number of dominance comparisons. The number of dominance comparisons performed by GBOS is $N-1$, which occurs due to the checking of duplicate solutions.

Let $N = 2^{16}$ solutions be equally divided into $\sqrt{N} = 2^8$ fronts such that each solution in a front dominates all the solutions in its succeeding front. Consider the number of objectives $M = 2^i (1 \leq i \leq 16)$. The number of dominance comparisons performed by ENS-SS, ENS-BS, BOS* and GBOS (GBOS-SS and GBOS-BS) are shown in Figure 3(c). The number of dominance comparisons performed by GBOS is lower than those performed by ENS-SS and ENS-BS.

Approach	Space Complexity	Time Complexity	
		Best Case	Worst Case
Naive approach	$\mathcal{O}(N)$	$\mathcal{O}(MN^2)$	$\mathcal{O}(MN^3)$
FNDS [1]	$\mathcal{O}(N^2)$	$\mathcal{O}(MN^2)$	$\mathcal{O}(MN^2)$
Jensen [5]	$\mathcal{O}(MN)$	$\mathcal{O}(N \log N)^{\S}$	$\mathcal{O}(N \log^{M-1} N)$
Deductive Sort [10]	$\mathcal{O}(N)$	$\mathcal{O}(MN\sqrt{N})$	$\mathcal{O}(MN^2)$
ENS-SS [9]	$\mathcal{O}(1)$	$\mathcal{O}(MN\sqrt{N})$	$\mathcal{O}(MN^2)$
ENS-BS [9]	$\mathcal{O}(1)$	$\mathcal{O}(MN \log N)$	$\mathcal{O}(MN^2)$
BOS*	$\mathcal{O}(MN)$	$\mathcal{O}(MN \log N)$	$\mathcal{O}(MN^2)$
T-ENS†[13]	$\mathcal{O}(MN)$	$\mathcal{O}(MN \log N/ \log M)$	$\mathcal{O}(MN^2)$
ENS-NDT[15]	$\mathcal{O}(N \log N)^{\ddagger}$	$\mathcal{O}(MN \log N)$	$\mathcal{O}(MN^2)$
GBOS-SS	$\mathcal{O}(MN)$	$\mathcal{O}(MN \log N)$	$\mathcal{O}(MN^2)$
GBOS-BS	$\mathcal{O}(MN)$	$\mathcal{O}(MN \log N)$	$\mathcal{O}(MN^2)$

§Best case time complexity when $M = 2$.

†Not suitable when solutions share identical values for any of the objectives [15].

‡Worst case: $\mathcal{O}(N \log N)$, Best case: $\mathcal{O}(\log N)$, Average case: $\mathcal{O}(N)$.

Table 3: Space and Time complexities of different non-dominated sorting approaches.

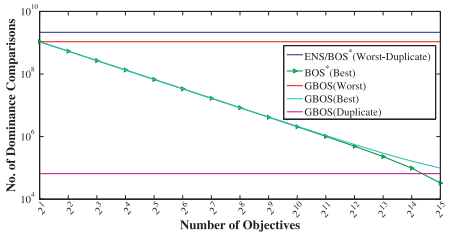
When the solutions in a front are duplicated, then the number of dominance comparisons is fixed irrespectively of the number of objectives. Similarly, in the worst case of GBOS, the number of dominance comparisons does not change when modifying the number of objectives. However, in the best case of GBOS, the dominance comparisons decreases with an increase in the number of objectives. As $M \geq \sqrt{N}$, the number of dominance comparisons remains fixed (see Eq. (35)). In this situation, the worst case of BOS* occurs when all the solutions in a particular front are the same. In the worst case, BOS* performs the same number of dominance comparisons as required by ENS-SS. However, in the best case of BOS*, the number of dominance comparisons decreases with an increase in the number of objectives and when $M \geq \sqrt{N}$, the number of dominance comparisons remains fixed.

5. Experimental Evaluation

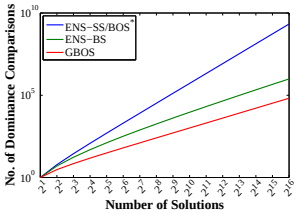
In this section, generalized BOS is compared with some of the existing approaches. We have compared GBOS with Fast non-dominated sort (FNDS), Deductive sort (DS), ENS-SS and ENS-BS. The corresponding experiments were carried out on a Windows 7 PC with a 3.30 GHz Intel $i5$ processor and 4 GB of RAM.

5.1. Fixed Front Dataset

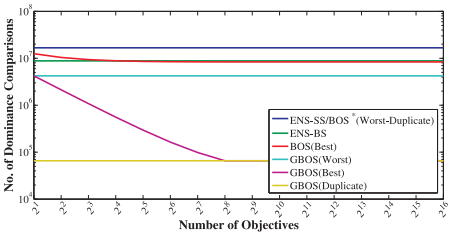
In the fixed front dataset, the solutions are equally divided into k fronts and the division is such that each solution in a front is dominated by all the solutions in its preceding front. For the experiments reported next, the population size was set to 10000. We have considered four different numbers of objectives – 5,



(a) All solutions are non-dominated



(b) Solutions are in different fronts

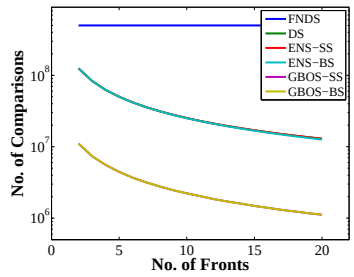


(c) Solutions are equally divided in $\sqrt{N}$ fronts

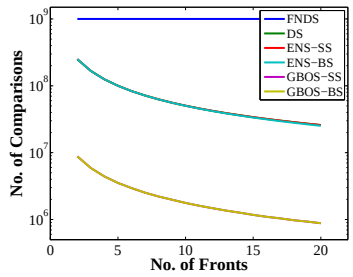
Figure 3: Number of dominance comparisons performed by several non-dominated sorting approaches in different scenarios (log based y-axis). GBOS corresponds to both GBOS-SS and GBOS-BS.

10, 15 and 20. The number of fronts is varied from 2 to 20 with a step size of 1. The population size, number of objectives and the number of fronts are the same as in [3]. The number of objective value comparisons and the running times required by different approaches for the fixed front dataset are shown in Figure 4. The number of objective value comparisons is shown in Figures 4(a)–4(d). The running time is shown in Figures 4(e)–4(h). From these figures it is clear that the number of objective value comparisons performed by FNDS is maximum as in FNDS each solution is compared with all other solutions. The running time of FNDS is also maximum as compared to other approaches because it performs the maximum number of objective value comparisons. In this dataset, the objective value comparisons performed by FNDS, Deductive Sort, ESN-SS and ENS-BS remains the same irrespectively of the change in the number of objectives because the number of solutions is the same and also the dominance relationship between the solutions is also the same. With an increase in the number of fronts, the number of objective value comparisons performed by ENS-BS is less than those performed by ENS-SS because ENS-BS uses a binary search based strategy and such a strategy performs better than a sequential search based strategy when the number of fronts is large [9]. The number of objective value comparisons performed by GBOS-SS and GBOS-BS is the same regardless of the number of objectives.

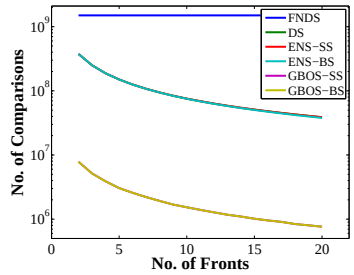
In this dataset, each solution in a front is dominated by all the solutions in its preceding front, so when the *sorted matrix* is traversed in a row-wise manner, then before the second occurrence of a solution in the kth fronts, all



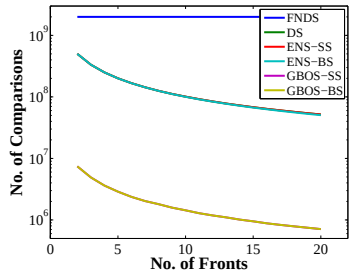
(a) $M = 5$



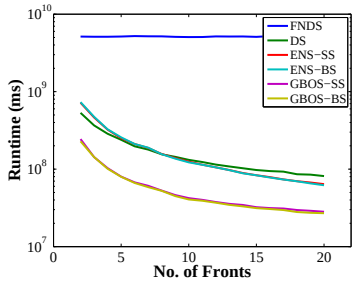
(b) $M = 10$



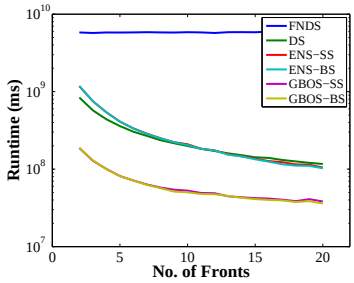
(c) $M = 15$



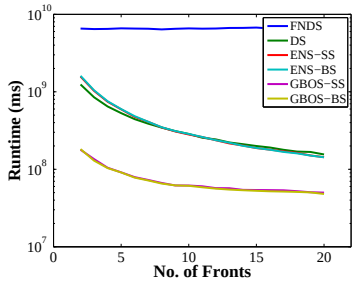
(d) $M = 20$



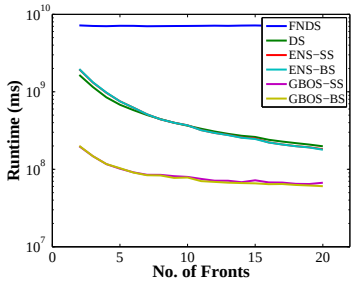
(e) $M = 5$



(f) $M = 10$

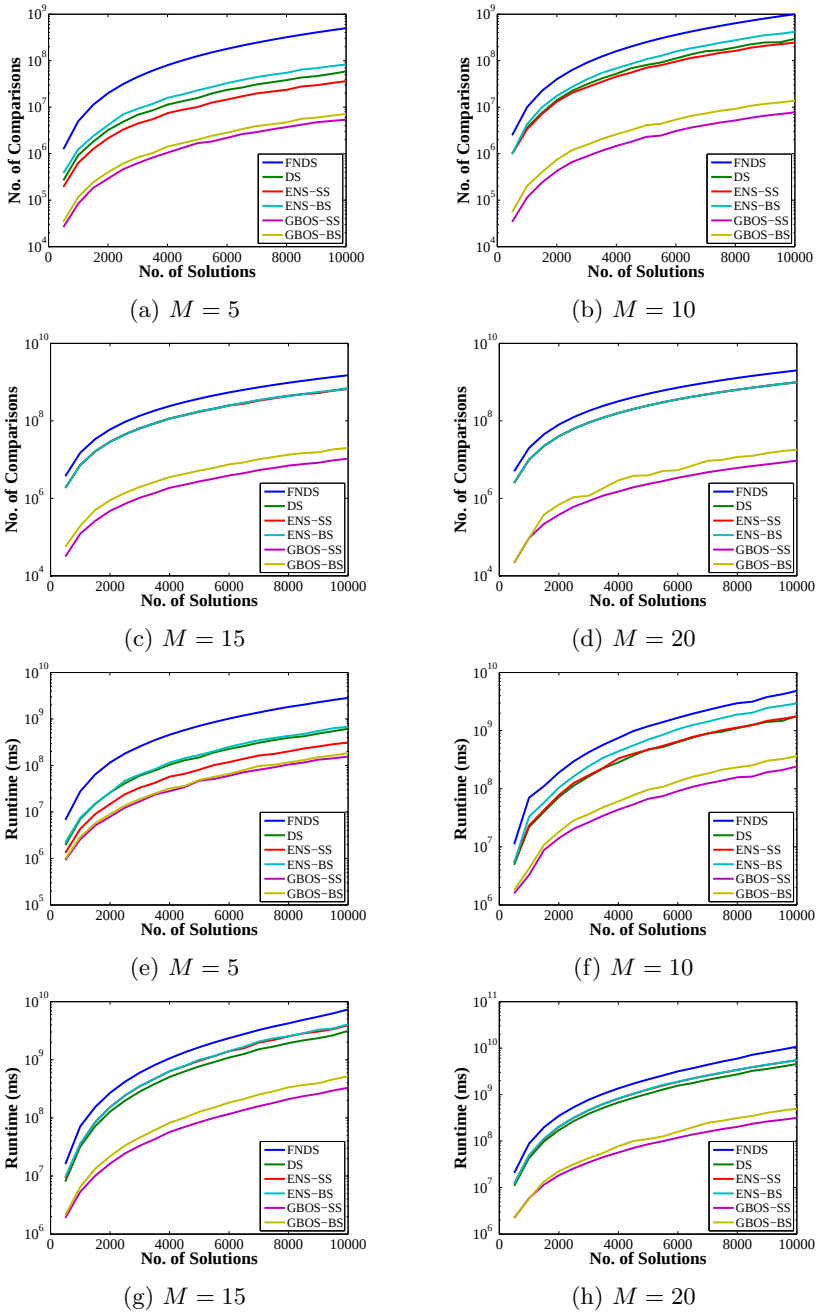


(g) $M = 15$



(h) $M = 20$

Figure 4: Performance of non-dominated sorting approaches for the fixed front dataset in terms of the number of objective value comparisons performed and the execution time required. log based y-axis.



(a) $M = 5$ (b) $M = 10$

(c) $M = 15$ (d) $M = 20$

(e) $M = 5$ (f) $M = 10$

(g) $M = 15$ (h) $M = 20$

Figure 5: Performance of non-dominated sorting approaches for the cloud dataset in terms of the number of objective value comparisons performed and the execution time required. log based y-axis.

the solutions in the $(k-1)^{th}$ front have been traversed in all the columns. So, the comparison set of the $(k-1)^{th}$ front becomes empty when the solutions of the kth front are ranked. So, the solutions are compared based on the objective values with respect to the solutions of the same front. Hence, the number of objective value comparisons performed by GBOS-SS and GBOS-BS is the same. The number of dominance comparisons performed by GBOS-SS and GBOS-BS is much lower than those performed by other approaches. The running time required by GBOS-SS and GBOS-BS is less as compared to that of other approaches.

5.2. Cloud Dataset

In the cloud dataset, the solutions are randomly generated where the objective values vary between 0 and 1. So, the number of fronts and the number of solutions in a particular front is also random. This kind of dataset mimics the initial stages of evolutionary algorithms. Here, we have varied the population size from 500 to 10000 with an increment of 500. We have considered four different numbers of objectives: 5, 10, 15 and 20. The population size and the number of objectives is the same as in [3]. The number of objective value comparisons is shown in Figures 5(a)–5(d). The running time is shown in Figures 5(e)–5(h).

From Figures 5(a)–5(d) it is clear that the number of objective value comparisons performed by FNDS is maximum as in FNDS each solution is compared with respect to all other solutions. The running time of FNDS is also maximum as compared to that of the other approaches because it performs the maximum number of objective value comparisons. With an increase in the number of objectives, the objective value comparisons performed by ENS-SS and ENS-BS are nearly the same because as the number of objective increases, the number of fronts decreases in the cloud dataset and with a small number of fronts, both sequential and binary search based approaches perform almost the same. The number of objective value comparisons performed by GBOS-SS and GBOS-BS are much lower as compared to those of the other approaches. Regarding running time, GBOS-SS and GBOS-BS require less time as compared to that of the other approaches.

5.3. Incorporation into NSGA-II

Our proposed approach along with other non-dominated sorting approaches are incorporated in NSGA-II [1] for solving four test problems: DTLZ1, DTLZ2, DTLZ3 and DTLZ4. We have considered four different numbers of objectives for these test problems: 5, 10, 15 and 20. The number of generations was set to 200 and the population size was set to 800 as in [3]. We have used simulated binary crossover (SBX) and polynomial-based mutation [23] as in [1] in our experiments. The crossover probability p_c was set to 0.9 and the mutation probability p_m was set to $1/n$ where n is the number of decision variables. The distribution index [23] for crossover was adopted as $\eta_c = 20$ and the distribution index for mutation η_m was set to 20. The number of objective value comparisons performed by the different non-dominated sorting approaches as well as

their corresponding running times are reported in Table 4. From this table, it is evident that our proposed approach is more efficient than the non-dominated sorting approaches with respect to which it was compared, in terms of both the number of objective value comparisons and the running time required. FNDS performs the maximum number of objective value comparisons and it also requires the maximum running time. GBOS-SS and GBOS-BS performs less objective value comparisons because of the sorting of solutions based on each objective and also because, few objectives are considered while comparing two solutions.

In case of FNDS, the number of objective value comparison remains the same for all the test problems and for all the objectives. This is because, in FNDS each solution is compared with respect to all the solutions irrespectively of the objective values of the solutions. However, in other approaches, this is not the case as the number of objective value comparisons depends on the dominance relationship between the solutions.

6. Conclusions and Future Work

In the current paper, an efficient approach for non-dominated sorting known as Best Order Sort was generalized to overcome its main limitations. Generalized BOS is also efficient in terms of the number of dominance comparisons. The comparison set concept is retained in the generalized BOS which further reduces the number of objective value comparisons required when two solutions are compared. Generalized BOS does not change its best and worst case time complexity, nor its space complexity.

As part of our future work, we are interested in extending our Generalized BOS to parallel architectures, with the aim of improving its efficiency even more.

Acknowledgements

The last author gratefully acknowledges support from CONACyT project no. 221551.

References

[1] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197.

[2] K. Deb, H. Jain, An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints, IEEE Transactions on Evolutionary Computation 18 (4) (2014) 577–601.

Test Problem	Obj.	FNDS		DS		ENS-SS		ENS-BS		GBOS-SS		GBOS-BS	
		#cmp	runtime	#cmp	runtime	#cmp	runtime	#cmp	runtime	#cmp	runtime	#cmp	runtime
DTLZ1	5	2.55e+9	1.02e+4	1.20e+9	4.75e+3	1.01e+9	3.81e+3	1.16e+9	4.47e+3	3.24e+7	8.69e+2	6.25e+7	1.10e+3
	10	5.11e+9	1.64e+4	2.47e+9	8.17e+3	2.27e+9	7.57e+3	2.41e+9	8.17e+3	4.21e+7	1.29e+3	8.02e+7	1.59e+3
	15	7.67e+9	2.27e+4	3.75e+9	1.30e+4	3.63e+9	1.04e+4	3.68e+9	1.06e+4	6.14e+7	1.80e+3	1.16e+8	2.20e+3
	20	1.02e+10	3.35e+4	5.03e+9	1.50e+4	4.93e+9	1.36e+4	4.96e+9	1.39e+4	7.26e+7	2.23e+3	1.36e+8	2.67e+3
DTLZ2	5	2.55e+9	1.13e+4	1.24e+9	5.57e+3	1.19e+9	5.07e+3	1.21e+9	5.19e+3	4.34e+7	1.01e+3	8.37e+7	1.44e+3
	10	5.11e+9	1.84e+4	2.52e+9	9.41e+3	2.48e+9	9.39e+3	2.48e+9	9.62e+3	5.62e+7	1.51e+3	1.08e+8	1.99e+3
	15	7.67e+9	2.54e+4	3.80e+9	1.29e+4	3.77e+9	1.22e+4	3.77e+9	1.21e+4	6.48e+7	1.96e+3	1.23e+8	2.45e+3
	20	1.02e+10	3.08e+4	5.08e+9	1.61e+4	5.05e+9	1.46e+4	5.06e+9	1.50e+4	8.00e+7	2.32e+3	1.50e+8	2.89e+3
DTLZ3	5	2.55e+9	7.96e+3	1.23e+9	3.67e+3	7.71e+8	2.69e+3	1.08e+9	3.94e+3	1.49e+7	5.74e+2	2.74e+7	6.60e+2
	10	5.11e+9	1.80e+4	2.49e+9	8.75e+3	2.38e+9	8.52e+3	2.42e+9	8.71e+3	5.15e+7	1.45e+3	9.85e+7	1.83e+3
	15	7.67e+9	2.49e+4	3.78e+9	1.21e+4	3.70e+9	1.16e+4	3.71e+9	1.15e+4	6.63e+7	1.92e+3	1.26e+8	2.37e+3
	20	1.02e+10	3.34e+4	5.05e+9	1.52e+4	4.99e+9	1.38e+4	5.00e+9	1.41e+4	7.73e+7	2.31e+3	1.45e+8	2.85e+3
DTLZ4	5	2.55e+9	1.03e+4	1.22e+9	4.93e+3	1.04e+9	4.11e+3	1.15e+9	4.62e+3	3.00e+7	8.02e+2	5.77e+7	1.04e+3
	10	5.11e+9	1.96e+4	2.51e+9	1.01e+4	2.44e+9	9.96e+3	2.45e+9	9.86e+3	3.03e+7	1.18e+3	5.73e+7	1.39e+3
	15	7.67e+9	2.89e+4	3.79e+9	1.48e+4	3.72e+9	1.36e+4	3.73e+9	1.38e+4	2.84e+7	1.53e+3	5.29e+7	1.72e+3
	20	1.02e+10	3.86e+4	5.06e+9	1.99e+4	5.00e+9	1.80e+4	5.00e+9	1.81e+4	2.36e+7	1.77e+3	4.25e+7	1.94e+3

Table 4: Performance study of different non-dominated sorting approaches in terms of the number of objective value comparisons (#cmp) and running time (in milliseconds) when those are incorporated in NSGA-II for DTLZ1, DTLZ2, DTLZ3 and DTLZ4.

[3] P. C. Roy, M. M. Islam, K. Deb, Best Order Sort: A New Algorithm to Non-dominated Sorting for Evolutionary Multi-objective Optimization, in: Proceedings of the 2016 Genetic and Evolutionary Computation Conference Companion, ACM Press, Denver, Colorado, USA, 2016, pp. 1113–1120.

[4] N. Srinivas, K. Deb, Muiltiobjective optimization using nondominated sorting in genetic algorithms, Evolutionary computation 2 (3) (1994) 221–248.

[5] M. T. Jensen, Reducing the run-time complexity of multiobjective eas: The NSGA-II and other algorithms, IEEE Transactions on Evolutionary Computation 7 (5) (2003) 503–515.

[6] F.-A. Fortin, S. Greiner, M. Parizeau, Generalizing the Improved Run-Time Complexity Algorithm for Non-Dominated Sorting, in: 2013 Genetic and Evolutionary Computation Conference (GECCO'2013), ACM Press, New York, USA, 2013, pp. 615–622, iSBN 978-1-4503-1963-8.

[7] H. Fang, Q. Wang, Y.-C. Tu, M. F. Horstemeyer, An Efficient Non-dominated Sorting Method for Evolutionary Algorithms, Evolutionary Computation 16 (3) (2008) 355–384.

[8] S. Tang, Z. Cai, J. Zheng, A Fast Method of Constructing the Non-dominated Set: Arena's Principle, in: 2008 Fourth International Conference on Natural Computation, IEEE Computer Society Press, Jinan, China, 2008, pp. 391–395, iSBN 978-0-7695-3304-9.

[9] X. Zhang, Y. Tian, R. Cheng, J. Yaochu, An Efficient Approach to Non-dominated Sorting for Evolutionary Multiobjective Optimization, IEEE Transactions on Evolutionary Computation 19 (2) (2015) 201–213.

[10] K. McClymont, E. Keedwell, Deductive Sort and Climbing Sort: New Methods for Non-Dominated Sorting, Evolutionary Computation 20 (1) (2012) 1–26.

[11] S. Mishra, S. Saha, S. Mondal, Divide and Conquer Based Non-Dominated Sorting for Parallel Environment, in: 2016 IEEE Congress on Evolutionary Computation (CEC'2016), IEEE Press, Vancouver, Canada, 2016, pp. 4297–4304, iSBN:978-1-5090-0623-6.

[12] M. Buzdalov, A. Shalyto, A Provably Asymptotically Fast Version of the Generalized Jensen Algorithm for Non-dominated Sorting, in: T. Bartz-Beielstein, J. Branke, B. Filipič, J. Smith (Eds.), Parallel Problem Solving from Nature - PPSN XIII, 13th International Conference, Springer. Lecture Notes in Computer Science Vol. 8672, Ljubljana, Slovenia, 2014, pp. 528–537.

[13] X. Zhang, Y. Tian, R. Cheng, Y. Jin, A Decision Variable Clustering-Based Evolutionary Algorithm for Large-Scale Many-Objective Optimization, IEEE Transactions on Evolutionary Computation 22 (1) (2018) 97–112.

[14] X. Zhang, Y. Tian, R. Cheng, Y. Jin, Empirical Analysis of A Tree-based Efficient Non-dominated Sorting Approach for Many-Objective Optimization, in: Proceedings of 2016 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE Press, Athens, Greece, 2016, iSBN 978-1-5090-4240-1.

[15] P. Gustavsson, A. Syberfeldt, A New Algorithm Using the Non-Dominated Tree to Improve Non-Dominated Sorting, Evolutionary Computation 26 (1) (2018) 89–116.

[16] M. Drozdik, Y. Akimoto, H. Aguirre, K. Tanaka, Computational Cost Reduction of Nondominated Sorting Using the M-Front, IEEE Transactions on Evolutionary Computation 19 (5) (2015) 659–678.

[17] M. Buzdalov, I. Yakupov, A. Stankevich, Fast Implementation of the Steady-State NSGA-II Algorithm for Two Dimensions Based on Incremental Non-Dominated Sorting, in: 2015 Genetic and Evolutionary Computation Conference (GECCO 2015), ACM Press, Madrid, Spain, 2015, pp. 647–654, iSBN 978-1-4503-3472-3.

[18] I. Yakupov, M. Buzdalov, Incremental Non-Dominated Sorting with O(N) Insertion for the Two-Dimensional Case, in: 2015 IEEE Congress on Evolutionary Computation (CEC'2015), IEEE Press, Sendai, Japan, 2015, pp. 1853–1860, iSBN 978-1-4799-7492-4.

[19] K. Li, K. Deb, Q. Zhang, Q. Zhang, Efficient Nondomination Level Update Method for Steady-State Evolutionary Multiobjective Optimization, IEEE Transactions on Cybernetics 47 (9) (2017) 2838–2849.

[20] S. Mishra, S. Mondal, S. Saha, Improved Solution to the Non-Domination Level Update Problem, Applied Soft Computing 60 (2017) 336–362.

[21] I. Yakupov, M. Buzdalov, Improved incremental non-dominated sorting for steady-state evolutionary multiobjective optimization, in: 2017 Genetic and Evolutionary Computation Conference (GECCO'2017), ACM Press, Berlin, Germany, 2017, pp. 649–656, iSBN 978-1-4503-4920-8.

[22] J. W. J. Williams, Algorithm-232 - Heapsort, Communications of the ACM 7 (6) (1964) 347–348.

[23] K. Deb, R. B. Agrawal, Simulated Binary Crossover for Continuous Search Space, Complex Systems 9 (2) (1995) 115–148.
are discussed.

Appendix A. Examples

In this section, some of the examples showing the behavior of GBOS (GBOS-SS and GBOS-BS) in three different scenarios are discussed.

Appendix A.1. All Solutions are Non-Dominated

Here, we discuss the examples where all the solutions are non-dominated, *i.e.*, in a single front.

Appendix A.1.1. Duplicate solutions

Figure A.6 shows eight solutions in a 4-dimensional space which are identical. Here, the sorted solutions based on different objectives have the same ordering of solutions.

Sol	**Objective**			
	O_1	O_2	O_3	O_4
s_1	1	2	3	4
s_2	1	2	3	4
s_3	1	2	3	4
s_4	1	2	3	4
s_5	1	2	3	4
s_6	1	2	3	4
s_7	1	2	3	4
s_8	1	2	3	4

(a) Objective values

Q_1	Q_2	Q_3	Q_4
s_1	s_1	s_1	s_1
s_2	s_2	s_2	s_2
s_3	s_3	s_3	s_3
s_4	s_4	s_4	s_4
s_5	s_5	s_5	s_5
s_6	s_6	s_6	s_6
s_7	s_7	s_7	s_7
s_8	s_8	s_8	s_8

(b) Sorted list

Figure A.6: Eight solutions in a 4-dimensional space which are identical.

Appendix A.1.2. Worst case

Figure A.7 shows the worst case scenario for eight solutions when the number of objectives is 4. In this case, the first two objective values of each of the solutions are the same. The last two objective values of the solutions are such that we can declare that all the solutions are non-dominated.

Sol	**Objective**			
	O_1	O_2	O_3	O_4
s_1	1	1	1	8
s_2	1	1	2	7
s_3	1	1	3	6
s_4	1	1	4	5
s_5	1	1	5	4
s_6	1	1	6	3
s_7	1	1	7	2
s_8	1	1	8	1

(a) Objective values

Q_1	Q_2	Q_3	Q_4
s_1	s_1	s_1	s_8
s_2	s_2	s_2	s_7
s_3	s_3	s_3	s_6
s_4	s_4	s_4	s_5
s_5	s_5	s_5	s_4
s_6	s_6	s_6	s_3
s_7	s_7	s_7	s_2
s_8	s_8	s_8	s_1

(b) Sorted list

Figure A.7: Eight solutions in a 4-dimensional space which are in a single front (Worst case scenario).

Appendix A.1.3. Best Case

Figure A.8 shows the best case scenario for eight solutions when the number of objectives is 4. Here, when the sorted list based on each objective is traversed in a row-wise manner, then before the second occurrence of a solution, all the solutions must be traversed at least once.

Sol	**Objective**			
	O_1	O_2	O_3	O_4
s_1	1	8	X	X
s_2	8	1	X	X
s_3	3	6	1	X
s_4	6	3	X	1
s_5	2	7	X	X
s_6	7	2	X	X
s_7	4	5	2	X
s_8	5	4	X	2

(a) Objective values

Q_1	Q_2	Q_3	Q_4
s_1	s_2	s_3	s_4
s_5	s_6	s_7	s_8
s_3	s_4	NA	NA
s_7	s_8	NA	NA
s_8	s_7	NA	NA
s_4	s_3	NA	NA
s_6	s_5	NA	NA
s_2	s_1	NA	NA

(b) Sorted list

Figure A.8: Eight solutions in a 4-dimensional space which are in a single front (Best case scenario). $NA = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$ but in each sorted list a solution occurs exactly once.

In Figure A.8, the first two objective values of the solutions allow us to declare that all these solutions are non-dominated with respect to each other. So, X in Figure A.8 represents any integer value greater than 2, *i.e.*, X > 2. The value of X is chosen such that when the sorted list based on each objective is traversed in a row-wise manner, then before the second occurrence of a solution, all the solutions are traversed at least once.

Appendix A.2. Solutions are in different fronts

An example in which all the solutions are in different fronts is discussed. Figure A.9 shows eight solutions when the number of objectives is 4.

Appendix A.3. N Solutions in $\sqrt{N}$ Fronts

Here, N solutions are equally divided into $\sqrt{N}$ fronts such that each solution in a front is dominated by all the solutions in its preceding front. In this case, when the sorted list based on each objective is traversed in a row-wise manner, then before the occurrence of a solution in the kth front all the solutions of the $(k-1)^{th}$ front are traversed in each of the objectives lists.

Appendix A.3.1. Solutions in a front are identical

Figure A.10 shows the scenario for sixteen solutions in a 4-dimensional space where all the solutions in a particular front are the same.

Sol	Objective			
	O_1	O_2	O_3	O_4
s_1	1	1	1	1
s_2	2	2	2	2
s_3	3	3	3	3
s_4	4	4	4	4
s_5	5	5	5	5
s_6	6	6	6	6
s_7	7	7	7	7
s_8	8	8	8	8

(a) Objective values

Q_1	Q_2	Q_3	Q_4
s_1	s_1	s_1	s_1
s_2	s_2	s_2	s_2
s_3	s_3	s_3	s_3
s_4	s_4	s_4	s_4
s_5	s_5	s_5	s_5
s_6	s_6	s_6	s_6
s_7	s_7	s_7	s_7
s_8	s_8	s_8	s_8

(b) Sorted list

Figure A.9: Eight solutions in a 4-dimensional space which are in eight different fronts.

Sol	Objective			
	O_1	O_2	O_3	O_4
s_1	1	2	3	4
s_2	1	2	3	4
s_3	1	2	3	4
s_4	1	2	3	4
s_5	5	6	7	8
s_6	5	6	7	8
s_7	5	6	7	8
s_8	5	6	7	8
s_9	9	10	11	12
s_{10}	9	10	11	12
s_{11}	9	10	11	12
s_{12}	9	10	11	12
s_{13}	13	14	15	16
s_{14}	13	14	15	16
s_{15}	13	14	15	16
s_{16}	13	14	15	16

(a) Objective values

Q_1	Q_2	Q_3	Q_4
s_1	s_1	s_1	s_1
s_2	s_2	s_2	s_2
s_3	s_3	s_3	s_3
s_4	s_4	s_4	s_4
s_5	s_5	s_5	s_5
s_6	s_6	s_6	s_6
s_7	s_7	s_7	s_7
s_8	s_8	s_8	s_8
s_9	s_9	s_9	s_9
s_{10}	s_{10}	s_{10}	s_{10}
s_{11}	s_{11}	s_{11}	s_{11}
s_{12}	s_{12}	s_{12}	s_{12}
s_{13}	s_{13}	s_{13}	s_{13}
s_{14}	s_{14}	s_{14}	s_{14}
s_{15}	s_{15}	s_{15}	s_{15}
s_{16}	s_{16}	s_{16}	s_{16}

(b) Sorted list

Figure A.10: Sixteen solutions in a 4-dimensional space which are in 4 front (Solutions in a front are duplicate).

Appendix A.3.2. Worst Case

Figure A.11 shows the worst case scenario for sixteen solutions when the number of objectives is 4. In this case, the first two objective values of each of the solutions in a particular front are the same. The last two objective values of the solutions in a front are such that we can declare that all the solutions in a particular front are non-dominated.

Sol	Objective			
	O_1	O_2	O_3	O_4
s_1	1	1	1	4
s_2	1	1	2	3
s_3	1	1	3	2
s_4	1	1	4	1
s_5	5	5	5	8
s_6	5	5	6	7
s_7	5	5	7	6
s_8	5	5	8	5
s_9	9	9	9	12
s_{10}	9	9	10	11
s_{11}	9	9	11	10
s_{12}	9	9	12	9
s_{13}	13	13	13	16
s_{14}	13	13	14	15
s_{15}	13	13	15	14
s_{16}	13	13	16	13

(a) Objective values

Q_1	Q_2	Q_3	Q_4
s_1	s_1	s_1	s_4
s_2	s_2	s_2	s_3
s_3	s_3	s_3	s_2
s_4	s_4	s_4	s_1
s_5	s_5	s_5	s_8
s_6	s_6	s_6	s_7
s_7	s_7	s_7	s_6
s_8	s_8	s_8	s_5
s_9	s_9	s_9	s_{12}
s_{10}	s_{10}	s_{10}	s_{11}
s_{11}	s_{11}	s_{11}	s_{10}
s_{12}	s_{12}	s_{12}	s_9
s_{13}	s_{13}	s_{13}	s_{16}
s_{14}	s_{14}	s_{14}	s_{15}
s_{15}	s_{15}	s_{15}	s_{14}
s_{16}	s_{16}	s_{16}	s_{13}

(b) Sorted list

Figure A.11: Sixteen solutions in a 4-dimensional space which are in 4 front (Worst case scenario).

Appendix A.3.3. Best Case

Figure A.12 shows the best case scenario for sixteen solutions when the number of objectives is 4. In this case, when the sorted list based on each objective is traversed in a row-wise manner, then before the second occurrence of a solution of a particular front, all the solutions of that front are traversed at least once. In Figure A.12, the first two objective values of the solutions allow us to declare that all the solutions in a particular front are non-dominated with each other. So $1 < \mathrm{X}_1 \leq 4$, $5 < \mathrm{X}_2 \leq 8$, $9 < \mathrm{X}_3 \leq 12$ and $13 < \mathrm{X}_4 \leq 16$. The values of $\mathrm{X}_1, \mathrm{X}_2, \mathrm{X}_3, \mathrm{X}_4$ are chosen such that the each solution in a front dominates all the solutions in its succeeding front.

Sol	**Objective**			
	$\boldsymbol{O_1}$	$\boldsymbol{O_2}$	$\boldsymbol{O_3}$	$\boldsymbol{O_4}$
s_1	1	4	X_1	X_1
s_2	4	1	X_1	X_1
s_3	2	3	1	X_1
s_4	3	2	X_1	1
s_5	5	8	X_2	X_2
s_6	8	5	X_2	X_2
s_7	6	7	5	X_2
s_8	7	6	X_2	5
s_9	9	12	X_3	X_3
s_{10}	12	9	X_3	X_3
s_{11}	10	11	9	X_3
s_{12}	11	10	X_3	9
s_{13}	13	16	X_4	X_4
s_{14}	16	13	X_4	X_4
s_{15}	14	15	13	X_4
s_{16}	15	14	X_4	13

(a) Objective values

$\boldsymbol{Q_1}$	$\boldsymbol{Q_2}$	$\boldsymbol{Q_3}$	$\boldsymbol{Q_4}$
s_1	s_2	s_3	s_4
s_3	s_4	NA_1	NA_1
s_4	s_3	NA_1	NA_1
s_2	s_1	NA_1	NA_1
s_5	s_6	s_7	s_8
s_7	s_8	NA_2	NA_2
s_8	s_7	NA_2	NA_2
s_6	s_5	NA_2	NA_2
s_9	s_{10}	s_{11}	s_{12}
s_{11}	s_{12}	NA_3	NA_3
s_{12}	s_{11}	NA_3	NA_3
s_{10}	s_9	NA_3	NA_3
s_{13}	s_{14}	s_{15}	s_{16}
s_{15}	s_{16}	NA_4	NA_4
s_{16}	s_{15}	NA_4	NA_4
s_{14}	s_{13}	NA_4	NA_4

(b) Sorted list

Figure A.12: Sixteen solutions in a 4-dimensional space which are in the 4 front (Best case scenario). $1 < X_1 \leq 4$, $5 < X_2 \leq 8$, $9 < X_3 \leq 12$ and $13 < X_4 \leq 16$. $NA_1 = \{s_1, s_2, s_3, s_4\}$, $NA_2 = \{s_5, s_6, s_7, s_8\}$, $NA_3 = \{s_9, s_{10}, s_{11}, s_{12}\}$, $NA_4 = \{s_{13}, s_{14}, s_{15}, s_{16}\}$ but in each sorted list, a solution occurs exactly once.