Divide-and-Conquer Based Non-dominated Sorting with Reduced Comparisons

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

[a]*Department of Computer Science & Engineering*
Indian Institute of Information Technology Guwahati, Guwahati, Assam – 781015, India
[b]*Departamento de Computación, CINVESTAV-IPN, Mexico City, Mexico*
[c]*Department of Computer Science & Engineering*
Indian Institute of Technology Patna, Patna, Bihar – 801106, India
[d]*Departamento de Sistemas, UAM-Azcapotzalco, Mexico City, Mexico*
(On sabbatical leave from Departamento de Computación
CINVESTAV-IPN, Mexico City, Mexico)

Abstract

Non-dominated sorting has attracted a lot of attention of the research community due to its use in solving multi- and many-objective optimization problems. In recent years, several approaches for non-dominated sorting have been proposed. In this paper, we have developed a non-dominated sorting framework, namely DCNSRC (Divide-and-Conquer based Non-dominated Sorting with Reduced Comparisons). Based on this framework, two approaches have been proposed by varying the search technique. These approaches perform a lower number of dominance comparisons than various other approaches. The duplicate solutions are also handled efficiently. These approaches save various comparisons while comparing the two solutions. The proposed approaches are validated using some theoretical analyses. The number of dominance comparisons performed by the proposed framework are theoretically analyzed in three different scenarios, both in the worst and the best cases. Experimental results on synthetic datasets and the benchmark problems show the superiority of the proposed approach over state-of-the-art algorithms.

Keywords: Evolutionary algorithm, Non-dominating sorting, Time complexity

1. Introduction

Non-dominated sorting is one of the key steps in Pareto-based multi-objective evolutionary algorithms (MOEAs). Non-dominated sorting sorts the solutions into different non-dominated fronts based on their dominance relationships. In

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

addition of being a key step in MOEAs, the application of the optimization algorithm based on non-dominated sorting can be found in many other fields such as reactive power planning [1], generation expansion planning [2, 3], feature selection for facial expression recognition [4], decision making in water distribution networks [5], vehicle routing problems [6], etc. In MOEAs, the set of solutions are often referred to as the population. Let $\mathbb{P} = \{s_1, s_2, \ldots, s_N\}$ be a population of N solutions where each solution $s_i, 1 \leq i \leq N$ is an n-dimensional vector $s_i = \left(s_i^1, s_i^2, \ldots, s_i^n\right)^T$. The objective function of s_i is $F(s_i) = (f_1(s_i), f_2(s_i), \ldots, f_M(s_i))^T$ where $f_m(s_i)$ is the objective value of solution s_i for m^{th} $(1 \leq m \leq M)$ objective. Let us assume that all the objectives are of minimization type. In non-dominated sorting, the solutions are sorted based on the dominance relationships among them, which is defined as follows. A solution s_i is said to dominate another solution s_j, denoted by $s_i \prec s_j$ if the two following conditions are satisfied:

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

Two solutions s_i and s_j are said to be non-dominated, if neither dominates the other, *i.e.*, neither $s_i \nprec s_j$ nor $s_j \nprec s_i$. In non-dominated sorting, population $\mathbb{P}$ of size N is divided into $K(1 \leq K \leq N)$ fronts $\mathcal{F} = \{F_1, F_2, \ldots, F_K\}$ arranged in decreasing order of their dominance. The division of these solutions in different fronts is such that all the following conditions are satisfied:

1. $\cup_{k=1}^{K} F_k = \mathbb{P}$
2. $\forall s_i, s_j \in F_k$: $s_i \nprec s_j$ and $s_j \nprec s_i,\ 1 \leq k \leq K$
3. $\forall s \in F_k, \exists s' \in F_{k-1}$: $s' \prec s,\ 2 \leq k \leq K$

There have been different approaches proposed in the past for non-dominated sorting. Srinivas *et al.* [7] proposed the first approach for non-dominated sorting. In this approach, a solution can be compared with other solutions a maximum of $N - 1$ times. Thus, its worst case time complexity is $\mathcal{O}(MN^3)$ with space complexity of $\mathcal{O}(N)$. The worst case occurs when all the solutions are in different fronts. The best case time complexity of this approach is $\mathcal{O}(MN^2)$ when all the solutions are non-dominated with each other. The fast non-dominated sorting [8] approach has improved the worst case time complexity to $\mathcal{O}(MN^2)$, but with an increase in the space complexity to $\mathcal{O}(N^2)$. In this approach, a solution is compared with other solutions only once. A recursive approach was developed by Jensen *et al.* [9] with a time complexity of $\mathcal{O}(N \log^{M-1} N)$. For two objectives, the time complexity of this approach is $\mathcal{O}(N \log N)$. This approach is not suitable when two solutions share the same value for an objective. This limitation of Jensen's approach is removed by Fortin *et al.* [10]. The average case time complexity of Fortin's approach is $\mathcal{O}(N \log^{M-1} N)$. However, the worst case time complexity of Fortin's approach is $\mathcal{O}(MN^2)$. Tang *et al.* [11] proposed a fast method for constructing the non-dominated set based on arena's principle. The best case time complexity of this approach is $\mathcal{O}(MN\sqrt{N})$ [12].

An efficient approach based on a divide-and-conquer strategy was proposed by Fang *et al.* [13]. This approach uses the dominance tree to reduce the number

of comparisons. The worst case time complexity of this algorithm is $\mathcal{O}(MN^2)$ when all the solutions are non-dominated with respect to each other. The lower bound time complexity of this algorithm is $\mathcal{O}(MN \log N)$. This approach considers one solution as dominated by another if both are identical. Two sorting algorithms (climbing sort and deductive sort) were proposed by McClymont *et al.* [14]. These algorithms use the dominance relation between the solutions to reduce the number of dominance comparisons. These algorithms have a worst case time complexity of $\mathcal{O}(MN^2)$. The space complexity of deductive sort is $\mathcal{O}(N)$, and its best case time complexity is $\mathcal{O}(MN\sqrt{N})$. Zhang *et al.* [12] proposed an efficient approach for non-dominated sorting, called efficient non-dominated sort (ENS). Two variants of ENS – one using sequential search (ENS-SS) and another using binary search (ENS-BS) were proposed. The worst case time complexity is $\mathcal{O}(MN^2)$ for these two variants. The best case time complexity of ENS-SS is $\mathcal{O}(MN\sqrt{N})$, whereas the best case time complexity of ENS-BS is $\mathcal{O}(MN \log N)$. Buzdalov *et al.* [15] proved that the time complexity of non-dominated sorting is $\mathcal{O}(N \log^{M-1} N)$. Mishra *et al.* [16, 17] proposed an approach based on a divide-and-conquer strategy. This algorithm has a best case time complexity of $\mathcal{O}(MN \log N)$ and a worst case time complexity of $\mathcal{O}(MN^2)$.

In some recently proposed approaches [18, 19, 20] when a solution is inserted in a front, it is not always compared with all the other solutions in that front. Based on this idea, an approach known as Best Order Sort (BOS) has been proposed, which is very efficient in terms of the number of dominance comparisons that it performs [18]. However, BOS is not suitable for duplicate solutions. Recently, BOS has been generalized to handle duplicate solutions by removing the comparison set concept[1]. In this paper, we call this modified BOS algorithm as BOS*. A tree based efficient non-dominated sorting approach known as T-ENS [19] is proposed with a worst case time complexity of $\mathcal{O}(MN^2)$. T-ENS has a better best case time complexity, which is $\mathcal{O}(MN \log N / \log M)$. Recently, an efficient non-dominated sorting approach that uses a non-dominated tree (ENS-NDT) [20] was developed with a worst case time complexity of $\mathcal{O}(MN^2)$. The best case time complexity of ENS-NDT is the same as ENS-BS, *i.e.*, $\mathcal{O}(MN \log N)$. Zhou *et al.* [21] have developed a dominance degree based approach for non-dominated sorting. The authors proposed the dominance degree matrix for the solutions in the population and also proposed an efficient approach to obtain this matrix. Based on the dominance degree matrix, an efficient approach called dominance degree approach for non-dominated sorting (DDA-NS) is proposed. In this approach, the comparisons between the objective function values of the solutions are only performed in the process of constructing the dominance degree matrix. Once the dominance degree matrix is obtained, the additional comparisons are required to assign solutions to different fronts. The additional comparisons are in the form of integer value comparisons between the elements of the dominance degree matrix. So,

[1] https://github.com/Proteek/Best-Order-Sort/

in this approach, two types of comparisons (objective value comparisons and integer value comparisons) occur. The average number of objective value comparisons required is $\mathcal{O}(MN \log N)$, and the number of integer value comparisons required is $\mathcal{O}(N^2)$. The number of integer additions is $\mathcal{O}(MN^2)$. Few other approaches like [22, 23, 24, 25] have been recently proposed for non-dominated sorting.

In some of the MOEAs, the population is decomposed into sub-populations and then non-dominated sorting is performed in each sub-population as done in [26]. The authors have used the fast non-dominated sorting approach proposed by Deb *et al.* [8] in [26]. However, other efficient sorting approaches can also be used to make non-dominated sorting more efficient.

There are some approaches [27, 28, 29, 30, 31, 32, 33] which have been proposed for incremental non-dominated sorting. This incremental non-dominated sorting is generally used in steady-state evolutionary algorithms [34, 35] where a solution needs to be inserted into the existing set of non-dominated fronts once a new solution is generated. Drozdik *et al.* [27] have developed an M-front based approach with worst case time complexity of $\mathcal{O}(MN^2)$ and best case time complexity of $\mathcal{O}(MN)$. The average case time complexity of this approach is $\mathcal{O}(M^2N^{2-\frac{1}{M-1}})$. An approach was proposed by Buzdalov *et al.* [28], which is only suitable for two objectives. A dominance matrix based approach was developed by Mishra *et al.* [31], which restricts the multiple comparisons between the same pairs of solutions in different generations of a steady-state evolutionary algorithm. By generalizing their previous approach for two objectives, Yakupov *et al.* [33] proposed an approach with time complexity $\mathcal{O}(N \log^{M-2} N)$. The incorporation of a new solution in the population does not change the entire non-domination level structure of the solutions. By exploiting this property, Li *et al.* [30] proposed an efficient non-domination level update approach to insert a new solution in the existing set of non-dominated fronts. The worst case time complexity of this approach is $\mathcal{O}(MN^2)$, however, the maximum number of dominance comparisons is $1/4N^2$. The best case time complexity of this approach is $\mathcal{O}(M)$. This best case time complexity is a great improvement over the best case time complexity of the non-dominated sorting approaches. Recently, Mishra *et al.* [32] also proposed a generalized approach for the non-domination level update problem with constant space complexity. A dominance binary tree based approach was also discussed. The worst and the best case time complexities of this approach are the same as the time complexity of the approach proposed by Li *et al.* [30].

Even though there have been several approaches proposed for non-dominated sorting, there is still a chance of improvement in terms of reducing the number of dominance comparisons. In general, when two solutions are compared to obtain their dominance relationships, all the objective values can be considered. Also, very few existing approaches handle duplicate solutions efficiently. So, we also focused on the non-dominated sorting problem to handle these issues in order to make the approach more efficient.

We have focused on developing a framework based on a divide-and-conquer strategy. Divide-and-conquer based approaches are easy to parallelize, so our approach also has this property. However, in this paper, we have not focused on the parallelism. There have been some approaches proposed based on a divide-and-conquer strategy such as [16, 17, 13]. However, in these approaches, while comparing two solutions, all the objective values between the solutions are compared. Also, before inserting a solution into a particular front, the solution is compared with all the solutions of that front. Duplicate solutions are also not handled efficiently in these approaches. In the current paper, a divide-and-conquer based strategy is developed, which can handle the drawbacks of the existing divide-and-conquer strategies. The key difference between other divide-and-conquer based approaches like [16, 17, 13] and ours is that when two solutions are compared to determine their dominance relationship, not all the objective values are considered. So, two solutions are compared efficiently. Apart from this difference from other divide-and-conquer based approaches, our proposed approach is also capable of handling duplicate solutions in the population efficiently. Also, before inserting a solution to a particular front, the solution is not compared with all the solutions of that front. However, in many approaches, a solution is compared with all the solutions of a front before being inserted into that front.

In this paper, we develop a framework which we call DCNSRC (Divide-and-Conquer based Non-dominated Sorting with Reduced Comparisons), which is proposed to reduce the number of dominance comparisons. This framework especially takes care of handling duplicate solutions efficiently. Also, in the proposed framework, when two solutions are compared to determine their dominance relationship, not all the objective values are considered. So, the solutions are also compared efficiently. Two variants of the developed framework are presented varying the search technique as in [12, 16]. We have theoretically analyzed the number of dominance comparisons performed by the proposed framework in different scenarios and observed that the number of dominance comparisons is less as compared to various other approaches. When all the solutions are non-dominated, then in the best case, the number of dominance comparisons can be zero. This can happen when $M \geq N$. In brief, the contributions in this paper are as follows:

- A framework for non-dominated sorting has been developed based on a divide-and-conquer strategy. We call this framework DCNSRC (Divide-and-Conquer based Non-dominated Sorting with Reduced Comparisons).
- The developed framework handles duplicate solutions efficiently.
- The number of dominance comparisons between the solutions is reduced.
- When two solutions are compared to determine their dominance relationship, not all the objective values are considered. So, the solutions are compared efficiently.
- Two approaches have been proposed based on the developed framework by varying the search technique as in [12].
- The number of dominance comparisons performed by DCNSRC in three

different scenarios in the worst and the best cases are also theoretically analyzed.

The rest of the paper is organized as follows. Section 2 discusses the proposed non-dominated sorting framework. An important component in the proposed framework is the merge procedure which is discussed in detail in Section 3. The complexity analysis of the framework is performed in Section 4. The experimental analysis is discussed in Section 5. Section 6 concludes the paper and provides some possible paths for future research.

Algorithm 1 DCNS Framework

Input: $\mathbb{P}$: Population of size N, M: Number of objectives
Output: Sorted solutions

/* - - - - - - - - - - First Phase Starts - - - - - - - - - - */
for each objective $j \in \{1, 2, \ldots, M\}$ **do**
  $Q_j \leftarrow$ Sort $\mathbb{P}$ based on the j^{th} objective
/* - - - - - - - - - - First Phase Ends - - - - - - - - - - */
/* - - - - - - - - - - Second Phase Starts - - - - - - - - - - */
/* -- Process to Find Duplicate Solutions Starts -- */
$sameAs_s \leftarrow \Phi$, $\forall s \in \mathbb{P}$ // Previous solution of s in Q_1 if it is same as s
$N' \leftarrow 1$ // Number of unique solutions in the population
for $i \leftarrow 2$ **to** N **do**
  if $Q_1(i)$ and $Q_1(i-1)$ are the same in terms of objective values **then**
    $sameAs_{Q_1(i)} \leftarrow Q_1(i-1)$ // $Q_1(i-1)$ is same as $Q_1(i)$
  else
    $N' \leftarrow N' + 1$
/* -- Process to Find Duplicate Solutions Ends -- */
$\mathbb{R} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_{N'}\}$ // Create N' *RC-matrices*
Initialize RC-matrices$(\mathbb{R}, M)$ // Using Algorithm 2
$\mathbb{F} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_{N'}\}$ // Create N' set of fronts
Initialize Set of Fronts$(\mathbb{F}, Q_1, Q_2, \ldots, Q_M)$ // Using Algorithm 3
/* -- Assignment of Solutions to their Respective Fronts Starts -- */
for $l \leftarrow 1$ to $\lceil \log_2 N' \rceil$ **do** // Consider each level
  for $i \leftarrow 1$ to $\lceil \frac{N'}{2^l} \rceil$ **do**
    $x \leftarrow 2^l \cdot i - \left(2^l - 1\right)$, $y \leftarrow x + 2^{l-1}$
    if $y \leq N'$ **then**
      Merge$(\mathbb{F}(x), \mathbb{F}(y), \mathbb{R}(x))$
/* -- Assignment of Solutions to their Respective Fronts Ends -- */
return $\mathbb{F}(1)$ // $\mathbb{F}(1)$ contains all the sorted solutions
/* - - - - - - - - - - Second Phase Ends - - - - - - - - - - */

2. Proposed Framework

A non-dominated sorting framework, namely DCNSRC (Divide-and-Conquer based Non-dominated Sorting with Reduced Comparisons) is proposed. The proposed framework consists of two-phases. The steps of the DCNSRC framework are summarized in Algorithm 1. The framework uses various symbols which are summarized in Table 1 along with their meaning.

2.1. First Phase: Pre-sorting

In the first phase, we sort the solutions based on each objective individually (lines $1-2$) as in [36, 18]. There are several advantages of sorting the solutions based on all the objectives. The first one is that the solution which comes later in the sorted list based on an objective does not dominate its previous solutions. Also, while ranking a solution, there is no need to compare the solution with all the solutions of a particular front. The dominance comparison between two solutions becomes efficient as there is no need to compare all the objective values of the two solutions.

While sorting the solutions based on the first objective, if two solutions are found to have the same value for that objective, then the second objective is taken into consideration. If the objective values for the second objective are the same for two solutions, then their sorting is done based on the third objective and so on. If the values of all the objectives for two solutions are the same, then any order of these two solutions can be followed.

We also need to sort the solutions based on the other objectives except for the first one. Let us sort the solutions based on the $m^{th}(2 \leq m \leq M)$ objective. While sorting the solutions based on the m^{th} objective, if two solutions are found to have the same value for that objective, then the sorted orders of the solutions based on the first objective are used to decide the ordering of these solutions based on the m^{th} objective.

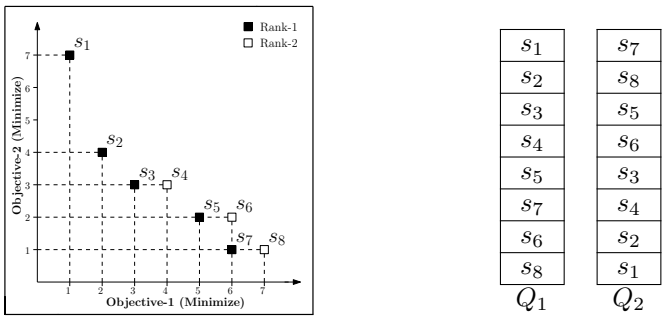
The sorted order of the solutions based on the j^{th} objective is denoted by $Q_j, 1 \leq j \leq M$. Thus, after the first phase, we get M sorted lists $Q_1, Q_2, \ldots, Q_M$, each of size N corresponding to each objective. Let the i^{th} solution of Q_j be denoted by $Q_j(i)$ where $1 \leq i \leq N$. If we consider any of these M sorted lists, then the solution which comes later in the sorted list cannot dominate its previous solution [12]. Thus, these sorted lists can help in reducing the number of dominance comparisons.

Example 1. *Consider a population with eight solutions in 2-dimensional objective space, as shown in Figure 1(a). These eight solutions need to be sorted based on both objectives. When the solutions are sorted based on the first objective (objective-1), then solutions s_6 and s_7 have the same value for the first objective. To decide the ordering between these two solutions, the values of the second objective are taken into consideration. As s_7 has a smaller value than s_6 for the second objective, so s_7 comes before s_6 in the sorted list based on the first objective. The sorted orders of the solutions based on the first objective, denoted by Q_1 are shown in Figure 1(b).*

Table 1: Symbols used in the paper.

Symbol	**Meaning of the symbol**
$\mathbb{P}$	Population of solutions
N	Number of solutions in the population
M	Number of objectives associated with each solution in the population
Q_j	Sorted order of the solutions based on the j^{th} objective
$Q_j(i)$	i^{th} solution of Q_j
N'	Number of unique solutions (in terms of objective values) in the population
Sorted matrix	A matrix of size $N \times M$ where the j^{th} column represents the sorted order of the solutions based on the j^{th} objective, *i.e.*, Q_j
$sameAs_s$	A variable to keep track of the previous solution of s in Q_1 if it is the same as s (in terms of objective values)
$\mathcal{F}$	Set of fronts
$\mathbb{F} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_{N'}\}$	Set of N' sets of fronts
$\mathbb{F}(i)$	Set of fronts in $\mathbb{F}$ which is at the i^{th} position, *i.e.*, $\mathcal{F}_i$
RC-matrix ($\mathcal{R}$)	Two-dimensional matrix which stores the solutions which have been ranked based on a particular objective
$\mathbb{R} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_{N'}\}$	Set of N' *RC-matrices*
$\mathbb{R}(i)$	*RC-matrix* in $\mathbb{R}$ which is at the i^{th} position, *i.e.*, $\mathcal{R}_i$
$\mathbb{R}(i)^r_j$	The cell of an *RC-matrix* corresponding to the i^{th} set of fronts which is in the r^{th} row and the j^{th} column
$first_s$	The objective list in which solution s is first found while traversing the *sorted matrix*
Pos_s	An array of size M associated with each solution $s \in \mathbb{P}$ which stores the position of s corresponding to each objective list in the *sorted matrix*
ObjList_s	A list associated with each solution $s \in \mathbb{P}$ which stores the objective list in the *sorted matrix* in which s has been traversed
Obj-Pos_s	An array of size M associated with each solution $s \in \mathbb{P}$ which stores the objective list and the position of s in which s has been traversed in the *sorted matrix* in the form of a pair. The objective list and the positions are stored in increasing order of the position.
α	Index of the front in $\mathcal{F}$ from where the solutions of front $F' \in \mathcal{F}'$ start comparison during the merging of the two sets of fronts $\mathcal{F}$ and $\mathcal{F}'$
$\Xi\,[1, 2, \ldots, K]$	Array of size K used to store the index of the fronts in $\mathcal{F}$ where each solution of front $F' \in \mathcal{F}'$ will be inserted during the merging of the two sets of fronts $\mathcal{F}$ and $\mathcal{F}'$
rank	Rank of a solution
hfi	A variable which stores the index of the front in $\mathcal{F}$ having the highest dominance in which the solutions of front $F' \in \mathcal{F}'$ are inserted during the merging of the two sets of fronts $\mathcal{F}$ and $\mathcal{F}'$

When the solutions are sorted based on the second objective (objective-2), then solutions s_3 and s_4 have the same value for the second objective. To decide the ordering between these two solutions, the sorted orders of the solutions based



(a) Eight solutions (b) Sorted list

Figure 1: Set of eight solutions along with the sorted solutions after sorting based on both the objectives.

on the first objective are used. In the sorted order of the solutions based on the first objective, s_3 comes before s_4, so s_3 precedes s_4 in the sorted list based on the second objective. Similarly, when the solutions are sorted based on the second objective, then solutions s_5 and s_6 have the same value for the second objective. To decide the ordering between these two solutions, the sorted orders of the solutions based on the first objective are used. In the sorted order of the solutions based on the first objective, s_5 comes before s_6, so s_5 precedes s_6 in the sorted list based on the second objective. In a similar way, when the solutions are sorted based on the second objective, then solutions s_7 and s_8 have the same value for the second objective. To resolve this issue, the sorted orders of the solutions based on the first objective are considered. In the sorted order of the solutions based on the first objective, s_7 comes before s_8, so s_7 precedes s_8 in the sorted list based on the second objective. The sorted order of the solutions based on the second objective, denoted by Q_2 is shown in Figure 1(b). □

2.2. Second Phase

In the second phase, the solutions are assigned to their respective fronts. This phase considers the sorted lists of solutions obtained from the first phase. This phase consists of various steps which are as follows.

i. Find duplicate solutions (line $3 - 9$)
ii. Create and Initialize *RC-matrices* (line $10 - 11$)
iii. Initialize set of fronts and obtain desired information for the solutions (line $12 - 13$)
iv. Sort the solutions (line $14 - 18$)

Now we discuss all these steps to understand the second phase.

2.2.1. Find Duplicate Solutions

We have focused on handling duplicate solutions efficiently, so we keep track of the duplicate solutions (in terms of objective values) in the population. For

this purpose, a variable $sameAs_s$ is associated with each solution $s \in \mathbb{P}$ to keep track of its previous solution in Q_1 if it is same as s in terms of objective values (see line 3). This variable helps in determining the duplicate solutions. The minimum number of unique solutions in the population can be 1, so we initialize the number of unique solutions, denoted as N' with 1 (line 4). When the solutions are sorted based on all the objectives, then duplicate solutions have the same order in all the objective lists. So, before assigning rank to the solutions, for each solution $s \in Q_1$, we keep the information whether the current solution is the same as its previous solution. If the current solution s is the same as its previous solution, then the previous solution of s is stored in variable $sameAs_s$ associated with s; otherwise, the number of unique solutions is incremented. The complete process of obtaining duplicate solutions and identifying the number of unique solutions is shown in lines $3-9$ of Algorithm 1.

The proposed framework is based on a divide-and-conquer strategy and there are N' unique solutions, so initially, we consider N' sets of fronts. A set of fronts can have multiple sub-fronts where each sub-front can have several solutions. All the fronts in any set of fronts are arranged in decreasing order of their dominance. Initially, a set of fronts contain only one solution, *i.e.*, each set of fronts has a single front, and this single front has only one solution. Let the set of these N' sets of fronts be denoted by $\mathbb{F} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_{N'}\}$ (see line 12 of Algorithm 1). The set of fronts in $\mathbb{F}$ which is at the i^{th} position is referred to as $\mathbb{F}(i)$. To sort the solutions, the merge operation is performed between two consecutive sets of fronts in a level by level manner. The number of levels $\mathcal{L}$ is $\log_2 N'$. At the l^{th} level, a total of $N'/2^l$ merge operations are performed. In the merge operation, all the solutions from the second set of fronts are inserted into their respective positions in the first set of fronts.

At the first level, the solutions from the set of fronts at index positions $2, 4, 6, \ldots$ are inserted into the set of fronts at index positions $1, 3, 5, \ldots$ respectively. So, after the merge operation at the first level, the set of fronts at index positions $2, 4, 6, \ldots$ will never be considered because the solutions of these sets of fronts are already inserted into another set of fronts in the merge operation. Similarly, at the second level, the solutions from the set of fronts at index positions $3, 7, 11, \ldots$ are inserted into the set of fronts at index positions $1, 5, 9, \ldots$, respectively. So, after the merge operation at the second level, the set of fronts at index positions $3, 7, 11, \ldots$ will never be considered. In general, at the l^{th} level, the set of fronts at index positions $2^{l-1}+1, 3 \cdot 2^{l-1}+1, 5 \cdot 2^{l-1}+1, \ldots$ are inserted into the set of fronts at index positions $1, 2^l+1, 2 \cdot 2^l+1, \ldots$, respectively. So, after the merge operation at the l^{th} level, the set of fronts at index positions $2^{l-1}+1, 3 \cdot 2^{l-1}+1, 5 \cdot 2^{l-1}+1, \ldots$ will never be considered.

Example 2. *Consider a population of size eight in 2-dimensional space, which is shown in Figure 1(a). So, $N = 8$ and $M = 2$. All the solutions are unique so $N' = 8$. As there are eight unique solutions, so there will be eight sets of fronts. Consider these eight sets of fronts $\{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_8\}$. As the number of sets of fronts is eight, so the merge operations are performed at three ($= \log 8$) levels. In the first level, four merge operations are performed. The sets of fronts at index*

positions 1 *and* 2 *are merged. Similarly, the set of fronts at index positions* 3 *and* 4 *are merged. The set of fronts at index positions* 5 *and* 6 *are merged and also the set of fronts at index positions* 7 *and* 8 *are merged. During the merge operations at the first level, all the solutions from the set of fronts at index positions* $2, 4, 6, 8$ *are inserted into the set of fronts at index positions* $1, 3, 5, 7$*, respectively. So, at the second level, the set of fronts at index positions* $2, 4, 6, 8$ *are not considered. At the second level, two merge operations are performed. At the second level, the sets of fronts at index positions* 1 *and* 3 *are merged, and the set of fronts at index positions* 5 *and* 7 *are merged. During the merge operations at the second level, all the solutions from the set of fronts at index positions* $3, 7$ *are inserted into the set of fronts at index positions* $1, 5$*, respectively. So at the third level, the set of fronts at index positions* $3, 7$ *are not considered. At the third level, only one merge operation is performed. The sets of fronts at index positions* 1 *and* 5 *are merged to get the final set of fronts. This complete procedure is shown in Figure 2.* □

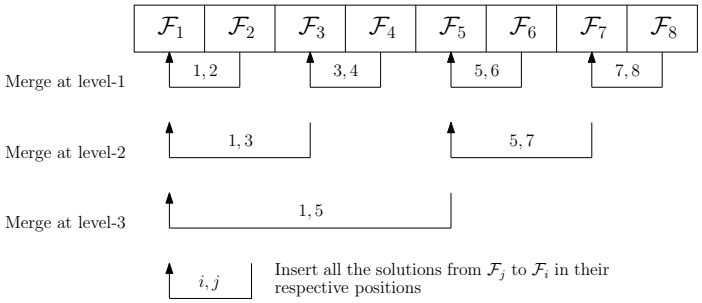


Figure 2: Merge operations between different sets of fronts

Algorithm 2 INITIALIZE RC-MATRICES($\mathbb{R}, M$)

Input: $\mathbb{R} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_{N'}\}$: Set of N' *RC-matrices* which need to be initialized, M: Number of objectives

Output: Set of N' *RC-matrices* after initialization

1: **for** $l \leftarrow 1$ to $\lceil \log_2 N' \rceil$ **do**
2: $r \leftarrow 2^{l-1}$, $incr \leftarrow 2^l$
3: **for** $i \leftarrow r + 1$ to N' **do**
4: $\mathbb{R}(i)^k_m \leftarrow \emptyset$, $\forall k = 1, 2, \ldots, r$, $\forall m = 1, 2, \ldots, M$
5: $i \leftarrow i + incr$
6: $\mathbb{R}(1)^k_m \leftarrow \emptyset$, $\forall k = 1, 2, \ldots, N'$, $\forall m = 1, 2, \ldots, M$

2.2.2. Create and Initialize RC-matrices

Besides handling duplicate solutions efficiently, we also focus on reducing the number of dominance comparisons. For this purpose, we keep the set of solutions which have been ranked based on a particular objective in the form

of a matrix as in [18]. Let us call this matrix '*RC-matrix*'. As our approach is based on a divide-and-conquer strategy, so initially, a *RC-matrix* corresponding to each set of fronts is created (see line 10 of Algorithm 1). Thus, there will be a total of N' *RC-matrices*. The process to initialize these N' *RC-matrices* is shown in Algorithm 2. Let the set of N' *RC-matrices* be denoted by $\mathbb{R} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_{N'}\}$. The *RC-matrix* in $\mathbb{R}$ which is at the i^{th} position is referred to as $\mathbb{R}(i)$. *RC-matrix* $\mathbb{R}(i)$ is the matrix corresponding to the set of fronts $\mathbb{F}(i)$. The cell of an *RC-matrix* corresponding to the i^{th} set of fronts which is in the r^{th} row and the j^{th} column is denoted by $\mathbb{R}(i)^r_j$. This cell stores the set of solutions with rank r which have been ranked based on the j^{th} objective in the i^{th} set of fronts. The number of columns in each of the N' *RC-matrices* is fixed to M because a solution can be ranked based on any of the M objectives. However, the number of rows varies between 1 to N' depending on the position of the *RC-matrix* in $\mathbb{R}$.

The maximum number of fronts in a set of fronts occurs when all the solutions in that set of fronts are in different fronts, *i.e.*, if there are k solutions then there will be k fronts. The merge operation is performed in a level by level manner. Initially, each set of fronts has a single front which contains a single solution. After the merge operation at the first level, the set of fronts at index positions $2, 4, 6, \ldots$ will never be considered. The number of fronts in the set of fronts at these index positions is 1. Thus, the number of rows in the *RC-matrices* corresponding to these index positions is 1. After the merge operation at the second level, the set of fronts at index positions $3, 7, 11, \ldots$ will never be considered. The maximum number of fronts in the set of fronts at these index positions is 2. Thus, the number of rows in the *RC-matrices* corresponding to these index positions is 2. In general, after the merge operation at the l^{th} level, the set of fronts at index positions $2^{l-1}+1, 3\cdot 2^{l-1}+1, 5\cdot 2^{l-1}+1, \ldots$ will never be considered. The maximum number of fronts in the set of fronts at these index positions is 2^{l-1}. Thus, the number of rows in the *RC-matrices* corresponding to these index positions is 2^{l-1}. At the first index position, the maximum number of fronts in the set of fronts can be N'. So, the number of rows in the *RC-matrix* at the first index position is N'. The space required for storing N' matrices of different sizes is as follows. $M\left(N' + \sum_{l=1}^{\mathcal{L}} \frac{N'}{2^l} \cdot 2^{l-1}\right) = \mathcal{O}(MN \log N)$.

Example 3. *Consider a population of size eight in 2-dimensional space, which is shown in Figure 1(a). So, $N = 8$ and $M = 2$. All the solutions are unique so $N' = 8$. These eight solutions are considered as a set of fronts. Thus, there will be eight RC-matrices of different sizes corresponding to each set of fronts.*

Initially, each set of fronts has a single front which contains a single solution. After the merge operation at the first level, the set of fronts at index positions $2, 4, 6, 8$ will never be considered. The number of fronts in the set of fronts at these index positions is 1. Thus, the number of rows in the RC-matrices corresponding to these index positions is 1. After the merge operation at the second level, the set of fronts at index positions $3, 7$ will never be considered. The maximum number of fronts in the set of fronts at these index positions is 2. Thus, the number of rows in the RC-matrices corresponding to these index

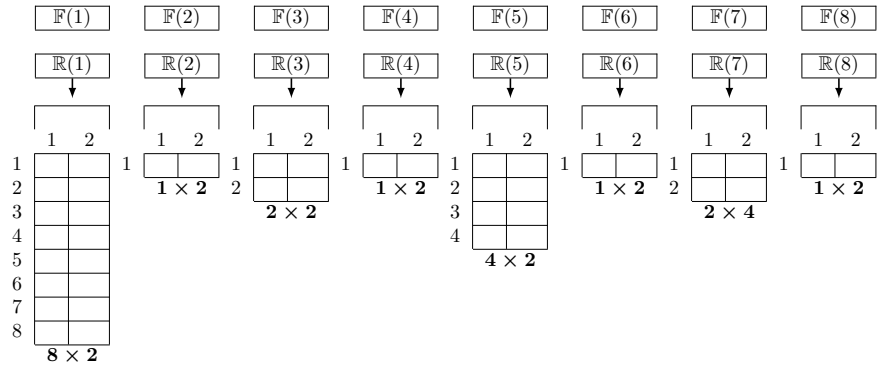


Figure 3: Set of eight *RC-matrices* of different sizes corresponding to eight sets of fronts. *RC-matrix* $\mathbb{R}(i)$ corresponds to set of fronts $\mathbb{F}(i)$. Each cell in these matrices represents an empty set $\mathbb{R}(i)_j^r$ which denotes the set of solutions with rank r which have been ranked based on the j^{th} objective in the i^{th} set of fronts ($1 \leq i \leq 8, 1 \leq r \leq 8, 1 \leq j \leq 2$).

positions is 2. *After the merge operation at the third level, the set of fronts at index position* 5 *will never be considered. The maximum number of fronts in the set of fronts at this index position is* 4. *Thus, the number of rows in the RC-matrix corresponding to this index position is* 4. *At the first index position, the maximum number of fronts in the set of fronts can be* 8. *So, the number of rows in the RC-matrix at the first index position is* 8. *These eight RC-matrices are shown in Figure 3.* □

2.2.3. Initialize Sets of Fronts and Obtain Desired Information for the Solutions

As our approach is based on a divide-and-conquer strategy, so initially we create N' sets of fronts (see line 12 of Algorithm 1). The process to initialize these N' sets of fronts is shown in Algorithm 3. Along with initializing the sets of fronts, information associated with each solution is also obtained in Algorithm 3.

The combinations of M sorted lists based on each objective $Q_1, Q_2, \ldots, Q_M$ can be thought of as a matrix of size $N \times M$ where the m^{th} column represents Q_m. We call this matrix '*sorted matrix*'. For sorting purposes, the solutions are considered in the *sorted matrix* in a row-wise manner, starting from the first row to the last. Each row is traversed from left to right. As soon as a solution is traversed for the first time in the *sorted matrix*, it is considered as a set of fronts (line $12 - 14$). In this set of fronts, there is only a single front which contains only one solution. Therefore, after traversing all the solutions at least once, we get N' sets of fronts corresponding to each of the N' unique solutions. While traversing the *sorted matrix*, we obtain some information for each solution s as detailed next.

- $\boldsymbol{first_s}$**:** A variable associated with each solution $s \in \mathbb{P}$ which denotes the objective list in which s is first found while traversing the *sorted matrix*.
- $\mathbf{Pos_s}$**:** An array of size M associated with each solution $s \in \mathbb{P}$ which stores the position of s corresponding to each column (objective list) in

Algorithm 3 INITIALIZE SET OF FRONTS($\mathbb{F}, Q_1, Q_2, \ldots, Q_M$)

Input: $\mathbb{F} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_{N'}\}$: Set of N' sets of fronts which need to be initialized, $Q_1, Q_2, \ldots, Q_M$: Sorted list of solutions based on each objective
Output: Set of N' set of fronts after initialization

1: $count \leftarrow 0$
2: $\chi[1, 2, \ldots, N'] \leftarrow \emptyset$
3: **for** $i \leftarrow 1$ **to** N **do**
4: **for** $j \leftarrow 1$ **to** M **do**
5: $s \leftarrow Q_j(i)$ // Take the i^{th} solution from Q_j
6: $\text{Pos}_s(j) \leftarrow M(i-1) + j$
7: $\text{ObjList}_s \leftarrow \text{ObjList}_s \cup \{j\}$
8: $\text{Obj-Pos}_s \leftarrow \text{Obj-Pos}_s \cup \{j, \text{Pos}_s(j)\}$
9: **if** s is traversed for the first time in *sorted matrix* **then**
10: $first_s \leftarrow j$ // Objective list where s occurs the first time
11: **if** $sameAs_s = \Phi$ **then** // Previous solution of s in Q_1 is different
12: $F \leftarrow \{s\}$ // Consider solution as a front
13: $\mathcal{F} \leftarrow \{F\}$ // Consider front as a set of fronts
14: $\mathbb{F} \leftarrow \mathbb{F} \cup \{\mathcal{F}\}$
15: $count \leftarrow count + 1$
16: $\mathbb{R}(count)_j^1 \leftarrow \{s\}$
17: $\chi(s) \leftarrow count$
18: **else**
19: $\mathbb{R}(\chi(sol))_j^1 \leftarrow \{s\}$
20: **if** $count = N'$ **then** // All the solutions have been traversed once
21: **BREAK**
22: **for** $k \leftarrow i + 1$ **to** N **do**
23: **for** $j \leftarrow 1$ **to** M **do**
24: $s \leftarrow Q_j(k)$ // Take the k^{th} solution from Q_j
25: $\text{Pos}_s(j) \leftarrow M(k-1) + j$
26: $\text{Obj-Pos}_s \leftarrow \text{Obj-Pos}_s \cup \{j, \text{Pos}_s(j)\}$

the *sorted matrix*. In the *sorted matrix*, each solution occurs exactly once in each column. So, the position of a solution in the *sorted matrix*, which is in the i^{th} row and the j^{th} column is obtained as $M(i-1)+j$ where $1 \leq i \leq N$ and $1 \leq j \leq M$. $\text{Pos}_s(j)$ stores the position of s in the j^{th} column.

- **ObjList_s:** A list associated with each solution $s \in \mathbb{P}$ which stores the column (objective list) in the *sorted matrix* in which s has been traversed.
- **Obj-Pos_s:** An array of size M associated with each solution $s \in \mathbb{P}$ which stores the objective list and the position of s in which s has been traversed in the *sorted matrix* in the form of a pair. The objective list and the positions are stored in increasing order of the position.
  i. $\text{Obj-Pos}_s(i)$.obj: The objective list where s is traversed for the i^{th}

time in the *sorted matrix*.

ii. Obj-Pos$_s(i)$.pos: The value of position for s when it is traversed for the i^{th} time in the *sorted matrix*.

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

8×2

(a) *Sorted Matrix* derived from Q_1 and Q_2

1	2
3	4
5	6
7	8
9	10
11	12
13	14
15	16

8×2

(b) Position of each of the solutions in *sorted matrix*

first$_s$	1	2	1	2	1	2	1	2
Pos$_s$	1 16	11 2	3 14	15 4	5 10	9 6	7 12	13 8
ObjList$_s$	1	2	1	2	1	2	1	2
Obj-Pos$_s$	$(1,1)$ $(2,16)$	$(2,2)$ $(1,11)$	$(1,3)$ $(2,14)$	$(2,4)$ $(1,15)$	$(1,5)$ $(2,10)$	$(2,6)$ $(1,9)$	$(1,7)$ $(2,12)$	$(2,8)$ $(1,13)$
Solutions	s_1 $Q_1(1)$	s_7 $Q_2(1)$	s_2 $Q_1(2)$	s_8 $Q_2(2)$	s_3 $Q_1(3)$	s_5 $Q_2(3)$	s_4 $Q_1(4)$	s_6 $Q_2(4)$

(c) The order of solutions after traversing the *sorted matrix* in Figure 4(a) in a row-wise manner along with *first*$_s$, Pos$_s$ and ObjList$_s$ for each solution s.

Figure 4: Obtaining the order of solutions for non-dominated sorting along with *first*$_s$, Pos$_s$ and ObjList$_s$ for each solution s.

Example 4. *Consider a population with eight solutions, as shown in Figure 1(a). These solutions are sorted based on each objective. The sorted lists of solutions based on both objectives are reported in Figure 1(b). Figure 4(a) shows the sorted solutions in the form of a matrix. The position of each cell in this sorted matrix is shown in Figure 4(b). Initially, solution s_1 is traversed in the sorted matrix (column-1) and then, solution s_7 is traversed (column-2). After this, the solutions in the second row are traversed and so on. When the solutions in the fourth row are traversed, then, as soon as the solution s_6 is traversed in column 2, we stop the traversal process as all the solutions have been traversed at least once. In this traversal, first$_s$, Pos$_s$, ObjList$_s$ and Obj-Pos$_s$ are obtained. Again, we traverse the solutions in the remaining rows of the sorted matrix to obtain Pos$_s$ and Obj-Pos$_s$.*

Solutions s_1, s_2, s_3 and s_4 have been first traversed in the first column of the sorted matrix so that their corresponding $first_s$ value is 1. *Similarly, s_5, s_6, s_7 and s_8 have been first traversed in the second column of the sorted matrix so their corresponding $first_s$ value is* 2.

The position of s_1 corresponding to the first column is 1 *and the one corresponding to the second column is* 16. *So $Pos_{s_1} = \{1, 16\}$. The position of s_7 corresponding to the first column is* 11 *and the one corresponding to the second column is* 2. *So, $Pos_{s_7} = \{11, 2\}$. Similarly, Pos_s corresponding to the rest of the solutions can be obtained.*

Solutions s_1, s_2, s_3 and s_4 have been first traversed in the first column of the sorted matrix so their corresponding $ObjList_s$ contains 1. *Similarly, s_5, s_6, s_7 and s_8 have been first traversed in the second column of the sorted matrix so their corresponding $ObjList_s$ contains* 2. *After traversing all the solutions at least once in the 4^{th} row, no further objective value is added to $ObjList_s$ of the solution s.*

Solution s_1 is first traversed in the first column of the sorted matrix and its corresponding position is 1. *Solution s_1 is again traversed in the second column and then its corresponding position is* 16. *Thus, Obj-$Pos_{s_1} = \{\{1, 1\}, \{2, 16\}\}$. Solution s_7 is first traversed in the second column of the sorted matrix and its corresponding position is* 2. *Solution s_7 is again traversed in the first column and then its corresponding position is* 11. *Thus, Obj-$Pos_{s_7} = \{\{2, 2\}, \{1, 11\}\}$. Similarly, Obj-$Pos_s$ corresponding to the rest of the solutions can be obtained.*

The order in which the solutions are obtained from the traversal of the sorted matrix (in Figure 4(a)) is shown in Figure 4(c). This figure also shows the values of $first_s$, Pos_s, $ObjList_s$ and Obj-Pos_s for each solution, s. □

2.2.4. Sort the Solutions

After initializing the set of fronts and *RC-matrices*, solutions are assigned to their respective fronts using merge operations between different sets of fronts at $\log N'$ levels. At each point in time during the merge operations, the solutions in the set of fronts are arranged in decreasing order of their dominance. So, we have used the concepts of local and global ranks which are described as follows:

Definition 1 (Local Rank). *The rank of a solution in the set of fronts is considered as its local rank if the set of fronts does not contain all the solutions in the population which take part in non-dominated sorting. Thus, the local rank of a solution is the actual rank considering only those solutions which are in the set of fronts.*

Definition 2 (Global Rank). *The rank of a solution in the set of fronts is considered as its global rank if the set of fronts contains all the solutions which take part in non-dominated sorting.*

Initially, each set of fronts has a single solution, so the local rank of each solution s is 1 and the solution is inserted in $\mathbb{R}(i)_m^1$ (line 16 of Algorithm 3) corresponding to the $m(= first_s)^{th}$ objective for the i^{th} set of fronts. The actual

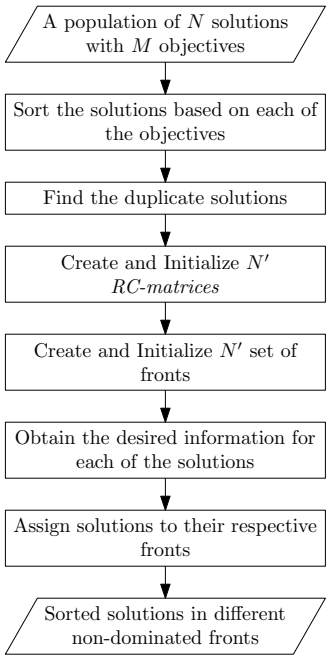


Figure 5: Flowchart of the proposed DCNSRC framework

sorting is performed considering N' sets of fronts. The merge operation is performed in a level by level manner and the set of fronts at the first index position contains all the solutions at the end of all the merge operations. So, as we proceed to the last level, the local rank of the solutions in the first set of fronts converges to the global rank. The flowchart of the proposed framework is provided in Figure 5.

Example 5. *Consider the population shown in Figure 1(a). The* sorted matrix *obtained from this population is shown in Figure 4(a). The order in which the solutions are obtained from the traversal of the sorted matrix is shown in Figure 4(c). As the number of unique solutions is eight, so initially, these eight solutions are considered as a set of fronts. These eight sets of fronts are shown in Figure 6 along with their corresponding RC-matrices. As the number of solutions is eight, so the merge operations will be performed at three different levels. The working flow of the proposed framework considering these eight solutions is shown in Figure 6. Along with the updated set of fronts, the updated RC-matrix is also shown in this figure.* □

The main procedure in this framework is the merge procedure. In the next section we discuss it in a detailed manner.

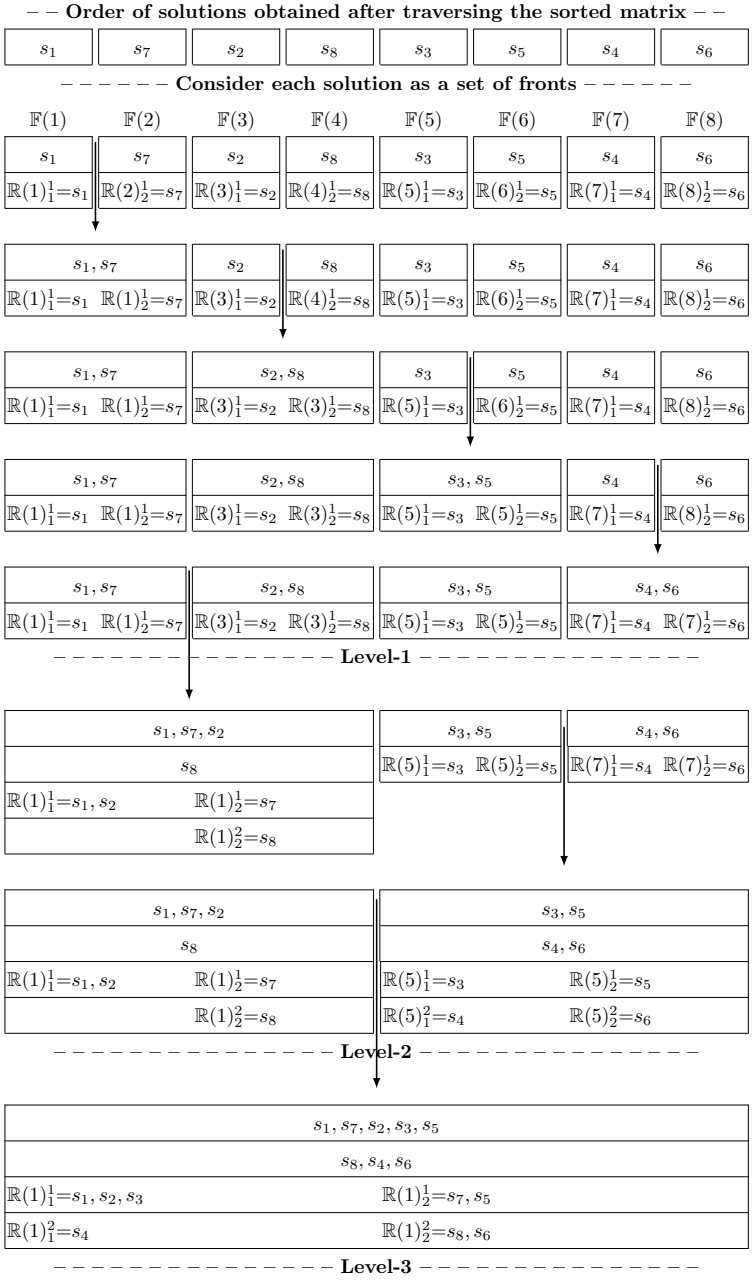


Figure 6: Working flow of the proposed framework. This working flow considers the same solutions as shown in Figure 1(a). ↓ indicates the merge operation between immediate left and right set of fronts. $s_i, \ldots, s_j$ represents that these solutions are non-dominated.

3. Merge Procedure

In this section we discuss the merge procedure in detail. This procedure merges two sets of fronts, *i.e.*, inserts all the solutions of the second set of fronts into the first set of fronts at their respective positions. The normal operation of merge sort cannot be directly applied here because of the dominance relationship. The dominance relationship does not follow the transitivity property (when two solutions are non-dominated with a particular solution, then it does not imply that both solutions are also non-dominated). So, the merge procedure is different than the usual merge operation in merge sort.

Let us have two sets of fronts $\mathcal{F} = \{F_1, F_2, \ldots, F_P\}$ and $\mathcal{F}' = \{F'_1, F'_2, \ldots, F'_Q\}$ which need to be merged at a particular level in the DCNSRC framework. Let the cardinality of front $F_p (1 \leq p \leq P)$, *i.e.*, $|F_p| = n_p$ and the cardinality of front $F'_q (1 \leq q \leq Q)$, *i.e.*, $|F'_q| = n_q$. The merge procedure merges these two sets of fronts $\mathcal{F}$ and $\mathcal{F}'$ using Algorithm 4. In the merge procedure, all the solutions of $\mathcal{F}'$ are inserted into their respective positions in $\mathcal{F}$. During the merge procedure, the *RC-matrix* corresponding to the set of fronts $\mathcal{F}$ is also considered so that after inserting all the solutions from $\mathcal{F}'$ into $\mathcal{F}$, the *RC-matrix* corresponding to $\mathcal{F}$ is also updated. In the merge procedure, the solutions of each front from $\mathcal{F}'$ are inserted in $\mathcal{F}$ sequentially starting from F'_1 to F'_Q. The solutions of a front $F' \in \mathcal{F}'$ are inserted in $\mathcal{F}$ using the INSERT() procedure which is summarized in Algorithm 5.

Algorithm 4 MERGE($\mathcal{F}, \mathcal{F}', \mathcal{R}$)

Input:

- $\mathcal{F} = \{F_1, F_2, \ldots, F_P\}$: First set of fronts
- $\mathcal{F}' = \{F'_1, F'_2, \ldots, F'_Q\}$: Second set of fronts whose solutions will be inserted into $\mathcal{F}$
- $\mathcal{R}$: *RC-matrix* corresponding to $\mathcal{F}$

Output: Updated $\mathcal{F}$ after insertion of all the solutions from $\mathcal{F}'$ into $\mathcal{F}$

1: $\alpha \leftarrow 0$
2: **for each** front $F' \in \mathcal{F}'$ **do**
3: $\quad \alpha \leftarrow$ INSERT($\mathcal{F}, F', \mathcal{R}, \alpha + 1$) // Insert all the solutions of F' in $\mathcal{F}$
4: $\quad$ **if** $\alpha = |\mathcal{F}|$ **then**
5: $\quad\quad$ **BREAK**
6: **for each** remaining front $F' \in \mathcal{F}'$ **do** // Add the remaining fronts of $\mathcal{F}'$ to $\mathcal{F}$ without comparison
7: $\quad \mathcal{F} \leftarrow \mathcal{F} \cup \{F'\}$ // Add F' to $\mathcal{F}$
8: $\quad \alpha \leftarrow \alpha + 1$ // Local rank of the solutions of F' in $\mathcal{F}$
9: $\quad$ **for each** solution $s' \in F'$ **do** // Update *RC-matrix* $\mathcal{R}$
10: $\quad\quad \forall m \in \text{ObjList}_{s'} : \mathcal{R}^{\alpha}_{m} \leftarrow \mathcal{R}^{\alpha}_{m} \cup \{s'\}$

Our aim is to reduce the unnecessary dominance comparisons between the solutions. For this purpose, we have considered the dominance relationships which are discussed in [16]. These relationships are very useful when the number of fronts is large in number. In the merge procedure, when the solutions of

front F'_1 are inserted in $\mathcal{F}$, the solutions of F'_1 determine their positions in $\mathcal{F}$ by comparing with solutions of each of the fronts in $\mathcal{F}$ starting from the first front F_1. However, when the solutions of the next front (*i.e.*, F'_2) are inserted in $\mathcal{F}$, then these solutions do not start comparing with the solutions of the first front in $\mathcal{F}$ because of the dominance relationship [16]. Let us assume that the index of the front in $\mathcal{F}$ from where the solutions of front $F' \in \mathcal{F}'$ start comparison be denoted by α. Thus, the solutions of front F' start comparison with front F_α in $\mathcal{F}$.

After inserting all the solutions of a front F' in $\mathcal{F}$, Algorithm 5 returns the index of the front with the highest dominance in $\mathcal{F}$, where the solutions of front F' have been inserted. This index is stored in variable α in Algorithm 4. In the merge procedure, when the next front of $\mathcal{F}'$ is inserted into $\mathcal{F}$, then the solutions of this front start dominance comparison with front $F_{\alpha+1}$ because of the dominance relationship. Whenever a front F' is inserted into $\mathcal{F}$, the value of α is updated. Thus, in the merge procedure, the solutions of a front are not compared with the solutions of each of the fronts in $\mathcal{F}$, thereby making the merge procedure efficient.

After insertion of a front F' in $\mathcal{F}$, if the value of α is equal to the cardinality of $\mathcal{F}$ (line 4 in Algorithm 4), then all the remaining fronts of $\mathcal{F}'$ are directly added to $\mathcal{F}$ without performing any dominance comparison with the solutions of $\mathcal{F}$ because of the dominance relationship (lines $6-10$ of Algorithm 4). When the solutions of the fronts are directly added to $\mathcal{F}$, then *RC-matrix* corresponding to $\mathcal{F}$, *i.e.*, $\mathcal{R}$ is also updated in lines $9-10$ of Algorithm 4. For each added front F', the index of the front in $\mathcal{F}$ where the solutions of F' are added, is obtained. This index is basically the local rank of the solutions of front F' in $\mathcal{F}$. This index is obtained by incrementing the value of α by 1. Each solution $s' \in F'$ is added to $\mathcal{R}^\alpha_m$ for each objective m where the solution has been traversed in the *sorted matrix*, *i.e.*, $m \in \text{ObjList}_{s'}$.

Illustration of the* INSERT() *procedure. This procedure inserts the solutions of a particular front $F' \in \mathcal{F}$ in $\mathcal{F}$. Once all the solutions of front F' have been inserted, this procedure returns the index of the front with the highest dominance in $\mathcal{F}$, where the solutions of front F' have been inserted. Let the index of fronts in $\mathcal{F}$ where the solutions of F' are inserted be $\iota_1, \iota_2, \ldots, \iota_{n_q}$ (when more than one solution is inserted into a single front, then the index of the front for these solutions will be same). If any solution creates a new front, then the index of this newly created front will be treated as $P+1$ where P is the number of fronts in $\mathcal{F}$ before insertion of front F'. The index of the front having highest dominance in which the solutions of front F' are inserted is $\xi = \min(\iota_1, \iota_2, \ldots, \iota_{n_q})$.

To keep track of this front index, a variable *hfi* (highest dominance front index) is considered. The insertion of a front F' in $\mathcal{F}$ can add one extra front. Thus, the value of *hfi* is initialized to $P+1$ (line 2) which is the maximum possible value of *hfi*.

An array Ξ of size $|F'|$ is considered to store the index of the front in $\mathcal{F}$ where the solutions of front F' will be inserted. This array is required to update the

Algorithm 5 INSERT($\mathcal{F}, F', \mathcal{R}, \alpha$)

Input:

- $\mathcal{F} = \{F_1, F_2, \ldots, F_P\}$: Set of fronts where all the solutions of a front need to be inserted
- F': A front whose solutions need to be inserted into $\mathcal{F}$
- $\mathcal{R}$: *RC-matrix* corresponding to $\mathcal{F}$
- α: Index of the front in $\mathcal{F}$ from where the solutions of F' start comparison

Output: Updated $\mathcal{F}$ after insertion of all the from $\mathcal{F}'$ into $\mathcal{F}$

1: $P \leftarrow |\mathcal{F}|$ // Number of fronts in $\mathcal{F}$
2: $hfi \leftarrow P + 1$
3: $\Xi[1, 2, \ldots, |F'|] \leftarrow \emptyset$ // Array of size $|F'|$ used to store the index of the fronts in $\mathcal{F}$ where each solution of front F' will be inserted
4: **for each** solution $s' \in F'$ **do**
5: $hfi \leftarrow$ INSERT-SS($\mathcal{F}, s', \mathcal{R}, \alpha, hfi, \Xi$) // Insert solution s' in $\mathcal{F}$
6: **for each** solution $s' \in F'$ **do** // Update $\mathcal{R}$
7: $rank \leftarrow \Xi(s')$
8: $\forall m \in \text{ObjList}_{s'} : \mathcal{R}_m^{\text{rank}} \leftarrow \mathcal{R}_m^{\text{rank}} \cup \{s'\}$
9: **return** *hfi*

RC-matrix corresponding to $\mathcal{F}$ after inserting all the solutions of front F'. All the solutions of F' are inserted one by one to their respective position in $\mathcal{F}$ (line $4 - 5$ of Algorithm 5). Once all the solutions of front F' have been inserted into $\mathcal{F}$, then $\mathcal{R}$ corresponding to $\mathcal{F}$ will be updated. For this purpose, the rank of each inserted solution $s' \in F'$ in $\mathcal{F}$ is obtained from Ξ. Let the rank of the inserted solution be '*rank*'. After obtaining the rank of the inserted solution, the solution is added to $R(x)_m^{\text{rank}}$ for each objective m where the solution s' has been traversed in the *sorted matrix*, *i.e.*, $m \in \text{ObjList}_{s'}$.

We can also avoid using array Ξ and update $\mathcal{R}$ as a solution is being inserted into $\mathcal{F}$. However, this will increase the number of dominance comparisons when multiple solutions from F' which have been first found on the same objective list (*i.e.*, have the same value of $first_{s'}$) are inserted into the same front in $\mathcal{F}$.

The insertion of a solution $s' \in F'$ in $\mathcal{F}$ can be performed either by a sequential search based technique or by a binary search based technique as in [12, 16]. The pseudo-code for sequential search based insertion is given in Algorithm 6 and the pseudo-code for binary search based insertion is given in Algorithm 8.

3.1. Sequential Search Based Insertion

This procedure inserts a solution $s' \in F'$ into $\mathcal{F}$ using a sequential search based technique. In this technique, a solution s' is compared with the solutions of each of the fronts in $\mathcal{F}$ starting from F_α to F_P in a sequential manner. The solutions are compared efficiently using the DOMINATIONCHECK() procedure which is described in Algorithm 7. Let $m = first_{s'}$. Here, solution s' is not

Algorithm 6 INSERT-SS($\mathcal{F}, s', \mathcal{R}, \alpha, hfi, \Xi[\ ]$)

Input:

- $\mathcal{F} = \{F_1, F_2, \ldots, F_P\}$: Set of fronts where a solution needs to be inserted
- s': A solution which needs to be inserted into $\mathcal{F}$
- $\mathcal{R}$: *RC-matrix* corresponding to $\mathcal{F}$
- α: Index of the front in $\mathcal{F}$ from where solution s' start comparison
- *hfi*: Index of the highest dominance front in $\mathcal{F}$ where the previous solutions of a front $F'(s' \in F')$ have been inserted
- $\Xi[\ ]$: An array which stores the index of the front in $\mathcal{F}$ where s' will be inserted

Output: Updated $\mathcal{F}$ after insertion of a solution s' into $\mathcal{F}$

1: *isInserted* $\leftarrow$ FALSE // s' is not yet inserted
2: $m \leftarrow first_s$ // Objective list where s' occurs first
3: **for** $p \leftarrow \alpha$ to P **do**
4: *isDominated* $\leftarrow$ FALSE
5: **for** each solution $s \in \mathcal{R}^p_m$ **do**
6: **if** $\text{Pos}_s(m) < \text{Pos}_{s'}(m)$ **and** DOMINATIONCHECK($s', s, \text{Pos}_{s'}(m)$) = TRUE **then**
7: *isDominated* $\leftarrow$ TRUE
8: **BREAK** // Check for next front in $\mathcal{F}$
9: **if** *isDominated* = FALSE **then** // s' is non-dominated with all the solutions of F_p
10: $F_p \leftarrow F_p \cup \{s'\}$ // Insert s' in F_p
11: *isInserted* $\leftarrow$ TRUE // s' is inserted
12: $\Xi(s') \leftarrow p$ // Store the index of the front in $\mathcal{F}$ where s' has been inserted
13: **if** $p <$ *hfi* **then**
14: *hfi* $\leftarrow p$ // Update *hfi*
15: **return** *hfi* // Insertion of s' is completed
16: **if** *isInserted* = FALSE **then** // s' is not yet inserted
17: $F_{P+1} \leftarrow F_{P+1} \cup \{s'\}$ // Insert s' in F_{P+1}
18: $\Xi(s') \leftarrow P + 1$ // Store the index of the front in $\mathcal{F}$ where s' has been inserted
19: **return** *hfi* // Insertion of s' is completed

compared with all the solutions in a front $F_p(\alpha \leq p \leq P)$; instead, it is only compared with those solutions in F_p whose local ranks are assigned corresponding to the m^{th} objective. This means that s' is compared with only $\mathcal{R}^p_m$. A solution $s \in \mathcal{R}^p_m$ whose position for the m^{th} objective is greater than that of s', cannot dominate s', *i.e.*, if $\text{Pos}_s(m) > \text{Pos}_{s'}(m)$ then s cannot dominate s'. This is because s' appears first in the sorted list Q_m, hence cannot be dominated by the solution which appears later in that sorted list. Hence, the actual comparison of s' occurs only with those solutions in $\mathcal{R}^p_m$ whose positions for the m^{th} objective are less than the position of s' for the m^{th} objective.

When s' is compared with the solutions of $\mathcal{R}_m^p$ and s' is found to be non-dominated with respect to all the solutions, then s' is inserted into front F_p and *hfi* is updated accordingly. If s' is dominated by any of the solutions of $\mathcal{R}_m^p$, then s' is compared with the solutions of the next front, *i.e.*, F_{p+1}. If s' is dominated by all the fronts, then s' is inserted into F_{P+1}. Here, a solution is not compared with all the solutions of a front, thereby, reducing the number of dominance comparisons.

Illustration of the* DominationCheck() *procedure. This procedure compares two solutions $s \in \mathcal{F}$ and $s' \in \mathcal{F}'$ in an efficient manner without considering all the objective values. This procedure is different from the normal procedure to compare two solutions as in this procedure all the objective values of the solutions are not compared. The procedure to obtain the dominance relation between two solutions s and s' is summarized in Algorithm 7. This procedure also considers the position of s' in the *sorted matrix* corresponding to the m^{th} objective, denoted as *pos*. In this procedure, s' is not compared with s with respect to all the objectives; instead, only those objectives whose corresponding position is greater than *pos* are compared. In this dominance comparison procedure, when two solutions are compared, not all the objective values are compared. Thus, the dominance comparison between two solutions is efficient.

Algorithm 7 DominationCheck(s', s, pos)

Input: Two solutions s and s', *pos*: Position of s' in the *sorted matrix* corresponding to the m^{th} objective
Output: False: If s' is non-dominated with s
True: If s' is dominated by s

```
1: for j ← M down to 1 do
2:   if Obj-Pos_s(j).pos > pos then
3:     m ← Obj-Pos_s(j).obj
4:     if f_m(s') < f_m(s) then
5:       return False                          // s' is non-dominated with s
6:   else
7:     return True                             // s' is dominated by s
```

3.2. Binary Search Based Insertion

This procedure inserts a solution $s' \in F'$ into $\mathcal{F}$ using a binary search based technique. Unlike the sequential search based technique, where a solution s' can be compared with the solutions of all the fronts, in this technique, solution s' is only compared with the solutions of $\lceil \log(P - \alpha + 2) \rceil$ fronts. For this purpose, a tree structure is followed. Here, the tree is not explicitly created, and instead, the sorted fronts are visualized as the tree.

Two variables *min* and *max* are used to follow the tree structure for binary search. Initially, *min* is set to α and *max* is set to P. At first, s' is compared with

Algorithm 8 INSERT-BS($\mathcal{F}, s', \mathcal{R}, \alpha, hfi, \Xi[\ ]$)

Input: Same as Algorithm 6
Output: Same as Algorithm 6

1: $min \leftarrow \alpha$, $max \leftarrow P$, $mid \leftarrow \lfloor \frac{min+max}{2} \rfloor$
2: $m \leftarrow first_{s'}$ // Objective list where s' occurs first
3: **while** TRUE **do** // Position of s' is not identified
4: $isDominated \leftarrow$ FALSE
5: **for** each solution $s \in \mathcal{R}_m^{mid}$ **do**
6: **if** $\text{Pos}_s(m) < \text{Pos}_{s'}(m)$ **and** DOMINATIONCHECK$(s', s, \text{Pos}_{s'}(m)) =$ TRUE **then**
7: $isDominated \leftarrow$ TRUE
8: **BREAK** // Check for other front in $\mathcal{F}$
9: **if** $isDominated =$ FALSE **then**
10: **if** $mid = min$ **then** // Leaf is explored
11: $F_{mid} \leftarrow F_{mid} \cup \{s'\}$ // Insert s' in F_{mid}
12: $\Xi(s') \leftarrow mid$ // Store the index of the front in $\mathcal{F}$ where s' has been inserted
13: **if** $mid < hfi$ **then**
14: $hfi \leftarrow mid$ // Update *hfi*
15: **return** *hfi* // Insertion of s' is completed
16: **else**
17: $max \leftarrow mid - 1$, $mid \leftarrow \lfloor \frac{min+max}{2} \rfloor$ // Explore left sub-tree
18: **else**
19: **if** $min = P$ **then** // Rightmost leaf is explored
20: $F_{P+1} \leftarrow F_{P+1} \cup \{s'\}$ Insert s' in F_{P+1}
21: $\Xi(s') \leftarrow P + 1$ // Store the index of the front in $\mathcal{F}$ where s' has been inserted
22: **return** *hfi* // Insertion of s' is completed
23: **else if** $mid = max$ **then**
24: $F_{max+1} \leftarrow F_{max+1} \cup \{s'\}$ // Insert s' in F_{max}
25: $\Xi(s') \leftarrow max + 1$ // Store the index of the front in $\mathcal{F}$ where s' has been inserted
26: **if** $max + 1 < hfi$ **then**
27: $hfi \leftarrow max + 1$ // Update *hfi*
28: **return** *hfi* // Insertion of s' is completed
29: **else**
30: $min \leftarrow mid + 1$, $mid \leftarrow \lfloor \frac{min+max}{2} \rfloor$ // Explore right sub-tree

F_{mid} where $mid = \lfloor \frac{min+max}{2} \rfloor$. Let $m = first_{s'}$. Here, solution s' is only compared with those solutions in F_{mid} whose local ranks are assigned corresponding to the m^{th} objective, *i.e.*, s' is compared with $\mathcal{R}_m^{\text{mid}}$. The actual comparison of s' occurs only with those solutions in $\mathcal{R}_m^{\text{mid}}$ whose positions for the m^{th} objective are less than the position of s' for the m^{th} objective. If s' is non-dominated

with respect to all the solutions of F_{mid} with which it is compared, then there are two possibilities:

- If a leaf of the tree is reached ($mid = min$), then s' is inserted in F_{mid} and *hfi* is updated accordingly. After this, the insertion process is completed.
- Otherwise, the root of the left sub-tree is checked.

If s' is dominated by any solution of F_{mid} with which it is compared, then there are three possibilities:

- If the rightmost node of the tree is reached (*i.e.*, $min = P$), then s' is dominated by the solutions of all the front and s' is inserted in F_{P+1}. After this, the process of insertion completes.
- If s' is dominated by the solution of the leaf node, (*i.e.*, $mid = max$), then s' is inserted in $F_{\text{max}+1}$ and *hfi* is updated. After this, the process of insertion completes.
- Otherwise, the root of the right sub-tree is checked.

Like sequential search based insertion, here also a solution is not compared with all the solutions, thereby reducing the number of dominance comparisons.

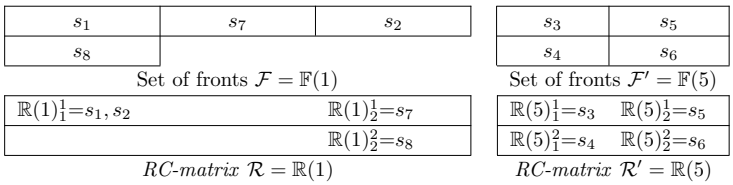
Based on the sequential and binary search based insertion, there are two approaches based on the DCNSRC framework

(i) DCNSRC-SS (DCNSRC approach with sequential search)
(ii) DCNSRC-BS (DCNSRC approach with binary search)

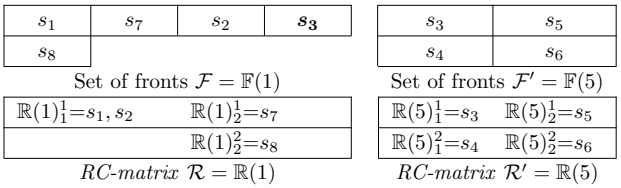
Example 6. *Consider two sets of fronts* $\mathcal{F} = \{F_1 = \{s_1, s_7, s_2\}, F_2 = \{s_8\}\}$ *and* $\mathcal{F}' = \{F'_1 = \{s_3, s_5\}, F'_2 = \{s_4, s_6\}\}$ *which are merged at the last level in Figure 6. In Figure 6, the set of fronts at index positions* 1 *and* 5 *are merged at the last level. Thus, we have merged* $\mathbb{F}(1)$ *and* $\mathbb{F}(5)$.

Figure 7 shows the working of the merge procedure to merge two sets of fronts $\mathcal{F}$ *and* $\mathcal{F}'$ *using a sequential search based strategy to insert a solution from* $\mathcal{F}'$ *in* $\mathcal{F}$. *Here, the* INSERT() *procedure is called twice because there are two fronts in* $\mathcal{F}'$. *In the first* INSERT() *procedure, the* INSERT-SS() *procedure is called twice because the first front in* $\mathcal{F}'$ *has two solutions. Similarly, in the second* INSERT() *procedure, the* INSERT-SS() *procedure is also called twice as there are also two solutions in the second front of* $\mathcal{F}'$. *The RC-matrices corresponding to both sets of fronts are also shown in Figure 7.*

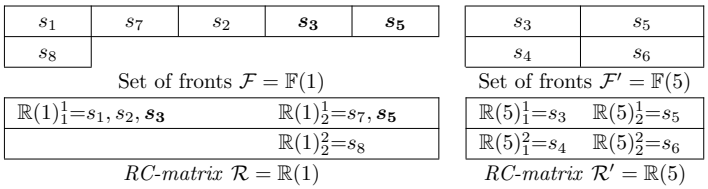
When the first solution of F'_1, *i.e.,* s_3 *is inserted in* $\mathcal{F}$, *then* s_3 *is not compared with respect to all the solutions of* F_1. *It is compared with those solutions of* F_1 *which have obtained their local ranks based on the first objective because* $s_3^f = 1$, *i.e.,* s_3 *is compared with* $\mathbb{R}(1)_1^1$. *Solution* s_3 *is non-dominated with respect to both solutions* $\{s_1, s_2\}$ *of* $\mathbb{R}(1)_1^1$, *so* s_3 *is added to* F_1. *The updated set of fronts is shown in Figure 7(b). When the second solution of* F'_1, *i.e.,* s_5 *is inserted in* $\mathcal{F}$, *then* s_5 *is compared with those solutions of* F_1 *which have obtained their local ranks based on the second objective because* $s_5^f = 2$, *i.e.,* s_5 *is compared with* $\mathbb{R}(1)_2^1$. *Solution* s_5 *is non-dominated with respect to the solution of* $\mathbb{R}(1)_2^1$, *so* s_5 *is added to* F_1. *The updated set of fronts is shown in Figure 7(c). Now*



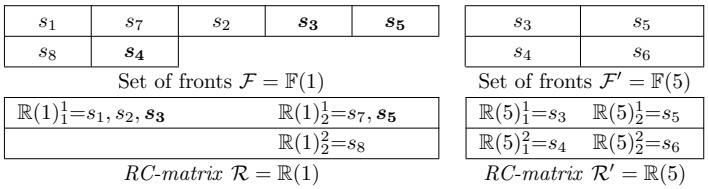
(a) Two sets of fronts along with their *RC-matrices*



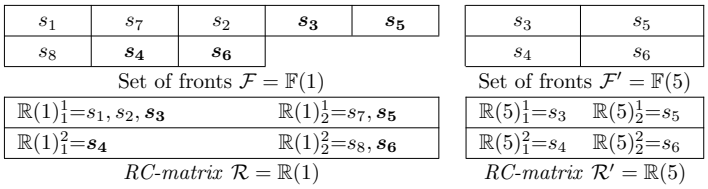
(b) Insertion of s_3 into $\mathcal{F}$



(c) Insertion of s_5 into $\mathcal{F}$



(d) Insertion of s_4 into $\mathcal{F}$



(e) Insertion of s_6 into $\mathcal{F}$

Figure 7: Working of the MERGE() procedure to merge two sets of fronts $\mathcal{F}$ and $\mathcal{F}'$. The solutions of each front in $\mathcal{F}'$ are inserted one by one in $\mathcal{F}$. The solutions which are added to $\mathcal{F}$ and to the *RC-matrix* are shown in **BOLDFACE** in $\mathcal{F}$ and $\mathcal{R}$ respectively.

both solutions of F_1' have been inserted in $\mathcal{F}$, so the RC-matrix corresponding to $\mathcal{F}$ will be updated. The updated RC-matrix corresponding to $\mathcal{F}$ after insertion of both solutions of F_1' is shown in Figure 7(c).

Both solutions of F_1' have been inserted in F_1, so the INSERT() *procedure returns* 1 *as the value of hfi. Thus, the solutions of the next front of $\mathcal{F}$, i.e., the solutions of F_2' will start comparing with the solutions of the second front in $\mathcal{F}$, i.e., with the solutions of F_2.*

When the first solution of F_2', i.e., s_4 is inserted in $\mathcal{F}$, then s_4 is not compared with all the solutions of F_2. It is compared with those solutions of F_2 which have obtained their local ranks based on the first objective because $s_4^f = 1$, i.e., s_4 is compared with $\mathbb{R}(1)_1^2$. As there is no solution in $\mathbb{R}(1)_1^2$, so s_4 is added to F_2'. The updated set of fronts is shown in Figure 7(d). When the second solution of F_2', i.e., s_6 is inserted in $\mathcal{F}$, then s_6 is compared with those solutions of F_2 which have obtained their local ranks based on the second objective because $s_6^f = 2$, i.e., s_6 is compared with $\mathbb{R}(1)_2^2$. Solution s_6 is non-dominated with respect to the solution s_8 of $\mathbb{R}(1)_2^2$, so s_6 is added to F_2. The updated set of fronts is shown in Figure 7(e). Now both solutions of F_2' have been inserted in $\mathcal{F}$, so the RC-matrix corresponding to $\mathcal{F}$ will be updated. The updated RC-matrix corresponding to $\mathcal{F}$ after insertion of both solutions of F_2' is shown in Figure 7(e). □

4. Complexity Analysis

In this section, the complexity analysis of the proposed framework is performed. When the solutions are sorted based on each objective individually, then heap sort is used which requires $\mathcal{O}(1)$ extra space. The sorted solutions based on each objective need to be stored in a separate list. Thus, the overall space complexity of the first phase is $\mathcal{O}(MN)$. The initialization of N matrices of different sizes takes $\mathcal{O}(MN \log N)$ space (from Section 2.2.2).

For each solution $s \in \mathbb{P}$, a variable $sameAs_s$ is considered and there are N solutions, so the space required for storing $sameAs_s$ is $\mathcal{O}(N)$. For each solution s, the position of s corresponding to each objective is stored in Pos_s which requires $\mathcal{O}(M)$ space. There are N solutions so the space required to store this information for all the solutions is $\mathcal{O}(MN)$. For each solution s, the objective list in which s is found is stored in ObjList_s. A solution s can be found in a maximum of M objective lists, so the space required to store ObjList_s for each solution is $\mathcal{O}(M)$. Thus, the space required to store this information for all the solutions is $\mathcal{O}(MN)$. For each solution s, a variable $first_s$ is stored which requires $\mathcal{O}(N)$ space. For each solution s, we consider obj-Pos_s which requires $\mathcal{O}(M)$ space. Thus, the space required to store this information for all the solutions is $\mathcal{O}(MN)$. In the INSERT() procedure, an array of size $|F'|$ is considered which requires a maximum $\mathcal{O}(N)$ space. Thus, the overall space complexity of the proposed framework is $\mathcal{O}(MN \log N)$.

The overall time complexity of the proposed framework is $T = T_1 + T_2 + T_3$ where T_1 is the time complexity of sorting the solutions based on each objective, checking the duplicate solutions and obtaining the order of solutions for the

actual sorting. T_2 is the time complexity of performing the objective value comparisons in the merge operation to insert the solutions of $\mathcal{F}'$ in $\mathcal{F}$ and T_3 is the time complexity to update *RC-matrices*.

The time complexity of sorting the solutions based on all the M objectives is $\mathcal{O}(MN \log N) + (M-1)\mathcal{O}(N \log N) = \mathcal{O}(MN \log N)$ [18]. The number of dominance comparisons required to check for duplicate solutions is $N-1$ as the i^{th} solution is compared with 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. Thus, the worst case time complexity of checking for duplicate solutions is $\mathcal{O}(MN)$. The *sorted matrix* is traversed to obtain the order of the solutions for the actual sorting and finding different values corresponding to each solution (for a solution s; $first_s$, Pos_s, ObjList_s and Obj-Pos_s are obtained). The time complexity of obtaining these values is $\mathcal{O}(MN)$. Thus, $T_1 = \mathcal{O}(MN \log N) + \mathcal{O}(MN) + \mathcal{O}(MN) = \mathcal{O}(MN \log N)$.

The merge operations are performed in a level by level manner. Let the merge operation be performed between two sets of fronts $\mathcal{F}$ and $\mathcal{F}'$. Let us consider A_l be the number of merge operations at each level and B_l be the time complexity when a solution of $\mathcal{F}'$ is inserted in $\mathcal{F}$. C_l is the number of solutions in $\mathcal{F}'$ which take part in the objective value comparison (some of the solutions from $\mathcal{F}'$ are added directly to $\mathcal{F}$ without dominance comparisons). In general, the time complexity of the merge operation is given by Eq. (1). The value of A_l is $N/2^l$ at the l^{th} level. However, B_l and C_l vary depending on the nature of the solutions.

$$T_2 = \sum_{l=1}^{\mathcal{L}} A_l \cdot B_l \cdot C_l \qquad \mathcal{L} = \text{Number of levels} \tag{1}$$

Now, we discuss the time complexity of the DCNSRC framework in different scenarios.

4.1. Solutions are in a Single Front

In this scenario, the time complexity of DCNSRC-SS and DCNSRC-BS is the same because of the existence of a single front. Here, we discuss the time complexity in two different cases. In the first case, the worst case time complexity occurs and in the second case, the best case time complexity occurs.

In the worst case, the first to the $(M-2)^{th}$ objective values of each of the solutions are the same and the last two objective values are such that they are able to declare all the solutions as non-dominated. However, the best case occurs when the traversal of the *sorted matrix* should be such that before the second occurrence of a solution, all the solutions must be traversed at least once.

In the worst case, the order of the solutions in the initial $M-1$ objective lists are the same, and in the last list, it is just reversed. Thus, a solution is first traversed in the first column or in the last column of the *sorted matrix*. Hence, 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*. Thus, two solutions are ranked in each row of the sorted matrix. In the best case, before

the second occurrence of a solution, all the solutions are traversed at least once, so M solutions are ranked in each row of the *sorted matrix*.

Here, $\mathcal{F}$ and $\mathcal{F}'$ both have a single front as all the solutions are non-dominated with respect to each other. Before the merge operation at the l^{th} level, the number of solutions in $\mathcal{F}$ and $\mathcal{F}'$ is 2^{l-1}. At the l^{th} level, for a solution $s' \in \mathcal{F}'$ to be inserted in a front in $\mathcal{F}$, it needs to be compared with $2^{l-(\log 2+1)} = 2^{l-2}$ solutions in $\mathcal{F}$ in the worst case and $2^{l-(\log M+1)}$ solutions in the best case, which are ranked based on the m^{th} objective where $m = \mathit{first}_{s'}$. The time complexity to compare two solutions is $\mathcal{O}(M)$. Thus, the value of B_l is $M \cdot 2^{l-2}$ for the worst case and $B_l = M \cdot 2^{l-(\log M+1)}$ for the best case. All the solutions from $\mathcal{F}'$ are compared with the solutions in $\mathcal{F}$ so $C_l = 2^{l-1}$. Thus, the time complexity of the merge operations in the worst and the best case is obtained by Eqs. (2) and (3), respectively. Thus, the number of dominance comparisons in the worst case is $^1/_4 N(N-2)$ and in the best case is $^1/_{2M} N(N-M)$. In the best case, the number of dominance comparisons decreases with an increase in the number of objectives and it will be zero when $M \geq N$.

$$T_{2_{\text{worst}}} = \sum_{l=1}^{\mathcal{L}} \left(\frac{N}{2^l}\right) \cdot M \cdot 2^{l-2} \cdot 2^{l-1} = \frac{1}{4} MN(N-2) \tag{2}$$

$$T_{2_{\text{best}}} = \sum_{l=\log M+1}^{\mathcal{L}} \left(\frac{N}{2^l}\right) \cdot M \cdot 2^{l-(\log M+1)} \cdot 2^{l-1} = \frac{1}{2} N(N-M) \tag{3}$$

In the worst case, all the solutions are explored for rank assignment in the initial $^N/_2$ rows. The solution which is explored for ranking in the first column, has been explored in the first to the $M-1$ columns and the solution which is explored for ranking in the last column, has been explored only in the last column. So, for a solution s which is ranked based on the first objective, ObjList_s contains the first to the $M-1$ objective and the solution s which is ranked based on the last objective, ObjList_s only contains the last objective. So, the time complexity of updating *RC-matrices* is obtained using Eq. (4). In the best case, a solution is explored based on only one objective and then matrix traversal stops. For a solution s which is ranked based on the m^{th} objective, ObjList_s contains only the m^{th} objective. So, the time complexity of updating *RC-matrices* in the best case is obtained using Eq. (5).

$$\begin{aligned} T_{3_{\text{worst}}} &= \frac{N}{2} \cdot 1 + \sum_{l=2}^{\log N} \left(\frac{N}{2^l}\right) \cdot \left[(M-1)\frac{2^{l-1}}{2} + \frac{2^{l-1}}{2}\right] \\ &= \frac{N}{2} + \frac{1}{4} MN(\log N - 1) = \mathcal{O}(MN \log N) \end{aligned} \tag{4}$$

$$T_{3_{\text{best}}} = \sum_{l=1}^{\log N} \left(\frac{N}{2^l}\right) \cdot 2^{l-1} = \frac{1}{2} N \log N = \mathcal{O}(N \log N) \tag{5}$$

Thus, the overall time complexity in the worst case is $\mathcal{O}(MN \log N) + \mathcal{O}(MN^2) + \mathcal{O}(MN \log N) = \mathcal{O}(MN^2)$. When $M \geq N$, then the time complexity in the best case is $\mathcal{O}(MN \log N) + \mathcal{O}(N \log N) = \mathcal{O}(MN \log N)$.

4.2. Solutions are in Different Fronts

In this scenario, DCNSRC-SS and DCNSRC-BS perform differently because the number of fronts is more than one. As all the solutions are in different fronts, so each front in $\mathcal{F}$ and $\mathcal{F}'$ has a single solution. Before the merge operation at the l^{th} level, the number of fronts (and hence the number of solutions) in both sets of fronts is 2^{l-1}.

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, when a solution $s' \in \mathcal{F}'$ is compared with the solution $s \in \mathcal{F}$ during the merge operation, then this comparison takes constant time. This is because, before the occurrence of s' in the *sorted matrix*, solution s has occurred in all the columns, so in the DominationCheck() procedure, the position of s' corresponding to the first objective is always greater than the position of s for all the objectives.

At the l^{th} level, for a solution from $\mathcal{F}'$ to be inserted in $\mathcal{F}$, it needs to be dominated by the solutions of all the fronts in $\mathcal{F}$. Thus, $B_l = 2^{l-1}$ for DCNSRC-SS and $B_l = \lceil \log\left(2^{l-1}+1\right) \rceil$ for DCNSRC-BS. Only the solution of the first front in $\mathcal{F}'$ is compared with the solutions of $\mathcal{F}$, and the solutions of the remaining fronts of $\mathcal{F}'$ are added directly to $\mathcal{F}$ because of the dominance relationship. So, $C_l = 1$ for both DCNSRC-SS and DCNSRC-BS. Hence, the time complexity of the merge operation of DCNSRC-SS and DCNSRC-BS is given by Eqs. (6) and (7), respectively. The number of dominance comparisons performed by the merge operation in this scenario is 0 as two solutions are compared in constant time.

$$T_{2_{\text{SS}}} = \sum_{l=1}^{\mathcal{L}} \left(\frac{N}{2^l}\right) \cdot 2^{l-1} \cdot 1 = \frac{1}{2} N \log N \tag{6}$$

$$T_{2_{\text{BS}}} = \sum_{l=1}^{\mathcal{L}} \left(\frac{N}{2^l}\right) \cdot \lceil \log\left(2^{l-1}+1\right) \rceil \cdot 1 = 2N - \log N - 2 \tag{7}$$

In this scenario, the order of the solutions in all Q_j after pre-sorting is the same. So, each of the solutions is ranked when it is explored in the first column of a particular row. While traversing the *sorted matrix*, a solution is traversed in each of the columns. So, for a solution s which is ranked based on the first objective, ObjList_s contains all the objectives. So the time complexity of updating *RC-matrices* is obtained using Eq. (8).

$$T_3 = \sum_{l=1}^{\mathcal{L}} \left(\frac{N}{2^l}\right) \cdot M \cdot 2^{l-1} = \frac{1}{2} MN \log N = \mathcal{O}(MN \log N) \tag{8}$$

The working details of the DCNSRC framework in the aforementioned two scenarios are discussed in Appendix B. The number of dominance comparisons performed by various existing non-dominated sorting approaches in three differ-

Table 2: Number of dominance comparisons performed by various 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 [8]	$\frac{N(N-1)}{2}$	$\frac{N(N-1)}{2}$	$\frac{N(N-1)}{2}$
Deductive [14]	$\frac{N(N-1)}{2}$	$\frac{N(N-1)}{2}$	$\frac{1}{2}(N-1)(\sqrt{N}+1)$
ENS-SS [12]	$\frac{N(N-1)}{2}$	$\frac{N(N-1)}{2}$	$N(\sqrt{N}-1)$
ENS-BS [12]	$\frac{N(N-1)}{2}$	$N\log N-(N-1)$	$\frac{N(\sqrt{N}-1)}{2}+N\log\sqrt{N}-\sqrt{N}(\sqrt{N}-1)$
BOS* [18]	$\frac{N(N-1)}{2}^{\dagger}$ $\frac{N(N-M)}{2M}^{\ddagger}$	$\frac{N(N-1)}{2}$	$N(\sqrt{N}-1)^{\dagger}$ $\frac{N(\sqrt{N}-1)}{2}+\frac{N(\sqrt{N}-M)}{2M}^{\ddagger}$
T-ENS [19]	$\frac{N(N-1)}{2}^{\dagger}$ $N\log_M N^{\ddagger}$	$\frac{N(N-1)}{2}$	$N(\sqrt{N}-1)^{\dagger}$ $\frac{N(\sqrt{N}-1)}{2}+N\log_M\sqrt{N}^{\ddagger}$
ENS-NDT [20]	$N-1+\frac{N(N-1)}{2}^{\dagger}$	$N\log N$	$N-1+\frac{N(\sqrt{N}-1)}{2}+\frac{(N+\sqrt{N}-1)\log N}{2}$ $-2N+3\sqrt{N}-1^{\dagger}$
BBOS [23]	$\frac{N(N-1)}{2}^{\dagger}$ $\frac{N(N-M)}{2M}^{\ddagger}$	$N\log N$	$N\log\sqrt{N}+\frac{N(\sqrt{N}-1)}{2}^{\dagger}$ $N\log\sqrt{N}+\frac{N(\sqrt{N}-M)}{2M}^{\ddagger}$
DCNSRC-SS	$N-1+\frac{N(N-2)}{4}^{\dagger}$ $N-1+\frac{N(N-M)}{2M}^{\ddagger}$	$N-1$	$N-1+\frac{N(\sqrt{N}-2)}{4}^{\dagger}$ $N-1+\frac{N(\sqrt{N}-M)}{2M}^{\ddagger}$
DCNSRC-BS	$N-1+\frac{N(N-2)}{4}^{\dagger}$ $N-1+\frac{N(N-M)}{2M}^{\ddagger}$	$N-1$	$N-1+\frac{N(\sqrt{N}-2)}{4}^{\dagger}$ $N-1+\frac{N(\sqrt{N}-M)}{2M}^{\ddagger}$

†Maximum number of dominance comparisons.

‡Minimum number of dominance comparisons.

ent scenarios are given in Table 2[2]. This table clearly reveals the effectiveness of the proposed approaches in terms of the number of dominance comparisons performed.

4.3. Role of the Merge Procedure on the Efficiency

The merge procedure is the procedure where the solutions are actually compared to determine their dominance nature. The time complexity of the pro-

[2]The calculation of the number of dominance comparisons performed in the third scenario is described in Appendix A.

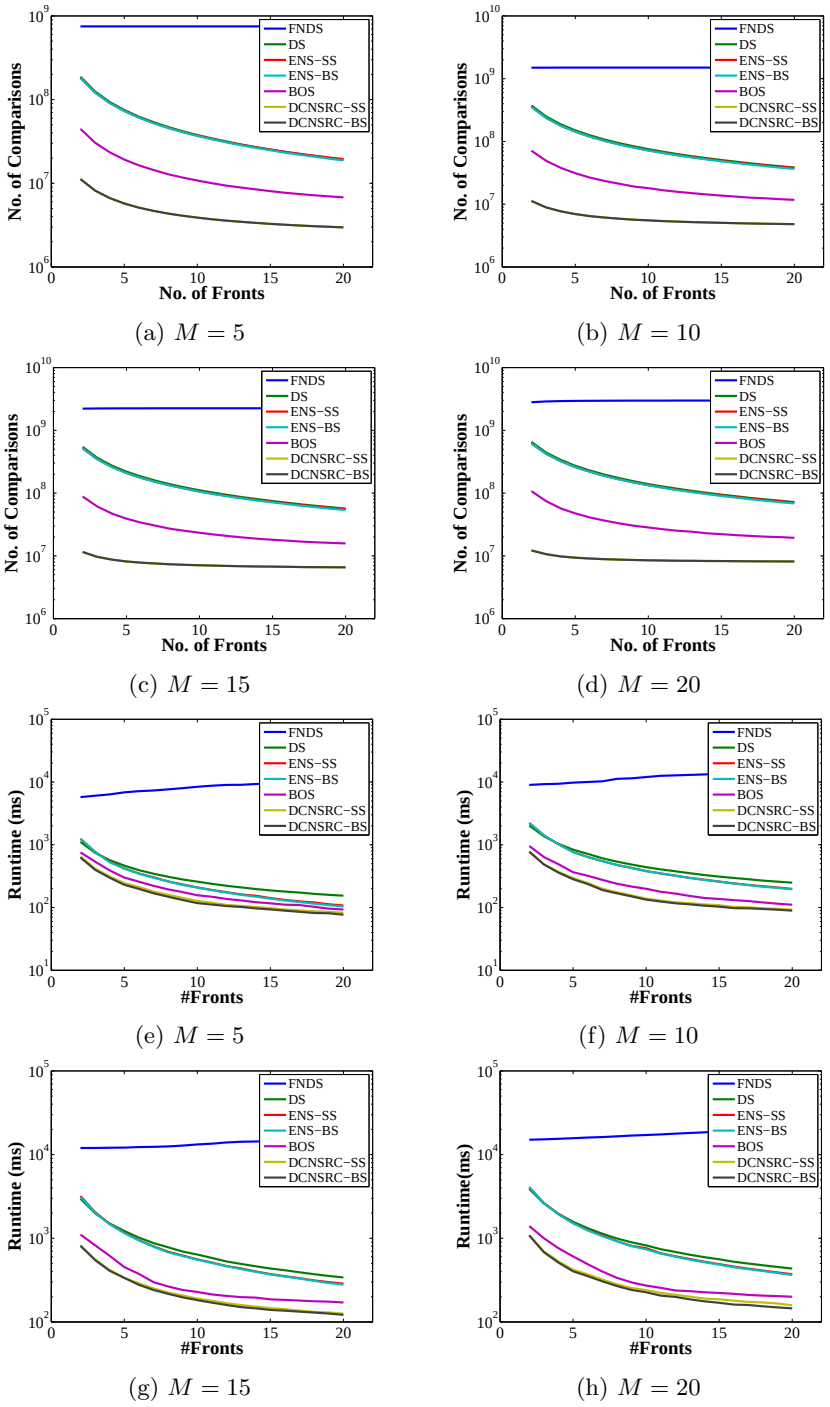
posed framework without considering the merge procedure is $\mathcal{O}(MN \log N)$. The time complexity of the merge procedure varies depending on the nature of the solutions. So, to make the whole framework efficient, we need to make the merge procedure efficient. To make the merge procedure efficient, we have considered some dominance relationships as discussed in [16, 17]. Note that non-consideration of these dominance relationships does not affect the correctness of the merge procedure. However, the number of dominance comparisons in the merge procedure may increase and thus the overall time complexity may increase.

Consider a scenario where all the solutions are in different fronts, *i.e.*, N solutions are in N different fronts. In this particular scenario without considering the dominance relationships, the solutions are compared $\mathcal{O}(N^2)$ times (considering DCNSRC-SS) with each other to find their dominance nature. However, when the dominance relationships as mentioned in [16, 17] are considered, then the solutions are considered only $\mathcal{O}(N \log N)$ times (considering DCNSRC-SS) with each other to find their dominance nature.

5. Experimental Evaluation

We have compared the performance of DCNSRC-SS and DCNSRC-BS with respect to the fast non-dominated sorting approach (FNDS) [8], deductive sort (DS) [14], ENS-SS [12], ENS-BS [12], BOS [18] and DDA-NS [21]. In the DDA-NS approach, two types of comparisons (objective value comparisons and integer value comparisons) occur. However, for the other approaches (FNDS, DS, ENS-SS, ENS-BS and BOS), objective value comparisons are required so, we have computed the objective value comparisons for these approaches. So, in this section, we have shown only the runtime for the DDA-NS [21] not the objective value comparisons as DDA-NS requires objective value comparisons, integer comparisons and integer additions. We are considering the updated version of BOS which can handle duplicate solutions, too. The algorithms were implemented in Java under Windows 7 running in a PC with a 3.30 GHz Intel core i5 processor and 4 GB of RAM.

Fixed Front Dataset. A population of size 10000 is considered where the number of objectives is varied in the range: 5, 10, 15 and 20 [18]. The number of fronts varies from 2 to 20 with an increment of 1 as in [18]. The number of comparisons and execution-times (in milliseconds) for this setup, required by different non-dominated sorting approaches, are shown in Figure 8. The number of comparisons required by DCNSRC-SS and DCNSRC-BS is the same for the fixed front dataset because the solutions of different fronts are compared in constant time. From this figure it is clear that the number of comparisons performed by DCNSRC-SS and DCNSRC-BS are much lower than those performed by the other approaches. As the number of fronts increases, the number of comparisons performed by different approaches, except for FNDS, decreases. The running times are also lower as compared to those of the other approaches. However, the running times are not much lower as compared to the number



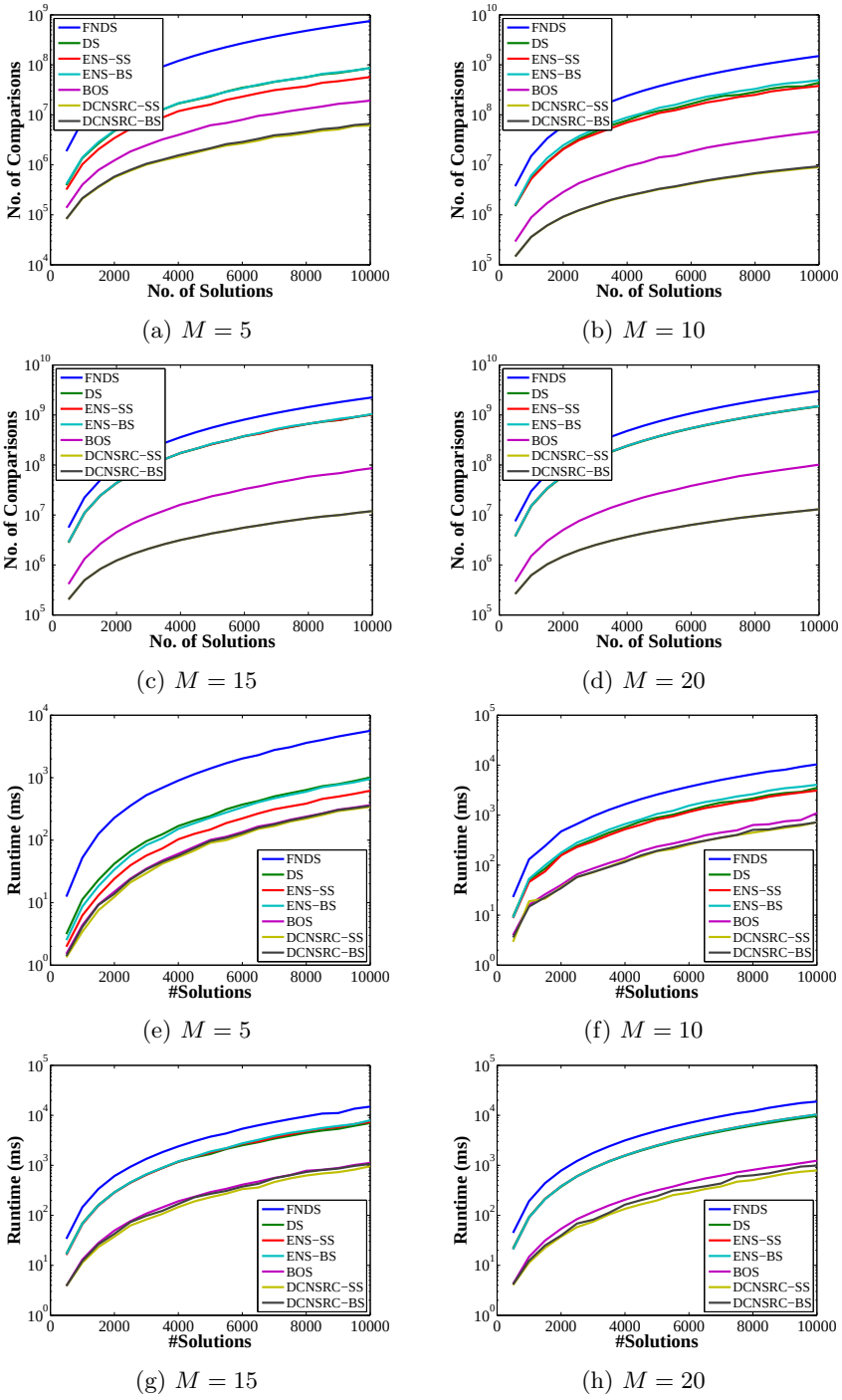
(a) $M = 5$ (b) $M = 10$

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

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

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

Figure 8: Performance of the non-dominated sorting approaches for the fixed front dataset in terms of the number of comparisons and execution time.



(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 9: Performance of the non-dominated sorting approaches for the cloud dataset in terms of the number of comparisons and execution time.

of comparisons. This is because in DCNSRC-SS and DCNSRC-BS, solutions which are ranked are also inserted into $\mathcal{R}$; this process contributes to increasing the execution time. As discussed in [21], the DDA-NS approach uses matrices and vectors heavily. Thus, the DDA-NS approach is especially suitable to be implemented in MATLAB, which is widely believed to be very fast for matrix operations. If the experiments are implemented on different programming platforms, the results of the comparative experiments will be different. We have implemented all the algorithms in Java, so the results of our comparative experiments are different than those reported in [21].

Cloud Dataset. In this dataset, the solutions are randomly generated where the objective values vary between 0 and 1. Thus, the number of non-dominated fronts and the number of solutions in different non-dominated fronts are random. This dataset is considered as it mimics the initial stages of MOEAs. The population size varies from 500 to 10000 with an increment of 500 where the numbers of objectives are 5, 10, 15 and 20 [18]. Thus, a total of 20 different populations are considered. The population size and the number of objectives are the same as in [18]. The number of comparisons of different non-dominated sorting algorithms for this setup is shown in Figures 9(a) – 9(d) and the execution time (in milliseconds) is shown in Figures 9(e) – 9(h). The number of comparisons performed by DCNSRC-SS and DCNSRC-BS is significantly lower as compared to that of the other approaches. The running times are also less as compared to the other approaches. However, the running times are not much lower as compared to the number of comparisons. This is because, in DCNSRC-SS and DCNSRC-BS, solutions which are ranked are also inserted into *RC-matrices*; this process contributes to increase the execution time.

Embedding in NSGA-II. Different non-dominated sorting approaches are incorporated in NSGA-II and then applied for solving test problems DTLZ1, DTLZ2, DTLZ3 and DTLZ4 with the number of objectives: 5, 10, 15 and 20 to assess the performance of the sorting approaches. The other parameters of the algorithm are as follows: population size = 800, number of generations = 200, crossover probability = 0.9, mutation probability = $1/n$ where n is the number of decision variables, crossover distribution index [37] $\eta_c = 20$ and mutation distribution index $\eta_m = 20$. Table 3 shows the number of comparisons and execution times (in milliseconds) of different non-dominated sorting approaches for this setup. This table clearly states that the number of comparisons and execution times required by the proposed approaches are less than those required by the other five approaches.

6. Conclusions and Future Work

In this paper, a framework for non-dominated sorting, namely DCNSRC, has been proposed. Based on this framework, two approaches have been proposed. The main advantage of the proposed framework is that with an increase in the number of objectives, there is a decrease in the number of dominance

Table 3: Performance of the non-dominated sorting approaches in terms of the number of comparisons (#cmp) and execution times (in milliseconds) when those are incorporated in NSGA-II for solving DTLZ1, DTLZ2, DTLZ3 and DTLZ4 test problems. The best values are shown in **boldface**.

Test Problem	**Obj.**	**FNDS**		**DS**		**ENS-SS**		**ENS-BS**		**BOS**		**DCNSRC-SS**		**DCNSRC-BS**		**DDA-NS**
		#cmp	runtime	#cmp	runtime	#cmp	runtime	#cmp	runtime	#cmp	runtime	#cmp	runtime	#cmp	runtime	runtime
DTLZ1	5	3.68E+9	1.16E+4	1.71E+9	5.44E+3	1.56E+9	4.20E+3	1.69E+9	4.91E+3	4.45E+8	1.95E+3	**5.38E+7**	**1.48E+3**	5.42E+7	1.69E+3	1.39E+4
	10	7.62E+9	2.39E+4	3.72E+9	1.14E+4	3.90E+9	1.10E+4	4.00E+9	1.12E+4	6.80E+8	3.34E+3	**1.05E+8**	**2.61E+3**	1.06E+8	2.79E+3	1.79E+4
	15	1.14E+10	3.33E+4	5.68E+9	1.63E+4	6.03E+9	1.37E+4	6.09E+9	1.40E+4	1.04E+9	4.68E+3	**1.56E+8**	**3.24E+3**	**1.56E+8**	3.64E+3	2.49E+4
	20	1.53E+10	4.21E+4	7.70E+9	2.18E+4	8.31E+9	1.74E+4	8.32E+9	1.80E+4	1.20E+9	4.96E+3	**2.04E+8**	**4.66E+3**	**2.04E+8**	4.61E+3	2.81E+4
DTLZ2	5	3.83E+9	1.36E+4	1.85E+9	6.55E+3	1.89E+9	5.67E+3	1.92E+9	6.07E+3	4.74E+8	2.51E+3	**6.72E+7**	**2.16E+3**	6.77E+7	2.33E+3	1.45E+4
	10	7.66E+9	2.56E+4	3.78E+9	1.26E+4	4.05E+9	1.29E+4	4.06E+9	1.20E+4	7.33E+8	3.81E+3	**1.14E+8**	**2.91E+3**	**1.14E+8**	3.17E+3	1.80E+4
	15	1.15E+10	3.48E+4	5.75E+9	1.93E+4	6.21E+9	1.55E+4	6.22E+9	1.56E+4	1.07E+9	5.26E+3	**1.63E+8**	**3.82E+3**	**1.63E+8**	4.69E+3	2.39E+4
	20	1.53E+10	4.41E+4	7.74E+9	2.30E+4	8.44E+9	1.94E+4	8.45E+9	1.98E+4	1.19E+9	5.02E+3	**2.08E+8**	**4.37E+3**	**2.08E+8**	4.55E+3	2.98E+4
DTLZ3	5	3.66E+9	1.04E+4	1.77E+9	4.88E+3	1.29E+9	3.38E+3	1.55E+9	5.04E+3	3.99E+8	1.95E+3	**4.50E+7**	**1.12E+3**	4.52E+7	1.17E+3	1.41E+4
	10	7.66E+9	2.33E+4	3.79E+9	1.17E+4	4.00E+9	1.17E+4	4.03E+9	1.13E+4	7.03E+8	3.59E+3	**1.10E+8**	**2.69E+3**	**1.10E+8**	2.83E+3	1.77E+4
	15	1.15E+10	3.40E+4	5.74E+9	1.90E+4	6.17E+9	1.48E+4	6.18E+9	1.51E+4	9.75E+8	4.56E+3	**1.58E+8**	**3.49E+3**	**1.58E+8**	3.87E+3	2.48E+4
	20	1.53E+10	4.40E+4	7.74E+9	2.22E+4	8.42E+9	1.86E+4	8.43E+9	1.87E+4	1.17E+9	5.01E+3	**2.06E+8**	**4.21E+3**	**2.06E+8**	4.43E+3	2.84E+4
DTLZ4	5	3.82E+9	1.29E+4	1.84E+9	6.36E+3	1.70E+9	5.05E+3	1.79E+9	5.61E+3	4.06E+8	1.91E+3	**5.84E+7**	**1.74E+3**	5.88E+7	1.85E+3	1.36E+4
	10	7.66E+9	2.63E+4	3.76E+9	1.29E+4	3.76E+9	1.27E+4	3.77E+9	1.26E+4	5.14E+8	2.84E+3	**1.06E+8**	**2.72E+3**	1.07E+8	2.92E+3	1.77E+4
	15	1.14E+10	4.18E+4	5.66E+9	2.12E+4	5.70E+9	1.70E+4	5.70E+9	1.80E+4	6.21E+8	**3.31E+3**	**1.54E+8**	3.54E+3	**1.54E+8**	4.21E+3	2.34E+4
	20	1.53E+10	5.49E+4	7.58E+9	2.87E+4	7.64E+9	2.28E+4	7.64E+9	2.35E+4	6.81E+8	**3.60E+3**	**2.00E+8**	4.15E+3	**2.00E+8**	4.41E+3	3.03E+4

comparisons. Duplicate solutions are also handled efficiently in this approach. This framework also does not consider all the objective values while comparing two solutions. The worst case time complexity of the proposed approaches is $\mathcal{O}(MN^2)$ with $\frac{1}{4}N(N-2)$ number of dominance comparisons when all the solutions are in the same front and the objective values of the solutions have specific values. The best case time complexity when all the solutions are in a single front is $\mathcal{O}(MN \log N)$. The best case time complexity of the proposed approach is $\mathcal{O}(MN \log N)$ when all the solutions are in different fronts. Generally, when all the solutions are non-dominated, then the number of dominance comparisons required by various approaches reaches its maximum. However, in our approach, in this scenario, the number of dominance comparisons can be zero also when $M \geq N$.

As part of our future work, we would like to implement our approach in a parallel environment as it is based on a divide-and-conquer strategy. We would also like to obtain the maximum theoretical speedup of the parallel version of the proposed approach. The development of a more efficient approach which can further reduce the number of comparisons can be a potential area for future work. Obtaining the lower bound time complexity in the worst case of non-dominated sorting can be an important future work.

Acknowledgements

Dr. Sriparna Saha would like to acknowledge the support of Early Career Research Award of Science and Engineering Research Board (SERB) of Department of Science and Technology India to carry out this research. The last author gratefully acknowledges support from CONACyT grant no. 2016-01-1920 (*Investigación en Fronteras de la Ciencia 2016*) and from a project from the 2018 SEP-Cinvestav Fund (application no. 4).

References

[1] S. Ramesh, S. Kannan, S. Baskar, Application of Modified NSGA-II Algorithm to Multi-Objective Reactive Power Planning, Applied Soft Computing 12 (2) (2012) 741–753.

[2] S. Kannan, S. Baskar, J. D. McCalley, P. Murugan, Application of NSGA-II Algorithm to Generation Expansion Planning, IEEE Transactions on Power systems 24 (1) (2009) 454–461.

[3] P. Murugan, S. Kannan, S. Baskar, Application of NSGA-II Algorithm to Single-Objective Transmission Constrained Generation Expansion Planning, IEEE Transactions on Power Systems 24 (4) (2009) 1790–1797.

[4] H. Soyel, U. Tekguc, H. Demirel, Application of NSGA-II to Feature Selection for Facial Expression Recognition, Computers & Electrical Engineering 37 (6) (2011) 1232–1240.

[5] M. Atiquzzaman, S.-Y. Liong, X. Yu, Alternative Decision Making in Water Distribution Network with NSGA-II, Journal of Water Resources Planning and Management 132 (2) (2006) 122–126.

[6] N. Jozefowiez, F. Semet, E.-G. Talbi, Enhancements of NSGA II and its Application to the Vehicle Routing Problem with Route Balancing, in: International Conference on Artificial Evolution (Evolution Artificielle), Springer, 2005, pp. 131–142.

[7] N. Srinivas, K. Deb, Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms, Evolutionary Computation 2 (3) (1994) 221–248.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] X. Zhang, Y. Tian, R. Cheng, Y. Jin, An Efficient Approach to Nondominated Sorting for Evolutionary Multiobjective Optimization, IEEE Transactions on Evolutionary Computation 19 (2) (2015) 201–213.

[13] 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.

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

[15] M. Buzdalov, A. Shalyto, A Provably Asymptotically Fast Version of the Generalized Jensen Algorithm for Non-dominated Sorting, in: 13th International Conference on Parallel Problem Solving from Nature – PPSN XIII, Springer. Lecture Notes in Computer Science Vol. 8672, Ljubljana, Slovenia, 2014, pp. 528–537.

[16] 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.

[17] S. Mishra, S. Saha, S. Mondal, C. A. C. Coello, A Divide-and-Conquer Based Efficient Non-dominated Sorting Approach, Swarm and Evolutionary Computation 44 (2018) 748–773.

[18] 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 on Genetic and Evolutionary Computation Conference Companion, ACM Press, Denver, Colorado, USA, 2016, pp. 1113–1120, ISBN: 978-1-4503-4323-7.

[19] 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.

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

[21] Y. Zhou, Z. Chen, J. Zhang, Ranking Vectors by Means of the Dominance Degree Matrix, IEEE Transactions on Evolutionary Computation 21 (1) (2017) 34–51.

[22] V. Palakonda, T. Pamulapati, R. Mallipeddi, P. P. Biswas, K. C. Veluvolu, Nondominated Sorting Based on Sum of Objectives, in: 2017 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2017, pp. 1–8.

[23] P. C. Roy, K. Deb, M. M. Islam, An Efficient Nondominated Sorting Algorithm for Large Number of Fronts, IEEE Transactions on Cybernetics (99) (2018) 1–11.

[24] A. Jaszkiewicz, T. Lust, ND-Tree-Based Update: A Fast Algorithm for the Dynamic Nondominance Problem, IEEE Transactions on Evolutionary Computation 22 (5) (2018) 778–791.

[25] S. Mishra, S. Mondal, S. Saha, C. A. C. Coello, GBOS: Generalized Best Order Sort algorithm for Non-dominated Sorting, Swarm and Evolutionary Computation 43 (2018) 244–264.

[26] H.-L. Liu, F. Gu, Q. Zhang, Decomposition of a Multiobjective Optimization Problem Into a Number of Simple Multiobjective Subproblems, IEEE Transactions Evolutionary Computation 18 (3) (2014) 450–455.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] S. Mishra, S. Mondal, S. Saha, Fast Implementation of Steady-State NSGA-II, in: 2016 IEEE Congress on Evolutionary Computation (CEC'2016), IEEE Press, Vancouver, Canada, 2016, pp. 3777–3784, ISBN: 978-1-5090-0623-6.

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

[33] 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.

[34] S. Mishra, C. A. C. Coello, An Approach for Non-domination Level Update Problem in Steady-State Evolutionary Algorithms With Parallelism, in: 2019 IEEE Congress on Evolutionary Computation (CEC'2019), IEEE Press, Wellington, New Zealand, 2019, pp. 1006–1013, ISBN: 978-1-7281-2153-6.

[35] J. Smith, On Replacement Strategies in Steady State Evolutionary Algorithms, Evolutionary Computation 15 (1) (2007) 29–59.

[36] J. Du, Z. Cai, Y. Chen, A Sorting Based Algorithm for Finding Non-dominated Set in Multi-Objective Optimization, in: ICNC 2007. Third International Conference on Natural Computation, 2007., IEEE press, Haikou, China, 2007, pp. 436–440, ISBN: 0-7695-2875-9.

[37] K. Deb, R. B. Agrawal, Simulated Binary Crossover for Continuous Search Space, Complex Systems 9 (1994) 1–34.

Appendix A. Solutions are Equally Divided in $\sqrt{N}$ Fronts

Here, N solutions are equally divided into $\sqrt{N}$ fronts such that each solution in the k^{th} front is dominated by all the solutions in the $(k-1)^{th}$ front. In this scenario, DCNSRC-SS and DCNSRC-BS perform differently because the number of fronts is more than one. Here, we discuss the number of dominance comparisons performed in two different cases. In the first case, the number of dominance comparisons is the maximum, and in the second case, the number of dominance comparison is the minimum.

When the *sorted matrix* is traversed in a row-wise manner, then before the occurrence of a solution in the k^{th} 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. So, when two solutions of different fronts are compared, then this comparison takes constant time. This is because, in the DominationCheck() procedure, the position of a solution of the k^{th} front corresponding to any of the objectives is always greater than the position of all the solutions in the $(k-1)^{th}$ front for all the objectives.

Let us assume that a solution s be first explored in the m^{th} 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 DCNSRC-SS, s is dominated by one of the solutions in each of the previous $k-1$ fronts which have been ranked based on the m^{th} objective. In case of DCNSRC-BS, it is dominated by one of the solutions in only $\lceil \log k \rceil$ previous fronts which have been ranked based on the m^{th} 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. This step does not contribute to the number of dominance comparison because two solutions of different fronts are checked in constant time.
- (b) s is non-dominated with respect to all the previous solutions in the k^{th} front which have been ranked based on the m^{th} objective for DCNSRC-SS and DCNSRC-BS.

Without loss of generality, let us assume $N = 2^{2a}$ and $M = 2^b$ where a, b are positive integers and $a, b \geq 1$.

Appendix A.1. Maximum Number of Dominance Comparisons

The maximum number of dominance comparisons occurs when the 1^{st} to the $(M-2)^{th}$ objective values of all the solutions in a particular front are the same. The last two objective values of all 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 2^{nd} 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. Thus, from each row, only two solutions are ranked (if not already ranked) – one from the first column and another 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 k^{th} 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 $(k-1)^{th}$ front are present. Thus, the number of dominance comparisons performed by DCNSRC-SS and DCNSRC-BS is given by Eq. (A.1). Here, the $N-1$ factor comes from checking for duplicate solutions.

$$\begin{aligned}\text{No. of dominance comparisons} &= N-1+\sqrt{N}\left[\sum\nolimits_{i=1}^{\sqrt{N}/2}(i-1)+(i-1)\right] \\ &= N-1+\frac{1}{4}N(\sqrt{N}-2) \qquad \text{(A.1)}\end{aligned}$$

Appendix A.2. Minimum Number of Dominance Comparisons

The minimum number of dominance comparisons 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. 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 k^{th} front are ranked in the initial $\sqrt{N}/M$ rows after $(k-1)\sqrt{N}$ rows. Thus, the number of dominance comparisons performed by DCNSRC-SS and DCNSRC-BS is given by Eq. (A.2). From this equation, it is clear that as the number of objectives 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 the duplicate solutions.

$$\text{No. of dominance comparisons} = N - 1 + \sqrt{N}\left[\sum_{k=1}^{\sqrt{N}/M} M(k-1)\right]$$
$$= N - 1 + \frac{1}{2M}N(\sqrt{N} - M) \tag{A.2}$$

Appendix B. Examples

In this section, some of the examples showing the behavior of the proposed approach in two different scenarios are discussed.

Appendix B.1. All Solutions are Non-Dominated

Examples of the worst and best case situations when all the solutions are in a single front are discussed.

Appendix B.1.1. Worst Case

Figure B.10 shows the worst case situation for eight solutions when the number of objectives is 4. Working flow of the proposed framework for the solutions given in Figure B.10, is shown in Figure B.14.

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 B.10: Eight solutions in 4-dimensional space which are in a single front (Worst case).

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	4	5	X	1
s_5	2	7	X	X
s_6	7	2	X	X
s_7	5	4	2	X
s_8	6	3	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_8	–	–
s_4	s_7	–	–
s_7	s_4	–	–
s_8	s_3	–	–
s_6	s_5	–	–
s_2	s_1	–	–

(b) Sorted list

Figure B.11: Eight solutions in 4-dimensional space which are in a single front (Best case).

Appendix B.1.2. Best Case

Figure B.11 shows the best case situation for eight solutions when the number of objectives is 4. It is possible to decide that the two solutions are non-dominated or not with the help of two objectives if they are contradictory. So, X in Figure B.11 represents any integer value greater than 2, *i.e.*, $X > 2$. In this figure, the first two objectives are sufficient to declare that all the eight solutions are non-dominated. 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.

For showing the working flow of the proposed framework in this situation, we consider $X = 3$. Eight solutions and the sorted order of the solutions based on all

Sol	Objective			
	O_1	O_2	O_3	O_4
s_1	1	8	3	3
s_2	8	1	3	3
s_3	3	6	1	3
s_4	4	5	3	1
s_5	2	7	3	3
s_6	7	2	3	3
s_7	5	4	2	3
s_8	6	3	3	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_8	s_1	s_1
s_4	s_7	s_5	s_5
s_7	s_4	s_4	s_3
s_8	s_3	s_8	s_7
s_6	s_5	s_6	s_6
s_2	s_1	s_2	s_2

(b) Sorted list

Figure B.12: Eight solutions in 4-dimensional space which are in a single front (Best case).

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 B.13: Eight solutions in 4-dimensional space which are in eight different fronts.

the four objectives are shown in Figure B.12. The working flow of the proposed framework for the solutions given in Figure B.12, is shown in Figure B.15.

Appendix B.2. Solutions are in Different Fronts

Figure B.13 shows the scenario when eight solutions are divided into 8 different fronts.

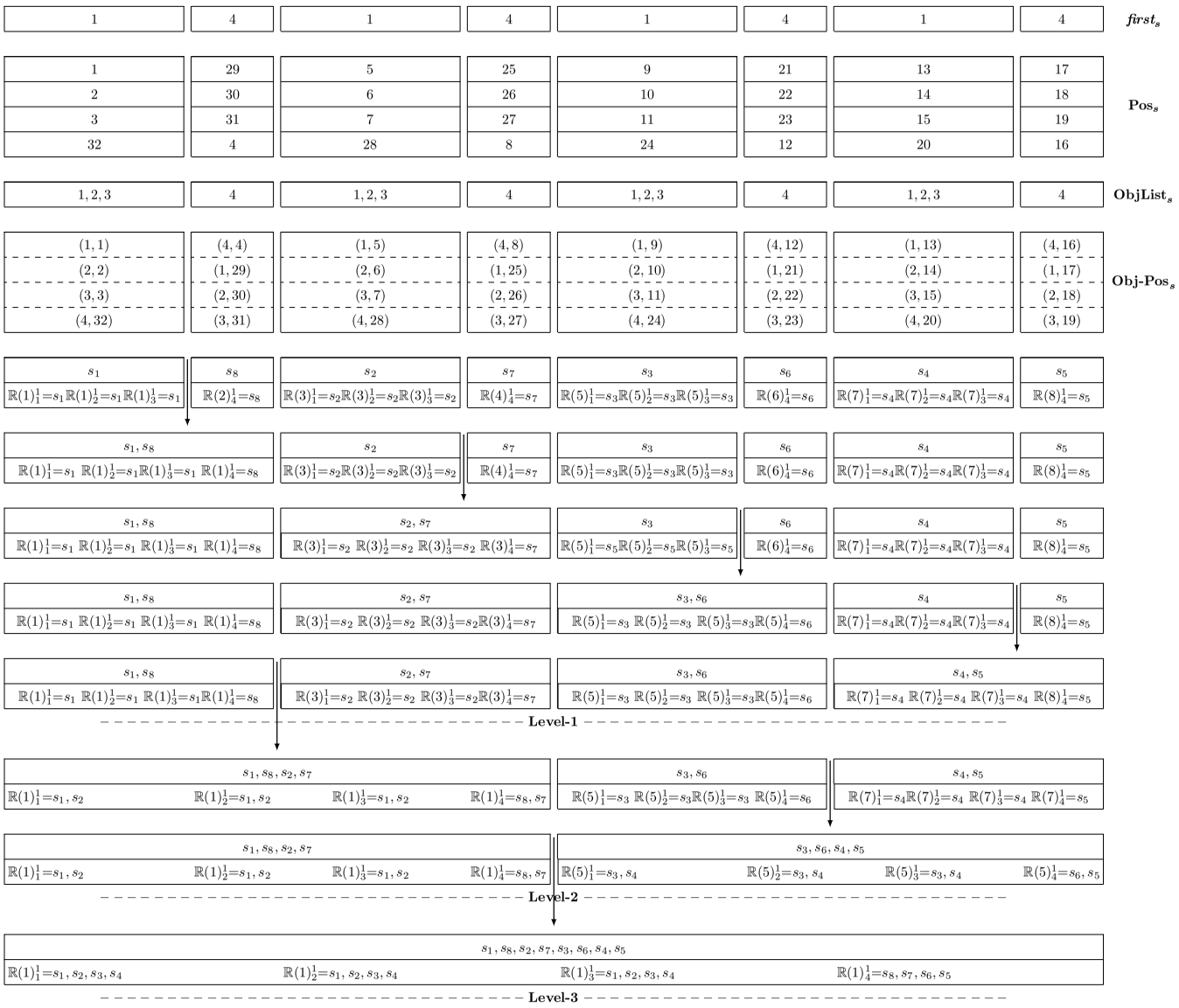


Figure B.14: Working flow of the proposed framework when all the solutions are in a single front in the worst case scenario. ↓ indicates the merge operation between the immediate left and the right set of fronts. $s_i, \ldots, s_j$ represents that these solutions are non-dominated with each other.

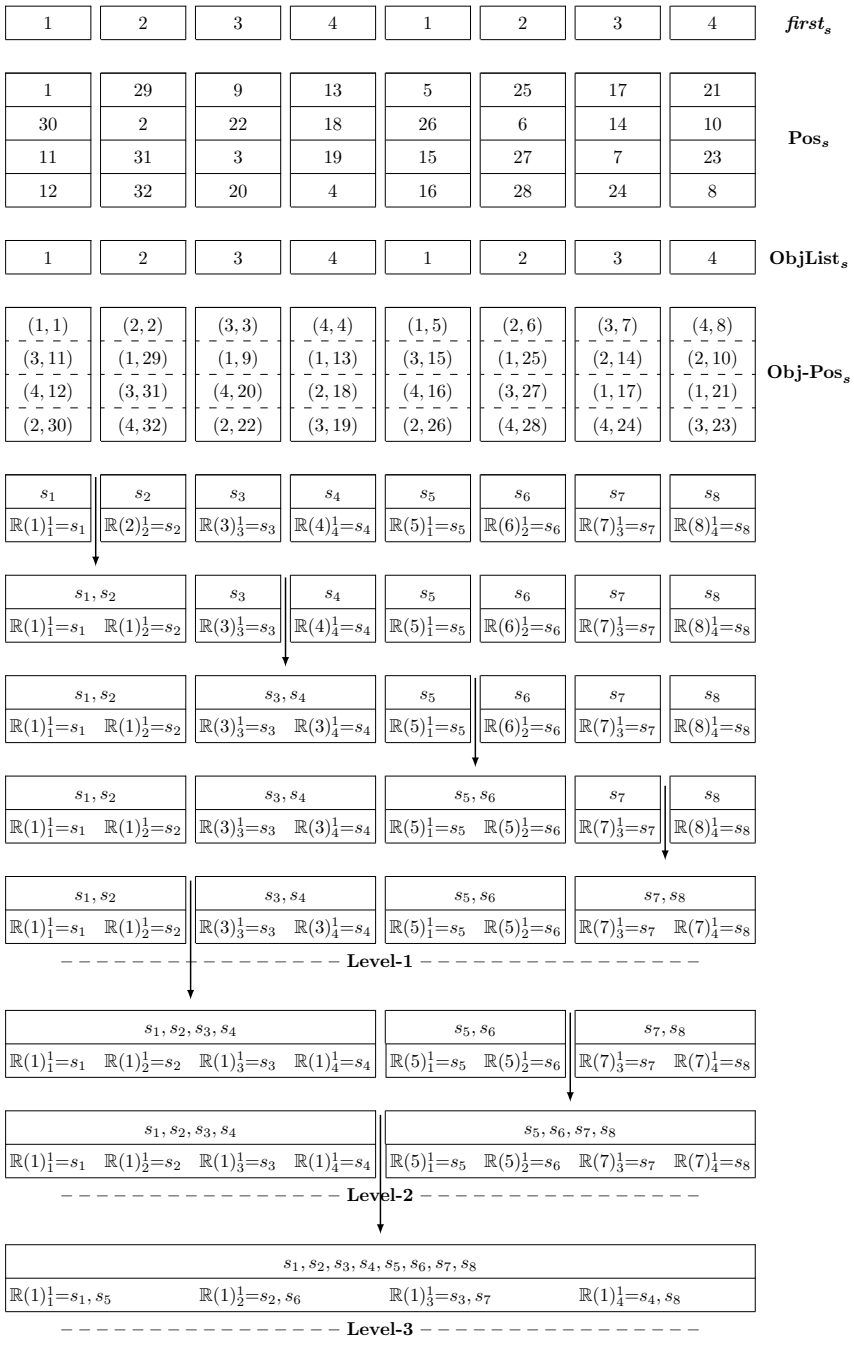


Figure B.15: Working flow of the proposed framework when all the solutions are in a single front in the best case scenario. ↓ indicates the merge operation between the immediate left and the right set of fronts. $s_i, \ldots, s_j$ represents that these solutions are non-dominated with each other.