A Divide-and-Conquer based Efficient Non-dominated Sorting Approach

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

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

Abstract

In general, evolutionary algorithms are very prevalent in solving multi-objective optimization problems. Pareto-based multi-objective evolutionary algorithms are popularly used in solving different multi-objective optimization problems. These algorithms work using several steps, non-dominated sorting being the most salient one. However, this non-dominated sorting step is associated with a high computational complexity. In the past, different approaches have been proposed for non-dominated sorting. In this paper, to address the problem of non-dominated sorting, a framework called DCNS (Divide-and-conquer based non-dominated sorting) is developed. Based on this DCNS framework, four different approaches are proposed. The best case time complexity of our proposed DCNS framework is proved to be $\mathcal{O}(N \log N + MN)$ for $M \geq 2$ where N is the number of solutions and M is the number of objectives. This best case time complexity is better than the best case time complexities of various other approaches. The number of dominance comparisons performed by the proposed framework is lower than those from other state-of-the-art approaches in different scenarios. The proposed framework has the parallelism property and the scope of parallelism is also discussed.

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

1. Introduction

Over the past years, various multi-objective evolutionary algorithms (MOEAs) have been developed for solving multi-objective optimization problems (MOOPs). The algorithmic complexity of Pareto-based MOEAs is still high because of the complexity of non-dominated sorting. The concept of non-dominated sorting is

*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)

also used in various fields of study such as game theory, economics, computational geometry and databases [1]. Let $\mathbb{P} = \{sol_1, sol_2, \ldots, sol_N\}$ be a population of N solutions in M-dimensional objective space where M is the number of objectives associated with each solution. A solution sol in M-dimensional objective space is represented as $sol = \{f_1(sol), f_2(sol), \ldots, f_M(sol)\}$ where $f_m(sol), 1 \leq m \leq M$ is the value of sol for the m^{th} objective. We are considering the minimization problem where all the objectives need to be minimized. In non-dominated sorting, the solutions are sorted based on dominance relationship between the solutions. Formally, the dominance relationship between two solutions is defined as follows.

Definition 1 (Dominance). *A solution sol_i is said to dominate another solution sol_j denoted as $sol_i \prec sol_j$ if the two following conditions are satisfied:*

- $f_m(sol_i) \leq f_m(sol_j), \forall m \in \{1, 2, \ldots, M\}$
- $f_m(sol_i) < f_m(sol_j), \exists m \in \{1, 2, \ldots, M\}$

The notation $sol_i \not\prec sol_j$ represents that sol_i does not dominate sol_j. Two solutions sol_i and sol_j are said to be non-dominated when neither dominates other, i.e., $sol_i \not\prec sol_j$ and $sol_j \not\prec sol_i$.

Now, we formally define non-dominated sorting.

Definition 2 (Non-dominated Sorting). *Non-dominated sorting divides a set of N solutions $\{sol_1, sol_2, \ldots, sol_N\}$ into different fronts $\{F_1, F_2, \ldots, F_K\}$ which are arranged in decreasing order of their dominance such that the two following conditions are satisfied:*

- $\forall sol_i, sol_j \in F_k$: $sol_i \not\prec sol_j$ *and* $sol_j \not\prec sol_i$ $(1 \leq k \leq K)$
- $\forall sol \in F_k$, $\exists sol' \in F_{k-1}$: $sol' \prec sol$ $(2 \leq k \leq K)$.

Front F_1 has the highest dominance, front F_2 has the second highest dominance and so on. The last front F_K has the lowest dominance.

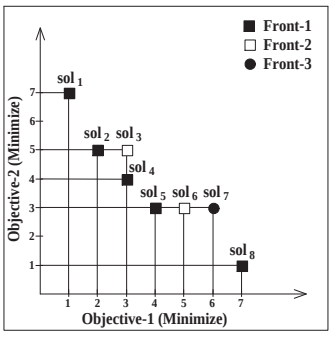


Figure 1: A population with eight solutions in 2-dimensional objective space which is divided into three fronts.

Example 1. *Let* $\mathbb{P} = \{sol_1, sol_2, \ldots, sol_8\}$ *be a population of eight solutions in a* 2*-dimensional objective space as shown in Figure 1. For a minimization problem, these eight solutions are divided into three fronts where* $F_1 = \{sol_1, sol_2, sol_4, sol_5, sol_8\}$*,* $F_2 = \{sol_3, sol_6\}$ *and* $F_3 = \{sol_7\}$*.*

In the current work, we present a divide-and-conquer based non-dominated sorting (DCNS) framework.[1] This work is an extension of our previous work [2]. In the previous work we have presented a divide-and-conquer based non-dominated sorting framework and experimentally validated it. The major changes with respect to this previous work are as follows. Here, we have explicitly presented the algorithm to reduce the number of dominance comparisons without using extra space. The number of dominance comparisons is further reduced with the use of extra space. The best case time complexity is proved to be $\mathcal{O}(N \log N + MN)$. Also, the parallelism is theoretically analyzed in three different scenarios in different ways. The main contributions of the current paper are thus summarized as follows:

- We propose a framework for non-dominated sorting based on a divide-and-conquer strategy which has the parallelism property.
- Four variants of this DCNS framework are introduced varying the search type and space requirements.
- For many cases, the number of dominance comparisons is significantly reduced by our proposed DCNS framework.
- DCNS has the best case time complexity of $\mathcal{O}(N \log N + MN)$ which occurs when all the solutions are in different fronts. This best case time complexity is better than the best case time complexity of most of the existing approaches for $M > 3$.
- According to [3], [4], the processing time of non-dominated sorting is bounded from below by $\mathcal{O}(N \log N)$. Jensen *et al.* [3] showed that this lower bound holds for $M = 2$. In this paper, we show that the processing time of non-dominated sorting is bounded from below by $\mathcal{O}(N \log N)$ for $M \geq 3$ and also when $M = \mathcal{O}(\log N)$. However, the upper bound on the processing time is still $\mathcal{O}(MN^2)$.

The remainder of this paper is organized as follows. Section 2 summarizes some of the previous works on non-dominated sorting. We discuss our proposed DCNS framework in Section 3. The detailed description of the merge procedure which is the basis of our DCNS framework is provided in Section 4. Our proposed DCNS framework is experimentally evaluated in Section 5. Finally, Section 6 summarizes the paper and provides some possible paths for future research.

[1] A preliminary version of this paper was presented at the *2016 IEEE Congress on Evolutionary Computation* [2].

2. Related Work

In this section, we discuss several approaches for non-dominated sorting. Srinivas *et al.* [5] presented a non-dominated sorting procedure which is considered as a naive approach. In the naive approach for non-dominated sorting, each solution is compared with respect to all the other solutions. The solutions which are not dominated by any other solution are assigned to the first front. All the solutions which are assigned to the first front are not considered now. Again, the solutions are compared with each other and the solutions which are not dominated by any other solution are assigned to the second front. This process is repeated until all the solutions are assigned to a front. The worst case time complexity of this approach is $\mathcal{O}(MN^3)$ when all the solutions are in different fronts. However, the best case time complexity is $\mathcal{O}(MN^2)$ when all the solutions are in a single front. The space complexity of this approach is $\mathcal{O}(N)$ [6]. Deb *et al.* [6] proposed a computationally cost effective approach called fast non-dominated sorting (FNDS) with time complexity $\mathcal{O}(MN^2)$ and space complexity $\mathcal{O}(N^2)$. In this approach, each solution is compared with other solutions only once. Jensen *et al.* [3] proposed a non-dominated sorting approach for 2 objectives. This approach first sorts all the solutions based on the first objective. If two solutions share identical values for the first objective, then the value of second objective is considered. After pre-sorting, the solutions are assigned to their respective fronts. The time complexity of this approach is $\mathcal{O}(N \log N)$ and the space complexity is $\mathcal{O}(1)$ [7]. For more than two objectives, Jensen *et al.* [3] proposed a recursive approach based on the divide-and-conquer strategy. The time complexity of this approach is $\mathcal{O}(N \log^{M-1} N)$ and the space complexity is $\mathcal{O}(MN)$. However, this approach is not suitable when two solutions have the same value for a particular objective.

Fang *et al.* [8] proposed an efficient non-dominated sorting method based on a divide-and-conquer strategy. The worst case time complexity of this method is $\mathcal{O}(MN^2)$ when all the solutions are non-dominated. The best case time complexity of this method is $\mathcal{O}(MN \log N)$. The space complexity of this method is $\mathcal{O}(MN)$. However, this algorithm considers one solution as dominated by another if two solutions are duplicates. A fast method for constructing the non-dominated set based on arena's principle is proposed by Tang *et al.* [9]. This approach can achieve a time complexity of $\mathcal{O}(MN\sqrt{N})$ in some cases [7]. Climbing sort and deductive sort were developed by McClymont *et al.* [10]. In general, the performance of deductive sort is better than climbing sort [10]. Deductive sort avoids some unnecessary dominance comparisons by inferring the dominance relationship between the solutions. The worst case time complexity of deductive sort is $\mathcal{O}(MN^2)$ with $\mathcal{O}(N)$ space complexity. The best case time complexity of deductive sort is $\mathcal{O}(MN\sqrt{N})$. Fortin *et al.* [11] removed the assumption of Jensen's algorithm, but removing this assumption increases the worst case time complexity to $\mathcal{O}(MN^2)$. However, the average case time complexity remains $\mathcal{O}(N \log^{M-1} N)$.

Corner sort is proposed by Wang *et al.* [12] with worst case time complexity $\mathcal{O}(MN^2)$. Efficient non-dominated sorting (ENS) was developed by Zhang *et*

al. [7]. ENS first pre-sorts the solutions. After pre-sorting, the solutions are assigned to their respective fronts using two search techniques. Based on the search technique, two variants of ENS – ENS-SS (ENS with sequential search) and ENS-BS (ENS with binary search) are developed. ENS-SS and ENS-BS both require $\mathcal{O}(MN^2)$ time in the worst case. The best case time complexity of ENS-SS and ENS-BS is $\mathcal{O}(MN\sqrt{N})$ and $\mathcal{O}(MN \log N)$, respectively. The time complexity of non-dominated sorting was proved to be $\mathcal{O}(N \log^{M-1} N)$ by Buzdalov *et al.* [13]. Bao *et al.* [14] proposed a Hierarchical Non-dominated Sorting (HNDS) for non-dominated sorting. HNDS first sorts the solution based on the first objective, then the solutions are assigned to their respective front. The best case time complexity of HNDS is $\mathcal{O}(MN\sqrt{N})$ and the worst case time complexity is $\mathcal{O}(MN^2)$ with space complexity $\mathcal{O}(N)$.

In general, for a solution to be inserted in a front, it needs to be compared with respect to all the solutions in that particular front. However, it has been shown that there is no need to do so in all cases. A solution can be inserted in a front by comparing it with some of the solutions in that front. Several approaches based on this idea have been recently proposed [1, 15, 16]. Best order sort (BOS) [1] first sorts the solutions based on each objective individually unlike ENS [7] where the solutions are sorted based on the first objective. BOS is very efficient in terms of the number of dominance comparisons. Another advantage of BOS is that when two solutions are compared in terms of dominance, there is no need to compare all the objectives because of the comparison set concept [1]. The worst case time complexity is $\mathcal{O}(MN^2)$ and the best case time complexity is $\mathcal{O}(MN \log N)$. The space complexity of BOS is $\mathcal{O}(MN)$. However, in its current form, BOS is not suitable when there are duplicate solutions in the population. BOS has been recently generalized to handle duplicate solutions by removing the comparison set concept.[2] If the comparison set concept is removed, then the time to compare two solutions may be increased as all the objectives have to be considered when solutions are compared.

A tree-based efficient non-dominated sorting approach known as T-ENS [15] has also been proposed in the literature. This approach first sorts the population based on the first objective to ensure that the latter solutions cannot dominate the former solutions. In this approach, a non-dominated front is represented as a tree which keeps track of the non-domination relationships between the solutions. The use of the tree saves many unnecessary dominance comparisons. The worst case time complexity of T-ENS is $\mathcal{O}(MN^2)$. The best case time complexity is $\mathcal{O}(MN \log N / \log M)$. However, T-ENS is not suitable when the solutions share identical values for any of the objectives [16]. Recently, an efficient non-dominated sorting approach with non-dominated tree (ENS-NDT) [16] has been developed by extending ENS-BS [7], with a worst case time complexity of $\mathcal{O}(MN^2)$. The best case time complexity of ENS-NDT is $\mathcal{O}(MN \log N)$ when $M > \log N$; otherwise, it is $\mathcal{O}(N \log^2 N)$. Few other approaches like [17, 18, 19, 20] have been recently proposed for non-dominated sorting.

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

Some of these approaches [21, 22, 23, 24, 25, 26, 27] were proposed for steady-state evolutionary algorithms where a solution needs to be inserted into a set of fronts.

3. Proposed Framework

In the current work, we have developed a divide-and-conquer based framework for non-dominated sorting. We call this framework DCNS (Divide-and-Conquer based Non-dominated Sorting). The DCNS framework is shown in Algorithm 1. DCNS is a two-phased framework.

In the first phase, sorting of the solutions is performed based on the first objective [2, 3, 7, 15, 16]. If two solutions have the same value for the first objective, then the values of the second objective are considered for sorting. If two solutions have the same value for the second objective, then the values of the third objective are considered for sorting and so on. If two solutions are the same (in terms of objectives values), then any order can be followed. In this manner, solutions are sorted. This phase is called Pre-sorting. After sorting the population based on the first objective, a solution sol_i will never dominate a solution sol_j if $i > j, 1 \leq i, j \leq N$ [2, 3, 7, 15, 16].

Algorithm 1 DCNS framework

Input: $\mathbb{P}$: Population in M-dimensional space
Output: Non-dominated fronts in sorted order

1: Sort $\mathbb{P}$ in ascending order based on the first objective
   // Assign the sorted solutions to $\mathbb{F}$
2: **for** $i \leftarrow 1$ to N **do** // Consider the solutions in sorted order
3: $F_1 \leftarrow \{\mathbb{P}(i)\}$ // Consider a solution as a front
4: $\mathcal{F}_i \leftarrow \{F_1\}$ // Consider a front as a set of fronts
5: $\mathbb{F} \leftarrow \mathbb{F} \cup \{\mathcal{F}_i\}$ // Add set of fronts to $\mathbb{F}$
   // Steps of non-dominated sorting
6: **for** $i \leftarrow 1$ to $\lceil \log_2 N \rceil$ **do** // For each level
7: **for each** set of fronts $\mathcal{F} \in \mathbb{F}$ **do**
8: $\mathcal{F}' \leftarrow$ Next set of fronts to $\mathcal{F}$
9: **if** $\mathcal{F}'$ exists **then**
10: MERGE($\mathcal{F}, \mathcal{F}'$) // Insert all the solutions from $\mathcal{F}'$ to $\mathcal{F}$
11: Delete $\mathcal{F}'$
12: **return** $\mathbb{F}(1)$ // Final non-dominated fronts are in $\mathbb{F}(1)$

In the second phase, the actual sorting is performed. Our approach is based on divide-and-conquer strategy. 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. Initially, a set of fronts contains only one solution, *i.e.*, each set of fronts has a single front and this single front has only one solution. Formally, let $\mathbb{F} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_N\}$ be the N sets of fronts where each set of fronts $\mathcal{F}_i = \{F_1\}$

and $F_1 = \{sol_i\}, 1 \leq i \leq N$. The set of fronts in $\mathbb{F}$ at the i^{th} position is referred to as $\mathbb{F}(i), 1 \leq i \leq N$.

The DCNS framework is based on a divide-and-conquer strategy. A binary tree type structure is followed which observes a bottom up strategy. The basis of the framework is a merge operation which merges two sets of fronts. The merge operations are performed in a total of $\lceil \log_2 N \rceil$ levels because of the height of the tree which is $\lceil \log_2 N \rceil$. At each level, two consecutive sets of fronts are merged. The merge operation is performed using the MERGE($\mathcal{F}, \mathcal{F}'$) procedure which is described in detail in Section 4. The following example illustrates the working flow of the proposed framework.

Example 2. *Let $\mathbb{P} = \{sol_1, sol_2, \ldots, sol_8\}$ be a population of eight solutions in 2-dimensional objective space as shown in Figure 1. The first phase sorts the solutions based on the first objective. In the second phase, the actual sorting is performed. As the number of solutions is eight, the height of the tree $\mathcal{L} = \lceil \log 8 \rceil = 3$. Hence, the merge operations are performed at three different levels. The complete procedure is shown in Figure 2.*

4. Merge Procedure

The merge procedure is summarized in Algorithm 2 which merges two sets of fronts $\mathcal{F} = \{F_1, F_2, \ldots, F_P\}$ and $\mathcal{F}' = \{F'_1, F'_2, \ldots, F'_Q\}$. The u^{th} solution in front $F_p (1 \leq p \leq P)$ is denoted as $F_p(u)$. Similarly, the v^{th} solution in front $F'_q (1 \leq q \leq Q)$ is denoted as $F'_q(v)$. The fronts in these sets of fronts are arranged in decreasing order of their dominance. In the merge procedure, initially the solutions of F'_1 are inserted into $\mathcal{F}$, then the solutions of F'_2 are inserted and so on. Whenever a solution is inserted in $\mathcal{F}$, it is removed from $\mathcal{F}'$ so that a solution occupies only one place. After the insertion of all the solutions from front $F' \in \mathcal{F}'$ in $\mathcal{F}$, front F' is also removed (see lines 4 and 9 of Algorithm 2). In the merge procedure, the dominance relationships as discussed in [2] are considered which helps in avoiding unnecessary dominance comparisons.

In the merge procedure, the insertion of a front $F' \in \mathcal{F}'$ in $\mathcal{F}$ is performed either by Algorithm 3 (INSERT() procedure) which uses constant space or by Algorithm 6 (INSERT-WS() procedure) which uses $\mathcal{O}(N)$ space.

Initially, the solutions of front F'_1 are inserted which find their positions in $\mathcal{F}$ by comparing with the solutions of different fronts starting from the first front. When the solutions of the next front (*i.e.*, F'_2) are inserted, then these solutions do not start comparing with the solutions of the first front in $\mathcal{F}$ because of the dominance relationship [2]. Let the index of the front in $\mathcal{F}$ from where the solutions of front $F' \in \mathcal{F}'$ start the comparison be denoted by α, *i.e.*, the solutions of F' start comparing with F_α. The solutions of the first front F'_1 start comparing with F_1 so the initial call to either INSERT() or INSERT-WS() is with $\alpha = 1$.

When a front F' is inserted into $\mathcal{F}$ using either the INSERT() or the INSERT-WS() procedure, then it returns an index value which indicates the index of the

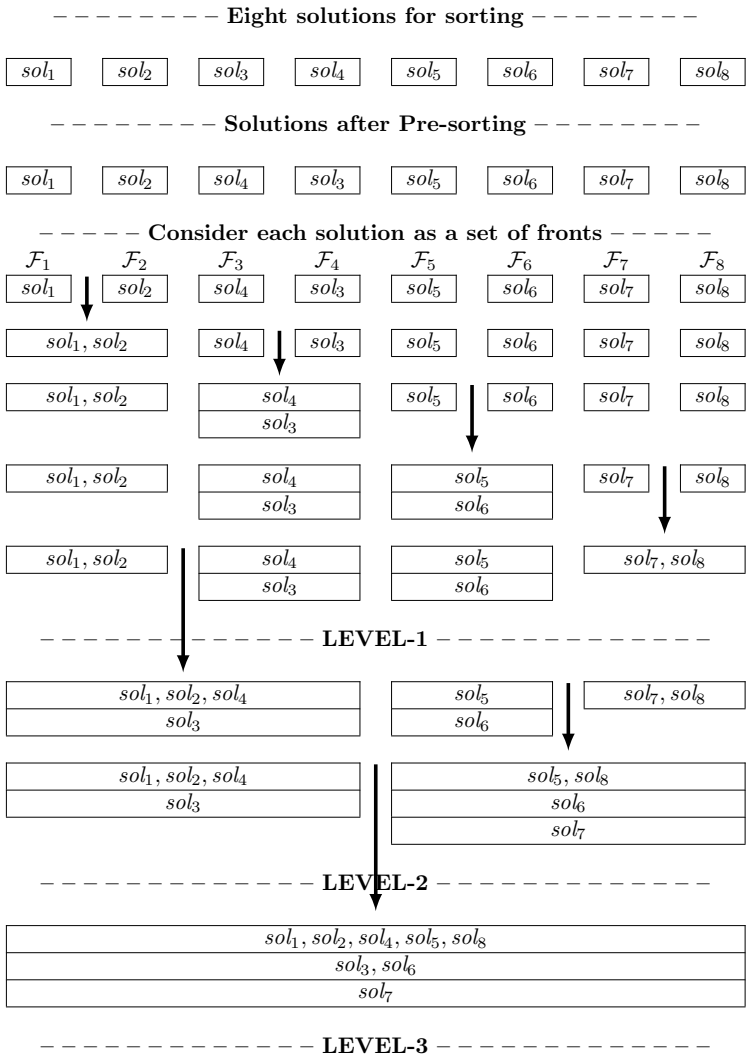


Figure 2: Working flow of the proposed DCNS framework. ↓ indicates the merge operation between the immediate left and right set of fronts. $sol_i, \ldots, sol_j$ represent that these solutions are non-dominated to each other. The solutions are merged at three different levels.

front in $\mathcal{F}$ having the highest dominance where the solutions of front F' have been inserted. When the next front of $\mathcal{F}'$ is inserted into $\mathcal{F}$, then solutions of this front start the dominance comparison with front $F_{\alpha+1}$. Each time a front F' is inserted into $\mathcal{F}$, α is updated accordingly. If the value of α is equal to the cardinality of $\mathcal{F}$ after the insertion of front F', *i.e.*, all the solutions of F' are either inserted in F_P or in F_{P+1}, then all the remaining fronts of $\mathcal{F}'$ are inserted into $\mathcal{F}$ directly without any dominance comparison with the solutions

Algorithm 2 MERGE($\mathcal{F}, \mathcal{F}'$)

Input: Two sets of fronts $\mathcal{F}$, $\mathcal{F}'$
Output: Updated $\mathcal{F}$ after insertion of all the solutions from $\mathcal{F}'$

1: $\alpha \leftarrow 0$
2: **for each** front $F' \in \mathcal{F}'$ **do**
3: $\quad \alpha \leftarrow$ INSERT($\mathcal{F}, F', \alpha + 1$) // Insert all the solutions from F' to $\mathcal{F}$
4: $\quad \mathcal{F}' \leftarrow \mathcal{F}' \setminus \{F'\}$ // Delete the inserted front from $\mathcal{F}'$
5: $\quad$ **if** $\alpha = |\mathcal{F}|$ **then**
6: $\quad\quad$ **BREAK**
// Add the solutions of remaining fronts of $\mathcal{F}'$ to $\mathcal{F}$ without comparison
7: **for each** front $F' \in \mathcal{F}'$ **do**
8: $\quad \mathcal{F} \leftarrow \mathcal{F} \cup \{F'\}$ // Add all the solutions of F' to $\mathcal{F}$
9: $\quad \mathcal{F}' \leftarrow \mathcal{F}' \setminus \{F'\}$ // Delete the inserted front from $\mathcal{F}'$

of $\mathcal{F}$ (see lines 7 – 9 of Algorithm 2). Next, we discuss the INSERT() and the INSERT-WS() procedures one by one in detail.

Algorithm 3 INSERT($\mathcal{F}, F', \alpha$)

Input: $\mathcal{F}$: Set of fronts, F': Front for insertion in $\mathcal{F}$, α: Index of the front in $\mathcal{F}$ from where the solutions of F' start the dominance comparison
Output: *hfi*: Index of the front in $\mathcal{F}$ having the highest dominance in which the solutions of front F' are inserted

1: $P \leftarrow |\mathcal{F}|$ // Store the cardinality of $\mathcal{F}$
2: $hfi \leftarrow P + 1$ // Initialize *hfi*
3: $\Upsilon_{\text{fIndex}} \leftarrow 0$, $\Upsilon_{\text{nSol}} \leftarrow 0$ // Initialize Υ
4: **for each** solution $sol \in F'$ **do**
5: $\quad$ INSERT-SS($\mathcal{F}, sol$) // Insert *sol* in $\mathcal{F}$
6: $\quad F' \leftarrow F' \setminus \{sol\}$ // Delete *sol* from F'
7: **return** *hfi*

4.1. Insert Procedure without Extra Space

The procedure to insert the solutions of a front F' in $\mathcal{F}$, which requires constant space is discussed in Algorithm 3. This procedure uses a variable *hfi* (*highest front index*) to keep track of α. *hfi* stores the index of the front in $\mathcal{F}$ having the highest dominance in which the solutions of front F' are inserted. As all the solutions of F' are non-dominated with each other, so, when the solutions of F' are inserted into $\mathcal{F}$, then the solutions of F' can add a maximum of one front in $\mathcal{F}$. The solutions of F' can also be inserted into the existing fronts in $\mathcal{F}$. It may also be possible that some of the solutions of F' are inserted into the existing fronts, whereas some are inserted into the newly created front. So, initially the value of *hfi* is set to $P + 1$ where P is the number of fronts in $\mathcal{F}$ before insertion of front F'.

A set $\Upsilon = \{\Upsilon_{\text{fIndex}}, \Upsilon_{\text{nSol}}\}$ stores two pieces of information – (i) the index of the front in $\mathcal{F}$ in which the previous solution of front F' was inserted (denoted by Υ_{fIndex}) and (ii) the number of solutions in $F_{\Upsilon_{\text{fIndex}}}$ when the last time a solution of front F' was inserted in a different front other than $F_{\Upsilon_{\text{fIndex}}}$ (denoted by Υ_{nSol}). The set Υ is used to avoid unnecessary dominance comparisons. Initially, Υ_{fIndex} is set to 0 and Υ_{nSol} is also set to 0. Consider the following example which illustrates the importance of Υ.

Example 3. *Let $\mathcal{F} = \{F_1\}$ be the set of fronts where there is only a single front. $F_1 = \{sol_1, sol_2, \ldots, sol_8\}$. Consider the solutions of a front $F' = \{sol_9, sol_{10}, \ldots, sol_{16}\}$ which need to be inserted in $\mathcal{F}$. Let all the solutions of front F' will be inserted in F_1. When sol_9 is inserted into $\mathcal{F}$, it is compared with all the eight solutions in F_1. Now, $\Upsilon_{fIndex} = 1$ and $\Upsilon_{nSol} = 8$. When sol_{10} is inserted into $\mathcal{F}$, it is compared with the first eight ($\Upsilon_{nSol} = 8$) solutions in F_1 and not with nine solutions. As sol_{10} and sol_9 are from front F' hence both are non-dominated, so there is no need to compare sol_{10} with sol_9. The remaining solutions of F' are only compared with the first eight solutions. The use of Υ helps in reducing the number of dominance comparisons. Table 1 shows the number of dominance comparisons after insertion of each solution of front F' in $\mathcal{F}$ considering Υ and without considering Υ. This table shows that the number of dominance comparisons is reduced when Υ is used.*

Table 1: Number of dominance comparisons required to insert the solutions of front F' in $\mathcal{F}$.

Inserted Solution	sol_9	sol_{10}	sol_{11}	sol_{12}	sol_{13}	sol_{14}	sol_{15}	sol_{16}
Dominance comparisons without Υ	8	9	10	11	12	13	14	15
Dominance comparisons with Υ	8	8	8	8	8	8	8	8

The solution $sol \in F'$ can be inserted in $\mathcal{F}$ using a sequential search based strategy or a binary search based strategy as in [7, 2].

4.1.1. Sequential Search based Strategy

The procedure to insert a solution *sol* in $\mathcal{F}$ using a sequential search based strategy is given in Algorithm 4. *sol* is compared with the solutions of each of the fronts in $\mathcal{F}$ starting from F_α to F_P in a sequential manner.

A set Υ is considered to reduce the unnecessary dominance comparisons. So, we first check whether the previous solution was inserted in the p^{th} front which is to be explored (line 5). If this condition is true, then *sol* is compared with the initial Υ_{nSol} solutions of F_p; otherwise, *sol* is compared with all the solutions of F_p. If *sol* is non-dominated with respect to all the solutions of F_p, then *sol* is inserted in F_p, and *hfi* and Υ are updated accordingly. If *sol* is dominated by any of the solutions in F_p, then *sol* is compared with the solutions of the next front, *i.e.*, F_{p+1}. If *sol* is dominated by each of the compared fronts, then *sol* is inserted into F_{P+1}.

Algorithm 4 INSERT-SS($\mathcal{F}$, *sol*)

Input: $\mathcal{F}$: Set of fronts, *sol*: Solution for insertion in $\mathcal{F}$
Output: Updated Υ and *hfi* based on the insertion of *sol*

1: *insertionDone* $\leftarrow$ FALSE // *sol* is not yet inserted
2: **for** $p \leftarrow \alpha$ to P **do** // Check for each front in $\mathcal{F}$ starting from α^{th} front
3: *domCount* $\leftarrow 0$
4: *nSolFront* $\leftarrow |F_p|$ // Number of solutions in F_p
5: **if** $p = \Upsilon_{\text{fIndex}}$ **then** // Previous solution of F'(*sol*$\in F'$) was inserted in F_p
6: *nSolFront* $\leftarrow \Upsilon_{\text{nSol}}$ // *sol* will be compared with maximum *nSolFront* solutions in F_p
7: **for** $u \leftarrow 1$ **to** *nSolFront* **do**
8: **if** *sol* is non-dominated with $F_p(u)$ **then**
9: *domCount* $\leftarrow$ *domCount* $+ 1$
10: **else**
11: **BREAK** // Check for next front in $\mathcal{F}$
12: **if** *domCount* $=$ *nSolFront* **then** // *sol* is non-dominated with all the solutions of F_p
13: $F_p \leftarrow F_p \cup \{sol\}$ // Insert *sol* in F_p
14: *insertionDone* $\leftarrow$ TRUE // Insertion of *sol* is done
15: **if** $p <$ *hfi* **then** // *sol* is inserted into higher dominance front than *hfi*
16: *hfi* $\leftarrow p$ // Update *hfi*
17: **if** $p \neq \Upsilon_{\text{fIndex}}$ **then**
18: $\Upsilon_{\text{fIndex}} \leftarrow p$, $\Upsilon_{\text{nSol}} \leftarrow$ *domCount* // Update Υ
19: **BREAK** // Do not check other fronts
20: **if** *insertionDone* $=$ FALSE **then** // *sol* is not yet inserted
21: $F_{P+1} \leftarrow F_{P+1} \cup \{sol\}$ // Insert *sol* in front F_{P+1}

4.1.2. Binary Search based Strategy

The procedure to insert a solution *sol* in $\mathcal{F}$ using a binary search based strategy is given in Algorithm 5. As opposed to the sequential search based strategy, *sol* is not compared with each of the fronts in $\mathcal{F}$ starting from F_α to F_P, but rather only $\lceil \log(P - \alpha + 2) \rceil$ fronts are considered for comparison purposes. Here, we are not creating the tree explicitly, rather the set of fronts are visualized as a tree.

We have considered two variables L and R to follow the tree structure. Initially, L is set to α and R is set to P. At first, *sol* is compared with F_{mid} where $mid = \lfloor (L+R)/2 \rfloor$. Like sequential search, we first check whether the previous solution was inserted in F_{mid} or not (line 5). If this condition is true, then *sol* is compared with the initial Υ_{nSol} solutions of F_{mid}; otherwise, *sol* is compared with all the solutions of F_{mid}. If *sol* is non-dominated with respect to all the solutions of F_{mid} to which it is compared (line 12), then there are two possibilities:

- If a leaf of the tree is reached (*i.e.*, $mid = L$), then *sol* is inserted in F_{mid}

Algorithm 5 INSERT-BS($\mathcal{F}, sol$)

Input: $\mathcal{F}$: Set of fronts, *sol*: Solution for insertion in $\mathcal{F}$
Output: Updated Υ and *hfi* based on the insertion of *sol*

1: $L \leftarrow \alpha$, $R \leftarrow P$, $mid \leftarrow \lfloor (L+R)/2 \rfloor$
2: **while** TRUE **do** // Position of *sol* in $\mathcal{F}$ is not identified
3: $domCount \leftarrow 0$
4: $nSolFront \leftarrow |F_{\text{mid}}|$ // Number of solutions in F_{mid}
5: **if** $mid = \Upsilon_{\text{fIndex}}$ **then** // Previous solution of $F'(sol \in F')$ was inserted in F_{mid}
6: $nSolFront \leftarrow \Upsilon_{\text{nSol}}$ // *sol* will be compared with a maximum of *nSolFront* solutions in F_{mid}
7: **for** $u \leftarrow 1$ **to** $nSolFront$ **do**
8: **if** *sol* is non-dominated with respect to $F_{\text{mid}}(u)$ **then**
9: $domCount \leftarrow domCount + 1$
10: **else**
11: **BREAK** // Check for other front in $\mathcal{F}$
12: **if** $domCount = nSolFront$ **then** // *sol* is non-dominated with respect to all the solu- tions of F_{mid}
13: **if** $mid = L$ **then** // The front at leaf is explored
14: $F_{\text{mid}} \leftarrow F_{\text{mid}} \cup \{sol\}$ // Insert *sol* in F_{mid}
15: **if** $mid < hfi$ **then** // *sol* is inserted into higher dominance front than *hfi*
16: $hfi \leftarrow mid$ // Update *hfi*
17: **if** $mid \neq \Upsilon_{\text{fIndex}}$ **then**
18: $\Upsilon_{\text{fIndex}} \leftarrow mid$, $\Upsilon_{\text{nSol}} \leftarrow domCount$ // Update Υ
19: **BREAK** // Insertion of *sol* is done
20: **else**
21: $R \leftarrow mid - 1$, $mid \leftarrow \lfloor (L+R)/2 \rfloor$ // Explore left sub-tree
22: **else**
23: **if** $L = P$ **then** // Right most leaf is explored
24: $F_{P+1} \leftarrow F_{P+1} \cup \{sol\}$ // Insert *sol* in F_{P+1}
25: **BREAK** // Insertion of *sol* is done
26: **else if** $mid = R$ **then** // *sol* is dominated by leaf node
27: $F_{R+1} \leftarrow F_{R+1} \cup \{sol\}$ // Insert *sol* in F_{R+1}
28: **BREAK** // Insertion of *sol* is done
29: **else**
30: $L \leftarrow mid + 1$, $mid \leftarrow \lfloor (L+R)/2 \rfloor$ // Explore right sub-tree

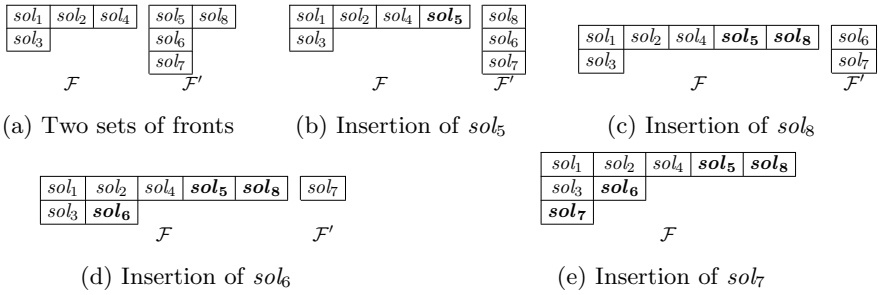
and *hfi* as well as Υ is updated. After this, the process terminates.

- Otherwise, ($mid \neq L$) the root of the left sub-tree is checked.

If *sol* is dominated by any of the solutions of F_{mid}, then there are three possibilities:

- If the rightmost node of the tree is reached (*i.e.*, $L = P$), then *sol* is inserted in F_{P+1} and the process terminates.
- If *sol* is dominated by the leaf node, (*i.e.*, $mid = R$), then *sol* is inserted in F_{R+1} and the process terminates.
- Otherwise, the root of the right sub-tree is checked.

Example 4. *Consider two sets of fronts* $\mathcal{F} = \{\{sol_1, sol_2, sol_4\}, \{sol_3\}\}$ *and* $\mathcal{F}' = \{\{sol_5, sol_8\}, \{sol_6\}, \{sol_7\}\}$ *which are merged at the last level in Figure 2. Figure 3 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 three times corresponding to the number of 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 and third* INSERT() *procedures, the* INSERT-SS() *procedure is called once as there is only a single solution in the second and third fronts of* $\mathcal{F}'$*.*



(a) Two sets of fronts (b) Insertion of sol_5 (c) Insertion of sol_8

(d) Insertion of sol_6 (e) Insertion of sol_7

Figure 3: 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 solution which is added to $\mathcal{F}$ is shown in **boldface** in $\mathcal{F}$.

Algorithm 6 INSERT-WS($\mathcal{F}, F', \alpha$)

Input: Same as Algorithm 3
Output: Same as Algorithm 3

1: $P \leftarrow |\mathcal{F}|$ // Store the cardinality of $\mathcal{F}$
2: *hfi* $\leftarrow P + 1$ // Initialize *hfi*
3: $\Psi[1, 2, \ldots, P-\alpha+1] \leftarrow \Phi$ // Initialize an array to store the cardinality of the fronts in $\mathcal{F}$
4: **for** $p \leftarrow \alpha$ to P **do**
5: $\quad \Psi[p - \alpha + 1] \leftarrow |F_p|$ // Store the cardinality of front F_p
6: **for each** solution $sol \in F'$ **do**
7: $\quad$ INSERT-SS-WS($\mathcal{F}, sol$) // Insert *sol* in $\mathcal{F}$
8: $\quad F' \leftarrow F' \setminus \{sol\}$ // Delete *sol* from F'
9: **return** *hfi*

Table 2: Maximum number of dominance comparisons required to insert the solutions of front F in $\mathcal{F}$.

Inserted Solution	sol_9	sol_{10}	sol_{11}	sol_{12}	sol_{13}	sol_{14}	sol_{15}	sol_{16}
Dominance comparisons with Υ	8	4	8	4	8	4	8	4
Dominance comparisons with Ψ	8	4	9	5	11	6	13	7

4.2. Extra space is required

The procedure to insert all the solutions of front F' in $\mathcal{F}$ is presented in Algorithm 6 when extra space is required for insertion of a solution in the set of fronts. When the solutions from a front F' are inserted into $\mathcal{F}$, then the number of fronts in $\mathcal{F}$ can be increased by 1. So, the value of *hfi* is set to $P+1$ where P is the number of fronts in $\mathcal{F}$ before insertion of front F'. In this procedure, before insertion of the solutions from front F' into $\mathcal{F}$, a vector Ψ of length $P - \alpha + 1$ is initialized with the number of solutions in each front $F_p, \alpha \leq p \leq P$. This is used to prevent unnecessary dominance comparisons when multiple solutions from front F' are inserted in a particular front F_p. When a solution from front F' is compared with the solutions of F_p, then it is compared with the initial $\Psi[p - \alpha + 1]$ solutions and not with all the solutions. This is because all the solutions of front F' are non-dominated with each other. So, there is no need to compare them with each other. Consider the following example which illustrates the benefit of storing the number of solutions in each front in $\mathcal{F}$.

Example 5. *Let $\mathcal{F} = \{F_1, F_2\}$ be the set of two fronts where $F_1 = \{sol_1, sol_2, sol_3, sol_4\}$ and $F_2 = \{sol_5, sol_6, sol_7, sol_8\}$. Consider the solutions of front $F' = \{sol_9, sol_{10}, \ldots, sol_{16}\}$ which need to be inserted into $\mathcal{F}$ in alternate fronts (sol_9 $sol_{11}, sol_{13}, sol_{15}$ are inserted in F_2 and $sol_{10}, sol_{12}, sol_{14}, sol_{16}$ are inserted in F_1). When the insertion is performed using Algorithm 3 (Υ is used) and Algorithm 6 (Ψ is used) considering a sequential search based strategy, then the maximum number of dominance comparisons corresponding to the insertion of each of the solutions is given in Table 2. The calculation of the number of dominance comparisons given in Table 2 is explained in Appendix A.*

There are two ways to insert a solution $sol \in F'$ in $\mathcal{F}$ depending on the search type – sequential or binary.

4.2.1. Sequential Search based Strategy

The sequential search based strategy to insert a solution *sol* in $\mathcal{F}$ is summarized in Algorithm 7. *sol* is compared with each of the fronts in $\mathcal{F}$ starting from F_α to F_P in a sequential manner. When *sol* is compared with a front F_p, then it is compared only with the initial $\Psi[p - \alpha + 1]$ solutions and not with all of them. If *sol* is non-dominated with respect to all the solutions to which it is compared, then it is inserted in F_p and *hfi* is updated accordingly. If *sol* is dominated by all the compared fronts, then *sol* is inserted into F_{P+1}.

4.2.2. Binary Search based Strategy

A binary search based strategy to insert a solution *sol* in $\mathcal{F}$ is summarized in Algorithm 8. Here also, when *sol* is compared with a front F_{mid}, then it is

Algorithm 7 INSERT-SS-WS($\mathcal{F}, sol$)

Input: $\mathcal{F}$: Set of fronts, *sol*: Solution for insertion in $\mathcal{F}$
Output: Updated *hfi* depending upon the insertion of *sol*

```
1: insertionDone ← FALSE                                   // sol is not yet inserted
2: for p ← α to P do          // Check for each front starting from α^th front
3:    domCount ← 0
4:    for u ← 1 to Ψ[p − α + 1] do
5:       if sol is non-dominated with F_p(u) then
6:          domCount ← domCount + 1
7:       else
8:          BREAK                                        // Check for next front in F
9:    if domCount = Ψ[p − α + 1] then  // sol is non-dominated with all the
       solutions of F_p
10:      F_p ← F_p ∪ {sol}                                // Insert sol in F_p
11:      insertionDone ← TRUE                          // Insertion of sol is done
12:      if p < hfi then     // sol is inserted into higher dominance front than hfi
13:         hfi ← p                                          // Update hfi
14:      BREAK                                      // Do not check other fronts
15: if insertionDone = FALSE then                        // sol is not yet inserted
16:    F_{P+1} ← F_{P+1} ∪ {sol}                     // Insert sol in front F_{P+1}
```

compared only with the initial $\Psi[mid - \alpha + 1]$ solutions and not with all the solutions.

Based on the space requirement and search type, there are four variants of our proposed DCNS framework.

(i). DCNS-SS: DCNS approach with constant space and sequential search
(ii). DCNS-BS: DCNS approach with constant space and binary search
(iii). DCNS-SS-WS: DCNS approach with linear space and sequential search
(iv). DCNS-BS-WS: DCNS approach with linear space and binary search

4.3. Time Complexity Analysis

The complexity of the DCNS framework is analyzed under various scenarios as discussed in [7] and [10]. Our DCNS framework consists of two phases. Heapsort [28] can be adopted in the first phase with time complexity $\mathcal{O}(N \log N)$ as in [7]. So, the worst case time complexity of the first phase is $\mathcal{O}(MN \log N)$. The best case time complexity is $\mathcal{O}(N \log N)$ when all the solutions have distinct values for the first objective. Let $\mathcal{L}$ be the height of the tree. Assume Γ_{SS}, $\Gamma_{\text{SS-WS}}$, Γ_{BS} and $\Gamma_{\text{BS-WS}}$ are the number of dominance comparisons performed by DCNS-SS, DCNS-SS-WS, DCNS-BS and DCNS-BS-WS, respectively. Now we discuss the time complexity of our framework in three different scenarios.

Algorithm 8 INSERT-BS-WS($\mathcal{F}, sol$)

Input: $\mathcal{F}$: Set of fronts, *sol*: Solution for insertion in $\mathcal{F}$
Output: Updated *hfi* depending upon the insertion of *sol*

1: $L \leftarrow \alpha$, $R \leftarrow P$, $mid \leftarrow \lfloor (L+R)/2 \rfloor$
2: **while** TRUE **do** // Position of *sol* in $\mathcal{F}$ is not identified
3: $domCount \leftarrow 0$
4: **for** $u \leftarrow 1$ **to** $\Psi[mid - \alpha + 1]$ **do**
5: **if** *sol* is non-dominated with $F_{\text{mid}}(u)$ **then**
6: $domCount \leftarrow domCount + 1$
7: **else**
8: **BREAK** // Check for other fronts in $\mathcal{F}$
9: **if** $domCount = \Psi[mid - \alpha + 1]$ **then** // *sol* is non-dominated with respect to all the solutions of F_{mid}
10: **if** $mid = L$ **then** // The front at leaf is explored
11: $F_{\text{mid}} \leftarrow F_{\text{mid}} \cup \{sol\}$ // Insert *sol* in F_{mid}
12: **if** $mid < hfi$ **then** // *sol* is inserted into higher dominance front than *hfi*
13: $hfi \leftarrow mid$ // Update *hfi*
14: **BREAK** // Insertion of *sol* is done
15: **else**
16: $R \leftarrow mid - 1$, $mid \leftarrow \lfloor (L+R)/2 \rfloor$ // Explore left sub-tree
17: **else**
18: **if** $L = P$ **then** // Right most leaf is explored
19: $F_{P+1} \leftarrow F_{P+1} \cup \{sol\}$ // Insert *sol* in F_{P+1}
20: **BREAK** // Insertion of *sol* is done
21: **else if** $mid = R$ **then** // *sol* is dominated by leaf node
22: $F_{R+1} \leftarrow F_{R+1} \cup \{sol\}$ // Insert *sol* in F_{R+1}
23: **BREAK** // Insertion of *sol* is done
24: **else**
25: $L \leftarrow mid + 1$, $mid \leftarrow \lfloor (L+R)/2 \rfloor$ // Explore right sub-tree

4.3.1. All Solutions are in a Single Front

In this case, binary search performs like sequential search as the number of fronts is one. The proposed approach with and without extra space also performs the same because there is a single front in each set of fronts at every level of the merge operations and the number of solutions in a single front is stored only. Thus, the values of Γ_{SS}, $\Gamma_{\text{SS-WS}}$, Γ_{BS} and $\Gamma_{\text{BS-WS}}$ are the same. Each set of fronts has a single front because all the solutions belong to a single front. Thus, in the MERGE() procedure, the INSERT() or the INSERT-WS() procedure is called just once. The number of solutions in each set of fronts at the l^{th} level is 2^{l-1}. So, the number of dominance comparisons using Algorithms 4, 5, 7 and 8 to insert a solution at the l^{th} level is 2^{l-1}. These algorithms are called 2^{l-1} times in Algorithms 3 or 6 corresponding to each of the 2^{l-1} solutions which need to be inserted. Thus, the number of dominance comparisons in the INSERT() or the

INSERT-WS() procedure at the l^{th} level is $2^{l-1} \times 2^{l-1}$. Therefore, the MERGE() procedure performs $2^{l-1} \times 2^{l-1}$ dominance comparisons at the l^{th} level. The number of merge operations at the l^{th} level is $N/2^l$. So, the total number of dominance comparisons in this case is obtained by Eq. (1).

$$\Gamma_{\text{SS}} = \sum_{l=1}^{\mathcal{L}} \left(\frac{N}{2^l}\right) \left(2^{l-1}\right) \left(2^{l-1}\right) = \frac{1}{2} N(N-1) \tag{1}$$

Thus, the time complexity of the second phase in this case is $\mathcal{O}(MN^2)$. The first phase has a worst case time complexity of $\mathcal{O}(MN \log N)$. So, the overall time complexity is $\mathcal{O}(MN^2)$.

4.3.2. All Solutions are in Separate Fronts

Here, binary and sequential search perform differently because the number of fronts is more than one. The proposed approach with and without extra space performs the same because each front has a single solution in all the set of fronts at every level of the merge operations and the number of solutions in a single front is stored only which is always 1. Thus, $\Gamma_{\text{SS}} = \Gamma_{\text{SS-WS}}$ and $\Gamma_{\text{BS}} = \Gamma_{\text{BS-WS}}$.

When two sets of fronts $\mathcal{F}$ and $\mathcal{F}'$ are merged, then after the insertion of the solution of the first front from $\mathcal{F}'$ in $\mathcal{F}$, the solutions of the remaining fronts from $\mathcal{F}'$ are added to $\mathcal{F}$ without performing any dominance comparison because all the solutions are in different fronts. Before the merge operation, the number of solutions in each set of fronts at the l^{th} level is 2^{l-1}. All the solutions in $\mathcal{F}'$ are dominated by each of the solutions of $\mathcal{F}$. So, a solution of $\mathcal{F}'$ needs to be compared and dominated by 2^{l-1} solutions in $\mathcal{F}$ using a sequential search based strategy. Using a binary search based strategy, a solution of $\mathcal{F}'$ needs to be compared and dominated by $\lceil \log\left(2^{l-1}+1\right) \rceil$ solutions in $\mathcal{F}$. The number of merge operation at the l^{th} level is $N/2^l$. Thus, the total number of dominance comparisons using sequential search is obtained by Eq. (2). The total number of dominance comparisons using binary search is obtained by Eq. (3).

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

$$\Gamma_{\text{BS}} = \sum_{l=1}^{\mathcal{L}} \left(\frac{N}{2^l}\right) \left(\left\lceil \log\left(2^{l-1}+1\right) \right\rceil\right) (1) = 2N - \log N - 2 \tag{3}$$

Before the merge operations at the l^{th} level, there are 2^{l-1} fronts in each set of fronts, out of which the solution of $2^{l-1}-1$ fronts are added directly to the $\mathcal{F}$ which requires $O(2^{l-1})$ time. Thus, the total time to directly add the solutions without dominance comparisons is $\sum_{l=1}^{\mathcal{L}} \frac{N}{2^l}\left(2^{l-1}-1\right) = \mathcal{O}(N \log N)$. Thus, the time complexity of the second phase is $\mathcal{O}(MN \log N + N \log N) = \mathcal{O}(MN \log N)$ using a sequential search based strategy, whereas $\mathcal{O}(MN + N \log N)$ using a binary search based strategy. The best case time complexity of the first phase is $\mathcal{O}(N \log N)$. Thus, the overall best case time complexity of both DCNS-BS and DCNS-BS-WS is $\mathcal{O}(N \log N + MN)$.

4.3.3. $\sqrt{N}$ solutions in each of the $\sqrt{N}$ fronts

Here, we discuss the time complexity when $\sqrt{N}$ fronts contain equal number of solutions. Each solution in a front dominates all the solutions in its succeeding front. Here, binary and sequential search perform differently because the number of fronts is more than one. Till the $\mathcal{L}/2^{th}$ level, there is a single front in each set of fronts at every level of the merge operations and the number of solutions in a single front is stored only. After the $\mathcal{L}/2^{th}$ level, the number of solutions in each front in both sets of fronts at every level of the merge operations is fixed to $\sqrt{N}$. So, the proposed approach with and without extra space performs the same because of the dominance nature of the solutions. Thus, $\Gamma_{\text{SS}} = \Gamma_{\text{SS-WS}}$ and $\Gamma_{\text{BS}} = \Gamma_{\text{BS-WS}}$.

As each front has $\sqrt{N} = 2^{\mathcal{L}/2}$ solutions, so till the $\mathcal{L}/2^{th}$ level, each set of fronts has a single front. Thus, in the MERGE() procedure, the INSERT() or the INSERT-WS() procedure is called just once. The number of solutions in each set of fronts at the l^{th} level is 2^{l-1}. So, the number of dominance comparisons using Algorithms 4, 5, 7 and 8 to insert a solution at the l^{th} level is 2^{l-1}. These algorithms are called 2^{l-1} times in Algorithms 3 or 6 corresponding to each of the 2^{l-1} solutions which need to be inserted. Thus, the number of dominance comparisons in the INSERT() or the INSERT-WS() procedure at the l^{th} level is $2^{l-1} \times 2^{l-1}$. Therefore, the MERGE() procedure performs $2^{l-1} \times 2^{l-1}$ dominance comparisons at the l^{th} level. The number of merge operations at the l^{th} level is $N/2^l$. So, the total number of dominance comparisons in this case till the $\mathcal{L}/2^{th}$ level is $\sum_{l=1}^{\mathcal{L}/2} \left(\frac{N}{2^l}\right) \left(2^{l-1}\right) \left(2^{l-1}\right)$.

After $\mathcal{L}/2^{th}$ level, each set of fronts has $2^{l-(\mathcal{L}/2+1)}$ fronts at the l^{th} level and each front has $\sqrt{N} = 2^{\mathcal{L}/2}$ solutions. When two sets of fronts $\mathcal{F}$ and $\mathcal{F}'$ are merged at the $l^{th}(\mathcal{L}/2+1 \leq l \leq \mathcal{L})$ level, then the solutions of only the first front of $\mathcal{F}'$ are compared with the solutions of $\mathcal{F}$. The solutions of the remaining $2^{l-(\mathcal{L}/2+1)} - 1$ fronts from $\mathcal{F}'$ are added directly to $\mathcal{F}$. The solutions of the first front in $\mathcal{F}'$ are compared with only a single solution in each of the fronts in $\mathcal{F}$ because each solution in a front is dominated by all the solutions in its preceding front. Thus, the number of dominance comparisons after the $\mathcal{L}/2^{th}$ level performed by DCNS-SS and DCNS-SS-WS is $\sum_{l=\mathcal{L}/2+1}^{\mathcal{L}} \left(\frac{N}{2^l}\right) \left(2^{l-(\mathcal{L}/2+1)}\right) \left(2^{\mathcal{L}/2}\right)$ and the number of dominance comparisons performed by DCNS-BS and DCNS-BS-WS is $\sum_{l=\mathcal{L}/2+1}^{\mathcal{L}} \left(\frac{N}{2^l}\right) \left(\left\lceil \log\left(2^{l-(\mathcal{L}/2+1)}+1\right)\right\rceil\right) \left(2^{\mathcal{L}/2}\right)$. The values of Γ_{SS} and $\Gamma_{\text{SS-WS}}$ can be obtained using Eq. (4). Similarly, the values of Γ_{BS} and $\Gamma_{\text{BS-WS}}$ can be obtained using Eq. (5).

$$\begin{aligned}\Gamma_{\text{SS}} &= \sum_{l=1}^{\mathcal{L}/2} \left(\frac{N}{2^l}\right) \left(2^{l-1}\right) \left(2^{l-1}\right) + \sum_{l=\mathcal{L}/2+1}^{\mathcal{L}} \left(\frac{N}{2^l}\right) \left(2^{l-\left(\frac{\mathcal{L}}{2}+1\right)}\right) \left(2^{\frac{\mathcal{L}}{2}}\right) \\ &= \frac{1}{2}N\left(\sqrt{N}-1\right) + \frac{1}{4}N\log N \qquad (4)\end{aligned}$$

$$\Gamma_{\text{BS}} = \sum_{l=1}^{\mathcal{L}/2} \left(\frac{N}{2^l}\right) \left(2^{l-1}\right) \left(2^{l-1}\right) + \sum_{l=\mathcal{L}/2+1}^{\mathcal{L}} \left(\frac{N}{2^l}\right) \left(\left\lceil \log\left(2^{l-\left(\frac{\mathcal{L}}{2}+1\right)}+1\right)\right\rceil\right) \left(2^{\frac{\mathcal{L}}{2}}\right)$$

$$= \frac{1}{2}N\left(\sqrt{N}-1\right) + \frac{1}{2}\sqrt{N}\left(4\sqrt{N}-\log N-4\right) \quad (5)$$

When two sets of fronts $\mathcal{F}$ and $\mathcal{F}'$ are merged at the $l^{th}(\mathcal{L}/2+1 \leq l \leq \mathcal{L})$ level, then the solutions of only the first front of $\mathcal{F}'$ are compared with the solutions of $\mathcal{F}$. The solutions of the remaining $2^{l-(\mathcal{L}/2+1)}-1$ fronts from $\mathcal{F}'$ are added directly to $\mathcal{F}$. So, a total of $\sum_{l=\mathcal{L}/2+1}^{\mathcal{L}}\left(\frac{N}{2^l}\right)\left(2^{l-(\mathcal{L}/2+1)}-1\right)\left(2^{\mathcal{L}/2}\right)$ solutions are added directly to $\mathcal{F}$ which requires $\mathcal{O}(N\log N)$ time. Thus, the time complexity of the second phase of all the four variants is $\mathcal{O}(MN\sqrt{N}+N\log N)$ which is $\mathcal{O}(MN\sqrt{N})$. The first phase has a worst case time complexity $\mathcal{O}(MN\log N)$. So, the overall time complexity is $\mathcal{O}(MN\sqrt{N})$.

The number of dominance comparisons performed by different non-dominated sorting approaches in three different scenarios are given in Table 3. This table clearly shows that the DCNS-based approaches are efficient regarding the number of dominance comparisons in two scenarios.

Table 3: Number of dominance comparisons performed by various non-dominated sorting approaches in three different scenarios.

Approach	**N solutions in single front**	**N solutions in N fronts**	**N solutions are equally divided into $\sqrt{N}$ fronts**
FNDS [6]	$\frac{1}{2}N(N-1)$	$\frac{1}{2}N(N-1)$	$\frac{1}{2}N(N-1)$
Deductive [10]	$\frac{1}{2}N(N-1)$	$\frac{1}{2}N(N-1)$	$\frac{1}{2}(N-1)(\sqrt{N}+1)^*$
ENS-SS [7]	$\frac{1}{2}N(N-1)$	$\frac{1}{2}N(N-1)$	$N(\sqrt{N}-1)$
ENS-BS [7]	$\frac{1}{2}N(N-1)$	$N\log N-(N-1)$	$\frac{1}{2}N(\sqrt{N}-1)+N\log\sqrt{N}-\sqrt{N}(\sqrt{N}-1)$
DCNS-SS	$\frac{1}{2}N(N-1)$	$\boldsymbol{\frac{1}{2}N\log N}$	$\boldsymbol{\frac{1}{2}N(\sqrt{N}-1)+\frac{1}{4}N\log N}$
DCNS-SS-WS			
DCNS-BS	$\frac{1}{2}N(N-1)$	$\boldsymbol{2N-\log N-2}$	$\boldsymbol{\frac{1}{2}N(\sqrt{N}-1)+\frac{1}{2}\sqrt{N}(4\sqrt{N}-\log N-4)}$
DCNS-BS-WS			

*Assumption: The first solution selected at each iteration is in the current front [10].

4.4. Space Complexity Analysis

Our DCNS framework consists of two phases. Heapsort [28] can be adopted in the first phase with a space complexity of $\mathcal{O}(1)$ as in [7]. The first phase of our DCNS framework works in constant space so the first phase does not have any role in the overall space complexity of our DCNS framework. The space complexity consists of the extra space required except for the initial population and the final fronts $F_k(1 \leq k \leq K)$. Each solution is considered as a set of fronts. Thus, initially there are N sets of fronts (which contain a single solution). As the sorting proceeds, these sets of fronts get reduced and finally it becomes one which contains the final fronts. The list data structure can be used to store the set of fronts.

The important point in space complexity is how the merge operation is preformed: (i) without storing the cardinality of each front in the first set

Table 4: Space and Time complexities of different approaches.

Approach	**Space Complexity**	**Time Complexity**	
		Best Case	**Worst Case**
Naive approach	$\mathcal{O}(N)$	$\mathcal{O}(MN^2)$	$\mathcal{O}(MN^3)$
FNDS [6]	$\mathcal{O}(N^2)$	$\mathcal{O}(MN^2)$	$\mathcal{O}(MN^2)$
Jensen [3]	$\mathcal{O}(MN)$	$\mathcal{O}(MN \log N)$	$\mathcal{O}(N \log^{M-1} N)$
Deductive Sort [10]	$\mathcal{O}(N)$	$\mathcal{O}(MN\sqrt{N})$	$\mathcal{O}(MN^2)$
ENS-SS [7]	$\mathcal{O}(1)$	$\mathcal{O}(MN\sqrt{N})$	$\mathcal{O}(MN^2)$
ENS-BS [7]	$\mathcal{O}(1)$	$\mathcal{O}(MN \log N)$	$\mathcal{O}(MN^2)$
BOS [1]	$\mathcal{O}(MN)$	$\mathcal{O}(MN \log N)$	$\mathcal{O}(MN^2)$
T-ENS† [15]	$\mathcal{O}(MN)$	$\mathcal{O}(^{MN \log N}/_{\log M})$	$\mathcal{O}(MN^2)$
ENS-NDT‡ [16]	$\mathcal{O}(N \log N)$	$\mathcal{O}(MN \log N)$	$\mathcal{O}(MN^2)$
DCNS-SS	$\mathcal{O}(1)$	$\mathcal{O}(MN \log N)$	$\mathcal{O}(MN^2)$
DCNS-BS	$\mathcal{O}(N)$	$\boldsymbol{\mathcal{O}(N \log N + MN)}$	$\mathcal{O}(MN^2)$
DCNS-SS-WS	$\mathcal{O}(N)$	$\mathcal{O}(MN \log N)$	$\mathcal{O}(MN^2)$
DCNS-BS-WS	$\mathcal{O}(N)$	$\boldsymbol{\mathcal{O}(N \log N + MN)}$	$\mathcal{O}(MN^2)$

†Not suitable when the solutions share identical values for any of the objectives [16].

‡Space complexity – worst: $\mathcal{O}(N \log N)$, best: $\mathcal{O}(\log N)$, average: $\mathcal{O}(N)$.

of fronts which is considered in DCNS-SS and DCNS-BS and (ii) storing the cardinality considered in DCNS-SS-WS and DCNS-BS-WS. In the first case, only Υ is used which takes constant space, whereas the space required by the second case is $\mathcal{O}(N)$. In case of a sequential search based strategy, the fronts in $\mathcal{F}$ are accessed sequentially. However, in the case of the binary search based strategy, there is a direct access to the fronts in $\mathcal{F}$. So, for direct access of the fronts in $\mathcal{F}$, the pointers to the fronts in $\mathcal{F}$ are stored in an array. So, the space required in this case is $\mathcal{O}(N)$. Hence, the space complexity of DCNS-SS is $\mathcal{O}(1)$ and the space complexity of DCNS-BS is $\mathcal{O}(N)$. The space complexity of DCNS-SS-WS and DCNS-BS-WS is $\mathcal{O}(N)$.

In a nutshell, the specification of our DCNS framework is given in Table 4. The worst and best case time complexities of several approaches along with their space complexity is also given in Table 4. The best case time complexity of our approach is better than all other approaches for $M > 3$. However, the worst case time complexity is the same as most of the other approaches.

4.5. Scope of Parallelism in our DCNS Framework

In this section, we discuss the scope of parallelism in our DCNS framework. The comparison between different solutions in the naive approach can also be performed simultaneously. So, the naive approach also has the parallelism property. The parallel version of fast non-dominated sort [6] is discussed in [29, 30, 31]. Jensen's approach [3] is a divide-and-conquer algorithm so it can also be implemented in a parallel environment [21].

In the first phase of ENS [7], solutions can be sorted based on the first objective using some parallel sorting algorithm like parallel merge sort [32]. In the second phase, a solution can be compared with all the solutions in a particular front simultaneously. Thus, ENS also has the parallelism property. However, when all the solutions are in different fronts, then each front has a single solution. So, when a solution is compared with the other fronts, then it cannot be performed in parallel because each front has a single solution. Hence, when all the solutions are in different fronts, then the second phase of ENS does not have the parallelism property.

In BOS [1], the solutions can be sorted based on the second to M^{th} objective simultaneously. Also, the solutions can be sorted based on each objective using some parallel sorting algorithm like parallel merge sort. Also, while assigning rank to the solutions in BOS, solutions can be ranked based on each objective simultaneously. Also, a solution can be compared with all the solutions which have been assigned the same rank based a particular objective, simultaneously.

In our DCNS framework, the parallelism in the first phase depends on the sorting algorithm used. If merge sort is used, then the first phase has the parallelism property. In the merge sort, all the merge operations at a level can be performed in parallel. However, to achieve a better speedup, the merge operation itself can be implemented in such a way that it also has the parallelism property [32].

In the second phase of our DCNS framework, all the merge operations at the same level are independent of each other. So, they can be performed simultaneously. In our DCNS framework, the merge operation itself has the parallelism property. There can be three ways to achieve parallelism in the second phase which are discussed as follows:

Version-1: Different merge operations at the same level can be performed simultaneously.

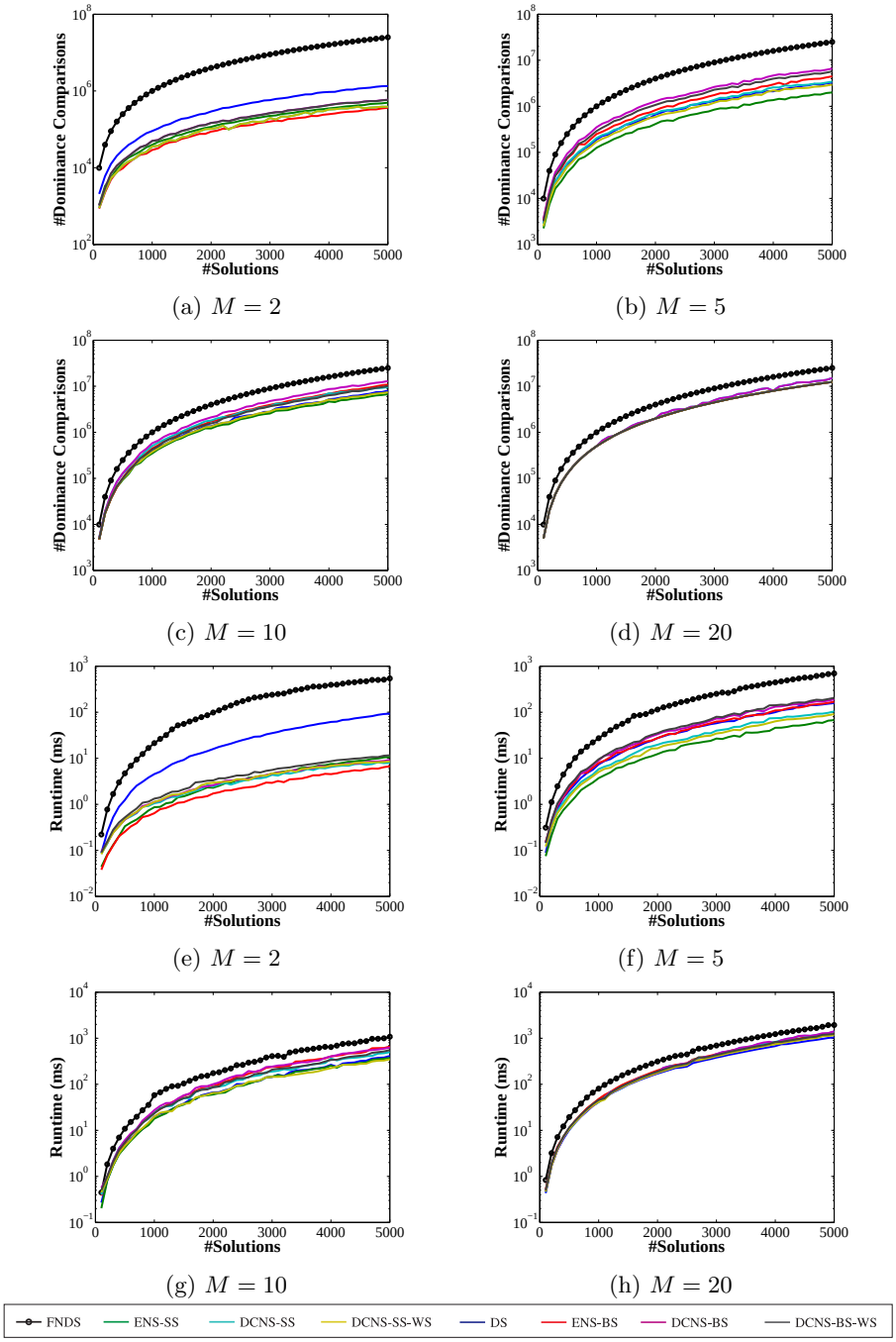
Version-2: Different merge operations at the same level can be performed simultaneously. Also all the solutions of a front $F' \in \mathcal{F}'$ can be compared with the solutions of a front $F \in \mathcal{F}$ simultaneously. However, a solution of a front F' is compared with all the solutions of a front F in a serial manner.

Version-3: Different merge operations at the same level can be performed simultaneously. Also all the solutions of a front $F' \in \mathcal{F}'$ can be compared with the solutions of a front $F \in \mathcal{F}$ simultaneously. Also, a solution of a front F' is compared with all the solutions of a front F simultaneously.

The parallelism in these three different ways are discussed in detail in Appendix B.

5. Experimental Analysis

In this section, we compare the performance of our proposed approaches namely DCNS-SS, DCNS-BS, DCNS-SS-WS and DCNS-BS-WS with four state-of-the-art non-dominated sorting approaches: fast non-dominated sort (FNDS) [6], deductive sort (DS) [10], ENS-SS [7] and ENS-BS [7].



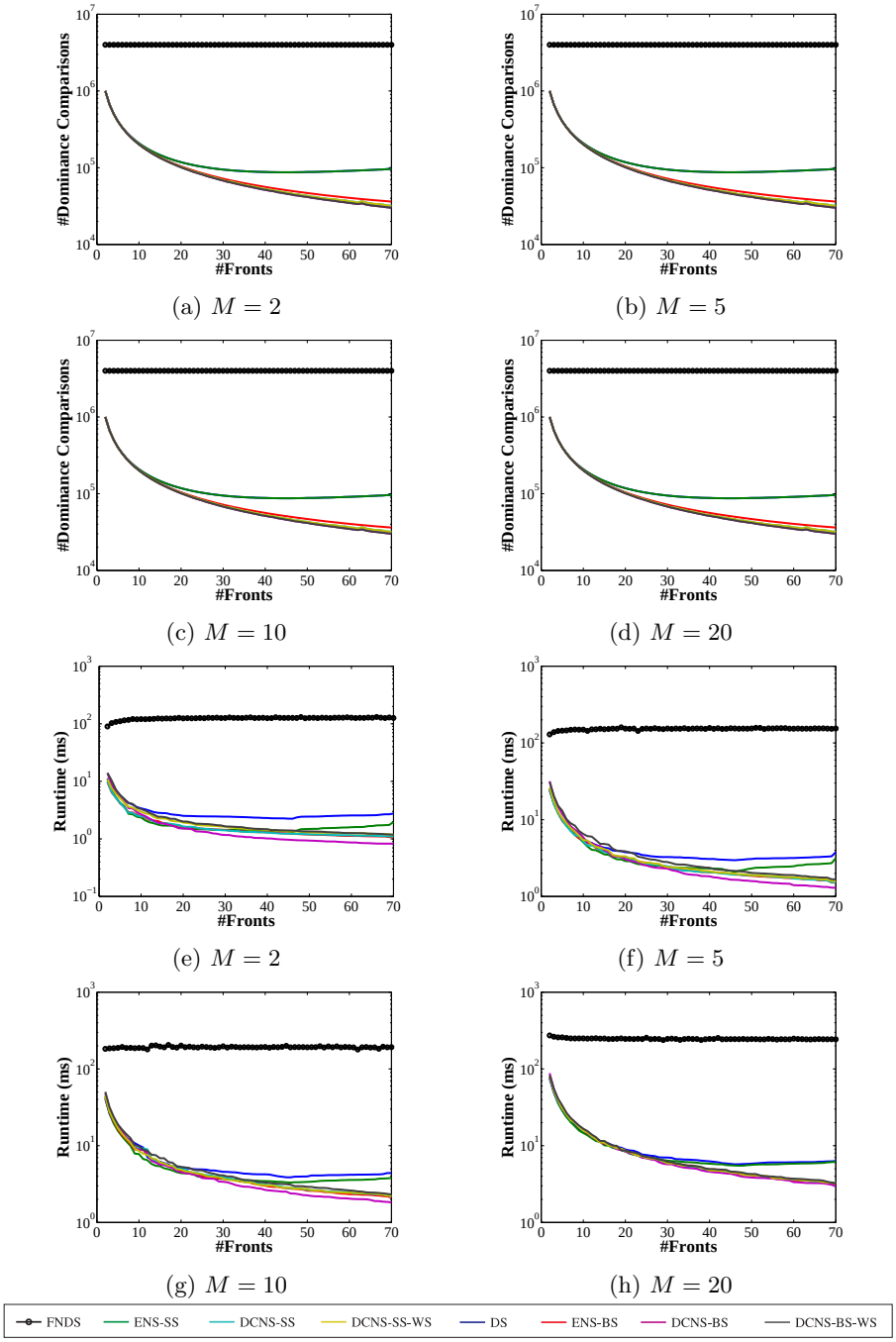
(a) $M = 2$ (b) $M = 5$

(c) $M = 10$ (d) $M = 20$

(e) $M = 2$ (f) $M = 5$

(g) $M = 10$ (h) $M = 20$

Figure 4: Performance of different non-dominated sorting approaches in terms of the number of dominance comparisons and running times for the cloud dataset. (a) – (d) show the number of dominance comparisons, whereas (e) – (h) show the running time (in milliseconds).



(a) $M = 2$ (b) $M = 5$

(c) $M = 10$ (d) $M = 20$

(e) $M = 2$ (f) $M = 5$

(g) $M = 10$ (h) $M = 20$

Figure 5: Performance of different non-dominated sorting approaches in terms of the number of dominance comparisons and running times for the fixed front dataset. (a) – (d) show the number of dominance comparisons, whereas (e) – (h) show the running time (in milliseconds).

5.1. Experiments with the Cloud Dataset

For our experiments, we adopted the cloud dataset [7], but we have varied the size of the population from 100 to 5000 with an increment of 100 as done in [7]. Four different objectives – 2, 5, 10 and 20 are considered. The number of dominance comparisons and the running times (in miliseconds) for this setup are shown in Figure 4. For two objectives, DCNS outperforms FNDS, Deductive sort and ENS-SS while it underperforms ENS-BS both in terms of the number of dominance comparisons as well as in terms of running time. For five objectives, ENS-SS outperforms the other approaches in terms of both the number of comparisons and running time. DCNS-SS and DCNS-SS-WS outperform ENS-BS and deductive sort. For ten objectives, DCNS-SS-WS and ENS-SS perform almost the same number of dominance comparisons whereas DCNS-SS-WS takes less time than the other approaches. For twenty objectives, DCNS-SS-WS and DCNS-BS-WS outperform DCNS-SS and DCNS-BS and require almost the same number of comparisons as the ENS-based approaches.

5.2. Experiments with the Fixed Front Dataset

We also adopted the fixed front dataset [7], with a population size of 2000. In this case, we vary the number of fronts from 2 to 20 with an increment of 1 as done in [7]. Four different numbers of objectives – 2, 5, 10 and 20 are considered. The number of dominance comparisons and running times (in miliseconds) for this setup are shown in Figure 5. ENS-SS and deductive sort perform a very similar number of dominance comparisons for sorting. ENS-BS outperforms ENS-SS when the number of fronts increases. As the number of fronts increases, DCNS based approaches outperform ENS-BS with respect to the number of dominance comparisons. From the Figures 5(a) – 5(d), it is clear that the number of dominance comparisons is the same for all the four objectives. This is because the number of fronts as well as the number of solutions inside the front are the same for a population with a different number of objectives. Also, the dominance nature of the solutions among the fronts is the same.

5.3. Experiments using NSGA-II as the Underlying Optimization Technique

We have also evaluated the performance of the proposed non-dominated sorting approaches when those are incorporated in NSGA-II [6]. Benchmark problems DTLZ1, DTLZ2, DTLZ3 and DTLZ4 [33] with 2, 5, 10 and 20 objectives were adopted to assess performance. There is no need to sort the entire population by Fast non-dominated sort and Deductive sort in MOEAs [7]. However, ENS-SS, ENS-BS and DCNS based algorithms require to sort the entire population. While evaluating the performance, this fact is taken into consideration. The number of generations is set to 250 and the population size is set to 200 as done in [7]. Other parameters are kept the same as done in [6]. Table 5 shows the number of dominance comparisons and the running times (in miliseconds) of different sorting approaches when those are embedded in NSGA-II. This table shows that FNDS requires the maximum number of dominance comparisons to sort the population because it compares each solution with other

Table 5: Performance of non-dominated sorting algorithms in terms of the number of dominance comparisons (#dcmp) and running times (in milliseconds) when they are incorporated in NSGA-II for solving DTLZ1, DTLZ2, DTLZ3 and DTLZ4. Best values are marked in **boldface**.

Test Problem	**Obj.**	**FNDS**		**DS**		**ENS-SS**		**ENS-BS**		**DCNS-SS**		**DCNS-BS**		**DCNS-SS-WS**		**DCNS-BS-WS**	
		#dcmp	time(ms)	#dcmp	time(ms)	#dcmp	time(ms)	#dcmp	time(ms)	#dcmp	time(ms)	#dcmp	time(ms)	#dcmp	time(ms)	#dcmp	time(ms)
DTLZ1	2	3.99e+7	4.19e+2	1.66e+7	1.36e+2	**1.37e+7**	**1.29e+2**	1.73e+7	2.02e+2	1.40e+7	1.56e+2	1.64e+7	1.88e+2	1.40e+7	1.54e+2	1.62e+7	2.43e+2
	5	3.99e+7	6.67e+2	1.89e+7	3.42e+2	**1.74e+7**	**2.75e+2**	1.83e+7	3.80e+2	1.80e+7	2.82e+2	1.86e+7	2.97e+2	1.75e+7	**2.75e+2**	1.81e+7	3.78e+2
	10	3.99e+7	1.23e+3	1.95e+7	6.23e+2	**1.89e+7**	5.10e+2	1.90e+7	5.89e+2	1.93e+7	4.93e+2	1.93e+7	5.23e+2	1.90e+7	**4.86e+2**	1.90e+7	6.09e+2
	20	3.99e+7	1.95e+3	1.97e+7	1.00e+3	**1.96e+7**	7.83e+2	**1.96e+7**	8.90e+2	**1.96e+7**	**7.62e+2**	**1.96e+7**	7.77e+2	**1.96e+7**	7.73e+2	**1.96e+7**	8.82e+2
DTLZ2	2	3.99e+7	3.76e+2	1.88e+7	1.37e+2	**1.14e+7**	**1.09e+2**	1.63e+7	1.90e+2	1.18e+7	1.36e+2	1.52e+7	1.83e+2	1.18e+7	1.33e+2	1.51e+7	2.26e+2
	5	3.99e+7	7.62e+2	1.94e+7	3.79e+2	**1.89e+7**	3.27e+2	**1.89e+7**	4.00e+2	**1.89e+7**	**3.15e+2**	**1.89e+7**	3.35e+2	**1.89e+7**	3.23e+2	**1.89e+7**	4.28e+2
	10	3.99e+7	1.20e+3	1.97e+7	6.36e+2	**1.94e+7**	5.04e+2	**1.94e+7**	6.00e+2	**1.94e+7**	**4.97e+2**	**1.94e+7**	5.17e+2	**1.94e+7**	5.03e+2	**1.94e+7**	6.10e+2
	20	3.99e+7	1.99e+3	1.98e+7	1.07e+3	**1.97e+7**	8.26e+2	**1.97e+7**	8.87e+2	**1.97e+7**	**7.76e+2**	**1.97e+7**	8.10e+2	**1.97e+7**	7.94e+2	**1.97e+7**	8.81e+2
DTLZ3	2	3.99e+7	3.94e+2	1.71e+7	1.34e+2	**1.13e+7**	**1.19e+2**	1.69e+7	1.98e+2	1.19e+7	1.40e+2	1.52e+7	1.78e+2	1.19e+7	1.39e+2	1.50e+7	2.26e+2
	5	3.99e+7	7.76e+2	1.92e+7	3.75e+2	**1.83e+7**	3.16e+2	1.85e+7	3.80e+2	1.87e+7	**3.05e+2**	1.89e+7	3.22e+2	1.84e+7	2.96e+2	1.85e+7	4.04e+2
	10	3.99e+7	1.22e+3	1.96e+7	6.36e+2	**1.91e+7**	5.03e+2	1.92e+7	5.95e+2	1.93e+7	**4.94e+2**	1.93e+7	5.23e+2	**1.91e+7**	5.12e+2	1.92e+7	6.33e+2
	20	3.99e+7	2.11e+3	1.97e+7	1.12e+3	**1.96e+7**	7.93e+2	**1.96e+7**	9.34e+2	**1.96e+7**	**7.84e+2**	**1.96e+7**	8.10e+2	**1.96e+7**	7.99e+2	**1.96e+7**	9.31e+2
DTLZ4	2	3.99e+7	4.26e+2	1.55e+7	1.34e+2	**8.97e+6**	**9.40e+1**	1.34e+7	1.60e+2	9.33e+6	1.30e+2	1.24e+7	1.50e+2	9.33e+6	1.28e+2	1.23e+7	1.98e+2
	5	3.99e+7	7.71e+2	1.92e+7	3.59e+2	**1.77e+7**	2.83e+2	1.82e+7	3.39e+2	1.80e+7	**2.82e+2**	1.83e+7	3.13e+2	1.78e+7	2.85e+2	1.81e+7	3.84e+2
	10	3.99e+7	1.30e+3	1.96e+7	6.87e+2	**1.92e+7**	**5.30e+2**	**1.92e+7**	6.31e+2	**1.92e+7**	5.40e+2	**1.92e+7**	5.71e+2	**1.92e+7**	5.53e+2	**1.92e+7**	6.75e+2
	20	3.99e+7	2.52e+3	1.97e+7	1.38e+3	**1.95e+7**	1.02e+3	**1.95e+7**	1.15e+3	**1.95e+7**	**1.01e+3**	**1.95e+7**	1.05e+3	**1.95e+7**	1.06e+3	**1.95e+7**	1.19e+3

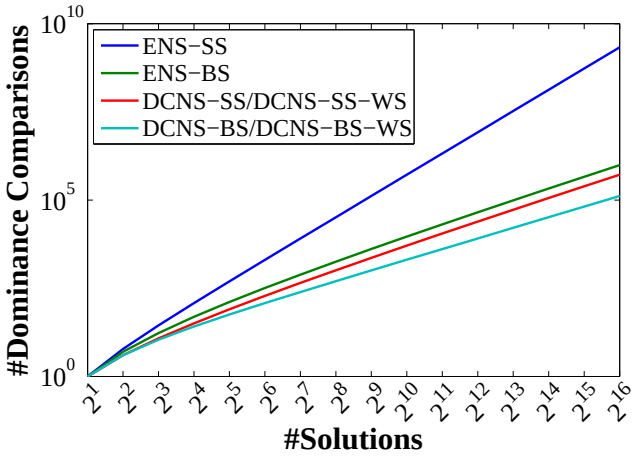


Figure 6: Number of dominance comparisons performed by different non-dominated sorting approaches when all the solutions are in separate fronts. The number of dominance comparisons performed by DCNS-SS and DCNS-SS-WS are the same. Also, the number of dominance comparisons performed by DCNS-BS and DCNS-BS-WS are the same.

solutions. The time taken by FNDS is also maximum. With an increase in the number of objectives, the number of dominance comparisons performed by both the DCNS-based approaches and the ENS-based approaches is almost the same.

5.4. Discussion & Analysis

The worst case time complexity of the DCNS framewrok is $\mathcal{O}(MN^2)$, whereas the best case time complexity is $\mathcal{O}(N \log N + MN)$. The same worst case time complexity is reported by various other approaches [1, 6, 7, 10, 12, 15, 16]. The obtained best case time complexity is better than the best case time complexity of most of the existing approaches for $M > 3$. A lower bound on the problem of identifying the non-dominated set is presented by Kung *et al.* [4]. According to this, the processing time is bounded from below by $\mathcal{O}(N \log N)$ [3]. Jensen *et al.* [3] claimed that it is trivial to see that this bound must also hold for non-dominated sorting. Jensen *et al.* [3] were able to show that this bound holds for $M = 2$ but not for $M \geq 3$. We have obtained the best case time complexity as $\mathcal{O}(N \log N + MN)$ for a general M ($M \geq 3$ also). Thus, we are able to reach the lower bound in the best case of our approach when $M = \mathcal{O}(\log N)$. The assumption $M = \mathcal{O}(\log N)$ seems to be valid, because M is generally very low as compared to N. However, the upper bound of the DCNS framework is still $\mathcal{O}(MN^2)$. In the DCNS framework, we can have parallelism in all the scenarios of the solutions. This framework is very generic and some of the existing approaches like BOS [1] can also exploit parallelism using this framework.

Figure 6 shows the number of dominance comparisons when all the solutions are in separate fronts. Here, the number of solutions is $N = 2^i (1 \leq i \leq 16)$. In this case, the DCNS-based approaches perform a lower number of dominance

Table 6: Number of dominance comparisons when $N = 2^a (a \geq 1)$ solutions are equally divided into $K = 2^b (1 \leq b \leq a)$ fronts.

(a) Each solution in a front is dominated by all the solutions in its preceding front.

$$
\begin{aligned}
\Gamma_{\text{ENS-SS}} &= \sum_{i=1}^{2^b} (i-1)2^{a-b}.1 + \sum_{i=1}^{2^b} \frac{2^{a-b}(2^{a-b}-1)}{2} \\
&= \frac{1}{2}N\left(\frac{N}{K} + K - 2\right) \\
\Gamma_{\text{ENS-BS}} &= \sum_{i=1}^{2^b} \lceil \log i \rceil 2^{a-b}.1 + \sum_{i=1}^{2^b} \frac{2^{a-b}(2^{a-b}-1)}{2} \\
&= \frac{1}{2}N\left(\frac{N}{K} - 1\right) + N\log K - \frac{N}{K}(K-1) \\
\Gamma_{\text{SS}} = \Gamma_{\text{SS-WS}} &= \sum_{i=1}^{a-b} \frac{N}{2^i} 2^{i-1} 2^{i-1} + \sum_{i=a-b+1}^{a} \frac{N}{2^i} 2^{i-(a-b+1)} 2^{a-b} \\
&= \frac{1}{2}N\left(\frac{N}{K} - 1\right) + \frac{1}{2}N\log K \\
\Gamma_{\text{BS}} = \Gamma_{\text{BS-WS}} &= \sum_{i=1}^{a-b} \frac{N}{2^i} 2^{i-1} 2^{i-1} + \sum_{i=a-b+1}^{a} \frac{N}{2^i} \lceil \log(2^{i-(a-b+1)} + 1) \rceil 2^{a-b} \\
&= \frac{1}{2}N\left(\frac{N}{K} - 1\right) + \frac{N}{K}\left(2K - (\log K + 2)\right)
\end{aligned}
$$

(b) Each solution in a front is dominated by only one solution in its preceding front.

$$
\begin{aligned}
\Gamma_{\text{ENS-SS}} &= \sum_{i=1}^{2^b} (i-1)2^{a-b}2^{a-b} + \sum_{i=1}^{2^b} \frac{2^{a-b}(2^{a-b}-1)}{2} \\
&= \frac{1}{2}N(N-1) \\
\Gamma_{\text{ENS-BS}} &= \sum_{i=1}^{2^b} \lceil \log i \rceil 2^{a-b}2^{a-b} + \sum_{i=1}^{2^b} \frac{2^{a-b}(2^{a-b}-1)}{2} \\
&= \frac{N^2}{K}\left(\log K - \frac{1}{2}\right) + \frac{N^2}{K^2} - \frac{1}{2}N \\
\Gamma_{\text{SS}} = \Gamma_{\text{SS-WS}} &= \sum_{i=1}^{a-b} \frac{N}{2^i} 2^{i-1} 2^{i-1} + \sum_{i=a-b+1}^{a} \frac{N}{2^i} 2^{i-(a-b+1)} 2^{a-b} 2^{a-b} \\
&= \frac{N^2}{2K}\left(\log K + 1\right) - \frac{1}{2}N \\
\Gamma_{\text{BS}} = \Gamma_{\text{BS-WS}} &= \sum_{i=1}^{a-b} \frac{N}{2^i} 2^{i-1} 2^{i-1} + \sum_{i=a-b+1}^{a} \frac{N}{2^i} \lceil \log(2^{i-(a-b+1)} + 1) \rceil 2^{a-b} 2^{a-b} \\
&= \frac{N^2}{K}\left(\frac{5}{2} - \frac{\log K + 2}{K}\right) - \frac{1}{2}N
\end{aligned}
$$

comparisons than those of the ENS-based approaches. DCNS-BS and DCNS-BS-WS perform the same number of dominance comparisons. Similarly, DCNS-SS and DCNS-SS-WS perform the same number of dominance comparisons. DCNS-BS and DCNS-BS-WS require a lower number of dominance comparisons than other approaches.

Let us assume that N solutions are equally divided into K fronts. Thus, each front has N/K solutions. Consider a situation, where each solution in a front is dominated by all the solutions in its preceding front. This shows the behavior of the fixed front dataset. Consider another situation, where each solution in a front is dominated by only one solution in its preceding front. Let the number of solutions $N = 2^a, a \geq 1$ and the number of fronts be $K = 2^b, 1 \leq b \leq a$. Thus, each front has 2^{a-b} solutions. Let the number of dominance comparisons performed by ENS-SS be denoted by $\Gamma_{\text{ENS-SS}}$ and the number of dominance comparisons performed by ENS-BS be denoted by $\Gamma_{\text{ENS-BS}}$. In this case, the number of dominance comparisons performed by different approaches in the first situation is given in Table 6(a), whereas for the second situation, it is given in Table 6(b). These tables show that the number of dominance comparisons

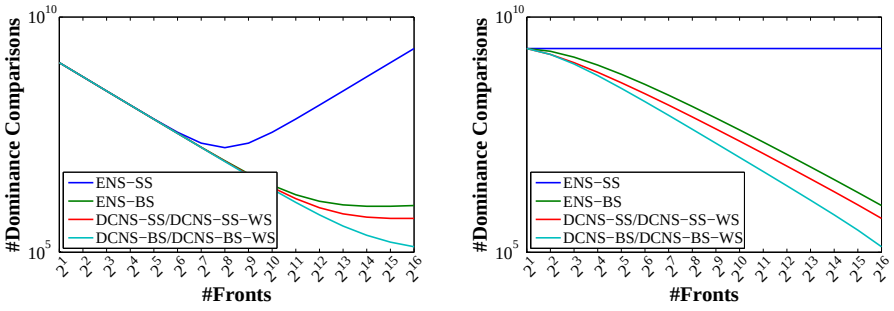


Figure 7: Number of dominance comparisons performed by different non-dominated sorting approaches. (a) Solutions are equally divided into fronts such that each solution in a front is dominated by all the solutions in its preceding front. (b) Solutions are equally divided into fronts such that each solution in a front is dominated by only one solution in its preceding front. The number of dominance comparisons performed by DCNS-SS and DCNS-SS-WS are the same. Also, the number of dominance comparisons performed by DCNS-BS and DCNS-BS-WS are the same.

performed by the DCNS-based approaches is less than that of the other two approaches. Let $N = 2^{16}$ be the number of solutions. The number of fronts $K = 2^b, 1 \leq b \leq 16$. In this case, the number of dominance comparisons in the first and the second situations are shown in Figures 7(a) and 7(b), respectively. In the first situation, for ENS-SS, the minimum number of dominance comparisons is attained at $K = 2^8$ because its best case occurs when $K = \sqrt{N} = \sqrt{2^{16}} = 2^8$. For other approaches, the number of dominance comparisons decreases when the number of fronts increases because the best cases of ENS-BS and the DCNS-based approaches occur when $K = N$. In the second situation, the number of dominance comparisons performed by ENS-SS remains fixed irrespective of the number of fronts. However, the number of dominance comparisons decreases with an increase in the number of fronts for the ENS-BS and DCNS based approaches. In both situations the DCNS-based approaches perform a lower number of dominance comparisons than the ENS-based approaches.

6. Conclusions and Future Work

In this paper, a framework for non-dominated sorting named DCNS is presented. Initially, the solutions are sorted based on the objectives and then solutions are allocated to different fronts. A total of four different versions of the DCNS framework are developed by varying the search type and space requirements. We have theoretically shown that the worst case time complexity of the framework is $\mathcal{O}(MN^2)$ which is the same as that of many existing approaches. The best case time complexity of two of our approaches (DCNS-BS and DCNS-BS-WS) is $\mathcal{O}(N \log N + MN)$ which is better than the best case time complexities of many other existing approaches. The lower bound of non-

dominated sorting as discussed in [3, 4] is also obtained in the best case of our approaches. However, the upper bound remains as $\mathcal{O}(MN^2)$.

BOS [1] can be generalized in the future to handle duplicate solutions by retaining its comparison set concept. T-ENS [15] can also be generalized in the future. The proposed framework has the parallelism property, so in the future we would like to implement this framework in a parallel environment and would like to observe how much speedup can be obtained in different scenarios. Theoretical speedup can also be calculated. It would be interesting to combine the proposed framework with BOS to further reduce the number of dominance comparisons.

References

[1] 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.

[2] 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.

[3] 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.

[4] H.-T. Kung, F. Luccio, F. P. Preparata, On finding the maxima of a set of vectors, Journal of the ACM (JACM) 22 (4) (1975) 469–476.

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

[6] 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.

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

[8] 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.

[9] 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.

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

[11] 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.

[12] H. Wang, X. Yao, Corner Sort for Pareto-Based Many-Objective Optimization, IEEE Transactions on Cybernetics 44 (1) (2014) 92–102.

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

[14] C. Bao, L. Xu, E. D. Goodman, L. Cao, A Novel Non-Dominated Sorting Algorithm for Evolutionary Multi-Objective Optimization, Journal of Computational Science 23 (2017) 31–43.

[15] 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.

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

[17] 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.

[18] 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.

[19] P. C. Roy, K. Deb, M. M. Islam, An efficient nondominated sorting algorithm for large number of fronts, IEEE Transactions on Cybernetics.

[20] A. Jaszkiewicz, T. Lust, Nd-tree-based update: a fast algorithm for the dynamic non-dominance problem, IEEE Transactions on Evolutionary Computation.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

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

[27] 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.

[28] J. W. J. Williams, Algorithm-232-heapsort (1964).

[29] C. Smutnicki, J. Rudy, D. Zelazny, Very fast non-dominated sorting, Decision Making in Manufacturing and Services 8 (1-2) (2014) 13–23.

[30] S. Gupta, G. Tan, A Scalable Parallel Implementation of Evolutionary Algorithms for Multi-Objective Optimization on GPUs, in: 2015 IEEE Congress on Evolutionary Computation (CEC'2015), IEEE Press, Sendai, Japan, 2015, pp. 1567–1574, ISBN: 978-1-4799-7492-4.

[31] G. Ortega, E. Filatovas, E. M. Garzon, L. G. Casado, Non-Dominated Sorting Procedure for Pareto Dominance Ranking on Multicore CPU and/or GPU, Journal of Global Optimization 69 (3) (2017) 607–627.

[32] T. H. C. and Charles E. Leiserson and Ronald L. Rivest and Clifford Stein, Introduction to Algorithms, MIT press, Cambridge, Massachusetts, USA, 2009, ISBN: 978-0-262-03384-8.

[33] K. Deb, L. Thiele, M. Laumanns, E. Zitzler, Scalable Test Problems for Evolutionary Multiobjective Optimization, in: Evolutionary Multiobjective Optimization. Theoretical Advances and Applications, Springer, USA, 2005, pp. 105–145.

Appendix A. Effect of Υ and Ψ on the number of dominance comparisons

In this section, we obtain the number of dominance comparisons when the solutions from a front F' are inserted into a set of fronts $\mathcal{F}$ using Υ and Ψ. For this purpose, Example 5 of the paper is considered where $\mathcal{F} = \{F_1, F_2\}$, $F_1 = \{sol_1, sol_2, sol_3, sol_4\}$ and $F_2 = \{sol_5, sol_6, sol_7, sol_8\}$. The front $F' = \{sol_9, sol_{10}, sol_{11}, sol_{12}, sol_{13}, sol_{14}, sol_{15}, sol_{16}\}$. The set of solutions in $\mathcal{F}$ to which each solution of front F' is compared for insertion in $\mathcal{F}$ without using Υ or Ψ is given in Table A.7.

Table A.7: Set of solutions in $\mathcal{F}$ to which each solution in front F' is compared before being inserted in $\mathcal{F}$ when neither Υ nor Ψ is used.

Inserted solution	**Compared solutions**
sol_9	$sol_1, sol_2, sol_3, sol_4, sol_5, sol_6, sol_7, sol_8$
sol_{10}	$sol_1, sol_2, sol_3, sol_4$
sol_{11}	$sol_1, sol_2, sol_3, sol_4, \mathbf{sol_{10}}, sol_5, sol_6, sol_7, sol_8, \mathbf{sol_9}$
sol_{12}	$sol_1, sol_2, sol_3, sol_4, \mathbf{sol_{10}}$
sol_{13}	$sol_1, sol_2, sol_3, sol_4, \mathbf{sol_{10}}, \mathbf{sol_{12}}, sol_5, sol_6, sol_7, sol_8, \mathbf{sol_9}, \mathbf{sol_{11}}$
sol_{14}	$sol_1, sol_2, sol_3, sol_4, \mathbf{sol_{10}}, \mathbf{sol_{12}}$
sol_{15}	$sol_1, sol_2, sol_3, sol_4, \mathbf{sol_{10}}, \mathbf{sol_{12}}, \mathbf{sol_{14}}, sol_5, sol_6, sol_7, sol_8, \mathbf{sol_9}, \mathbf{sol_{11}}, \mathbf{sol_{13}}$
sol_{16}	$sol_1, sol_2, sol_3, sol_4, \mathbf{sol_{10}}, \mathbf{sol_{12}}, \mathbf{sol_{14}}$

Appendix A.1. Insertion using Υ

We discuss how the solutions of front F' are inserted in $\mathcal{F}$ using Υ. Initially, $\Upsilon = \{\Upsilon_{\text{fIndex}}, \Upsilon_{\text{nSol}}\} = \{0, 0\}$.

Insert sol_9: When sol_9 is inserted into $\mathcal{F}$, it is compared with all the solutions of $\mathcal{F}$ and inserted in F_2. The updated value of $\Upsilon = \{2, 4\}$ because sol_9 is inserted in the **second** front F_2 ($\Upsilon_{\text{fIndex}} = 2$) and has been compared with **four** solutions ($\Upsilon_{\text{sSol}} = 4$) in F_2.

Insert sol_{10}: When sol_{10} is inserted into $\mathcal{F}$, it is compared with all the solutions of front F_1 and inserted in F_1. The updated value of $\Upsilon = \{1, 4\}$ because sol_{10} is inserted in the **first** front F_1 ($\Upsilon_{\text{fIndex}} = 1$) and has been compared with

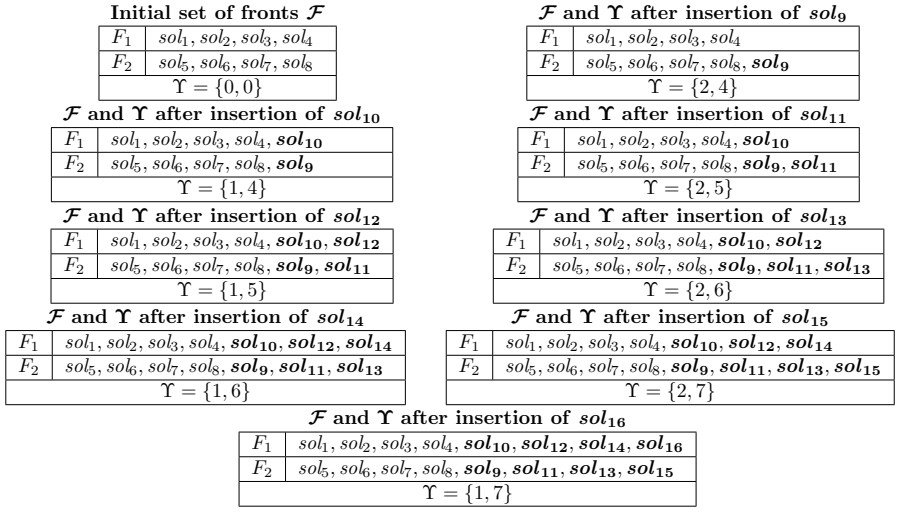


Figure A.8: Insertion of all the solutions of front F' in $\mathcal{F}$.

Table A.8: Set of solutions in $\mathcal{F}$ to which each solution in F' is compared before being inserted in $\mathcal{F}$ using Υ.

Inserted solution	**Compared solutions**
sol_9	$sol_1, sol_2, sol_3, sol_4, sol_5, sol_6, sol_7, sol_8$
sol_{10}	$sol_1, sol_2, sol_3, sol_4$
sol_{11}	$sol_1, sol_2, sol_3, sol_4, sol_5, sol_6, sol_7, sol_8, \mathbf{sol_9}$
sol_{12}	$sol_1, sol_2, sol_3, sol_4, \mathbf{sol_{10}}$
sol_{13}	$sol_1, sol_2, sol_3, sol_4, \mathbf{sol_{10}}, sol_5, sol_6, sol_7, sol_8, \mathbf{sol_9}, \mathbf{sol_{11}}$
sol_{14}	$sol_1, sol_2, sol_3, sol_4, \mathbf{sol_{10}}, \mathbf{sol_{12}}$
sol_{15}	$sol_1, sol_2, sol_3, sol_4, \mathbf{sol_{10}}, \mathbf{sol_{12}}, sol_5, sol_6, sol_7, sol_8, \mathbf{sol_9}, \mathbf{sol_{11}}, \mathbf{sol_{13}}$
sol_{16}	$sol_1, sol_2, sol_3, sol_4, \mathbf{sol_{10}}, \mathbf{sol_{12}}, \mathbf{sol_{14}}$

four solutions ($\Upsilon_{\text{sSol}} = 4$) in F_1.

Insert sol_{11}: At this point of time, front F_1 has five solutions as sol_{10} has already been inserted into F_1. As the value of $\Upsilon = \{1, 4\}$ so when sol_{11} starts comparison with the solutions of front F_1, it is compared with the initial **four** ($\Upsilon_{\text{fIndex}} = 1$ and $\Upsilon_{\text{nSol}} = 4$) solutions, instead of five. Solution sol_{11} is compared with all the five solutions in front F_2. After the insertion of sol_{11} in front F_2, the updated value of $\Upsilon = \{2, 5\}$ because sol_{11} is inserted in the **second** front F_2 ($\Upsilon_{\text{fIndex}} = 2$) and has been compared with **five** solutions ($\Upsilon_{\text{sSol}} = 5$) in F_2.

In a similar way, the remaining solutions from front F' will be inserted in $\mathcal{F}$. Figure A.8 shows the updated $\mathcal{F}$ after insertion of each of the solutions of F'. This figure also shows the value of Υ after insertion of each of the solutions from F' in $\mathcal{F}$. The set of solutions to which each of the solutions of F' is compared for insertion in $\mathcal{F}$ using Υ is given in Table A.8.

Table A.9: Set of solutions in $\mathcal{F}$ to which each solution in F' is compared before being inserted in $\mathcal{F}$ using Ψ.

Inserted solution	**Compared solutions**
sol_9	$sol_1, sol_2, sol_3, sol_4, sol_5, sol_6, sol_7, sol_8$
sol_{10}	$sol_1, sol_2, sol_3, sol_4$
sol_{11}	$sol_1, sol_2, sol_3, sol_4, sol_5, sol_6, sol_7, sol_8$
sol_{12}	$sol_1, sol_2, sol_3, sol_4$
sol_{13}	$sol_1, sol_2, sol_3, sol_4, sol_5, sol_6, sol_7, sol_8$
sol_{14}	$sol_1, sol_2, sol_3, sol_4$
sol_{15}	$sol_1, sol_2, sol_3, sol_4, sol_5, sol_6, sol_7, sol_8$
sol_{16}	$sol_1, sol_2, sol_3, sol_4$

Table A.10: Number of dominance comparisons when the solutions from front F' are inserted in $\mathcal{F}$ without using Υ or Ψ, using Υ and using Ψ.

Solution	**Simple**	**Use of Υ**	**Use of Ψ**
sol_9	8	8	8
sol_{10}	4	4	4
sol_{11}	10	9	8
sol_{12}	5	5	4
sol_{13}	12	11	8
sol_{14}	6	6	4
sol_{15}	14	13	8
sol_{16}	7	7	4
Total dominance comparisons	**66**	**63**	**48**

Appendix A.2. Insertion using Ψ

We discuss how the solutions of front F' are inserted in $\mathcal{F}$ using Ψ. As there are two fronts in $\mathcal{F}$, so the cardinality of Ψ will be 2. $\Psi[1]$ stores the cardinality of front F_1 and $\Psi[2]$ stores the cardinality of front F_2. Thus, $\Psi[1] = |F_1| = 4$ and $\Psi[2] = |F_2| = 4$.

Insert sol_9: When sol_9 is inserted into $\mathcal{F}$, it is compared with the initial $\Psi[1]$ solutions of F_1 and the initial $\Psi[2]$ solutions of F_2 and inserted in F_2.

Insert sol_{10}: When sol_{10} is inserted into $\mathcal{F}$, it is compared with the initial $\Psi[1]$ solutions of F_1 and inserted in F_1.

Insert sol_{11}: When sol_{11} is inserted into $\mathcal{F}$, it is compared with the initial $\Psi[1]$ solutions of F_1 and the initial $\Psi[2]$ solutions of F_2 and inserted in F_2.

In a similar way, the remaining solutions from front F' will be inserted in $\mathcal{F}$. The set of solutions to which each of the solutions of front F' is compared for insertion in $\mathcal{F}$ using Ψ is given in Table A.9. Table A.10 shows the number of dominance comparisons when the solutions from front F' are inserted in $\mathcal{F}$ without using Υ or Ψ, using Υ and using Ψ. From this table, it is clear that the number of dominance comparisons is the minimum when Ψ is considered.

Appendix B. Scope of Parallelism

In this section, we thoroughly discuss the scope of parallelism in the proposed approach. As the approach has two phases, we discuss the parallelism in both of them. Let us assume that the number of solutions be N and the number of objectives be M.

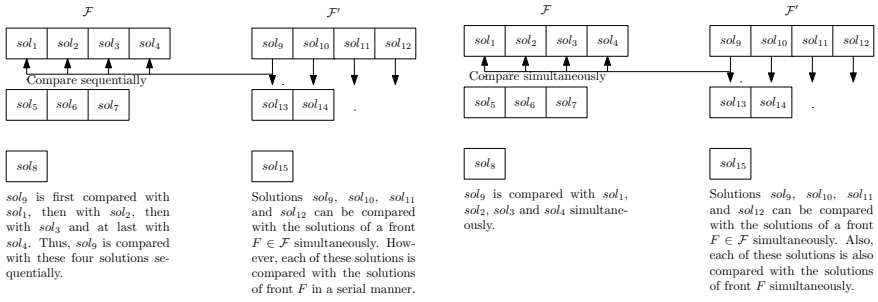
Our approach is very much similar to merge sort [32]. In merge sort, parallelism can be achieved in different ways. The simplest way is to perform all the merge operations at the same level simultaneously. In this manner, the time complexity of the parallel version of merge sort becomes $\mathcal{O}(N)$ for N numbers which is an $\mathcal{O}(\log N)$ times improvement over the serial version. As suggested in [32], to improve the time complexity of parallel merge sort further, the merge operation itself can also be preformed in a parallel manner. If the merge operation is performed in a parallel manner along with performing different merge operations at the same level simultaneously, then the time complexity of the parallel version of merge sort becomes $\mathcal{O}(\log^3 N)$ which is an $\mathcal{O}(N/\log^2 N)$ times improvement over the serial version.

Parallelism in the first phase:. There is parallelism in the first phase of the proposed approach if some parallel algorithm for sorting can be used such as parallel merge sort [32]. The worst case time complexity of the first phase is $\mathcal{O}(MN \log N)$ and the best case time complexity is $\mathcal{O}(N \log N)$. Using parallel merge sort where all the merge operations at the same level are performed simultaneously, the worst case time complexity becomes $\mathcal{O}(MN)$ and the best case time complexity becomes $\mathcal{O}(N)$. This time complexity can be improved if the merge operation can itself be implemented in a parallel manner along with performing different merge operations at a level simultaneously as discussed in [32]. In this manner, the worst case time complexity becomes $\mathcal{O}(M \log^3 N)$ and the best case time complexity becomes $\mathcal{O}(\log^3 N)$.

Parallelism in the second phase:. In the second phase, the basic operation is a merge operation which merges two sets of fronts. Let the first set of fronts be denoted as $\mathcal{F}$ and the second set of fronts be denoted as $\mathcal{F}'$. The parallelism can be achieved in a different manner. Here, we discuss the parallelism in the second phase of the proposed approach in three different manners. Similar to merge sort, in the first version of our parallel approach, we have performed all the merge operations simultaneously. In the other two variants, we have focused on the parallel implementation of the merge operation along with performing different merge operations at the same level simultaneously. Now, we discuss the parallelism in the second phase of the proposed approach:

1. Version-1:. All the merge operations at the same level are performed simultaneously.

2. Version-2:. All the merge operations at the same level are performed simultaneously. Also, the position of the different solutions of a front $F' \in \mathcal{F}'$ are identified in $\mathcal{F}$ simultaneously and they are added to their respective front



(a) Parallel version-2 (b) Parallel version-3

Figure B.9: Parallel version

in a sequential manner to avoid a write collision (or critical section). Here, a solution of front F' is compared with the solutions of a front $F \in \mathcal{F}$ sequentially.

Let $\mathcal{F} = \{\{sol_1, sol_2, sol_3, sol_4\}, \{sol_5, sol_6, sol_7\}, \{sol_8\}\}$ and $\mathcal{F}' = \{\{sol_9, sol_{10}, sol_{11}, sol_{12}\}, \{sol_{13}, sol_{14}\}, \{sol_{15}\}\}$. These two sets of fronts are shown in Figure B.9(a). In parallel version-2 of the proposed approach, solutions $sol_9, sol_{10}, sol_{11}, sol_{12}$ are compared with the solutions of the first front in $\mathcal{F}$ simultaneously. However, each of these solutions is compared with the solutions of the first front in $\mathcal{F}$ sequentially.

3. Version-3:. All the merge operations at the same level are performed simultaneously. Also, the position of the different solutions of a front $F' \in \mathcal{F}'$ is identified in $\mathcal{F}$ simultaneously and they are added to their respective front in a sequential manner to avoid a write collision (or critical section). Also, each solution of front F' is compared with the solutions of a front $F \in \mathcal{F}$ simultaneously.

Consider the same set of fronts as considered in parallel version-2. These two sets of fronts are also shown in Figure B.9(b). In parallel version-3 of the proposed approach, solutions $sol_9, sol_{10}, sol_{11}, sol_{12}$ are compared with the solutions of the first front in $\mathcal{F}$ simultaneously. Also, each of these solutions is compared with the solutions of the first front in $\mathcal{F}$ simultaneously.

Now, we obtain the recurrence relation of the serial and parallel version of the proposed approach in three different scenarios and obtain the maximum theoretical speedup. These three scenarios are discussed in [1, 2, 7, 10].

Appendix C. All the Solutions are in a Single Front

We establish the recurrence relation for our non-dominated sorting approach when all the solutions are in a single front and obtain the maximum theoretical speedup.

Appendix C.1. Serial Algorithm

The recurrence relation of the serial version of the non-dominated sorting approach is given by Eq. (C.1). In this recurrence relation, the first part $M\left(N/2\right)^2$

corresponds to the time to obtain the dominance relationship of the solutions in $\mathcal{F}'$ with the solutions in $\mathcal{F}$. The second part $N/2$ corresponds to the time to add the solutions in $\mathcal{F}'$ to $\mathcal{F}$.

$$T_1(N,M) = \begin{cases} M+1 & \text{if } N=2 \\ 2T_1\left(N/2, M\right) + M\left(N/2\right)^2 + N/2 & \text{otherwise} \end{cases} \tag{C.1}$$

The recurrence relation in Eq. (C.1) is solved using Eq. (C.2).

$$\begin{aligned} T_1(N,M) &= \left[M\left(N/2\right)^2 + N/2\right] + 2\left[M\left(N/4\right)^2 + N/4\right] + \ldots + N/2\left[M\left(N/N\right)^2 + N/N\right] \\ &= 1/2MN(N-1) + 1/2N\log N \end{aligned} \tag{C.2}$$

Appendix C.2. Parallel Algorithm: Version-1

The recurrence relation of parallel version-1 of the non-dominated sorting approach is given by Eq. (C.3).

$$T_\infty(N,M) = \begin{cases} M+1 & \text{if } N=2 \\ T_\infty\left(N/2, M\right) + M\left(N/2\right)^2 + N/2 & \text{otherwise} \end{cases} \tag{C.3}$$

The recurrence relation in Eq. (C.3) is solved using Eq. (C.4).

$$\begin{aligned} T_\infty(N,M) &= \left[M\left(N/2\right)^2 + N/2\right] + \left[M\left(N/4\right)^2 + N/4\right] + \ldots + \left[M\left(N/N\right)^2 + N/N\right] \\ &= 1/3M(N^2-1) + (N-1) \end{aligned} \tag{C.4}$$

The speedup using parallel version-1 of the non-dominated sorting approach is obtained in Eq. (C.5).

$$\text{Speedup} = \frac{T_1(N,M)}{T_\infty(N,M)} = \frac{1/2MN(N-1) + 1/2N\log N}{1/3M(N^2-1) + (N-1)} \equiv 3/2 \tag{C.5}$$

Appendix C.3. Parallel Algorithm: Version-2

The recurrence relation of parallel version-2 of the non-dominated sorting approach is given by Eq. (C.6).

$$T_\infty(N,M) = \begin{cases} M+1 & \text{if } N=2 \\ T_\infty\left(N/2, M\right) + M\left(N/2\right) + N/2 & \text{otherwise} \end{cases} \tag{C.6}$$

The recurrence relation in Eq. (C.6) is solved using Eq. (C.7).

$$\begin{aligned} T_\infty(N,M) &= \left[M\left(N/2\right) + N/2\right] + \left[M\left(N/4\right) + N/4\right] + \ldots + \left[M\left(N/N\right) + N/N\right] \\ &= M(N-1) + (N-1) \end{aligned} \tag{C.7}$$

The speedup using parallel version-2 of the non-dominated sorting approach is obtained in Eq. (C.8).

$$\text{Speedup} = \frac{T_1(N,M)}{T_\infty(N,M)} = \frac{1/2MN(N-1) + 1/2N\log N}{M(N-1) + (N-1)} \equiv 1/2N \tag{C.8}$$

Appendix C.4. Parallel Algorithm: Version-3

The recurrence relation of parallel version-3 of the non-dominated sorting approach is given by Eq. (C.9). In this recurrence relation, the first part, $M + \log(N/2)$, corresponds to the time to obtain the dominance relationship of the solutions in $\mathcal{F}'$ with the solutions in $\mathcal{F}$. The second part, $N/2$, corresponds to the time to add the solutions in $\mathcal{F}'$ to $\mathcal{F}$.

$$T_\infty(N, M) = \begin{cases} M + 1 & \text{if } N = 2 \\ T_\infty(N/2, M) + M + \log(N/2) + N/2 & \text{otherwise} \end{cases} \tag{C.9}$$

The recurrence relation in Eq. (C.9) is solved using Eq. (C.10).

$$\begin{aligned} T_\infty(N, M) = & [M + \log(N/2) + N/2] + [M + \log(N/4) + N/4] + \ldots + \\ & [M + \log(N/N) + N/N] \\ = & M \log N + 1/2(2N + \log^2 N - \log N - 2) \end{aligned} \tag{C.10}$$

The speedup using parallel version-3 of the non-dominated sorting approach is obtained in Eq. (C.11).

$$\text{Speedup} = \frac{T_1(N, M)}{T_\infty(N, M)} = \frac{1/2MN(N-1) + 1/2N \log N}{M \log N + 1/2(2N + \log^2 N - \log N - 2)} \equiv 1/2MN \tag{C.11}$$

Appendix D. All the Solutions are in Different Fronts

We establish the recurrence relation for the non-dominated sorting approach when all the solutions are in different fronts and obtain the maximum theoretical speedup. In this case, all the three parallel versions will perform the same because each front has a single solution. As the number of fronts is more than one so sequential and binary search based approaches perform differently.

Appendix D.1. Sequential Search based Approach

Here, the recurrence relation is established using a sequential search based strategy.

Appendix D.1.1. Serial Algorithm

The recurrence relation of the serial version of the non-dominated sorting approach using a sequential search based strategy is given by Eq. (D.1). This recurrence relation is solved using Eq. (D.2).

$$T_1(N, M) = \begin{cases} M + 1 & \text{if } N = 2 \\ 2T_1(N/2, M) + M(N/2) + N/2 & \text{otherwise} \end{cases} \tag{D.1}$$

$$\begin{aligned} T_1(N, M) = & [M(N/2) + N/2] + 2[M(N/4) + N/4] + \ldots + N/2[M(N/N) + N/N] \\ = & 1/2MN \log N + 1/2N \log N \end{aligned} \tag{D.2}$$

Appendix D.1.2. Parallel Algorithm

The recurrence relation of the parallel version of the non-dominated sorting approach using a sequential search based strategy is given by Eq. (D.3). This recurrence relation is solved using Eq. (D.4).

$$T_\infty(N,M) = \begin{cases} M+1 & \text{if } N=2 \\ T_\infty\left({}^N\!/_2, M\right) + M\left({}^N\!/_2\right) + {}^N\!/_2 & \text{otherwise} \end{cases} \tag{D.3}$$

$$\begin{aligned} T_\infty(N,M) &= \left[M\left({}^N\!/_2\right) + {}^N\!/_2\right] + \left[M\left({}^N\!/_4\right) + {}^N\!/_4\right] + \ldots + \left[M\left({}^N\!/_N\right) + {}^N\!/_N\right] \\ &= M(N-1) + (N-1) \end{aligned} \tag{D.4}$$

The speedup using the parallel version of the non-dominated sorting approach is obtained in Eq. (D.5).

$$\text{Speedup} = \frac{T_1(N,M)}{T_\infty(N,M)} = \frac{{}^1\!/_2 MN\log N + {}^1\!/_2 N\log N}{M(N-1)+(N-1)} \equiv {}^1\!/_2 \log N \tag{D.5}$$

Appendix D.2. Binary Search based Approach

Here, the recurrence relation is established using a binary search based strategy.

Appendix D.2.1. Serial Algorithm

The recurrence relation of the serial version of the non-dominated sorting approach using a binary search based strategy is given by Eq. (D.6). This recurrence relation is solved using Eq. (D.7).

$$T_1(N,M) = \begin{cases} M+1 & \text{if } N=2 \\ 2T_1\left({}^N\!/_2, M\right) + M\lceil \log\left({}^N\!/_2 + 1\right)\rceil + {}^N\!/_2 & \text{otherwise} \end{cases} \tag{D.6}$$

$$\begin{aligned} T_1(N,M) &= \left[M\lceil \log\left({}^N\!/_2+1\right)\rceil + {}^N\!/_2\right] + 2\left[M\lceil \log\left({}^N\!/_4+1\right)\rceil + {}^N\!/_4\right] + \ldots + \\ &\quad {}^N\!/_2\left[M\lceil \log\left({}^N\!/_N+1\right)\rceil + {}^N\!/_N\right] \\ &= M(2N - \log N - 2) + {}^1\!/_2 N\log N \end{aligned} \tag{D.7}$$

Appendix D.2.2. Parallel Algorithm

The recurrence relation of the parallel version of the non-dominated sorting approach using a binary search based strategy is given by Eq. (D.8). This recurrence relation is solved using Eq. (D.9).

$$T_\infty(N,M) = \begin{cases} M+1 & \text{if } N=2 \\ T_\infty\left({}^N\!/_2, M\right) + M\lceil \log\left({}^N\!/_2 + 1\right)\rceil + {}^N\!/_2 & \text{otherwise} \end{cases} \tag{D.8}$$

$$\begin{aligned} T_\infty(N,M) &= \left[M\lceil \log\left({}^N\!/_2+1\right)\rceil + {}^N\!/_2\right] + \left[M\lceil \log\left({}^N\!/_4+1\right)\rceil + {}^N\!/_4\right] + \ldots + \\ &\quad \left[M\lceil \log\left({}^N\!/_N+1\right)\rceil + {}^N\!/_N\right] \\ &= {}^1\!/_2 M(\log^2 N + \log N) + (N-1) \end{aligned} \tag{D.9}$$

The speedup using the parallel version of the non-dominated sorting approach is obtained in Eq. (D.10).

$$\text{Speedup}=\frac{T_1(N,M)}{T_\infty(N,M)}=\frac{M(2N-\log N-2)+{}^1\!/\!_2 N\log N}{{}^1\!/\!_2 M(\log^2 N+\log N)+(N-1)}\equiv {}^1\!/\!_2\log N \qquad \text{(D.10)}$$

Appendix E. N Solutions are Equally Divided into $\sqrt{N}$ Fronts

We establish the recurrence relation for the non-dominated sorting approach when N solutions are equally divided in $\sqrt{N}$ fronts such that each solution in a front dominates all the solutions in its succeeding front. We also obtain the maximum theoretical speedup. As the number of fronts is more than one, the sequential and the binary search based approaches perform differently. Let us consider $N'=\sqrt{N}$.

Appendix E.1. Sequential Search based Approach

Here, the recurrence relation is established using a sequential search based strategy.

Appendix E.1.1. Serial Algorithm

The recurrence relation of the serial version of the non-dominated sorting approach using a sequential search based strategy is given by Eq. (E.1). Here, till the ${}^{\mathcal{L}}\!/\!_{2^{th}}$ level, in the merge operation, the solutions are non-dominated. After the merge operation at the ${}^{\mathcal{L}}\!/\!_{2^{th}}$ is finished, each set of fronts has a single front which contains $\sqrt{N}$ solutions. So, the first part of Eq. (E.1) corresponds to the process of non-dominated sorting till the ${}^{\mathcal{L}}\!/\!_{2^{th}}$ level. After the ${}^{\mathcal{L}}\!/\!_{2^{th}}$ level, each set of fronts has a higher number of fronts. Whenever a merge operation is performed, all the fronts in $\mathcal{F}'$ have a lower dominance than that of the fronts in $\mathcal{F}$. Only the solutions of the first front in $\mathcal{F}'$ are compared with the solutons of $\mathcal{F}$. The solutions of the remaining fronts in $\mathcal{F}'$ are added directly to $\mathcal{F}$ because of the dominance relationship as discussed in [2]. So, the second part of Eq. (E.1) corresponds to the process of non-dominated sorting after the ${}^{\mathcal{L}}\!/\!_{2^{th}}$ level.

$$T_1(N,M)=N'T_{11}(\sqrt{N},M)+\underbrace{T_{12}(N,M)}_{N>N'} \qquad \text{(E.1)}$$

$$T_{11}(\sqrt{N},M)=\begin{cases} M+1 & \text{if } \sqrt{N}=2 \\ 2T_{11}\left({}^{\sqrt{N}}\!/\!_2,M\right)+M\left({}^{\sqrt{N}}\!/\!_2\right)^2+{}^{\sqrt{N}}\!/\!_2 & \text{otherwise}\end{cases} \qquad \text{(E.2)}$$

$$\underbrace{T_{12}(N,M)}_{N>N'}=\begin{cases} MN'+N' & \text{if } N=2N' \\ 2T_{12}\left({}^N\!/\!_2,M\right)+M\left({}^N\!/\!_{2N'}N'\right)+\left({}^N\!/\!_{2N'}\right)N' & \text{otherwise}\end{cases}$$

$$=\begin{cases} MN'+N' & \text{if } N=2N' \\ 2T_{12}\left({}^N\!/\!_2,M\right)+M\left({}^N\!/\!_2\right)+{}^N\!/\!_2 & \text{otherwise}\end{cases} \qquad \text{(E.3)}$$

$$T_{11}(\sqrt{N},M)=\left[M\left(\sqrt{N}/2\right)^2+\sqrt{N}/2\right]+2\left[M\left(\sqrt{N}/4\right)^2+\sqrt{N}/4\right]+\ldots+$$
$$\sqrt{N}/2\left[M\left(\sqrt{N}/\sqrt{N}\right)^2+\sqrt{N}/\sqrt{N}\right]$$
$$=1/2M\sqrt{N}(\sqrt{N}-1)+1/4\sqrt{N}\log N \quad \text{(E.4)}$$
$$T_{12}(N,M)=[M\left(N/2\right)+N/2]+2\left[M\left(N/4\right)+N/4\right]+\ldots+N'/2\left[M\left(N/N'\right)+N/N'\right]$$
$$=1/4MN\log N+1/4N\log N \quad \text{(E.5)}$$

The solution to the recurrence relation in Eq. (E.1) is obtained in Eq. (E.6).

$$T_1(N,M)=N'T_{11}(\sqrt{N},M)+T_{12}(N,M)=\sqrt{N}T_{11}(\sqrt{N},M)+T_{12}(N,M)$$
$$=\sqrt{N}\left[1/2M\sqrt{N}(\sqrt{N}-1)+1/4\sqrt{N}\log N\right]+1/4MN\log N+1/4N\log N$$
$$=1/4MN(2\sqrt{N}+\log N-2)+1/2N\log N \quad \text{(E.6)}$$

Appendix E.1.2. Parallel Algorithm: Version-1

The recurrence relation of parallel version-1 using a sequential search based strategy is given by Eq. (E.7).

$$T_\infty(N,M)=N'T_{\infty 1}(\sqrt{N},M)+\underbrace{T_{\infty 2}(N,M)}_{N>N'} \quad \text{(E.7)}$$

$$T_{\infty 1}(\sqrt{N},M)=\begin{cases}M+1 & \text{if } \sqrt{N}=2\\ T_{\infty 1}\left(\sqrt{N}/2,M\right)+M\left(\sqrt{N}/2\right)^2+\sqrt{N}/2 & \text{otherwise}\end{cases} \quad \text{(E.8)}$$

$$\underbrace{T_{\infty 2}(N,M)}_{N>N'}=\begin{cases}MN'+N' & \text{if } N=2N'\\ T_{\infty 2}\left(N/2,M\right)+M\left(N/2N'N'\right)+\left(N/2N'\right)N' & \text{otherwise}\end{cases}$$
$$=\begin{cases}MN'+N' & \text{if } N=2N'\\ T_{\infty 2}\left(N/2,M\right)+M\left(N/2\right)+N/2 & \text{otherwise}\end{cases} \quad \text{(E.9)}$$

$$T_{\infty 1}(\sqrt{N},M)=\left[M\left(\sqrt{N}/2\right)^2+\sqrt{N}/2\right]+\left[M\left(\sqrt{N}/4\right)^2+\sqrt{N}/4\right]+\ldots+$$
$$\left[M\left(\sqrt{N}/\sqrt{N}\right)^2+\sqrt{N}/\sqrt{N}\right]$$
$$=1/3M(N-1)+(\sqrt{N}-1) \quad \text{(E.10)}$$
$$T_{\infty 2}(N,M)=[M\left(N/2\right)+N/2]+[M\left(N/4\right)+N/4]+\ldots+[M\left(N/N'\right)+N/N']$$
$$=M\sqrt{N}(\sqrt{N}-1)+\sqrt{N}(\sqrt{N}-1) \quad \text{(E.11)}$$

The solution to the recurrence relation in Eq. (E.7) is obtained in Eq. (E.12).

$$T_\infty(N,M)=N'T_{\infty 1}(\sqrt{N},M)+T_{\infty 2}(N,M)=\sqrt{N}T_{\infty 1}(\sqrt{N},M)+T_{\infty 2}(N,M)$$
$$=\sqrt{N}\left[1/3M(N-1)+(\sqrt{N}-1)\right]+M\sqrt{N}(\sqrt{N}-1)+\sqrt{N}(\sqrt{N}-1)$$
$$=1/3M\sqrt{N}(N+3\sqrt{N}-4)+2\sqrt{N}(\sqrt{N}-1) \quad \text{(E.12)}$$

The speedup using parallel version-1 is obtained in Eq. (E.13).

$$\text{Speedup}=\frac{T_1(N,M)}{T_\infty(N,M)}=\frac{{}^1\!/\!_4 MN(2\sqrt{N}+\log N-2)+{}^1\!/\!_2 N\log N}{{}^1\!/\!_3 M\sqrt{N}(N+3\sqrt{N}-4)+2\sqrt{N}(\sqrt{N}-1)}\equiv{}^3\!/\!_2 \quad \text{(E.13)}$$

Appendix E.1.3. Parallel Algorithm: Version-2

The recurrence relation of parallel version-2 using a sequential search based strategy is given by Eq. (E.14).

$$T_\infty(N,M)=N'T_{\infty 1}(\sqrt{N},M)+\underbrace{T_{\infty 2}(N,M)}_{N>N'} \quad \text{(E.14)}$$

$$T_{\infty 1}(\sqrt{N},M)=\begin{cases} M+1 & \text{if } \sqrt{N}=2 \\ T_{\infty 1}\left({}^{\sqrt{N}}\!/\!_2,M\right)+M\left({}^{\sqrt{N}}\!/\!_2\right)+{}^{\sqrt{N}}\!/\!_2 & \text{otherwise}\end{cases} \quad \text{(E.15)}$$

$$\begin{aligned}\underbrace{T_{\infty 2}(N,M)}_{N>N'}&=\begin{cases} M+N' & \text{if } N=2N' \\ T_{\infty 2}\left({}^N\!/\!_2,M\right)+M\left({}^N\!/\!_{2N'}\right)+\left({}^N\!/\!_{2N'}\right)N' & \text{otherwise}\end{cases} \\ &=\begin{cases} M+N' & \text{if } N=2N' \\ T_{\infty 2}\left({}^N\!/\!_2,M\right)+M\left({}^N\!/\!_{2N'}\right)+{}^N\!/\!_2 & \text{otherwise}\end{cases}\end{aligned} \quad \text{(E.16)}$$

$$\begin{aligned}T_{\infty 1}(\sqrt{N},M)=&\left[M\left({}^{\sqrt{N}}\!/\!_2\right)+{}^{\sqrt{N}}\!/\!_2\right]+\left[M\left({}^{\sqrt{N}}\!/\!_4\right)+{}^{\sqrt{N}}\!/\!_4\right]+\ldots+ \\ &\left[M\left({}^{\sqrt{N}}\!/\!_{\sqrt{N}}\right)+{}^{\sqrt{N}}\!/\!_{\sqrt{N}}\right]=M(\sqrt{N}-1)+(\sqrt{N}-1)\end{aligned} \quad \text{(E.17)}$$

$$\begin{aligned}T_{\infty 2}(N,M)=&\left[M\left({}^N\!/\!_{2N'}\right)+{}^N\!/\!_2\right]+\left[M\left({}^N\!/\!_{4N'}\right)+{}^N\!/\!_4\right]+\ldots+ \\ &\left[M\left({}^N\!/\!_{N'N'}\right)+{}^N\!/\!_{N'}\right]=M(\sqrt{N}-1)+\sqrt{N}(\sqrt{N}-1)\end{aligned} \quad \text{(E.18)}$$

The solution to the recurrence relation in Eq. (E.14) is obtained in Eq. (E.19).

$$\begin{aligned}T_\infty(N,M)&=N'T_{\infty 1}(\sqrt{N},M)+T_{\infty 2}(N,M)=\sqrt{N}T_{\infty 1}(\sqrt{N},M)+T_{\infty 2}(N,M) \\ &=\sqrt{N}\left[M(\sqrt{N}-1)+(\sqrt{N}-1)\right]+M(\sqrt{N}-1)+\sqrt{N}(\sqrt{N}-1) \\ &=M(N-1)+2\sqrt{N}(\sqrt{N}-1)\end{aligned} \quad \text{(E.19)}$$

The speedup using parallel version-2 is obtained in Eq. (E.20).

$$\text{Speedup}=\frac{T_1(N,M)}{T_\infty(N,M)}=\frac{{}^1\!/\!_4 MN(2\sqrt{N}+\log N-2)+{}^1\!/\!_2 N\log N}{M(N-1)+2\sqrt{N}(\sqrt{N}-1)}\equiv{}^1\!/\!_2\sqrt{N} \quad \text{(E.20)}$$

Appendix E.1.4. Parallel Algorithm: Version-3

The recurrence relation of parallel version-3 using a sequential search based strategy is given by Eq. (E.21).

$$T_\infty(N,M)=N'T_{\infty 1}(\sqrt{N},M)+\underbrace{T_{\infty 2}(N,M)}_{N>N'} \quad \text{(E.21)}$$

$$T_{\infty 1}(\sqrt{N},M)=\begin{cases}M+1 & \text{if } \sqrt{N}=2\\ T_{\infty 1}\left(\sqrt{N}/2,M\right)+M+\log\left(\sqrt{N}/2\right)+\sqrt{N}/2 & \text{otherwise}\end{cases} \quad \text{(E.22)}$$

$$\underbrace{T_{\infty 2}(N,M)}_{N>N'}=\begin{cases}M+\log N'+N' & \text{if } N=2N'\\ T_{\infty 2}\left(N/2,M\right)+N/2N'\left(M+\log N'\right)+\left(N/2N'\right)N' & \text{otherwise}\end{cases}$$

$$=\begin{cases}M+\log N'+N' & \text{if } N=2N'\\ T_{\infty 2}\left(N/2,M\right)+N/2N'\left(M+\log N'\right)+N/2 & \text{otherwise}\end{cases} \quad \text{(E.23)}$$

$$\begin{aligned}T_{\infty 1}(\sqrt{N},M)=&\left[M+\log\left(\sqrt{N}/2\right)+\sqrt{N}/2\right]+\left[M+\log\left(\sqrt{N}/4\right)+\sqrt{N}/4\right]+\ldots+\\ &\left[M+\log\left(\sqrt{N}/\sqrt{N}\right)+\sqrt{N}/\sqrt{N}\right]\\ =&1/2M\log N+1/8(8\sqrt{N}+\log^2 N-2\log N-8) \quad \text{(E.24)}\\ T_{\infty 2}(N,M)=&\left[N/2N'\left(M+\log N'\right)+N/2\right]+\left[N/4N'\left(M+\log N'\right)+N/4\right]+\ldots+\\ &\left[N/N'N'\left(M+\log N'\right)+N/N'\right]\\ =&M(\sqrt{N}-1)+1/2(2N+\sqrt{N}\log N-2\sqrt{N}-\log N) \quad \text{(E.25)}\end{aligned}$$

The solution to the recurrence relation in Eq. (E.21) is obtained in Eq. (E.26).

$$\begin{aligned}T_{\infty}(N,M)=&\sqrt{N}T_{\infty 1}(\sqrt{N},M)+T_{\infty 2}(N,M)=\sqrt{N}T_{\infty 1}(\sqrt{N},M)+T_{\infty 2}(N,M)\\ =&\sqrt{N}\left[1/2M\log N+1/8(8\sqrt{N}+\log^2 N-2\log N-8)\right]+\\ &M(\sqrt{N}-1)+1/2(2N+\sqrt{N}\log N-2\sqrt{N}-\log N)\\ =&1/2M(\sqrt{N}\log N+2\sqrt{N}-2)+\\ &1/8(16N+\sqrt{N}\log^2 N+2\sqrt{N}\log N-16\sqrt{N}-4\log N) \quad \text{(E.26)}\end{aligned}$$

The speedup using parallel version-3 is obtained in Eq. (E.27).

$$\text{Speedup}=\frac{T_1(N,M)}{T_{\infty}(N,M)}\equiv 1/4M\sqrt{N}\equiv 1/4M\sqrt{N} \quad \text{(E.27)}$$

Appendix E.2. Binary Search based Approach

Here, the recurrence relation is established using a binary search based strategy.

Appendix E.2.1. Serial Algorithm

The recurrence relation of the serial version of the non-dominated sorting approach using a binary search based strategy is given by Eq. (E.28).

$$T_1(N,M)=N'T_{11}(\sqrt{N},M)+\underbrace{T_{12}(N,M)}_{N>N'} \quad \text{(E.28)}$$

$$T_{11}(\sqrt{N},M)=\begin{cases}M{+}1 & \text{if } \sqrt{N}{=}2\\ 2T_{11}\left({}^{\sqrt{N}}/_{2},M\right)+M\left({}^{\sqrt{N}}/_{2}\right)^2+{}^{\sqrt{N}}/_{2} & \text{otherwise}\end{cases} \tag{E.29}$$

$$\begin{aligned}\underbrace{T_{12}(N,M)}_{N>N'}&=\begin{cases}MN'{+}N' & \text{if } N{=}2N'\\ 2T_{12}\left({}^{N}/_{2},M\right)+M\left(\lceil\log\left({}^{N}/_{2N'}{+}1\right)\rceil N'\right)+\left({}^{N}/_{2N'}\right)N' & \text{otherwise}\end{cases}\\ &=\begin{cases}MN'{+}N' & \text{if } N{=}2N'\\ 2T_{12}\left({}^{N}/_{2},M\right)+M\left(\lceil\log\left({}^{N}/_{2N'}{+}1\right)\rceil N'\right)+{}^{N}/_{2} & \text{otherwise}\end{cases}\end{aligned} \tag{E.30}$$

$$\begin{aligned}T_{11}(\sqrt{N},M)=&\left[M\left({}^{\sqrt{N}}/_{2}\right)^2+{}^{\sqrt{N}}/_{2}\right]+2\left[M\left({}^{\sqrt{N}}/_{4}\right)^2+{}^{\sqrt{N}}/_{4}\right]+\ldots+\\ &{}^{\sqrt{N}}/_{2}\left[M\left({}^{\sqrt{N}}/_{\sqrt{N}}\right)^2+{}^{\sqrt{N}}/_{\sqrt{N}}\right]\\ =&{}^{1}/_{2}M\sqrt{N}(\sqrt{N}{-}1)+{}^{1}/_{4}\sqrt{N}\log N\end{aligned} \tag{E.31}$$

$$\begin{aligned}T_{12}(N,M)=&\left[M\left(\lceil\log\left({}^{N}/_{2N'}{+}1\right)\rceil N'\right)+{}^{N}/_{2}\right]+\\ &2\left[M\left(\lceil\log\left({}^{N}/_{4N'}{+}1\right)\rceil N'\right)+{}^{N}/_{4}\right]+\ldots+\\ &{}^{N'}/_{2}\left[M\left(\lceil\log\left({}^{N}/_{N'N'}{+}1\right)\rceil N'\right)+{}^{N}/_{N'}\right]\\ =&{}^{1}/_{2}M\sqrt{N}(4\sqrt{N}-\log N{-}4)+{}^{1}/_{4}N\log N\end{aligned} \tag{E.32}$$

The solution to the recurrence relation in Eq. (E.28) is obtained in Eq. (E.33).

$$\begin{aligned}T_1(N,M)=&N'T_{11}(\sqrt{N},M)+T_{12}(N,M)=\sqrt{N}T_{11}(\sqrt{N},M)+T_{12}(N,M)\\ =&\sqrt{N}\left[{}^{1}/_{2}M\sqrt{N}(\sqrt{N}{-}1)+{}^{1}/_{4}\sqrt{N}\log N\right]+\\ &{}^{1}/_{2}M\sqrt{N}(4\sqrt{N}-\log N{-}4)+{}^{1}/_{4}N\log N\\ =&{}^{1}/_{2}M\sqrt{N}(N{+}3\sqrt{N}-\log N{-}4)+{}^{1}/_{2}N\log N\end{aligned} \tag{E.33}$$

Appendix E.2.2. Parallel Algorithm: Version-1

The recurrence relation of parallel version-1 using a binary search based strategy is given by Eq. (E.34).

$$T_\infty(N,M)=N'T_{\infty 1}(\sqrt{N},M)+\underbrace{T_{\infty 2}(N,M)}_{N>N'} \tag{E.34}$$

$$T_{\infty 1}(\sqrt{N},M)=\begin{cases}M{+}1 & \text{if } \sqrt{N}{=}2\\ T_{\infty 1}\left({}^{\sqrt{N}}/_{2},M\right)+M\left({}^{\sqrt{N}}/_{2}\right)^2+{}^{\sqrt{N}}/_{2} & \text{otherwise}\end{cases} \tag{E.35}$$

$$\underbrace{T_{\infty 2}(N,M)}_{N>\sqrt{N}}=\begin{cases}MN'{+}N' & \text{if } N{=}2N'\\ T_{\infty 2}\left({}^{N}/_{2},M\right)+M\left(\lceil\log\left({}^{N}/_{2N'}{+}1\right)\rceil N'\right)+ & \\ \left({}^{N}/_{2N'}\right)N' & \text{otherwise}\end{cases}$$

$$= \begin{cases} MN'+N' & \text{if } N=2N' \\ T_{\infty 2}\left({}^{N}/_{2}, M\right) + M\left(\lceil \log\left({}^{N}/_{2N'}+1\right)\rceil N'\right) + {}^{N}/_{2} & \text{otherwise} \end{cases} \tag{E.36}$$

$$T_{\infty 1}(\sqrt{N}, M) = \left[M\left({}^{\sqrt{N}}/_{2}\right)^2 + {}^{\sqrt{N}}/_{2}\right] + \left[M\left({}^{\sqrt{N}}/_{4}\right)^2 + {}^{\sqrt{N}}/_{4}\right] + \ldots + \left[M\left({}^{\sqrt{N}}/_{\sqrt{N}}\right)^2 + {}^{\sqrt{N}}/_{\sqrt{N}}\right] = {}^{1}/_{3}M(N-1) + (\sqrt{N}-1) \tag{E.37}$$

$$\begin{aligned} T_{\infty 2}(N, M) &= \left[M\left(\lceil \log\left({}^{N}/_{2N'}+1\right)\rceil N'\right) + {}^{N}/_{2}\right] + \left[M\left(\lceil \log\left({}^{N}/_{4N'}+1\right)\rceil N'\right) + {}^{N}/_{4}\right] \\ &\quad + \ldots + \left[M\left(\lceil \log\left({}^{N}/_{N'N'}+1\right)\rceil N'\right) + {}^{N}/_{N'}\right] \\ &= {}^{1}/_{8}M\sqrt{N}(\log^2 N + 2\log N) + \sqrt{N}(\sqrt{N}-1) \end{aligned} \tag{E.38}$$

The solution to the recurrence relation in Eq. (E.34) is obtained in Eq. (E.39).

$$\begin{aligned} T_{\infty}(N, M) &= N'T_{\infty 1}(\sqrt{N}, M) + T_{\infty 2}(N, M) = \sqrt{N}T_{\infty 1}(\sqrt{N}, M) + T_{\infty 2}(N, M) \\ &= \sqrt{N}\left[{}^{1}/_{3}M\left(N-1\right) + (\sqrt{N}-1)\right] + \\ &\quad {}^{1}/_{8}M\sqrt{N}(\log^2 N + 2\log N) + \sqrt{N}(\sqrt{N}-1) \\ &= {}^{1}/_{24}M\sqrt{N}(8N + 3\log^2 N + 6\log N - 8) + 2\sqrt{N}(N-1) \end{aligned} \tag{E.39}$$

The speedup using parallel version-1 is obtained in Eq. (E.40).

$$\text{Speedup} = \frac{T_1(N, M)}{T_{\infty}(N, M)} = \frac{{}^{1}/_{2}M\sqrt{N}(N + 3\sqrt{N} - \log N - 4) + {}^{1}/_{2}N\log N}{{}^{1}/_{24}M\sqrt{N}(8N + 3\log^2 N + 6\log N - 8) + 2\sqrt{N}(N-1)} \equiv {}^{3}/_{2} \tag{E.40}$$

Appendix E.2.3. Parallel Algorithm: Version-2

The recurrence relation of parallel version-2 using a binary search based strategy is given by Eq. (E.41).

$$T_{\infty}(N, M) = N'T_{\infty 1}(\sqrt{N}, M) + \underbrace{T_{\infty 2}(N, M)}_{N > N'} \tag{E.41}$$

$$T_{\infty 1}(\sqrt{N}, M) = \begin{cases} M+1 & \text{if } \sqrt{N}=2 \\ T_{\infty 1}\left({}^{\sqrt{N}}/_{2}, M\right) + M\left({}^{\sqrt{N}}/_{2}\right) + {}^{\sqrt{N}}/_{2} & \text{otherwise} \end{cases} \tag{E.42}$$

$$\begin{aligned} \underbrace{T_{\infty 2}(N, M)}_{N > \sqrt{N}} &= \begin{cases} M+N' & \text{if } N=2N' \\ T_{\infty 2}\left({}^{N}/_{2}, M\right) + M\left(\lceil \log\left({}^{N}/_{2N'}+1\right)\rceil\right) + \left({}^{N}/_{2N'}\right)N' & \text{otherwise} \end{cases} \\ &= \begin{cases} M+N' & \text{if } N=2N' \\ T_{\infty 2}\left({}^{N}/_{2}, M\right) + M\left(\lceil \log\left({}^{N}/_{2N'}+1\right)\rceil\right) + {}^{N}/_{2} & \text{otherwise} \end{cases} \end{aligned} \tag{E.43}$$

$$T_{\infty 1}(\sqrt{N},M)=\left[M\left(\sqrt{N}/2\right)+\sqrt{N}/2\right]+\left[M\left(\sqrt{N}/4\right)+\sqrt{N}/4\right]+\ldots+\left[M\left(\sqrt{N}/\sqrt{N}\right)+\sqrt{N}/\sqrt{N}\right]=M(\sqrt{N}-1)+(\sqrt{N}-1) \quad \text{(E.44)}$$

$$\begin{aligned}T_{\infty 2}(N,M)&=\left[M\left(\lceil\log\left(N/2N'+1\right)\rceil\right)+N/2\right]+\left[M\left(\lceil\log\left(N/4N'+1\right)\rceil\right)+N/4\right]+\\&\quad\ldots+\left[M\left(\lceil\log\left(N/N'N'+1\right)\rceil\right)+N/N'\right]\\&=1/8M(\log^2 N+2\log N)+\sqrt{N}(\sqrt{N}-1)\end{aligned} \quad \text{(E.45)}$$

The solution to the recurrence relation in Eq. (E.41) is obtained in Eq. (E.46).

$$\begin{aligned}T_\infty(N,M)&=N'T_{\infty 1}(\sqrt{N},M)+T_{\infty 2}(N,M)=\sqrt{N}T_{\infty 1}(\sqrt{N},M)+T_{\infty 2}(N,M)\\&=\sqrt{N}\left[M(\sqrt{N}-1)+(\sqrt{N}-1)\right]+1/8M(\log^2 N+2\log N)+\sqrt{N}(\sqrt{N}-1)\\&=1/8M(8N-8\sqrt{N}+\log^2 N+2\log N)+2\sqrt{N}(\sqrt{N}-1)\end{aligned} \quad \text{(E.46)}$$

The speedup using parallel version-2 is obtained in Eq. (E.47).

$$\text{Speedup}=\frac{T_1(N,M)}{T_\infty(N,M)}\equiv 1/2\sqrt{N} \quad \text{(E.47)}$$

Appendix E.2.4. Parallel Algorithm: Version-3

The recurrence relation of parallel version-3 using a binary search based strategy is given by Eq. (E.48).

$$T_\infty(N,M)=\sqrt{N}T_{\infty 1}(\sqrt{N},M)+\underbrace{T_{\infty 2}(N,M)}_{N>\sqrt{N}} \quad \text{(E.48)}$$

$$T_{\infty 1}(\sqrt{N},M)=\begin{cases}M+1 & \text{if } \sqrt{N}=2\\ T_{\infty 1}\left(\sqrt{N}/2,M\right)+M+\log\left(\sqrt{N}/2\right)+\sqrt{N}/2 & \text{otherwise}\end{cases} \quad \text{(E.49)}$$

$$\begin{aligned}\underbrace{T_{\infty 2}(N,M)}_{N>\sqrt{N}}&=\begin{cases}M+\log N'+N' & \text{if } N=2N'\\ T_{\infty 2}\left(N/2,M\right)+\lceil\log\left(N/2N'+1\right)\rceil\left(M+\log N'\right)+\\ \left(N/2N'\right)N' & \text{otherwise}\end{cases}\\&=\begin{cases}M+\log N'+N' & \text{if } N=2N'\\ T_{\infty 2}\left(N/2,M\right)+\lceil\log\left(N/2N'+1\right)\rceil\left(M+\log N'\right)+\\ N/2 & \text{otherwise}\end{cases}\end{aligned} \quad \text{(E.50)}$$

Table E.11: Maximum theoretical speedup achieved by the parallel version of the second phase of the non-dominated sorting approach in three different scenarios.

Scenario	**Version-1**		**Version-2**		**Version-3**	
	Sequential	**Binary**	**Sequential**	**Binary**	**Sequential**	**Binary**
N solutions in single front	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{1}{2}N$	$\frac{1}{2}N$	$\frac{1}{2}MN$	$\frac{1}{2}MN$
N solutions in N front	$\frac{1}{2}\log N$	$\frac{1}{2}\log N$	$\frac{1}{2}\log N$	$\frac{1}{2}\log N$	$\frac{1}{2}\log N$	$\frac{1}{2}\log N$
N solutions in $\sqrt{N}$ front†	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{1}{2}\sqrt{N}$	$\frac{1}{2}\sqrt{N}$	$\frac{1}{4}M\sqrt{N}$	$\frac{1}{4}M\sqrt{N}$

†: Each solution in a front is dominated by all the solutions in its preceding front.

$$\begin{aligned}T_{\infty 1}(\sqrt{N},M)=&\left[M+\log\left(\sqrt{N}/2\right)+\sqrt{N}/2\right]+\left[M+\log\left(\sqrt{N}/4\right)+\sqrt{N}/4\right]+\ldots+\\&\left[M+\log\left(\sqrt{N}/\sqrt{N}\right)+\sqrt{N}/\sqrt{N}\right]\\=&1/2M\log N+1/8(8\sqrt{N}+\log^2 N-2\log N-8)\end{aligned}\tag{E.51}$$

$$\begin{aligned}T_{\infty 2}(N,M)=&\left[\lceil\log\left(N/2N'+1\right)\rceil\left(M+\log N'\right)+N/2\right]+\\&\left[\lceil\log\left(N/4N'+1\right)\rceil\left(M+\log N'\right)+N/4\right]+\ldots+\\&\left[\lceil\log\left(N/N'N'+1\right)\rceil\left(M+\log N'\right)+N/N'\right]\\=&1/8M(\log^2 N+2\log N)+1/16(16N-16\sqrt{N}+\log^3 N+2\log^2 N)\end{aligned}\tag{E.52}$$

The solution to the recurrence relation in Eq. (E.48) is obtained in Eq. (E.53).

$$\begin{aligned}T_\infty(N,M)=&N'T_{\infty 1}(\sqrt{N},M)+T_{\infty 2}(N,M)=\sqrt{N}T_{\infty 1}(\sqrt{N},M)+T_{\infty 2}(N,M)\\=&\sqrt{N}\left[1/2M\log N+1/8(8\sqrt{N}+\log^2 N-2\log N-8)\right]+\\&1/8M(\log^2 N+2\log N)+1/16(16N-16\sqrt{N}+\log^3 N+2\log^2 N)+\\&\sqrt{N}(\sqrt{N}-1)\\=&1/8M(4\sqrt{N}\log N+\log^2 N+2\log N)+\\&1/16(32N-32\sqrt{N}+2\sqrt{N}\log^2 N-4\sqrt{N}\log N+\log^3 N+2\log^2 N)\end{aligned}\tag{E.53}$$

The speedup using parallel version-3 is obtained in Eq. (E.54).

$$\text{Speedup}=\frac{T_1(N,M)}{T_\infty(N,M)}\equiv 1/4M\sqrt{N}\tag{E.54}$$

Summary of maximum theoretical speedup:. Table E.11 summarizes the maximum theoretical speedup by different parallel versions of the second phase of the non-dominated sorting approach in three different scenarios.

The theoretical speedup of the parallel version of the second phase of the non-dominated sorting approach can be further improved if the dominance relationship between each pair of solutions can be obtained before the merge operations. The dominance relationship between different solutions can be computed in $\mathcal{O}(M)$ time [29]. Table E.12 summarizes the maximum theoretical speedup

achieved by different parallel versions of the second phase of the non-dominated sorting approach in three different scenarios when the dominance relationship between different solutions can be obtained beforehand.

Table E.12: Maximum theoretical speedup achieved by the parallel version of the second phase of the non-dominated sorting approach in three different scenarios when the dominance relationship between different solutions can be obtained beforehand.

Scenario	**Version-1**		**Version-2**		**Version-3**	
	Sequential	**Binary**	**Sequential**	**Binary**	**Sequential**	**Binary**
N solutions in single front	$\frac{3}{2}M$	$\frac{3}{2}M$	$\frac{1}{4}MN$	$\frac{1}{4}MN$	$\frac{1}{2}MN$	$\frac{1}{2}MN$
N solutions in N front	$\frac{1}{4}M\log N$	$\frac{1}{2}\log N$	$\frac{1}{4}M\log N$	$\frac{1}{2}\log N$	$\frac{1}{4}M\log N$	$\frac{1}{2}\log N$
N solutions in $\sqrt{N}$ front†	$\frac{3}{2}M$	$\frac{3}{2}M$	$\frac{1}{6}M\sqrt{N}$	$\frac{1}{6}M\sqrt{N}$	$\frac{1}{4}M\sqrt{N}$	$\frac{1}{4}M\sqrt{N}$

†: Each solution in a front is dominated by all the solutions in its preceding front.