A divide-and-conquer based efficient non-dominated sorting approach

Sumit Mishra a,∗, Sriparna Saha b, Samrat Mondal b, Carlos A. Coello Coello a

ARTICLE INFO

Keywords:
Non-dominated sorting
Dominance relation
Computational complexity
Parallelism

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 $O(N\log N + MN)$ 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.

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 also used in various fields of study such as game theory, economics, computational geometry and databases [1]. Let $P = \{sol_1, sol_2, \ldots, sol_N\}$ be a population of $N$ solutions in $M$-dimensional objective space where $M$ is the number of objectives associated with each solution. A solution $sol_i$ 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:

- $vsol_1, sol_j \in F_k: sol_i \not\prec sol_j$ and $sol_j \not\prec sol_i (1 \leq k \leq K)$
- $vsol_1 \in F_k, \exists sol_j \in F_{k+1}: sol_i \prec sol_j (2 \leq k \leq K)$.

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.

Example 1. Let $P = \{sol_1, sol_2, \ldots, sol_N\}$ be a population of eight solutions in a 2-dimensional objective space as shown in Fig. 1. For a mini-

---

© 2018 Elsevier B.V. All rights reserved.

Please cite this article in press as: S. Mishra, et al., A divide-and-conquer based efficient non-dominated sorting approach, Swarm and Evolutionary Computation (2018), https://doi.org/10.1016/j.sweco.2018.08.011
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 \(\Theta(N \log N + MN)\). 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 \(\Theta(N \log N + MN)\) 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 \(\Theta(N \log N)\). 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 \(\Theta(N \log N)\) for \(M \geq 3\) and also when \(M = \Theta(\log N)\). However, the upper bound on the processing time is still \(\Theta(MN^2)\).

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 \(\Theta(MN^3)\) when all the solutions are in different fronts. However, the best case time complexity is \(\Theta(MN^2)\) when all the solutions are in a single front. The space complexity of this approach is \(\Theta(N)\) [6]. Deb et al. [6] proposed a computationally cost effective approach called fast non-dominated sorting (FNDS) with time complexity \(\Theta(MN^2)\) and space complexity \(\Theta(N^2)\). 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 \(\Theta(N \log N)\) and the space complexity is \(\Theta(MN)\). 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 \(\Theta(MN^2)\) when all the solutions are non-dominated. The best case time complexity of this method is \(\Theta(MN \log N)\). The space complexity of this method is \(\Theta(MN)\). 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 \(\Theta(MN \sqrt{N})\) 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 \(\Theta(MN^2)\) with \(\Theta(N)\) space complexity. The best case time complexity of deductive sort is \(\Theta(MN \sqrt{N})\). Fortin et al. [11] removed the assumption of Jensen’s algorithm, but removing this assumption increases the worst case time complexity to \(\Theta(MN^2)\). However, the average case time complexity remains \(\Theta(N \log M^{-1} N)\).

Corner sort is proposed by Wang et al. [12] with worst case time complexity \(\Theta(MN^2)\). 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 \(\Theta(MN^2)\) time in the worst case. The best case time complexity of ENS-SS and ENS-BS is \(\Theta(N \log N)\) and \(\Theta(MN \log N)\), respectively. The time complexity of non-dominated sorting was proved to be \(\Theta(N \log M^{-1} N)\) by Buzdalov et al. [13]. Bao et al. [14] proposed a Hierarchical Non-dominated Sorting (HNDSS) for non-dominated sorting. HNDSS first sorts the solution based on the first objective, then the solutions are assigned to their respective front. The best case time complexity of HNDSS is \(\Theta(MN \sqrt{N})\) and the worst case time complexity is \(\Theta(MN^2)\) with space complexity \(\Theta(N)\).
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 $O(MN^2)$ and the best case time complexity is $O(MN \log N)$. The space complexity of BOS is $O(MN)$. 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 $O(MN^2)$. The best case time complexity is $O(MN \log N / \log M)$. 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 $O(MN^2)$. The best case time complexity of ENS-NDT is $O(MN \log N)$ when $M > \log N$; otherwise, it is $O(N \log^2 N)$. Few other approaches like [17–20] have been recently proposed for non-dominated sorting.

Some of these approaches [21–27] 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].

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 $F = \{F_1, F_2, \ldots, F_N\}$ be the $N$ sets of fronts where each set of fronts $F_i = \{sol_i\}, 1 \leq i \leq N$. The set of fronts in $F$ at the $i$th position is referred to as $F(i), 1 \leq i \leq N$.

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 $\log_2 N$ levels because of the height of the tree which is $\log_2 N$. At each level, two consecutive sets of fronts are merged. The merge operation is performed using the $\text{MERGE}(F, F')$ procedure which is described in detail in Section 4. The following example illustrates the working flow of the proposed framework.

**Example 2.** Let $\mathcal{P} = \{sol_1, sol_2, \ldots, sol_8\}$ be a population of eight solutions in 2-dimensional objective space as shown in Fig. 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 $\mathcal{L} = \log_8 8 = 3$. Hence, the merge operations are performed at three different levels. The complete procedure is shown in Fig. 2.

### 4. Merge procedure

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

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

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

When a front $F' \in F'$ is inserted into $F$ using either the INSERT() or the INSERT-WS() procedure, then it returns an index value which indicates the index of the front in $F$ having the highest dominance where the solutions of front $F'$ have been inserted. When the next front of $F'$ is inserted into $F$, then solutions of this front start the dominance comparison with front $F_{a+1}$. Each time a front $F' \in F'$ is inserted into $F'$, a is updated accordingly. If the value of $a$ is equal to the cardinality of $F$ 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 $F'$ are inserted into $F$ directly without any dominance comparison with the solutions of $F'$ (see lines 7–9 of Algorithm 2). Next, we discuss the INSERT() and the INSERT-WS() procedures one by one in detail.

#### 4.1. Insert procedure without extra space

The procedure to insert the solutions of a front $F'$ in $F$, which requires constant space is discussed in Algorithm 3. This procedure uses a variable $hif$ (highest front index) to keep track of $a$. $hif$ stores the index of the front in $F$ having the highest dominance in which the solutions of front $F'$ are inserted. As all the solutions of $F'$ are non-dominated.

---

\(^2\) https://github.com/Protek/Best-Order-Sort/.
Eight solutions for sorting

---

Solutions after Pre-sorting

---

Consider each solution as a set of fronts

---

LEVEL-1

---

LEVEL-2

---

LEVEL-3

---

Fig. 2. Working flow of the proposed DCNS framework. ✓ indicates the merge operation between the immediate left and right set of fronts. sol1, ..., sol8 represent that these solutions are non-dominated to each other. The solutions are merged at three different levels.
Algorithm 1 DCNS framework.
Input: $P$: Population in $M$-dimensional space
Output: Non-dominated fronts in sorted order
1: Sort $P$ in ascending order based on the first objective
   $//$ Assign the sorted solutions to $F$
2: for $i \leftarrow 1$ to $N$ do $//$ Consider the solutions in sorted order
3: $F_1 \leftarrow \{P(i)\}$ $//$ Consider a solution as a front
4: $F_2 \leftarrow \{F_1\}$ $//$ Consider a front as a set of fronts
5: $F \leftarrow F \cup \{F_2\}$ $//$ Add set of fronts to $F$
   $//$ Steps of non-dominated sorting
6: for $i \leftarrow 1$ to $[\log_2 N]$ do $//$ For each level
7: for each set of fronts $\mathcal{F} \in F$ do
8: $\mathcal{F}' \leftarrow$ Next set of fronts to $\mathcal{F}$
9: if $\mathcal{F}'$ exists then
10: $\text{MERGE}(\mathcal{F}, \mathcal{F}')$ $//$ Insert all the solutions from $\mathcal{F}'$ to $\mathcal{F}$
11: Delete $\mathcal{F}'$
12: return $F(1)$ $//$ Final non-dominated fronts are in $F(1)$

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

with each other, so, when the solutions of $\mathcal{F}'$ are inserted into $\mathcal{F}$, then the solutions of $\mathcal{F}'$ can add a maximum of one front in $\mathcal{F}$. The solutions of $\mathcal{F}'$ can also be inserted into the existing fronts in $\mathcal{F}$. It may also be possible that some of the solutions of $\mathcal{F}'$ 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 $\mathcal{F}'$.

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

Example 3. Let $\mathcal{F} = \{F_1\}$ be the set of fronts where there is only a single front. $F_1 = \{\text{sol}_1, \text{sol}_2, \ldots, \text{sol}_8\}$. Consider the solutions of a front $\mathcal{F}' = \{\text{sol}_9, \text{sol}_{10}, \ldots, \text{sol}_{16}\}$ which need to be inserted in $\mathcal{F}$. Let all the solutions of front $\mathcal{F}'$ will be inserted in $F_1$. When $\text{sol}_9$ is inserted into $\mathcal{F}'$, it is compared with all the eight solutions in $F_1$. Now, $Y_{\text{index}} = 1$ and $Y_{\text{sol}} = 8$. When $\text{sol}_{10}$ is inserted into $\mathcal{F}'$, it is compared with the first eight (i.e., $Y_{\text{sol}} = 8$) solutions in $F_1$ and not with nine solutions. As $\text{sol}_{10}$ and $\text{sol}_9$ are from front $\mathcal{F}'$ hence both are non-dominated, so there is no need to compare $\text{sol}_{10}$ with $\text{sol}_9$. The remaining solutions of $\mathcal{F}'$ are only compared with the first eight solutions. The use of $\mathcal{Y}$ helps in reducing the number of dominance compar-

Algorithm 3 $\text{INSERT}(\mathcal{F}, \mathcal{F}', \alpha)$.
Input: $\mathcal{F}$: Set of fronts, $\mathcal{F}'$: Front for insertion in $\mathcal{F}$, $\alpha$: Index of the front in $\mathcal{F}$ from where the solutions of $\mathcal{F}'$ start the dominance comparison
Output: $hfi$: Index of the front in $\mathcal{F}$ having the highest dominance in which the solutions of front $\mathcal{F}'$ are inserted
1: $P \leftarrow |\mathcal{F}|$ $//$ Store the cardinality of $\mathcal{F}$
2: $hfi \leftarrow P + 1$ $//$ Initialize $hfi$
3: $Y_{\text{index}} \leftarrow 0$, $Y_{\text{sol}} \leftarrow 0$ $//$ Initialize $\mathcal{Y}$
4: for each solution $\text{sol} \in \mathcal{F}'$ do
5: $\text{INSERT-SS}(\mathcal{F}, \text{sol})$ $//$ Insert $\text{sol}$ in $\mathcal{F}$
6: $\mathcal{F}' \leftarrow \mathcal{F}' \setminus \{\text{sol}\}$ $//$ Delete $\text{sol}$ from $\mathcal{F}'$
7: return $hfi$
isons. Table 1 shows the number of dominance comparisons after insertion of each solution of front \( F' \) in \( F \) considering \( Y \) and without considering \( Y \). This table shows that the number of dominance comparisons is reduced when \( Y \) is used.

The solution \( s_1 \in F' \) can be inserted in \( F \) using a sequential search based strategy or a binary search based strategy as in [7,2].

### 4.1. Sequential search based strategy

The procedure to insert a solution \( s_1 \in F' \) using a sequential search based strategy is given in Algorithm 4. \( s_1 \) is compared with the solutions of each of the fronts in \( F' \) starting from \( F_4 \) to \( F_p \) in a sequential manner.

A set \( Y \) is considered to reduce the unnecessary dominance comparisons. So, we first check whether the previous solution was inserted in the \( pth \) front which is to be explored (line 5). If this condition is true, then \( s_1 \) is compared with the initial \( F_{sol} \) solutions of \( F_p \); otherwise, \( s_1 \) is compared with all the solutions of \( F_p \). If \( s_1 \) is non-dominated with respect to all the solutions of \( F_p \), then \( s_1 \) is inserted in \( F_p \) and \( hfi \) and \( Y \) are updated accordingly. If \( s_1 \) is dominated by any of the solutions in \( F_p \), then \( s_1 \) is compared with the solutions of the next front, i.e., \( F_{p+1} \). If \( s_1 \) is dominated by each of the compared fronts, then \( s_1 \) is inserted into \( F_{p+1} \).

### 4.1.2. Binary search based strategy

The procedure to insert a solution \( s_1 \in F' \) using a binary search based strategy is given in Algorithm 5. As opposed to the sequential search based strategy, \( s_1 \) is not compared with each of the fronts in \( F' \) starting from \( F_4 \) to \( F_p \), but rather only \( \lfloor \log_2(P + a + 1) \rfloor \) 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 0 and \( R \) is set to \( P \). At first, \( s_1 \) 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 \( s_1 \) is compared with the initial \( F_{sol} \) solutions of \( F_{mid} \); otherwise, \( s_1 \) is compared with all the solutions of \( F_{mid} \). If \( s_1 \) 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 \( s_1 \) is inserted in \( F_{mid} \) and \( hfi \) as well as \( Y \) is updated. After this, the process terminates.
- Otherwise, \( (mid < L) \) the root of the left sub-tree is checked.

If \( s_1 \) 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 \( s_1 \) is inserted in \( F_{p+1} \) and the process terminates.
- If \( s_1 \) is dominated by the leaf node, (i.e., \( mid = R \)), then \( s_1 \) is inserted in \( F_{p+1} \) and the process terminates.
- Otherwise, the right of the right sub-tree is checked.

### Example 4

Consider two sets of fronts \( F = \{ s_{ol1}, s_{ol2}, s_{ol3} \} \) and \( F' = \{ s_{ol6}, s_{ol7} \} \) which are merged at the last level in Fig. 2. Fig. 3 shows the working of the merge procedure to merge two sets of fronts \( F \) and \( F' \) using a sequential search based strategy to insert a solution from \( F' \) in \( F \). Here, the \texttt{INSERT()} procedure is called three times corresponding to the number of fronts in \( F' \). In the first \texttt{INSERT()} procedure, the \texttt{INSERT-SS()} procedure is called twice because the first front in \( F' \) has two solutions. Similarly, in the second and third \texttt{INSERT()} procedures, the \texttt{INSERT-SS()} procedure is called once as there is only a single solution in the second and third fronts of \( F' \).

### 4.2. Extra space is required

The procedure to insert all the solutions of front \( F' \) in \( F \) 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 \( F \), then the number of fronts in \( F \) can be increased by 1. So, the value of \( hfi \) is set to \( P + 1 \) where \( P \) is the number of fronts in \( F' \) before insertion of front \( F' \). In this procedure, before insertion of the solutions from front \( F' \) into \( F \), a vector \( \Psi \) of length \( P - a + 1 \) is initialized with the number of solutions in each front \( F_p \), \( a \leq p \leq P \). 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 \( \Psi[p - a + 1] \) 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 \( F \).

### Example 5

Let \( F = \{ F_1, F_2 \} \) be the set of two fronts where \( F_1 = \{ s_{ol1}, s_{ol2}, s_{ol3}, s_{ol4} \} \) and \( F_2 = \{ s_{ol5}, s_{ol6}, s_{ol7}, s_{ol8} \} \). Consider the solutions of front \( F' = \{ s_{ol9}, s_{ol10}, \ldots, s_{ol16} \} \) which need to be inserted into \( F \) in alternate fronts (\( s_{ol11}, s_{ol12}, s_{ol13}, s_{ol14}, s_{ol15}, s_{ol16} \) are inserted in \( F_2 \) and \( s_{ol10}, s_{ol12}, s_{ol14}, s_{ol16} \) are inserted in \( F_1 \)). When the insertion is performed using Algorithm 3 (\( Y \) is used) and Algorithm 6 (\( \Psi \) 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 \( s_1 \in F' \) in \( F \) depending on the search type – sequential or binary.

#### 4.2.1. Sequential search based strategy

The sequential search based strategy to insert a solution \( s_1 \in F' \) is summarized in Algorithm 7. \( s_1 \) is compared with each of the fronts in \( F' \) starting from \( F_4 \) to \( F_p \) in a sequential manner. When \( s_1 \) is compared with a front \( F_p \), then it is compared only with the initial \( \Psi[p - a + 1] \) solutions and not with all of them. If \( s_1 \) 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 \( s_1 \) is dominated by all the compared fronts, then \( s_1 \) is inserted into \( F_{p+1} \).

#### 4.2.2. Binary search based strategy

A binary search based strategy to insert a solution \( s_1 \in F' \) is summarized in Algorithm 8. Here also, when \( s_1 \) is compared with a front \( F_{mid} \), then it is compared only with the initial \( \Psi[mid - a + 1] \) 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

### Table 1

<table>
<thead>
<tr>
<th>Inserted Solution</th>
<th>( s_{ol1} )</th>
<th>( s_{ol2} )</th>
<th>( s_{ol3} )</th>
<th>( s_{ol4} )</th>
<th>( s_{ol5} )</th>
<th>( s_{ol6} )</th>
<th>( s_{ol7} )</th>
<th>( s_{ol8} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dominance comparisons without ( Y )</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>Dominance comparisons with ( Y )</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>
Algorithm 4  \textsc{Insert-SS}(F, sol).

\textbf{Input:}  \(F\): Set of fronts, \(sol\): Solution for insertion in \(F\)

\textbf{Output:} Updated \(T\) and \(hhi\) based on the insertion of \(sol\)

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

\begin{figure}
\centering
\includegraphics[width=\textwidth]{merge.png}
\caption{Working of the MERGE() procedure to merge two sets of fronts \(F\) and \(F'\). The solutions of each front in \(F'\) are inserted one by one in \(F\). The solution which is added to \(F\) is shown in \textbf{boldface} in \(F\).}
\end{figure}

(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 Refs. [7] and [10]. Our DCNS framework consists of two phases. Heapsort [28] can be adopted in the first phase with time complexity \(O(N \log N)\) as in Ref. [7]. So, the worst case time complexity of the first phase is \(O(MN \log N)\). The best case time complexity is \(O(N \log N)\) when all the solutions have distinct values for the first objective. Let \(L\) be the height of the tree. Assume \(\Gamma_{SS}, \Gamma_{SS_WS}, \Gamma_{BS}\) and \(\Gamma_{BS_WS}\) 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.

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 \(\Gamma_{SS}, \Gamma_{SS_WS}, \Gamma_{BS}\) and \(\Gamma_{BS_WS}\) are the same. Each set of fronts has a single front because all the solutions belong to a single front. Thus, in the \texttt{MERGE()} procedure, the \texttt{INSERT()} or the \texttt{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 \texttt{INSERT()} or the \texttt{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 total number of dominance comparisons in this case is obtained by Eq. (1).
Algorithm 5 \textit{INSERT-BS}(F, sol).

\textbf{Input:} F: Set of fronts, sol: Solution for insertion in F

\textbf{Output:} Updated $\Gamma$ and $\Psi$ based on the insertion of sol

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

Table 2
Maximum number of dominance comparisons required to insert the solutions of front $F$ in $F'$.

<table>
<thead>
<tr>
<th>Inserted Solution</th>
<th>$sol_0$</th>
<th>$sol_0$</th>
<th>$sol_1$</th>
<th>$sol_2$</th>
<th>$sol_3$</th>
<th>$sol_4$</th>
<th>$sol_5$</th>
<th>$sol_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dominance comparisons with $\Upsilon$</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>Dominance comparisons with $\Psi$</td>
<td>8</td>
<td>4</td>
<td>9</td>
<td>5</td>
<td>11</td>
<td>6</td>
<td>13</td>
<td>7</td>
</tr>
</tbody>
</table>

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

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, $\Gamma_{SS} = \Gamma_{SS-WS}$ and $\Gamma_{SS} = \Gamma_{BS-WS}$.

When two sets of fronts $F$ and $F'$ are merged, then after the insertion of the solution of the first front from $F'$ in $F$, the solutions of the remaining fronts from $F'$ are added to $F$ 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 $F'$ are dominated by each of the solutions of $F$. So, a solution of $F'$ needs to be compared and dominated by $2^{l-1}$ solutions in $F$ using a sequential search based strategy. Using a binary search based strategy, a solution of $F'$ needs to be compared and dominated by $\lceil \log(2^{l-1} + 1) \rceil$ solutions in $F$. 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).

\[
\Gamma_{SS} = \sum_{l=1}^{c} \left( \frac{N}{2} \right) (2^{l-1}) (2^{l-1}) = \frac{1}{2} N(N - 1) \tag{1}
\]

\[
\Gamma_{SS} = \sum_{l=1}^{c} \left( \frac{N}{2} \right) \left( 2^{l-1} \right) (1) = \frac{1}{2} N \log N \tag{2}
\]
Algorithm 6 INSERT-WS($F, F', a$).

Input: Same as Algorithm 3

Output: Same as Algorithm 3

1. $P \leftarrow |F|$ // Store the cardinality of $F$
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 $F$
4. for $p \leftarrow \alpha$ to $P$ do
   5. $\Psi[p − \alpha + 1] \leftarrow |F_p|$ // Store the cardinality of front $F_p$
7. for each solution $sol \in F'$ do
   8. $F'' \leftarrow F' \setminus \{sol\}$ // Delete $sol$ from $F''$
9. return $hfi$

Algorithm 7 INSERT-SS-WS($F, sol$).

Input: $F$: Set of fronts, $sol$: Solution for insertion in $F$

Output: Updated $hfi$ depending upon 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 starting from $\alpha^{th}$ front
3. $domCount \leftarrow 0$
4. for $u \leftarrow 1$ to $\Psi[p − \alpha + 1]$ do
5. if $sol$ is non-dominated with $F_p(u)$ then
6. $domCount \leftarrow domCount + 1$
7. else
8. $\text{BREAK}$ // Check for next front in $F$
9. if $domCount = \Psi[p − \alpha + 1]$ then // $sol$ is non-dominated with all the solutions of $F_p$
10. $F_p \leftarrow F_p \cup \{sol\}$ // Insert $sol$ in $F_p$
11. $insertionDone \leftarrow$ TRUE // Insertion of $sol$ is done
12. if $p < hfi$ then // $sol$ is inserted into higher dominance front than $hfi$
13. $hfi \leftarrow p$ // Update $hfi$
14. $\text{BREAK}$ // Do not check other fronts
15. if $insertionDone =$ FALSE then // $sol$ is not yet inserted
16. $F_{p+1} \leftarrow F_{p+1} \cup \{sol\}$ // Insert $sol$ in front $F_{p+1}$

\[ \Gamma_{BS} = \sum_{i=1}^{\ell} \left( \left\lfloor \log \left( \frac{2^{l-1} + 1}{2} \right) \right\rfloor \right) (1) = 2N − log N − 2 \]  \hspace{1cm} (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 $F$ which requires $O(2^{l-1})$ time. Thus, the total time to directly add the solutions without dominance comparisons is $\sum_{i=1}^{\ell} \frac{2^{l-1}}{2} (2^{l-1} − 1) = O(N log N)$. Thus, the time complexity of the second phase is $O(MN log N + N log N) = O(MN log N)$ using a sequential search based strategy, whereas $O(MN + N log N)$ using a binary search based strategy. The best case time complexity of the first phase is $O(N log N)$. Thus, the overall best case time complexity of both DCNS-BS and DCNS-BS-WS is $O(N log N + MN)$.

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 $\ell/2$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 $\ell/2$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, $\Gamma_{SS} = \Gamma_{SS-WS}$ and $\Gamma_{BS} = \Gamma_{BS-WS}$.

As each front has $\sqrt{N} = 2^{\ell/2}$ solutions, so till the $\ell/2$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 comparions 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 $\ell/2$th level is $2^\ell (2^{l-1} − 1)(2^{l-1})$.

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

The values of $\Gamma_{SS}$ and $\Gamma_{SS-WS}$ can be obtained using Eq. (4). Similarly, the values of $\Gamma_{BS}$ and $\Gamma_{BS-WS}$ can be obtained using Eq. (5).

$$\Gamma_{SS} = \sum_{\ell=\lceil\ell/2\rceil+1}^{\ell} \left(\frac{N}{2}\right) \left(2^{\ell-1}\right) \left(2^{\ell/2}\right)$$

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

$$\Gamma_{BS} = \sum_{\ell=\lceil\ell/2\rceil+1}^{\ell} \left(\frac{N}{2}\right) \left(2^{\ell-1}\right) \left(2^{\ell/2}\right)$$

$$+ \sum_{\ell=\lceil\ell/2\rceil+1}^{\ell} \left(\frac{N}{2}\right) \left(\frac{\log\left(2^{\ell-\lceil\ell/2\rceil+1}+1\right)}{2}\right) \left(2^{\ell/2}\right)$$

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

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

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.

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 $O(1)$ as in Ref. [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 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 $Y$ is used which takes constant space, whereas the space required by the second case is $O(N)$. In case of a sequential search based strategy, the fronts in $F$ are accessed sequentially. However, in the case of the binary search based strategy, there is a direct access to the fronts in $F$: So, for direct access of the fronts in $F$, the pointers to the fronts in $F$ are stored in an array. So, the space required in this case is $O(N)$. Hence, the space complexity of DCNS-SS is $O(1)$ and the space complexity of DCNS-BS is $O(N)$. So, the overall time complexity is $O(MN\sqrt{N})$.
The best case time complexity of our approach is better than all other approaches in Table 4. The worst and best case time complexities of several algorithms are compared as follows:

<table>
<thead>
<tr>
<th>Approach</th>
<th>N solutions in single front</th>
<th>N solutions in N fronts</th>
<th>N solutions are equally divided into √N fronts</th>
</tr>
</thead>
<tbody>
<tr>
<td>FNDS [6]</td>
<td>(N(N - 1))</td>
<td>(N(N - 1))</td>
<td>(N(N - 1))</td>
</tr>
<tr>
<td>Deductive [10]</td>
<td>(N(N - 1))</td>
<td>(N(N - 1))</td>
<td>(N(N - 1))</td>
</tr>
<tr>
<td>ENS-BS [7]</td>
<td>(1/N(N - 1))</td>
<td>(1/N(N - 1))</td>
<td>(1/N(N - 1))</td>
</tr>
<tr>
<td>ENS-SS-WS</td>
<td>(1/N(N - 1))</td>
<td>(1/N(N - 1))</td>
<td>(1/N(N - 1))</td>
</tr>
<tr>
<td>DCNS-SS</td>
<td>(1/N(N - 1))</td>
<td>(N log N)</td>
<td>(N log N)</td>
</tr>
<tr>
<td>DCNS-BS-WS</td>
<td>(1/N(N - 1))</td>
<td>(2N - log N - 2)</td>
<td>(N(N - 1)) + 1/√N - 1 + [√N - log N - 4)</td>
</tr>
</tbody>
</table>

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

Table 4: Space and Time complexities of different approaches.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Space Complexity</th>
<th>Time Complexity</th>
<th>Best Case</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive approach</td>
<td>O(N)</td>
<td>O(MN)</td>
<td>O(MN(N))</td>
<td></td>
</tr>
<tr>
<td>FNDS [6]</td>
<td>O(N^2)</td>
<td>O(MN^2)</td>
<td>O(MN^2)</td>
<td></td>
</tr>
<tr>
<td>Deductive Sort [10]</td>
<td>O(N)</td>
<td>O(MN√N)</td>
<td>O(MN√N)</td>
<td></td>
</tr>
<tr>
<td>ENS-SS [7]</td>
<td>O(1)</td>
<td>O(MN log N)</td>
<td>O(MN log N)</td>
<td></td>
</tr>
<tr>
<td>ENS-BS [7]</td>
<td>O(1)</td>
<td>O(MN log N)</td>
<td>O(MN log N)</td>
<td></td>
</tr>
<tr>
<td>BOS [1]</td>
<td>O(MN)</td>
<td>O(MN log N)</td>
<td>O(MN log N)</td>
<td></td>
</tr>
<tr>
<td>T-ENS’ [15]</td>
<td>O(MN)</td>
<td>O(MN log N log M)</td>
<td>O(MN log N log M)</td>
<td></td>
</tr>
<tr>
<td>ENS-NDET [16]</td>
<td>O(N log N)</td>
<td>O(MN log N)</td>
<td>O(MN log N)</td>
<td></td>
</tr>
<tr>
<td>DCNS-SS</td>
<td>O(1)</td>
<td>O(MN log N)</td>
<td>O(MN log N)</td>
<td></td>
</tr>
<tr>
<td>DCNS-BS</td>
<td>O(N)</td>
<td>O(MN log N + MN)</td>
<td>O(MN log N + MN)</td>
<td></td>
</tr>
<tr>
<td>DCNS-SS-WS</td>
<td>O(N)</td>
<td>O(MN log N)</td>
<td>O(MN log N)</td>
<td></td>
</tr>
<tr>
<td>DCNS-BS-WS</td>
<td>O(N)</td>
<td>O(MN log N + MN)</td>
<td>O(MN log N + MN)</td>
<td></td>
</tr>
</tbody>
</table>

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

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 Refs. [29–31]. 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. 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 Mth 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 on 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 ∈ F can be compared with the solutions of a front F ∈ 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 ∈ F can be compared with the solutions of a front F ∈ F simultaneously. Also, a solution of a front F is compared with all the solutions of a front F simultaneously.

The space complexity of DCNS-BS is O(N). The space complexity of DCNS-SS-WS and DCNS-BS-WS is O(N).

In BOS [1], the solutions can be sorted based on the second to Mth 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 on 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 ∈ F can be compared with the solutions of a front F ∈ 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 ∈ F can be compared with the solutions of a front F ∈ 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].

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 Ref. [7]. Four different objectives – 2, 5, 10 and 20 are considered. The number of dominance comparisons and the running times (in milliseconds) for this setup are shown in Fig. 4. For two objectives, DCNS outperforms FNDS, Deductive sort and ENS-SS while it underperforms ENS-BS both in terms of the number of comparisons as well as running time. DCNS-SS and DCNS-SS-WS outperform ENS-BS. For five objectives, ENS-SS outperforms the other approaches in terms of both the number of comparisons and running time. 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 Ref. [7]. Four different numbers of objectives – 2, 5, 10 and 20 are considered. The number of dominance comparisons and running times (in milliseconds) for this setup are shown in Fig. 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 Fig. 5(a) – (b), 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
Table 5 shows the number of dominance comparisons and the running time (in milliseconds) for solving the test problems DTLZ2, DTLZ3, and DTLZ4 [33].

Table 6 shows the number of dominance comparisons and the running times (in milliseconds) for solving the test problem DTLZ2, DTLZ3, and DTLZ4 [33].

The number of dominance comparisons and the running times (in milliseconds) for solving the test problem DTLZ2, DTLZ3, and DTLZ4 [33].
5.4. Discussion & analysis

The worst case time complexity of the DCNS framework is $O(MN^2)$, whereas the best case time complexity is $O(N \log N + MN)$. The same worst case time complexity is reported by various other approaches \cite{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. \cite{4}. According to this, the processing time is bounded from below by $O(N \log N)$ \cite{3}. Jensen et al. \cite{3} claimed that it is trivial to see that this bound must also hold for non-dominated sorting. Jensen et al. \cite{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 $O(N \log N + MN)$ 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 $M = O(\log N)$. The assumption $M = O(\log N)$ seems to be valid, because $M$ is generally very low as compared to $N$. However, the upper bound of the DCNS framework is still $O(MN^2)$. 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 \cite{1} can also exploit parallelism using this framework.

Fig. 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 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-SS and DCNS-SS-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 $\Gamma_{ENS-SS}$ and the number of dominance comparisons performed by ENS-BS be denoted by $\Gamma_{ENS-BS}$. 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 performed by the DCNS-based approaches is less than that of the other two approaches. Let $N = 2^b$ be the number of solutions. The number of fronts $K = 2^a$, $1 \leq b \leq 16$. In this case, the number of dominance comparisons in the first and the second situations are shown in Fig. 7(a) and (b), respectively. In the first situation, for ENS-SS, the minimum number of dominance comparisons is attained at $K = 2^b$ because its best case occurs when $K = \sqrt{N} = \sqrt{2^b} = 2^b$. 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 $O(MN^2)$ 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 $O(N \log N + MN)$ which is better than the best case time complexities of many other existing approaches. The lower bound of non-dominated sorting as discussed in Refs. \cite{3,4} is also obtained in the best case of our approaches. However, the upper bound remains as $O(MN^2)$.

BOS \cite{1} can be generalized in the future to handle duplicate solutions by retaining its comparison set concept. T-ENS \cite{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.
Appendix A. Effect of $\Upsilon$ and $\Psi$ 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 $F$ using $\Upsilon$ and $\Psi$. For this purpose, Example 5 of the paper is considered where $F = \{F_1, F_2\}$, $F_1 = \{s_{10}, s_{11}, s_{12}, s_{13}, s_{14}\}$ and $F_2 = \{s_{15}, s_{16}, s_{17}, s_{18}\}$. The front $F' = \{s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15}, s_{16}\}$. The set of solutions in $F'$ to which each solution of front $F'$ is compared for insertion in $F$ without using $\Upsilon$ or $\Psi$ is given in Table A.7.

<table>
<thead>
<tr>
<th>Inserted solution</th>
<th>Compared solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_{10}$</td>
<td>$s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15}, s_{16}$</td>
</tr>
<tr>
<td>$s_{11}$</td>
<td>$s_{11}, s_{12}, s_{13}, s_{14}, s_{15}, s_{16}$</td>
</tr>
<tr>
<td>$s_{12}$</td>
<td>$s_{12}, s_{13}, s_{14}, s_{15}, s_{16}$</td>
</tr>
<tr>
<td>$s_{13}$</td>
<td>$s_{13}, s_{14}, s_{15}, s_{16}$</td>
</tr>
<tr>
<td>$s_{14}$</td>
<td>$s_{14}, s_{15}, s_{16}$</td>
</tr>
<tr>
<td>$s_{15}$</td>
<td>$s_{15}, s_{16}$</td>
</tr>
<tr>
<td>$s_{16}$</td>
<td>$s_{16}$</td>
</tr>
</tbody>
</table>

Table A.7

Set of solutions in $F'$ to which each solution in front $F'$ is compared before being inserted in $F$ when neither $\Upsilon$ nor $\Psi$ is used.

Appendix A.1. Insertion using $\Upsilon$

We discuss how the solutions of front $F'$ are inserted in $F$ using $\Upsilon$. Initially, $\Upsilon = \{\Upsilon_{\mathrm{index}}, \Upsilon_{\mathrm{Sol}}\} = \{0, 0\}$.

**Insert $s_{10}$**: When $s_{10}$ is inserted into $F_1$, it is compared with all the solutions of $F_1$ and inserted in $F_2$. The updated value of $\Upsilon = \{2, 4\}$ because $s_{10}$ is inserted in the second front $F_2$ ($\Upsilon_{\mathrm{index}} = 2$) and has been compared with four solutions ($\Upsilon_{\mathrm{Sol}} = 4$) in $F_2$.

**Insert $s_{11}$**: When $s_{11}$ is inserted into $F_1$, it is compared with all the solutions of front $F_1$ and inserted in $F_1$. The updated value of $\Upsilon = \{1, 4\}$ because $s_{11}$ is inserted in the first front $F_1$ ($\Upsilon_{\mathrm{index}} = 1$) and has been compared with four solutions ($\Upsilon_{\mathrm{Sol}} = 4$) in $F_1$.

**Insert $s_{12}$**: At this point of time, front $F_1$ has five solutions as $s_{10}$ has already been inserted into $F_1$. As the value of $\Upsilon = \{1, 4\}$ so when $s_{11}$ starts comparison with the solutions of front $F_1$, it is compared with the initial four ($\Upsilon_{\mathrm{index}} = 1$ and $\Upsilon_{\mathrm{Sol}} = 4$) solutions, instead of five. Solution $s_{11}$ is compared with all the five solutions in front $F_2$. After the insertion of $s_{11}$ in front $F_2$, the updated value of $\Upsilon = \{2, 5\}$ because $s_{11}$ is inserted in the second front $F_2$ ($\Upsilon_{\mathrm{index}} = 2$) and has been compared with five solutions ($\Upsilon_{\mathrm{Sol}} = 5$) in $F_2$.

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

<table>
<thead>
<tr>
<th>Initial set of fronts $\mathcal{F}$</th>
<th>$\mathcal{F}$ and $\Upsilon$ after insertion of $s_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$ $s_{10}, s_{11}, s_{12}, s_{13}, s_{14}$</td>
<td>$F_1$ $s_{10}, s_{11}, s_{12}, s_{13}, s_{14}$</td>
</tr>
<tr>
<td>$F_2$ $s_{10}, s_{11}, s_{12}, s_{13}, s_{14}$</td>
<td>$F_2$ $s_{10}, s_{11}, s_{12}, s_{13}, s_{14}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\mathcal{F}$ and $\Upsilon$ after insertion of $s_{11}$</th>
<th>$\mathcal{F}$ and $\Upsilon$ after insertion of $s_{12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$ $s_{10}, s_{11}, s_{12}, s_{13}, s_{14}$</td>
<td>$F_1$ $s_{10}, s_{11}, s_{12}, s_{13}, s_{14}$</td>
</tr>
<tr>
<td>$F_2$ $s_{10}, s_{11}, s_{12}, s_{13}, s_{14}$</td>
<td>$F_2$ $s_{10}, s_{11}, s_{12}, s_{13}, s_{14}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\mathcal{F}$ and $\Upsilon$ after insertion of $s_{13}$</th>
<th>$\mathcal{F}$ and $\Upsilon$ after insertion of $s_{14}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$ $s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15}$</td>
<td>$F_1$ $s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15}$</td>
</tr>
<tr>
<td>$F_2$ $s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15}$</td>
<td>$F_2$ $s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\mathcal{F}$ and $\Upsilon$ after insertion of $s_{15}$</th>
<th>$\mathcal{F}$ and $\Upsilon$ after insertion of $s_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$ $s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15}, s_{16}$</td>
<td>$F_1$ $s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15}, s_{16}$</td>
</tr>
<tr>
<td>$F_2$ $s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15}, s_{16}$</td>
<td>$F_2$ $s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15}, s_{16}$</td>
</tr>
</tbody>
</table>

Fig. A.8. Insertion of all the solutions of front $F'$ in $F$. 

16
Appendix A.2. Insertion using $\Psi$

We discuss how the solutions of front $F'$ are inserted in $F$ using $\Psi$. As there are two fronts in $F$, so the cardinality of $\Psi$ 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 $F$, 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 $F$, 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 $F$, 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 $F$. The set of solutions to which each of the solutions of front $F'$ is compared for insertion in $F$ using $\Psi$ is given in Table A.9. Table A.10 shows the number of dominance comparisons when the solutions from front $F'$ are inserted in $F$ without using $Y$ or $\Psi$, using $Y$ and using $\Psi$. From this table, it is clear that the number of dominance comparisons is the minimum when $\Psi$ is considered.

### Table A.8
Set of solutions in $F$ to which each solution in $F'$ is compared before being inserted in $F$ using $Y$.

<table>
<thead>
<tr>
<th>Inserted solution</th>
<th>Compared solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$sol_9$</td>
<td>$sol_{1-6}, sol_{10}, sol_{11}, sol_{12}, sol_{13}, sol_{14}, sol_{15}$</td>
</tr>
<tr>
<td>$sol_{10}$</td>
<td>$sol_{1-6}, sol_{10}$</td>
</tr>
<tr>
<td>$sol_{11}$</td>
<td>$sol_{1-6}, sol_{11}$</td>
</tr>
<tr>
<td>$sol_{12}$</td>
<td>$sol_{1-6}, sol_{12}$</td>
</tr>
<tr>
<td>$sol_{13}$</td>
<td>$sol_{1-6}, sol_{13}$</td>
</tr>
<tr>
<td>$sol_{14}$</td>
<td>$sol_{1-6}, sol_{14}$</td>
</tr>
<tr>
<td>$sol_{15}$</td>
<td>$sol_{1-6}, sol_{15}$</td>
</tr>
<tr>
<td>$sol_{16}$</td>
<td>$sol_{1-6}, sol_{16}$</td>
</tr>
</tbody>
</table>

### Table A.9
Set of solutions in $F$ to which each solution in $F'$ is compared before being inserted in $F$ using $\Psi$.

<table>
<thead>
<tr>
<th>Inserted solution</th>
<th>Compared solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$sol_9$</td>
<td>$sol_{1-2}, sol_{5-6}, sol_{10}, sol_{11}, sol_{12}, sol_{13}, sol_{14}$</td>
</tr>
<tr>
<td>$sol_{10}$</td>
<td>$sol_{1-2}, sol_{5-6}, sol_{10}$</td>
</tr>
<tr>
<td>$sol_{11}$</td>
<td>$sol_{1-2}, sol_{5-6}, sol_{11}$</td>
</tr>
<tr>
<td>$sol_{12}$</td>
<td>$sol_{1-2}, sol_{5-6}, sol_{12}$</td>
</tr>
<tr>
<td>$sol_{13}$</td>
<td>$sol_{1-2}, sol_{5-6}, sol_{13}$</td>
</tr>
<tr>
<td>$sol_{14}$</td>
<td>$sol_{1-2}, sol_{5-6}, sol_{14}$</td>
</tr>
<tr>
<td>$sol_{15}$</td>
<td>$sol_{1-2}, sol_{5-6}, sol_{15}$</td>
</tr>
<tr>
<td>$sol_{16}$</td>
<td>$sol_{1-2}, sol_{5-6}, sol_{16}$</td>
</tr>
</tbody>
</table>

### Table A.10
Number of dominance comparisons when the solutions from front $F'$ are inserted in $F$ without using $Y$ or $\Psi$, using $Y$ and using $\Psi$.

<table>
<thead>
<tr>
<th>Solution</th>
<th>Simple</th>
<th>Use of Y</th>
<th>Use of $\Psi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$sol_9$</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>$sol_{10}$</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>$sol_{11}$</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>$sol_{12}$</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>$sol_{13}$</td>
<td>12</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>$sol_{14}$</td>
<td>6</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>$sol_{15}$</td>
<td>14</td>
<td>13</td>
<td>8</td>
</tr>
<tr>
<td>$sol_{16}$</td>
<td>7</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>Total dominance comparisons</td>
<td>66</td>
<td>63</td>
<td>48</td>
</tr>
</tbody>
</table>

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 $O(N)$ for $N$ numbers which is an $O(\log N)$ times improvement over the serial version. As suggested in Ref. [32], to improve the time complexity of parallel merge sort further, the merge operation itself can also be performed 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 $O(\log^2 N)$ which is an $O(N/\log^2 N)$ times improvement over the serial version.

**Parallellism 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 $O(MN)$ and the best case time complexity is $O(N \log N)$. Using parallel merge sort where all the merge operations at the same level are performed simultaneously, the worst case time complexity becomes $O(MN)$ and the best case time complexity becomes $O(N)$. 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 Ref. [32]. In this manner, the worst case time complexity becomes $O(M \log^2 N)$ and the best case time complexity becomes $O(\log^3 N)$.

**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 $F$ and the second set of fronts be denoted as $F'$. 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 $F' \in F'$ are identified in $F$ simultaneously and they are added to their respective front in a sequential manner to avoid a write collision (or critical section). Here, a solution of front $F$ is compared with a solution of a front $F' \in F$ sequentially.

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

Consider the same set of fronts as considered in parallel version-2. These two sets of fronts are also shown in Fig. B.9(b). In parallel version-3 of the proposed approach, solutions $sol_5, sol_{10}, sol_{11}, sol_{12}$ are compared with the solutions of the first front in $F$ simultaneously. Also, each of these solutions is compared with the solutions of the first front in $F$ simultaneously.

### 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. These three scenarios are discussed in Refs. [1,2,7,10].

#### 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 $M(N/2)^2$ corresponds to the time to obtain the dominance relationship of the solutions in $F$ with the solutions in $F'$. The second part $N/2$ corresponds to the time to add the solutions in $F'$ to $F$.

$$T_s(N,M) = \begin{cases} M + 1 & \text{if } N = 2 \\ 2T_s(N/2,M) + M(N/2)^2 + N/2 & \text{otherwise} \end{cases}$$

(C.1)

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

$$T_s(N,M) = \left[ M(N/2)^2 + N/2 \right] + 2 \left[ M(N/4)^2 + N/4 \right] + \ldots + N/2 \left[ M(N/N)^2 + N/N \right]$$

$$= 1/2 MN(N-1) + 1/2 N \log N$$

(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).

$$T_{ps}(N,M) = \begin{cases} M + 1 & \text{if } N = 2 \\ T_{ps}(N/2,M) + M(N/2)^2 + N/2 & \text{otherwise} \end{cases}$$

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

\[ T_{\infty}(N, M) = \left[ \frac{M(N/2)^2 + N/2}{1/3 M (N^2 - 1) + (N - 1)} \right] + \left[ \frac{M(N/4)^2 + N/4}{1/3 M (N^2 - 1) + (N - 1)} \right] + \ldots + \left[ \frac{M(N/N)^2 + N/N}{1/3 M (N^2 - 1) + (N - 1)} \right] \]

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

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

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

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

\[ T_{\infty}(N, M) = \left[ \frac{M(N/2)^2 + N/2}{1/3 M (N^2 - 1) + (N - 1)} \right] + \left[ \frac{M(N/4)^2 + N/4}{1/3 M (N^2 - 1) + (N - 1)} \right] + \ldots + \left[ \frac{M(N/N)^2 + N/N}{1/3 M (N^2 - 1) + (N - 1)} \right] \]

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

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

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

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

\[ T_{\infty}(N, M) = \left[ \frac{M(N/2)^2 + N/2}{1/3 M (N^2 - 1) + (N - 1)} \right] + \left[ \frac{M(N/4)^2 + N/4}{1/3 M (N^2 - 1) + (N - 1)} \right] + \ldots + \left[ \frac{M(N/N)^2 + N/N}{1/3 M (N^2 - 1) + (N - 1)} \right] \]

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

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

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

\[ T_1(N, M) = \left[ \frac{M(N/2)^2 + N/2}{1/3 M (N^2 - 1) + (N - 1)} \right] + \left[ \frac{M(N/4)^2 + N/4}{1/3 M (N^2 - 1) + (N - 1)} \right] + \ldots + \left[ \frac{M(N/N)^2 + N/N}{1/3 M (N^2 - 1) + (N - 1)} \right] \]

\[ = 1/2 MN \log N + \frac{1}{2} N \log N \] (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).

\[
T_{\infty}(N, M) = \begin{cases} 
    M + 1 & \text{if } N = 2 \\
    T_{\infty}(N/2, M) + M(N/2) + N/2 & \text{otherwise} 
\end{cases}
\]  

(D.3)

\[
T_{\infty}(N, M) = [M(N/2) + N/2] + [M(N/4) + N/4] + \ldots + [M(N/N) + N/N] = M(N - 1) + (N - 1)
\]  

(D.4)

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

\[
\text{Speedup} = \frac{T_1(N, M)}{T_{\infty}(N, M)} = \frac{1/2MN \log N + 1/2N \log N}{M(N - 1) + (N - 1)} \equiv 1/2 \log N
\]  

(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).

\[
T_1(N, M) = \begin{cases} 
    M + 1 & \text{if } N = 2 \\
    2T_1(N/2, M) + M[\log(N/2 + 1)] + N/2 & \text{otherwise} 
\end{cases}
\]  

(D.6)

\[
T_1(N, M) = [M[\log(N/2 + 1)] + N/2] + 2[M[\log(N/4 + 1)] + N/4] + \ldots + N/2[M[\log(N/N + 1)] + N/N]
\]  

\[= M(2N - \log N - 2) + 1/2N \log N
\]  

(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).

\[
T_{\infty}(N, M) = \begin{cases} 
    M + 1 & \text{if } N = 2 \\
    T_{\infty}(N/2, M) + M[\log(N/2 + 1)] + N/2 & \text{otherwise} 
\end{cases}
\]  

(D.8)

\[
T_{\infty}(N, M) = [M[\log(N/2 + 1)] + N/2] + [M[\log(N/4 + 1)] + N/4] + \ldots + [M[\log(N/N + 1)] + N/N]
\]  

\[= 1/2M[\log^2N + \log N] + (N - 1)
\]  

(D.9)

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

\[
\text{Speedup} = \frac{T_1(N, M)}{T_{\infty}(N, M)} = \frac{M(2N - \log N - 2) + 1/2N \log N}{1/2M[\log^2N + \log N] + (N - 1)} \equiv 1/2 \log N
\]  

(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 level 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 \(P'\) have a lower dominance than that of the fronts in \(P\). Only the solutions of the first front in \(P'\) are compared with the solutions of \(P\). The solutions of the remaining fronts in \(P'\) are added directly to \(P\) because of the dominance relationship as discussed in Ref. [2]. So, the second part of Eq. (E.1) corresponds to the process of non-dominated sorting after the \(\mathcal{L}/2\) th level.
The solution to the recurrence relation in Eq. (E.1) is obtained in Eq. (E.6).

\[ T_1(N, M) = N'T_{11}(\sqrt{N}, M) + T_{12}(N, M) \]

(E.1)

\[ T_{11}(\sqrt{N}, M) = \begin{cases} M + 1 & \text{if } \sqrt{N} = 2 \\ 2T_{11}(\sqrt{N}/2, M) + M(\sqrt{N}/2)^2 + \sqrt{N}/2 & \text{otherwise} \end{cases} \]

(E.2)

\[ T_{12}(N, M) = \begin{cases} MN' + N' & \text{if } N = 2N' \\ 2T_{12}(N/2, M) + M(N/2N') + (N/2N')N' & \text{otherwise} \end{cases} \]

(E.3)

\[ T_{11}(\sqrt{N}, M) = \frac{M}{2} \sqrt{N}(\sqrt{N} - 1) + \frac{1}{4} \sqrt{N} \log N \]

(E.4)

\[ T_{12}(N, M) = [M(N/2) + N/2] + 2[M(N/4) + N/4] + \ldots + N'/2 [M(N/N') + N/N'] \]

(E.5)

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

\[ T_1(N, M) = N'T_{11}(\sqrt{N}, M) + T_{12}(N, M) = \sqrt{N} \left[ \frac{1}{2} M \sqrt{N} \sqrt{N} - 1 + \frac{1}{4} \sqrt{N} \log N \right] + \frac{1}{4} MN \log N + \frac{1}{4} N \log N \]

(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).

\[ T_{oo}(N, M) = N'T_{oo1}(\sqrt{N}, M) + T_{oo2}(N, M) \]

(E.7)

\[ T_{oo1}(\sqrt{N}, M) = \begin{cases} M + 1 & \text{if } \sqrt{N} = 2 \\ T_{oo1}(\sqrt{N}/2, M) + M(\sqrt{N}/2)^2 + \sqrt{N}/2 & \text{otherwise} \end{cases} \]

(E.8)

\[ T_{oo2}(N, M) = \begin{cases} MN' + N' & \text{if } N = 2N' \\ T_{oo2}(N/2, M) + M(N/2N') + (N/2N')N' & \text{otherwise} \end{cases} \]

(E.9)

\[ T_{oo1}(\sqrt{N}, M) = \frac{M}{2} \sqrt{N}(\sqrt{N} - 1) + \frac{1}{3} \sqrt{M}(\sqrt{N} - 1) \]

(E.10)

\[ T_{oo2}(N, M) = [M(N/2) + N/2] + [M(N/4) + N/4] + \ldots + [M(N/N') + N/N'] \]

(E.11)

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

\[ T_{oo}(N, M) = N'T_{oo1}(\sqrt{N}, M) + T_{oo2}(N, M) = \sqrt{N} \left[ \frac{1}{3} M(N - 1) + \frac{1}{3} \sqrt{M}(\sqrt{N} - 1) \right] + \frac{1}{3} M \sqrt{N}(\sqrt{N} + 3 \sqrt{N} - 4) + 2 \sqrt{N}(\sqrt{N} - 1) \]

(E.12)

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

\[ \text{Speedup} = \frac{T_o(N, M)}{T_{oo}(N, M)} = \frac{1/4 MN(2\sqrt{N} + \log N - 2) + 1/2 N \log N}{1/3 M \sqrt{N}(N + 3 \sqrt{N} - 4) + 2 \sqrt{N}(\sqrt{N} - 1)} \equiv 3/2 \]

(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).

\[ T_{\infty}(N, M) = N^2 T_{\infty1}(\sqrt{N}, M) + \frac{T_{\infty2}(N, M)}{N^{2N'}} \]  
(E.14)

\[ T_{\infty1}(\sqrt{N}, M) = \begin{cases} 
M + 1 & \text{if } \sqrt{N} = 2 \\
T_{\infty1}\left(\sqrt{N/2}, M\right) + M\left(\sqrt{N}/2\right) + \sqrt{N}/2 & \text{otherwise}
\end{cases} \]  
(E.15)

\[ T_{\infty2}(N, M) = \begin{cases} 
M + N' & \text{if } N = 2N' \\
T_{\infty2}(N/2, M) + N/2 & \text{otherwise}
\end{cases} \]  
(E.16)

\[ T_{\infty1}(\sqrt{N}, M) = M\left(\sqrt{N}/2\right) + \sqrt{N}/2 + M\left(\sqrt{N}/4\right) + \sqrt{N}/4 + \ldots + \left[M\left(\sqrt{N}/\sqrt{N}\right) + \sqrt{N}/\sqrt{N}\right] = M(\sqrt{N} - 1) + (\sqrt{N} - 1) \]  
(E.17)

\[ T_{\infty2}(N, M) = M(\sqrt{N}/2N') + N/2 + \ldots + [M(\sqrt{N}/N')N'] = M(\sqrt{N} - 1) + (\sqrt{N} - 1) \]  
(E.18)

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

\[ T_{\infty}(N, M) = N^2 T_{\infty1}(\sqrt{N}, M) + T_{\infty2}(N, M) = \sqrt{N} T_{\infty1}(\sqrt{N}, M) + T_{\infty2}(N, M) = \sqrt{N}\left[M(\sqrt{N} - 1) + (\sqrt{N} - 1)\right] + M(\sqrt{N} - 1) + (\sqrt{N} - 1) = M(N - 1) + 2\sqrt{N}(\sqrt{N} - 1) \]  
(E.19)

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

\[ \text{Speedup} = \frac{T_{\infty}(N, M)}{T_{\infty}(N, M)} = \frac{1/4 MN(2\sqrt{N} + \log N - 2) + 1/2 N \log N}{M(N - 1) + 2\sqrt{N}(\sqrt{N} - 1)} = 1/2\sqrt{N} \]  
(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).

\[ T_{\infty}(N, M) = N^2 T_{\infty1}(\sqrt{N}, M) + \frac{T_{\infty2}(N, M)}{N^{2N'}} \]  
(E.21)

\[ T_{\infty1}(\sqrt{N}, M) = \begin{cases} 
M + 1 & \text{if } \sqrt{N} = 2 \\
T_{\infty1}\left(\sqrt{N/2}, M\right) + M + \log\left(\sqrt{N}/2\right) + \sqrt{N}/2 & \text{otherwise}
\end{cases} \]  
(E.22)

\[ T_{\infty2}(N, M) = \begin{cases} 
M + N' & \text{if } N = 2N' \\
T_{\infty2}(N/2, M) + N/2N' + (M + \log N') + (N/2N')N' & \text{otherwise}
\end{cases} \]  
(E.23)

\[ T_{\infty1}(\sqrt{N}, M) = M + \log\left(\sqrt{N}/2\right) + \sqrt{N}/2 + M + \log\left(\sqrt{N}/4\right) + \sqrt{N}/4 + \ldots + M + \log\left(\sqrt{N}/\sqrt{N}\right) + \sqrt{N}/\sqrt{N} = 1/2 M \log N + 1/8(8\sqrt{N} + \log^2 N - 2 \log N - 8) \]  
(E.24)

\[ T_{\infty2}(N, M) = [N/2N' + (M + \log N') + N/2 + [N/4N' + (M + \log N') + N/4] + \ldots + [N/N'N' + (M + \log N') + N/N'] = M(\sqrt{N} - 1) + 1/2(2N + \sqrt{N} \log N - 2 \sqrt{N} - \log N) \]  
(E.25)

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

\[ T_{\infty}(N, M) = \sqrt{N} T_{\infty1}(\sqrt{N}, M) + T_{\infty2}(N, M) = \sqrt{N} T_{\infty1}(\sqrt{N}, M) + T_{\infty2}(N, M) = \sqrt{N}\left[1/2 M \log N + 1/8(8\sqrt{N} + \log^2 N - 2 \log N - 8)\right] + M(\sqrt{N} - 1) + 1/2(2N + \sqrt{N} \log N - 2 \sqrt{N} - \log N) = 1/2 M(\sqrt{N} \log N + 2 \sqrt{N} - 2) + 1/8(16 N + \sqrt{N} \log^2 N + 2 \sqrt{N} \log N - 16 \sqrt{N} - 4 \log N) \]  
(E.26)

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

\[ \text{Speedup} = \frac{T_{\infty}(N, M)}{T_{\infty}(N, M)} = 1/4 M \sqrt{N} = 1/4 M \sqrt{N} \]  
(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).

\[
T_1(N, M) = N'T_{11}(\sqrt{N}, M) + T_{12}(N, M) \quad \text{if } N > N'
\]

(E.28)

\[
T_{11}(\sqrt{N}, M) = \begin{cases} 
M + 1 & \text{if } \sqrt{N} = 2 \\
2T_{11}(\sqrt{N}/2, M) + M\left(\sqrt{N}/2\right)^2 + \sqrt{N}/2 & \text{otherwise} 
\end{cases}
\]

(E.29)

\[
T_{12}(N, M) = \begin{cases} 
MN' + N' \quad & \text{if } N = 2N' \\
2T_{12}(N/2, M) + M\left(\log(N/2N' + 1)\right)N' + (N/2N')N' \quad & \text{otherwise} 
\end{cases}
\]

(E.30)

\[
T_{11}(\sqrt{N}, M) = M\left(\sqrt{N}/2\right)^2 + \sqrt{N}/2 + 2\left[M\left(\sqrt{N}/4\right)^2 + \sqrt{N}/4\right] + \ldots + \frac{\sqrt{N}/2}{M\left(\sqrt{N}/\sqrt{N}\right)^2 + \sqrt{N}/\sqrt{N}}
\]

(E.31)

\[
T_{12}(N, M) = M\left(\log(N/2N' + 1)\right)N' + N/2 + 2M\left(\log(N/4N' + 1)\right)N' + N/4 + \ldots + N'/2 \left[M\left(\log(N/N'N' + 1)\right)N' + N/N'\right]
\]

(E.32)

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

\[
T_1(N, M) = N'T_{11}(\sqrt{N}, M) + T_{12}(N, M) = \sqrt{N}T_{11}(\sqrt{N}, M) + T_{12}(N, M)
\]

(E.33)

\[
= \sqrt{N} \left[1/2M \sqrt{N} (\sqrt{N} - 1) + 1/4\sqrt{N} \log N\right] + 1/2M \sqrt{N} (4\sqrt{N} - \log N - 4) + 1/4 N \log N
\]

(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).

\[
T_{\text{oa}}(N, M) = N'T_{\text{oa}1}(\sqrt{N}, M) + T_{\text{oa}2}(N, M) \quad \text{if } N > N'
\]

(E.34)

\[
T_{\text{oa}1}(\sqrt{N}, M) = \begin{cases} 
M + 1 \quad & \text{if } \sqrt{N} = 2 \\
T_{\text{oa}1}(\sqrt{N}/2, M) + M\left(\sqrt{N}/2\right)^2 + \sqrt{N}/2 \quad & \text{otherwise} 
\end{cases}
\]

(E.35)

\[
T_{\text{oa}2}(N, M) = \begin{cases} 
MN' + N' \quad & \text{if } N = 2N' \\
T_{\text{oa}2}(N/2, M) + M\left(\log(N/2N' + 1)\right)N' + (N/2N')N' \quad & \text{otherwise} 
\end{cases}
\]

(E.36)

\[
T_{\text{oa}1}(\sqrt{N}, M) = M\left(\sqrt{N}/2\right)^2 + \sqrt{N}/2 + \ldots + M\left(\sqrt{N}/\sqrt{N}\right)^2 + \sqrt{N}/\sqrt{N}
\]

(E.37)

\[
T_{\text{oa}2}(N, M) = M\left(\log(N/2N' + 1)\right)N' + N/2 + 2M\left(\log(N/4N' + 1)\right)N' + N/4 + \ldots + M\left(\log(N/N'N' + 1)\right)N' + N/N'
\]

(E.38)

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

\[
T_{\text{oa}}(N, M) = N'T_{\text{oa}1}(\sqrt{N}, M) + T_{\text{oa}2}(N, M) = \sqrt{N}T_{\text{oa}1}(\sqrt{N}, M) + T_{\text{oa}2}(N, M)
\]

(E.39)

\[
= \sqrt{N} \left[1/3M(N - 1) + (\sqrt{N} - 1)\right] + 1/8 M \sqrt{N}(\log^2 N + 2 \log N) + \sqrt{N}(\sqrt{N} - 1)
\]

(E.39)
The speedup using parallel version-1 is obtained in Eq. (E.40).

\[
\frac{T_1(N, M)}{T_{\omega}(N, M)} = \frac{1/2 M \sqrt{N(N + 3\sqrt{N} - \log N - 4)} + 1/2 N \log N}{1/24 M \sqrt{N}(8 N + 3\log^2 N + 6 \log N - 8) + 2 \sqrt{N}(N - 1)} \equiv 3/2
\] (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).

\[
T_{\omega}(N, M) = N^2 T_{\omega1}(\sqrt[4]{N}, M) + T_{\omega2}(N, M)
\] (E.41)

\[
T_{\omega1}(\sqrt[4]{N}, M) = \begin{cases} 
M + 1 & \text{if } \sqrt[4]{N} = 2 \\
T_{\omega1}\left(\sqrt[4]{N/2}, M\right) + M \left(\sqrt[4]{N/2}\right) + \sqrt[4]{N/2} & \text{otherwise}
\end{cases}
\] (E.42)

\[
T_{\omega2}(N, M) = \begin{cases} 
M + N' & \text{if } N = 2N' \\
T_{\omega2}(N/2, M) + M \left(\lfloor \log (N/2N') + 1 \rfloor \right) + (N/2N')N' & \text{otherwise}
\end{cases}
\] (E.43)

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

\[
T_{\omega}(N, M) = N^2 T_{\omega1}(\sqrt[4]{N}, M) + T_{\omega2}(N, M) = \sqrt{N} T_{\omega1}(\sqrt{N}, M) + T_{\omega2}(N, M)
\] (E.46)

\[
= \sqrt{N} \left[ M(\sqrt{N} - 1) + \sqrt{N} - 1 \right] + 1/8 M(log^2 N + 2\log N) + \sqrt{N}(\sqrt{N} - 1)
\] (E.46)

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

\[
\frac{T_1(N, M)}{T_{\omega}(N, M)} \equiv 1/2 \sqrt{N}
\] (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).

\[
T_{\omega}(N, M) = \sqrt{N} T_{\omega1}(\sqrt{N}, M) + T_{\omega2}(N, M)
\] (E.48)

\[
T_{\omega1}(\sqrt{N}, M) = \begin{cases} 
M + 1 & \text{if } \sqrt{N} = 2 \\
T_{\omega1}\left(\sqrt{N/2}, M\right) + M + \log \left(\sqrt{N/2}\right) + \sqrt{N/2} & \text{otherwise}
\end{cases}
\] (E.49)

\[
T_{\omega2}(N, M) = \begin{cases} 
M + \log N' + N' & \text{if } N = 2N' \\
T_{\omega2}(N/2, M) + \lfloor \log (N/2N') + 1 \rfloor \left(M + \log N'\right) + (N/2N')N' & \text{otherwise}
\end{cases}
\] (E.50)

\[
T_{\omega1}(\sqrt{N}, M) = M + \log \left(\sqrt{N/2}\right) + \sqrt{N/2} + \left[M + \log \left(\sqrt{N/4}\right) + \sqrt{N/4}\right] + \ldots + \left[M + \log \left(\sqrt{N/N}\right) + \sqrt{N/N}\right]
\] (E.51)

\[
T_{\omega2}(N, M) = \left[\lfloor \log (N/2N') + 1 \rfloor \left(M + \log N'\right) + N/2 \right] + \ldots + \left[\lfloor \log (N/N'N' + 1) \rfloor \left(M + \log N'\right) + N/N'\right]
\] (E.52)
The solution to the recurrence relation in Eq. (E.48) is obtained in Eq. (E.53).

\[ T_{\text{oc}}(N, M) = N^2 T_{\text{oc}}(\sqrt{N}, M) + T_{\text{oc2}}(N, M) = \sqrt{N} T_{\text{oc}}(\sqrt{N}, M) + T_{\text{oc2}}(N, M) 
\]

\[ = \sqrt{N} \left[ 1/2 M \log N + 1/8 (8 \sqrt{N} + \log^2 N - 2 \log N - 8) \right] + 1/16 (16 N - 16 \sqrt{N} + \log^2 N + 2 \log N) + \sqrt{N} (\sqrt{N} - 1) \]

\[ = 1/8 M (4 \sqrt{N} \log N + \log^2 N + 2 \log N) + 1/16 (32 N - 32 \sqrt{N} + 2 \sqrt{N} \log^2 N - 4 \sqrt{N} \log N + \log^3 N + 2 \log^2 N) \]  

(E.53)

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

Speedup = \frac{T_{\text{oc}}(N, M)}{T_{\text{oc}}(\sqrt{N}, M)} = 1 \sqrt{M N} 

(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.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Version-1</th>
<th>Version-2</th>
<th>Version-3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sequential Binary</td>
<td>Sequential Binary</td>
<td>Sequential Binary</td>
</tr>
<tr>
<td><strong>N solutions in single front</strong></td>
<td>( \frac{1}{2} \log N )</td>
<td>( \frac{1}{2} \log N )</td>
<td>( \frac{1}{2} \log N )</td>
</tr>
<tr>
<td><strong>N solutions in ( \sqrt{N} ) front</strong></td>
<td>( \frac{1}{2} \log N )</td>
<td>( \frac{1}{2} \log N )</td>
<td>( \frac{1}{2} \log N )</td>
</tr>
<tr>
<td><strong>N solutions in ( \sqrt{N} ) front</strong></td>
<td>( \frac{1}{2} \log N )</td>
<td>( \frac{1}{2} \log N )</td>
<td>( \frac{1}{2} \log N )</td>
</tr>
</tbody>
</table>

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

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 \( O(M) \) 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>
<thead>
<tr>
<th>Scenario</th>
<th>Version-1</th>
<th>Version-2</th>
<th>Version-3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sequential Binary</td>
<td>Sequential Binary</td>
<td>Sequential Binary</td>
</tr>
<tr>
<td><strong>N solutions in single front</strong></td>
<td>( \frac{1}{2} M )</td>
<td>( \frac{1}{2} M )</td>
<td>( \frac{1}{2} M )</td>
</tr>
<tr>
<td><strong>N solutions in ( \sqrt{N} ) front</strong></td>
<td>( \frac{1}{2} M \log N )</td>
<td>( \frac{1}{2} M \log N )</td>
<td>( \frac{1}{2} M \log N )</td>
</tr>
<tr>
<td><strong>N solutions in ( \sqrt{N} ) front</strong></td>
<td>( \frac{1}{2} M \log N )</td>
<td>( \frac{1}{2} M \log N )</td>
<td>( \frac{1}{2} M \log N )</td>
</tr>
</tbody>
</table>

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

References


