2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)
October 11-14, 2020. Toronto, Canada

DDA-ENS: Dominance Degree Approach based Efficient Non-dominated Sort

Sumit Mishra
Department of CSE, IIIT Guwahati, India
sumit@iiitg.ac.in

Rakesh Senwar
Department of CSE, IIIT Guwahati, India
rsenwar448@gmail.com

***Abstract*—In Pareto based multi- and many-objective evolutionary algorithms (MOEAs and MaOEAs), non-dominated sorting is an important step that divides the set of solutions into different disjoint non-dominated fronts. In the last 20 years, there have been various approaches proposed for non-dominated sorting. Recently, an approach known as Dominance Degree Approach for Non-Dominated Sorting (DDA-NS) has been proposed. This approach guarantees that the number of floating-point comparisons is always bounded by $\mathcal{O}(MN\log N)$ for N solutions and M objectives, which is not true for various approaches. However, the worst-case time complexity of DDA-NS is recently proved to be $\Theta(MN^2+N^3)$. In this paper, we develop an approach on the top of DDA-NS, which also guarantees $\mathcal{O}(MN\log N)$ floating-point comparisons and its worst-case time complexity is proved to be $\Theta(MN^2)$ as opposed to $\Theta(MN^2+N^3)$ of DDA-NS.**

***Index Terms*—Non-dominated solutions, Time complexity, Number of floating-point comparisons**

I. Introduction

Non-dominated sorting is one of the essential steps in Pareto based MOEAs and MaOEAs [1], [2]. It has application in other domains like data clustering, graph theory, computer vision, economics and game theory, database and others [2]. Non-dominated sorting sorts the solutions into different non-dominated fronts which are arranged in decreasing order of their dominance. In MOEAs, the set of solutions are often referred to as the population. Let $\mathbb{P}=\{s_1,s_2,\ldots,s_N\}$ be a population of N solutions in M-dimensional objective space. i^{th} solution of $\mathbb{P}$; s_i is denoted as `P[i]`. A solution s_i in M-dimensional objective space is a vector of size M represented as $\langle s_{i_1},s_{i_2},\ldots,s_{i_M}\rangle$ where s_{i_m} is the objective value of solution s_i for m^{th} $(1\le m\le M)$ objective.

Without loss of generality, let us assume that all the objectives are of minimization type. In non-dominated sorting, the dominance relationship is defined as follows. A solution s_i is said to dominate another solution s_j, denoted by $s_i\prec s_j$ if both the following conditions are satisfied:

(i). $s_{i_m}\le s_{j_m},\ \forall m\in\{1,2,\ldots,M\}$
(ii). $s_{i_m}< s_{j_m},\ \exists m\in\{1,2,\ldots,M\}$

Here, a solution does not dominate itself as well as it does not dominate any other solution with identical objective values. Two solutions s_i and s_j are said to be non-dominated, if neither dominates the other, *i.e.*, neither $s_i\not\prec s_j$ nor $s_j\not\prec s_i$. In non-dominated sorting, population $\mathbb{P}$ of size N is divided into $K(1\le K\le N)$ dis-joint non-dominated fronts $\mathcal{F}=\{F_1,F_2,\ldots,F_K\}$ which are arranged in decreasing order of their dominance. The solutions are divided into these fronts such that all the following conditions are satisfied:

(i). $\cup_{k=1}^{K}F_k=\mathbb{P}$
(ii). $\forall s_i,s_j\in F_k$: $s_i\not\prec s_j$ and $s_j\not\prec s_i,\ 1\le k\le K$
(iii). $\forall s\in F_k,\ \exists s'\in F_{k-1}$: $s'\prec s,\ 2\le k\le K$

There have been various approaches proposed in the past to solve the non-dominated sorting problem. In general, there are two classes of non-dominated sorting algorithms – sequential, and divide-and-conquer algorithms. First, we discuss the sequential and then divide-and-conquer approaches.

Srinivas et al. [3] have developed an approach that compares a solution with other solutions several times and has a time complexity of $\mathcal{O}(MN^3)$. Deb et al. [1] have improved this $\mathcal{O}(MN^3)$ time complexity and proposed a fast non-dominated sorting approach that requires $\mathcal{O}(MN^2)$ time. After Deb's work, various other works have been proposed to reduce the number of comparisons between the solutions/objective values by identifying some of the relationships between the solutions. Deductive Sort [4] is one such approach where the transitive relationship has been explored. The best-case time complexity of this approach is $\mathcal{O}(MN\sqrt{N})$, and the worst-case time complexity (as claimed by the authors) is $\mathcal{O}(MN^2)$. However, recently its worst-case time complexity is proved to be $\mathcal{O}(MN^3)$ [5]. Wang et al. [6] have developed Corner Sort by utilizing another comparison saving strategy with worst-case time complexity $\mathcal{O}(MN^2)$.

A framework named Efficient Non-dominated Sort (ENS) has been proposed by Zhang et al. [7]. Two approaches based on ENS framework – ENS-SS (ENS with sequential search) and ENS-BS (ENS with binary search) have been proposed. The best-case time complexity of ENS-SS is $\mathcal{O}(MN\sqrt{N})$, and that of ENS-BS is $\mathcal{O}(MN\log N)$. By extending ENS-BS, an approach ENS-NDT (Efficient Non-dominated Sort based on Non-dominated Tree) [8] has been proposed to reduce the number of comparisons further. Roy et al. [9] proposed an approach named Best Order Sort (BOS), which reduces the number of comparisons between the solutions to a great extent. There are two main advantages of BOS – (i) It does not compare a solution with all the solutions of a particular front, and (ii) All the objective values are not compared to obtain the dominance relationship between the solutions. By extending the idea of BOS, some other approaches have been developed like [2], [10], [11]. In [10], the authors have handled the duplicate solutions which were not handled in original BOS. However, in this process, the second advantage of BOS has been sacrificed. To retain both the advantages of BOS, a generalized version of BOS (GBOS) has been proosed [11]. The authors of BOS also extended their approach and developed Bounded Best Order Sort [2]. Recently, Filter Sort is proposed to reduce the execution time of the algorithm. There is no complexity analysis done by the authors. Recently, Moreno et al. [12] have developed the Merge Non-dominated Sorting (MNDS) algorithm, which is written in Java and is very effective even for a small number of fronts. The worst-case time complexity of this approach is $\mathcal{O}(MN^2)$.

Divide-and-conquer approaches work by dividing the population and assign solutions to different fronts. Jensen et al. [13] started the work in this direction by extending the work of Kung et al. [14], which is used to find the first non-dominated front. The time complexity of Jensen's algorithm is $\mathcal{O}(N \log^{N-1} M)$. Jensen's algorithm is not able to handle the solutions that have the same value for a particular objective. This limitation has been removed in [15]. However, the limitation is removed at the cost of time, and the worst-case time complexity becomes $\mathcal{O}(MN^2)$. The limitation of Jensen's approach is removed without compromising its worst-case time complexity by Buzdalov et al. [16]. Recently, van Emde Boas Tree [17] based non-dominated sorting approach has been proposed [18]. This approach has a better worst-case time complexity, which is $\mathcal{O}(N \log^{M-2} N \log\log N)$. A different kind of divide-and-conquer approach is proposed in [19]–[21], where the objective space is not divided unlike [13], [15], [16], [18]. Divide-and-conquer based approach is developed by Fang *et al.* [19] using a hierarchical data structure named as a dominance tree. Mishra et al. [20] have developed a Divide-and-Conquer based Non-dominated Sorting (DCNS) framework. Two approaches based on this framework – DCNS-SS (DCNS with sequential search) and DCNS-BS (DCNS with binary search) have been proposed. The worst-case time complexity of both these approaches is $\mathcal{O}(MN^2)$. However, the best case time complexity of DCNS-BS is $\mathcal{O}(N \log N + MN)$.

There are various other approaches also in both categories. Recently, a method known as Dominance Degree Approach for Non-Dominated Sorting (DDA-NS) [22] has been proposed. The main advantage of this approach over other approaches is that it guarantees that the number of floating-point comparisons is always bounded by $\mathcal{O}(MN \log N)$, which is not true for various approaches. DDA-NS stores the dominance relationship between each pair of solutions in a $2D$ matrix known as the dominance degree matrix. The worst-case time complexity of DDA-NS, according to the original paper, is $\mathcal{O}(MN^2)$. However, Mishra et al. [23] have recently proved that the worst-case time complexity of DDA-NS is $\Theta(MN^2 + N^3)$ which occurs when N solutions are divided into N non-dominated fronts. The space complexity of DDA-NS is $\Theta(N^2)$. In this paper, we aim to improve the DDA-NS approach and propose a method that we call DDA-ENS

Algorithm 1 CONSTRUCT-COMPARISON-MATRIX(w)

```
Input: An $N$-dimensional row vector $w$
Output: Comparison matrix $\mathbb{C}_w$ corresponding to vector $w$
    /* Sort elements of $w$ in ascending order */
 1: [A, B] ← SORT($w$) ▷ A is the sorted vector in ascending order, and B is
    the index vector satisfying A[i] = $w$[B[i]]   $1 \le i \le N$
    /* Construct the comparison matrix $\mathbb{C}_w$ of size $N \times N$ */
 2: for $j \leftarrow 1$ to $N$ do
 3:     $\mathbb{C}_w$[B[1]][j] ← 1 ▷ Elements of the B[1]$^{th}$ row in $\mathbb{C}_w$ are all
        set to 1
 4: for $i \leftarrow 2$ to $N$ do
 5:     if A[i] = A[i-1] then
 6:         for $j \leftarrow 1$ to $N$ do
 7:             $\mathbb{C}_w$[B[i]][j] ← $\mathbb{C}_w$[B[i-1]][j]    ▷ Rows in $\mathbb{C}_w$
                corresponding to the same elements in $w$ are identical
 8:     else
 9:         for $j \leftarrow i$ to $N$ do
10:             $\mathbb{C}_w$[B[i]][B[j]] ← 1
11: return $\mathbb{C}_w$
```

(Dominance Degree Approach based Efficient Non-dominated Sort). The number of floating-point comparisons in DDA-ENS is also $\mathcal{O}(MN \log N)$, and the worst-case time complexity is $\Theta(MN^2)$. The space complexity of our method is also $\Theta(N^2)$.

Our proposed approach is thoroughly discussed in Section II. In this section, we have also given an example to illustrate the proposed approach, along with time and space complexity. The approach is experimentally verified in Section III. Finally, Section IV concludes the paper along with some direction for future research.

II. PROPOSED APPROACH

Our approach works in two phases as DDA-NS. In the first phase, a $2D$ matrix known as a dominance degree matrix is obtained, which stores the dominance relationship between each pair of solutions. This phase is the same as DDA-NS. In the second phase, the solutions are assigned to their corresponding fronts using the dominance degree matrix, and this phase is different from DDA-NS.

A. First Phase (Adapted from DDA-NS)

We discuss the concepts of DDA-NS which is used in our proposed approach. Initially, we discuss the dominance degree which is defined as follows.

Definition 1 (Dominance Degree). *Dominance degree $d(s_i, s_j)$ for two solutions s_i to s_j is the count of the number of element pairs (s_{i_m}, s_{j_m}) that satisfy $s_{i_m} \le s_{j_m}$. Formally, dominance degree $d(sol_i, sol_j)$ is defined as follows.* $d(s_i, s_j) = |\{i | i \in \{1, 2, \ldots, m\}, s_{i_m} \le s_{j_m}\}|$ *where $|\cdot|$ denotes the cardinality of a set.*

The value of dominance degree $0 \le d(s_i, s_j) \le M$. The maximum value of $d(s_i, s_j)$ occurs when either s_i dominates s_j or s_i and s_j have the same objective vector. The dominance degree matrix $\mathbb{D}$ is $N \times N$ size $2D$ matrix whose $\mathbb{D}$[i][j] entry is the dominance degree for solution s_j to solution s_i. The value of i^{th} row and j^{th} column in the dominance degree matrix, *i.e.*, $\mathbb{D}$[i][j] is equal to $d(s_i, s_j)$.

Algorithm 2 DOMINANCE-DEGREE-MATRIX($\mathbb{P}$)

```
Input: $\mathbb{P} = \{s_1, s_2, \dots, s_N\}$: Set of $N$ solutions in $M$-dimensional space
Output: Dominance degree matrix $\mathbb{D}$ of size $N \times N$
 1: $\mathbb{D} \leftarrow (0)_{N\times N}$          ▷ Initialize dominance degree matrix with zero
 2: for $j \leftarrow 1$ to $M$ do
 3:     $w_j \leftarrow \left\langle s_{1_j}, s_{2_j}, \dots, s_{N_j} \right\rangle$
 4:     $\mathbb{C}_{w_j} \leftarrow$ CONSTRUCT-COMPARISON-MATRIX $(w_j)$   ▷ Algorithm 1
 5:     $\mathbb{D} \leftarrow \mathbb{D} + \mathbb{C}_{w_j}$          ▷ Sum of comparison matrices
    /* For duplicate solutions, set the corresponding elements of $\mathbb{D}$ to zero */
 6: for $i \leftarrow 1$ to $N$ do
 7:     for $j \leftarrow i$ to $N$ do
 8:         if $\mathbb{D}$[i][j] $= M$ and $\mathbb{D}$[j][i] $= M$ then  ▷ Solutions $s_i$
            and $s_j$ are with identical objective vectors
 9:             $\mathbb{D}$[i][j] $\leftarrow 0$, $\mathbb{D}$[j][i] $\leftarrow 0$   ▷ Remove the dominance
10: return $\mathbb{D}$
```

We first discuss the approach (proposed in [22]) to compute the dominance degree matrix efficiently. For this purpose, N solutions with M objectives are considered as a matrix of size $M \times N$ where j^{th} column represents the objective vector of solution s_j. The comparison matrix is created for each of the M row vectors. For a row vector $\boldsymbol{w} = \langle w_1, w_2, \dots, w_N \rangle$ of size N, the comparison matrix $\mathbb{C}_w$ is a $N \times N$ matrix whose ij^{th} entry is defined as follows

$$\mathbb{C}_w\texttt{[i][j]} = \begin{cases} 1, & \text{if } w_i \leq w_j \\ 0, & \text{otherwise} \end{cases}$$

A naive approach to obtain the comparison matrix is to compare each pair of the elements in $\boldsymbol{w}$. However, obtaining the comparison matrix in this way requires $\Theta(N^2)$ floating-point comparisons. To reduce the number of floating-point comparisons, an efficient approach has been proposed in [22] and summarized in Algorithm 1. For this purpose, all the elements in the row vector are sorted in ascending order, which also gives an index vector of the same size. This index vector stores the order of the sorted elements in the original row vector. This index vector is the key to reduce the number of floating-point comparisons in the process of obtaining the comparison matrix. The sorting of the row vector takes $\mathcal{O}(N \log N)$ time. This time is spent on comparing floating-point numbers considering the objective values of the solutions are floating-point numbers. The comparison matrix $\mathbb{C}_w$ has a nice property. When two elements of the vector $\boldsymbol{w}$ are the same, then their corresponding two rows in $\mathbb{C}_w$ are identical. Hence, this property needs to be given attention while obtaining the comparison matrix. The comparison matrix requires $\Theta(N^2)$ time, considering the sorted row vector and index vector. Thus, the overall time complexity to obtain the comparison matrix is $\Theta(N^2)$.

As there are M row vectors of size N, so M comparison matrices are obtained. All these comparison matrices are added to get the dominance degree matrix. The process to obtain the dominance degree matrix is summarized in Algorithm 2. Obtaining a comparison matrix requires $\mathcal{O}(N \log N)$ floating-point comparisons, and there are M such comparison matrices. Thus, the number of floating-point comparisons in obtaining the dominance degree matrix is $\mathcal{O}(MN \log N)$. The time complexity to obtain a comparison matrix is $\Theta(N^2)$, and we require M comparison matrices to obtain the dominance degree matrix. Thus, the time complexity to obtain the dominance degree matrix is $\Theta(MN^2)$.

Algorithm 3 ENS-DDA($\mathbb{P}$)

```
Input: $\mathbb{P} = \{s_1, s_2, \dots, s_N\}$: Set of $N$ solutions in $M$-dimensional space,
    $\mathbb{D}$: Dominance degree matrix
Output: $\mathcal{F} = \{F_1, F_2, \dots, F_K\}$ set of $K$ fronts arranged in decreasing
    order of their dominance
 1: $\mathbb{D} \leftarrow$ DOMINANCE-DEGREE-MATRIX($\mathbb{P}$)   ▷ Construct the dominance
    degree matrix from the population using Algorithm 2
 2: Sort the solutions in $\mathbb{P}$ based on the first objective
 3: $\mathcal{F} \leftarrow \emptyset$                    ▷ Initialize the set of fronts
 4: $n_{\texttt{front}} \leftarrow 0$                   ▷ Number of fronts obtained
 5: for each solution $s \in \mathbb{P}$ do
 6:     INSERT($s, \mathcal{F}, n_{\texttt{front}}$)  ▷ Assign solution $s$ to $\mathcal{F}$ using Algorithm 4
```

B. Second Phase

In the second phase, the solutions are ranked. For this purpose, the solutions are sorted based on the first objective as in [7], [20]. Now the solutions are considered in this sorted order. The first solution is assigned to the first front. The second solution is compared with the first solution, and if this solution is non-dominated with the first solution, then this solution is added to the first front else a new front is created. The remaining solutions are considered in the sorted order and compared with already assigned solutions only. A particular solution is compared with the already ranked solutions sequentially starting from the first front. In this process, whenever the solution becomes non-dominated with all the solutions of a particular front, the solution is added to that front. In case the solution is dominated by all the existing fronts, then a new front is created, and the solution is inserted into the newly created front. The overall process of our approach is summarized in Algorithm 3, where a solution is inserted into its corresponding front using Algorithm 4. Now we discuss our approach using an example.

Algorithm 4 INSERT(s)

```
Input: $s$: Solution to be assigned, $\mathcal{F}$: Set of non-dominated fronts, $n_{\texttt{front}}$:
    Number of fronts obtained
Output: Updated set of fronts $\mathcal{F}$ after assigning solution $s$
 1: isInserted $\leftarrow$ FALSE  ▷ Solution $s$ is not yet inserted into any front
 2: for $k \leftarrow 1$ to $n_{\texttt{front}}$ do          ▷ Check each front sequentially
 3:     isDominated $\leftarrow$ FALSE      ▷ Solution $s$ is not dominated by the
        solutions of $F_k$
 4:     for each solution $s' \in F_k$ do
 5:         if $\mathbb{D}$[s'][s] $= M$ then                  ▷ $s'$ dominates $s$
 6:             isDominated $\leftarrow$ TRUE    ▷ Solution $s$ is dominated by $s'$
 7:             BREAK                              ▷ Check next front
 8:     if isDominated $=$ FALSE then  ▷ Solution $s$ is not dominated by
        the solutions of $F_k$
 9:         $F_k \leftarrow F_k \cup \{s\}$                     ▷ Insert $s$ into $F_k$
10:         isInserted $\leftarrow$ TRUE          ▷ Solution $s$ has been inserted
11:         BREAK
12: if isInserted $=$ FALSE then      ▷ $s$ is not yet inserted to any front
13:     $n_{\texttt{front}} \leftarrow n_{\texttt{front}} + 1$               ▷ Create a new front
14:     $F_{n_{\texttt{front}}} \leftarrow F_{n_{\texttt{front}}} \cup \{s\}$      ▷ Insert $s$ into newly created front
```

Example 1. *Let $\mathbb{P} = \{s_1, s_2, s_3, s_4, s_5, s_6\}$ be a set of six solutions in 3-dimensional objective space. These solu-*

tions are as follows: $s_1 = \langle 0.2031, 0.4031, 0.3946\rangle$*,* $s_2 = \langle 0.7894, 0.8041, 0.9640\rangle$*,* $s_3 = \langle 0.5678, 0.4940, 0.4947\rangle$*,* $s_4 = \langle 0.4940, 0.4954, 0.5494\rangle$*,* $s_5 = \langle 0.1343, 0.4131, 0.4113\rangle$ *and* $s_6 = \langle 0.2031, 0.4031, 0.3946\rangle$*. From these solutions with three objectives, we create three vectors, each of size six. These three vectors are as follows.*

$$w_1 = \langle 0.2031, 0.7894, 0.5678, 0.4940, 0.1343, 0.2031\rangle \quad (1)$$
$$w_2 = \langle 0.4031, 0.8041, 0.4940, 0.4954, 0.4131, 0.4031\rangle \quad (2)$$
$$w_3 = \langle 0.3946, 0.9640, 0.4947, 0.5494, 0.4113, 0.3946\rangle \quad (3)$$

We need to obtain the comparison matrices corresponding to all three vectors. To create the comparisons matrices, two other vectors are required ($\mathtt{A}$ *and* $\mathtt{B}$*). The process to obtain the three comparisons matrices are as follows.*

Comparison matrix $\mathbb{C}_{w_1}$ ***corresponding to row vector*** w_1***:***

$$\mathtt{A}_1 = \langle 0.1343, 0.2031, 0.2031, 0.4940, 0.5678, 0.7894\rangle$$
$$\mathtt{B}_1 = \langle 5, 1, 6, 4, 3, 2\rangle$$

$$\mathbb{C}_{w_1} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \quad (4)$$

Comparison matrix $\mathbb{C}_{w_2}$ ***corresponding to row vector*** w_2***:***

$$\mathtt{A}_2 = \langle 0.4031, 0.4031, 0.4131, 0.4940, 0.4954, 0.8041\rangle$$
$$\mathtt{B}_2 = \langle 1, 6, 5, 3, 4, 2\rangle$$

$$\mathbb{C}_{w_2} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (5)$$

Comparison matrix $\mathbb{C}_{w_3}$ ***corresponding to row vector*** w_3***:***

$$\mathtt{A}_3 = \langle 0.3946, 0.3946, 0.4113, 0.4947, 0.5494, 0.9640\rangle$$
$$\mathtt{B}_3 = \langle 1, 6, 5, 3, 4, 2\rangle$$

$$\mathbb{C}_{w_3} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (6)$$

The dominance degree matrix is obtained by adding all the three comparison matrices $\mathbb{C}_{w_1}$*,* $\mathbb{C}_{w_2}$ *and* $\mathbb{C}_{w_3}$*. Two solutions* s_1 *and* s_6 *share identical objective vectors. So we need to update the dominance degree matrix to handle the solutions with identical objective vectors. The normal and updated dominance degree matrix is given by Eq. (7). The element in RED color shows that these elements have been made* 0*.*

$$\mathbb{D} = \begin{bmatrix} 3 & 3 & 3 & 3 & 2 & 3 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 3 & 3 & 2 & 0 & 0 \\ 0 & 3 & 1 & 3 & 0 & 0 \\ 1 & 3 & 3 & 3 & 3 & 1 \\ 3 & 3 & 3 & 3 & 2 & 3 \end{bmatrix} \xrightarrow[solutions]{Identical} \begin{bmatrix} \mathbf{0} & 3 & 3 & 3 & 2 & \mathbf{0} \\ 0 & \mathbf{0} & 0 & 0 & 0 & 0 \\ 0 & 3 & \mathbf{0} & 2 & 0 & 0 \\ 0 & 3 & 1 & \mathbf{0} & 0 & 0 \\ 1 & 3 & 3 & 3 & \mathbf{0} & 1 \\ \mathbf{0} & 3 & 3 & 3 & 2 & \mathbf{0} \end{bmatrix} \quad (7)$$

With the help of this updated matrix, the dominance relationship between the solutions can be obtained in $\Theta(1)$ *time. Now we assign the solutions to their respective fronts. For this purpose, we need the sorted list of solutions based on the first objective. The solutions after sorting based on the first objective is as follows* $\langle s_5, s_1, s_6, s_4, s_3, s_2\rangle$*. The complete process to sort the solutions into different non-dominated fronts is shown in Fig. 1.*

Solution	Compared Solution	Final set of fronts
s_5	ϕ	$F_1 = \{s_5\}$
s_1	s_5	$F_1 = \{s_5, s_1\}$
s_6	s_5, s_1	$F_1 = \{s_5, s_1, s_6\}$
s_4	s_5	$F_1 = \{s_5, s_1, s_6\}$ $F_2 = \{s_4\}$
s_3	s_5, s_4	$F_1 = \{s_5, s_1, s_6\}$ $F_2 = \{s_4, s_3\}$
s_2	s_5, s_4	$F_1 = \{s_5, s_1, s_6\}$ $F_2 = \{s_4, s_3\}$ $F_3 = \{s_2\}$

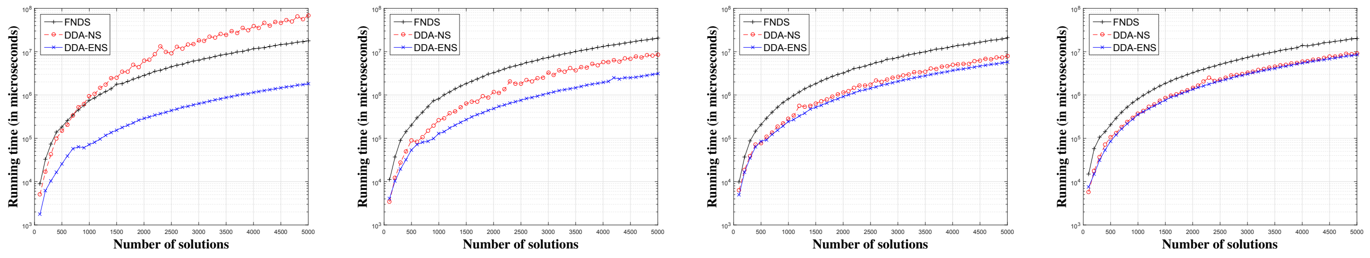
Fig. 1: The process to sort the solutions after obtaining the dominance degree matrix.

Complexity Analysis: The time complexity to obtain the dominance degree matrix is $\Theta(MN^2)$. The best-case time complexity to sort the solutions based on the first objective is $\mathcal{O}(N \log N)$ when all the solutions have a distinct value for the first objective. However, the worst-case time complexity is $\mathcal{O}(MN \log N)$, which occurs when all the solutions have the same objective vector. In this worst-case, the number of objective value comparisons is also $\mathcal{O}(MN \log N)$. The worst-case time complexity of our approach occurs when all the solutions are either in the same front or all the solutions are in different fronts. In the worst-case, a solution is compared with all the previously assigned solutions. Thus, the number of comparisons between the solutions in the worst-case is given by Eq. (8).

$$\#\mathtt{Comp} = \sum_{i=1}^{N} (i-1) = {}^{1}\!/_{2} N(N-1) = \Theta(N^2) \quad (8)$$

The time required to find the dominance relationship between two solutions is $\Theta(1)$ with the help of the dominance degree matrix. Thus, assigning the solutions to their respective fronts after sorting the solutions based on the first objective takes $\Theta(N^2)$ time given the dominance degree matrix. Hence, the worst-case time complexity of our approach is the sum of three factors – (i) time to compute the dominance degree matrix which is $\Theta(MN^2)$, (ii) time to sort the solutions based on the first objectives which take $\mathcal{O}(MN \log N)$ time and (iii) the time to assign the solutions to their respective fronts which takes $\Theta(N^2)$ time. Thus, the overall worst-case time complexity of our approach is $\Theta(MN^2)$. The number of floating-point comparisons by our approach is $\mathcal{O}(MN \log N)$.

The comparison matrix requires $\Theta(N^2)$ space. Thus, the space complexity to obtain the dominance degree matrix is also $\Theta(N^2)$. There is no need to obtain all the M comparison matrices and then add them all. We can obtain the first comparison matrix and add the subsequent comparison matrices in the first comparison matrix as performed in Algorithm 2 to make the process space-efficient. The space required to store



(a) $M = 2$ (b) $M = 5$ (c) $M = 10$ (d) $M = 15$

Fig. 2: Execution time of different non-dominated sorting approaches for cloud dataset where the number of solutions N varies between 100 to 5000 with an increment of 100 solutions. Different number of objectives $(2, 5, 10, 15)$ have been considered.

the sorted order of the solutions based on the first objective is $\Theta(N)$. The assignment of solutions to their respective fronts requires $\Theta(1)$ space apart from the final set of fronts. Thus, the space complexity of our approach is $\Theta(N^2)$, which is the same as DDA-NS.

III. Experimental Evaluation

In this section, we experimentally verify our approach DDA-ENS with DDA-NS and the most popular non-dominated sorting algorithm – Fast Non-dominated Sort (FNDS) [1]. The corresponding experiments are carried out on a Ubuntu 19.04 PC with a 1.70 GHz Intel(R) Core(TM) i5-4210U CPU, 8 GB RAM and 64 bit operating system. For experimental purposes, we have considered two types of datasets – Cloud dataset and Fixed front dataset, as in [7], [9], [20].

A. *Cloud dataset*

In this dataset, the solutions are randomly sampled from a uniform distribution on the interval $[0, 1]$ [6], [9]. As the solutions are randomly generated, the number of fronts, and the number of solutions in the front is random. This kind of dataset is used as it mimics the initial stages of MOEAs. For our experiment, we have varied the number of solutions from 100 to 5000 with an increment of 100. Four different number of objectives – 2, 5, 10 and 15 are considered.

The running time for the cloud dataset is shown in Fig 2. The runtime for bi-objective case is shown in Fig. 2(a), whereas Fig. 2(b) shows the execution time for 5 objectives. The runtime for 10 and 15 objectives is shown in Fig. 2(c) and 2(d) respectively. From all these figures, it is clear that the runtime for DDA-ENS is less than DDA-NS. However, as the number of objectives increases, the runtime of both these approaches is almost the same. This is because for a random set of solutions, as the number of objectives increases, the number of fronts starts decreasing [24], [25]. This is visible from Fig. 4 where we have shown the number of fronts for the same set of solutions for which we have computed the runtime considering 2, 5, 10, and 15 objectives.

DDA-NS moves towards its best case when the number of fronts starts decreasing, and its best-case occurs when the number of fronts is 1 (time complexity $\Theta(MN^2)$), and it's worst-case occurs when the number of fronts is N (time complexity $\Theta(MN^2 + N^3)$) [23]. Thus, when the number of objectives increases, the performance of DDA-NS and our approach behaves similarly. The time required by fast non-dominated sorting [1] is more than the other two approaches when the number of fronts is 5, 10 and 15. There is an overlap in the time of DDA-NS and fast non-dominated sorting for the bi-objective case. The reason for this is that the DDA-NS moves towards its worst-case with an increase in the number of fronts, and for K fronts, it requires $\Theta(MN^2 + KN^2)$ time [23]. When the number of fronts is large in number, even though the floating-point comparisons are less, but the integer comparison takes more time as it needs to be performed multiple times.

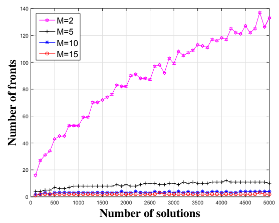


Fig. 4: Number of fronts for each of the 50 population considering all the four objectives for cloud dataset.

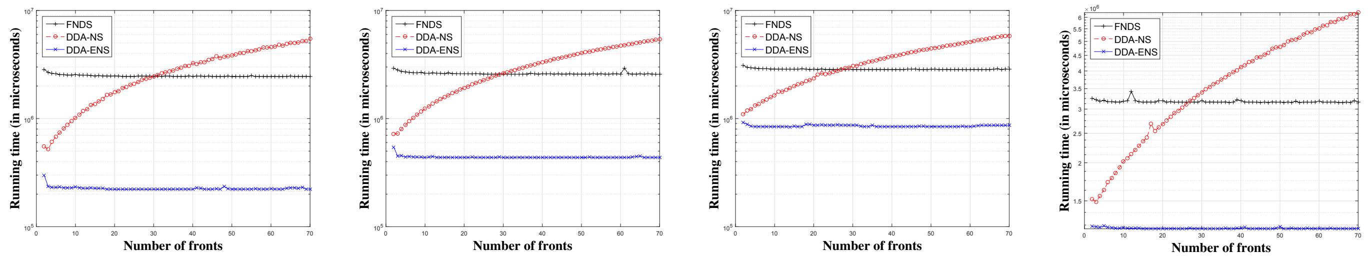
B. *Fixed front dataset*

In this dataset, the solutions are equally divided into K number of fronts. This division is such that each solution in a front is dominated by all the solutions in its preceeding front. The number of solutions in initial $K - 1$ fronts is $\lfloor N/K \rfloor$ and the number of solutions in the last front is $N - \lfloor N/K \rfloor \cdot (K - 1)$. For our experiment, we have fixed the number of solutions to be 2000, as considered in [7]. The number of fronts has been varied from 2 to 70 with an increment of 1. Here also, four different number of objectives – 2, 5, 10 and 15 are considered.

The running time for the fixed front dataset is shown in Fig 3. From this figure, it is clear that the runtime for DDA-ENS is less than the other two approaches. As the number of fronts increases, the runtime of DDA-NS increases. The overall description of the experiments, including dataset and the codes, is available at Github: `https://github.com/rsenwar/Non-Dominated-Sorting-Algorithms`.

IV. Conclusion & Future Work

In this paper, we have proposed an approach for non-dominated sorting, which guarantees the number of floating-



(a) $M = 2$ (b) $M = 5$ (c) $M = 10$ (d) $M = 15$

Fig. 3: Execution time of different non-dominated sorting approaches for fixed front dataset where the number of solutions is fixed to 2000, and the number of fronts varies between 2 to 70 with an increment of 1. Four different number of objectives – $2, 5, 10, 15$ have been considered.

point comparisons to be $\mathcal{O}(MN \log N)$ with worst-case time complexity $\Theta(MN^2)$. This approach is beneficial for the cases when comparing the objective values of the solutions is an expensive operation. This approach reduces the number of floating-point comparisons to a great extent. Thus, in the future, we would like to incorporate this approach with some efficient methods like Corner Sort [6] where the solutions of the fronts are obtained together so that the overall running time of the algorithm can be reduced.

References

[1] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, April 2002.

[2] P. C. Roy, K. Deb, and M. M. Islam, "An Efficient Nondominated Sorting Algorithm for Large Number of Fronts," *IEEE Transactions on Cybernetics*, vol. 49, no. 3, pp. 859–869, 2019.

[3] N. Srinivas and K. Deb, "Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms," *Evolutionary Computation*, vol. 2, no. 3, pp. 221–248, Fall 1994.

[4] K. McClymont and E. Keedwell, "Deductive Sort and Climbing Sort: New Methods for Non-Dominated Sorting," *Evolutionary Computation*, vol. 20, no. 1, pp. 1–26, Spring 2012.

[5] S. Mishra and M. Buzdalov, "If Unsure, Shuffle: Deductive Sort is $\Theta(MN^3)$, but $O(MN^2)$ in Expectation over Input Permutations," in *Proceedings of Genetic and Evolutionary Computation Conference*, 2020, pp. 516–523.

[6] H. Wang and X. Yao, "Corner Sort for Pareto-Based Many-Objective Optimization," *IEEE Transactions on Cybernetics*, vol. 44, no. 1, pp. 92–102, January 2014.

[7] X. Zhang, Y. Tian, R. Cheng, and J. Yaochu, "An Efficient Approach to Nondominated Sorting for Evolutionary Multiobjective Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 2, pp. 201–213, April 2015.

[8] P. Gustavsson and A. Syberfeldt, "A New Algorithm Using the Non-Dominated Tree to Improve Non-Dominated Sorting," *Evolutionary Computation*, vol. 26, no. 1, pp. 89–116, September 2018.

[9] P. C. Roy, M. M. Islam, and K. Deb, "Best Order Sort: A New Algorithm to Non-Dominated Sorting for Evolutionary Multi-Objective Optimization," in *Proceedings of Genetic and Evolutionary Computation Conference*. Denver, Colorado, USA: ACM Press, July 20-24 2016, pp. 1113–1120, ISBN: 978-1-4503-4323-7.

[10] S. Mishra, S. Saha, and S. Mondal, "MBOS: Modified Best Order Sort Algorithm for Performing Non-dominated Sorting," in *2018 IEEE Congress on Evolutionary Computation (CEC'2018)*. Rio de Janeiro, Brazil: IEEE Press, July 8–13 2018, pp. 725–732, ISBN: 978-1-5090-6017-7.

[11] S. Mishra, S. Mondal, S. Saha, and C. A. Coello Coello, "GBOS: Generalized Best Order Sort Algorithm for Non-Dominated Sorting," *Swarm and Evolutionary Computation*, vol. 43, pp. 244–264, December 2018.

[12] J. Moreno, D. Rodriguez, A. J. Nebro, and J. A. Lozano, "Merge Nondominated Sorting Algorithm for Many-Objective Optimization," *IEEE Transactions on Cybernetics*, 2020, accepted for publication.

[13] M. T. Jensen, "Reducing the Run-Time Complexity of Multiobjective EAs: The NSGA-II and Other Algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 5, pp. 503–515, October 2003.

[14] H.-T. Kung, F. Luccio, and F. P. Preparata, "On Finding the Maxima of a Set of Vectors," *Journal of the ACM (JACM)*, vol. 22, no. 4, pp. 469–476, 1975.

[15] F.-A. Fortin, S. Greiner, and M. Parizeau, "Generalizing the Improved Run-Time Complexity Algorithm for Non-Dominated Sorting," in *Proceedings of Genetic and Evolutionary Computation Conference*. New York, USA: ACM Press, July 2013, pp. 615–622, ISBN: 978-1-4503-1963-8.

[16] M. Buzdalov and A. Shalyto, "A Provably Asymptotically Fast Version of the Generalized Jensen Algorithm for Non-Dominated Sorting," in *Parallel Problem Solving from Nature - PPSN XIII, 13th International Conference*. Ljubljana, Slovenia: Springer. Lecture Notes in Computer Science Vol. 8672, September 13-17 2014, pp. 528–537.

[17] P. van Emde Boas, "Preserving Order in a Forest in Less Than Logarithmic Time," in *16th Annual Symposium on Foundations of Computer Science (SFCS 1975)*. IEEE, 1975, pp. 75–84.

[18] M. Buzdalov, "Make Evolutionary Multiobjective Algorithms Scale Better with Advanced Data Structures: Van Emde Boas Tree for Non-Dominated Sorting," in *10th International Conference on Evolutionary Multi-Criterion Optimization, EMO 2019*. East Lansing, MI, USA: Springer. Lecture Notes in Computer Science Vol. 11411, March 10–13 2019, pp. 66–77, ISBN: 978-3-030-12597-4.

[19] H. Fang, Q. Wang, Y.-C. Tu, and M. F. Horstemeyer, "An Efficient Non-Dominated Sorting Method for Evolutionary Algorithms," *Evolutionary Computation*, vol. 16, no. 3, pp. 355–384, Fall 2008.

[20] S. Mishra, S. Saha, and S. Mondal, "Divide and Conquer Based Non-Dominated Sorting for Parallel Environment," in *2016 IEEE Congress on Evolutionary Computation (CEC'2016)*. Vancouver, Canada: IEEE Press, 24-29 July 2016, pp. 4297–4304, ISBN: 978-1-5090-0623-6.

[21] S. Mishra, S. Saha, S. Mondal, and C. A. Coello Coello, "A Divide-And-Conquer Based Efficient Non-Dominated Sorting Approach," *Swarm and Evolutionary Computation*, vol. 44, pp. 748–773, February 2019.

[22] Y. Zhou, Z. Chen, and J. Zhang, "Ranking Vectors by Means of the Dominance Degree Matrix," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 1, pp. 34–51, 2017.

[23] S. Mishra, M. Buzdalov, and R. Senwar, "Time Complexity Analysis of the Dominance Degree Approach for Non-Dominated Sorting," in *Proceedings of Genetic and Evolutionary Computation Conference Companion*, 2020, pp. 169–170.

[24] K. Li, K. Deb, Q. Zhang, and Q. Zhang, "Efficient Nondomination Level Update Method for Steady-State Evolutionary Multiobjective Optimization," *IEEE Transactions on Cybernetics*, vol. 47, no. 9, pp. 2838–2849, 2016.

[25] S. Mishra, S. Mondal, and S. Saha, "Improved Solution to the Non-domination Level Update Problem," *Applied Soft Computing*, vol. 60, pp. 336–362, 2017.