P-ENS: Parallelism in Efficient Non-dominated Sorting

Sumit Mishra
Department of Computer Science
CINVESTAV-IPN
Mexico City 07360, Mexico
Email: smishra@computacion.cs.cinvestav.mx

Carlos A. Coello Coello
Department of Computer Science
CINVESTAV-IPN
Mexico City 07360, Mexico
Email: ccoello@cs.cinvestav.mx

***Abstract*—In recent years, several non-dominated sorting approaches have been proposed. Non-dominated sorting is an essential part of Pareto dominance-based multi-objective evolutionary algorithms (MOEAs) and therefore the relevance of being able to perform such process as efficiently as possible. As the use of parallelism has become increasingly popular within MOEAs, there is an evident need for parallel implementations of non-dominated sorting algorithms. In this paper, we have focused on an efficient non-dominated sorting (ENS) approach and explored its parallelization. The time complexity of the parallel version of ENS is theoretically analyzed in four different scenarios.**

***Keywords*: Non-dominated sorting (NDS), Evolutionary algorithms (EA), Parallelism.**

I. Introduction

Multi-objective optimization evolutionary algorithms (MOEAs) are used by many researchers to solve many-objective optimization problems. Non-dominated sorting is one of the most important steps in Pareto dominance-based MOEAs and it consists of a process that classifies solutions into different non-dominated fronts. Let there be N solutions in the population $\mathbb{P}$. Let population $\mathbb{P} = \{sol_1, sol_2, \ldots, sol_N\}$ where each solution has M objectives associated with it, *i.e.*, these solutions are in an M-dimensional objective space. A solution *sol* of population $\mathbb{P}$ in M-dimensional objective space is represented as follows: $sol = \{f_1(sol), f_2(sol), \ldots, f_M(sol)\}$ where $f_m(sol)$ is the value of *sol* for the $m^{th}(1 \leq m \leq M)$ objective. To sort the solutions into different fronts, first their dominance relationship needs to be established. Here, we are considering minimization problems. A solution sol_i in the population dominates another solution sol_j denoted as $sol_i \prec sol_j$ if it satisfies the two following conditions:

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

The notation $sol_i \not\prec sol_j$ represents that solution sol_i does not dominate solution sol_j. We say that two solutions sol_i and sol_j are non-dominated or non-comparable when none of the solutions dominate the other one, *i.e.*, $sol_i \not\prec sol_j$ and $sol_j \not\prec sol_i$. Now, we provide the formal definition of non-dominated sorting.

Definition 1 (Non-dominated sorting). *Non-dominated sorting divides the set of N solutions $\{sol_1, sol_2, \ldots, sol_N\}$ in the population $\mathbb{P}$ into $K(1 \leq K \leq N)$ different non-dominated fronts $\{F_1, F_2, \ldots, F_K\}$ classified in decreasing order of their dominance. The division of the solutions in these fronts is such that*

- *No solution in a front dominates others, i.e., $\forall sol_i, sol_j \in F_k$: $sol_i \not\prec sol_j$ and $sol_j \not\prec sol_i$ $(1 \leq k \leq K)$*
- *Each solution in a front is dominated by at least one of the solutions in its preceding front, i.e., $\forall sol \in F_k$, $\exists sol' \in F_{k-1}$: $sol' \prec sol$ $(2 \leq k \leq K)$*

There are different approaches proposed for NDS [1]–[17] . Some of these approaches have also been parallelized. Specifically, the approaches described in [9], [11], [15] focus on the parallelization of a the very popular NDS approach developed by Deb *et al.* [2]. However, the parallelism property exists in other approaches such as the naive approach [1], ENS [10], DCNS [13], BOS [12], etc. This paper explores the scope of parallelism in ENS. The time complexity of the serial and the parallel versions of ENS is also theoretically analyzed in four different scenarios.

The rest of this paper is organized as follows. Some of the approaches that have been proposed for non-dominated sorting are briefly discussed in Section II. The serial version of ENS along with its time complexity in four scenarios is discussed in Section III. Parallelism in ENS is illustrated in Section IV. The time complexity in four scenarios in which the serial version of ENS is analyzed, is also discussed in this section. Finally, Section V concludes the paper and also provides some possible paths for future research.

II. Related Work

The past few years have witnessed different approaches for non-dominated sorting. One of the earliest algorithms for non-dominated sorting was proposed by Srinivas *et al.* [1] and is now considered as a naive approach where a solution can be compared with other solutions multiple times. The worst case time complexity of this naive approach is $\mathcal{O}(MN^3)$ when N solutions are in N different fronts and the best case time complexity is $\mathcal{O}(MN^2)$ when N solutions are in a single front. The space complexity of this naive approach is $\mathcal{O}(N)$. To improve the time complexity of this naive approach, Deb *et al.* [2] proposed the so-called fast non-dominated sorting,

which has a time complexity $\mathcal{O}(MN^2)$ and a space complexity $\mathcal{O}(N^2)$. Jensen *et al.* [3] proposed a divide-and-conquer based approach with a time complexity $\mathcal{O}(N \log^{M-1} N)$. For the bi-objective case, the time complexity of Jensen's approach is $\mathcal{O}(N \log N)$. The main limitation of Jensen's approach is that it is not suitable when solutions share the same value for any of the objectives. Similarly to Jensen's approach, Fang *et al.* [4] also proposed a divide-and-conquer based approach with a worst case time complexity $\mathcal{O}(MN^2)$ and a best case time complexity $\mathcal{O}(MN \log N)$. However, in case of having two duplicate solutions, one of them is considered to be dominated by the other one. Tang *et al.* [5] proposed an approach based on arena's principle with a worst case time complexity $\mathcal{O}(MN^2)$ and a best case time complexity $\mathcal{O}(MN\sqrt{N})$ [10].

Deductive and Climbing sort were developed by McClymont *et al.* [6]. In general, deductive sort performs better than climbing sort with a worst case time complexity $\mathcal{O}(MN^2)$ and a best case time complexity $\mathcal{O}(MN\sqrt{N})$. Wang *et al.* [18] proposed an approach known as corner sort specifically for many-objective cases. The worst case time complexity of this approach is $\mathcal{O}(MN^2)$. The limitation of Jensen's approach is removed by Fortin's approach [7]. Removing the limitation increases the worst case time complexity to $\mathcal{O}(MN^2)$, however, the average case time complexity remains the same as in Jensen's approach. Furthermore, the time complexity of non-dominated sorting has been proved to be $\mathcal{O}(N \log^{M-1} N)$ by Buzdalov *et al.* [8].

Mishra *et al.* [13] developed a divide-and-conquer based approach named as DCNS. This approach works in two phases. The first phase sorts the solutions based on a particular objective and the second phase assigns rank to the solutions. The worst case time complexity of DCNS is $\mathcal{O}(MN^2)$ and the best case time complexity is $\mathcal{O}(MN \log N)$. A Hierarchical Non-dominated Sorting known as HNDS was recently proposed by Bao *et al.* [17]. The worst case time complexity of HNDS is $\mathcal{O}(MN^2)$ and the best case time complexity is $\mathcal{O}(MN\sqrt{N})$. Roy *et al.* [12] proposed an efficient approach in terms of the number of dominance comparisons which is known as Best Order Sort (BOS). This approach also works in two phases. The first phase sorts solutions based on each of the M objectives and the second phase assigns rank to the solutions. The worst case time complexity of BOS is $\mathcal{O}(MN^2)$ and the best case time complexity is $\mathcal{O}(MN \log N)$. Zhang *et al.* [14] developed a tree-based approach known as T-ENS. The fronts are represented in the form of a tree to save unnecessary dominance comparisons. The worst case time complexity of T-ENS is $\mathcal{O}(MN^2)$ and the best case time complexity is $\mathcal{O}(MN \log N / \log M)$. This approach also suffers from the same limitation as Jensen's approach [16]. Gustavsson *et al.* [16] proposed an Efficient Non-dominated Sort based on Non-dominated Tree (ENS-NDT). The worst case time complexity of ENS-NDT is $\mathcal{O}(MN^2)$ and the best case time complexity is $\mathcal{O}(MN \log N)$. This approach reduces the number of dominance comparisons to a great extent. This approach is also able to handle duplicate solutions efficiently.

There are some other approaches also proposed for NDS [19]–[26]. In the recent past, authors have also focused on the incremental version of the non-dominated sorting problem where a set of non-dominated fronts is updated when a new solution is inserted. Drozdik *et al.* [27] developed an approach based on M-fronts with a worst case time complexity $\mathcal{O}(MN^2)$. For the two-objective case, an approach was developed by Buzdalov *et al.* [28] with a time complexity $\mathcal{O}(N)$. For the general case, Li *et al.* [29] developed an approach with a worst case time complexity $\mathcal{O}(MN^2)$ and a best case time complexity $\mathcal{O}(M)$. However, the maximum number of dominance comparisons is $\frac{1}{4}N^2$. There is a great improvement over the best case time complexity of the NDS approaches. Mishra *et al.* [30] proposed a generalized approach with constant space complexity. To further improve the performance in some cases, a dominance binary tree based approach is discussed. Recently, Yakupov *et al.* [31] proposed an approach with time complexity $\mathcal{O}(N \log N^{M-2})$.

Smutnicki *et al.* [9] proposed a Very Fast Non-Dominated Sorting which explores the scope of parallelism in fast non-dominated sort [2]. Parallelism is explored in two different manners. The time complexity of the first version of the parallel algorithm is $\mathcal{O}(M+N \log N)$ and the time complexity of the second version is $\mathcal{O}(M+N)$. A parallel non-dominated sort based on the use of GPUs was proposed by Gupta *et al.* [11]. Ortega *et al.* [15] developed three parallel versions of non-dominated sorting. The first one is based on multicores, the second one is based on Graphical Processing Units (GPUs) and the third one is based on multicores and GPUs.

III. Efficient Non-dominated Sorting

Efficient non-dominated sort (ENS) was developed by Zhang *et al.* [10]. ENS works in two phases. The first phase sorts the solutions based on the first objective. In the second phase, different sets of fronts are obtained. Algorithm 1 summarizes the flow of the ENS approach.

Algorithm 1 ENS framework

Input: $\mathbb{P}$: Population of size N where each solution is associated with M objectives

Output: Non-dominated fronts in sorted order

1: Sort population based on the first objective in ascending order
2: $\mathcal{F} \leftarrow \Phi$ *// Initialize the set of fronts*
3: **for** each solution $sol \in \mathbb{P}$ **do** *// Consider the solution sequentially from the sorted list*
4: INSERT-SS$(\mathcal{F}, sol)$ *// Sequential search based strategy is used. A binary search based strategy can also be used*
5: **return** $\mathcal{F}$

A. First Phase: Pre-sorting

During pre-sorting, solutions are sorted based on the first objective. In spite of sorting the solutions based on the first objective, the solutions can be sorted based on other objectives as well. When the solutions are sorted based on the first

objective, if the objective values of two solutions for the first objective are the same, then, the objective values of the second objective are considered. Similarly, if the objective values of two solutions for the second objective are the same, then, the objective values of the third objective are considered. If two solutions have the same values for all the objectives, then any order of the solutions can be considered.

In the sorted list of solutions, the solution which comes later cannot dominate the solutions which come earlier in the list. So, when the solutions in this list are compared then the solutions which come later cannot dominate the previous solutions. Thus, it can save unnecessary dominance comparisons.

B. Second Phase: Rank Assignment

In the second phase, the rank is assigned to the solutions. For this purpose, the solutions are considered from the sorted list of solutions based on the first objective. Initially, the first solution is assigned to the first front as it cannot be dominated by any other solution in the sorted list. In this approach, the solution is only compared with those solutions which have been already assigned to their respective front. In the ENS approach, a solution can be assigned to their respective front using two search techniques – sequential search and binary search.

1) Sequential Search based Strategy: In a sequential search based strategy, a solution *sol* is compared with the solutions of existing fronts in a sequential manner, starting from the first front. Initially, *sol* is compared with the first front solutions and if it is non-dominated with these solutions, then *sol* is assigned to the first front. If *sol* is dominated by any of the solutions of F_1, then it is compared with the solutions of the second front F_2. In general, if *sol* is dominated by any of the solutions of the kth front, then *sol* is compared with the solutions of the $k+1^{th}$ front. If *sol* is dominated by at least one of the solutions of all the existing fronts, then *sol* is inserted into the newly created front. The sequential search based strategy is provided in Algorithm 2. Since a sequential search based strategy is adopted in this case, this variant of ENS is known as ENS-SS.

2) Binary Search based Strategy: In a binary search based strategy, a solution *sol* is not compared with the solutions of all the existing solutions, as happens when using a sequential search based strategy. In this case, the set of fronts is visualized as a dominance binary search tree. In a dominance binary search tree, a node has lower dominance than its left sub-tree and a higher dominance than its right sub-tree [30]. The binary search based strategy is summarized in Algorithm 3. Let there be K fronts which have been identified. So, here *sol* is compared with the maximum of $\log K$ fronts. As the approach is based on a binary tree structure, we have used three variables to follow the tree structure – *min*, *max* and *mid*; *mid* represents the root of the sub-tree. Initially, $min = 1$ and $max = K$ so *sol* is compared with the F_{mid} front where $mid = \lceil (min+max)/2 \rceil$. If *sol* is non-dominated with each of the solutions of F_{mid}, then there are two possibilities:

Algorithm 2 INSERT-SS($\mathcal{F}$, *sol*)

Input: $\mathcal{F}$: Set of fronts $\{F_1, F_2, \ldots, F_K\}$, *sol*: Solution for insertion in $\mathcal{F}$
Output: Updated set of fronts

```
done ← FALSE                                   // Initially sol is not inserted
K ← |F|                                        // Obtain the number of fronts
for k ← 1 to K do   // Check for each front in F sequentially starting from the first front
    flag ← TRUE
    for u ← n_k down to 1 do   // Check for each solution in F_k sequentially starting from the last solution
        if sol is dominated by F_k(u) then
            flag ← FALSE
            BREAK                              // Check for next front
    if flag = TRUE then   // sol is non-dominated with F_k
        F_k ← F_k ∪ {sol}                      // Insert sol in F_k
        isInserted ← TRUE                      // sol has been inserted
        BREAK
if isInserted = FALSE then   // sol has not been inserted yet
    F_{K+1} ← F_{K+1} ∪ {sol}                  // Insert sol in a new front
```

- If $mid = min$, *i.e.*, *sol* is non-dominated with respect to each of the solutions of the leaf node of the dominance tree, then insert *sol* in F_{mid}.
- Otherwise, *sol* can be non-dominated with respect to solutions of a higher dominance front. So, the root of the left sub-tree is explored.

If *sol* is dominated by any of the solutions of F_{mid}, then there can be one of the following three conditions:

- If $min = K$, *i.e.*, *sol* is dominated by the solutions in the last front, then *sol* will create a new front F_{K+1}.
- If $max = mid + 1$, *i.e.*, *sol* is dominated by the leaf node, then insert *sol* into F_{max}.
- Otherwise, *sol* is compared with the root of the right sub-tree.

Since a this binary search based strategy is adopted in this case, this variant of ENS is known as ENS-BS.

C. Complexity Analysis

Now, we discuss the time complexity of both variants of ENS in four different scenarios.

1) All the solutions are in a single front: Here, ENS-SS and ENS-BS perform the same. In this case, the ith solution is compared with all the previous $i-1$ ranked solutions. Thus, the time complexity in this case is given by Eq. 1. This is the worst case time complexity of both ENS-SS and ENS-BS.

$$\begin{aligned} T_1 &= 1 + \sum_{i=2}^{N} [M(i-1)+1] \\ &= 1 + M \sum_{i=2}^{N} (i-1) + \sum_{i=2}^{N} 1 \\ &= 1 + M\frac{1}{2}N(N-1) + N - 1 = \frac{1}{2}MN(N-1) + N \\ &= \mathcal{O}(MN^2) \end{aligned} \tag{1}$$

Algorithm 3 INSERT-BS($\mathcal{F}, sol$)

Input: $\mathcal{F}$: Set of fronts $\{F_1, F_2, \ldots, F_K\}$, *sol*: Solution for insertion in $\mathcal{F}$

Output: Updated set of fronts

```
K ← |F|                                   // Obtain the number of fronts
min ← 1
max ← K
mid ← ⌊(1+K)/2⌋
while TRUE do                             // sol has not been inserted yet
    flag ← TRUE
    for u ← n_mid down to 1 do            // Check for each solution in F_mid sequentially starting from the last solution
        if sol is dominated by F_mid(u) then
            flag ← FALSE
            BREAK                         // sol cannot be inserted in F_mid
    if flag = TRUE then                   // sol is non-dominated with F_mid
        if mid = min then                 // The front at leaf is explored
            F_mid ← F_mid ∪ {sol}         // Insert sol in F_mid
            BREAK                         // sol has been inserted
        else
            max ← mid, mid ← ⌊(min+max)/2⌋ // Explore left sub-tree
    else
        if min = K then                   // The front at right most leaf is explored
            F_{K+1} ← F_{K+1} ∪ {sol}     // Insert sol in a new front
            BREAK                         // sol has been inserted
        else if max = mid + 1 then
            F_max ← F_max ∪ {sol}         // Insert sol in F_max
            BREAK                         // sol has been inserted
        else
            min ← mid + 1, mid ← ⌊(min+max)/2⌋ // Explore right sub-tree
```

2) All the solutions are in different fronts: In this case, ENS-SS and ENS-BS perform differently. In case of ENS-SS, the ith solution is compared with the previous $i-1$ ranked solutions. Thus, the time complexity of ENS-SS is given by Eq. 1. In case of ENS-BS, the ith solution is compared with the previous $\lceil \log i \rceil$ ranked solutions. Thus, the time complexity of ENS-BS is given by Eq. 2. This is the best case time complexity of ENS-BS.

$$\begin{aligned} T_{1_{\text{BS}}} &= 1 + \sum_{i=2}^{N} \left[M \lceil \log i \rceil + 1 \right] \\ &= 1 + M \sum_{i=2}^{N} \lceil \log i \rceil + \sum_{i=2}^{N} 1 \\ &= 1 + MN \log N - M(N-1) + (N-1) \\ &= \mathcal{O}(MN \log N) \end{aligned} \tag{2}$$

3) Equal division of N solutions in $\sqrt{N}$ fronts such that all the solutions in a front are dominated by each of the solutions in its preceding front: Here, ENS-SS and ENS-BS perform differently as the number of fronts is $\sqrt{N}$. In this case, for a solution to be assigned to the kth front – (a) it should be dominated by at least one of the solutions of the previous $k-1$ fronts and (b) it should be non-dominated with respect to all the previous solutions of the kth front. A solution which needs to be assigned to the kth front, is dominated by the first solution of all the previous fronts with respect to which it is compared. This is because each solution in a front is dominated by each of the solutions in its preceding front. Let the time complexity corresponding to (a) be T_{1A} and the one corresponding to (b) be T_{1B}.

In case of ENS-SS, a solution is assigned to F_k if (a) it is dominated by the first solution of all the previous $k-1$ fronts and (b) it is non-dominated with respect to each of the solutions in F_k. The time complexity of ENS-SS is obtained using Eq. 3. This is the best case time complexity of ENS-SS.

$$\begin{aligned} T_{1_{\text{SS}}} &= T_{1A} + T_{1B} \\ &= \sum_{k=2}^{\sqrt{N}} \sqrt{N} M(k-1) + \sqrt{N} \left[1 + \sum_{i=2}^{\sqrt{N}} M(i-1) + 1 \right] \\ &= MN(\sqrt{N}-1) + N \\ &= \mathcal{O}(MN\sqrt{N}) \end{aligned} \tag{3}$$

In the case of ENS-BS, a solution is assigned to F_k if (a) is dominated by the first solution of the previous $\lceil \log k \rceil$ fronts and (b) is non-dominated with respect to each of the solutions in F_k. The time complexity of ENS-BS is obtained using Eq. 4.

$$\begin{aligned} T_{1_{\text{BS}}} &= T_{1A} + T_{1B} \\ &= \sum_{k=2}^{\sqrt{N}} \sqrt{N} M \left(\lceil \log k \rceil \right) + \sqrt{N} \left[1 + \sum_{i=2}^{\sqrt{N}} M(i-1) + 1 \right] \\ &= \frac{1}{2} MN \log N - M(N - \sqrt{N}) + \frac{1}{2} MN(\sqrt{N}-1) + N \\ &= \mathcal{O}(MN\sqrt{N}) \end{aligned} \tag{4}$$

4) Equal division of N solutions in $\sqrt{N}$ fronts such that all the solutions in a front are dominated by only one solution in its preceding front: Here, ENS-SS and ENS-BS perform differently as the number of fronts is $\sqrt{N}$. In this case, for a solution to be assigned to the kth front – (a) it should be dominated by at least one of the solutions of the previous $k-1$ fronts and (b) it should be non-dominated with respect to all the previous solutions of the kth front. A solution which needs to be assigned to the kth front, is dominated by the last solution of all the previous fronts with respect to which it is compared. This is because each solution in a front is dominated by only one solution in its preceding front and for the number of dominance comparisons to be maximum, the solution should be dominated by the last solution of the previous fronts to which it is compared after being non-dominated with respect to the rest of the solutions of the same front. Let the time complexity corresponding to (a) be T_{1A} and the one for (b) be T_{1B}.

In case of ENS-SS, a solution is assigned to F_k if (a) it is dominated by the last solution of all the previous $k-1$ fronts

and (b) it is non-dominated with each of the solutions in F_k. The time complexity of ENS-SS is obtained using Eq. 5.

$$\begin{aligned}T_{1_{\text{SS}}} &= T_{1A} + T_{1B} \\ &= \sum_{k=2}^{\sqrt{N}} \sqrt{N} M(k-1)\sqrt{N} + \sqrt{N}\left[1 + \sum_{i=2}^{\sqrt{N}} M(i-1)+1\right] \\ &= \frac{1}{2}MN(N-1)+N \\ &= \mathcal{O}(MN^2) \end{aligned} \tag{5}$$

In case of ENS-BS, a solution is assigned to F_k if (a) it is dominated by the last solution of the previous $\lceil \log k \rceil$ fronts and (b) it is non-dominated with respect to each of the solutions in F_k. The time complexity of ENS-BS is obtained using Eq. 6.

$$\begin{aligned}T_{1_{\text{BS}}} &= T_{1A} + T_{1B} \\ &= \sum_{k=2}^{\sqrt{N}} \sqrt{N} M \left(\lceil \log k \rceil\right) \sqrt{N} + \sqrt{N}\left[1 + \sum_{i=2}^{\sqrt{N}} M(i-1)+1\right] \\ &= \frac{1}{2}MN\sqrt{N}\log N - MN(\sqrt{N}-1) + \frac{1}{2}MN(\sqrt{N}-1) + N \\ &= \mathcal{O}(MN\sqrt{N}\log N) \end{aligned} \tag{6}$$

IV. Scope of Parallelism

Parallelism is possible in the first phase of ENS if a parallel sorting algorithm can be used, for example, parallel merge sort [32]. Using parallel merge sort, the worst case time complexity becomes $\mathcal{O}(MN)$ and the best case time complexity becomes $\mathcal{O}(N)$ when each of the merge operations at a particular level are performed simultaneously. This time complexity can be improved further if the merge operation can itself be implemented in a parallel manner as discussed in [32]. In this manner, the worst case time complexity becomes $\mathcal{O}(M \log^3 N)$ and the best case time complexity becomes $\mathcal{O}(\log^3 N)$.

To obtain parallelism in the second phase, a solution can be compared with each of the solutions in a particular front simultaneously. After comparing, it can be decided whether the solution can be inserted in that front or not. Thus, the second phase also has the parallelism property.

The parallel version of the sequential search based strategy to insert a solution in the identified set of fronts is summarized in Algorithm 4. Similarly, the parallel version of the binary search based strategy to insert a solution in the identified set of fronts is summarized in Algorithm 5. In both algorithms, a solution *sol* is compared with each of the solutions of a particular front simultaneously and their dominance relation is stored in an array of size n_k (n_k is the number of solutions in that particular front). This array stores whether solution *sol* is non-dominated with respect to the solutions of that particular front or not. After comparing *sol* with the solutions of a particular front simultaneously and storing their dominance relation in an array, we process this array in a parallel manner using Algorithm 6 to know whether the solution is non-dominated with respect to each of the solutions of that particular front or not.

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

Input: $\mathcal{F}$: Set of fronts $\{F_1, F_2, \ldots, F_K\}$, *sol*: Solution for insertion in $\mathcal{F}$
Output: Updated set of fronts

```
done ← FALSE                                   // Initially sol is not inserted
K ← |F|                                        // Obtain the number of fronts
for k ← 1 to K do                              // Check for each front in F sequentially starting from the first front
    isNondominated[1 ... n_k] ← FALSE          // Initialize an array of size n_k to store whether sol is non-dominated with the solutions of F_k or not
    /* PARALLEL SECTION STARTS */
    for u ← n_k down to 1 do                   // Check for each solution in F_k simultaneously
        if sol and F_k(u) are non-dominated then
            isNondominated[u] ← TRUE           // sol is non-dominated with solution F_k(u)
    /* PARALLEL SECTION ENDS */
    // Check whether sol is non-dominated with F_k or not in a parallel manner
    if INCLUSION(isNondominated[ ]) = TRUE then
        F_k ← F_k ∪ {sol}                      // Insert sol in a F_k
        isInserted ← TRUE                      // sol has been inserted
        BREAK                                  // Do not check other fronts
if done = FALSE then                           // sol has not been inserted yet
    F_{K+1} ← F_{K+1} ∪ {sol}                  // Insert sol in a new front
```

As the size of the array is n_k, so this array is processed at $\log n_k$ levels. At the lth level $n_k/2^l$ 'AND' operations are performed. We perform 'AND' operations because for a solution to be inserted in a front, it needs to be non-dominated with respect to each of the solutions of that front and 'AND' produces 'TRUE' if its two inputs are 'TRUE'. After the single 'AND' operation at the last level, if 'TRUE' is obtained, then it means that *sol* is non-dominated with respect to each of the solutions of that particular front. The time complexity to compare a solution with respect to each of the solutions of a particular front simultaneously is $\mathcal{O}(M)$. The time complexity to process the array of size n_k which stores whether *sol* is non-dominated with the solutions of a front or not is $\mathcal{O}(\log n_k)$ as all the 'AND' operations at a particular level can be performed simultaneously.

Example 1. *Let's assume we have eight solutions $\{sol_1, sol_2, \ldots, sol_8\}$ in a front. Solution sol is compared with all these eight solutions simultaneously and whether or not sol is non-dominated with respect to these solutions, it is stored in an array. This array is also processed in a parallel manner at $\log 8$ levels. Here, we are getting 'FALSE' after the 'AND' operation at the last level, which means that sol is dominated by at least one of the eight solutions. All the eight solutions along with the array which stores the dominance*

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

Input: $\mathcal{F}$: Set of fronts $\{F_1, F_2, \ldots, F_K\}$, *sol*: Solution for insertion in $\mathcal{F}$
Output: Updated set of fronts

$K \leftarrow |\mathcal{F}|$ // *Obtain the number of fronts*
$min \leftarrow 1$
$max \leftarrow K$
$mid \leftarrow \left\lfloor \frac{1+K}{2} \right\rfloor$
while TRUE **do** // *sol has not been inserted yet*
  isNondominated$[1 \ldots n_{\text{mid}}] \leftarrow$ FALSE // *Initialize an array of size n_{mid} to store whether sol is non-dominated with the solutions of F_k or not*
  /* PARALLEL SECTION STARTS */
  for $u \leftarrow n_{\text{mid}}$ **down to** 1 **do** // *Check for each solution in F_{mid} simultaneously*
    if *sol* and $F_{\text{mid}}(u)$ are non-dominated **then**
      isNondominated$[u] \leftarrow$ TRUE // *sol is non-dominated with solution $F_{mid}(u)$*
  /* PARALLEL SECTION ENDS */
  // *Check whether sol is non-dominated with F_k or not in a parallel manner*
  if INCLUSION(isNondominated[ ]) = TRUE **then**
    if $mid = min$ **then** // *The front at leaf is explored*
      $F_{\text{mid}} \leftarrow F_{\text{mid}} \cup \{sol\}$ // *Insert sol in F_{mid}*
      BREAK // *sol has been inserted*
    else
      $max \leftarrow mid$, $mid \leftarrow \left\lfloor \frac{min+max}{2} \right\rfloor$ // *Explore left sub-tree*
  else
    if $min = K$ **then** // *The front at right most leaf is explored*
      $F_{K+1} \leftarrow F_{K+1} \cup \{sol\}$ // *Insert sol in a new front*
      BREAK // *sol has been inserted*
    else if $max = mid + 1$ **then**
      $F_{\text{max}} \leftarrow F_{\text{max}} \cup \{sol\}$ // *Insert sol in F_{max}*
      BREAK // *sol has been inserted*
    else
      $min \leftarrow mid + 1$, $mid \leftarrow \left\lfloor \frac{min+max}{2} \right\rfloor$ // *Explore right sub-tree*

Algorithm 6 INCLUSION(isNondominated[ ])

Input: isNondominated[ ]: A boolean array
Output: TRUE : If all the values in the array is TRUE, FALSE : otherwise

$n_k \leftarrow |\text{isNondominated}|$ // *Obtain the size of isNondominated[ ]*
for $i \leftarrow 1$ to $\lceil \log_2 n_k \rceil$ **do**
  /* PARALLEL SECTION STARTS */
  for $j \leftarrow 1$ to $\left\lceil \frac{n_k}{2^i} \right\rceil$ **do**
    $a \leftarrow 2^i.j - (2^i - 1)$
    $b \leftarrow a + 2^{i-1}$
    if $b \leq n_k$ **then**
      isNondominated$[a] \leftarrow$ isNondominated$[a]$ && isNondominated$[b]$
  /* PARALLEL SECTION ENDS */
if isNondominated$[a]$ = TRUE **then**
  return TRUE // *sol is non-dominated with each of the solutions of the front*
else
  return FALSE // *sol is dominated by at least one of the solutions of the front*

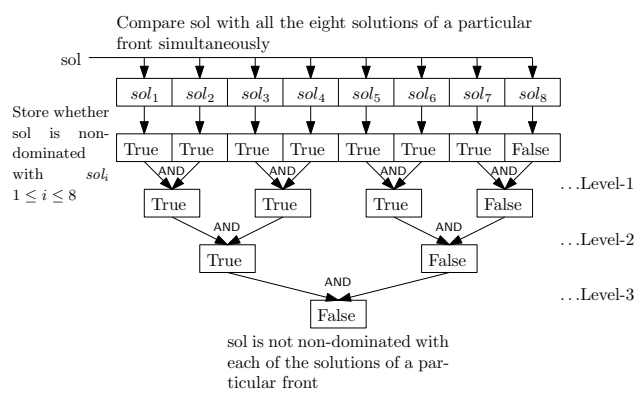

Fig. 1: Parallel Comparison of a solution with respect to each of the solutions of a particular front and to know whether or not a solution is non-dominated with respect to each of the solutions.

relationship is shown in Fig. 1.

Now, we discuss the time complexity of parallel versions of both variants of ENS in different scenarios.

A. All the solutions are in a single front

Here, ENS-SS and ENS-BS perform the same. The time complexity of the parallel version is given by Eq. 7.

$$\begin{aligned} T_\infty &= 1 + \sum_{i=2}^{N} \left[M + \lceil \log i \rceil + 1 \right] \\ &= 1 + \sum_{i=2}^{N} M + \sum_{i=2}^{N} \lceil \log i \rceil + \sum_{i=2}^{N} 1 \\ &= 1 + M(N-1) + N \log N - (N-1) + (N-1) \\ &= M(N-1) + N \log N + 1 \\ &= \mathcal{O}(MN + N \log N) \end{aligned} \tag{7}$$

If we obtain the dominance relationship between the solutions beforehand and store it in a matrix as done in [9], then the time complexity of the parallel version is given by Eq. 8.

$$\begin{aligned} T_\infty &= M + \left[1 + \sum_{i=2}^{N} \left[1 + \lceil \log i \rceil + 1 \right] \right] \\ &= M + 1 + \sum_{i=2}^{N} 1 + \sum_{i=2}^{N} \lceil \log i \rceil + \sum_{i=2}^{N} 1 \\ &= M + N + N \log N \\ &= \mathcal{O}(M + N \log N) \end{aligned} \tag{8}$$

B. All the solutions are in different fronts

In this case, there is no parallelism in the second phase as each front has a single solution. However, if we obtain the dominance relationship between the solutions beforehand as

in [9], then the time complexity of parallel ENS-SS is given by Eq. 9 and the time complexity of parallel ENS-BS is given by Eq. 10.

$$\begin{aligned} T_{\infty_{\text{SS}}} &= M+\left[1+\sum_{i=2}^{N}[(i-1)+1]\right] \\ &= M+1+\sum_{i=2}^{N}(i-1)+\sum_{i=2}^{N}1 \\ &= M+1+\frac{(N-1)(N+2)}{2} \\ &= \mathcal{O}(M+N^2) \end{aligned} \tag{9}$$

$$\begin{aligned} T_{\infty_{\text{BS}}} &= M+1+\sum_{i=2}^{N}\left[\lceil\log i\rceil+1\right] \\ &= M+1+\sum_{i=2}^{N}\lceil\log i\rceil+\sum_{i=2}^{N}1 \\ &= M+1+N\log N-(N-1)+(N-1) \\ &= \mathcal{O}(M+N\log N) \end{aligned} \tag{10}$$

C. Equal division of N solutions in $\sqrt{N}$ fronts

In the case of the serial version, we have discussed two scenarios when there is an equal division of N solutions in $\sqrt{N}$ fronts. The time complexity in these two scenarios were also different in case of the serial version. However, in the case of the parallel version, the time complexity remains the same for both scenarios. This is because in the case of the parallel version, a solution is compared with respect to each of the solutions of a particular front simultaneously and then it is decided whether it is non-dominated with respect to each of the solutions of that front or it is dominated by at least one of the solutions of that front.

Let the time complexity of the first part be $T_{\infty A}$ and the second part be $T_{\infty B}$. The time complexity of ENS-SS is obtained using Eq. 11.

$$\begin{aligned} T_{\infty_{\text{SS}}} &= T_{\infty A}+T_{\infty B} \\ &= \sum_{k=2}^{\sqrt{N}}\sqrt{N}\left(M+\lceil\log\sqrt{N}\rceil\right)(k-1)+ \\ &\quad \sqrt{N}\left[1+\sum_{i=2}^{\sqrt{N}}M+\lceil\log i\rceil+1\right] \\ &= \frac{1}{2}(M+\lceil\log\sqrt{N}\rceil)N(\sqrt{N}-1)+ \\ &\quad \frac{1}{2}N\log N+MN-M\sqrt{N}+\sqrt{N} \\ &= \mathcal{O}(MN\sqrt{N}+N\sqrt{N}\log N) \end{aligned} \tag{11}$$

The time complexity of ENS-BS is obtained using Eq. 12.

$$\begin{aligned} T_{\infty_{\text{BS}}} &= T_{\infty A}+T_{\infty B} \\ &= \sum_{k=2}^{\sqrt{N}}\sqrt{N}\left(M+\lceil\log\sqrt{N}\rceil\right)\lceil\log k\rceil+ \\ &\quad \sqrt{N}\left[1+\sum_{i=2}^{\sqrt{N}}M+\lceil\log i\rceil+1\right] \\ &= \left(M+\lceil\log\sqrt{N}\rceil\right)\left[N\log\sqrt{N}-(N+\sqrt{N})\right]+ \\ &\quad \frac{1}{2}N\log N+MN-M\sqrt{N}+\sqrt{N} \\ &= \mathcal{O}(MN\log N+N\log^2 N) \end{aligned} \tag{12}$$

When each solution in a front is dominated by each of the solutions in its preceding front, then in case of the parallel versions of ENS-SS and ENS-BS, the time complexity because of (b) has been improved as compared to the serial version. However, the time complexity has not been improved because of (a). This is because, in the case of the serial version, a solution is only compared with a single solution of each of the previous fronts and the time complexity of comparing a solution to one of the solutions is $\mathcal{O}(M)$. However, in the case of the parallel version, it is compared with each of the solutions of a particular front in $\mathcal{O}(M)$ time and then the array which stores the dominance relationship with $\sqrt{N}$ solutions of a particular front is processed in a parallel manner in $\mathcal{O}(\log\sqrt{N})$ time. So, the overall time complexity becomes $\mathcal{O}(M+\log\sqrt{N})$ in spite of the $\mathcal{O}(M)$ time complexity of the serial version.

If the dominance relationship between the solutions can be obtained beforehand as done in [9], then the time complexity of the parallel version of ENS-SS is given by Eq. 13 and the one for ENS-BS is obtained by Eq. 14.

$$\begin{aligned} T_{\infty_{\text{SS}}} &= T_{\infty A}+T_{\infty B} \\ &= M+\sum_{k=2}^{\sqrt{N}}\sqrt{N}\left(1+\lceil\log\sqrt{N}\rceil\right)(k-1)+ \\ &\quad \sqrt{N}\left[1+\sum_{i=2}^{\sqrt{N}}1+\lceil\log i\rceil+1\right] \\ &= M+\frac{1}{2}\left(1+\lceil\log\sqrt{N}\rceil\right)N(\sqrt{N}-1)+N\left(\log\sqrt{N}+1\right) \\ &= \mathcal{O}(M+N\sqrt{N}\log N) \end{aligned} \tag{13}$$

$$\begin{aligned} T_{\infty_{\text{BS}}} &= T_{\infty A}+T_{\infty B} \\ &= M+\sum_{k=2}^{\sqrt{N}}\sqrt{N}\left(1+\lceil\log\sqrt{N}\rceil\right)\lceil\log k\rceil+ \\ &\quad \sqrt{N}\left[1+\sum_{i=2}^{\sqrt{N}}1+\lceil\log i\rceil+1\right] \\ &= M+\left(1+\lceil\log\sqrt{N}\rceil\right)\left[N\log\sqrt{N}-(N+\sqrt{N})\right]+ \\ &\quad N\left(\log\sqrt{N}+1\right) \\ &= \mathcal{O}(M+N\log^2 N) \end{aligned} \tag{14}$$

V. Conclusions & Future Work

In this paper, a parallel version of Efficient Non-dominated Sort (ENS) is discussed. The time complexity of such parallel version is also analyzed. The worst case time complexity of the serial version of ENS is $\mathcal{O}(MN^2)$ when all the solutions are non-dominated with respect to each other. However, in the same scenario, the time complexity of the parallel version is $\mathcal{O}(M+N\log N)$. The best case time complexity of the serial version of ENS-SS is $\mathcal{O}(MN\sqrt{N})$ when there is an equal division of N solutions in $\sqrt{N}$ fronts such that all the solutions in a front are dominated by each of the solutions in its preceding front. However, in the same scenario, the time complexity of the parallel version is $\mathcal{O}(M+N\sqrt{N}\log N)$. The best case time complexity of the serial version of ENS-BS is $\mathcal{O}(MN\log N)$ when N solutions are divided into

N different fronts. However, in the same scenario, the time complexity of this parallel version is $\mathcal{O}(M + N \log N)$.

As part of our future work, we would like to explore the parallelization of other non-dominated sorting approaches such as the naive approach [1], DCNS [13] and BOS [12]. It would also be interesting to develop new approaches for non-dominated sorting. In this paper, we have theoretically proved the time complexity of a parallel version of ENS. It would be interesting to see the actual speedup of our proposed parallel version with respect to its corresponding serial version.

Acknowledgements

The authors acknowledge support from CONACyT project no. 221551.

References

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

[2] 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.

[3] 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.

[4] 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.

[5] S. Tang, Z. Cai, and J. Zheng, "A Fast Method of Constructing the Non-dominated Set: Arena's Principle," in *2008 Fourth International Conference on Natural Computation*. Jinan, China: IEEE Computer Society Press, 18-20 October 2008, pp. 391–395.

[6] 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.

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

[8] 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.

[9] C. Smutnicki, J. Rudy, and D. Zelazny, "Very Fast Non-Dominated Sorting," *Decision Making in Manufacturing and Services*, vol. 8, no. 1-2, pp. 13–23, 2014.

[10] 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.

[11] S. Gupta and G. Tan, "A Scalable Parallel Implementation of Evolutionary Algorithms for Multi-Objective Optimization on GPUs," in *2015 IEEE Congress on Evolutionary Computation (CEC'2015)*. Sendai, Japan: IEEE Press, 25-28 May 2015, pp. 1567–1574, iSBN 978-1-4799-7492-4.

[12] 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 the 2016 Genetic and Evolutionary Computation Conference Companion*. Denver, Colorado, USA: ACM Press, July 20-24 2016, pp. 1113–1120, ISBN: 978-1-4503-4323-7.

[13] 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, 2016, pp. 4297–4304, ISBN 978-1-5090-0624-3.

[14] X. Zhang, Y. Tian, R. Cheng, and Y. Jin, "A Decision Variable Clustering-Based Evolutionary Algorithm for Large-scale Many-objective Optimization," *IEEE Transactions on Evolutionary Computation*, 2018, (in press).

[15] G. Ortega, E. Filatovas, E. Garzón, and L. Casado, "Non-dominated sorting procedure for Pareto dominance ranking on multicore CPU and/or GPU," *Journal of Global Optimization*, vol. 69, no. 3, pp. 607–627, November 2017.

[16] P. Gustavsson and A. Syberfeldt, "A new algorithm using the non-dominated tree to improve non-dominated sorting," *Evolutionary Computation*, 2018, (in press).

[17] C. Bao, L. Xu, E. D. Goodman, and L. Cao, "A Novel Non-Dominated Sorting Algorithm for Evolutionary Multi-objective Optimization," *Journal of Computational Science*, vol. 23, pp. 31–43, November 2017.

[18] 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.

[19] L. Ding, S. Zheng, and L. Kang, "A Fast Algorithm on Finding the Non-dominated Set in Multi-objective Optimization," in *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'2003)*, vol. 4. Canberra, Australia: IEEE Press, December 2003, pp. 2565–2571.

[20] C. Shi, M. Chen, and Z. Shi, "A Fast Nondominated Sorting Algorithm," in *International Conference on Neural Networks and Brain 2005 (ICNN&B'05)*, vol. 3. Beijing, China: IEEE Press, 13-15 October 2005, pp. 1605–1610, ISBN 0-7803-9422-4 .

[21] X. Zhou, J. Shen, and J. Shen, "An Immune Recognition Based Algorithm for Finding Non-dominated Set in Multi-objective Optimization," in *PACIIA: 2008 Pacific-Asia Workshop on Computational Intelligence and Industrial Application*. Wuhan, China: IEEE Computer Society Press, December 19-20 2008, pp. 291–296, ISBN 978-1-4244-4204-1.

[22] J. Zheng, Z. Shi, C. X. Ling, and Y. Xie, "A New Method to Construct the Non-Dominated Set in Multi-Objective Genetic Algorithms," in *International Conference on Intelligent Information Processing (IIP'2004)*. Beijing, China: Springer, October 21-23 2004, pp. 457–470.

[23] J. E. Fieldsend, R. M. Everson, and S. Singh, "Using Unconstrained Elite Archives for Multiobjective Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 3, pp. 305–323, June 2003.

[24] K. Deb and S. Tiwari, "Omni-optimizer: A Procedure for Single and Multi-objective Optimization," in *Evolutionary Multi-Criterion Optimization. Third International Conference, EMO 2005*. Guanajuato, México: Springer. Lecture Notes in Computer Science Vol. 3410, March 2005, pp. 47–61.

[25] J. Zheng, Z. Shi, C. X. Ling, and Y. Xie, "Some Discussions about MOGAs: Individual Relations, Non-dominated Set, and Application on Automatic Negotiation," in *2004 Congress on Evolutionary Computation (CEC'2004)*, vol. 1. Portland, Oregon, USA: IEEE Service Center, June 2004, pp. 706–712.

[26] J. Du and Z. Cai, "A Sorting Based Algorithm for Finding a Non-Dominated Set in Multi-Objective Optimization," in *Third International Conference on Natural Computation (ICNC 2007)*, vol. 4. Haikou, China: IEEE Computer Society Press, 24-27 August 2007, pp. 436–440, ISBN 978-0-7695-2875-5.

[27] M. Drozdik, Y. Akimoto, H. Aguirre, and K. Tanaka, "Computational Cost Reduction of Nondominated Sorting Using the M-Front," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 5, pp. 659–678, October 2015.

[28] M. Buzdalov, I. Yakupov, and A. Stankevich, "Fast Implementation of the Steady-State NSGA-II Algorithm for Two Dimensions Based on Incremental Non-Dominated Sorting," in *2015 Genetic and Evolutionary Computation Conference (GECCO 2015)*. Madrid, Spain: ACM Press, July 11-15 2015, pp. 647–654.

[29] 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, September 2017.

[30] S. Mishra, S. Mondal, and S. Saha, "Improved solution to the non-domination level update problem," *Applied Soft Computing*, vol. 60, pp. 336–362, November 2017.

[31] I. Yakupov and M. Buzdalov, "Improved Incremental Non-Dominated Sorting for Steady-State Evolutionary Multiobjective Optimization," in *2017 Genetic and Evolutionary Computation Conference (GECCO'2017)*. Berlin, Germany: ACM Press, 15-19 July 2017, pp. 649–656.

[32] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, 2009, ISBN: 978-0262033-848.