Soft Computing (2021) 25:73–84
https://doi.org/10.1007/s00500-020-05450-1

FOUNDATIONS

A parallel naive approach for non-dominated sorting: a theoretical study considering PRAM CREW model

Sumit Mishra[1,2] · Carlos A. Coello Coello[1]

Published online: 26 November 2020

Abstract

Pareto-based multi-objective evolutionary algorithms use non-dominated sorting as an intermediate step. These algorithms are easy to parallelize as various steps of these algorithms are independent of each other. Researchers have focused on the parallelization of non-dominated sorting in order to reduce the execution time of these algorithms. In this paper, we focus on one of the initial approaches for non-dominated sorting also known as naive approach, proposed by Srinivas et al. and explore the scope of parallelism in this approach. Parallelism is explored in the considered approach in three different ways considering Parallel Random Access Machine, Concurrent Read Exclusive Write model. The time and space complexities of three different parallel versions are also analyzed. Analysis of parallel algorithms is usually carried out under the assumption that an unbounded number of processors are available. Thus, the same assumption has been considered in our analysis too and we have obtained the maximum number of processors required for three parallel versions.

Keywords Multi-objective optimization · Non-dominated sorting · Parallelism · PRAM CREW model

1 Introduction

Pareto-based multi- and many-objective evolutionary algorithms use non-dominated sorting as an intermediate step (Deb et al. 2002; Deb and Jain 2014). Non-dominated sorting is also used in other domains such as economics, databases, game theory and computational geometry. In non-dominated sorting, the solutions are sorted based on the dominance relationship between the solutions. Let $\mathbb{P} = \{sol_1, sol_2, \ldots, sol_N\}$ be a set of N solutions where each solution is associated with M objectives. The set of solutions is known as population. Actually, these solutions are in an M-dimensional objective space. A particular solution $sol_i (1 \leq i \leq N)$ in the population is represented as $sol_i = \{f_1(sol_i), f_2(sol_i), \ldots, f_M(sol_i)\}$ where $f_m(sol_i), 1 \leq m \leq M$ is the value of sol_i for the m^{th} objective. We consider here the optimization problems where the focus is to minimize all the objectives. For sorting the solutions, the dominance relation between them is required, which is defined as follows.

Communicated by A. Di Nola.

✉ Sumit Mishra
smishra@computacion.cs.cinvestav.mx

Carlos A. Coello Coello
ccoello@cs.cinvestav.mx

1 Departamento de Computación, CINVESTAV-IPN, 07360 Mexico City, Mexico

2 Department of Computer Science and Engineering , Indian Institute of Information Technology Guwahati, Guwahati, Assam 781015, India

Definition 1 (Dominance) A solution sol_i is said to dominate another solution sol_j which is represented as $sol_i \prec sol_j$ *iff* it satisfies the two following conditions:

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

The notation $sol_i \nprec sol_j$ is used to represent that solution sol_i does not dominate solution sol_j. We call two solutions sol_i and sol_j as non-dominated when neither solution dominates another, *i.e.*, neither $sol_i \nprec sol_j$ nor $sol_j \nprec sol_i$. In non-dominated sorting, the set of solutions are divided into non-dominated fronts as formally defined next.

Definition 2 (Non-dominated sorting) Given a set of N solutions $\{sol_1, \; sol_2, \ldots, sol_N\}$. In non-dominated sorting, these solutions are divided in $K(1 \leq K \leq N)$ non-dominated fronts $\{F_1, F_2, \ldots, F_K\}$ in decreasing order of their dominance such that

1. $\cup_{i=1}^{K} F_i = \mathbb{P}$
2. $\forall sol_i, sol_j \in F_k$: $sol_i \nprec sol_j$ and $sol_j \nprec sol_i$ $(1 \leq k \leq K)$
3. $\forall sol \in F_k, \exists sol' \in F_{k-1}$: $sol' \prec sol$ $(2 \leq k \leq K)$

In these sorted fronts, F_1 has the highest dominance, F_2 has the second highest dominance and so on. The last front F_K has the lowest dominance.

Parallel programming has attracted a lot of attention in recent years as a means to reduce the execution time of algorithms. Evolutionary algorithms are normally easy to parallelize due to their low data dependency, and this has motivated a considerable amount of research in this area (Luna and Alba 2015; Van Veldhuizen et al. 2003; Kim and Smith 2004; Maulik and Sarkar 2010; Shinde et al. 2011; Wong and Cui 2013). Consequently, the parallelization of non-dominated sorting algorithms has also attracted the interest of several researchers (see for example Smutnicki et al. 2014; Gupta and Tan 2015; Ortega et al. 2017; Moreno et al. 2018; Mishra and Coello 2018).

The main focus of the current research in this area has been on the parallelization of the naive approach for non-dominated sorting proposed by Srinivas and Deb (1994). However, there are several other non-dominated sorting approaches which have also the parallelism property such as the fast non-dominated sorting Deb et al. (2002), ENS Zhang et al. (2015), BOS Roy et al. (2016), DCNS Mishra et al. (2016), T-ENS Zhang et al. (2018) and ENS-NDT Gustavsson and Syberfeldt (2018), among others.

It is worth noticing that approaches such as ENS Zhang et al. (2015), BOS Roy et al. (2016), DCNS Mishra et al. (2016), ENS-NDT Gustavsson and Syberfeldt (2018) and T-ENS Zhang et al. (2018), require to sort $2N$ solutions (in case of NSGA-II Deb et al. (2002), NSGA-III Deb and Jain (2014), and others) unlike the naive approach (Srinivas and Deb 1994), fast non-dominated sort (Deb et al. 2002) and deductive sort (McClymont and Keedwell 2012) where the process of sorting can be stopped when we have enough fronts which contain N solutions. Let the N^{th} solution be inserted into the k^{th} front. As soon as any other solution is inserted into the $k+1^{th}$ front, the process of non-dominated sorting stops in case of the naive approach (Srinivas and Deb 1994), fast non-dominated sort (Deb et al. 2002) and deductive sort (McClymont and Keedwell 2012). However, ENS Zhang et al. (2015), BOS Roy et al. (2016), DCNS Mishra et al. (2016), ENS-NDT Gustavsson and Syberfeldt (2018) and T-ENS Zhang et al. (2018) continue the process until all $2N$ solutions are sorted. So, we have focused on the scope of parallelism in the naive approach. The worst case time complexity of the naive approach is $\mathcal{O}(MN^3)$, and the best case time complexity is $\mathcal{O}(MN^2)$ (Srinivas and Deb 1994). The best case of the naive approach occurs when all the solutions are non-dominated with respect to each other. Generally, when all the solutions are in single front (non-dominated with respect to each other), several approaches have their worst case (e.g., deductive sort McClymont and Keedwell 2012, ENS Zhang et al. (2015), DCNS Mishra et al. (2016), T-ENS Zhang et al. (2018) and ENS-NDT Gustavsson and Syberfeldt (2018)), having a $\mathcal{O}(MN^2)$ time complexity. The naive approach performs close to its best case when the number of fronts are less in number.

The organization of the rest of the paper is as follows. Some of the approaches for non-dominated sorting are described in Sect. 2. The naive approach is illustrated in Sect. 3 along with its time and space complexities. Parallelism in the naive approach is explored in Sect. 4, and three parallel versions are proposed. The time and space complexities of the parallel versions are mathematically derived. Finally, Sect. 5 concludes the paper with some possible paths for future research.

2 Previous related work

We discuss here some of the approaches for non-dominated sorting that have been proposed in the specialized literature. Deb et al. (2002) proposed fast non-dominated sort which requires $\mathcal{O}(MN^2)$ time and has a space complexity of $\mathcal{O}(N^2)$. Jensen (2003) proposed a recursive approach based on a divide-and-conquer strategy with time complexity $\mathcal{O}(N \log^{M-1} N)$ and space complexity $\mathcal{O}(MN)$. When two solutions have the same value for a particular objective, then this approach is not able to correctly sort the solutions. For two objectives, the time complexity of Jensen's approach is $\mathcal{O}(N \log N)$. Fang et al. (2008) proposed another approach based on a divide-and-conquer strategy. The worst case time complexity of this approach is $\mathcal{O}(MN^2)$. However, the best case time complexity is $\mathcal{O}(MN \log N)$ and its space complexity is $\mathcal{O}(MN)$. This approach has improved the best case time complexity. However, in case of duplicate solutions, this approach considers one solution as dominated by another. Tang et al. (2008) proposed an approach based on arena's principle with a worst case time complexity $\mathcal{O}(MN^2)$ and a best case time complexity $\mathcal{O}(MN\sqrt{N})$.

McClymont and Keedwell (2012) proposed two approaches: Climbing sort and Deductive sort. The worst case time complexity of both the approaches is $\mathcal{O}(MN^2)$. However, deductive sort performs half the comparisons of climbing sort in the worst case. The best case time complexity of deductive sort is $\mathcal{O}(MN\sqrt{N})$. This approach reduces the number of comparisons by inferring the dominance relationship between the solutions. The limitation of Jensen's approach is removed by adopting the proposal of Fortin et al. (2013). However, the limitation is removed at the cost of an increased worst case time complexity which is $\mathcal{O}(MN^2)$. The average case time complexity of Fortin's approach is same as Jensen's approach which is $\mathcal{O}(N \log^{M-1} N)$.

An efficient approach for non-dominated sorting known as ENS was proposed by Zhang et al. (2015). ENS works in two phases. In the first phase, the solutions are sorted based on a particular objective (generally the first objective). The solutions are ranked in the second phase. There are two variants of ENS based on how a solution is added to the existing set of fronts. The first one is ENS-SS which is based on sequential search, and the second one is ENS-BS which is based on binary search. ENS-SS and ENS-BS both have worst case time complexity $\mathcal{O}(MN^2)$. However, the best case time complexity of ENS-SS is $\mathcal{O}(MN\sqrt{N})$ and the best case time complexity of ENS-BS is $\mathcal{O}(MN \log N)$. Buzdalov and Shalyto (2014) proposed an approach with time complexity $\mathcal{O}(N \log^{M-1} N)$. A Hierarchical Non-dominated Sorting (HNDS) scheme was proposed by Bao et al. (2017). Like ENS, this is also a two-phased approach where the solutions are sorted based on a particular objective in the first phase and the solutions are ranked in the second phase. The worst case time complexity of HNDS is $\mathcal{O}(MN^2)$, and the best case time complexity is $\mathcal{O}(MN\sqrt{N})$. Corner sort was proposed by Wang and Yao (2014) with a worst case time complexity $\mathcal{O}(MN^2)$.

There are several other recent proposals (Roy et al. 2016; Zhang et al. 2018; Gustavsson and Syberfeldt 2018; Mishra et al. 2018b, a; Roy et al. 2018) where, for a solution to be inserted into a front, there is no need to compare it with all the other solutions. Best Order Sort (BOS) (Roy et al. 2016) is based on this same concept. In BOS, the solutions are initially sorted based on each of the objectives, considered separately. Then, the solutions are assigned to their respective front. The worst case time complexity of BOS is $\mathcal{O}(MN^2)$, whereas its best case time complexity is $\mathcal{O}(MN \log N)$. BOS has two advantages: (i) the number of dominance comparisons is reduced to a great extent and (ii) all the objectives values of two solutions are not considered while obtaining the dominance relationship between them. The second advantage of BOS is because of the comparison set concept which contains all the objective values of the solution. In spite of these two advantages, BOS is not able to handle duplicate solutions properly. BOS has been recently updated[1] to remove its limitation. However, BOS loses its second advantage in the process of removing its limitation. Mishra et al. (2018b) also worked on the same limitation of BOS and handled duplicate solutions efficiently without retaining the second advantage of BOS. Recently, the generalized version of BOS called "Generalized Best Order Sort" (GBOS) has been proposed which handles duplicate solutions efficiently and retains the comparison set concept of BOS (Mishra et al. 2018a). Thus, it makes BOS more effective in terms of the number of comparisons. Bounded Best Order Sort (BBOS) (Roy et al. 2018) is an improved version of BOS. BBOS works better for a large number of fronts. A heuristic is proposed to reduce the computational effort of solution comparisons. The worst case time complexity of BBOS is $\mathcal{O}(MN^2)$, and the best case time complexity is $\mathcal{O}(MN \log N)$. BBOS can achieve $\mathcal{O}(MN \log N + N^2)$ time complexity in case of a single front. The same time complexity has also been achieved by Roy et al. (2016); Mishra et al. (2018a, b).

A tree-based approach known as T-ENS was proposed by Zhang et al. (2018). In this approach, a non-dominated front is represented in the form of a tree to reduce the number of comparisons. The worst case time complexity of T-ENS is $\mathcal{O}(MN^2)$, and the best case time complexity is $\mathcal{O}(^{MN \log N}/_{\log M})$. By extending ENS-BS (Zhang et al. 2015), a tree-based approach known as ENS-NDT (Efficient Non-dominated Sort based on Non-dominated Tree) was proposed by Gustavsson and Syberfeldt (2018). This approach is able to handle duplicate solutions efficiently. The worst case time complexity of ENS-NDT is $\mathcal{O}(MN^2)$, and its best case time complexity is $\mathcal{O}(MN \log N)$ when $M > \log N$; otherwise, it is $\mathcal{O}(N \log^2 N)$.

There has been also some research on the parallelization of non-dominated sorting. A very fast non-dominated sort was proposed by Smutnicki et al. (2014). This approach focuses on exploring the parallelization of fast non-dominated sort (Deb et al. 2002). Parallelism is considered in two different manners. The time complexity of the first parallel version is $\mathcal{O}(M + N \log N)$, and the second parallel version is $\mathcal{O}(M + N)$. Gupta and Tan (2015) proposed a GPU-based parallel algorithm for non-dominated sorting which is also based on fast non-dominated sort (Deb et al. 2002). Ortega et al. (2017) also explores the parallelism in fast non-dominated sort (Deb et al. 2002). Three parallel versions were developed in this regard (*i.e.*, first based on GPUs, a second one based on multicores and a third one based on both GPUs and multicores). Recently, the parallelism in BOS (Roy et al. 2016) has been explored by Moreno et al. (2018). They have proposed two different parallel versions based on multicore processors and the GPU. Recently, the parallel version of ENS Zhang et al. (2015) has been discussed in Mishra and Coello (2018).

There are also other approaches (Drozdik et al. 2015; Mishra et al. 2016; Li et al. 2017; Mishra et al. 2017; Yakupov and Buzdalov 2017) where in spite of performing the complete non-dominated sorting, an offspring solution is inserted into its proper place in the existing sorted set of fronts. This kind of scenario is generally used in steady-state multi-objective evolutionary algorithms (Mishra et al. 2017).

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

3 Naive approach: serial version

In this section, we discuss the naive approach (Srinivas and Deb 1994) in its serial version. In the naive approach, each solution is compared with all the other solutions. After comparing the solutions with each other, the solutions which are not dominated by any other solution are assigned to the first front. Now, the solutions of the first front are not considered. The rest of the solutions are compared with each other. Now, 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 their respective front. The naive approach is described in Algorithm 1.

Algorithm 1 Naive Approach

```
Input: $\mathbb{P}$: Population of size $N$ where each solution is associated with
    $M$ objectives
Output: Ranked solutions
 1: rank ← 1
 2: repeat
 3:   isDominated[$1, 2, \ldots, |\mathbb{P}|$] ← FALSE // Initialize an array of size
      $|\mathbb{P}|$ to store whether a solution is dominated by any other solution
      in $\mathbb{P}$ or not
 4:   for each solution $sol \in \mathbb{P}$ do
 5:     for each solution $sol' \in \mathbb{P}$ do
 6:       if $sol$ is dominated by $sol'$ then
 7:         isDominated[$sol$] ← TRUE
 8:         BREAK // $sol$ is dominated by $sol'$ so there is no need to
    compare $sol$ with others
 9:       end if
10:     end for
11:   end for
      /* Check for each solution in $\mathbb{P}$ whether it is not dominated by
    any other solution */
12:   for each solution $sol \in \mathbb{P}$ : isDominated[$sol$] = FALSE do
13:     $sol_{\text{rank}}$ ← rank                          // Assign rank to sol
14:     $\mathbb{P} \leftarrow \mathbb{P} \setminus \{sol\}$   // Remove sol from $\mathbb{P}$ as it has been ranked
15:   end for
16:   rank ← rank + 1                          // Increase the value of rank
17: until $\mathbb{P}$ becomes empty
```

The worst case of the naive approach occurs when all the solutions are in the different fronts. In this case, in each iteration of the algorithm, a single solution is ranked. Thus, the time complexity in the worst case is given by Eq. (1).

$$T1_{\text{worst}} = \sum_{i=1}^{N} \left[\left\{ \sum_{j=1}^{N-i+1} M(N-i) \right\} + (N-i+1) \right]$$
$$= \frac{1}{3} MN \left(N^2 - 1\right) + \frac{1}{2} N(N+1) = \mathcal{O}\left(MN^3\right) \tag{1}$$

The best case of the naive approach occurs when all the solutions are in the same front. In this case, each solution is compared with other solutions only once and they are ranked. Thus, the best case time complexity is given by Eq. (2).

$$T1_{\text{best}} = \sum_{i=1}^{1} \left[\left\{ \sum_{j=1}^{N-i+1} M(N-i) \right\} + (N-i+1) \right]$$
$$= MN(N-1) + N = \mathcal{O}\left(MN^2\right) \tag{2}$$

To sort the solutions into different fronts, an array 'isDominated[ ]' is needed to store whether a solution is dominated by any other solution in the population or not. The maximum size of the population is N. Thus, the space complexity of the naive approach is $\mathcal{O}(N)$.

4 Scope of parallelism

In this section, we discuss the scope of parallelism in the naive approach. Parallelism is analyzed in the naive approach in three different manners. Before discussing the parallelism in detail, we first discuss the computing environment for which parallelism will be explored.

4.1 Computing environment

In our study, we are considering the PRAM CREW (Parallel random-access machine with Concurrent Read, Exclusive Write) model as considered in Smutnicki et al. (2014). The PRAM CREW model is earliest and best-known model of parallel computation (JáJá 1992; Kumar et al. 1994). In this model, simultaneous read at the same memory location is allowed. However, simultaneous write is not allowed. As simultaneous write operations are not allowed, so there will be no concurrent write operations in our parallel version. Analysis of parallel algorithms is usually carried out under the assumption that an unbounded number of processors is available (Niculescu 2007; Mikloško and Kotov 1984). So we have also considered this assumption. In our analysis, we have also obtained the maximum number of processors which can be required for the parallel version of the algorithm.

4.2 Parallelism in dominance comparisons

In general, the dominance relation between two solutions can be obtained in $\mathcal{O}(M)$ time as M objectives need to be compared. The dominance relation between each pair of solutions in the population can be obtained simultaneously. The dominance relationship between each pair of solutions can be stored in a matrix of size $N \times N$. We call this matrix *dominance matrix*. Thus, the *dominance matrix* can be obtained in $\mathcal{O}(M)$ time if the dominance relation between each pair of solutions can be obtained simultaneously. Smutnicki et al. (2014) has obtained the *dominance matrix* in the same manner. The time complexity of obtaining the *dominance matrix*

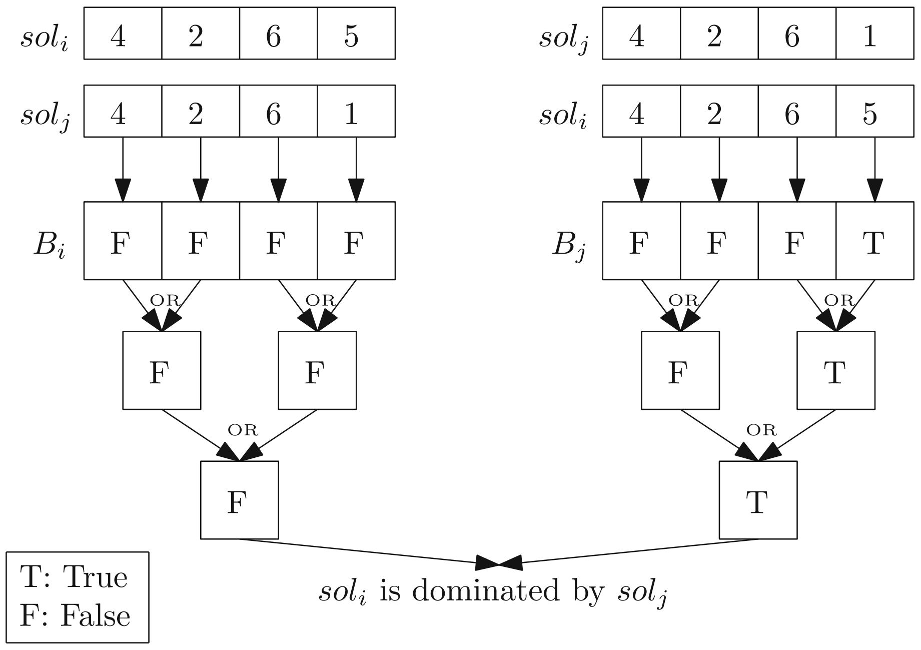


Fig. 1 The process to obtain the dominance relationship between the two solutions sol_i and sol_j in a parallel manner

can be further improved if the time complexity of obtaining the dominance relation can be improved. Now, we discuss the improved way to compute the dominance relation between two solutions.

Let us have two solutions sol_i and sol_j. For these two solutions, we create two Boolean arrays, each of size M. Let the first array be B_i and the second array be B_j. B_i is used to store whether sol_i is better than sol_j for each of the M objectives. Similarly, B_j is used to store whether sol_j is better than sol_i for each of the M objectives. If the objective value of sol_i is better than (less than as we focus to minimize all the objectives) the objective value of sol_j for the same objective, then the corresponding cell of B_i is set to 'TRUE'; otherwise, it is set to 'FALSE'. Similarly, B_j is also filled.

These two arrays B_i and B_j are processed simultaneously. As the size of both the arrays is M, so these arrays are processed at $\log M$ levels. At each level, 'OR' operations between two consecutive array cells are performed. The number of 'OR' operations at the l^{th} level is $M/2^l$. Thus, the number of 'OR' operations at the last level is one and we get either 'TRUE' or 'FALSE' after the 'OR' operation at the last level. Two values are obtained after processing both the arrays B_i and B_j. Let the value obtained from B_i be V_i and the value obtained from B_j be V_j. The dominance relationship between two solutions sol_i and sol_j can be obtained from V_i and V_j based on the following four conditions:

1. $V_i = V_j =$ FALSE: Solutions sol_i and sol_j are the same in terms of the objective values.
2. $V_i = V_j =$ TRUE: Solutions sol_i and sol_j are non-dominated.
3. $V_i =$ TRUE **and** $V_j =$ FALSE: Solution sol_i dominates sol_j.
4. $V_i =$ FALSE **and** $V_j =$ TRUE: Solution sol_i is dominated by sol_j.

The arrays B_i and B_j can be filled in $\mathcal{O}(1)$ time, in parallel. Different 'OR' operations at the same level can be performed simultaneously in both Boolean arrays, so the time complexity of processing these arrays in a parallel manner is $\mathcal{O}(\log M)$. Thus, the dominance relationship between a pair of solutions can be obtained in $\mathcal{O}(\log M)$ time. Hence, the *dominance matrix* can also be obtained in $\mathcal{O}(\log M)$ time in parallel, as the dominance relation between each pair of solutions can be obtained simultaneously. Once the *dominance matrix* is obtained, the time complexity of obtaining the dominance relationship between two solutions is $\mathcal{O}(1)$ as only a lookup in the *dominance matrix* is required.

The dominance relation between two solutions is obtained using two Boolean arrays of size M which require $\mathcal{O}(M)$ space. There are a total of N^2 pairs of solutions, so the overall space required to obtain the dominance relation between each pair of solutions is $\mathcal{O}(MN^2)$. Also, the space required to store the *dominance matrix* is $\mathcal{O}(N^2)$. Thus, the overall space complexity to obtain the *dominance matrix* in a parallel manner is $\mathcal{O}(MN^2)$.

Example 1 Let us consider two solutions $sol_i = \{4, 2, 6, 5\}$ and $sol_j = \{4, 2, 6, 1\}$ which are in four-dimensional objective space. For obtaining the dominance relationship between these two solutions, we fill the Boolean arrays B_i and B_j. After obtaining two Boolean arrays, these arrays are processed simultaneously using 'OR' operations. The final value obtained after processing B_i is 'FALSE' and after processing B_j is 'TRUE'. So, sol_i is dominated by sol_j. The complete process of filling both Boolean arrays and processing them is shown in Fig. 1.

Boolean array B_i can be filled using maximum of M processors in parallel as its size is M. In the same manner, array B_j can also be filled using maximum of M processors. Both these arrays can be filled simultaneously. Thus, the maximum number of processors required to fill both the Boolean arrays is $2M$. To obtain the dominance relationship between solutions sol_i and sol_j, Boolean arrays B_i and B_j are processed at $\log M$ levels. The maximum number of processors required to process any of the Boolean arrays is $M/2$. Both the Boolean arrays can be processed simultaneously, thus, the maximum number of processors required to process both the Boolean arrays is $M/2 + M/2 = M$. Hence, using maximum $2M$ processors, the dominance relationship between two solutions can be obtained. As the dominance relationship between each pair of the solutions can be obtained simultaneously and there are N^2 pairs of solutions, so the maximum number of processors required to obtain the *dominance matrix* in a parallel manner is $2MN^2$.

4.3 Parallel version-1

The parallel version-1 of the naive approach is described in Algorithm 2. Here, each solution *sol* can be simultaneously compared with other solutions (the outer *for* loop in lines $4 - 11$ can be implemented in a parallel manner). However, a particular solution *sol* is compared to other solution *sol'* in a serial manner (the inner *for* loop in lines $5 - 10$ is implemented in a serial manner). This scenario is shown in Fig. 2a. In this figure, all the N solutions in the top array are simultaneously compared with other solutions in the bottom array. However, each of the solutions in the top array is compared with all the solutions in the bottom array sequentially.

In this parallel version, after comparing each solution with all the other solutions, we check each solution sequentially to see whether it has been dominated by any other solution or not (lines $12 - 15$). The solution which is not dominated by any other solution is assigned a rank and removed from the population so that it do not take part is rank assignment process again. We repeat this process until all the solutions are ranked.

The time complexity in the worst case is given by Eq. (3).

$$T_{\infty_{\text{worst}}} = \sum_{i=1}^{N} [M(N-i) + (N-i+1)] = \frac{1}{2}MN(N-1) + \frac{1}{2}N(N+1) = \mathcal{O}(MN^2) \tag{3}$$

The time complexity in the best case is given by Eq. (4).

Algorithm 2 Naive Approach: Parallel Version-1

```
Input: ℙ: Population of size $N$ where each solution is associated with
    $M$ objectives
Output: Ranked solutions
 1: rank ← 1
 2: repeat
 3:   isDominated[1, 2, ..., |ℙ|] ← FALSE // Initialize an array of size
      |ℙ| to store whether a solution is dominated by any other solution
      in ℙ or not
      /* PARALLEL SECTION STARTS */
 4:   for each solution sol ∈ ℙ do          // Each solution sol is
      simultaneously compared with other solutions
 5:     for each solution sol' ∈ ℙ do  // sol is compared with all the
      solutions in a serial manner
 6:       if sol is dominated by sol' then
 7:         isDominated[sol] ← TRUE
 8:         BREAK
 9:       end if
10:     end for
11:   end for
      /* PARALLEL SECTION ENDS */
      /* Check for each solution in ℙ whether it is not dominated by
      any other solution */
12:   for each solution sol ∈ ℙ : isDominated[sol] = FALSE do
13:     $sol_{\text{rank}}$ ← rank                    // Assign rank to sol
14:     ℙ ← ℙ \ {sol}    // Remove sol from ℙ as it has been ranked
15:   end for
16:   rank ← rank + 1                  // Increase the value of rank
17: until ℙ becomes empty
```

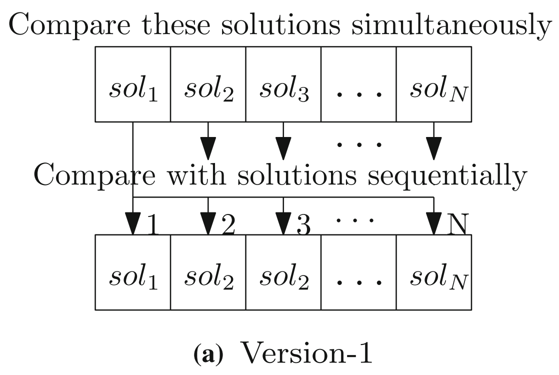
$$T_{\infty_{\text{best}}} = \sum_{i=1}^{1} [M(N-i) + (N-i+1)] = M(N-1) + N = \mathcal{O}(MN) \tag{4}$$

To sort the solutions into different fronts, an array 'isDominated[ ]' is needed to store whether a solution is dominated by any other solution in the population or not. The maximum size of the population is N. Thus, the space complexity of this parallel version is $\mathcal{O}(N)$. So, this parallel version does not add any extra overhead in terms of the space complexity as compared to the serial version.

In this parallel version, each solution *sol* can be simultaneously compared with other solutions. However, a particular solution *sol* is compared with other solutions in a serial manner. There are maximum N solutions; thus, the maximum number of processors required by this approach is N.

In this parallel version, if the dominance relationship between different solutions can be obtained initially as described in Sect. 4.2 and stored in *dominance matrix*, then the time complexity in the worst case is given by Eq. (5) and the time complexity in the best case is given by Eq. (6).

$$T_{\infty_{\text{worst}}} = \log M + \sum_{i=1}^{N} [(N-i) + (N-i+1)] = \log M + \frac{1}{2}N(N-1) + \frac{1}{2}N(N+1) = \mathcal{O}(\log M + N^2) \tag{5}$$


Compare these solutions simultaneously
sol_1 sol_2 sol_3 . . . sol_N
Compare with solutions sequentially
1 2 3 · · · N
sol_1 sol_2 sol_2 . . . sol_N
(a) Version-1

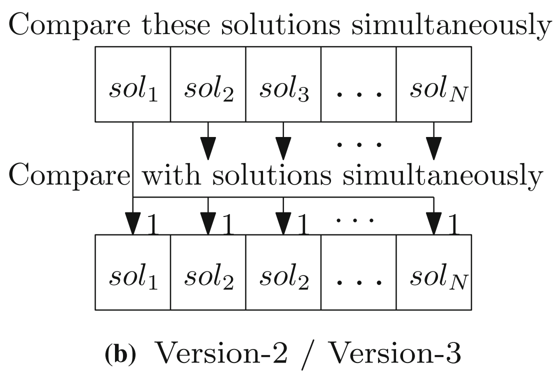

Compare these solutions simultaneously
sol_1 sol_2 sol_3 . . . sol_N
Compare with solutions simultaneously
1 1 1 · · · 1
sol_1 sol_2 sol_2 . . . sol_N
(b) Version-2 / Version-3

Fig. 2 Different types of parallelism in the naive approach. $1, 2, 3, \ldots, N$ in **a** denotes the sequential comparisons whereas $1, 1, 1, \ldots, 1$ in **b** denotes the parallel comparisons

$$\begin{aligned} T_{\infty\text{best}} &= \log M + \sum_{i=1}^{1} [(N-i) + (N-i+1)] \\ &= \log M + (N-1) + N = \mathcal{O}(\log M + N) \end{aligned} \quad (6)$$

If the dominance relationship between different solutions can be obtained initially and stored in a matrix as described in Sect. 4.2, then the space required for obtaining the *dominance matrix* is $\mathcal{O}(MN^2)$. Thus, the overall space complexity of this parallel version, when the dominance relationship between different solutions can be obtained beforehand, is $\mathcal{O}(MN^2)$.

The maximum number of processors required by this approach is N. When the dominance relationship between different solutions can be obtained initially and stored in a matrix as described in Sect. 4.2, then the maximum number of processors required by parallel version-1 is $2MN^2$.

4.4 Parallel version-2

The parallel version-2 of the naive approach is described in Algorithm 3. In this parallel version, each solution *sol* can be simultaneously compared with other solutions (The outer *for* loop in lines 4 − 15 is implemented in a parallel manner). Also, a particular solution *sol* is compared with other solutions *sol′* in a parallel manner (the inner *for* loop in lines 6 − 10 is also implemented in a parallel manner). This scenario is shown in Fig. 2b. In this figure, all the N solutions in the top array are simultaneously compared with other solutions in the bottom array. Also, each of the solutions in the top array is compared with all the solutions in the bottom array, simultaneously.

After comparing *sol* with all the other solutions simultaneously, we check whether *sol* is dominated by any of the solutions or not (line 11 − 14). For this purpose, the dominance relation of *sol* with respect to all the other solutions is stored in an array '*isDom*[ ]'. Let the size of '*isDom*[ ]' be N which stores whether a particular solution *sol* is dominated by other solutions $sol' \in \mathbb{P}$. A TRUE value in this array indicates that *sol* is dominated by *sol′* and FALSE indicates that it is not dominated.

Algorithm 3 Naive Approach: Parallel Version-2

```
Input: $\mathbb{P}$: Population of size $N$ where each solution is associated with
    $M$ objectives
Output: Ranked solutions
1: rank ← 1
2: repeat
3:   isDominated[$1, 2, \ldots, |\mathbb{P}|$] ← FALSE // Initialize an array of size
     $|\mathbb{P}|$ to store whether a solution is dominated by any other solution
     in $\mathbb{P}$ or not
     /* PARALLEL SECTION STARTS */
4:   for each solution $sol \in \mathbb{P}$ do          // Each solution sol is
     simultaneously compared with other solutions
5:     isDom[$1, 2, \ldots, |\mathbb{P}|$] ← FALSE // Initialize an array of size $|\mathbb{P}|$
       to store whether sol is dominated by $sol' \in \mathbb{P}$ or not
       /* PARALLEL SECTION STARTS */
6:     for each solution $sol' \in \mathbb{P}$ do   // sol is compared with all the
       solutions simultaneously
7:       if $sol$ is dominated by $sol'$ then
8:         isDom[$sol'$] ← TRUE
9:       end if
10:    end for
       /* PARALLEL SECTION ENDS */
       /* PARALLEL SECTION STARTS */
11:    Identify whether $sol$ is dominated by any other solution or not
       considering isDom[ ]   // It can be done in $\mathcal{O}(\log N)$ time in
       parallel manner if the size of isDom[ ] is $N$
12:    if $sol$ is dominated by $sol'$ then
13:      isDominated[$sol$] ← TRUE
14:    end if
       /* PARALLEL SECTION ENDS */
15:  end for
     /* PARALLEL SECTION ENDS */
     /* Check for each solution in $\mathbb{P}$ whether it is not dominated by
     any other solution */
16:  for each solution $sol \in \mathbb{P}$ : isDominated[$sol$] = FALSE do
17:    $sol_{\text{rank}}$ ← rank                        // Assign rank to sol
18:    $\mathbb{P} \leftarrow \mathbb{P} \setminus \{sol\}$   // Remove sol from $\mathbb{P}$ as it has been ranked
19:  end for
20:  rank ← rank + 1                       // Increase the value of rank
21: until $\mathbb{P}$ becomes empty
```

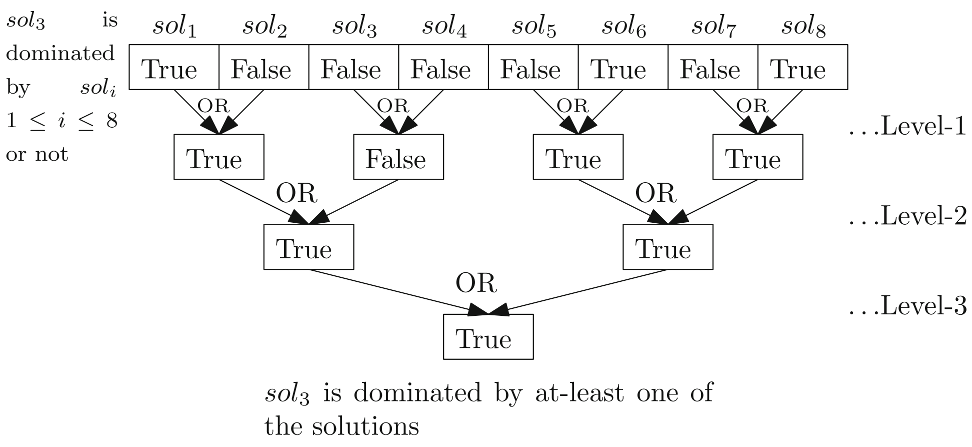

sol_3 is dominated by sol_i $1 \le i \le 8$ or not
sol_1 sol_2 sol_3 sol_4 sol_5 sol_6 sol_7 sol_8
True False False False False True False True
OR
True False True True
OR
True True
OR
True
...Level-1
...Level-2
...Level-3
sol_3 is dominated by at-least one of the solutions

Fig. 3 Simultaneously check whether a solution *sol* is dominated by at least one of the solutions or not in $\mathcal{O}(\log N)$ time considering N solutions

To know whether *sol* is dominated by any other solution or not, the array '*isDom*[ ]' is processed in a parallel manner at $\log N$ levels. At each level, an 'OR' operation is performed between two consecutive array cells. At the l^{th} level, $N/2^l$ 'OR' operations are performed. We are considering 'OR' operations because a solution cannot be ranked even if it is dominated by at least one of the solutions and 'OR' gives TRUE if any of its inputs is TRUE. The time complexity of processing the '*isDom*[ ]' array in a parallel manner is $\mathcal{O}(\log N)$ as the 'OR' operation is performed at $\log N$ levels and at each level all the 'OR' operations are performed simultaneously. At the last level, if TRUE is obtained, it means that *sol* is dominated by at least one of the solutions.

Now, we discuss the processing of the '*isDom*[ ]' array in a parallel manner using an example.

Example 2 Let $\mathbb{P} = \{sol_1, sol_2, \ldots, sol_8\}$ be a population of eight solutions. So, the size of the 'isDom[ ]' array will also be eight. Let solution sol_3 be checked to see whether it is dominated by other solutions in $\mathbb{P}$ or not. The array 'isDom[ ]' stores whether sol_3 is dominated by the solutions of $\mathbb{P}$ or not. Figure 3 shows the 'isDom[ ]' array. As the size of this array is eight, it is processed at $3(= \log 8)$ levels. At the last level, we are getting TRUE, so sol_3 is dominated by at least one of the solutions of population $\mathbb{P}$.

After comparing each solution with respect to all the others, we check for the solutions which are dominated by at least one of the solutions. This process is implemented in a parallel manner (lines $11 - 14$). The solutions which are not dominated by any other solution are assigned rank one. This complete process is repeated until all the solutions are ranked.
The time complexity in the worst case is given by Eq. (7).

$$T_{\infty_{\text{worst}}} = \sum\nolimits_{i=1}^{N} \left[M + \lceil\log(N-i+1)\rceil\right] + (N-i+1)$$
$$= MN + N\log N - (N-1) + \frac{1}{2}N(N+1) = \mathcal{O}(MN + N^2) \quad (7)$$

The time complexity in the best case is given by Eq. (8).

$$T_{\infty_{\text{best}}} = \sum\nolimits_{i=1}^{1} \left[M + \lceil\log(N-i+1)\rceil\right] + (N-i+1)$$
$$= (M + \log N) + N = \mathcal{O}(M+N) \quad (8)$$

In this parallel version, an array of population size 'isDominated[ ]' is needed to store which solution is dominated by any other solution in the population. Along with this, for each solution *sol*, an array 'isDom[ ]' of the size equal to the size of population is also created. The space required to store the 'isDominated[ ]' array is $\mathcal{O}(N)$. The space required to store 'isDom[ ]' array is also $\mathcal{O}(N)$, and this array 'isDom[ ]' is created for each of the solutions. Thus, the overall space required to store 'isDom[ ]' is $\mathcal{O}(N^2)$. Thus, the space complexity of this parallel version is $\mathcal{O}(N^2)$.

In this parallel version, each solution *sol* is simultaneously compared with other solutions. Also, a particular solution *sol* is compared with other solutions simultaneously. Once a solution *sol* has been compared with other solutions simultaneously (line $6 - 10$), we check whether *sol* is dominated by any other solutions to whom it has been simultaneously compared (line $11 - 14$). The number of processors require to compare a solution with all the solutions simultaneously is N. The maximum number of processors required to check whether a solution is dominated by any other solution or not in a parallel manner is $N/2$. Thus, the maximum number of processors required by this approach is N^2.

Here, if the dominance relationship between different solutions can be obtained initially as described in Sect. 4.2, then the time complexity in the worst case is given by Eq. (9) and the time complexity in the best case is given by Eq. (10).

$$T_{\infty_{\text{worst}}} = \log M + \sum\nolimits_{i=1}^{N} \left[1 + \lceil\log(N-i+1)\rceil\right] + (N-i+1)$$
$$= \log M + N + N\log N - (N-1) + \frac{1}{2}N(N+1)$$
$$= \mathcal{O}(\log M + N^2) \quad (9)$$

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

If the dominance relationship between different solutions can be obtained initially and stored in a matrix as described in Sect. 4.2, then the space required for obtaining the *dominance matrix* is $\mathcal{O}(MN^2)$. Thus, the overall space complexity of this parallel version, when the dominance relationship between different solutions can be obtained beforehand, is $\mathcal{O}(MN^2)$.

The maximum number of processors required by this approach is N^2 without considering *dominance matrix*. The maximum number of processors required to obtain the *dominance matrix* in a parallel manner is $2MN^2$. Thus, the maximum number of processors required by parallel version-2, when the dominance relation between the solution is obtained in constant time considering *dominance matrix*, is $2MN^2$.

4.5 Parallel version-3

The parallel version-3 of the naive approach is described in Algorithm 4. In this case, each solution *sol* can be simultaneously compared with other solutions (the outer *for* loop in lines $5-16$ is implemented in a parallel manner). Also, a particular solution *sol* is compared with other solutions sol' in a parallel manner (the inner *for* loop in lines $7-11$ is also implemented in a parallel manner). This scenario is shown in Fig. 2b.

In the previous versions of the naive approach, the solutions which are ranked are removed from the population. However, in this version, the ranked solutions are not removed from the population but instead, they are marked so that they are not ranked again. After comparing solutions with each other, we have to check whether a solution *sol* is dominated by another solution sol' or not. This can be done in a parallel manner by processing an array of size N in $\mathcal{O}(\log N)$ time as discussed in Parallel Version-2.

At last, we check the solutions which are not dominated by any other solutions and are also not ranked, in a parallel manner (lines $17-20$). After assigning rank to the solutions which are not dominated by any other non-ranked solutions, we check whether all the solutions have been ranked or not. This can also be checked in $\mathcal{O}(\log N)$ time if performed in parallel. This whole process is repeated until all the solutions are ranked.

Now, we discuss the process of knowing whether all the solutions are ranked or not in a parallel manner using an example.

Example 3 Let $\mathbb{P} = \{sol_1, sol_2, \ldots, sol_8\}$ be a population of eight solutions. Consider five solutions $\{sol_1, sol_4, sol_5, sol_6, sol_8\}$ which are ranked. In version-1 and version-2, after obtaining the set of solutions belonging to a particular front,

Algorithm 4 Naive Approach: Version-3

```
Input: $\mathbb{P}$: Population of size $N$ where each solution is associated with
   $M$ objectives
Output: Ranked solutions
 1: rank ← 1
    /* Initialize an array of size $|\mathbb{P}|$ to store whether a solution has been
    ranked or not */
 2: isRanked[1, 2, ..., $|\mathbb{P}|$] ← FALSE
 3: repeat
      /* Initialize an array of size $|\mathbb{P}|$ to store whether a solution is
    dominated by any other solution in $\mathbb{P}$ or not */
 4:   isDominated[1, 2, ..., $|\mathbb{P}|$] ← FALSE
      /* PARALLEL SECTION STARTS */
 5:   for each solution $sol \in \mathbb{P}$ : isRanked[$sol$] = FALSE do   // Each
    solution sol is simultaneously compared with other solutions
 6:     isDom[1, 2, ..., $|\mathbb{P}|$] ← FALSE // Initialize an array of size $|\mathbb{P}|$
        to store whether sol is dominated by $sol' \in \mathbb{P}$ or not
        /* PARALLEL SECTION STARTS */
 7:     for each solution $sol' \in \mathbb{P}$ : isRanked[$sol'$]=FALSE do   // sol is
    compared with all the solutions simultaneously which are not
    ranked
 8:       if $sol$ is dominated by $sol'$ then
 9:         isDom[$sol'$] ← TRUE
10:       end if
11:     end for
        /* PARALLEL SECTION ENDS */
        /* PARALLEL SECTION STARTS */
12:     Identify whether $sol$ is dominated by any other solution or not
        considering isDom[ ]   // It can be done in $\mathcal{O}(\log N)$ time in
        parallel manner if the size of isDom[ ] is $N$
13:     if $sol$ is dominated by $sol'$ then
14:       isDominated[$sol$] ← TRUE
15:     end if
        /* PARALLEL SECTION ENDS */
16:   end for
      /* PARALLEL SECTION ENDS */
      /* PARALLEL SECTION STARTS */
      /* Check for each solution in $\mathbb{P}$ whether it is not dominated by
    any other solution */
17:   for each solution $sol \in \mathbb{P}$ : isDominated[$sol$] = FALSE and
                            isRanked[$sol$] = FALSE do
18:     $sol_{\text{rank}}$ ← rank                        // Assign rank to sol
19:     isRanked[$sol$] ← TRUE   // Marked the solution sol as ranked
20:   end for
      /* PARALLEL SECTION ENDS */
21:   rank ← rank + 1                         // Increase the value of rank
22: until All the solutions are ranked              // It can be checked in
    $\mathcal{O}(\log N)$ time in parallel
```

these solutions are removed from the population. However, in version-3 the solutions are not deleted and to know which solutions are ranked or not, an array of size equal to the population size is considered. The array corresponding to eight solutions is shown in Fig. 4.

When a solution is ranked, the corresponding cell in the array is marked as TRUE which signifies that the solution is ranked. After obtaining the solutions belonging to a particular front, we have to check whether all the solutions have been ranked or not. For this purpose, the array is processed in a parallel manner. As the length of the array is N (equal to the population size), so the array is processed at $\log N$ levels

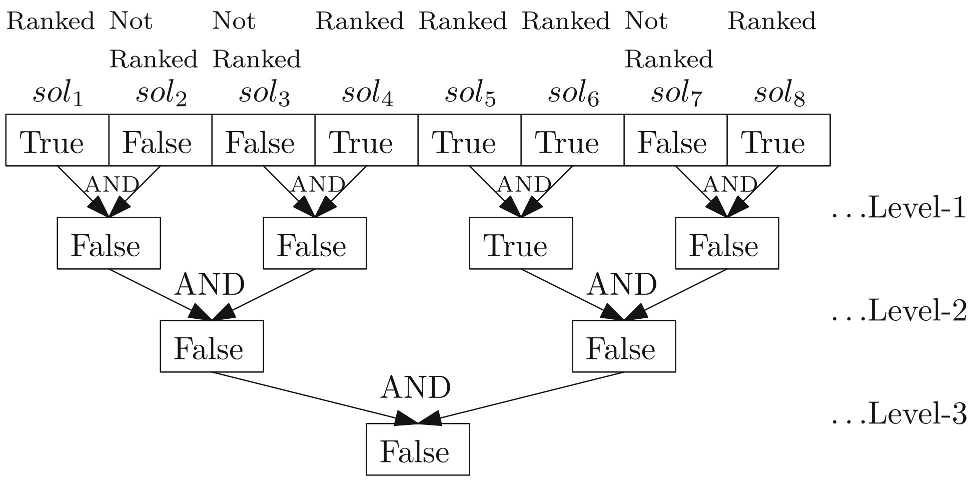


Fig. 4 Simultaneously check whether N solutions are ranked or not in $\mathcal{O}(\log N)$ time considering N solutions

where at each level, an 'AND' operation is performed in consecutive array cells. At the last level, if TRUE is obtained, then all the solutions are ranked; otherwise, all the solutions are not ranked. In Fig. 4, as there are eight solutions, so the parallel operation is performed at three different levels. At the last level, FALSE is obtained after an 'AND' operation which means that all the solutions are not ranked.

The time complexity in the worst case is given by Eq. (11).

$$T_{\infty_{\text{worst}}} = \sum_{i=1}^{N} \left[M + \lceil \log N + 1 \rceil\right] + \lceil \log N \rceil = MN + N + 2N\lceil \log N \rceil = \mathcal{O}(MN + N \log N) \tag{11}$$

The time complexity in the best case is given by Eq. (12).

$$T_{\infty_{\text{best}}} = \sum_{i=1}^{1} \left[M + \lceil \log N \rceil + 1\right] + \lceil \log N \rceil = M + 1 + 2\lceil \log N \rceil = \mathcal{O}(M + \log N) \tag{12}$$

In this parallel version, an array 'isRanked[ ]' of size N is created to store which solution has been ranked. The space required to store this array is $\mathcal{O}(N)$. The analysis of the space complexity remains the same as the parallel version-2. Thus, the space complexity of this parallel version is $\mathcal{O}(N^2)$.

In this parallel version, each solution *sol* is simultaneously compared with other solutions. Also, a particular solution *sol* is compared with other solutions simultaneously. Once a solution *sol* has been compared with other solutions simultaneously (line $7-11$), we check whether *sol* is dominated by any other solutions to whom it has been simultaneously compared (line $12-15$). The number of processors require to compare a solution with all the solutions simultaneously is N. The maximum number of processors required to check whether a solution is dominated by any other solution or not in a parallel manner is $N/2$. After this, for each solution *sol* which is not dominated by any other solution and has been already ranked is assigned a rank. This operation can be carried out in parallel using maximum N processors. Thus, the maximum number of processors required by this approach is N^2.

Here, if the dominance relationship between different solutions can be obtained initially as described in Sect. 4.2, then the time complexity in the worst case is given by Eq. (13) and the time complexity in the best case is given by Eq. (14).

$$T_{\infty_{\text{worst}}} = \log M + \sum_{i=1}^{N} \left[1 + \lceil \log N \rceil + 1\right] + \lceil \log N \rceil = \log M + 2N + 2N\lceil \log N \rceil = \mathcal{O}(\log M + N \log N) \tag{13}$$

$$T_{\infty_{\text{best}}} = \log M + \sum_{i=1}^{1} \left[1 + \lceil \log N \rceil + 1\right] + \lceil \log N \rceil = \log M + 2 + 2\lceil \log N \rceil = \mathcal{O}(\log M + \log N) \tag{14}$$

If the dominance relationship between different solutions can be obtained initially and stored in a matrix as described in Sect. 4.2, then the space required for obtaining the *dominance matrix* is $\mathcal{O}(MN^2)$. Thus, the overall space complexity of this parallel version, when the dominance relationship between different solutions can be obtained beforehand, is $\mathcal{O}(MN^2)$.

The maximum number of processors required by this approach is N^2 without considering *dominance matrix*. The maximum number of processors required to obtain the *dominance matrix* in a parallel manner is $2MN^2$. Thus, the maximum number of processors required by parallel version-3, when the dominance relation between the solution is obtained in constant time considering *dominance matrix*, is $2MN^2$.

The worst case time complexity of the naive approach is $\mathcal{O}(MN^3)$, and the best case time complexity is $\mathcal{O}(MN^2)$. The time complexity of the parallel version of the non-dominated sorting was proved to be $\mathcal{O}(M+N)$ by Smutnicki et al. (2014). The worst and the best case time complexities of the parallel version-1 and parallel version-2 of the naive approach are $\mathcal{O}(\log M + N^2)$ and $\mathcal{O}(\log M + N)$, respectively. The best case time complexity is better than the time complexity as reported in Smutnicki et al. (2014). However, the worst case time complexity is not. The worst and best case time complexities of the parallel version-3 of the naive approach are $\mathcal{O}(\log M + N \log N)$ and $\mathcal{O}(\log M + \log N)$, respectively. The best case time complexity of the parallel version-3 is $\mathcal{O}(\log M + \log N)$ which is better than the time complexity as reported in Smutnicki et al. (2014). However, the worst case time complexity of the parallel version-3 is $\mathcal{O}(\log M + N \log N)$ which is not good as compared to the time complexity reported in Smutnicki et al. (2014). However, as discussed in Sect. 1, as the number of fronts decreases, the naive approach performs near to its best case. So as the evolutionary algorithm proceeds, the parallel naive approach can be advantageous because the number of non-dominated fronts start reducing.

5 Conclusions & future work

In this paper, we have explored the scope of parallelism in the naive approach. We have identified parallelism in the naive approach in three different ways. The worst case time complexity of the parallel version is $\mathcal{O}(\log M + N \log N)$, and the best case time complexity is $\mathcal{O}(\log M + \log N)$. The best case occurs when all the solutions are in a single front. As the evolutionary algorithm proceeds, the number of fronts decreases and the approach performs either in the best case or near to its best case. As part of our future work, we would like to find the scope of parallelism in other approaches as well. It would also be interesting to see the actual speedup when different parallel methods of the naive approach are implemented.

Acknowledgements The Carlos A. Coello Coello gratefully acknowledges financial support from CONACyT Grant Nos. 2016 − 01 − 1920 and 221551 (Investigación en Fronteras de la Ciencia 2016) and from a project from the 2018 SEP-Cinvestav Fund (Application No. 4).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Human and animal rights This article does not contain any studies with human participants or animals performed by any of the authors.

References

Bao C, Xu L, Goodman ED, Cao L (2017) A novel non-dominated sorting algorithm for evolutionary multi-objective optimization. J Comput Sci 23:31–43

Buzdalov M, Shalyto A (2014) A provably asymptotically fast version of the generalized Jensen Algorithm for non-dominated sorting. In: 13th International Conference Parallel Problem Solving from Nature—PPSN XIII, Springer. Lecture Notes in Computer Science vol 8672, Ljubljana, Slovenia, pp 528–537

Deb K, Jain H (2014) An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, Part I: solving problems with box constraints. IEEE Trans Evol Comput 18(4):577–601

Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGAII. IEEE Trans Evol Comput 6(2):182–197

Drozdik M, Akimoto Y, Aguirre H, Tanaka K (2015) Computational cost reduction of nondominated sorting using the M-Front. IEEE Trans Evol Comput 19(5):659–678

Fang H, Wang Q, Tu YC, Horstemeyer MF (2008) An efficient non-dominated sorting method for evolutionary algorithms. Evol Comput 16(3):355–384

Fortin FA, Greiner S, Parizeau M (2013) Generalizing the improved run-time complexity algorithm for non-dominated sorting. In: 2013 genetic and evolutionary computation conference (GECCO'2013), ACM Press, New York, USA, pp 615–622, ISBN: 978-1-4503-1963-8

Gupta S, Tan G (2015) A scalable parallel implementation of evolutionary algorithms for multi-objective optimization on GPUs. In: 2015 IEEE congress on evolutionary computation (CEC'2015), IEEE Press, Sendai, Japan, pp 1567–1574, ISBN: 978-1-4799-7492-4

Gustavsson P, Syberfeldt A (2018) A new algorithm using the non-dominated tree to improve non-dominated sorting. Evol Comput 26(1):89–116

JáJá J (1992) An introduction to parallel algorithms, vol 17. Addison-Wesley, Boston

Jensen MT (2003) Reducing the run-time complexity of multiobjective EAs: the NSGA-II and other algorithms. IEEE Trans Evol Comput 7(5):503–515

Kim K, Smith RL (2004) Parallel multiobjective evolutionary algorithms for waste solvent recycling. Ind Eng Chem Res 43(11):2669–2679

Kumar V, Grama A, Gupta A, Karypis G (1994) Introduction to parallel computing: design and analysis of algorithms, vol 400. Benjamin/Cummings, Redwood City

Li K, Deb K, Zhang Q, Zhang Q (2017) Efficient nondomination level update method for steady-state evolutionary multiobjective optimization. IEEE Trans Cybern 47(9):2838–2849

Luna F, Alba E (2015) Parallel multiobjective evolutionary algorithms. In: Springer handbook of computational intelligence, Springer, pp 1017–1031

Maulik U, Sarkar A (2010) Evolutionary rough parallel multi-objective optimization algorithm. Fundamenta Informaticae 99(1):13–27

McClymont K, Keedwell E (2012) Deductive sort and climbing sort: new methods for non-dominated sorting. Evol Comput 20(1):1–26

Mikloško J, Kotov VE (1984) Algorithms, software and hardware of parallel computers. Springer, Berlin

Mishra S, Coello CA (2018) P-ENS: Parallelism in efficient non-dominated sorting. In: 2018 IEEE congress on evolutionary computation (CEC'2018), IEEE Press, Rio de Janeiro, Brazil, pp 508–515, ISBN: 978-1-5090-6017-7

Mishra S, Mondal S, Saha S (2016) Fast implementation of steady-state NSGA-II. In: 2016 IEEE congress on evolutionary computation (CEC'2016), IEEE Press, Vancouver, Canada, pp 3777–3784, ISBN:978-1-5090-0623-6

Mishra S, Saha S, Mondal S (2016) Divide and conquer based non-dominated sorting for parallel environment. In: 2016 IEEE congress on evolutionary computation (CEC'2016), IEEE Press, Vancouver, Canada, pp 4297–4304, ISBN: 978-1-5090-0623-6

Mishra S, Mondal S, Saha S (2017) Improved Solution to the Non-Domination Level Update Problem. Appl Soft Comput 60:336–362

Mishra S, Mondal S, Saha S, Coello CAC (2018) GBOS: generalized best order sort algorithm for non-dominated sorting. Swarm Evolut Comput. https://doi.org/10.1016/j.swevo.2018.06.003

Mishra S, Saha S, Mondal S (2018) MBOS: Modified best order sort algorithm for performing non-dominated sorting. In: 2018 IEEE congress on evolutionary computation (CEC'2018), IEEE Press, Rio de Janeiro, Brazil, pp 725–732, ISBN: 978-1-5090-6017-7

Moreno J, Ortega G, Filatovas E, Martínez J, Garzón E (2018) Improving the performance and energy of non-dominated sorting for evolutionary multiobjective optimization on GPU/CPU platforms. J Glob Optim 7:631

Niculescu V (2007) Data-distributions in powerlist theory. In: International colloquium on theoretical aspects of computing, Springer, pp 396–409

Ortega G, Filatovas E, Garzon EM, Casado LG (2017) Non-dominated sorting procedure for pareto dominance ranking on multicore CPU and/or GPU. J Global Optim 69(3):607–627

Roy PC, Islam MM, Deb K (2016) Best order sort: a new algorithm to non-dominated sorting for evolutionary multi-objective optimization. In: Proceedings of the 2016 on genetic and evolutionary computation conference companion, ACM Press, Denver, Colorado, USA, pp 1113–1120, ISBN: 978-1-4503-4323-7

Roy PC, Deb K, Islam MM (2018) An efficient nondominated sorting algorithm for large number of fronts. IEEE Trans Cybern 49(3):859–869

Shinde G, Jagtap SB, Pani SK (2011) Parallelizing multi-objective evolutionary genetic algorithms. In: Proceedings of the world congress on engineering, vol 2

Smutnicki C, Rudy J, Zelazny D (2014) Very fast non-dominated sorting. Decis Making Manuf Serv 8(1–2):13–23

Srinivas N, Deb K (1994) Multiobjective optimization using nondominated sorting in genetic algorithms. Evol Comput 2(3):221–248

Tang S, Cai Z, Zheng J (2008) A fast method of constructing the non-dominated set: arena's principle. In: 2008 fourth international conference on natural computation, IEEE Computer Society Press, Jinan, China, pp 391–395, ISBN: 978-0-7695-3304-9

Van Veldhuizen DA, Zydallis JB, Lamont GB (2003) Considerations in engineering parallel multiobjective evolutionary algorithms. IEEE Trans Evol Comput 7(2):144–173

Wang H, Yao X (2014) Corner sort for pareto-based many-objective optimization. IEEE Trans Cybern 44(1):92–102

Wong ML, Cui G (2013) Data Mining Using Parallel Multi-objective Evolutionary Algorithms on Graphics Processing Units. In: Massively Parallel Evolutionary Computation on GPGPUs, Springer, pp 287–307, ISBN: 978-3-642-37958-1

Yakupov I, Buzdalov M (2017) Improved incremental non-dominated sorting for steady-state evolutionary multiobjective optimization. In: 2017 genetic and evolutionary computation conference (GECCO'2017), ACM Press, Berlin, Germany, pp 649–656, ISBN: 978-1-4503-4920-8

Zhang X, Tian Y, Cheng R, Yaochu J (2015) An efficient approach to nondominated sorting for evolutionary multiobjective optimization. IEEE Trans Evol Comput 19(2):201–213

Zhang X, Tian Y, Cheng R, Jin Y (2018) A decision variable clustering-based evolutionary algorithm for large-scale many-objective optimization. IEEE Trans Evol Comput 22(1):97–112