Filter Sort Is (N3) in the Worst Case

Mishra, Sumit; Buzdalov, Maxim

Published in:
Parallel Problem Solving from Nature – PPSN XVI - 16th International Conference, PPSN 2020, Proceedings

DOI:
10.1007/978-3-030-58115-2_47

Publication date:
2020

Citation for published version (APA):
Mishra, S., & Buzdalov, M. (2020). Filter Sort Is (N^3) in the Worst Case. In T. Bäck, M. Preuss, A. Deutz, M. Emmerich, H. Wang, C. Doerr, & H. Trautmann (Eds.), *Parallel Problem Solving from Nature – PPSN XVI - 16th International Conference, PPSN 2020, Proceedings* (pp. 675-685). (Lecture Notes in Computer Science; Vol. 12270). Springer Nature. https://doi.org/10.1007/978-3-030-58115-2_47

Filter Sort is $\Omega(N^3)$ in the Worst Case

Sumit Mishra[1] and Maxim Buzdalov[2]

[1] IIIT Guwahati, Guwahati, India, `sumit@iiitg.ac.in`
[2] ITMO University, Saint Petersburg, Russia, `mbuzdalov@gmail.com`

Abstract. Non-dominated sorting is a crucial operation used in many popular evolutionary multiobjective algorithms. The problem of non-dominated sorting, although solvable in polynomial time, is surprisingly difficult, and no algorithm is yet known which solves any instance on N points and M objectives in time asymptotically smaller than MN^2.
For this reason, many algorithm designers concentrate on reducing the leading constant and on (implicitly) tailoring their algorithms to inputs typical to evolutionary computation. While doing that, they sometimes forget to ensure that the worst-case running time of their algorithm is still $O(MN^2)$. This is undesirable, especially if the inputs which make the algorithm work too slow can occur spontaneously. However, even if a counterexample is hard to find, the fact that it exists is still a weak point, as this can be exploited and lead to denial of service and other kinds of misbehaving.
In this paper we prove that a recent algorithm for non-dominated sorting, called Filter Sort, has the worst-case complexity of $\Omega(N^3)$. In particular, we present a scenario which requires Filter Sort to perform $\Theta(N^3)$ dominance comparisons, where each comparison, however, needs only $O(1)$ elementary operations. Our scenario contains $\Theta(N)$ non-domination layers, which is a necessary, but by no means a sufficient condition for being difficult for Filter Sort.

Keywords: Non-dominated sorting, Filter Sort, time complexity.

1 Introduction

Optimizers that rank solutions based on the Pareto dominance relation arguably prevail in evolutionary multiobjective optimization for already more than two decades, and only relatively recently they started to partially lose many-objective ground to decomposition-based methods. In turn, among the ranking methods that employ the Pareto dominance relation, non-dominated sorting is probably one of the most frequently used. With a relatively small computation cost and a possibility of writing a relatively easy implementation, it is now used not only in the algorithm NSGA-II [9] that popularized it, but in a wide range of algorithms belonging to different paradigms, such as Strength Pareto Evolutionary Algorithm (SPEA), Pareto Envelope-Based Selection Algorithm (PESA) [6], Pareto Archived Evolution Strategy (PAES) [14], M-Pareto Archived Evolution Strategy (M-PAES) [13], Micro-GA [5], KnEA [38], NSGA-III [8] and many others.

Assume that there are M objectives and, without loss of generality, that we are required to minimize all objectives. A solution p is said to dominate, in Pareto sense, another solution q, which is written as $p \prec q$, if the following conditions are satisfied:

1. $\forall i \in [1..M]$ it holds that $p_i \le q_i$;
2. $\exists j \in [1..M]$ such that $p_j < q_j$;

where the notation is as follows: $[a..b]$ is the set of integers $\{a, a+1, \ldots, b-1, b\}$.

Non-dominated sorting can then be defined as follows. Let $\mathbb{P} = \{\mathbb{P}_1, \ldots, \mathbb{P}_N\}$ be a population of N evaluated solutions. The problem is to divide $\mathbb{P}$ into several fronts $\mathbb{F} = \{\mathbb{F}_1, \mathbb{F}_2, \ldots\}$, such that the following conditions are satisfied:

1. The fronts constitute a partition, that is:
   - $\bigcup_i \mathbb{F}_i = \mathbb{P}$;
   - if $i \ne j$, then $\mathbb{F}_i \cap \mathbb{F}_j = \emptyset$;
2. For any two solutions $s, t \in \mathbb{F}_i$, neither of them dominates another one;
3. For $i > 1$, for any $s \in \mathbb{F}_i$ there exists some $t \in \mathbb{F}_{i-1}$ such that $t \prec s$.

We shall explicitly state here that, for the soundness of the definition above, a solution does not dominate itself (as well as it does not dominate any other solution with identical objective values), as otherwise the ordering of the solutions would affect the results of the procedure in an implementation-dependent way. However, it is often desirable, that, whenever there are three solutions p_1, p_2, q such that $p_1 = p_2$ and $p_1 \prec q$, the solution q gets a worse rank than it would have without p_2. The reader is welcome to an extended discussion about possible sound extensions of this version of non-dominated sorting, as well as other similar problems, in a recent paper [2].

Outside evolutionary computation, non-dominated sorting is often known under other names (such as *layers of maxima* or *longest chains*) and has applications in various domains like data clustering [11], graph theory [17], computer vision, economics and game theory [16], database [1] and others [7,28,29].

Since in this paper we are interested in algorithms for non-dominated sorting, and do not investigate its applications, we do not differentiate between solutions and their objective vectors, treat them as points in an M-dimensional space and use "points" and "solutions" interchangeably.

The apparent simplicity of the non-dominated sorting problem makes it surprising that, despite a huge effort, no algorithms are still known that solve this problem in time $o(MN^2)$ for any input of N points and M objectives. Below, we give a quick summary of the basic ideas of these algorithms.

There are plenty of algorithms that run in $\Theta(MN^2)$ time in the worst case, beginning with the algorithm called "fast non-dominated sorting" that accompanied NSGA-II [9], as well as more advanced approaches [10,23–27,30,31,33,36,37] and some others. All these algorithms focus on the running time on inputs with rather small values of N, and on those that are distributed similar to typical populations in evolutionary multiobjective optimization. The basic desire driving most of these designs is to somehow "reduce the number of unnecessary comparisons" with certain heuristics that work reasonably well under uniform or other similar distributions. Most of these algorithms cannot cope with $N = 10^5$ points, however, there are notable exceptions, such as the kd-tree-based algorithm called ENS-NDT [10] and, to some extent, the flavours of Best Order Sort [23, 25, 30, 31].

The algorithms belonging to a different group apply the divide-and-conquer paradigm in a particular manner that allows an asymptotically better upper bound of $O(N(\log N)^{M-1})$ to be proven. This expression holds when M is constant with regards to N, and all these algorithms also satisfy the $O(MN^2)$ upper bound, which they quickly reach with large enough M. The divide-and-conquer flavour in question is suggested long time ago in [15], and it was applied for the first time to non-dominated sorting in [12]. Subsequent development involved modifications to reliably work on every input [4] and various practical runtime improvements typically involving hybridization with other algorithms [18, 19], and word-RAM data structures [3].

However, some algorithms have even worse running time guarantees. The original NSGA algorithm [32] featured a particularly naive algorithm that works in $O(MN^3)$ time, where one of the N factors is the number of fronts in the output. Unfortunately, some of the algorithms are also that slow. On some occasions it is trivial to show, as it was the case with the DDA sorting [22, 39], however, sometimes the original paper features a wrong optimistic bound, such which can be non-obvious to prove, and even more difficult to persuade the scientific community that it is true [20, 21].

In this paper, we focus on a fairly recent algorithm called Filter Sort [34]. Although this algorithm bears a large resemblance with Best Order Sort, which is $O(MN^2)$, it loses this worst-case bound in an attempt to speed-up. We present the test scenario which requires this algorithm to perform $\Omega(N^3)$ dominance comparisons between the points; however, because of a particular property of this algorithm, we cannot guarantee that even a constant fraction of these comparisons will take $\Omega(M)$ time, so our running time lower bound is as well $\Omega(N^3)$.

The rest of the paper is structured as follows. Section 2 explains Filter Sort in necessary detail. Then we present our test scenario in Section 3 and show that Filter Sort runs in $\Omega(N^3)$ on this scenario. Finally, we conclude the paper and discuss the consequences of our results in Section 4.

2 Filter Sort

In this section, we shortly discuss Filter Sort, which is outlined in Algorithm 1. This algorithm is based on an idea that a solution that minimizes any linear combination of objectives, or even of functions that grow monotonically with an index of an objective, cannot be dominated. Furthermore, if such a function is chosen to be noticeably different from each objective, one can efficiently filter (hence the name) the solutions that can be non-dominated assuming there is a pre-sorted list of solutions for each objective. In Filter Sort, the sum of ranks of solution's objectives is chosen as such a linear combination, which is arguably a choice that requires as few assumptions as possible.

Algorithm 1 Filter Sort

```
Require: $\mathbb{P} = \{\mathbb{P}_1, \mathbb{P}_2, \ldots, \mathbb{P}_N\}$: point in $M$-dimensional space
Ensure: $\mathbb{F} = \{\mathbb{F}_1, \mathbb{F}_2, \ldots\}$: points from $\mathbb{P}$ split into fronts
 1: for $j \in [1..M]$ do                                          ▷ Phase 1: Pre-sorting
 2:     $\mathbb{O}_j \leftarrow \mathbb{P}$ sorted by objective $j$, compared lexicographically if equal
 3:     for $i \in [1..N]$ do
 4:         $\mathbb{I}_{ij} \leftarrow$ index of $\mathbb{P}_i$ in $\mathbb{O}_j$
 5:     end for
 6: end for
 7: for $i \in [1..N]$ do                              ▷ Phase 2: Finding objective statistics
 8:     $\mathbb{B}_{\mathbb{P}_i} \leftarrow \arg\min_j \mathbb{I}_{ij}$      ▷ Find the best objective of $\mathbb{P}_i$ according to its index
 9:     $\mathbb{W}_{\mathbb{P}_i} \leftarrow \arg\max_j \mathbb{I}_{ij}$      ▷ Find the worst objective of $\mathbb{P}_i$ according to its index
10:     $\mathbb{S}_{\mathbb{P}_i} \leftarrow \sum_j \mathbb{I}_{ij}$                        ▷ Find the sum of objective indices
11: end for
12: $\mathbb{T} \leftarrow \mathbb{P}$ sorted by $\mathbb{S}$                                  ▷ Phase 3: Creating filters
13: for $r \in \{1, 2, \ldots\}$ do                                   ▷ Phase 4: Actual sorting
14:     if $|\mathbb{T}| = 0$ then
15:         break                                                ▷ No more solutions left
16:     end if
17:     $t \leftarrow \mathbb{T}_1$                      ▷ Choose filter solution with the smallest index sum
18:     $C \leftarrow \emptyset$                                  ▷ Candidate solutions, initially empty
19:     for $j \in [1..M]$ do
20:         $k \leftarrow 1$
21:         while $\mathbb{O}_{jk} \neq t$ do      ▷ Add all solutions from $\mathbb{O}_j$ preceding $t$ to candidates
22:             $C \leftarrow C \cup \{\mathbb{O}_{jk}\}, k \leftarrow k + 1$
23:         end while
24:     end for
25:     $\mathbb{F}_r \leftarrow \mathbb{F}_r \cup \{t\}$, remove $t$ from $\mathbb{T}$ and $\mathbb{O}_j$, $j \in [1..M]$          ▷ Rank and remove $t$
26:     for $c \in C$ do                                ▷ Try each candidate for being non-dominated
27:         isDominated $\leftarrow$ FALSE, $k \leftarrow 1$, $b \leftarrow \mathbb{B}_c$, $w \leftarrow \mathbb{W}_c$
28:         $L \leftarrow \mathbb{O}_b$        ▷ Compare $c$ with the shortest list of maybe-dominating points
29:         while $L_k \neq c_b$ do                         ▷ When $c$ is hit, the rest cannot dominate
30:             if $(L_k)_w \leq c_w$ and $L_k \prec c$ then            ▷ Check the worst objective first
31:                 isDominated $\leftarrow$ TRUE, break
32:             end if
33:             $k \leftarrow k + 1$
34:         end while
35:         if isDominated then
36:             $C \leftarrow C \setminus \{c\}$                                   ▷ Remove $c$ if it was dominated
37:         end if
38:     end for
39:     for $c \in C$ do
40:         $\mathbb{F}_r \leftarrow \mathbb{F}_r \cup \{c\}$, remove $c$ from $\mathbb{T}$ and
41:     end for
42: end for
```

The first three phases are rather straightforward. Phase 1 (lines 1–6 in Algorithm 1) performs sorting of the population by each of the objectives, using lexicographical sorting in the case of ties. This phase can be done in $O(MN \log N)$ using a quicksort-like $O(MN + N \log N)$ algorithm for lexicographical sorting, that also sorts the points by the first objective, and $M - 1$ runs of any efficient sorting algorithm in $O(N \log N)$. The points sorted by the j-th objective are stored in $\mathbb{O}_j$. During this phase, the indices of each point i in the sorted order along each objective j are stored in $\mathbb{I}_{ij}$, which can easily be done from within the sorting algorithms.

Phase 2 (lines 7–11) computes, using the indices $\mathbb{I}_{ij}$ from the previous stage, the best objective of each solution (that is, the objective, for which this point comes earlier in the corresponding list $\mathbb{O}_j$), the worst objective, and the sum of objective indices. This phase is done in $O(MN)$. Next, Phase 3 (line 12) sorts the population according to the sum of objective indices, again in $O(N \log N)$, and stores the result in a list $\mathbb{T}$.

Note that the lists $\mathbb{O}_j$ and $\mathbb{T}$ subsequently require fast removal of elements from arbitrary locations. One of the possible choices is to create them as doubly linked lists, or to convert them to such lists soon after creation, for which the most efficient implementation would probably be to store the next/previous pointers in the point itself. An alternative solution would be to use auxiliary Boolean arrays that store whether a solution was deleted, and to compact the arrays and the (non-linked) lists when enough solutions are removed.

Finally, the actual non-dominated sorting happens in Phase 4 (lines 13–42). If there are any remaining solutions, the filter solution t is first chosen as the first solution in the list $\mathbb{T}$. As no other solutions have a smaller sum of objective ranks, t is guaranteed to be non-dominated. Then, in lines 18–24, the algorithm collects the solutions which precede t in at least one objective by joining the corresponding prefixes of all $\mathbb{O}_j$ for each objective j, effectively filtering out all the solutions that are dominated by t. Next, the filter solution t is removed from all the lists and is added to the currently populated front $\mathbb{F}_r$. Finally, each of the candidates c is tested for non-dominance. For that, c is compared with all the solutions that come before c in the objective list $\mathbb{O}_b$, where $b = \mathbb{B}_c$ is the best objective of c (populated in line 8 in Algorithm 1). To further speed-up the comparison, first the comparison in the worst objective of c is performed, as if c is not dominated in this objective, then it is not dominated at all. All the candidate points that passed the non-dominance checks are also added to $\mathbb{F}_r$, after which this front is declared complete.

3 Worst-Case Running Time Analysis

Now we turn to the worst-case running time analysis of Filter Sort. Our analysis consists of a nearly-trivial upper bound and a much more involved lower bound, which we state as two separate lemmas.

Lemma 1. *The worst-case running time of Filter Sort is $O(MN^3)$.*

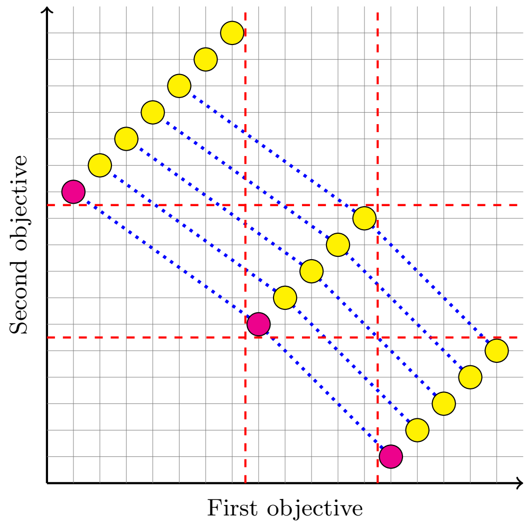

Fig. 1. Test example for $N = 17$, $M = 2$. Points from the same front are connected with blue lines

Proof. This follows from the simple static analysis of Algorithm 1. Indeed, Phases 1–3 require $O(MN \log N)$ time in common. The number of iterations of the main loop (lines 13–41) in Phase 4 coincides with the number of reported fronts, which is $O(N)$. In each iteration, the time spent in lines 14–25, as well as 39–41, cannot exceed $O(MN)$. The size of the candidate set C is at most $N-1$, the number of iterations of the while-loop in lines 29–33 is at most $N-1$ since $\mathbb{O}_b$ cannot contain more than N points, and the dominance comparison in line 30 cannot take more than $O(M)$ time.

In total, each loop in lines 29–34 is at most $O(MN)$, each iteration in lines 26–38 is at most $O(MN^2)$, and the whole algorithm cannot take more than $O(MN^3)$ time. □

We proceed with the lower bound. We first present the analysis for $M = 2$ and then we produce a hard input for any $M > 1$ based on this analysis.

Lemma 2. *There exists an input* $\mathbb{P}$ *with* N *two-dimensional points which requires Filter Sort to run for* $\Omega(N^3)$ *time.*

Proof. In the proof, we use the notation (x, y) to denote a two-dimensional point with objective values x and y. We assume $N_3 = \lfloor \frac{N-1}{3} \rfloor$ and use the test consisting of three sets of points as below, depicted on Figure 1:

– left points: $(i, 2N_3 + i)$ for $i = 1, \ldots, N - 2N_3$;
– middle points: $(N - 2N_3 + i, N_3 + i)$ for $i = 1, \ldots, N_3$.
– right points: $(N - N_3 + i, i)$ for $i = 1, \ldots, N_3$;

We chose N_3 this way so that $N - N_3 > 2N_3$, that is, the number of left points is always greater than the number of middle points and of right points, which is crucial in our analysis.

Note that this test example has exactly $N - 2N_3$ fronts, however, only the first N_3 of them are the complete fronts that consist of three points each. What is more, as long as $N \geq 4$, when we compute and remove the first front, the remaining test would be essentially the same test for $N' = N - 3$ points and larger gaps between the point groups, which does not influence the way Filter Sort works. This consideration makes our analysis much simpler.

The three points that compete for being a filter element are the first left point $(1, 2N_3+1)$, the first middle point $(N-2N_3+1, N_3+1)$ and the first right point $(N - N_3 + 1, 1)$, which are highlighted in Figure 1. With our choice of objective values, the sum of ranks is the same as the sum of objectives themselves, hence the best middle and right points have these sums equal to $N - N_3 + 2$ and the best left point has the sum equal to $2N_3 + 2$. As, per our choice, $N - N_3 > 2N_3$, the best left point is unambiguously chosen as the filter point t.

Next, Filter Sort constructs the set of candidate solutions. By our construction, every middle and every right point has the smaller second objective than t, so these points constitute the candidate set C.

To prove that the loop at lines 26–38 requires $\Theta(N^2)$ point comparisons, we note that the best objective of every middle point is the second objective, since the offset in the second objective is N_3 and in the first objective it is $N - 2N_3$, which is greater. For this reason, every middle point would necessary be compared with every right point, which yields $N_3^2 = \Theta(N^2)$ comparisons. Note that each such comparison terminates early and costs $O(1)$, because the worst objective of each middle point is the first objective, and in this objective every middle point is better than every right point.

As a result, sorting of the entire input of this sort would require at least

$$\sum_{i=1}^{\lfloor \frac{N-1}{3} \rfloor} i^2 = \frac{\lfloor \frac{N-1}{3} \rfloor (\lfloor \frac{N-1}{3} \rfloor + 1)(2\lfloor \frac{N-1}{3} \rfloor + 1)}{6} = \frac{N^3}{81} + O(N^2)$$

point comparisons and running time. □

Lemma 3. *There exists an input $\mathbb{P}$ with N points of dimension M which requires Filter Sort to run for $\Omega(N^3)$ time.*

Proof. We first construct an auxiliary set of N two-dimensional points $\mathbb{Q}$ using the method provided in Lemma 2. Next, we define each point $\mathbb{P}_i$ as follows:

- for $1 \leq j \leq M - 1$, $\mathbb{P}_{ij} \leftarrow \mathbb{Q}_{i1}$;
- for $j = M$, $\mathbb{P}_{ij} \leftarrow \mathbb{Q}_{i2}$.

In this case, Filter Sort will still select the filter element from the equivalent of left points, since the objective index sum would be the smallest for such a point. For every equivalent of a middle point, the best objective would be the last one, and the worst objective would be any objective except the last one. As a result, Filter Sort would make exactly the same choices for $\mathbb{P}$ as it would do for $\mathbb{Q}$, hence it will also make $\Omega(N^3)$ point comparisons for the input $\mathbb{P}$. $\square$

Now we can formulate and prove the main theorem of the paper.

Theorem 1. *The worst-case running time of Filter Sort is $\Omega(N^3)$ and $O(MN^3)$.*

Proof. The upper bound is proven in Lemma 1 and the lower bound is proven in Lemma 3. $\square$

4 Conclusion and Discussion

We have proven that Filter Sort, despite the reports on its wall-clock time efficiency compared to some other algorithms, can be forced to perform $\Omega(N^3)$ dominance comparisons, which is much worse than $O(MN^2)$ ensured by many other algorithms.

As a result, we suggest that the authors of evolutionary multiobjective software use Filter Sort with caution (if at all) even if they like its typical performance. One recipe would be to track the number of dominance comparisons and switch to any algorithm that is less efficient in average, but has asymptotically better worst-case running time, for example, from the ENS family [35]. The availability of the non-modified algorithm that can be forced to work much slower than expected is, in fact, a security breach that can exposes a DoS-attack in the case the evolutionary multiobjective software is accessible as a service.

Concerning the possible improvements of Filter Sort, we do not currently have much to propose. One of the important weaknesses is that the list L of the points which may dominate the current candidate c, as in line 28 of Algorithm 1, may contain former candidate solutions that have already been dominated by some other candidate solution. One can get rid of that by making deep copies of all the lists $\mathbb{O}_j$ before line 26, using these copies in line 28 instead of the originals, and removing the former candidate solutions from these copies in line 36 together with the removal from the set of candidate solutions. However, just making these copies, although taking at most $O(MN^2)$ total time, may introduce a huge overhead in typical scenarios, essentially destroying the "average" benefits of Filter Sort.

It is currently an open question whether $\Omega(N^3)$ is the best lower bound we can prove (e.g. there is a matching $O(N^3 + MN^2)$ bound), or our principle of constructing hard test cases is not the best one, and a strictly better lower bound holds. It appears now that tracking the worst objective and using it first to compare points is a crucial component of Filter Sort that makes it harder to propose $\Omega(MN^3)$ tests. However, we find it difficult now to prove or disprove that the $O(N^3 + MN^2)$ bound actually holds.

Acknowledgment. This research is financially supported by The Russian Science Foundation, Agreement No. 17-71-30029 with co-financing of Bank Saint Petersburg.

References

1. Borzsony, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings of 17th International Conference on Data Engineering. pp. 421–430. IEEE (2001)
2. Buzdalov, M.: Generalized offline orthant search: One code for many problems in multiobjective optimization. In: Proceedings of Genetic and Evolutionary Computation Conference. pp. 593–600. ACM (2018)
3. Buzdalov, M.: Make evolutionary multiobjective algorithms scale better with advanced data structures: Van Emde Boas tree for non-dominated sorting. In: 10th International Conference on Evolutionary Multi-Criterion Optimization, pp. 66–77. No. 11411 in Lecture Notes in Computer Science, Springer (2019)
4. Buzdalov, M., Shalyto, A.: A provably asymptotically fast version of the generalized Jensen algorithm for non-dominated sorting. In: Parallel Problem Solving from Nature — PPSN XIII, pp. 528–537. No. 8672 in Lecture Notes in Computer Science, Springer (2014)
5. Coello Coello, Carlos A. Toscano Pulido, G.: A micro-genetic algorithm for multiobjective optimization. In: 1st International Conference on Evolutionary Multi-Criterion Optimization, pp. 126–140. No. 1993 in Lecture Notes in Computer Science, Springer (2001)
6. Corne, D.W., Knowles, J.D., Oates, M.J.: The Pareto envelope-based selection algorithm for multiobjective optimization. In: Parallel Problem Solving from Nature – PPSN VI. pp. 839–848. No. 1917 in Lecture Notes in Computer Science, Springer (2000)
7. Deb, K., Hussein, R., Roy, P., Toscano, G.: Classifying metamodeling methods for evolutionary multi-objective optimization: First results. In: 9th International Conference on Evolutionary Multi-Criterion Optimization, pp. 160–175. No. 10173 in Lecture Notes in Computer Science, Springer (2017)
8. Deb, K., Jain, H.: An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints. IEEE Transactions on Evolutionary Computation **18**(4), 577–601 (2014)
9. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation **6**(2), 182–197 (2002)
10. Gustavsson, P., Syberfeldt, A.: A new algorithm using the non-dominated tree to improve non-dominated sorting. Evolutionary Computation **26**(1), 89–116 (2018)
11. Handl, J., Knowles, J.: Exploiting the trade-off-the benefits of multiple objectives in data clustering. In: 3rd International Conference on Evolutionary Multi-Criterion Optimization, pp. 547–560. No. 3410 in Lecture Notes in Computer Science, Springer (2005)
12. Jensen, M.T.: Reducing the run-time complexity of multiobjective EAs: The NSGA-II and other algorithms. IEEE Transactions on Evolutionary Computation **7**(5), 503–515 (2003)
13. Knowles, J., Corne, D.: M-PAES: A memetic algorithm for multiobjective optimization. In: Proceedings of IEEE Congress on Evolutionary Computation. vol. 1, pp. 325–332. IEEE (2000)
14. Knowles, J.D., Corne, D.W.: Approximating the nondominated front using the Pareto archived evolution strategy. Evolutionary Computation **8**(2), 149–172 (2000)

15. Kung, H.T., Luccio, F., Preparata, F.P.: On finding the maxima of a set of vectors. Journal of the ACM **22**(4), 469–476 (1975)
16. Leyton-Brown, K., Shoham, Y.: Essentials of game theory: A concise multidisciplinary introduction. Synthesis Lectures on Artificial Intelligence and Machine Learning **2**(1), 1–88 (2008)
17. Lou, R.D., Sarrafzadeh, M.: An optimal algorithm for the maximum three-chain problem. SIAM Journal on Computing **22**(5), 976–993 (1993)
18. Markina, M., Buzdalov, M.: Hybridizing non-dominated sorting algorithms: Divide-and-conquer meets Best Order Sort. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 153–154. ACM (2017)
19. Markina, M., Buzdalov, M.: Towards large-scale multiobjective optimisation with a hybrid algorithm for non-dominated sorting. In: Parallel Problem Solving from Nature — PPSN XV, vol. 1, pp. 347–358. No. 11101 in Lecture Notes in Computer Science, Springer (2018)
20. McClymont, K., Keedwell, E.: Deductive Sort and Climbing Sort: New methods for non-dominated sorting. Evolutionary Computation **20**(1), 1–26 (2012)
21. Mishra, S., Buzdalov, M.: If unsure, shuffle: Deductive sort is $\Theta(MN^3)$, but $O(MN^2)$ in expectation over input permutations. In: Proceedings of Genetic and Evolutionary Computation Conference. ACM (2020). https://doi.org/10.1145/3377930.3390246, accepted for publication
22. Mishra, S., Buzdalov, M., Senwar, R.: Time complexity analysis of the dominance degree approach for non-dominated sorting. In: Proceedings of Genetic and Evolutionary Computation Conference Companion. ACM (2020). https://doi.org/10.1145/3377929.3389900, accepted for publication
23. Mishra, S., Mondal, S., Saha, S., Coello Coello, C.A.: GBOS: Generalized Best Order Sort algorithm for non-dominated sorting. Swarm and Evolutionary Computation **43**, 244–264 (2018)
24. Mishra, S., Saha, S., Mondal, S.: Divide and conquer based non-dominated sorting for parallel environment. In: Proceedings of IEEE Congress on Evolutionary Computation. pp. 4297–4304. IEEE (2016)
25. Mishra, S., Saha, S., Mondal, S.: MBOS: Modified Best Order Sort algorithm for performing non-dominated sorting. In: Proceedings of IEEE Congress on Evolutionary Computation. pp. 725–732. IEEE (2018)
26. Mishra, S., Saha, S., Mondal, S., Coello Coello, C.A.: A divide-and-conquer based efficient non-dominated sorting approach. Swarm and Evolutionary Computation **44**, 748–773 (2019)
27. Moreno, J., Rodriguez, D., Nebro, A.J., Lozano, J.A.: Merge nondominated sorting algorithm for many-objective optimization. IEEE Transactions on Cybernetics (2020). https://doi.org/10.1109/TCYB.2020.2968301, accepted for publication
28. Roy, P., Hussein, R., Deb, K.: Metamodeling for multimodal selection functions in evolutionary multi-objective optimization. In: Proceedings of Genetic and Evolutionary Computation Conference. pp. 625–632. ACM (2017)
29. Roy, P.C., Deb, K.: High dimensional model representation for solving expensive multi-objective optimization problems. In: Proceedings of IEEE Congress on Evolutionary Computation. pp. 2490–2497. IEEE (2016)
30. Roy, P.C., Deb, K., Islam, M.M.: An efficient nondominated sorting algorithm for large number of fronts. IEEE Transactions on Cybernetics **49**(3), 859–869 (2019)
31. Roy, P.C., Islam, M.M., Deb, K.: Best Order Sort: A new algorithm to non-dominated sorting for evolutionary multi-objective optimization. In: Proceedings of Genetic and Evolutionary Computation Conference Companion. pp. 1113–1120. ACM (2016)

32. Srinivas, N., Deb, K.: Multiobjective optimization using nondominated sorting in genetic algorithms. Evolutionary Computation **2**(3), 221–248 (1994)
33. Tang, S., Cai, Z., Zheng, J.: A fast method of constructing the non-dominated set: Arena's principle. In: 4th International Conference on Natural Computation. pp. 391–395. IEEE (2008)
34. Wang, J., Li, C., Diao, Y., Zeng, S., Wang, H.: An efficient nondominated sorting algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 203–204. ACM (2018)
35. Zhang, X., Tian, Y., Cheng, R., Jin, Y.: An efficient approach to nondominated sorting for evolutionary multiobjective optimization. IEEE Transactions on Evolutionary Computation **19**(2), 201–213 (2015)
36. Zhang, X., Tian, Y., Cheng, R., Jin, Y.: A decision variable clustering-based evolutionary algorithm for large-scale many-objective optimization. IEEE Transactions on Evolutionary Computation **22**(1), 97–112 (2018)
37. Zhang, X., Tian, Y., Cheng, R., Yaochu, J.: An efficient approach to nondominated sorting for evolutionary multiobjective optimization. IEEE Transactions on Evolutionary Computation **19**(2), 201–213 (2015)
38. Zhang, X., Tian, Y., Jin, Y.: A knee point-driven evolutionary algorithm for many-objective optimization. IEEE Transactions on Evolutionary Computation **19**(6), 761–776 (2015)
39. Zhou, Y., Chen, Z., Zhang, J.: Ranking vectors by means of the dominance degree matrix. IEEE Transactions on Evolutionary Computation **21**(1), 34–51 (2017)