Best Order Sort: A New Algorithm to Non-dominated Sorting for Evolutionary Multi-objective Optimization

Proteek Chandan Roy
Computer Science and Engineering
Michigan State University
1220 Trowbridge Rd, East Lansing
Michigan, USA
royprote@msu.edu

Md. Monirul Islam
Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka-1000, Bangladesh
monirultalha@gmail.com

Kalyanmoy Deb
Electrical and Computer Engineering
Michigan State University
1220 Trowbridge Rd, East Lansing
Michigan, USA
kdeb@egr.msu.edu

ABSTRACT

Finding the non-dominated sorting of a given set vectors has applications in Pareto based evolutionary multi-objective optimization (EMO), finding convex hull, linear optimization, nearest neighbor, skyline queries in database and many others. Among these, EMOs use this method for survival selection. The worst case complexity of this problem is found to be $O(N\log^{M-1} N)$ when the number of objectives M is constant and the size of solutions N is varying. But this bound becomes too large when M depends on N. In this paper we are proposing a new algorithm with worst case complexity $O(MN\log N + MN^2)$, however, with reduced running time in many objective cases. This algorithm can make use of the faster implementation of sorting algorithms. It removes unnecessary comparisons among the solutions which improves the running time. The proposed algorithm is compared with four other competing algorithms on three different datasets. Experimental results show that our approach, namely, best order sort (BOS) is computationally more efficient than all other compared algorithms with respect to running time.

1. INTRODUCTION

In many fields of study such as evolutionary multi-objective optimization, computational geometry, economics, game theory and databases, the concept of non-dominated sorting or Pareto set is used. By definition, a vector valued point or solution $A = (a_1, a_2, \ldots, a_M)$ dominates another solution $B = (b_1, b_2, \ldots, b_M)$ by Pareto-dominance relation if A is better or equal in each dimension or objective than that of B. In other words, $a_i \geq b_i \;\; \forall i = 1, \ldots, M$. Here M is the number of objectives or the dimension of a vector. Given a set of solutions P, finding the solutions which are not dominated by any other solutions in that set is called the problem of finding Pareto set and these solutions are denoted as rank 1 solutions. If we remove rank 1 solutions from P and find the Pareto set again we will end up finding rank 2 solutions. We can perform this process repeatedly until all the solutions are ranked. This process is called non-dominated sorting or non-dominated ranking. Each solution having rank $r > 1$ is dominated by at least one solution of rank $(r-1)$. In Fig. 1 there are four non-dominated fronts: $\{a, b, c, f\}, \{h, e\}, \{g\}$ and $\{d\}$ of rank 1, 2, 3 and 4 respectively.

Some of the most important EMO algorithms use the idea of non-dominated sorting for their survival selection. Among them, we can refer NSGA [23], NSGA-II [7], SPEA2 [28], MOGA [21], NPGA [14], PAES [17], MOPSO [5] and recently published NSGA-III [6], DM1 [1] and EPCS [22]. Non-dominated sorting takes most of the time of these optimization algorithms. So it is very important to find efficient algorithm for ranking.

The rest of the paper is organized as follows. Section 2 discusses about some of the approaches for finding Pareto set and non-dominated sorting. Time complexities of those approaches are also discussed. In Section 3, we talk about the main idea behind the proposed algorithm. Proof of correct-

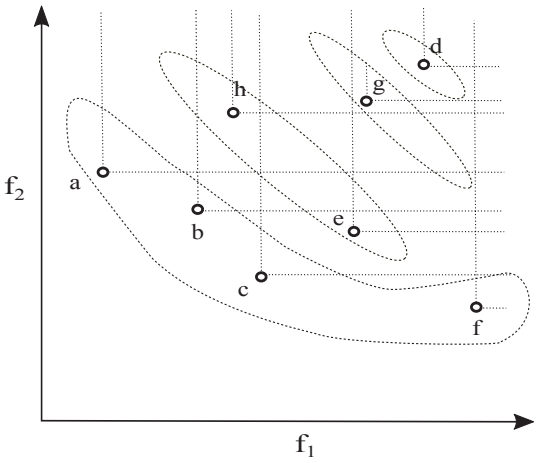


Figure 1: An example with eight points in a two dimensional minimization problem. It has four fronts $\{a, b, c, f\}$, $\{h, e\}$, $\{g\}$ and $\{d\}$ which should be found by a non-dominated sorting algorithm.

GECCO'16 Companion, July 20-24, 2016, Denver, CO, USA

DOI: http://dx.doi.org/10.1145/2908961.2931684

ness, best case time complexity and space complexity is also presented in that section. The comparison with four well known algorithms are presented in Section 4 and results are discussed in Section 5. Section 6 concludes the paper with overall remarks and future work.

2. RELATED WORK

The methods for finding Pareto set and non-dominated sorting can be divided into two categories – sequential approach and divide-and-conquer approach. Given a set of vectors, sequential brute force method for finding the Pareto set is to compare each solution to every other solution to check whether they dominate each other. If one solution is dominated by another one, then it is removed from the current set. This algorithm can be used to find the non-dominated sorting by repeating the process and removing the ranked solutions or points from the set [23]. The algorithm has $O(MN^3)$ complexity because of repeated comparisons. Due to high computational complexity of [23], Deb *et al.* [7] described a computationally faster version, termed as fast non-dominated sort, whose complexity is $O(MN^2)$. It uses the fact that pairwise comparisons can be saved and used later to find the rank of solutions other than the Pareto set. However, space complexity is $O(N^2)$ because it saves pairwise comparisons.

McClymont and Keedwell [19] described a set of algorithms which improves the space complexity to $O(N)$ with similar time complexity $O(MN^2)$. Among them deductive sort is reported to work best. This algorithm consists of multiple passes and one pass is completed by removing dominated solutions with an arbitrary unranked solution. Corner sort [25] is a new approach to find non-dominated sorting of vectors which works similar to deductive sort. But instead of choosing an arbitrary vector for checking dominance, it always chooses a vector which is guaranteed to be in current rank. It works better than deductive sort in some cases but has the same worst case complexity. Recently, an efficient approach of non-dominated sorting called ENS [27] has been described. This method uses the idea of sorting the population with respect to their first objective values by in-place heap sort. In case of tie, the authors use lexicographic ordering. This algorithm can achieve best case complexity $O(MN \log N)$ although it has $O(MN^2)$ in worst cases. Other methods, e.g. dominance tree based non-dominated sorting [10], non-dominated rank sort or omni-optimizer [9] and arena's principle [24] can also be used to improve the best case time complexity upto $O(MN\sqrt{N})$. However, the worst case time complexity remains $O(MN^2)$. Recently, parallel GPU based NSGA-II algorithm [13] has been proposed to speed up the non-dominated sorting and other steps of the evolutionary algorithm.

Unlike the sequential algorithms, the set of divide-and-conquer algorithms work by repeatedly dividing the data using objective values. These methods are asymptotically faster than sequential ones in the worst case for fixed number of objectives. The first divide-and-conquer method was proposed by Kung *et al.* [18] for finding Pareto set. This method is later analyzed by Bentley [2]. This algorithm divides the data and reduces dimensions recursively. When the remaining dimensions become less than three, a special-case algorithm is applied that has complexity $O(N \log N)$. If the dimension is not fixed then its complexity is bounded by $O(MN^2)$ [4, 20]. These algorithms exhibit many unnecessary comparisons which increases with the number of objectives [12]. The space complexity is said to be $O(N)$ for Kung's algorithm. Bentley [3] improved the average case of this divide-and-conquer algorithm. It assumes the fact that size of Pareto set of vectors is equal to $O(\log^{M-1} N)$ on average. One can find the non-dominated sorting by repeating Kung's algorithm the number of times equal to number of ranks which gives complexity $O(N^2 \log^{M-2} N)$. By removing the repeated comparisons, Jensen [16] and Yukish [26] both extended Kung's algorithm to find the Pareto set of vectors and perform non-dominated sorting in time $O(N \log^{M-1} N)$. Jensen's algorithm assumes that, for any objective, no two vectors have the same objective value [10]. Because of this assumption it generates different Pareto ranking from the baseline algorithm of NSGA-II [7]. It was corrected later in [4, 11]. Buzdalov *et al.* proved the time complexity of non-dominated sorting to be $O(N \log^{M-1} N)$ for fixed dimension. Our aim is to find better algorithms in terms of N and M both.

3. PROPOSED METHOD

3.1 Basic Idea

In this section we propose an algorithm named best order sort (BOS) which reduces the number of comparisons for sorting. The main idea of the algorithm is described in Fig. 2. For each solution s, we can get a set of solutions those are not worse than s in a particular objective. So there will be M such sets for M objectives. To find the rank of s, only one of the sets, T is sufficient to be considered. Some members of the set T dominate s while others are non-dominated with s. Suppose highest rank of that dominating subset is r. The rank of s will then be $(r + 1)$. This is because, if a solution is dominated by a set of points, then rank of that solution is one plus highest rank of that set. Although any of the M sets can be considered for sorting, our method finds the smallest set by sorting the population with their objective values.

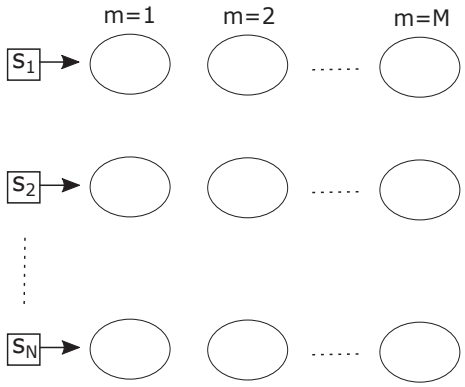


Figure 2: The basic idea of the proposed method is that there are M sets for each solution which denote the 'not-worse' solutions in corresponding objective. The algorithm finds the smallest set to compare and finds their ranks.

3.2 The Algorithm

At first, we discuss about the necessary data structures used in this algorithm. The algorithm starts with initial-

izing $N \times M$ empty sets denoted by L_i^j (see Algorithm 1). Here N is the size of population and M is the number of objectives. The algorithm saves the sorted population in Q_j where j-th objective value is used for sorting. It goes over each of the sorted lists and the solutions found in those lists are ranked and saved in L_j^r. Here L_j^r represents the set of solutions which have rank r and they are found in j-th objective. It maintains an objective list $C_u = \{1, 2, \ldots, M\}$ for each solution u. It signifies that, if we want to check whether a solution s is dominated by u then only the objectives in C_u needs to be compared. The variable $isRanked(s)$ remembers whether the solution s is ranked yet or not. We initialize the total number of solutions ranked SC to be zero. Ranks of solutions are saved in $R(s)$ for all $s \in P$. Number of fronts found so far $RC = 1$ as there will be at least one solution in the population.

At the beginning, we sort the solutions according to each objective j and put those into sorted list Q_j (see line 8 of Algorithm 1). We use lexicographic order if two objective values are same. In that case, if the first objective values are same then sorting will be based on the second objective value. Note that we just need to perform single lexicographic ordering for the first objective. We can then use the information of first objective to find lexicographic order of other objectives.

Algorithm 1: INITIALIZATION

```
   Data: Population P of size N and objective M
   Result: Sorted set of solutions in Q_j
   // global variables
1  L_j^i ← ∅,  ∀ j = 1,2,...,M,  ∀ i = 1,2,...,N;
   // Solution sets
2  C_i ← {1,2,...,M}  ∀ i = 1,2,...,N;      // comparison
   set
3  isRanked(P) ← false;   // solutions ranked or not
4  SC ← 0;     // number of solutions already ranked
5  RC ← 1;                          // at least one front
6  R(P) ← 0;                         // Rank of solutions
7  for j = 1 to M do
8  |  Q_j ← Sort P by j-th objective value, use
   |  lexicographic order in case of tie;
9  end
```

Algorithm 2 describes main procedure for finding the non-dominated sorting. In each step, it takes one solution from lexicographically sorted population Q_j for objective j. It takes the first element s from sorted list Q_1 which is denoted by $Q_1(1)$. Then it excludes objective $\{1\}$ from the list C_s. This is because, if other solution t is compared with s later, t is already dominated in objective 1. Next, the algorithm checks whether s is already ranked or not. If it is ranked then it will be included to the corresponding list $L_1^{R(s)}$. For instance, if s has to be included in L_2^5, then s's rank is 5 and it is found in second objective. L_j is the set of all required solutions to find rank of s (see Algorithm 3) because they are not worse than s in objective j. At the end of this algorithm, each solution should appear in every objective set L_j only once, if line 14 of Algorithm 2 is not executed.

If the solution s is not ranked then FINDRANK(s, j) procedure is called. It saves the rank of s in $R(s)$. After returning from this method, we assign $isRanked(s)$ to be true so that it never gets ranked again. We increment the number of solutions done (SC) by 1. The algorithm then goes through the next objective to find the next solution of corresponding list Q_j. After finishing loop at line 2-12, the algorithm then checks if number of solutions done is equal to total population N. If it is not, then it takes the next element from lists Q_j. Once all solutions are ranked it breaks out of the loop. Note that, a solution s is ranked when it is observed first time in one of the lists. Therefore, it guarantees to compare with the smallest set of solutions over all the lists obtained from each objective.

Algorithm 2: MAIN LOOP

```
   Data: Sorted Population, Q
   Result: Rank of each solution, R
1  for i = 1 to N do // for all solutions
2  |  for j = 1 to M do // for all sorted set
3  |  |  s ← Q_j(i) ;      // Take i-th element from Q_j
4  |  |  C_s ← C_s − {j};       // reduce comparison set
5  |  |  if isRanked(s) = True then
6  |  |  |  L_j^{R(s)} = L_j^{R(s)} ∪ {s};  // Include s to L_j^{R(s)}
7  |  |  else
8  |  |  |  FINDRANK(s, j);                 // Find R(s)
9  |  |  |  isRanked(s) ← True;     // non-dominated
   |  |  |  sorting done
10 |  |  |  SC ← SC + 1// total done
11 |  |  end
12 |  end
13 |  if SC = N then // if all solutions are done
14 |  |  break;                          // sorting ended
15 |  end
16 end
17 return R;                                        // return
```

Given the set of solutions already discovered in objective j in sets L_j^r, we can use Algorithm 3 to find out the rank of the solution found in objective j. Suppose s is discovered first time in objective j. The algorithm starts by comparing s with all the solutions t of first rank $L_j^k(k = 1)$. If s is not better in the objectives defined in C_t then it is dominated by t (see Algorithm 4). Then s will be compared to the next rank solutions L_j^{k+1}. If s is not dominated by any solution of some rank k, then its rank will be k. If s is dominated by at least one solution of all ranks then s is discovering a new front and its rank will be $RC + 1$. At the end of this procedure we will find rank of s, update rank count RC and update set $L_j^{R(s)}$. Algorithm 4 describes the procedure of checking Pareto-domination with sets C. The algorithm finishes execution as soon as all the solutions are ranked. In the next section, we will see an example describing this algorithm.

3.3 Illustrative Example

In Fig. 1, population with eight individuals (namely a, b, c, d, e, f, g and h) are shown graphically in a two objective minimization problem. The algorithm sorts the solutions by objectives f_1 and f_2 and puts them into Q sets (Fig. 3). After sorting, the algorithm takes the elements in this order $a, f, b, c, h, e, c, b, e, a, g, h, d$ (see Fig. 3) to find their ranks. Solutions are ranked only when they are discovered for the first time in line 3 of MAIN LOOP. The ranked solutions are distributed in different sets L_1^1, L_1^2, L_2^1 etc. For solution d,

Algorithm 3: FINDRANK

Data: Solution s, List number j
Result: Rank of s

```
1  done = False;                                   // done bit
2  for k = 1 to RC do // for all discovered ranks
3  |  check = False;                              // check bit
4  |  for t ∈ L_j^k do // for all solutions in L_j^k
5  |  |  check ← DominationCheck(s, t);
6  |  |  if check = True then // if dominated
7  |  |  |  break;                               // break the loop
8  |  |  end
9  |  end
10 |  if check = False then // rank found
11 |  |  R(s) ← k;                                // update rank
12 |  |  done = True;                            // update done bit
13 |  |  L_j^R(s) = L_j^R(s) ∪ {s}// Include s to L_j^R(s)
14 |  |  break;                                   // break the loop
15 |  end
16 end
17 if done = False then // if not done
18 |  RC ← RC + 1;                                // update fronts count
19 |  R(s) ← RC;                                  // update rank
20 |  L_j^R(s) = L_j^R(s) ∪ {s};                  // Include s to L_j^R(s)
21 end
22 return
```

Algorithm 4: DOMINATIONCHECK

Data: Solution s and t
Result: True if t dominates s, false otherwise

```
1  for j ∈ C_t do // for all objectives in C_t
2  |  if s is better than t in objective j then
3  |  |  return False;                            // t cannot dominate s
4  |  end
5  end
6  return True;                                    // t dominates s
```

it will be compared with a, b, h, c, e and g which belongs to L_1^1, L_1^2 and L_1^3 (see Fig. 4). By FINDRANK method, d will be compared to L_1^1 first and then L_1^2 and finally L_1^3. L_1^4 will be discovered by d. While going over the solutions, the algorithm drops the corresponding objective from the comparison list (C) of that solution. C entries in four successive steps are shown in Fig. 5. After first step, objective 1 from a and objective 2 from f is dropped. At the end of 4th step, C_b and C_c becomes empty. It means that any solution which is discovered after this step will be considered dominated by b and c.

3.4 Correctness

LEMMA 1. *The proposed algorithm finds the correct rank of a solution $s \in P$.*

PROOF. From the definition of non-dominated sorting it is clear that if a solution s is dominated by a set of points $u \in U$ then the rank of s is one plus maximum rank of U. When a solution is ranked (line 8, Algorithm 2) then it is only compared to L_j sets of solutions. L_j sets contain only those solutions which are not worse than s in objective j. Because we know that only L_j solutions can dominate s, the algorithm FINDRANK ensures that s gets the rank one plus the highest rank of L_j. Inside FINDRANK algorithm, if we find that s is dominated by a solution of rank $k = 1$ then the algorithm moves to higher ranks ($k > 1$) to check for domination. Once a rank is found where no other solution of that rank dominates s, the correct rank of s is identified as the same rank as of those. If s is found to be dominated by at least one solution of each rank found so far, a new rank ($RC + 1$) is introduced with the solution s. In each case, comparing solutions of s have smaller set of

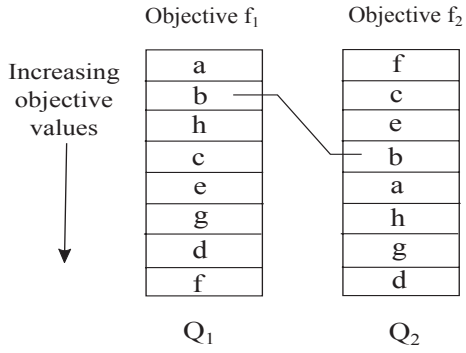


Figure 3: Sorted lists of population in different objectives. For solution b, it will be compared to only $\{a\}$, if we use objective f_1. The set becomes larger with $\{f, c, e\}$ when considering f_2. The proposed algorithm will use $\{a\}$ set because b will be found in f_1 for the first time.

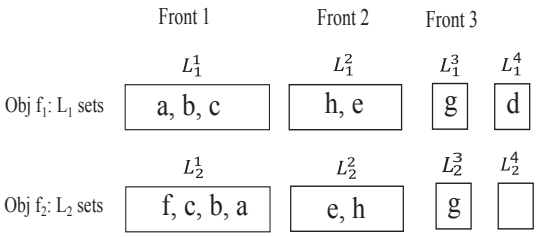


Figure 4: Four different fronts are discovered by the algorithm which are again distributed in four different fronts. Rank 1 solutions are given by union of L_1^1 and L_2^1 sets etc. The algorithm exits as soon as all the solutions are ranked, otherwise it would have been true that $L_1^1 = L_2^1$, $L_1^2 = L_2^2$ and so on.

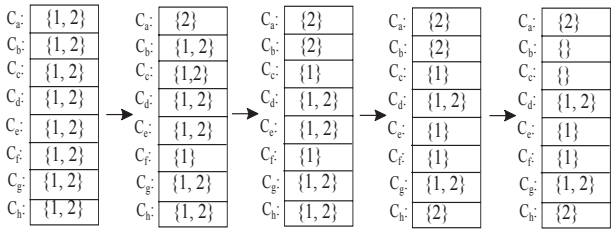
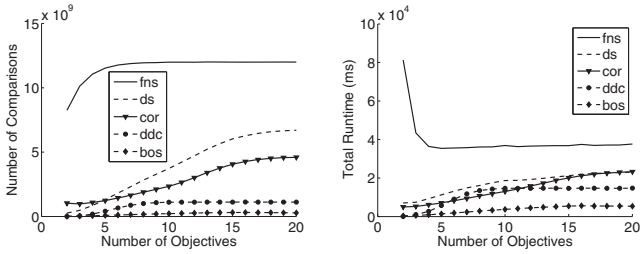


Figure 5: Four steps execution of the loop at line 1 in Algorithm 2 is shown. Dropping some objectives from the list indicates that a solution discovered later is already dominated on those objectives.



(a) Comparison (b) Time

Figure 6: Number of comparisons and runtime (in milliseconds) for cloud dataset of size 10,000 for increasing number of objectives. Results for fast non-dominated sort (fns), deductive sort (ds), corner sort (cor), divide-and-corner sort (ddc) and best order sort (bos) is shown.

objectives to compare. The objectives in which s is already dominated, are dropped from the objective list. Therefore each comparison is correct. Thus each solution finds its rank correctly. □

3.5 Time Complexity

Time complexity of the proposed method depends on the data structure and implementation. Our method tries to minimize the number of comparing solutions to find a rank. This comparing set is obtained by sorting the values of each objective. However, while we minimize the number of solutions to compare, running time of sorting part can be an overhead. Best case of this algorithm happens when the population has N fronts, each front having only one solution. In this case, we will get the similar order in all Q_j after sorting by Algorithm 1. While executing line 2 of Algorithm 2 with $j = 1$ up to $j = M$, all the objectives will be deleted (see line 4) from the objective list C_s. So there will be no objective value comparison. Total execution time for Algorithm 2 will be $O(MN)$. Therefore we get the best case time complexity $O(MN \log N)$. Average case analysis is left for future work.

4. EXPERIMENTAL RESULTS

We compared the proposed algorithm with four different algorithms – fast non-dominated sort [7], deductive sort [19], corner sort [25] and divide-and-conquer algorithm [4]. These algorithms are compared in cloud dataset, fixed front dataset and dataset obtained from multi-objective evolutionary algorithm (MOEA). Cloud dataset is a uniform random data. In fixed front data, the number of fronts (K) is controlled. We have used the procedure described in [25] for generating cloud and fixed front datasets. We vary size of population N from 500 to 10,000 with an increment of 500 in cloud dataset. In another test (Fig. 6), number of objectives are varied from 2 to 20 to evaluate performance with population size 10,000. For fixed front dataset, the number of fronts (K) is varied from 1 to 10 where number of solution is kept 10,000 with objectives 5, 10, 15 and 20. MOEA dataset is obtained by running 200 generations of NSGA-II algorithm with 800 population in DTLZ1 and DTLZ2 [8], WFG1 and WFG2 [15] problems with 5, 10, 15 and 20 objectives. We saved the MOEA data in files to run the algorithms. In these cases, all the parameter values are kept as standard ones. For example, simulated binary crossover with polynomial mutation are employed with probabilities 0.80 and (1/number of variables) respectively. For every dataset and every tested value of N and K, each algorithm was run on 30 different test instances to get the total running time. All the algorithms are optimized and implemented in Java Development Kit 1.8 update 65 and run in Dell computer with 3.2 GHz Intel core i7 and 64 bit Windows 7 machine. The source code of our algorithm can be found at GitHub [1]

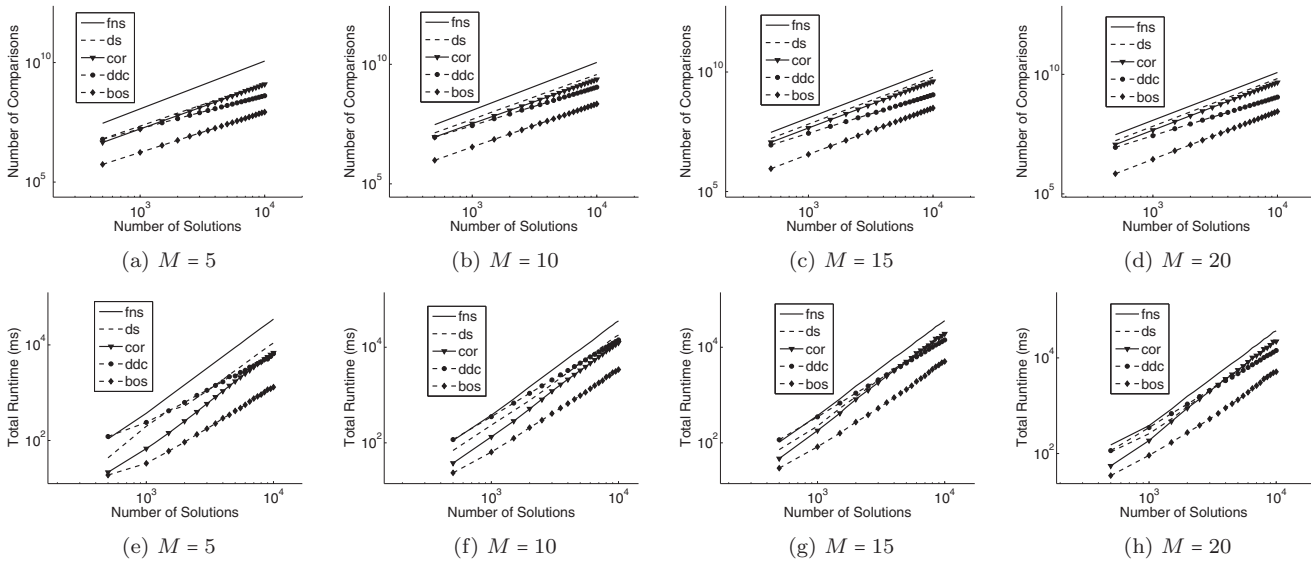
5. DISCUSSION

The results describe the average case behavior of the algorithms in three different cases. Fig. 6 shows that with increased number of objectives, number of comparisons and runtime increases for deductive sort, corner sort, divide-and-conquer sort and best order sort. Fast non-dominated sort performs worst in two objectives compare to other number of objectives (see Fig. 6). This is because, the number of fronts is very high in two objective random data and fast non-dominated sort takes most of the time (in milliseconds) just for saving dominated solutions in a list of size $O(N^2)$. This behavior is exhibited because of lower running time (few hundred milliseconds) and large amount of memory accesses (having list data structure). Best order sort performs the best followed by divide-and-conquer, corner sort and deductive sort. Log-based plots in Fig. 7 show that fast non-dominated sort has the highest and best order sort has the lowest order in terms of number of comparisons and runtime in objectives 5, 10, 15 and 20. Corner sort performs better than deductive sort in terms of comparisons in most of the cases but the runtime performance deteriorates with increasing number of objectives. Divide-and-conquer algorithm performs better than most sequential type algorithms in lower dimensions but it is worse when the number of objectives increases. The number of comparisons and runtime decreases with the increasing number of fronts (Fig. 8) except fast non-dominated sort and divide-and-conquer sort. In those two cases, runtime and number of comparisons increases with the increased number of fronts. Best order sort performs better than all other algorithms followed by corner sort and deductive sort respectively. In MOEAs (Table 1), divide-and-conquer algorithm has fewest number of comparisons in most of the cases but running time is slightly worse than best order sort. Best order sort becomes second in terms of comparisons followed by corner sort and deductive sort. Divide-and-conquer algorithm has advantage over data having small M and small N, for example, in MOEAs. Best order sort outperforms all the comparing algorithms in most of the cases.

6. CONCLUSION

In this paper we have proposed a non-dominated sorting algorithm for many objective evolutionary algorithms. Basic idea of the proposed method is to use faster sorting algorithms inside non-dominated sorting. The algorithm contains two distinguishable part – sorting by every dimension and ranking of the solutions, having upper bounds $O(MN \log N)$ and $O(MN^2)$ respectively. The experimental results show that this algorithm is very efficient in practice. This method can also be used to find first layer i.e. maximal vectors of a set of points. One drawback is that sorting time is increased with the number of objectives and it might exceed

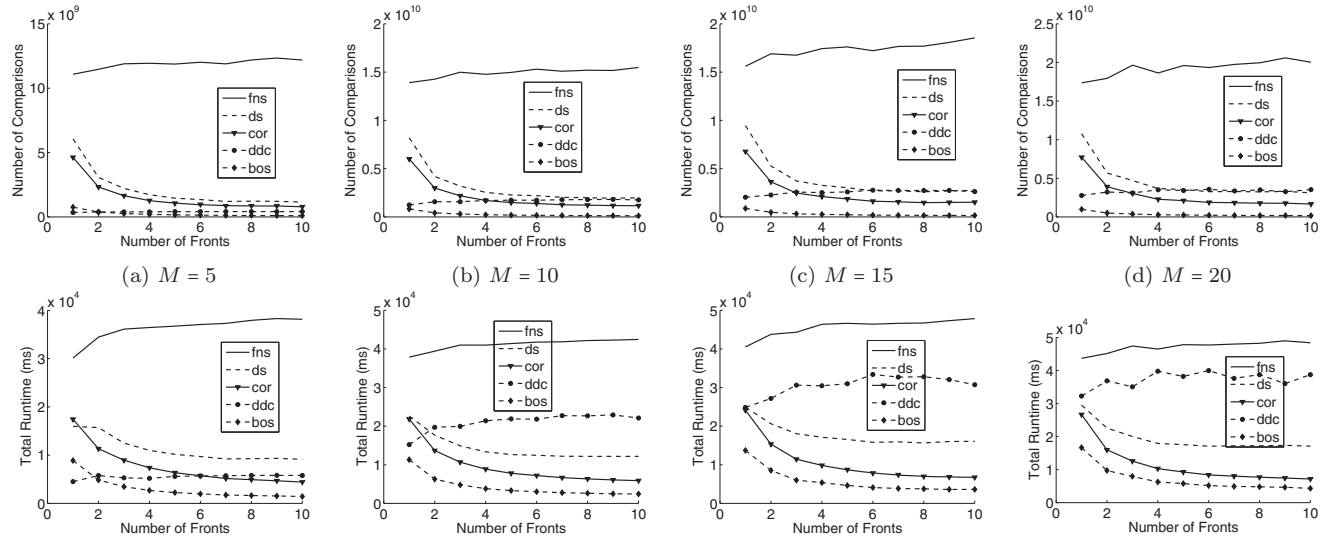
[1] https://github.com/Proteek/Best-Order-Sort



(a) $M = 5$ (b) $M = 10$ (c) $M = 15$ (d) $M = 20$

(e) $M = 5$ (f) $M = 10$ (g) $M = 15$ (h) $M = 20$

Figure 7: The number of comparisons and total runtime (in milliseconds) with increasing population size for cloud dataset in objectives 5, 10, 15 and 20. Results for fast non-dominated sort (fns), deductive sort (ds), corner sort (cor), divide-and-conquer sort (ddc) and best order sort (bos) are shown.



(a) $M = 5$ (b) $M = 10$ (c) $M = 15$ (d) $M = 20$

(e) $M = 5$ (f) $M = 10$ (g) $M = 15$ (h) $M = 20$

Figure 8: The number of comparisons and total runtime (in milliseconds) of 10,000 solutions with increasing number of fronts for fixed front dataset in objectives 5, 10, 15 and 20. Results for fast non-dominated sort (fns), deductive sort (ds), corner sort (cor), divide-and-conquer sort (ddc) and best order sort (bos) are shown.

the time for ranking. A good balance between these two parts should be identified. In future, the idea of this algorithm can be extended to parallel architecture. We would also like to find a progressive or incremental version of this algorithm.

Acknowledgment

This material is based in part upon work supported by the National Science Foundation under Cooperative Agreement No. DBI-0939454. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

[1] S. Adra and P. Fleming. Diversity management in evolutionary many-objective optimization. *Evolutionary Computation, IEEE Transactions on*, 15(2):183–195, April 2011.

[2] J. L. Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, Apr. 1980.

[3] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D.

Table 1: Total number of comparisons (#cmp) and total running time (in milliseconds) of 200 generations of data for DTLZ1, DTLZ2, WFG1 and WFG2 problems in 5, 10, 15 and 20 objectives.

Test Problem	**Obj.**	FNS		DS		COR		DDC		BOS	
		#cmp	time(ms)	#cmp	time(ms)	#cmp	time(ms)	#cmp	time(ms)	#cmp	time(ms)
DTLZ1	5	3.25e+08	1.03e+03	1.02e+08	4.48e+02	7.12e+07	2.18e+02	9.41e+06	3.79e+02	**7.95e+06**	**1.06e+02**
	10	5.13e+08	1.21e+03	2.74e+08	7.33e+02	1.71e+08	4.05e+02	**1.44e+07**	5.03e+02	2.03e+07	**2.53e+02**
	15	7.09e+08	1.41e+03	4.23e+08	1.02e+03	2.67e+08	5.52e+02	**1.50e+07**	5.42e+02	2.86e+07	**3.87e+02**
	20	8.98e+08	1.59e+03	5.67e+08	1.21e+03	3.55e+08	6.56e+02	**1.56e+07**	5.55e+02	3.51e+07	**4.72e+02**
DTLZ2	5	2.97e+08	8.59e+02	1.24e+08	4.77e+02	8.22e+07	2.58e+02	**9.52e+06**	3.52e+02	1.07e+07	**1.17e+02**
	10	4.30e+08	1.11e+03	2.31e+08	6.82e+02	1.59e+08	4.36e+02	**1.55e+07**	5.46e+02	1.80e+07	**2.35e+02**
	15	5.58e+08	1.27e+03	3.31e+08	8.37e+02	2.20e+08	5.41e+02	**1.63e+07**	5.84e+02	2.20e+07	**3.15e+02**
	20	6.95e+08	1.40e+03	4.34e+08	1.02e+03	2.81e+08	6.36e+02	**1.65e+07**	5.97e+02	2.49e+07	**3.73e+02**
WFG1	5	2.67e+08	7.99e+02	1.12e+08	4.38e+02	6.59e+07	2.44e+02	**9.89e+06**	3.53e+02	1.11e+07	**1.18e+02**
	10	2.95e+08	9.30e+02	1.47e+08	5.26e+02	1.03e+08	3.64e+02	**2.19e+07**	7.74e+02	2.09e+07	**2.63e+02**
	15	3.26e+08	9.65e+02	1.75e+08	5.85e+02	1.27e+08	4.47e+02	**2.41e+07**	8.74e+02	2.58e+07	**3.64e+02**
	20	3.57e+08	1.07e+03	2.00e+08	6.34e+02	1.47e+08	5.06e+02	**2.46e+07**	8.99e+02	2.91e+07	**4.50e+02**
WFG2	5	3.00e+08	9.18e+02	1.10e+08	4.69e+02	6.68e+07	2.06e+02	**9.64e+06**	3.55e+02	1.11e+07	**1.25e+02**
	10	5.56e+08	1.16e+03	2.80e+08	7.19e+02	1.78e+08	3.27e+02	**1.53e+07**	5.30e+02	3.02e+07	**3.28e+02**
	15	8.98e+08	1.52e+03	5.03e+08	1.06e+03	3.26e+08	4.71e+02	**1.58e+07**	5.53e+02	5.60e+07	**5.36e+02**
	20	1.26e+09	1.75e+03	7.46e+08	1.47e+03	4.88e+08	6.30e+02	**1.53e+07**	**5.40e+02**	8.30e+07	7.22e+02

Thompson. On the average number of maxima in a set of vectors and applications. *J. ACM*, 25(4):536–543, Oct. 1978.

[4] M. Buzdalov and A. Shalyto. A provably asymptotically fast version of the generalized Jensen algorithm for non-dominated sorting. In T. Bartz-Beielstein, J. Branke, B. Filipic, and J. Smith, editors, *Parallel Problem Solving from Nature - PPSN XIII*, volume 8672 of *Lecture Notes in Computer Science*, pages 528–537. Springer International Publishing, 2014.

[5] C. A. Coello Coello and M. Lechuga. MOPSO: a proposal for multiple objective particle swarm optimization. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 2, pages 1051–1056, 2002.

[6] K. Deb and H. Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints. *Evolutionary Computation, IEEE Transactions on*, 18(4):577–601, Aug 2014.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, Apr 2002.

[8] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable multi-objective optimization test problems. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 1, pages 825–830, May 2002.

[9] K. Deb and S. Tiwari. Omni-optimizer: A procedure for single and multi-objective optimization. In *Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization*, EMO'05, pages 47–61, Berlin, Heidelberg, 2005. Springer-Verlag.

[10] H. Fang, Q. Wang, Y.-C. Tu, and M. F. Horstemeyer. An efficient non-dominated sorting method for evolutionary algorithms. *Evol. Comput.*, 16(3):355–384, Sept. 2008.

[11] F.-A. Fortin, S. Grenier, and M. Parizeau. Generalizing the improved run-time complexity algorithm for non-dominated sorting. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 615–622, New York, NY, USA, 2013. ACM.

[12] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 229–240. VLDB Endowment, 2005.

[13] S. Gupta and G. Tan. A scalable parallel implementation of evolutionary algorithms for multi-objective optimization on gpus. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 1567–1574, May 2015.

[14] J. Horn, N. Nafpliotis, and D. Goldberg. A niched Pareto genetic algorithm for multiobjective optimization. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 82–87 vol.1, Jun 1994.

[15] S. Huband, P. Hingston, L. Barone, and L. While. A review of multiobjective test problems and a scalable test problem toolkit. *Evolutionary Computation, IEEE Transactions on*, 10(5):477–506, Oct 2006.

[16] M. Jensen. Reducing the run-time complexity of multiobjective EAs: The NSGA-II and other algorithms. *Evolutionary Computation, IEEE Transactions on*, 7(5):503–515, Oct 2003.

[17] J. Knowles and D. Corne. The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimisation. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 1, page 105 Vol. 1, 1999.

[18] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, Oct. 1975.

[19] K. McClymont and E. Keedwell. Deductive sort and climbing sort: New methods for non-dominated sorting. *Evol. Comput.*, 20(1):1–26, Mar. 2012.

[20] L. Monier. Combinatorial solutions of multidimensional divide-and-conquer recurrences. *J. Algorithms*, 1(1):60–74, 1980.

[21] T. Murata and H. Ishibuchi. MOGA: multi-objective genetic algorithms. In *Evolutionary Computation, 1995., IEEE International Conference on*, volume 1, pages 289–294, Nov 1995.

[22] P. Roy, M. Islam, K. Murase, and X. Yao. Evolutionary path control strategy for solving many-objective optimization problem. *Cybernetics, IEEE Transactions on*, 45(4):702–715, April 2015.

[23] N. Srinivas and K. Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evol. Comput.*, 2(3):221–248, Sept. 1994.

[24] S. Tang, Z. Cai, and J. Zheng. A fast method of constructing the non-dominated set: Arena's principle. In *Proceedings of the 2008 Fourth International Conference on Natural Computation - Volume 01*, ICNC '08, pages 391–395, Washington, DC, USA, 2008. IEEE Computer Society.

[25] H. Wang and X. Yao. Corner sort for pareto-based many-objective optimization. *Cybernetics, IEEE Transactions on*, 44(1):92–102, Jan 2014.

[26] M. A. Yukish. *Algorithms to Identify Pareto Points in Multi-dimensional Data Sets*. PhD thesis, Pennsylvania State University, 2004. AAI3148694.

[27] X. Zhang, Y. Tian, R. Cheng, and Y. Jin. An efficient approach to nondominated sorting for evolutionary multiobjective optimization. *Evolutionary Computation, IEEE Transactions on*, 19(2):201–213, April 2015.

[28] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In K. Giannakoglou et al., editors, *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001)*, pages 95–100. International Center for Numerical Methods in Engineering (CIMNE), 2002.