Hybridizing Non-dominated Sorting Algorithms: Divide-and-Conquer Meets Best Order Sort

Margarita Markina
ITMO University
49 Kronverkskiy ave.
Saint-Petersburg, Russia 197101
margaritam2706@gmail.com

Maxim Buzdalov
ITMO University
49 Kronverkskiy ave.
Saint-Petersburg, Russia 197101
mbuzdalov@gmail.com

ABSTRACT

Many production-grade algorithms benefit from combining an asymptotically efficient algorithm for solving big problem instances, by splitting them into smaller ones, and an asymptotically inefficient algorithm with a very small implementation constant for solving small subproblems. A well-known example is stable sorting, where mergesort is often combined with insertion sort to achieve a constant but noticeable speed-up.

We apply this idea to non-dominated sorting. Namely, we combine the divide-and-conquer algorithm, which has the currently best known asymptotic runtime of $O(N(\log N)^{M-1})$, with the Best Order Sort algorithm, which has the runtime of $O(N^2M)$ but demonstrates the best practical performance out of quadratic algorithms.

Empirical evaluation shows that the hybrid's running time is typically not worse than of both original algorithms, while for large numbers of points it outperforms them by at least 20%. For smaller numbers of objectives, the speedup can be as large as four times.

CCS CONCEPTS

•Theory of computation → Sorting and searching;

KEYWORDS

Non-dominated sorting, divide-and-conquer, best order sort, hybrid algorithms.

ACM Reference format:
Margarita Markina and Maxim Buzdalov. 2017. Hybridizing Non-dominated Sorting Algorithms: Divide-and-Conquer Meets Best Order Sort. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 2 pages.
DOI: http://dx.doi.org/10.1145/3067695.3076074

1 INTRODUCTION

Many Pareto-based evolutionary multiobjective algorithms belong to one of big groups according to how solutions are selected or ranked: those which maintain non-dominated solutions [2], perform non-dominated sorting [3], use domination count [5], or domination strength [12].

Non-dominated sorting assigns ranks to solutions in the following way: the non-dominated solutions get rank 0, and the solutions which are dominated only by solutions of rank at most i get rank $i + 1$. In [3], this procedure runs in $O(N^2M)$, where N is the population size and M is the number of objectives.

As the quadratic complexity is quite large, both from theoretical and practical points of view, many researchers concentrated on improving practical running times [4, 7, 10, 11], however, without improving the worst-case $O(N^2M)$ complexity. Jensen was the first to adapt the earlier result of Kung et at. [9], who solved the problem of finding non-dominated solutions in $O(N(\log N)^{\max(1, M-2)})$, to non-dominated sorting. This algorithm has the worst-case complexity of $O(N(\log N)^{M-1})$. This algorithm could not handle coinciding objective values, which was corrected in subsequent works [1, 6].

A family of $O(N^2M)$ algorithms for non-dominated sorting resembles a family of quadratic algorithms for comparison based sorting, and the $O(N(\log N)^{M-1})$ non-dominated sorting algorithms seem to take up the niche of $O(N \log N)$ sorting algorithms (such as mergesort, heapsort, and randomized versions of quicksort). In this domain, quadratic algorithms are often much simpler and demonstrate better performance on small data. The efficient divide-and-conquer algorithms are able to incorporate quadratic algorithms to solve small subproblems, which improves the overall speed.

This inspired us to apply the similar idea to non-dominated sorting. For the "outer" divide-and-conquer algorithm, we use the only available algorithm family of this sort [1, 6, 8]. For the quadratic algorithm to solve smaller subproblems, we adapt the Best Order Sort [10], as it was shown to typically outperform other quadratic algorithms. Our result is a hybrid algorithm which uses primarily the divide-and-conquer strategy and decides when to switch to Best Order Sort using a formula which depends on the number of points in the subproblem and the number of remaining objectives to consider.

2 HYBRIDIZING THE ALGORITHMS

Our hybridization scheme is similar to that of production-grade sorting algorithms tuned for performance. As the top-level algorithm, we use the divide-and-conquer algorithm. For each subproblem it decides, using certain heuristic, whether to continue using the divide-and-conquer strategy or to run Best Order Sort for this subproblem. In turn, Best Order Sort runs uninterrupted until it solves the assigned subproblem.

Two problems need to be solved for this scheme to work. First, the original Best Order Sort algorithm cannot be straightforwardly applied to solve subproblems, because subproblems may feature non-zero lower bounds on ranks of some points, which appear from comparisons of these points with other points, which are out of the scope of the current subproblem. In addition, there are actually two

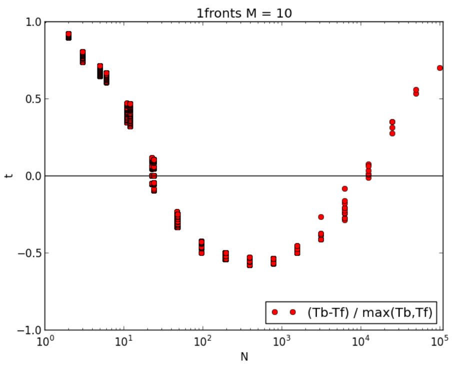

Figure 1: Example result of preliminary experiments on a dataset with 10 objectives and one non-domination level. The dataset has $N = 10^5$ points, all other points correspond to divide-and-conquer subproblems for this dataset. T_f is the running time of the divide-and-conquer algorithm, and T_b is the running time of Best Order Sort. The value of $(T_b - T_f)/\max(T_f, T_b)$ is plotted.

kinds of subproblems: (i) to perform non-dominated sorting of the given set of points, taking into accounts first m objectives and the existing lower bounds on ranks, and (ii) given two sets of points, where the first set A has completely evaluated ranks of points, and the second set B is dominated by the first set in objectives $[m+1; M]$, perform all necessary comparisons between points from A on the left, and points from B on the right. Fortunately, Best Order Sort can be easily adapted to solve the modified problem.

Second, the particular kind of heuristic to determine when to run Best Order Sort is unclear. The main problem with it is that it should have a low computation complexity: at most $O(N)$, because otherwise evaluation of this heuristic worsens the complexity of the divide-and-conquer algorithm. This means we cannot perform any complicated analysis, such as, for instance, principal component analysis, to predict which algorithm is best.

To understand the possible kind of the heuristic algorithm to use for deciding whether to use Best Order Sort for a certain subproblem, we conducted a series of preliminary experiments. In these experiments, we considered a series of datasets, where every dataset had $N = 10^5$ points with $M \in [3; 20]$ objectives and was generated either by uniformly random objective sampling (from the $[0; 1]^M$ hypercube) or by sampling from a hyperplane (which yields a dataset with exactly one non-domination level). Then we ran the divide-and-conquer algorithm on each of these datasets and recorded all subproblems created during the run. After that, we measured the running times of both the divide-and-conquer algorithm and Best Order Sort on all these subproblems.

Fig. 1 shows an example of such experiment. The point above the abscissa axis means that for the corresponding subproblem the divide-and-conquer algorithm took less time than Best Order Sort, while a point below zero means the opposite. One can see in Fig. 1 that Best Order Sort behaves best, compared to the divide-and-conquer algorithm, for not too small and not too large N.

As the similar effect has been noticed for all other datasets as well, we attempted to deduce formulas for the left and right bounds of the higher efficiency range of Best Order Sort. The following empirically constructed formulas were found to fit our data rather well: $n_{\min} = m \ln(m + 1)$ and $n_{\max} = 150m((\ln(d + 1))^{0.9} - 1.5)$, where m is the current number of first objectives to consider, $n_{\min}$ is the left bound of the range, and $n_{\max}$ is the right bound. As a result, the hybrid algorithm switches to Best Order Sort whenever the number of points n falls between $n_{\min}$ and $n_{\max}$.

3 EXPERIMENTS: SHORT OVERVIEW

In the main body of experiments, our hybrid algorithm performs generally at least as well as its parts, except for certain ranges around the switchpoint between the algorithms at higher dimensions. This is an indicator that our heuristic on when to switch is not perfect yet and has a room for improvement. Nevertheless, for the wide range of testing data (3 to 30 objectives, 1 to 20 non-domination levels) our algorithm performs at least 20% better than the best of its parts for large numbers of points (such as $N = 10^5$), and the speedup can be up to 4x for smaller M. In a sense, this means that our hybridization scheme is rather robust.

This research was financially supported by the Government of Russian Federation, Grant 074-U01. The extended version of this paper, including more detailed introduction and experimental results, is available at arXiv[1].

REFERENCES

[1] Maxim Buzdalov and Anatoly Shalyto. 2014. A Provably Asymptotically Fast Version of the Generalized Jensen Algorithm for Non-Dominated Sorting. In *Parallel Problem Solving from Nature – PPSN XIII*. Number 8672 in Lecture Notes in Computer Science. Springer, 528–537.
[2] David W. Corne, Nick R. Jerram, Joshua D. Knowles, and Martin J. Oates. 2001. PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization. In *Proceedings of Genetic and Evolutionary Computation Conference*. Morgan Kaufmann Publishers, 283–290.
[3] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. 2002. A Fast and Elitist Multi-Objective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197.
[4] Hongbing Fang, Qian Wang, Yi-Cheng Tu, and Mark F. Horstemeyer. 2008. An Efficient Non-dominated Sorting Method for Evolutionary Algorithms. *Evolutionary Computation* 16, 3 (2008), 355–384.
[5] C. M. Fonseca and P. J. Fleming. 1996. Nonlinear System Identification with Multiobjective Genetic Algorithm. In *Proceedings of the World Congress of the International Federation of Automatic Control*. 187–192.
[6] Félix-Antoine Fortin, Simon Grenier, and Marc Parizeau. 2013. Generalizing the Improved Run-time Complexity Algorithm for Non-dominated Sorting. In *Proceedings of Genetic and Evolutionary Computation Conference*. ACM, 615–622.
[7] Patrik Gustavsson and Anna Syberfeldt. 2017. A New Algorithm Using the Non-dominated Tree to improve Non-dominated Sorting. *Evolutionary Computation* (Jan. 2017). Just Accepted publication.
[8] M. T. Jensen. 2003. Reducing the Run-time Complexity of Multiobjective EAs: The NSGA-II and Other Algorithms. *IEEE Transactions on Evolutionary Computation* 7, 5 (2003), 503–515.
[9] H. T. Kung, Fabrizio Luccio, and Franco P. Preparata. 1975. On Finding the Maxima of a Set of Vectors. *Journal of ACM* 22, 4 (1975), 469–476.
[10] Proteek Chandan Roy, Md. Monirul Islam, and Kalyanmoy Deb. 2016. Best Order Sort: A New Algorithm to Non-dominated Sorting for Evolutionary Multi-objective Optimization. In *Proceedings of Genetic and Evolutionary Computation Conference Companion*. 1113–1120.
[11] Xingyi Zhang, Ye Tian, Ran Cheng, and Yaochu Jin. 2016. A Decision Variable Clustering-Based Evolutionary Algorithm for Large-scale Many-objective Optimization. *IEEE Transactions on Evolutionary Computation* (2016). DOI: http://dx.doi.org/10.1109/TEVC.2016.2600642
[12] E. Zitzler, M. Laumanns, and L. Thiele. 2001. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In *Proceedings of the EUROGEN'2001 Conference*. 95–100.

[1]http://arxiv.org/abs/1704.04205