Generalized Offline Orthant Search: One Code for Many Problems in Multiobjective Optimization

Maxim Buzdalov
ITMO University
Saint Petersburg, Russia
mbuzdalov@gmail.com

ABSTRACT

We introduce generalized offline orthant search, an algorithmic framework that can be used to solve many problems coming from evolutionary multiobjective optimization using a common well-optimized algorithmic core and relatively cheap reduction procedures. The complexity of the core procedure is $O(n \cdot (\log n)^{k-1})$ for n points of dimension k , and it has a good performance in practice.

We show that the presented approach can perform various flavors of non-dominated sorting, dominance counting, evaluate the ϵ -indicator and perform initial fitness assignment for IBEA. It is either competitive with the state-of-the-art algorithms or completely outperforms them for higher problem sizes. We hope that this approach will simplify future development of efficient algorithms for evolutionary multiobjective optimization.

CCS CONCEPTS

• **Theory of computation** → **Divide and conquer**; *Sorting and searching*; • **Applied computing** → **Multi-criterion optimization and decision-making**;

KEYWORDS

Multiobjective optimization, performance, non-dominated sorting, epsilon-indicator, IBEA, dominance counting.

ACM Reference Format:

Maxim Buzdalov. 2018. Generalized Offline Orthant Search: One Code for Many Problems in Multiobjective Optimization. In *GECCO '18: Genetic and Evolutionary Computation Conference, July 15–19, 2018, Kyoto, Japan*. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3205455.3205469>

1 INTRODUCTION

Multiobjective optimization is a research area dedicated to solving problems with more than one objective to optimize simultaneously. It is tightly connected to the field of multiple criteria decision making. Most often, the objectives to optimize conflict with each other, so there is no single optimal solution. Thus, many setups of multiobjective optimization problems require to find a diverse set of solutions, of which no one is better than another in every objective. One possible way to solve such problems is to use algorithms that work with sets of solutions rather than single solutions. It is very

natural to apply evolutionary algorithms in this setting, so a rich body of multiobjective optimization problems is solved nowadays with evolutionary algorithms and other population-based randomized search heuristics.

The concepts of Pareto-dominance and Pareto-optimality are in the core of many approaches to multiobjective optimization. For minimization problems (which we consider from now on without loss of generality), a point P in the objective space is said to *strictly dominate* a point Q , denoted as $P < Q$, if for every objective i it holds that $P_i \leq Q_i$, and there exists an objective j such that $P_j < Q_j$. *Weak domination*, denoted as $P \leq Q$, removes the latter condition.

A Pareto-optimal subset of a set of points S is its subset $O \subseteq S$ such that no point $o \in O$ is strictly dominated by any point $s \in S$. If one takes a set of objective values for all possible solutions of a problem, then a Pareto-optimal subset of this set, called a *Pareto front* of this problem, corresponds to, informally speaking, a set of “good” solutions for this problem. Finding and enumerating all points from the Pareto front is usually infeasible, so algorithms are often required to find a good approximation of the Pareto front using some predefined number of points, typically equal to the population size or to the archive size.

Wide usage of the Pareto-dominance relation within the domain of evolutionary multiobjective optimization has generated a number of computationally intensive problems, which are used within evolutionary algorithms for multiobjective optimization. These problems are often formulated as computational geometry problems in arbitrary dimensions. The most prominent example is the hypervolume indicator [29]: while it possesses many useful properties, it is computationally expensive in the general case. It was shown to be #P-complete [1], and its solution range in their performance from being exponential in best case to achieving, in recent works, the worst-case complexity of $O(n^{\lfloor (k-1)/2 \rfloor + 1})$ for n points and dimension k [14] or $O(kn^{1+\epsilon}(\log n)^{k-2})$ for an arbitrary $\epsilon > 0$ [21]. This problem still attracts significant attention [11].

Non-dominated sorting is probably the second most popular problem of this sort. The problem was introduced together with the NSGA algorithm [22], and the algorithm was proposed having the time complexity of $O(n^3k)$. The paper which announced NSGA-II [4] also brought an algorithm with the time complexity of $O(n^2k)$, while increasing the memory complexity to $O(n^2)$. Although the worst-case quadratic complexity is much smaller than the exponential complexity of the hypervolume, it still feels too slow when the population size n increases. Already in 2003, this problem was shown to have an $O(n \cdot (\log n)^{k-1})$ divide-and-conquer algorithm, however, certain implementation difficulties have deferred its practical usage [24] until 2013, when it was properly generalized to hold all possible inputs [8] and, after a slight modification, retained the

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
GECCO '18, July 15–19, 2018, Kyoto, Japan
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5618-3/18/07.
<https://doi.org/10.1145/3205455.3205469>

same worst-case complexity [2]. An algorithm for $k = 3$ with the complexity of $O(n \cdot (\log \log n)^2)$ is independently proposed in [19].

The divide-and-conquer nature of the algorithm developed in [2, 8, 12] made it impractical to use with relatively small n and large k . A recent work [16] suggested resorting to an $O(n^2k)$ algorithm for smaller values of n within the main divide-and-conquer algorithm, which made the entire algorithm faster and brought it to competitiveness even at smaller n . Meanwhile, an impressive number of worst-case quadratic algorithms has been developed in order to reduce the running time of non-dominated sorting: [5, 17, 18, 24, 26] and many more. Of them, a separate attention deserves Best Order Sort [20] as well as the very recent k-d tree algorithm [9]. These algorithms, while still being prone to degeneration to the $\Theta(n^2k)$ running time, are typically very fast in practice. Some work was also done on incremental non-dominated sorting [15, 25], which is useful in steady-state algorithms, as well as in parallel environments.

In [12] it was mentioned that some other problems, which are common in evolutionary multiobjective optimization, can be solved, within asymptotically better time than the naive approaches, in a similar way to non-dominated sorting. Some of these ways have been described already in 1975 [13]. These included dominance counting, employed by SPEA2 [28], and finding non-dominated solutions, which is used by a wide variety of algorithms for archive maintenance. Recently, binary ε -indicator computation [30] was shown to be accelerated in a similar but more tricky way [23].

This paper exploits this fact for the profit of the community of evolutionary multiobjective optimization. In Section 2, we develop a formalism to support solving various problems in a unified way, which we call *generalized offline orthant search*. In Section 3, we present reductions of several problems to this formalism, some of which are known and some are new. In Section 4, we give an algorithm to solve the generalized offline orthant search problem, which has the time complexity of $O(n \cdot (\log n)^{k-1})$ and the memory complexity of $O(nk)$. Experimental investigations of the performance of solving problems with the presented algorithm, compared to the existing algorithms, are presented and discussed in Section 5. Finally, Section 6 concludes.

2 DEFINITION OF GENERALIZED OFFLINE ORTHANT SEARCH

In this section, we abstract ourselves from evolutionary algorithms and multiobjective problems, and introduce the problem of *generalized offline orthant search* in terms of points in certain multidimensional space. For convenience, we assume that points come from $\mathbb{R}^k$, however, both the problem formulation and the algorithm to solve this problem can be extended with no changes to arbitrary domains having a total ordering on their elements.

Assume we have a collection P of points from $\mathbb{R}^k$, and the size of this collection is $|P| = n$. It is not necessary that this collection is a set, since it may contain equal points. However, we still need to distinguish these points. We do it by assigning each point its *index*, which runs from 1 to n , so every two different points have distinct indices. We denote the i -th element of the collection P by $P[i]$, and the i -th coordinate of a point p by p_i .

Some coordinates may be specified as *strict* when determining whether a point p dominates a point q . More precisely, a set $S \subseteq \{1, 2, \dots, k\}$ is given which contains the indices of strict coordinates, and the S -dominance relation $<_S$ is defined as:

$$p <_S q \leftrightarrow p \neq q \text{ and } \begin{cases} p_i \leq q_i & \text{for all } i \notin S; \\ p_i < q_i & \text{for all } i \in S. \end{cases}$$

Some of these points are *data points*. A set $D \subseteq \{1, 2, \dots, n\}$ contains the indices of data points. With each data point, we associate a value using a mapping $V : D \rightarrow \mathcal{K}$, where $\mathcal{K}$ is a commutative monoid with a commutative aggregation operation $\oplus : \mathcal{K} \times \mathcal{K} \rightarrow \mathcal{K}$ and a neutral element $\square$ such that $\forall v \in \mathcal{K}$ it holds that $\square \oplus v = v$.

Some of these points are *query points*. A set $Q \subseteq \{1, 2, \dots, n\}$ contains the indices of query points. The problem is to compute, for each query point with index q , an aggregation of values associated with all data points which S -dominate $P[q]$, that is, the following expression:

$$A(q) = \bigoplus_{\substack{d \in D \\ P[d] <_S P[q]}} D(d). \quad (1)$$

However, it is not required that $D \cap Q = \emptyset$. What is more, for every query point q there is an associated operation, *query-to-data* or $Q2D$, which can arbitrarily alter some data values (e.g. what the mapping V returns) for itself or for points d such that $q <_S d$. In particular, if a point belongs to $D \cap Q$, it can update its own associated data value. This operation must be called once for every query point right after (1) is completely evaluated for it. This operation is designed to, and is expected to be, fast enough, and we treat its complexity as $O(1)$ in the subsequent analysis.

Due to the existence of the $Q2D$ operation, we cannot evaluate (1) in an arbitrarily order. Thus we require that, for any two query points p and q such that $p <_S q$, point p is completely evaluated strictly before point q considers the data value associated with p .

3 REDUCTION OF VARIOUS PROBLEMS TO ORTHANT SEARCH

In this section, we outline reduction of several problems known from evolutionary multiobjective optimization to generalized offline orthant search (which we will call simply *orthant search* from now on). In this section, we assume that there exists an algorithm for orthant search that works in $O(n \cdot (\log n)^{k-1})$, where n is the number of points and k is the dimension of these points. The algorithm itself will be given later in Section 4.

3.1 Non-dominated Sorting

Non-dominated sorting [4, 22] takes a set P of points and assigns them *ranks* as follows:

- points not dominated by any points get rank 0;
- points dominated by points with rank 0 get rank 1;
- points dominated by points with rank ≤ 1 get rank 2;
- points dominated by points with rank $\leq r$ get rank $r + 1$.

To have this procedure well-defined, we have to use strict dominance here, since otherwise the result of rank assignment would depend on the order in which equal points are considered.

In our reduction of non-dominated sorting to orthant search, every point is simultaneously a query point and a data point. As a

rank of a point q is defined to be one plus the maximum of ranks of points that strictly dominate q , and zero in the case no points strictly dominate q , we can decide that points will have their ranks as their data values, and define the components of orthant search as follows:

- $\mathcal{K} \leftarrow \mathbb{Z}_{\geq -1}$;
- $\square \leftarrow -1$;
- $\oplus \leftarrow \max$;
- $D \leftarrow \{1, 2, \dots, n\}$;
- $Q \leftarrow \{1, 2, \dots, n\}$;
- $S \leftarrow \emptyset$;
- $Q2D(i): V(i) \leftarrow A(i) + 1$;
- initially, $V(i) \leftarrow -1$ for all i .

The final rank of a point with index i can be achieved, after the orthant search ends, either directly as $V(i)$, or as $A(i) + 1$. One can easily see that, with these substitutions, orthant search produces exactly the ranks for non-dominated sorting. This can be shown using induction by ranks. The base of the induction is rank zero: every non-dominated point with index i will have $A(i) = -1$, so its rank $V(i)$ will be set to zero. Assuming that all points with ranks up to $r - 1$ are assigned correctly by the proposed reduction, we now prove the same for rank r . Consider an arbitrary point p whose correct rank is r . By definition of non-dominated sorting, every point with rank r is dominated by points with ranks not exceeding $r - 1$ only, and there is at least one such point with rank exactly $r - 1$. We know that orthant search evaluates points in such an order that all data values of points that dominate p are seen with their final data values from point p . This means that the algorithm will evaluate a maximum of all ranks of points that dominate p , which will be $r - 1$ by the induction assumption. The $Q2D$ operation will then set the data value of point p to r , which proves the correctness of this reduction.

3.2 “Bug-Compatible” Non-dominated Sorting

In the definition of non-dominated sorting we have used strict dominance to determine whether one point dominates another and should influence the rank of the latter. When there are no two equal points in the input, one can also use weak dominance to do the same, which is slightly faster and simpler to implement. However, the results will be quite different in the presence of equal points. In the case of strict dominance equal points get equal ranks, while they get incrementing ranks when using weak domination.

This feature seems more like a bug in the implementation of an algorithm. In particular, which point among the equal ones gets the smallest rank, and thus will subsequently be preferred by selection procedures, heavily depends on the order of points in the input collection, as well as on tiny details of the implementation. For instance, when the input collection is formed by concatenating the previous generation first and the new individuals second, the individuals from the previous generation may have an unfair advantage over new individuals which are phenotypically equal to the former. This makes experimental results harder to reproduce and less reliable.

However, the side effect of this feature is that individuals dominated by few equal individuals are preferred to individuals dominated by many equal individuals, so it can be desirable [24]. One may ensure fairness of rank assignment by shuffling equal individuals before assigning them incrementing ranks, or by shuffling the entire population before sorting if rank assignment is done deterministically.

We show that orthant search can be used to implement this behavior. One needs to retain only unique points in the input,

which can be done in $O(nk)$ expected time using hashing, and to evaluate multiplicity $M(i)$ of each point i . Then the only change to the orthant search setup is the $Q2D$ operation, which becomes $V(i) \leftarrow A(i) + M(i)$. The actual ranks of $M(i)$ equal points will be $A(i) + 1, A(i) + 2, \dots, A(i) + M(i)$.

3.3 Non-dominated Sorting, Strict Coordinates

The paper [24] also mentions, among others, the need to support coordinatewise strict dominance relation in non-dominated sorting, i.e. to treat $p < q$ as $p_i < q_i$ for all coordinates i . With the framework for orthant search, this need can be easily addressed by specifying all coordinates to be strict, that is, by using $S = \{1, 2, \dots, k\}$ instead of $S = \emptyset$. One can also choose which coordinates are strict and which are not by appropriately initializing S .

3.4 Dominance Count and Dominance Rank

Given a collection of k -dimensional points P , *dominance count* of a point p , as defined in [28], is the number of points $q \in P$ such that $p < q$. Another quantity, *dominance rank* of a point p , which is employed in older algorithms such as MOGA [7] and NPGA [10], is the number of points $q \in P$ such that $q < p$. Finding both these values is possible in $O(n \cdot (\log n)^{k-1})$ time as proposed in [13], which was also mentioned in [12]. We give the configuration for orthant search, which is the same for both problems, except that all coordinates need to be negated for dominance count:

- $\mathcal{K} \leftarrow \mathbb{Z}_{\geq 0}$;
- $\square \leftarrow 0$;
- $\oplus \leftarrow +$;
- $D \leftarrow \{1, 2, \dots, n\}$;
- $Q \leftarrow \{1, 2, \dots, n\}$;
- $S \leftarrow \emptyset$;
- $Q2D$: do nothing;
- $V(i) \leftarrow 1$ for all i .

If desired, one can also set one or more coordinates to be strict by appropriately initializing the set S .

3.5 Additive ε -indicator

In multiobjective optimization, an *indicator* is a function from one or more point sets to a real value. Examples include the (unary) hypervolume indicator [29], the (binary) ε -indicator [30] and many more.

For two point sets M (for “moving”) and F (for “fixed”), the *additive binary ε -indicator* is defined as the smallest ε such that for every point $f \in F$ there exists a point $m \in M$ such that $m - (\varepsilon, \varepsilon, \dots, \varepsilon) \leq f$. This definition is given in [30] along with the multiplicative binary ε -indicator, which features $m \cdot (1 - \varepsilon) \leq f$ and can be reduced to the additive one by taking logarithm of every coordinate, as well as unary versions of both indicators, which simply fix F to be some reference front.

There exists a straightforward algorithm to evaluate ε -indicator in $O(|M| \cdot |F| \cdot k)$ for k objectives. One can also apply the algorithm for finding non-dominated points together with binary search for the answer, which results in an $O(\log |\mathcal{D}| \cdot n \cdot (\log n)^{k-2})$ algorithm, where $\mathcal{D}$ is the domain of possible values for ε .

In [23], a different algorithm is proposed with the complexity of $O(k^2 n + kn \cdot (\log n)^{\max(1, k-2)})$, which is more efficient since typically k is smaller than $\log |\mathcal{D}|$. This algorithm reduces the problem of finding the minimum ε to k calls to orthant minimum search. We show how the reduction from [23] works, since we need these ideas in the following subsections.

We evaluate the value ε for every point $f \in F$ separately, then take a maximum of them. For a point $f \in F$, we have:

$$\varepsilon(f) = \min_{m \in M} \max_{i=1}^k (m_i - f_i).$$

We represent the set M as a union of sets M_j such that the maximum of $m_i - f_i$ is attained in coordinate j , then simplify the expression:

$$\varepsilon(f) = \min_{j=1}^k \min_{m \in M_j} \max_{i=1}^k (m_i - f_i) = \min_{j=1}^k \min_{m \in M_j} (m_j - f_j).$$

As in M_j for every coordinate i it holds that $m_j - f_j \geq m_i - f_i$, in turn it holds that $m_i - m_j \leq f_i - f_j$. If we define a projection operator $\mathcal{P}_j(p) = (p_1 - p_j, p_2 - p_j, \dots, p_k - p_j)$, then $m \in M_j$ can be rewritten as $\mathcal{P}_j(m) \leq \mathcal{P}_j(f)$, so:

$$\varepsilon(f) = \min_{j=1}^k \min_{\substack{m \in M \\ \mathcal{P}_j(m) \leq \mathcal{P}_j(f)}} (m_j - f_j) = \min_{j=1}^k \left(\min_{\substack{m \in M \\ \mathcal{P}_j(m) \leq \mathcal{P}_j(f)}} m_j \right) - f_j,$$

where the inner minimum is a result of a query to orthant search.

The j -th reduction to orthant search can be defined as:

- $\mathcal{K} \leftarrow \mathbb{R};$
- $\square \leftarrow +\infty;$
- $\oplus \leftarrow \min;$
- $S \leftarrow \emptyset;$
- $D \leftarrow \{1, \dots, |M|\};$
- $Q \leftarrow \{|M| + 1, |M| + 2, \dots, |M| + |F|\};$
- $Q2D$: do nothing;
- $V(i) \leftarrow M[i]_j$ for all i ;

with the collection of points P defined by concatenation of points from M after applying the operator $\mathcal{P}_j$, removing the identity-zero coordinate $p_j - p_j$ and adding 0 as the k -th coordinate, and of points from F after the same transformation but adding 1 as the k -th coordinate. The values in the k -th coordinate can be arbitrary as long as every point from M has a smaller value than any point from F .

For each reduction, the algorithm for orthant search will get rid of the k -th coordinate in linear time and will do the rest in $O(n \cdot (\log n)^{\max(1, k-2)})$ time. The total time needed for all reductions is bounded by $O(k^2 n)$. It follows that the total running time is $O(k^2 n + kn \cdot (\log n)^{\max(1, k-2)})$.

3.6 Fitness assignment in IBEA

Binary indicators, such as the above-mentioned ε -indicator, are typically used to assess the quality of solutions in multiobjective optimization. The IBEA algorithm [27] makes use of binary indicators in optimization itself. It can use an arbitrary binary indicator $I(A, B)$ that is dominance-preserving, e.g. for any two points $a < b$ it holds that $I(\{a\}, \{b\}) < I(\{b\}, \{a\})$, and for any third point c it holds that $I(\{c\}, \{a\}) \geq I(\{c\}, \{b\})$. Note that the binary ε -indicator is dominance-preserving. Within an arbitrary set of points P it assigns fitness to these points in the following way:

$$F(p) = - \sum_{q \in P \setminus \{p\}} e^{-I(\{q\}, \{p\})/\kappa}$$

where $\kappa > 0$ is a damping parameter. In the description of IBEA itself, the initial fitness assignment is performed in $O(n^2 k)$, where $n = |P|$ and k is the dimension of the objective space.

We show how to evaluate the initial fitness assignment with the use of the ε -indicator in $O(k^2 n + kn \cdot (\log n)^{\max(1, k-2)})$. The idea is similar to the one which was used to evaluate the ε -indicator itself.

We note that $I(\{q\}, \{p\}) = \max_{i=1}^k (q_i - p_i)$, so we split the set $P \setminus \{p\}$ again in j parts, such that the j -th component contains points q which attain the maximum at the j -th coordinate. However, since we now have sum instead of minimum as an aggregation function, we must ensure that these sets are disjoint. We define them as follows:

- $P_1 = \{q \in P \setminus \{p\} \mid q_1 - p_1 = \max_{i=1}^k (q_i - p_i)\};$
- $P_2 = \{q \in P \setminus \{p\} \setminus P_1 \mid q_2 - p_2 = \max_{i=1}^k (q_i - p_i)\};$
- $\dots$
- $P_j = \{q \in P \setminus \{p\} \setminus (\bigcup_{t=1}^{j-1} P_t) \mid q_j - p_j = \max_{i=1}^k (q_i - p_i)\}.$

With the projection operator $\mathcal{P}_j(p) = (p_1 - p_j, p_2 - p_j, \dots, p_k - p_j)$ defined as above, we turn the definitions of sets P_j into the ones based on domination as follows:

- $P_1 = \{q \in P \mid \mathcal{P}_1(q) < \mathcal{P}_1(p)\};$
- $P_2 = \{q \in P \mid \mathcal{P}_2(q) <_{\text{strict in } 1} \mathcal{P}_2(p)\};$
- $\dots$
- $P_j = \{q \in P \mid \mathcal{P}_j(q) <_{\text{strict in } [1:j-1]} \mathcal{P}_j(p)\};$

where we employ dominance relations which are strict in some range of coordinates and non-strict in other coordinates. With this decomposition, fitness assignment now looks as follows:

$$\begin{aligned} F(p) &= - \sum_{j=1}^k \sum_{q \in P_j} e^{-I(\{q\}, \{p\})/\kappa} = - \sum_{j=1}^k \sum_{q \in P_j} e^{(p_j - q_j)/\kappa} \\ &= - \sum_{j=1}^k e^{p_j/\kappa} \left(\sum_{q \in P_j} e^{-q_j/\kappa} \right), \end{aligned} \quad (2)$$

where the inner sum is a result of an orthant sum query.

The j -th reduction to orthant search can be defined as:

- $\mathcal{K} \leftarrow \mathbb{R};$
- $\square \leftarrow 0;$
- $\oplus \leftarrow +;$
- $D \leftarrow \{1, 2, \dots, n\};$
- $Q \leftarrow \{1, 2, \dots, n\};$
- $S \leftarrow \{1, 2, \dots, j-1\};$
- $Q2D$: do nothing;
- $V(i) \leftarrow e^{-P[i]_j/\kappa}$ for all i .

In addition, the first reduction that corresponds to $j = 1$ should also handle the fact that every two distinct points p and q such that $\mathcal{P}_1(p) = \mathcal{P}_1(q)$ must take each other into account when assigning fitness. To do this, it suffices to merge all projected points before handing them out to orthant search, while also adding up the corresponding data values. When collecting the results of queries and subtracting them, according to (2), from the actual fitness values, the difference between the sum of data values associated with the merged point and the actual data value shall also be multiplied by $e^{p_j/\kappa}$ and subtracted from the fitness value. Note that this is needed only for $j = 1$, since for $j > 1$ the dominance relations are strict in the first coordinate, so points with equal projections do not have to interact this way.

Similarly to the case of ε -indicator, the running time of this algorithm is $O(k^2 n + kn \cdot (\log n)^{\max(1, k-2)})$.

We should note here that this reduction corresponds to the initial fitness assignment in IBEA. This algorithm removes worst points one by one and recomputes the assignment once a point is removed. The naive way of doing this would perform $O(nk)$ operations for a single deletion, which makes up $O(n^2 k)$ operations in total. In its present form, the generalized offline orthant search, being *offline*, cannot optimize this second part. However, we believe that certain extensions to its methodology will improve the running time

Algorithm 1 The entry point of orthant search

```

function ORTHANTSEARCH( $P, \mathcal{K}, \square, \oplus, D, Q, S, V, A, Q2D$ )
  - Runs orthant search on a collection of points  $P$ 
  -  $\mathcal{K}$  is the data type for data values and query answers
  -  $\oplus : \mathcal{K} \times \mathcal{K} \rightarrow \mathcal{K}$  is the aggregation function
  -  $\square \in \mathcal{K}$  is the neutral element with regards to  $\oplus$ 
  -  $D \subseteq \{1, \dots, |P|\}$  is a set of indices of data points
  -  $Q \subseteq \{1, \dots, |P|\}$  is a set of indices of query points
  -  $S \subseteq \{1, \dots, |P[1]|\}$  is a set of “strict” coordinates
  -  $V : \{1, \dots, |P|\} \rightarrow \mathcal{K}$  maps point indices to data values
  -  $A : \{1, \dots, |P|\} \rightarrow \mathcal{K}$  maps point indices to query answers
  -  $Q2D(i)$  rewrites parts of  $V$  once query point  $i$  is evaluated
  - All arguments are global for all subsequent functions
  if  $|P| > 0$  then
     $I \leftarrow \text{LEXSORT}(P)$             $\triangleright$  indices in lexicographical order
     $k \leftarrow |P[1]|$               $\triangleright$  get the dimension of points
    if  $k = 1$  then
      SOLVE1D( $I$ )                  $\triangleright$  one-dimensional case
    else
      HELPERA( $I, k$ )                $\triangleright$  general case
    end if
  end if
end function

```

Algorithm 2 The special one-dimensional case of orthant search

```

function SOLVE1D( $I$ )
   $w \leftarrow \square$                   $\triangleright$  aggregation of all seen data values
   $j \leftarrow 1$                   $\triangleright$  first non-aggregated point
  for  $i \in I$  do
    while  $P[I[j]]_1 < P[i]_1$  do  $\triangleright$  get remaining smaller points
      if  $j \in D$  then
         $w \leftarrow w \oplus V(j)$     $\triangleright$  if data point, aggregate it
      end if
       $j \leftarrow j + 1$             $\triangleright$  move on to the next point
    end while
    if  $i \in Q$  then              $\triangleright$  if query point, answer the query
       $A(i) \leftarrow w$             $\triangleright$  write the current aggregation
      Call  $Q2D(i)$                $\triangleright$  update subsequent data points
    end if
  end for
end function

```

complexity of the “decremental” part as well. We also think that there are legitimate usages of the proposed reduction on its own. For instance, it can replace the crowding distance as the diversity measure in NSGA-II.

4 ALGORITHM FOR ORTHANT SEARCH

In this section, we outline the algorithm for orthant search with the time complexity of $O(n \cdot (\log n)^{k-1})$ and the memory complexity of $O(nk)$, where n is the number of points and k is the dimension of these points. Before we do it, we shall note that a trivial algorithm exists for orthant search with the time complexity $O(n^2k)$ and the memory complexity of $O(\log n)$. It can be implemented using

lexicographical pre-sorting of points in $O(n \log n + nk)$ and then comparing all pairs of points at indices $i < j$ for dominance.

The proposed algorithm follows the lines of the divide-and-conquer algorithm for non-dominated sorting proposed in [12] and later refined in [2, 8]. The necessary changes introduced to this algorithm include the change from point ranks to values of an arbitrary commutative monoid $\mathcal{K}$, as well as the changes related to the support of the set S of strict coordinates. We also switch to the trivial $O(n^2k)$ algorithm on small subproblem sizes similar to [16].

The entry point of the proposed algorithm, along with explanation of variables that are “global” to all other function which this algorithm uses, is given in Algorithm 1. It first sorts the given points lexicographically in $O(n \log n + nk)$. Then, depending on k , it calls either a function for the general case, called HELPERA, or a special function to handle the one-dimensional case SOLVE1D which works in $O(n)$. The pseudocode of the one-dimensional case, which is rather simple, is given in Algorithm 2.

The general case is solved by a recursive procedure HELPERA, which is outlined in Algorithm 3. It takes two arguments: I , the collection of point indices in lexicographical order, and k , the number of dimensions to consider. This function has the following properties:

- (1) It assumes that $k \geq 2$ and in dimensions higher than k nothing contradicts that for any two different point indices $i_1, i_2 \in I$, such that i_1 comes before i_2 , it can be that $P[i_1] <_S P[i_2]$.
- (2) It assumes that for any query point $q \in I$ the value $A(d)$ is equal to

$$\bigoplus_{\substack{d \in D, d \notin I \\ P[d] <_S P[q]}} D(d),$$

that is, all necessary data values from points outside I are already aggregated.

- (3) During its run, this function will aggregate all necessary data values of points from $I \cap D$ to the query answers of points from $I \cap Q$, and will call $Q2D$ on points from $I \cap Q$.
- (4) As a result, all query points from I will contain the final answers to their queries, and all data points from I will have data values which will not be subsequently changed.

It works by considering a few cases. First, if $|I| \leq 2$ or if $|I|$ is below a certain threshold depending on k , the comparisons are performed explicitly by a $O(n^2k)$ naive algorithm called NAIVEA. Second, if $k = 2$, this function delegates its work to a special two-dimensional case called SWEEPA. The third case happens if all k -th coordinates are the same in all points from I . Then, if k is a strict coordinate, $Q2D$ is called on all query points from I , since no two points from I can dominate each other. Otherwise, HELPERA($I, k-1$) is called.

If nothing from the above holds, the procedure performs the divide-and-conquer step. It computes the median of all k -th coordinates of all points from I and splits I into three sets: L consists of points with the k -th coordinate smaller than the median, M equal to the median, H greater than the median. Since no points from $M \cup H$ can dominate any point from L , the preconditions of HELPERA are fulfilled for L and HELPERA(L, k) is called. Then, as the data values in L are all computed, we can use them to refine

Algorithm 3 The general case procedure: HELPERA

```

function HELPERA( $I, k$ )
  if  $|I| \leq 1$  then
    Call  $Q2D(i)$  for  $i \in I$ 
  else if  $|I| = 2$  or  $|I| \leq$  a threshold for  $k$  then
    NAIVEA( $I, k$ )
  else if  $k = 2$  then
    SWEEPA( $I$ )
  else if  $\min X_k = \max X_k$  for  $X_k = \{P[i]_k \mid i \in I\}$  then
    if  $k \in S$  then
      Call  $Q2D(i)$  for  $i \in I$ 
    else
      HELPERA( $I, k - 1$ )
    end if
  else
     $L, M, H \leftarrow I$  split by a median of  $\{P[i]_k \mid i \in I\}$ 
    HELPERA( $L, k$ )
    HELPERB( $L \cap D, M \cap Q, k - 1$ )
    if  $k \in S$  then
      Call  $Q2D(i)$  for  $i \in M$ 
    else
      HELPERA( $M, k - 1$ )
    end if
    HELPERB( $(L \cup M) \cap D, H \cap Q, k - 1$ )
    HELPERA( $H, k$ )
  end if
end function

```

query results in M . To do that, we call a special helper procedure $\text{HELPERB}(L \cap D, M \cap Q, k - 1)$, which will be described later. It will update query results in M by the data values from L whenever needed. After this, since no point from H can dominate M , the only data points that can influence query points in M are again points from M . This means we can call $\text{HELPERA}(M, k)$. However, since we know that all k -th coordinates are the same in M , we can inline this call using the fourth case from above. We proceed doing the same for the last set H . Namely, first we refine the query answers for points in H by already complete points from $L \cup M$ using $\text{HELPERB}((L \cup M) \cap D, H \cap Q, k - 1)$, and then we finish the work by $\text{HELPERA}(H, k)$. At this moment, all query points from I have their final answers, and all data points from I have their final data values, so we have fulfilled the requirements for $\text{HELPERA}(I, k)$ and can exit this procedure.

The recursive procedure HELPERB is outlined in Algorithm 4. It takes three arguments: the collections of lexicographically sorted point indices L and H , and the number of dimensions to consider k . Its properties are:

- (1) It assumes that L contains only data points and H contains only query points.
- (2) It assumes that $k \geq 2$ and in dimensions higher than k nothing contradicts that for any point indices $i_L \in L$ and $i_H \in H$ it can be that $P[i_L] <_S P[i_H]$.
- (3) It assumes that the query no point from H has ever aggregated a data value from any point from L .

Algorithm 4 The cross-assignment procedure: HELPERB

```

function HELPERB( $L, H, k$ )
  if  $|L| \leq 1$  or  $|H| \leq 1$  or  $|L| + |H| \leq$  a threshold for  $k$  then
    NAIVEB( $L, H, k$ )
  else if  $k = 2$  then
    SWEEPB( $L, H$ )
  else
     $L_k = \{P[i]_k \mid i \in L\}, L_{\min} = \min L_k, L_{\max} = \max L_k$ 
     $H_k = \{P[i]_k \mid i \in H\}, H_{\min} = \min H_k, H_{\max} = \max H_k$ 
    if  $L_{\min} > H_{\max}$  or ( $k \in S$  and  $L_{\min} = H_{\max}$ ) then
      return  $\triangleright$  no  $d \in L$  can dominate  $q \in H$ 
    else if  $L_{\max} < H_{\min}$  or ( $k \notin S$  and  $L_{\max} = H_{\min}$ ) then
      HELPERB( $L, H, k - 1$ )
    else
       $m \leftarrow \text{MEDIAN}(L_k \cup H_k)$ 
       $L_L, L_M, L_H \leftarrow$  split  $L$  by  $m$ 
       $H_L, H_M, H_H \leftarrow$  split  $H$  by  $m$ 
      HELPERB( $L_L, H_L, k$ )
      HELPERB( $L_H, H_H, k$ )
      HELPERB( $L_L, H_M, k - 1$ )
      HELPERB( $L_L, H_H, k - 1$ )
      HELPERB( $L_M, H_H, k - 1$ )
      if  $k \notin S$  then
        HELPERB( $L_M, H_M, k - 1$ )
      end if
    end if
  end if
end function

```

- (4) At the end of the run, for any point $q \in H$ the value $A(q)$ will be aggregated with exactly

$$\bigoplus_{\substack{d \in L \\ P[d] <_S P[q]}} D(d).$$

The principle of functioning of HELPERB is similar to that of HELPERA . The only significant change is that the median, which is used while splitting the sets, is taken among all k -th coordinates of $L \cup H$, and there are more corner cases based on how minimum and maximum k -th coordinates relate to each other in L and H . One can show the correctness of this procedure in a way similar to how it was done for HELPERA .

The only remaining procedures are NAIVEA , NAIVEB , SWEEPA and SWEEPB . The former two procedures are straightforward, and their pseudocode is given in Algorithm 5. The sweep procedures use any prefix aggregation data structure, such as the Fenwick tree [6] or a suitably adapted balanced search tree, to perform the same task in two dimensions. Each add or query operation with such data structure is done in $O(\log \text{size}) = O(\log n)$ time. The nickname “sweep” comes from the algorithmic design pattern called “sweep line”, which forms the basis of both procedures. Their pseudocode is given in Algorithm 6.

The running time complexity of SWEEPA is $O(|I| \log |I|)$, and the complexity of SWEEPB is $O((|L| + |H|) \log |L|)$. Using the so-called “Master theorem” from [3] and induction by k , one can show,

Algorithm 5 The “naive” backends of HELPERA and HELPERB

```

function NAIVEA( $I, k$ )
  for  $q \in I \cap Q$  do
    for  $d \in I \cap D$  do
      if  $P[d] \prec_{S[1:k]} P[q]$  then
         $A(q) \leftarrow A(q) \oplus V(d)$ 
      end if
    end for
    Call  $Q2D(q)$ 
  end for
end function

function NAIVEB( $L, H, k$ )
  for  $q \in H$  do
    for  $d \in L$  do
      if  $P[d] \prec_{S[1:k]} P[q]$  then
         $A(q) \leftarrow A(q) \oplus V(d)$ 
      end if
    end for
  end for
end function

```

similarly to [2], that the running time of both HELPERA(I, k) and HELPERB(L, H, k) is $O(n \cdot (\log n)^{k-1})$ where $n = |I|$ or $n = |L| + |H|$.

5 EXPERIMENTS

The algorithms for orthant search (both the naive one, which works in $O(n^2k)$, and the proposed one) were implemented in the Java programming language to work with points from $\mathbb{R}^k$ and arbitrary configurations for $\mathcal{K}$ and $Q2D$. The latter were abstracted out in a separate class which knows the precise type of collections of values from $\mathcal{K}$ as well as how to perform the $\oplus$ operation and to fill parts of collections with the $\square$ value. This was done to overcome the impossibility of specialization of generics with primitive data types, such as `int` or `double`, in Java 8 or earlier, and to eliminate the performance loss when using boxed types. The threshold values, which are used to decide when to switch to NAIVEA or NAIVEB, were set to 50 for $k = 3$ and 100 for $k \geq 4$ based on experiments. We have also tested the version that does not use thresholds.

We have also implemented the concrete orthant search configurations and the necessary reduction procedures, along with naive implementations and a unit test suite, for all reductions listed in Section 3. For every reduction, a performance test suite has also been implemented. To cover evenly a variety of problem sizes, we used problem sizes $n \in \{10, 31, 100, 316, 1000, 3162, 10000\}$ to mimic $\sqrt{10^t}$ for integer $t \in [2; 8]$, and for dimensions we used $k \in \{2, 3, 4, 5, 7, 10, 15, 20\}$. We used three patterns for point generation:

- the hypercube, also known as the “cloud” dataset, where each coordinate is sampled independently and uniformly from $[0; 1]$;
- the discrete hypercube, where each coordinate is sampled independently and uniformly from $\{1, 2, \dots, 10\}$;
- the hyperplane, where coordinates are sampled from $[0; 1]$ such that $\sum_{i=1}^k x_i = 1$ and the points are distributed uniformly in the corresponding hyper-triangle.

Algorithm 6 The “sweep” backends of HELPERA and HELPERB

```

function SWEEPA( $I$ )
   $F \leftarrow$  an empty prefix aggregation data structure
   $d \leftarrow 1$   $\triangleright$  the first point which is not added
  for  $q \in I \cap Q$  do
    while  $P[I[d]] \prec_{S[1:2]} P[q]$  do
      if  $I[d] \in D$  then
         $F \leftarrow F \cup \langle \text{key} = P[I[d]]_2, \text{value} = V(I[d]) \rangle$ 
      end if
       $d \leftarrow d + 1$ 
    end while
    if  $2 \in S$  then
       $A(q) \leftarrow A(q) \oplus F(\text{key} < P[q]_2)$   $\triangleright$  strict prefix query
    else
       $A(q) \leftarrow A(q) \oplus F(\text{key} \leq P[q]_2)$   $\triangleright$  non-strict query
    end if
    Call  $Q2D(q)$ 
  end for
end function

function SWEEPB( $L, H$ )
   $F \leftarrow$  an empty prefix aggregation data structure
   $d \leftarrow 1$   $\triangleright$  the first point which is not added
  for  $q \in H$  do
    while  $P[L[d]] \prec_{S[1:2]} P[q]$  do
       $F \leftarrow F \cup \langle \text{key} = P[L[d]]_2, \text{value} = V(L[d]) \rangle$ 
    end while
     $d \leftarrow d + 1$ 
    if  $2 \in S$  then
       $A(q) \leftarrow A(q) \oplus F(\text{key} < P[q]_2)$   $\triangleright$  strict prefix query
    else
       $A(q) \leftarrow A(q) \oplus F(\text{key} \leq P[q]_2)$   $\triangleright$  non-strict query
    end if
  end for
end function

```

For each combination of n and k we have generated 10 independent instances of hypercube (of which five are real-valued and five are discrete) and of hyperplane, which we subsequently used to test the performance of algorithms for all problems from Section 3. To achieve reliable measurements, we used the Java Microbenchmark Harness suite¹, which is the current state-of-the-art for benchmarking on the Java platform. The time measurements were done for a single run including 10 instances in order to reduce the total amount of measurements and make conclusions about the average running times. The source code is available on GitHub².

For non-dominated sorting, we also used for comparison the three state-of-the-art algorithms, namely, Best Order Sort [20], the k -d tree-based algorithm called ENS-NDT [9], and the hybrid algorithm combining the divide-and-conquer idea [2] and the ENS algorithm [26] similar to how this was done in this paper. The implementations were borrowed from the GitHub repository dedicated to non-dominated sorting³.

¹<http://openjdk.java.net/projects/code-tools/jmh>

²<https://github.com/mbuzdalov/orthant-search>

³<https://github.com/mbuzdalov/non-dominated-sorting>

Due to the space restrictions we cannot present the results of the experiments, in the form of plots and tables, in the paper itself. These were moved to the Supplementary Materials, which are available along with the paper. We provide only basic insights.

We have never observed the case when naive orthant search outperformed the divide-and-conquer algorithm. The latter algorithm is also always not slower when the non-trivial thresholds are used; the speedups are typically 3x to 4x at $n = 10^4$ and $k = 10$.

Comparing the divide-and-conquer orthant search using thresholds with the naive implementations shows that, on problems directly reducible to orthant search, the former algorithm overcomes the naive implementations at $n \geq 30$ for $k = 2$, at $n \geq 100$ for $3 \leq k \leq 7$, and at $n \geq 300$ for $10 \leq k \leq 20$.

The results on the problems regarding the ε -indicator are less inspiring for $k \geq 4$, for instance, orthant search becomes faster than the naive implementation of the ε -indicator for $k = 10$ only at $n \approx 10^4$, however, the slopes always show better asymptotic behaviour of the orthant search. IBEA fitness assignment favours the orthant search slightly more, since it does fewer expensive exponent evaluations.

Finally, the comparison with the state-of-the-art non-dominated sorting algorithms shows that orthant search loses only a constant factor of 1.2 to 2.0 to the specialized hybrid divide-and-conquer algorithm, which decreases as k increases. Orthant search typically outperforms Best Order Sort at $n = 10^4$, except for $4 \leq k \leq 7$ on hypercube datasets. The ENS-NDT algorithm is typically better than orthant search at $n = 10^4$ for $k \geq 4$ by a factor of at most 2.2, and this difference gets smaller as k grows above 10.

6 CONCLUSION

We have shown that a variety of problems, which arise in design of algorithms for evolutionary multiobjective optimizations, can be solved more efficiently with the use of the same divide-and-conquer paradigm. Once efficiently implemented, it can serve in many algorithms and provide asymptotic speed-ups of algorithmically tightest components with very little work needed for adaptation.

We want to note that the proposed algorithm for orthant search can be implemented for parallel environments with the use of fork-join paradigm. The median-based strategy for divide-and-conquer in HELPERB ensures that two “heavy” subproblems which have the non-decreased value of k share no data, so they can be solved in parallel with no need for data synchronization. It also ensures that no extra memory allocation is needed to achieve parallelization. Preliminary experiments suggest good scalability on large enough problem sizes.

This research was supported by the Russian Scientific Foundation, agreement No. 17-71-20178.

REFERENCES

- [1] Karl Bringmann and Tobias Friedrich. 2009. Approximating the Least Hypervolume Contributor: NP-Hard in General, But Fast in Practice. In *Proceedings of the 5th International Conference on Evolutionary Multi-Criterion Optimization*. 6–20.
- [2] 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.
- [3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2001. *Introduction to Algorithms, 2nd Ed.* MIT Press, Cambridge, Massachusetts.
- [4] 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.
- [5] 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.
- [6] Peter M. Fenwick. 1994. A new data structure for cumulative frequency tables. *Software: Practice and Experience* 24, 3 (1994), 327–336.
- [7] Carlos M. Fonseca and Peter 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.
- [8] 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.
- [9] Patrik Gustavsson and Anna Syberfeldt. 2018. A New Algorithm Using the Non-dominated Tree to Improve Non-dominated Sorting. *Evolutionary Computation* 26, 1 (2018), 89–116.
- [10] J. Horn, N. Nafpliotis, and D.E. Goldberg. 1994. A Niche Pareto Genetic Algorithm for Multiobjective Optimization. In *Proceedings of IEEE Conference on Evolutionary Computation*, Vol. 1. 82–87.
- [11] Andrzej Jaszkiewicz. 2018. Improved quick hypervolume algorithm. *Computers & Operations Research* 90 (2018), 72–83.
- [12] Mikkel 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.
- [13] Hsiang-Tsung 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.
- [14] Renaud Lacour, Kathrin Klamroth, and Carlos M. Fonseca. 2017. A Box Decomposition Algorithm to Compute the Hypervolume Indicator. *Computers & Operations Research* 79 (2017), 347–360.
- [15] Ke Li, Kalyanmoy Deb, Qingfu Zhang, and Qiang Zhang. 2017. Efficient Non-domination Level Update Method for Steady-State Evolutionary Multiobjective Optimization. *IEEE Transactions on Cybernetics* 47, 9 (2017), 2838–2849.
- [16] Margarita Markina and Maxim Buzdalov. 2017. Hybridizing Non-dominated Sorting Algorithms: Divide-and-Conquer Meets Best Order Sort. In *Proceedings of Genetic and Evolutionary Computation Conference Companion*. 153–154.
- [17] Kent McClymont and Ed Keedwell. 2012. Deductive Sort and Climbing Sort: New Methods for Non-Dominated Sorting. *Evolutionary computation* 20, 1 (2012), 1–26.
- [18] Sumit Mishra, Sriparna Saha, and Samrat Mondal. 2016. Divide and conquer based non-dominated sorting for parallel environment. In *Proceedings of Congress on Evolutionary Computation*. 4297–4304.
- [19] Yakov Nekrich. 2011. A Fast Algorithm for Three-Dimensional Layers of Maxima Problem. In *Algorithms and Data Structures*. Number 6844 in Lecture Notes in Computer Science. 607–618.
- [20] 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.
- [21] Luis M. S. Russo and Alexandre P. Francisco. 2014. Quick Hypervolume. *IEEE Transactions on Evolutionary Computation* 18, 4 (2014).
- [22] N. Srinivas and Kalyanmoy Deb. 1994. Multiobjective Optimization Using Non-dominated Sorting in Genetic Algorithms. *Evolutionary Computation* 2, 3 (1994), 221–248.
- [23] Andrey Vasin and Maxim Buzdalov. 2016. A Faster Algorithm for the Binary Epsilon Indicator Based on Orthant Minimum Search. In *Proceedings of Genetic and Evolutionary Computation Conference*. 613–620.
- [24] Handing Wang and Xin Yao. 2014. Corner Sort for Pareto-Based Many-Objective Optimization. *IEEE Transactions on Cybernetics* 44, 1 (2014), 92–102.
- [25] Ilya Yakupov and Maxim Buzdalov. 2017. Improved Incremental Non-dominated Sorting for Steady-State Evolutionary Multiobjective Optimization. In *Proceedings of Genetic and Evolutionary Computation Conference*. 649–656.
- [26] Xingyi Zhang, Ye Tian, Ran Cheng, and Yaochu Jin. 2015. An Efficient Approach to Nondominated Sorting for Evolutionary Multiobjective Optimization. *IEEE Transactions on Evolutionary Computation* 19, 2 (2015), 201–213.
- [27] Eckart Zitzler and Simon Künzli. 2004. Indicator-Based Selection in Multiobjective Search. In *Parallel Problem Solving from Nature – PPSN VIII*. Number 3242 in Lecture Notes in Computer Science. 832–842.
- [28] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. 2001. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In *Proceedings of the EUROGEN'2001 Conference*. 95–100.
- [29] Eckart Zitzler and Lothar Thiele. 1999. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation* 3, 4 (1999), 257–271.
- [30] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. 2003. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation* 7, 2 (2003), 117–132.