ND-Tree-based update: a Fast Algorithm for the Dynamic Non-Dominance Problem

Andrzej Jaszkiewicz, Thibaut Lust

ND-Tree-based update: a Fast Algorithm for the Dynamic Non-Dominance Problem

Andrzej Jaszkiewicz, Thibaut Lust

***Abstract*—In this paper we propose a new method called ND-Tree-based update (or shortly ND-Tree) for the dynamic non-dominance problem, i.e. the problem of online update of a Pareto archive composed of mutually non-dominated points. It uses a new ND-Tree data structure in which each node represents a subset of points contained in a hyperrectangle defined by its local approximate ideal and nadir points. By building subsets containing points located close in the objective space and using basic properties of the local ideal and nadir points we can efficiently avoid searching many branches in the tree. ND-Tree may be used in multiobjective evolutionary algorithms and other multiobjective metaheuristics to update an archive of potentially non-dominated points. We prove that the proposed algorithm has sub-linear time complexity under mild assumptions. We experimentally compare ND-Tree to the simple list, Quad-tree, and M-Front methods using artificial and realistic benchmarks with up to 10 objectives and show that with this new method substantial reduction of the number of point comparisons and computational time can be obtained. Furthermore, we apply the method to the non-dominated sorting problem showing that it is highly competitive to some recently proposed algorithms dedicated to this problem.**

***Index Terms*—Multiobjective optimization, Pareto archive, Many-objective optimization, Dynamic non-dominance problem, Non-dominated sorting**

I. Introduction

In this paper we consider the dynamic non-dominance problem [1], i.e. the problem of online update of a Pareto archive with a new candidate point. The Pareto archive is composed of mutually non-dominated points and this property must remain fulfilled following the addition of the new point.

The dynamic non-dominance problem is typically used in multiobjective evolutionary algorithms (MOEAs) and more generally in other multiobjective metaheuristics (MOMHs), whose goal is to generate a good approximation of the Pareto front. Many MOEAs and other MOMHs use an external archive of potentially non-dominated points, i.e. a Pareto archive containing points not dominated by any other points generated so far, see e.g. [2], [3], [4], [5], [6], [7], [8], [9]. We consider here MOEAs that generate iteratively new candidate points and use them immediately to update a Pareto archive. Updating a Pareto archive with a new point y means that:

- y is added to the Pareto archive if it is non-dominated w.r.t. any point in the Pareto archive,
- all points dominated by y are removed from the Pareto archive.

The time needed to update a Pareto archive, in general, increases with a growing number of objectives and a growing number of points. In some cases it may become a crucial part of the total running time of a MOEA. The simplest data structure for storing a Pareto archive is a plain list. When a new point y is added, y is compared to all points in the Pareto archive until either all points are checked or a point dominating y is found. In order to speed up the process of updating a Pareto archive some authors proposed the use of specialized data structures and algorithms, e.g. Quad-tree [10]. However, the results of computational experiments reported in literature are not conclusive and in some cases such data structures may in fact increase the update time compared to the simple list.

A frequently used approach allowing reduction of the time needed to update a Pareto archive is the use of bounded archives [11] where the number of points is limited and some potentially non-dominated points are discarded. Please note, however, that such an approach always reduces the quality of the archive. In particular, one of the discarded points could be the one that would be selected by the decision maker if the full archive was known. Bounded archives may be especially disadvantageous in the case of many-objective problems, since with a growing number of dimensions it is more and more difficult to represent a large set with a smaller sample of points. The use of bounded archives may also lead to some technical difficulties in MOEAs (see [11]). Summarizing, if an unbounded archive can be efficiently managed and updated, it is advantageous to use this kind of archive.

In this paper, our contribution is fourfold: firstly, we propose a new method, called ND-Tree-based update, for the dynamic non-dominance problem. The method is based on a dynamic division of the objective space into hyperrectangles, which allows to avoid many comparisons of objective function values. Secondly, we show that the new method has sub-linear time complexity under mild assumptions. Thirdly, a thorough experimental study on different types of artificial and realistic sets shows that we can obtain substantial computational time reductions compared to state-of-the-art methods. Finally, we apply ND-Tree-based update to the non-dominated sorting problem obtaining promising results in comparison to some recently proposed dedicated algorithms.

The remainder of the paper is organized as follows. Basic definitions related to multiobjective optimization are given in Section II. In Section III, we present a state of the art of the methods used for online updating a Pareto archive. The main contribution of the paper, i.e. ND-Tree-based update method

A. Jaszkiewicz is with Poznan University of Technology, Faculty of Computing, Institute of Computing Science, ul. Piotrowo 2, 60-965 Poznan, Poland, e-mail: andrzej.jaszkiewicz@put.poznan.pl.

T. Lust is with Sorbonne Universités, UPMC, Université Paris 06, CNRS, LIP6, UMR 7606, F-75005, Paris, France, e-mail: thibaut.lust@lip6.fr.

is described in Section IV. Computational experiments are reported and discussed in Section V. In sections VI, ND-Tree-based is applied to the non-dominated sorting problem.

II. Basic definitions

A. Multiobjective optimization

We consider a general multiobjective optimization (MO) problem with a feasible set of solutions $\mathcal{X}$ and p objective functions $y_k(x)$ to minimize. The image of the feasible set in the objective space is a set of points $\mathcal{Y} = y(\mathcal{X})$ where $y(x) = (y_1(x), y_2(x), \ldots, y_p(x))$.

In MO, points are usually compared according to the *Pareto dominance relation*:

Definition 1. *Pareto dominance relation: we say that a point* $u = (u_1, ..., u_p)$ dominates *a point* $v = (v_1, ..., v_p)$ *if, and only if,* $u_k \leq v_k \; \forall\, k \in \{1, \ldots, p\} \wedge \exists\, k \in \{1, \ldots, p\} : u_k < v_k$. *We denote this relation by* $u \succ v$.

Definition 2. *Non-dominated point: a point* y^* *corresponding to a feasible solution is called* non-dominated *if there does not exist any other point* $y \in \mathcal{Y}$ *such that* $y \succ y^*$. *The set* $\mathcal{Y}_N$ *of all non-dominated points is called* Pareto front.

Definition 3. *Coverage relation: we say that a point* u covers *a point* v *if* $u \succ v$ *or* $u = v$. *We denote this relation by* $u \succeq v$.

Please note that coverage relation is sometimes referred to as weak dominance [12].

Definition 4. *Mutually non-dominated relation: we say that two points are mutually non-dominated or non-dominated w.r.t. each other if neither of the two points covers the other one.*

Definition 5. *Pareto archive (*Y_N*): set of points such that any pair of points in the set are mutually non-dominated, i.e* $\forall y \in Y_N, \nexists y' \in Y_N \,|\, y' \succeq y$.

In the context of MOEAs, the Pareto archive contains the mutually non-dominated points generated so far (i.e. at a given iteration of a MOEA) that approximates the Pareto front $\mathcal{Y}_N$. In other words Y_N contains points that are potentially non-dominated at a given iteration of the MOEA.

Please note that in MOEAs not only points but also representations of solutions are preserved in the Pareto archive, but the above definition is sufficient for the purpose of this paper.

The new method ND-Tree-based update is based on the (approximate) local ideal and nadir points that we define below.

Definition 6. *The local ideal point of a subset* $\mathcal{S} \subseteq Y_N$ *denoted as* $z^*(\mathcal{S})$ *is the point in the objective space composed of the best coordinates of all points belonging to* $\mathcal{S}$, *i.e.* $z^*_k(\mathcal{S}) = \min\limits_{y \in \mathcal{S}} \{y_k\}, \forall\, k \in \{1, \ldots, p\}$. *A point* $\widehat{z}^*(\mathcal{S})$ *such that* $\widehat{z}^*(\mathcal{S}) \succeq z^*(\mathcal{S})$ *will be called* Approximate local ideal point.

Naturally, the (approximate) local ideal point covers all points in $\mathcal{S}$.

Definition 7. *The local nadir point of a subset* $\mathcal{S} \subseteq Y_N$ *denoted as* $z_*(\mathcal{S})$ *is the point in the objective space composed of the worst coordinates of all points belonging to* $\mathcal{S}$, *i.e.* $z_{*k}(\mathcal{S}) = \max\limits_{y \in \mathcal{S}} \{y_k\}, \forall\, k \in \{1, \ldots, p\}$. *A point* $\widehat{z}_*(\mathcal{S})$ *such that* $z_*(\mathcal{S}) \succeq \widehat{z}_*(\mathcal{S})$ *will be called* Approximate local nadir point.

Naturally, the (approximate) local nadir point is covered by all points in $\mathcal{S}$.

B. Dynamic non-dominance problem

The problem of updating a Pareto archive (also called *non-dominance problem*), can be divided into two classes: the *static* non-dominance problem is to find the set of non-dominated points Y_N among a set of points Y. The other class is the *dynamic* non-dominance problem [1] that typically occurs in MOEAs. We formally define this problem as follows. Consider a candidate point y and a Pareto archive Y_N. The problem is to update Y_N with y and consists in the following operations. If y is covered by at least one point in Y_N, y is discarded and Y_N remains unchanged. Otherwise, y is added to Y_N. Moreover, if some points in Y_N are dominated by y, all these points are removed from Y_N, in order to keep only mutually non-dominated points (see Algorithm 1).

Algorithm 1 `DynamicNonDominance`

Parameter $\updownarrow$: A Pareto archive Y_N
Parameter $\downarrow$: New candidate point y

if $(\nexists\, y' \in Y_N \mid y' \succeq y)$ **then**
  $Y_N \leftarrow Y_N \cup \{y\}$
  for each $(y' \in Y_N \,|\, y \succ y')$ **do**
    $Y_N \leftarrow Y_N \backslash \{y'\}$

In this work we consider only the dynamic non-dominance problem. Note that in general static problems may be solved more effectively than their dynamic counterparts since they have access to richer information. Indeed, some efficient algorithms for static non-dominance problem have been proposed, see [13], [14], [15], [16].

MOEAs and other MOMHs usually update the Pareto archive using the dynamic version of the non-dominance problem, i.e. the Pareto archive is updated with each newly generated candidate point. In some cases it could be possible to store all candidate points and then solve the static non-dominance problem. The latter approach has, however, some disadvantages:

- MOEAs need to store not only points in the objective space but also full representations of solutions in the decision space. Thus, storing all candidate points with corresponding solutions may be very memory consuming.
- Some MOEAs use the Pareto archive during the run of the algorithm, i.e. Pareto archive is not just the final output of the algorithm. For example in [17], one of the parents is selected from the Pareto archive. In [7] the success of adding a new point to the Pareto archive influences the probability of selecting weight vectors in further iterations. The same applies to other MOMHs as well. For example, the Pareto local search (PLS) method [18] works directly with the Pareto archive and searches

neighborhood of each solution from the archive. In such methods, computation of the Pareto archive cannot be postponed till the end of the algorithm.

Note that as suggested in [19] the dynamic non-dominance problem may also be used to speed up the non-dominated sorting procedure used in many MOEAs. As the Pareto archive contains all non-dominated points generated so far the first front is immediately known and the non-dominated sorting may be applied only to the subset of dominated points. Using this technique, Drozdík *et al.* showed that their new method called M-Front could obtain better performance than Deb's fast nondominated sorting [20] and Jensen-Fortin's algorithm [21], [22], one of the fastest non-dominated sorting algorithms.

III. State of the art

We present here a number of methods for the dynamic non-dominance problem proposed in literature and used in the comparative experiment. This review is not supposed to be exhaustive. Other methods can be found in [23], [24] and reviews in [25], [26]. We describe linear list, Quad-tree and one recent method, M-Front [19].

A. Linear List

1) General case: In this structure, a new point is compared to all points in the list until a covering point is found or all points are checked. The point is only added if it is non-dominated w.r.t. all points in the list, that is in the worst case we need to browse the whole list before adding a point. The complexity in terms of number of points comparison is thus in $\mathcal{O}(N)$ with N the size in the list.

2) Biobjective case: sorted list: When only two objectives are considered, we can use the following specific property: if we sort the list according to one objective (let's say the first), the non-dominated list is also sorted according to the second objective. Therefore, roughly speaking, updating the list can be efficiently done in the following way. We first determine the potential position i of the new candidate point in the sorted list according to its value of the first objective, with a binary search. If the new point is not dominated by the preceding one in the list (if there is one), the new point is not dominated and can be inserted at position i. If the new point has been added, we need to check if there are some dominated points: we browse the next points in the list, until a point is found that has a better evaluation according to the second objective. All the points found that have a worse evaluation according to the second objective have to be removed since they are dominated by the new point.

The worst-case complexity is still in $\mathcal{O}(N)$ since it can happen that a new point has to be compared to all the other points (in the special case where we add a new point in the first position and all the points in the sorted list are dominated by this new point). But on average, experiments show that the behavior of this structure for handling biobjective updating problems is much better than the simple list.

The algorithm of this method is given in Algorithm 2 (for the sake of clarity, we only present the case where the candidate point y has a distinct value for the first objective compared to all the other points in the archive Y_N).

Algorithm 2 `Sorted list`

Parameter $\updownarrow$: A biobjective Pareto archive Y_N
Parameter $\downarrow$: New candidate point y

if $Y_N = \emptyset$ **then**
  $Y_N \leftarrow Y_N \cup \{y\}$
else
  --| Looking for the position i of y in Y_N
  $i \leftarrow \texttt{BinarySearch}(Y_N, y_1)$
  if $(i = 0)$ **or** $(y_2 < y_2^{(i-1)})$ **then**
    --| y is added at position i in Y_N
    $\texttt{Insert}(Y_N, y, i)$
    $j \leftarrow i + 1$
    while $(j < |Y_N|)$ **and** $(y_2 \leq y_2^j)$ **do**
      --| y^j is dominated
      $Y_N \leftarrow Y_N \backslash \{y^j\}$
      $j \leftarrow j + 1$

B. Quad-tree

The use of Quad-tree for storing potentially non-dominated points was proposed by Habenicht [27] and further developed by Sun and Steuer [28] and Mostaghim and Teich [10]. In Quad-tree, points are located in both internal nodes and leaves. Each node may have p^2 children corresponding to each possible combination of results of comparisons on each objective where a point can either be better or not worse. In the case of mutually non-dominated points $p^2 - 2$ children are possible since the combinations corresponding to dominating or covered points are not used. Quad-tree allows for a fast checking if a new point is dominated or covered. A weak point of this data structure is that when an existing point is removed its whole sub-tree has to be re-inserted to the structure. Thus, removal of a dominated point is in general costly.

C. M-Front

M-Front has been proposed relatively recently by Drozdík *et al.* [19]. The idea of of M-Front is as follows. Assume that in addition to the new point y a reference point ref relatively close to y and belonging to the Pareto archive Y_N is known. The authors define two sets:

$$RS_U(y, ref) = \{z \in Y_N \mid \exists k \in \{1, \ldots, p\} : z_k \geq ref_k \wedge z_k \leq y_k\}$$

$$RS_L(y, ref) = \{z \in Y_N \mid \exists k \in \{1, \ldots, p\} : z_k \geq y_k \wedge z_k \leq ref_k\}$$

and prove that if a point $z \in Y_N$ is dominated by y then it belongs to $RS_L(y, ref)$ and if z dominates y then it belongs to $RS_U(y, ref)$. Thus, it is sufficient to compare the new points to sets $RS_L(y, ref)$ and $RS_U(y, ref)$ only. To find all points with objective values in a certain interval M-Front uses additional indexes one for each objective. Each index sorts the Pareto archive according to one objective.

To find a reference point close to y, M-Front uses the k-d tree data structure. The k-d tree is a binary tree, in which each intermediate node divides the space into two parts based on a value of one objective. While going down the tree the algorithm cycles over particular objectives, selecting one

objective for each level. Drozdík *et al.* [19] suggest to store references to points in leaf nodes only, while intermediate nodes keep only split values.

IV. ND-Tree-based update

A. Presentation

In this section we present the main contribution of the paper. The new method for updating a Pareto archive is based on the idea of recursive division of archive Y_N into subsets contained in different hyperrectangles. This division allows to considerably reduce the number of comparisons to be made.

More precisely, consider a subset $\mathcal{S} \subseteq Y_N$ composed of mutually non-dominated points and a new candidate point y. Assume that some approximate local ideal $\widehat{z}^*(\mathcal{S})$ and approximate local nadir points $\widehat{z}_*(\mathcal{S})$ of $\mathcal{S}$ are known. In other words, all points in $\mathcal{S}$ are contained in the axes-parallel hyperrectangle defined by $\widehat{z}^*(\mathcal{S})$ and $\widehat{z}_*(\mathcal{S})$.

We can define the following simple properties that allow to compare a new point y to the whole set $\mathcal{S}$:

Property 1. If y is covered by $\widehat{z}_*(\mathcal{S})$, then y is covered by each point in $\mathcal{S}$ and thus can be rejected. This property is a straightforward consequence of the transitivity of the coverage relation.

Property 2. If y covers $\widehat{z}^*(\mathcal{S})$, then each point in $\mathcal{S}$ is covered by y. This property is also a straightforward consequence of the transitivity of the coverage relation.

Property 3. If y is non-dominated w.r.t. both $\widehat{z}_*(\mathcal{S})$ and $\widehat{z}^*(\mathcal{S})$, then y is non-dominated w.r.t. each point in $\mathcal{S}$.

Proof. If y is non-dominated w.r.t. $\widehat{z}_*(\mathcal{S})$ then there is at least one objective on which y is worse than $\widehat{z}_*(\mathcal{S})$ and thus worse than each point in $\mathcal{S}$. If y is non-dominated w.r.t. $\widehat{z}^*(\mathcal{S})$ then there is at least one objective on which y is better than $\widehat{z}^*(\mathcal{S})$ and thus better than each point in $\mathcal{S}$. So, there is at least one objective on which y is better and at least one objective on which y is worse than each point in $\mathcal{S}$. □

If none of the above properties holds, i.e. y is neither covered by $\widehat{z}_*(\mathcal{S})$, does not cover $\widehat{z}^*(\mathcal{S})$, nor is non-dominated w.r.t. both $\widehat{z}_*(\mathcal{S})$ and $\widehat{z}^*(\mathcal{S})$, then all situations are possible, i.e. y may either be non-dominated w.r.t. all points in $\mathcal{S}$, covered by some points in $\mathcal{S}$ or dominate some points in $\mathcal{S}$. This can be illustrated by showing examples of each of the situations. Consider for example a set $\mathcal{S} = \{(1,1,1),(0,2,2),(2,2,0)\}$ with $\widehat{z}^*(\mathcal{S}) = z^*(\mathcal{S}) = (0,1,0)$ and $\widehat{z}_*(\mathcal{S}) = z_*(\mathcal{S}) = (2,2,2)$. A new point $(1,1,0)$ dominates a point in $\mathcal{S}$, a new point (1, 1, 2) is dominated (thus covered) by a point in $\mathcal{S}$, and points $(0,3,0)$ and $(2,0,1)$ are non-dominated w.r.t. all points in $\mathcal{S}$.

The properties are graphically illustrated for the biobjective case in Figure 1. As can be seen in this figure, in the biobjective case, if y is covered by $\widehat{z}^*(\mathcal{S})$ and y is non-dominated w.r.t. $\widehat{z}_*(\mathcal{S})$ then y is dominated by at least one point in $\mathcal{S}$. Note, however, that this does not hold in the case of three and more objectives as shown in the above example - the point $(0,3,0)$ is covered by $\widehat{z}^*(\mathcal{S}) = (0,1,0)$, non-dominated w.r.t. $\widehat{z}_*(\mathcal{S}) = (2,2,2)$ and $(0,3,0)$ is not dominated by any points in $\mathcal{S}$.

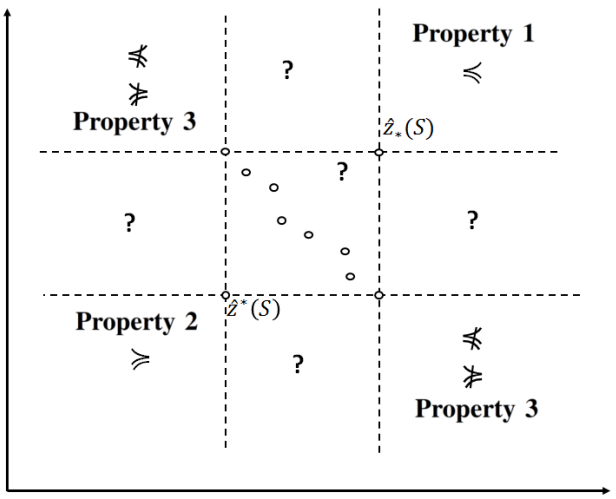

Fig. 1. Comparison of a new point to all points in set $\mathcal{S}$ based on comparisons to $\widehat{z}^*(\mathcal{S})$ and $\widehat{z}_*(\mathcal{S})$ only.

In fact it is possible to distinguish more specific situations when none of the three properties hold, e.g. a situation when a new point may be covered but cannot dominate any point, but since we do not distinguish them in the proposed algorithm we do not define them formally.

The above properties allow in some cases to quickly compare a new candidate point y to all points in set $\mathcal{S}$ without the need for further comparisons to individual points belonging to $\mathcal{S}$. Such further comparisons are necessary only if none of the three properties hold. Intuitively, the closer the approximate local ideal and nadir points the more likely it is that further comparisons can be avoided. To obtain close approximate local ideal and nadir points we should:

- Split the whole set of non-dominated points into subsets of points located close in the objective space.
- Have good approximations of the exact local ideal and nadir points. On the other hand calculation of the exact points may be computationally demanding and a reasonable approximation may assure the best overall efficiency.

Based on these properties, we can now define the ND-Tree data structure.

Definition 8. *ND-Tree data structure is a tree with the following properties:*

1) *With each node n is associated a set of points $\mathcal{S}(n)$.*
2) *Each leaf node contains a list $\mathcal{L}(n)$ of points and $\mathcal{S}(n) = \mathcal{L}(n)$.*
3) *For each internal node n, $\mathcal{S}(n)$ is the union of disjoint sets associated with all children of n.*
4) *Each node n stores an approximate ideal point $\widehat{z^*}(\mathcal{S}(n))$ and approximate nadir point $\widehat{z}_*(\mathcal{S}(n))$.*
5) *If n' is a child of n, then $\widehat{z^*}(\mathcal{S}(n)) \succeq \widehat{z^*}(\mathcal{S}(n'))$ and $\widehat{z}_*(\mathcal{S}(n')) \succeq \widehat{z}_*(\mathcal{S}(n))$.*

The algorithm for updating a Pareto archive with ND-Tree is given in Algorithm 3. The idea of the algorithm is as follows. We start by checking if the new point y is covered or non-dominated w.r.t. all points in Y_N by going through the nodes

of ND-Tree and skipping children (and thus their sub-trees) for which **Property 3** holds. This procedure is presented in Algorithm 4.

The new point is first compared to the approximate ideal point ($\widehat{z^*}(\mathcal{S}(n))$ and nadir point ($\widehat{z_*}(\mathcal{S}(n))$ of the current node. If the new point is dominated by $\widehat{z_*}(\mathcal{S}(n)$ it is immediately rejected (**Property 1**). If $\widehat{z^*}(\mathcal{S}(n)$ is covered, the node and its whole sub-tree is deleted (**Property 2**). Otherwise if $\widehat{z^*}(\mathcal{S}(n)) \succeq y$ or $y \succeq \widehat{z_*}(\mathcal{S}(n))$, the node needs to be analyzed. If n is an internal node we call the algorithm recursively for each child. If n is a leaf node, y may be dominated by or dominate some points of n and it is necessary to browse the whole list $\mathcal{L}(n)$ of the node n. If a point dominating y is found, y is rejected, and if a point dominated by y is found, the point is deleted from $\mathcal{L}(n)$.

If after checking ND-Tree the new point was found to be non-dominated it is inserted by adding it to a close leaf (Algorithm 5). To find a proper leaf we start from the root and always select a child with closest distance to y. As a distance measure we use the Euclidean distance to the *middle point*, i.e. a point lying in the middle of line segment connecting approximate ideal and approximate nadir points.

Once we have reached a leaf node, we add the point y to the list $\mathcal{L}(n)$ of the node and possibly update the ideal and nadir points of the node n (Algorithm 7). However, if the size of $\mathcal{L}(n)$ became larger than the maximum allowed size of a leaf set, we need to split the node into a predefined number of children. To create children that contain points that are more similar to each other than to those in other children, we use a simple clustering heuristic based on Euclidean distance (see Algorithm 6).

The approximate local ideal and nadir points are updated only when a point is added. We do not update them when point(s) are removed since it is a more complex operation. This is why we deal with approximate (not exact) local ideal and nadir points.

Algorithm 3 `ND-Tree-based update`

Parameter $\updownarrow$: A Pareto archive Y_N organized as ND-Tree
Parameter $\downarrow$: New candidate point y

if $Y_N = \emptyset$ **then**
  Create a leaf node n with $\mathcal{L}(n) = \{y\}$ and use it as a root
else
  $n \leftarrow$ root node
  if `UpdateNode`($n \updownarrow$,$y \downarrow$) **then**
    `Insert`($n \updownarrow$,$y \downarrow$)

Algorithm 4 `UpdateNode`

Parameter $\updownarrow$: A node n
Parameter $\downarrow$: New candidate point y
Parameter $\uparrow$: Boolean (True if y is not dominated by any points in the tree of root n, False otherwise)

if $\widehat{z_*}(\mathcal{S}(n)) \succeq y$ **then**
  --| Property 1, y is rejected
  return False
else if $y \succeq \widehat{z^*}(\mathcal{S}(n))$ **then**
  --| Property 2
  Remove n and its whole sub-tree
else if $\widehat{z^*}(\mathcal{S}(n)) \succeq y$ **or** $y \succeq \widehat{z_*}(\mathcal{S}(n))$ **then**
  if n is a leaf node **then**
    for each $z \in \mathcal{L}(n)$ **do**
      if $z \succeq y$ **then**
        return False
      else if $y \succ z$ **then**
        $\mathcal{L}(n) \leftarrow \mathcal{L}(n) \backslash \{z\}$
  else
    for each Child n' of n **do**
      if not `UpdateNode` ($n' \updownarrow$,$y \downarrow$) **then**
        return False
      else
        if n' became empty **then**
          Remove n'
    if there is only one child n' remaining **then**
      Remove node n and use n' in place of n
else
  --| Property 3
  Skip this node
return True

Algorithm 5 `Insert`

Parameter $\updownarrow$: A node n
Parameter $\downarrow$: New candidate point y

if n is a leaf node **then**
  $\mathcal{L}(n) \leftarrow \mathcal{L}(n) \cup \{y\}$
  `UpdateIdealNadir` ($n \updownarrow$,$y \downarrow$)
  if Size of $\mathcal{L}(n)$ became larger than maximum size of a leaf set **then**
    `Split` ($n \updownarrow$)
else
  Find child n' of n being closest to y
  `Insert`($n' \updownarrow$,$y \downarrow$)

B. Comparison to existing methods

Like other methods ND-Tree-based update uses a tree structure to speed up the process of updating the Pareto archive. The tree and its use is, however, quite different from Quad-tree or k-d tree used in M-Front. For example, both Quad-tree or k-d tree partition the objective space based on comparisons on particular objectives, while in ND-Tree the space is partitioned based on the distances of points. Both Quad-tree or k-d tree have strictly defined degrees. In k-d tree it is always two (binary tree) while in Quad-tree it depends on the number of objectives. In ND-Tree the degree is a parameter. In Quad-tree the points are kept in both internal nodes and leaves, while ND-Tree keeps points in leaves only. In M-Front k-d tree is used to find an approximate nearest neighbor and the Pareto archive is updated using other structures (sorted lists for each objective). In our case, ND-Tree is the only data structure used

Algorithm 6 `Split`

Parameter $\updownarrow$: A node n

Find the point $z \in \mathcal{L}(n)$ with the highest average Euclidean distance to all other points in $\mathcal{L}(n)$
Create a new child n' with list set $\mathcal{L}(n') = \{z\}$
$\mathcal{L}(n) \leftarrow \mathcal{L}(n) \backslash \{z\}$
`UpdateIdealNadir` ($n' \updownarrow$,$z \downarrow$)
while The required number of children are not created **do**
  Find the point $z \in \mathcal{L}(n)$ with the highest average Euclidean distance to all points in all children of n
  Create a new child n' with an empty list set $\mathcal{L}(n')$
  $\mathcal{L}(n') \leftarrow \mathcal{L}(n') \cup \{z\}$
  `UpdateIdealNadir` ($n' \updownarrow$,$z \downarrow$)
  $\mathcal{L}(n) \leftarrow \mathcal{L}(n) \backslash \{z\}$
while $\mathcal{L}(n)$ is not empty **do**
  $z \leftarrow$ first point in $\mathcal{L}(n)$
  Find child n' of n being closest to z
  $\mathcal{L}(n') \leftarrow \mathcal{L}(n') \cup \{z\}$
  `UpdateIdealNadir` ($n' \updownarrow$,$z \downarrow$)
  $\mathcal{L}(n) \leftarrow \mathcal{L}(n) \backslash \{z\}$

Algorithm 7 `UpdateIdealNadir`

Parameter $\updownarrow$: A node n
Parameter $\downarrow$: New candidate point y

Check in any component of y is lower than corresponding component in $\widehat{z^*}(\mathcal{S}(n))$ or greater than corresponding component in $\widehat{z}_*(\mathcal{S}(n))$ and update the points if necessary
if $\widehat{z^*}(\mathcal{S}(n))$ or $\widehat{z}_*(\mathcal{S}(n))$ have been changed **then**
  if n is not a root **then**
    $np \leftarrow$ parent of n
    `UpdateIdealNadir` ($np \updownarrow$,$y \downarrow$)

by the algorithm.

C. Computational complexity

1) Worst case: The worst case for the `UpdateNode` and `Insert` algorithm is when we need to compare the new point to each intermediate node of the tree. For example, consider the following particular case: two objectives, ND-Tree with maximum leaf size equal to 2, 2 children, and constructed by processing the following list of points (with $K \in \mathbb{N}^*$): (0, 0), (1, -1), (2^K, -2^K), (2^{K-1}, -2^{K-1}),..., (4, -4), (2, -2). This list of points is constructed in such a way that the first two points are put in one leftmost leaf and the third point creates a separate leaf. Then each further point is closer to the child on the left side but finally after splitting the leftmost node the new point creates a new leaf. The ND-Tree obtained is shown in Figure 2.

Consider now that the archive is updated with point (0.5, -0.5). This point will need to be compared to all $N-1$ intermediate nodes and then to both points in leftmost leaf.

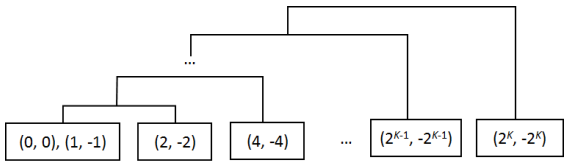

Fig. 2. Example of fully unbalanced ND-Tree.

In this case:

$$T(N) = 4 + T(N-1) \tag{1}$$

where N is the number of points in the archive and $T(N)$ is the number of point comparisons needed to update an archive of size N. Term 4 appears because we check two children and for each child approximate ideal and nadir points are compared. Solving the recurrence we get:

$$T(N) = 4 + 4N \tag{2}$$

Thus, the algorithm has $\mathcal{O}(N)$ time complexity.

We are not aware of any result showing that the worst-case time complexity of any algorithm for the dynamic non-dominance problem may be lower than $\mathcal{O}(N)$ in terms of point comparisons. So, our method does not improve the worst-case complexity but according to our experiments performs significantly better in practical cases.

2) Best case: Assume first that the candidate point is not covered by any point in Y_N. In the optimistic case, at each intermediate node the points are equally split into predefined number of children and there is only one child that has to be further processed (i.e. there is only one child for which none of the three properties hold). In fact we could consider even more optimistic distribution of points when the only node that has to be processed contains just one point, but the equal split is much more realistic assumption. In this case:

$$T(N) = 2C + T(N/C) = \Theta(\log_C N) \tag{3}$$

where C is the number of children. If the candidate point is covered by a point in Y_N the `UpdateNode` algorithm may stop even earlier and there will be no need to run `Insert` algorithm.

3) Average case: Analysis and even definition of average case for such complex algorithms is quite difficult. The simplest case to analyze is when each intermediate node has two children which allows us to follow the analysis of well-known algorithms like binary search or Quicksort. If a node has N points then one of the two children may have $1, \ldots, N-1$ points and the other child the remaining number of points. Assuming that each split has equal probability and only one child is selected:

$$T(N) = 4 + \frac{1}{N} \sum_{k=1}^{N-1} T(k) \tag{4}$$

Multiplying both sides by N:

$$NT(N) = 4N + \sum_{k=1}^{N-1} T(k) \tag{5}$$

Assuming that $N \geq 2$:

$$(N-1)T(N-1) = 4(N-1) + \sum_{k=1}^{N-2} T(k) \tag{6}$$

Subtracting equations 5 and 6:

$$\begin{aligned} NT(N) - (N-1)T(N-1) = \\ 4N - 4(N-1) + \sum_{k=1}^{N-1} T(k) - \sum_{k=1}^{N-2} T(k) \end{aligned} \tag{7}$$

Simplifying:

$$T(N) = \frac{4}{N} + T(N-1) \tag{8}$$

Solving this recurrence:

$$T(N) = 4H_N - 1 \tag{9}$$

where H_N is N-th harmonic number. Using well-known properties of harmonic numbers we get $T(N) = \Theta(\log N)$.

We can expect, however, that in realistic cases more than one child will need to be further processed in `UpdateNode` algorithm because the candidate point may cover approximate nadir points or may be covered by approximate ideal points of more than one child. Assume that the probability of selecting both children is equal to c_1. Then:

$$T(N) = 4 + \frac{1}{N}(1 + c_1) \sum_{k=1}^{N-1} T(k) \tag{10}$$

Following the above reasoning we get:

$$T(N) = \frac{2}{N} + \frac{N + c_1}{N} T(N-1) \tag{11}$$

Solving this recurrence, we obtain:

$$T(N) = \frac{\Gamma(c_1 + N + 1)}{\Gamma(N+1)} - \frac{2}{c_1} \tag{12}$$

(that can be checked by substituting (12) in (11)).
Since

$$\lim_{N \to \infty} \frac{\Gamma(N+\alpha)}{\Gamma(N) N^{\alpha}} = 1 \tag{13}$$

We have:

$$T(N) = \Theta(N^{c_1}) \tag{14}$$

and the algorithm remains sub-linear for any $c_1 < 1$. In the worst case, both children need to be selected, so $c_1 = 1$ and $\mathrm{T}(N) = \Theta(N)$ which confirms the analysis presented above.

This analysis may give only approximate insight into behaviour of the algorithm since in `UpdateNode` algorithm c_1 will not be constant at each level. We may rather expect that while going down the tree from the root towards leaves the probability that two children will need to be processed will decrease because approximate nadir and ideal points of the children will lie closer. Anyway, this analysis shows that the performance of `UpdateNode` algorithm may be improved by decreasing the probability that a child has to be processed. This is why we try to locate in one node points lying close to each other in `Insert` and `Split` algorithms.

In `Insert` algorithm always one child is processed, so the time complexity remains $\Theta(\log N)$ in average case.

The main part of `Split` algorithm has constant time complexity since it depends only on the maximum size of a leaf set which is a constant parameter of the algorithm.

`UpdateIdealNadir` algorithm goes up the tree starting from a leaf which is equivalent to going down the tree and selecting just one child. So, its analysis is exactly the same as of `Insert` algorithm.

We also need to consider the complexity of the operation of removal of node n and its sub-tree. In the worst case, the removed node is the root, and thus all N point need to be removed. Such situation is very unlikely, since it happens when the new point dominates all points in the current archive. Typically, the new point will dominate only few points.

V. Computational experiments

We will show results obtained with ND-Tree and other methods in two different cases:

A) Results for artificially generated sets which allow us to easily control the number of points in the sets and the quality of the points.
B) Results for sets generated by a MOEA, namely MOEA/D [2] for the multiobjective knapsack problem.

We compare the simple list, sorted list (biobjective case), Quad-tree, M-Front and ND-Tree for these sets according to the CPU time [ms]. To avoid the influence of implementation details all methods were implemented from the scratch in C++ in as much homogeneous way as possible, i.e. when possible the same code was used to perform the same operations like Pareto dominance checks.

For the implementation of Quad-tree, we use Quad-tree2 version as described by Mostaghim and Teich [10].

For M-Front, we use as much as possible the description found in the paper of Drozdík *et al.* [19]. However the authors do not give the precise algorithm of k-d tree used in their method. In our implementation of k-d tree, when a leaf is reached a new division is made using the average value of the current level objective. The split value is average between the value of new point and the point in the leaf. Also like in Drozdík *et al.* [19] the approximate nearest neighbor is found exactly as in the standard exact nearest neighbor search, but only four evaluations of the distance are allowed. Note that at https://sites.google.com/site/ndtreebasedupdate/ we present results of an additional experiment showing that for a higher number of objectives the details of the implementation of k-d tree do not have any substantial influence on the running time.

We also noticed that a number of elements of M-Front can be improved to further reduce the running time. The improvement is in our opinion significant enough to call the new method M-Front-II. In particular in some cases M-Front-II was several times faster than original M-Front in our computational experiments. The modifications we introduced are as follows:

- In original M-Front the sets $RS_L(y, ref)$ and $RS_U(y, ref)$ are built explicitly and only then the points contained in these sets are compared to y. In M-Front-II we build them only implicitly, i.e. we

immediately compare the points that would be added to the sets to y.

- We analyze the objectives in such a way that we start with objectives for which $ref_k \leq y_k$. In other words, we start with points from set $RS_U(y, ref)$. Since many new points are dominated this allows to stop the search immediately when a point dominating y is found. Note that a similar mechanism is in fact used in original M-Front but only after the sets $RS_L(y, ref)$ and $RS_U(y, ref)$ are explicitly built.
- The last modification is more technical. M-Front uses linked lists (`std::list` in C++) to store the indexes and a hash-table (`std::unordered_map` in C++) to link points with their positions in these lists. We observed, however, that even basic operations like iterating over the list are much slower with linked lists than with static or dynamic arrays (like `std::vector` in C++). Thus we use dynamic arrays for the indexes. In this case, however, there is no constant iterator that could be used to link the points with their positions in these indexes. So, we use a binary search to locate a position of a point in the sorted index whenever it is necessary. The overhead of the binary search is anyway smaller than the savings due to the use of faster indexes.

For ND-Tree we use 20 as the maximum size of a leaf and $p+1$ as the number of children. These values of the parameters were found to perform well in many cases. We analyze the influence of these parameters later in this section.

The code, as well as test instances and data sets, are available at https://sites.google.com/site/ndtreebasedupdate/. All of the results have been obtained on an Intel Core i7-5500U CPU at 2.4 GHz.

A. Artificial sets

1) Basic, globally convex sets: The artificial sets are composed of n points with p objectives to minimize. The sets are created as follows. We generate randomly n points y^i in $\{0, \ldots, V_{max}\}^p$ with the following constraint: $\sum_{k=1}^{p}(V_{max} - y_k^i)^2 \leq V_{max}^2$. With this constraint, all the non-dominated points will be located inside the hypersphere with the center at $(V_{max}, \ldots, V_{max})$ and with a radius of length equal to V_{max}. In order to control the quality of the generated points, we also add a quality constraint: $\sum_{i=k}^{p}(V_{max} - y_k^i)^2 \geq (1-\epsilon) * V_{max}^2$. In this way, with a small ϵ, only high-quality points will be generated. We believe that it is a good model for points generated by real MOEAs since a good MOEA should generate points lying close to the true Pareto front. The hypersphere is a model of the true Pareto front and parameter ϵ controls the maximum distance from the hypersphere. We have generated data sets composed of 100 000 and 200 000 points, with $V_{max} = 10000$, and for $p = 2$ to 10. In the main experiment we use data sets with 100 000 points because for the larger sets running times of some methods became very long. Also because of very high running times for sets with many objectives, in the main experiment we used sets with up to 6 objectives. For each value of p, five different quality levels are considered: quality q1, $\epsilon = 0.5$; q2, $\epsilon = 0.25$; q3, $\epsilon = 0.1$; q4, $\epsilon = 0.05$; q5, $\epsilon = 0.01$. The fraction of non-dominated points grows both with increasing quality and number of objectives and in extreme cases all points may be non-dominated (see Table I).

TABLE I
NUMBERS OF NON-DOMINATED POINTS IN ARTIFICIAL SETS

p	Quality	$\lvert Y_N \rvert$ convex	$\lvert Y_N \rvert$ non-convex	$\lvert Y_N \rvert$ clustered
2	q1	519	379	449
2	q2	713	613	552
2	q3	1046	1037	785
2	q4	1400	1454	1059
2	q5	2735	2748	1781
3	q1	4588	2587	3729
3	q2	6894	5344	5514
3	q3	12230	11497	9720
3	q4	19095	18648	15502
3	q5	53813	53255	44173
4	q1	14360	6853	11963
4	q2	21680	16420	18120
4	q3	39952	37709	35460
4	q4	64664	63140	57725
4	q5	98283	98243	97137
5	q1	28944	13437	23966
5	q2	42246	34357	38028
5	q3	77477	75796	71063
5	q4	96002	95867	93842
5	q5	99975	99975	98521
6	q1	45879	22956	40483
6	q2	65195	57966	61096
6	q3	96687	96480	94978
6	q4	99788	99786	99652
6	q5	100000	100000	99999

2) Globally non-convex sets: In order to test whether the global convexity of the above sets influences the behavior of the tested methods we have also generated sets whose Pareto fronts are globally non-convex. They were obtained by simply changing the sign of each objective in the basic sets.

3) Clustered sets: In these sets, the points are located in small clusters. We have generated sets composed of 100 clusters, where each cluster contains 1000 points (the sets are thus composed of 100 000 points). The sets have been obtained as follows: we start with the 200 000 points from the basic convex sets. We select from the set one random point, and we then select the 999 points closest to this point (using the Euclidean distance). We repeat this operation 100 times to obtain the 100 clusters of the sets.

The shapes of exemplary biobjective globally convex, globally non-convex and clustered data sets can be seen at https://sites.google.com/site/ndtreebasedupdate/.

Each method was run 10 times for each set, with the points processed in a different random order for each run. The average running times for basic sets are presented in Figures 3 to 7. We use average values since the different values were generally well distributed around the average with small deviations. Please note that because of large differences the running time is presented in logarithmic scale.

In addition, in Figure 8 we illustrate the evolution of the running times according to the number of objectives for the data sets of intermediate quality q3. In this case we use sets with up to 10 objectives. With 7 and more objectives even the sets of intermediate quality q3 contain almost only non-dominated points (see Table I). This is why the running times

of the simple list are practically constant for 7 and more objectives, because the simple list boils down in this case to the comparison of each new point to all points in the list. The running times of all other methods including ND-Tree increase with a growing number of objectives, but ND-Tree remains 5.5 times faster than the second best method (Quad-tree) for sets with 10 objectives.

Furthermore, we measured the number of comparisons of points with the dominance relation for the data sets of intermediate quality q3 with $p = 2, ..., 10$. For ND-Tree-based

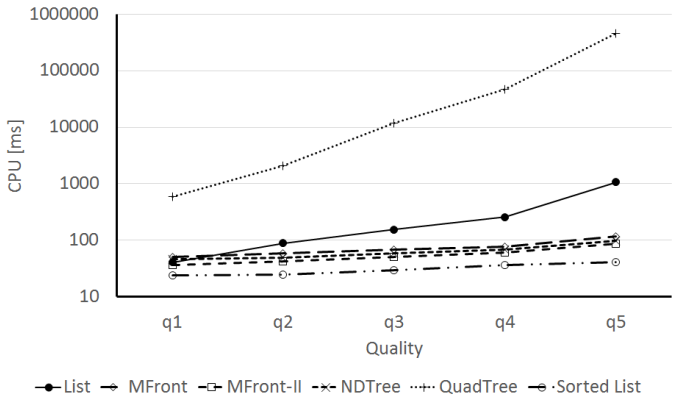

Fig. 3. CPU time (logarithmic scale) for biobjective convex data sets.

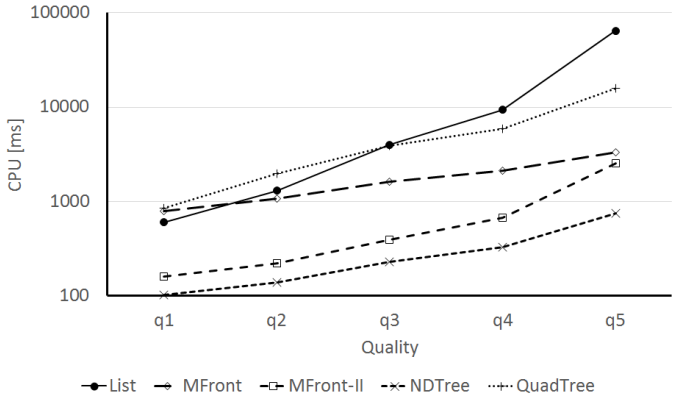

Fig. 4. CPU time (logarithmic scale) for three-objective convex data sets.

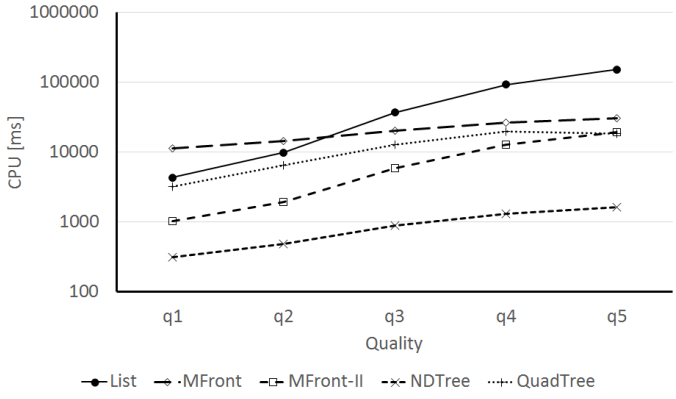

Fig. 5. CPU time (logarithmic scale) for four-objective convex data sets.

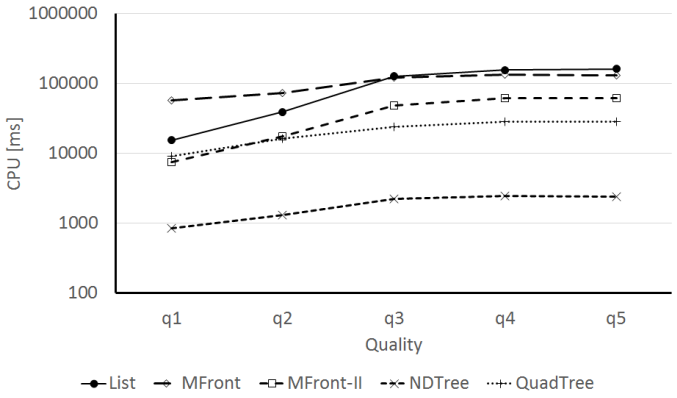

Fig. 6. CPU time (logarithmic scale) for five-objective convex data sets.

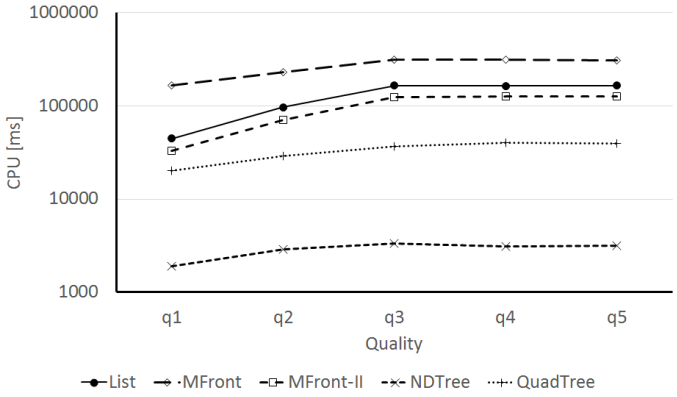

Fig. 7. CPU time (logarithmic scale) for six-objective convex data sets.

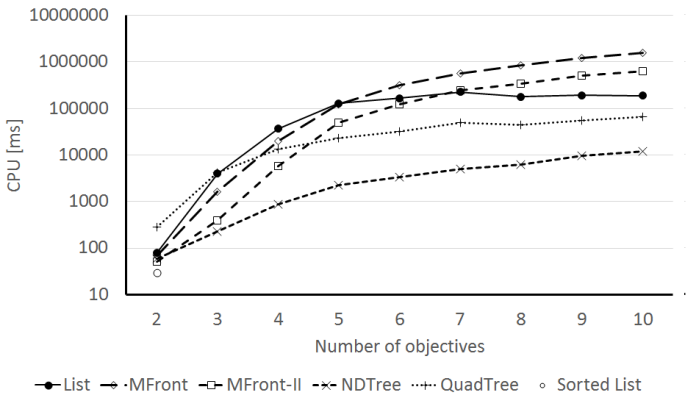

Fig. 8. CPU time (logarithmic scale) vs. number of objectives for convex data sets of quality q3.

TABLE II
POINT COMPARISONS PER MS

Method	Comparisons per ms
List	26 752
M-Front	3 476
M-Front-II	8 370
ND-Tree	17 733
Quad-tree	9 040

update it includes also comparisons to the approximate ideal and nadir points and for Quad-tree comparisons to sub-nodes. The results are presented in Figure 9. Please note that the results for the two versions of M-Front overlap in the Figure.

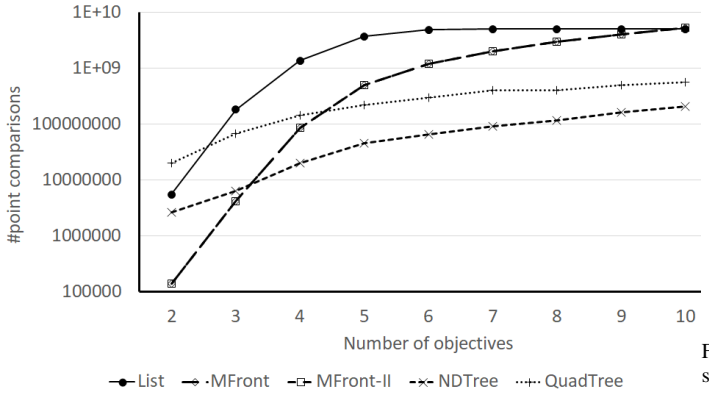

Fig. 9. Number of point comparisons (logarithmic scale) vs. number of objectives for convex data sets of quality q3.

The differences in running times cannot be fully explained by the number of point comparisons because the methods differ significantly in the number of point comparisons per milliseconds (see Table II). This ratio is highest for the simple list because this method performs very few additional operations. It is also relatively high for ND-Tree. Other methods perform many other operations than point comparisons that strongly influence their running times. This is particularly clear in comparison of M-Front and M-Front-II. These method perform the same number of point comparisons, but M-Front-II is several times faster than M-Front because in the latter method the sets $RS_L(y, ref)$ and $RS_U(y, ref)$ are built explicitly and this method uses slower linked lists. Overall, ND-Tree performs fewest number of point comparisons for data sets with $p \geq 4$. These results indicate that ND-Tree-based update substantially reduces the number of comparisons with respect to the simple list. For example for $p = 10$, all points in the data set are non-dominated, thus on average each of the 100000 new points has to be compared to an archive composed of 49999.5 points, while with ND-Tree-based update it only requires 2029 point comparisons on average.

The results obtained for non-convex and clustered sets were very similar to the results with basic sets. Thus, in Figures 10 to 13 we show only exemplary results for the three- and six-objective cases. These results indicate that the running times of the tested methods are not substantially affected by the global shape of the Pareto front.

4) Discussion of the results for artificial sets: The main observations from this experiment are:

- ND-Tree performs the best in terms of CPU time for all test sets with three and more objectives. In some cases the differences to other methods are of two orders of magnitude and in some cases the difference to the second best method is of one order of magnitude. ND-Tree behaves also very predictably, its running time grows slowly with increasing number of objectives and increasing fraction of non-dominated points.

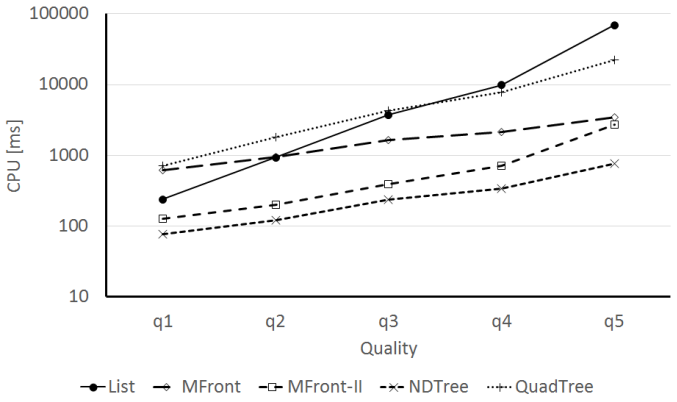

Fig. 10. CPU time (logarithmic scale) for three-objective non-convex data sets.

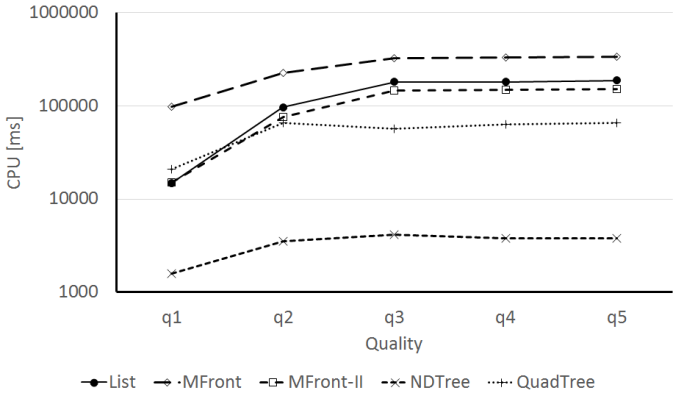

Fig. 11. CPU time (logarithmic scale) for six-objective non-convex data sets.

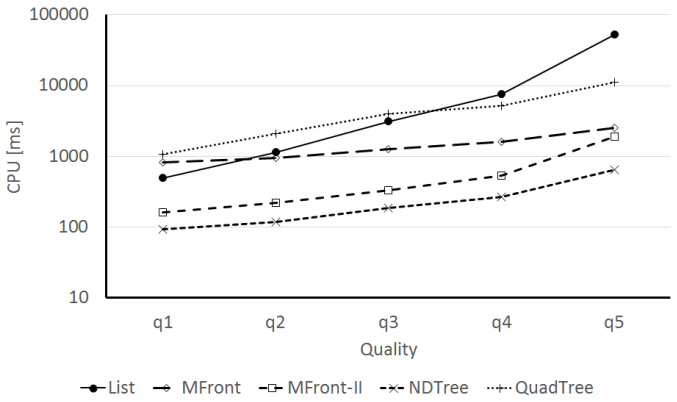

Fig. 12. CPU time (logarithmic scale) for three-objective clustered data sets.

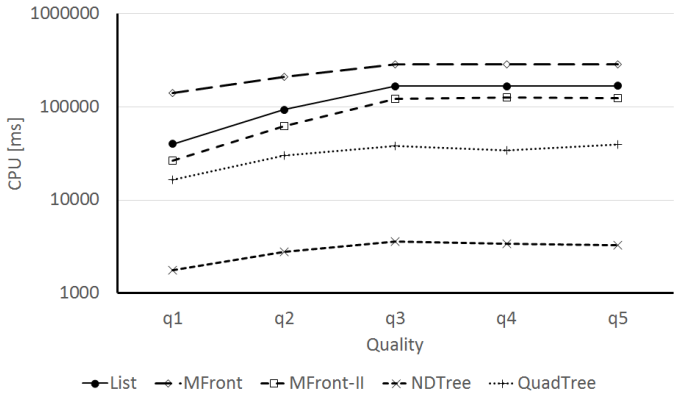

Fig. 13. CPU time (logarithmic scale) for six-objective clustered data sets.

- For biobjective instances sorted list is the best choice. In this case, M-Front and M-Front-II also behave very well since they become very similar to sorted list.
- The simple list obtains its best performances for data sets with many dominated points like $p = 2$ with lowest quality. In this case the new point is often dominated by many points, so the search process is quickly stopped after finding a dominating point.
- Quad-tree performs very badly for data sets with many dominated points, e.g. in biobjective instances where it is the worst method in all cases. In this case, many points added to Quad-tree are then removed and the removal of a point from Quad-tree is a costly operation. As discussed above when an existing point is removed its whole subtree has to be re-inserted to the structure. On the other hand, it is the second best method for most data sets with six and more objectives.
- M-Front-II is much faster than M-Front on data sets with larger fraction of dominated points. In this case, M-Front-II may find a dominating point faster without building explicitly the whole sets $RS_L(y, ref)$ and $RS_U(y, ref)$.
- The performance of both M-Front and M-Front-II deteriorates with an increasing number of objectives. With six and more objectives M-Front is the slowest method in all cases. Intuitively this can be explained by the fact that M-Front (both versions) uses each objective individually to reduce the search space. In the case of two objectives the values of one objective carry a lot of information since the order on one objective induces also the order on the other one. The more objectives, the less information we get from an order on one of them. Furthermore, sets $RS_L(y, ref)$ and $RS_U(y, ref)$ are in fact unions of corresponding sets for particular objectives, which also results in their growth. Finally, in many objective case, a reference point close on Euclidean distance does not need to be very close on each objective, since it will rather have a good balance of differences on many coordinates.

In an additional experiment we analyzed the evolution of the running times of all methods with increasing number of points. We decided to use one intermediate globally convex data set with $p = 4$ and quality q3. We used 200 000 points in this case and 10 runs for each method (see Figure 14). The CPU time is cumulative time of processing a given number of points. In addition, since the running time is much smaller for ND-Tree its results are presented in Figure 15 separately. We see that ND-Tree is the fastest method for any number of points and its cumulative running time grows almost linearly with the number of points. In other words, time of processing a single point is almost constant. Please note that unlike in other figures the linear scale is used in these two figures in order to make them more informative.

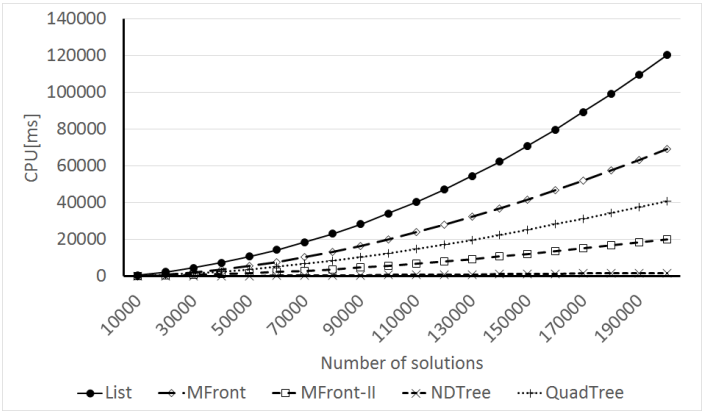

Fig. 14. CPU time (linear scale) vs. the number of points for four-objective convex data sets of quality q3.

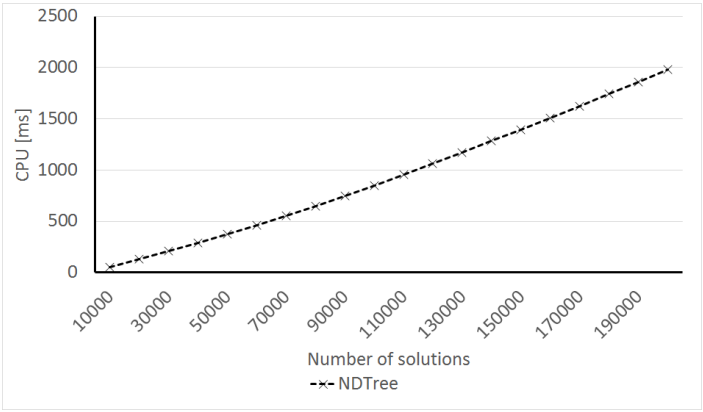

Fig. 15. CPU time (linear scale) vs. the number of points for ND-Tree only (four-objective convex data sets of quality q3).

ND-Tree has two parameters - the maximum size (number of points) of a leaf, and the number of children, so the question arises how sensitive it is to the setting of these parameters. To study it we again use the intermediate data set with $p = 4$ and quality q3 and run ND-Tree with various parameters, see Figure 16. Please note that number of children cannot be larger than the maximum size of a leaf $+1$ since after exceeding the maximum size the leaf is split into the given number of children. We see that ND-Tree is not very sensitive to the two parameters: the CPU time remains between about one and three seconds regardless of the values of the parameters. The best results are obtained with 20 for the maximum size of a leaf and with 6 for the number of children.

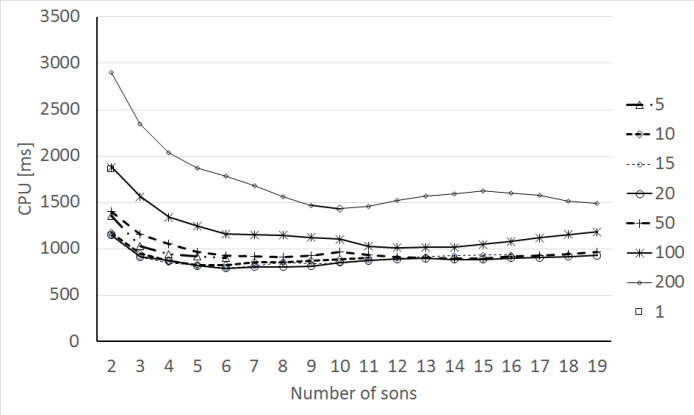

Fig. 16. ND-Tree CPU time for different values of parameters (four-objective convex data sets of quality q3). Data series correspond to maximum leaf size. CPU time in linear scale.

In our opinion the results confirm that ND-Tree performs relatively well for a wide range of the values of the parameters. In fact, it would remain the best method for this data set with any of the parameters settings tested.

B. Sets generated by MOEA/D

In order to test if the observations made for artificial sets hold for sets generated by real evolutionary algorithms, we use sets of points generated by well-known MOEA/D algorithm [2] for multiobjective multidimensional knapsack problem instances with 2 to 6 objectives. We used the code available at http://dces.essex.ac.uk/staff/zhang/webofmoead.htm [2]. We used the instances with 500 items available with the code with 2 to 4 objectives. The profits and weights of these instances were randomly generated uniformly between 1 and 100. We have generated ourselves the 5 and 6 objectives instances by adding random profits and weights, between 1 and 100. MOEA/D was run for at least 100 000 iterations and the first 100 000 points generated by the algorithm were stored for the purpose of this experiment. The numbers of non-dominated points are given in Table III.

Figure 17 presents running times of each of the tested methods as well the running times of MOEA/D excluding the time needed to update the Pareto archive. These results confirm that the observations made for artificial sets also hold in the case of real sets. ND-Tree is the fastest method for three and more objectives. Quad-tree performs particularly badly in the biobjective case. Both versions of M-Front relatively deteriorate with a growing number of objectives. Furthermore, these results show that the time of updating the Pareto archive may be higher than the remaining running time of MOEA/D. In particular, for the six-objective instance the running time of M-Front is 5 times higher than the remaining running time of MOEA/D. The running time of the simple list, Quad-tree and M-Front-II are comparable to the remaining running time of MOEA/D, and only the running time of ND-Tree is 10 times shorter. This confirms that the selection of an appropriate method for updating the Pareto archive may have a crucial influence on the running time of a MOEA.

TABLE III
NUMBERS OF NON-DOMINATED POINTS IN SETS GENERATED BY MOEA/D

p	$\lvert Y_N \rvert$
2	140
3	1789
4	5405
5	10126
6	16074

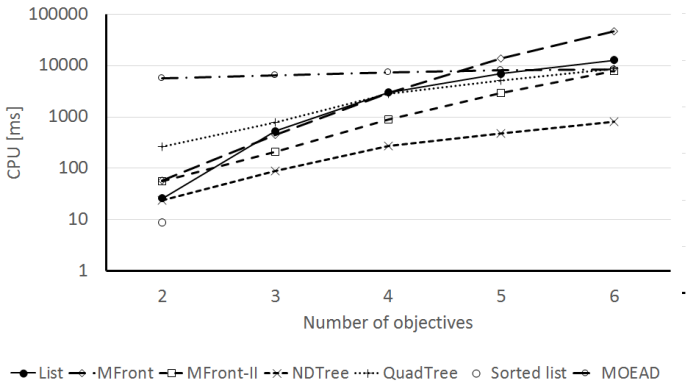

Fig. 17. CPU time (logarithmic scale) on MOEA/D sets.

We have also generated sets of points by applying another MOMH, namely Pareto local search [18] to solve the multiobjective traveling salesman problem (MOTSP). These results can be found at https://sites.google.com/site/ndtreebasedupdate/ (the same conclusions apply).

VI. ND-TREE-BASED NON-DOMINATED SORTING

ND-Tree-based update may also be applied to the problem of the non-dominated sorting. This problem arises in the non-dominated sorting-based MOEAs, e.g. NSGA-III [29] where a population of solutions needs to be assigned to different fronts based on their dominance relationships.

We solve the non-dominated sorting problem in the very straightforward way, i.e. we find the first front by updating an initially empty Pareto archive with each point in the population. Then the points from the first front are removed from the population and the next front is found in the same way. This process is repeated until all fronts are found.

We compare this approach to some recently proposed non-dominated sorting algorithms, i.e. ENS-BS/SS [30] and DDA-NS [31]. ENS-BS/SS algorithm sorts the points lexicographically based on the values of objectives. Then it considers each solution using this order to efficiently find the last front that contains a point dominating the considered point. For each front this method in fact solves the dynamic non-dominance problem with the simple list and if the considered solution is non-dominated within this front it needs to be compared to each solution in this front. If there is just one front in the population, the method boils down to solving the dynamic non-dominance problem with the simple list and requires $\mathcal{O}(N^2)$ comparisons. DDA-NS algorithm sorts the population according to each objective which requires $\mathcal{O}(N \log N)$ objective function comparisons and builds comparison matrices for each

objectives. Then it uses some matrix operations which in general have $\mathcal{O}(N^2)$ complexity to build the fronts. We also compare our algorithm to M-Front-II [19] applied in the same way as ND-Tree-based update. Please note that similarly to what was done in [31] we apply M-Front-II for finding each front, while in [19] only the first front was found by M-Front.

Also, like [30], [31] we used populations of size 5000. We used both random populations drawn with uniform probability from a hypercube, and populations composed of 5000 randomly selected points from our data sets of intermediate quality q3. For each number of objectives 10 populations were drawn.

The results are presented in Figures 18 to 21. We report both CPU times and the number of comparisons of points. We also show the number of fronts on the right y-axis. Please note that we do not show the number of comparisons for DDA-NS, since this algorithm does not explicitly compares points with the dominance relation. The slowest algorithm is DDA-NS. Our results are quite contradictory to the results reported in [31] where DDA-NS is the fastest algorithm in most cases. Please note, however, that in [31] the algorithms were implemented in MATLAB which, as the authors note, is very efficient in matrix operations. On the other hand, the CPU times reported in [31] are of orders of magnitude higher compared to our experiment which suggests rather that the MATLAB implementation of M-Front and ENS-BS/SS is quite inefficient.

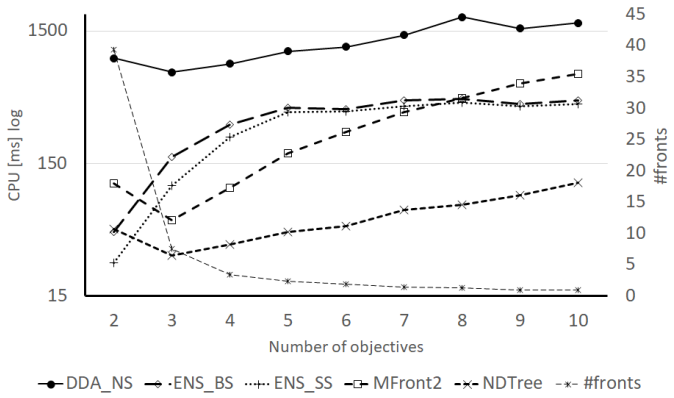

Fig. 18. CPU times of non-dominated sorting algorithms for artificial sets with quality q3.

ENS-BS/SS performs well for populations with many fronts but its performance deteriorates when the number of fronts is reduced. As it was mentioned above if there is just one front in the population, the method boils down to the dynamic non-dominance problem solved with the simple list. This is why in the case of populations drawn from our sets which contain points of higher quality than random populations, and for higher numbers of objectives, where the populations often contain just one front, the number of comparisons saturates at a constant level.

In most cases ND-Tree-based non-dominated sorting is the most efficient method in terms of both CPU time and the number of comparisons. These results are very promising but only preliminary and further experiments, especially with

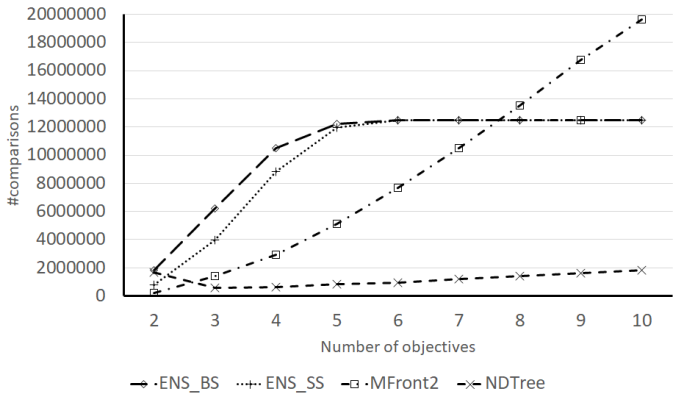

Fig. 19. #comparisons in non-dominated sorting algorithms for artificial sets with quality q3.

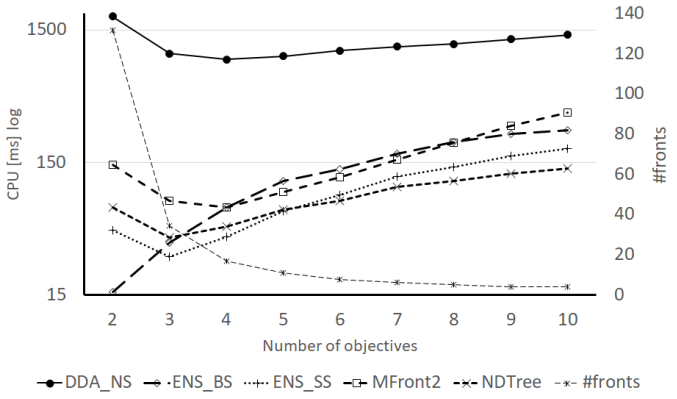

Fig. 20. CPU times of non-dominated sorting algorithms for random populations.

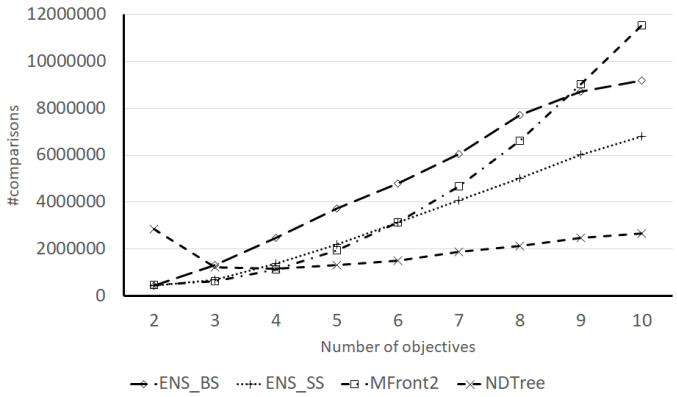

Fig. 21. #comparisons in non-dominated sorting algorithms for random populations.

populations generated by real MOEAs, are necessary. Please note that as suggested in [19] the practical efficiency of the non-dominated sorting in the context of a MOEA may be further improved by maintaining the first front in a Pareto archive.

VII. Conclusions

We have proposed a new method for the dynamic non-dominance problem. According to the theoretical analysis the method remains sub-linear with respect to the number of points in the archive under mild assumptions and the time of processing a single point observed in the computational experiments are almost constant. The results of computational experiments with both artificial data sets of various global shapes, as well as results with sets of points generated by two different multiobjective methods, i.e. MOEA/D evolutionary algorithm and Pareto local search metaheuristic, indicate that the proposed method outperforms competitive methods in the case of three- and more objective problems. In biobjective case the best choice remains the sorted list.

We believe that with the proposed method for updating a Pareto archive, new state-of-the art results could be obtained for many multiobjective problems with more than two objectives. Indeed, results of our computational experiment indicate that the choice of an appropriate method for updating the Pareto archive may have crucial influence on the running time of multiobjective evolutionary algorithms and other metaheuristics especially in the case of higher number of objectives.

Interesting directions for further research are to adapt ND-Tree to be able to deal with archives of a relatively large but bounded size or to solve the static non-dominance problem.

We have also obtained promising results by applying ND-Tree-based update to the non-dominated sorting and comparing it to some state-of-the-art algorithms for this problem. Further experiments are, however, necessary especially with populations generated by real MOEAs. Furthermore, good results of ENS algorithm for populations with many fronts suggest that a combination of ENS with ND-Tree may be an interesting direction for further research.

Acknowledgment

The research of Andrzej Jaszkiewicz was funded by the the Polish National Science Center, grant no. UMO-2013/11/B/ST6/01075.

References

[1] O. Schütze, "A new data structure for the nondominance problem in multi-objective optimization," in *Proceedings of the 2nd International Conference on Evolutionary Multi-criterion Optimization*, ser. EMO'03. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 509–518.

[2] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *Evolutionary Computation, IEEE Transactions on*, vol. 11, no. 6, pp. 712–731, 2007.

[3] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm," Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Zurich, Switzerland, Tech. Rep. 103, May 2001.

[4] C. A. C. Coello, G. T. Pulido, and M. S. Lechuga, "Handling multiple objectives with particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 256–279, June 2004.

[5] S. Agrawal, B. K. Panigrahi, and M. K. Tiwari, "Multiobjective particle swarm algorithm with fuzzy clustering for electrical power dispatch," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 5, pp. 529–541, Oct 2008.

[6] B. Li, K. Tang, J. Li, and X. Yao, "Stochastic ranking algorithm for many-objective optimization based on multiple indicators," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 6, pp. 924–938, Dec 2016.

[7] X. Cai, Y. Li, Z. Fan, and Q. Zhang, "An external archive guided multiobjective evolutionary algorithm based on decomposition for combinatorial optimization," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 4, pp. 508–523, Aug 2015.

[8] S. Yang, M. Li, X. Liu, and J. Zheng, "A grid-based evolutionary algorithm for many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 5, pp. 721–736, Oct 2013.

[9] L. Tang and X. Wang, "A hybrid multiobjective evolutionary algorithm for multiobjective optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 1, pp. 20–45, Feb 2013.

[10] S. Mostaghim and J. Teich, "Quad-trees: A data structure for storing Pareto sets in multiobjective evolutionary algorithms with elitism," in *EMO, Book Series: Advanced Information and Knowledge*, A. Abraham, L. Jain, and R. Goldberg, Eds. Springer-Verlag, 2004, pp. 81–104.

[11] J. Fieldsend, R. Everson, and S. Singh, "Using unconstrained elite archives for multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 3, pp. 305–323, June 2003.

[12] W. Brockhoff, *Theory of Randomized Search Heuristics: Foundations and Recent Developments*. World Sc. Publ. Comp., 2010, ch. Theoretical aspects of evolutionary multiobjective optimization, pp. 101–139.

[13] H. T. Kung, F. Luccio, and F. P. Preparata, "On finding the maxima of a set of vectors," *J. ACM*, vol. 22, no. 4, pp. 469–476, Oct. 1975.

[14] H. N. Gabow, J. L. Bentley, and R. E. Tarjan, "Scaling and related techniques for geometry problems," in *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '84. New York, NY, USA: ACM, 1984, pp. 135–143.

[15] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*. New York, NY, USA: Springer-Verlag New York, Inc., 1985.

[16] P. Gupta, R. Janardan, M. Smid, and B. Dasgupta, "The rectangle enclosure and point-dominance problems revisited," *International Journal of Computational Geometry & Applications*, vol. 7, no. 5, pp. 437–455, 1997.

[17] K. Deb, M. Mohan, and S. Mishra, "Evaluating the ϵ-domination based multi-objective evolutionary algorithm for a quick computation of pareto-optimal solutions," *Evol. Comput.*, vol. 13, no. 4, pp. 501–525, Dec. 2005.

[18] L. Paquete, M. Chiarandini, and T. Stützle, "Pareto local optimum sets in the biobjective traveling salesman problem: an experimental study," in *Metaheuristics for Multiobjective Optimisation*, X. Gandibleux, M. Sevaux, K. Sörensen, and V. T'kindt, Eds. Berlin: Springer. Lecture Notes in Economics and Mathematical Systems Vol. 535, 2004, pp. 177–199.

[19] M. Drozdík, Y. Akimoto, H. Aguirre, and K. Tanaka, "Computational cost reduction of nondominated sorting using the M-front," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 5, pp. 659–678, 2015.

[20] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr 2002.

[21] M. Jensen, "Reducing the run-time complexity of multiobjective EAs: The NSGA-II and other algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 5, pp. 503–515, Oct 2003.

[22] 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*. New York, NY, USA: ACM, 2013, pp. 615–622.

[23] O. Schütze, S. Mostaghim, M. Dellnitz, and J. Teich, "Covering Pareto sets by multi-level evolutionary subdivision techniques," in *Proceedings of the Second Int. Conf. on Evolutionary Multi-Criterion Optimization*. Berlin: Springer, 2003, pp. 118–132.

[24] J. E. Fieldsend, R. M. Everson, and S. Singh, "Using unconstrained elite archives for multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 3, pp. 305–323, 2003.

[25] N. Altwaijry and M. Bachir Menai, "Data structures in multi-objective evolutionary algorithms," *Journal of Computer Science and Technology*, vol. 27, no. 6, pp. 1197–1210, 2012.

[26] S. Mostaghim, J. Teich, and A. Tyagi, "Comparison of data structures for storing Pareto-sets in MOEAs," in *Evolutionary Computation, 2002. CEC '02.*, vol. 1, May 2002, pp. 843–848.

[27] W. Habenicht, *Essays and Surveys on MCDM: Proceedings of the Fifth International Conference on Multiple Criteria Decision Making,*

Mons, Belgium, August 9–13, 1982. Springer, 1983, ch. Quad Trees, a Datastructure for Discrete Vector Optimization Problems, pp. 136–145.

[28] M. Sun and R. Steuer, "Quad trees and linear list for identifying non-dominated criterion vectors," *INFORM Journal on Computing*, vol. 8, no. 4, pp. 367–375, 1996.

[29] 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," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, Aug 2014.

[30] X. Zhang, Y. Tian, R. Cheng, and Y. Jin, "An efficient approach to nondominated sorting for evolutionary multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 2, pp. 201–213, April 2015.

[31] Y. Zhou, Z. Chen, and J. Zhang, "Ranking vectors by means of the dominance degree matrix," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 1, pp. 34–51, Feb 2017.