ND-Tree: a Fast Online Algorithm for Updating the Pareto Archive

... i.e. return to Algorithms and Data structures

Andrzej Jaszkiewicz, Thibaut Lust
"minimize" \( z_1 = f_1(x) \)

... 

"minimize" \( z_J = f_J(x) \)

s.t. \( x \in D \)
Dominance relation

- Dominated
- Non-dominated
- Dominating
- Non-dominating

Diagram illustrating the relationship between dominated and non-dominated points.
(Approximation of) Pareto front
**Definition 1.** Pareto dominance relation: we say that a vector $u = (u_1, \ldots, u_p)$ dominates a vector $v = (v_1, \ldots, v_p)$ if, and only if, $u_k \leq v_k \forall k \in \{1, \ldots, p\} \land \exists k \in \{1, \ldots, p\} : u_k < v_k$. We denote this relation by $u \prec v$.

**Definition 6.** Coverage relation: we say that a point $u$ covers a point $v$ if $u \prec v$ or $u = v$. We denote this relation by $u \preceq v$. We will also use the coverage relation w.r.t. solutions as well, i.e. $x \preceq x^* \iff y(x) \leq y(x^*)$. 
initialize empty Pareto archive

do

  generate a new solution \( x \)
  update Pareto archive with \( x \)

while (…)

return Pareto archive
Updating Pareto archive – dynamic nondominance problem

• Input parameter: new solution $x$
• $x$ is added to Pareto archive if it is not covered by any solution in the archive
• All solutions dominated by $x$ (if any) are removed from the archive
Dominated (rejected) solution
Non-dominated solution
Dominating solution
• The archive is organized as a list of solutions with no specific order
• New solution is compared until a dominating solution is found or all solutions are checked
• Especially poor behavior if new solutions are relatively good (few or none dominating solutions)
In this bi-objective case, the sorted list is as follows:

<table>
<thead>
<tr>
<th>Objective 1</th>
<th>Objective 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>2</td>
<td>31</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>6</td>
<td>28</td>
</tr>
<tr>
<td>8</td>
<td>25</td>
</tr>
<tr>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td>15</td>
<td>23</td>
</tr>
<tr>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>23</td>
<td>18</td>
</tr>
<tr>
<td>25</td>
<td>15</td>
</tr>
<tr>
<td>26</td>
<td>7</td>
</tr>
<tr>
<td>27</td>
<td>5</td>
</tr>
</tbody>
</table>

The pair (14,...) is considered a dominating pair by both objectives.
• An index on each objective is kept
• The indexes are used to define sets $U$ and $L$
• Reference solution is found with k-d tree and approximate nearest neighbor search

• Sets U and L are not built explicitly
• We start with solutions in set U that could dominate new solution
• Dynamic arrays (like `std::vector` in C++) and binary search instead of linked lists (`std::list` in C++) and a hash-table (`std::unordered_map` in C++) for indexes
• Some other methods of course
• Inconclusive results – advanced methods/data structures sometimes worse than simple list
• Rarely used
initialize Pareto archive

do

  for each solution $y$ in Pareto archive

    add to Pareto archive all potentially nondominated solutions in neighborhood of $y$

while at least one new solution was added to Pareto archive

Return Pareto archive
Pareto local search
Pareto local search
Pareto local search
• Standalone PLS starting from random solutions is very inefficient since it spends a lot of time generating large numbers of solutions being still very far from the Pareto front.

• PLS, however, is used as a crucial component in some of the best methods for multiobjective knapsack, biobjective traveling salesperson problem (bTSP) and set covering problem.

• The general idea of such methods is to start PLS from a set of high quality solutions generated by some other methods, e.g. the powerful Lin-Kernighan heuristic for TSP.
The search towards and along Pareto front
The search towards and along Pareto front

- PLS is especially good in the search along Pareto front
- Achieves good synergy when combined with a method very good in search towards Pareto front
• On-line updated Pareto archive is a crucial element of PLS

• PLS generates large number of new candidate solutions in a very short time

• New solutions are usually relatively good, i.e. even if the new solution is dominated there are only relatively few dominating solutions
Main idea of our method
Main idea

\[ z^* \cdot (S) \]

\[ \hat{z}^* (S) \]
\[ z^* \leq S \]

Main idea
Definition 13. ND-Tree data structure is a tree with the following properties:

1) With each node \( n \) is associated a set of solutions \( S(n) \).
2) Each leaf node contains a list \( L(n) \) of solutions and \( S(n) = L(n) \).
3) For each internal node \( n \), \( S(n) \) is the union of sets associated with all sons of \( n \).
4) Each node \( n \) stores an approximate ideal point \( \widehat{z}^*(S(n)) \) and approximate nadir point \( \widehat{z}_*(S(n)) \).
5) If \( n' \) is a son of \( n \), then \( \widehat{z}^*(S(n)) \leq \widehat{z}^*(S(n')) \) and \( \widehat{z}_*(S(n')) \leq \widehat{z}_*(S(n)) \).
Distance measure – Euclidean distance to center point

\[ \hat{z}_*(S) \rightarrow \text{Center point} \]

\[ \hat{z}_*(S) \]
ND-Tree algorithm

- Go through the tree using Properties 1-3 to skip some (many) branches. Stop if the new solution is dominated. Dominated solutions are removed.
- If the new solutions was not dominated add it to the tree. Starting from the root select closest node until leaf node is found.
- If the leaf contains too many solutions split it by simple clustering to subnodes containing close solutions.
Algorithm 2 Update

Parameter $\uparrow$: A Pareto archive $\hat{\mathcal{X}}_E$ organized as ND-Tree
Parameter $\downarrow$: New candidate solution $x$

\begin{align*}
\textbf{if } \hat{\mathcal{X}}_E = \emptyset \textbf{ then} \\
&\text{Create a leaf node } n \text{ with an empty list set } \mathcal{L}(n) \text{ and } \\
&\quad \text{use it as a root} \\
&\quad \mathcal{L}(n) \leftarrow \mathcal{L}(n) + x \\
\textbf{else} \\
&n \leftarrow \text{root node} \\
&\text{UpdateNode}(n \uparrow, x \downarrow) \\
&\textbf{if } x \text{ was not covered by any solution in } \hat{\mathcal{X}}_E \textbf{ then} \\
&\quad \text{Insert}(n \uparrow, x \downarrow)
\end{align*}
Algorithm 3 UpdateNode

Parameter $\uparrow$: A node $n$
Parameter $\downarrow$: New candidate solution $x$

Compare $x$ to $\hat{z}^*(S(n))$ and $\hat{z}^*(S(n))$

if $\hat{z}^*(S(n)) \leq x$ then
  $\quad$ - $|$ Property 1
  $\quad$ STOP: $x$ is rejected
else if $x \leq \hat{z}^*(S(n))$ then
  $\quad$ - $|$ Property 2
  Remove $n$ and its whole sub-tree
else if $\hat{z}^*(S(n)) \leq x$ or $x \leq \hat{z}^*(S(n))$ then
  $\quad$ - $|$ Property 4
  if $n$ is a leaf node then
    for each $y \in \mathcal{L}(n)$ do
      if $y \leq x$ then
        STOP: $x$ is rejected
      else if $x < y$ then
        Remove $y$
    else
      for each Subnode $n'$ of $n$ do
        UpdateNode ($(n', \uparrow, x \downarrow)$
        if $n'$ became empty then
          Remove $n'$
Algorithm 4 Insert

Parameter $\uparrow$: A node $n$
Parameter $\downarrow$: New candidate solution $x$

if $n$ is a leaf node then
    $\mathcal{L}(n) \leftarrow \mathcal{L}(n) + x$
    UpdateIdealNadir ($n \uparrow, x \downarrow$)
    if Size of $\mathcal{L}(n)$ became larger than maximum size of a leaf set then
        Split ($n \uparrow$)
    else
        Find subnode $n'$ of $n$ being closest to $x$
        Insert($n' \uparrow, x \downarrow$)
Algorithm 5 Split

Parameter ♦: A node $n$

Find the solution $y \in \mathcal{L}(n)$ with the highest average Euclidean distance to all other solutions in $\mathcal{L}(n)$
Create a new subnode $n'$ with an empty list set $\mathcal{L}(n')$
$\mathcal{L}(n') \leftarrow \mathcal{L}(n') + y$
UpdateIdealNadir $(n' \uparrow, y \downarrow)$
$\mathcal{L}(n) \leftarrow \mathcal{L}(n) - y$

while The required number of subnodes are not created do
  Find the solution $y \in \mathcal{L}(n)$ with the highest average Euclidean distance to all solutions in all subnodes of $n$
  Create a new subnode $n'$ with an empty list set $\mathcal{L}(n')$
  $\mathcal{L}(n') \leftarrow \mathcal{L}(n') + y$
  UpdateIdealNadir $(n' \uparrow, y \downarrow)$
  $\mathcal{L}(n) \leftarrow \mathcal{L}(n) - y$

while $\mathcal{L}(n)$ is not empty do
  $y \leftarrow$ first solution in $\mathcal{L}(n)$
  Find subnode $n'$ of $n$ being closest to $y$
  $\mathcal{L}(n') \leftarrow \mathcal{L}(n') + y$
  UpdateIdealNadir $(n' \uparrow, y \downarrow)$
  $\mathcal{L}(n) \leftarrow \mathcal{L}(n) - y$
**Algorithm 6** UpdateIdealNadir

Parameter $\uparrow$: A node $n$
Parameter $\downarrow$: New candidate solution $x$

Check in any component of $x$ is lower than corresponding component in $\hat{z}^*(S(n))$ or greater than corresponding component in $\hat{z}_*(S(n))$ and update the points if necessary

if $\hat{z}^*(S(n))$ or $\hat{z}_*(S(n))$ have been changed then

if $n$ is not a root then

    $np \leftarrow$ parent of $n$

    UpdateIdealNadir ($np \uparrow, x \downarrow$)
# Artificial data sets

<table>
<thead>
<tr>
<th>#objectives</th>
<th>Quality</th>
<th>#non-dominated solutions</th>
<th>#objectives</th>
<th>Quality</th>
<th>#non-dominated solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>519</td>
<td>5</td>
<td>1</td>
<td>28944</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>713</td>
<td>5</td>
<td>2</td>
<td>42246</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1046</td>
<td>5</td>
<td>3</td>
<td>77477</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1400</td>
<td>5</td>
<td>4</td>
<td>96002</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>2735</td>
<td>5</td>
<td>5</td>
<td>99975</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4588</td>
<td>6</td>
<td>1</td>
<td>45879</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>6894</td>
<td>6</td>
<td>2</td>
<td>65195</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>12230</td>
<td>6</td>
<td>3</td>
<td>96687</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>19095</td>
<td>6</td>
<td>4</td>
<td>99788</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>53813</td>
<td>6</td>
<td>5</td>
<td>100000</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>14360</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>21680</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>39952</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>64664</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>98283</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2-objective data sets

The graph shows the CPU time in milliseconds (ms) for different quality levels (1 to 5) across various data sets. The x-axis represents the quality level, while the y-axis shows the CPU time in logarithmic scale.

- **List**
- **MFront**
- **MFront-II**
- **NDTree**
- **QuadTree**
- **Sorted List**

The data sets show varying performance across the quality levels, with some showing a steep increase in CPU time as the quality increases, while others remain relatively stable.
3-objective data sets

The graph illustrates the CPU time (in milliseconds) as a function of quality for different algorithms: List, MFront, MFront-II, NDTree, and QuadTree. The x-axis represents the quality level, ranging from 1 to 5, and the y-axis represents the CPU time, ranging from 100 to 100,000 milliseconds.
4-objective data sets
5-objective data sets

![Chart showing CPU time in milliseconds for different quality levels and data sets]

- List
- MFront
- MFront-II
- NDTTree
- QuadTree
6-objective data sets

The graph shows the CPU time (in ms) for different quality levels (1 to 5) for various data structures: List, MFront, MFront-II, NDTTree, and QuadTree. The x-axis represents the quality level, and the y-axis represents the CPU time. Each data structure has a different trend as the quality level increases.
Evolution with the number of objectives
Evolution with the number of solutions

The graph shows the CPU time (in milliseconds) on the y-axis against the number of solutions on the x-axis. The lines represent different algorithms:
- **List**
- **MFront**
- **MFront-II**
- **NDTree**
- **QuadTree**

As the number of solutions increases, the CPU time for all algorithms grows, with **QuadTree** having the lowest CPU time and **List** having the highest.
Evolution with the number of solutions ND-Tree only

CPU [ms]

Number of solutions

- NDTree
Observations

- ND-Tree performs the best 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 solutions.
Observations

• For bi-objective 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
• Simple list obtains its best performances for data sets with many dominated solutions
• Quad-tree performs very bad for data sets with many dominated solutions, e.g. on biobjective instances where it is worst in all cases
• The performance of both M-Front and M-Front-II deteriorates with increasing number of objectives
Sensitivity to parameters

CPU [ms]

Number of sons

- 5
- 10
- 15
- 20
- 50
- 100
- 200
- 1
<table>
<thead>
<tr>
<th>#objectives</th>
<th>Size</th>
<th></th>
<th>NDP</th>
<th></th>
<th>CPU [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>List</td>
<td>ND-Tree</td>
</tr>
<tr>
<td>2</td>
<td>500</td>
<td>32842</td>
<td>24.26</td>
<td>24.82</td>
<td>0.98</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>94343</td>
<td>3517.52</td>
<td>33.26</td>
<td>106</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td>84013</td>
<td>1299.30</td>
<td>18.81</td>
<td>69</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>359581</td>
<td>20130.35</td>
<td>282.14</td>
<td>71</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>49341</td>
<td>257.93</td>
<td>12.09</td>
<td>21</td>
</tr>
</tbody>
</table>
Test sets generated by PLS
Conclusions

• ND-Tree should be a method of choice for storing and updating a Pareto archive in the case of three and more objectives problems
• For bi-objective instances sorter list is the best choice
• Many objective PLS became feasible

• http://arxiv.org/abs/1603.04798