

# Datastructure for Filtering and Storing Non-Dominated Points

MOPGP'2017: Int. Conf. on "Multi-Objective Programming and Goal-Programming"  
October 30-31, 2017 – Université de Lorraine (Metz), France.

---

Dorian DUMEZ, Xavier GANDIBLEUX, Irena RUSU

October 30, 2017

Université de Nantes, UFR Sciences et Techniques  
Laboratoire des sciences du numérique de Nantes UMR CNRS 6004  
France

Supported by

## **vOpt** Research Project

Exact Efficient Solution of Mixed Integer Programming Problems with Multiple Objective Functions (ANR/DFG-14-CE35-0034-01)

Université de Nantes, France – Technische Universität Kaiserslautern, Germany.

# 1. Introduction

# Context

Background:

- $y \in S \subseteq \mathbb{R}^p$
- points of  $S$  are dynamically revealed
- identify  $S_N \subseteq S$ , the set of non-dominated points  $y \in S$

Remark: if  $S$  is static, and contains  $n$  points, then

- this is the problem of finding all maximal points, sometimes called the *maxima set problem* in computational geometry.
- for  $p = \{2, 3\}$ , this problem can be solved in time  $O(n \log n)$  [Kung et al., 1975]

Assumptions:

- $p = 2$
- ranges of components of  $S_N$  are a priori unknown

# Context

Background:

- $y \in S \subseteq \mathbb{R}^p$
- points of  $S$  are dynamically revealed
- identify  $S_N \subseteq S$ , the set of non-dominated points  $y \in S$

Remark: if  $S$  is static, and contains  $n$  points, then

- this is the problem of finding all maximal points, sometimes called the *maxima set problem* in computational geometry.
- for  $p = \{2, 3\}$ , this problem can be solved in time  $O(n \log n)$  [Kung et al., 1975]

Assumptions:

- $p = 2$
- ranges of components of  $S_N$  are a priori unknown

# Context

Background:

- $y \in S \subseteq \mathbb{R}^p$
- points of  $S$  are dynamically revealed
- identify  $S_N \subseteq S$ , the set of non-dominated points  $y \in S$

Remark: if  $S$  is static, and contains  $n$  points, then

- this is the problem of finding all maximal points, sometimes called the *maxima set problem* in computational geometry.
- for  $p = \{2, 3\}$ , this problem can be solved in time  $O(n \log n)$  [Kung et al., 1975]

Assumptions:

- $p = 2$
- ranges of components of  $S_N$  are a priori unknown

# Example

For

$$y = \left\{ \begin{pmatrix} 5 \\ 6 \end{pmatrix}; \begin{pmatrix} 7 \\ 7 \end{pmatrix}; \begin{pmatrix} 2 \\ 4 \end{pmatrix}; \begin{pmatrix} 6 \\ 3 \end{pmatrix}; \begin{pmatrix} 8 \\ 4 \end{pmatrix}; \begin{pmatrix} 3 \\ 7 \end{pmatrix}; \begin{pmatrix} 7 \\ 1 \end{pmatrix} \right\},$$

and the minimization case, the operations are:

iteration	$y$	operations	$S_N$
0	-	-	$\emptyset$
1	$\begin{pmatrix} 5 \\ 6 \end{pmatrix}$	added	$\left\{ \begin{pmatrix} 5 \\ 6 \end{pmatrix} \right\}$
2	$\begin{pmatrix} 7 \\ 7 \end{pmatrix}$	dominated by $\begin{pmatrix} 5 \\ 6 \end{pmatrix} \Rightarrow$ deleted	$\left\{ \begin{pmatrix} 5 \\ 6 \end{pmatrix} \right\}$
3	$\begin{pmatrix} 2 \\ 4 \end{pmatrix}$	added; dominates $\begin{pmatrix} 5 \\ 6 \end{pmatrix} \Rightarrow$ removed	$\left\{ \begin{pmatrix} 2 \\ 4 \end{pmatrix} \right\}$
4	$\begin{pmatrix} 6 \\ 3 \end{pmatrix}$	added	$\left\{ \begin{pmatrix} 2 \\ 4 \end{pmatrix}; \begin{pmatrix} 6 \\ 3 \end{pmatrix} \right\}$
5	$\begin{pmatrix} 8 \\ 4 \end{pmatrix}$	weakly dominated by $\begin{pmatrix} 2 \\ 4 \end{pmatrix} \Rightarrow$ deleted	$\left\{ \begin{pmatrix} 2 \\ 4 \end{pmatrix}; \begin{pmatrix} 6 \\ 3 \end{pmatrix} \right\}$
6	$\begin{pmatrix} 3 \\ 7 \end{pmatrix}$	dominated by $\begin{pmatrix} 2 \\ 4 \end{pmatrix} \Rightarrow$ deleted	$\left\{ \begin{pmatrix} 2 \\ 4 \end{pmatrix}; \begin{pmatrix} 6 \\ 3 \end{pmatrix} \right\}$
7	$\begin{pmatrix} 7 \\ 1 \end{pmatrix}$	added	$\left\{ \begin{pmatrix} 2 \\ 4 \end{pmatrix}; \begin{pmatrix} 6 \\ 3 \end{pmatrix}; \begin{pmatrix} 7 \\ 1 \end{pmatrix} \right\}$

# Motivations

Multiobjective Optimization:

- discrete variables (MOIP/MOCO)
- compute  $Y_N$ , the set of non-dominated points

Algorithms:

- exact:  
such a branch and bound (to handle local sets of points)  
     $\hookrightarrow$  identify  $Y_N$
- approximation:  
such a (meta)heuristics or a local search (to maintain the elite population)  
     $\hookrightarrow$  identify  $Y_{PN}$



# Question investigated

Problematic:

- $S_N = \emptyset$
- **to maintain  $S_N$ :** for each new point  $y$  considered,  
add or not  $y$  in the set, remove one or several points from the set
  - ↳ initialize  $S_N$
  - push  $y$  into  $S_N$
  - is  $y$  ND? in  $S_N$
  - getall  $z_1, z_2$  from  $S_N$

Question:

- which data structure/algorithm to recommend for performing efficiently this problematic?

# Main approaches in the MOO literature

Linear structures:

- List (L)
- Sorted list (SL)

Tree structures:

- Quad-Tree (QT)                      Habenicht, 1982; Sun and Steuer, 1996
- ND-Tree (ND)                                      Jazskiewicz and Lust, 2016
- Balanced Binary Tree (Adelson-Velskii-Landis, AVL)      Dumez, 2016

# Fast analysis

	L	SL	QT	ND	AVL
implementation	easy (immediate)	easy (maintain)	intermediate (maintain)	intermediate (maintain)	intermediate (maintain)
specific operation	none	yes <sup>1</sup>	yes <sup>2</sup>	yes <sup>3</sup>	yes <sup>4</sup>
require a tuning	no	no	no	yes 2 parameters	no
complexity (worst case)	linear	linear	linear	unknown	logarithmic
limit on $p$	no	yes $p = 2$	no	no	yes $p = 2$

1: at the insertion, the list must be maintained sorted

2: at the deletion, the sub-tree has to be re-inserted to the structure

3: at the insertion, splitting points into hypercube cluster

4: balance

Best for  $p = 2$ : **SL** according [Jaszkiewicz and Lust 2016]

# Objectives of this study

- (re)visit the AVL trees for the identified problematic; to the best of our knowledge, no such paper exists in the MCDM literature;
- propose “the AVL for non-dominated points on a dynamic set (AVL\_ND) algorithm” with proofs and theoretical complexity results;
- experiment numerically the structures SL / QT / NDtree / AVL-ND in practice.

## 2. AVL\_ND

stands for

AVL tree for non-dominated points on a dynamic set

# Main phases of the algorithm

- 1 - Search for the position of  $y$  in the tree
- 2 - If  $y$  is not dominated, add it and refresh info at the node
- 3 - During the ascent, look for dominated points
- 4 - Balance all nodes for which we proceeded additions or deletions in their subtrees

# Main phases of the algorithm

- 1 - Search for the position of  $y$  in the tree
- 2 - If  $y$  is not dominated, add it and refresh info at the node
- 3 - During the ascent, look for dominated points
- 4 - Balance all nodes for which we proceeded additions or deletions in their subtrees

# Main phases of the algorithm

- 1 - Search for the position of  $y$  in the tree
- 2 - If  $y$  is not dominated, add it and refresh info at the node
- 3 - During the ascent, look for dominated points
- 4 - Balance all nodes for which we proceeded additions or deletions in their subtrees



# Main phases of the algorithm

- 1 - Search for the position of  $y$  in the tree
- 2 - If  $y$  is not dominated, add it and refresh info at the node
- 3 - During the ascent, look for dominated points
- 4 - Balance all nodes for which we proceeded additions or deletions in their subtrees

When we introduce a point  $y$  in  $S$  we can say that :

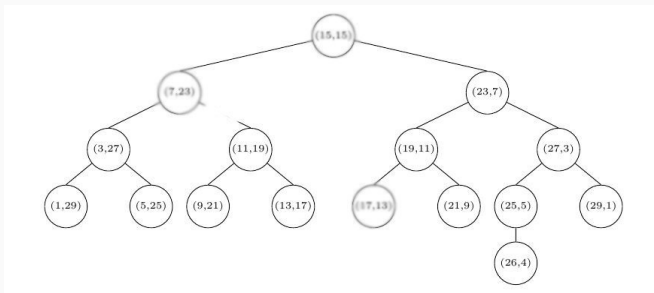
- if  $y$  is dominated then it will be dominated by  $\max_{\text{lex}}\{x \in S_N \mid x \leq_{\text{lex}} y\}$
- if  $y$  dominates points then it will dominates  $\min_{\text{lex}}\{x \in S_N \mid x \geq_{\text{lex}} y\}$
- all points dominated by  $y$  in  $S_N$  are consecutive

## Phases 2 and 3

When we add a point we can quickly look at its father, and brother, to determine whether they are dominated or not.

Furthermore, starting from the node of  $y$  and all left father of it :

- we may be able to delete the whole right subtree
- otherwise we have to dive into it. Then at each level we go in the left subtree or in the right one (and we can delete the left one)



We can balance a node  $a$  in  $O(|balance(a)|)$ .

---

**Algorithm 1:** Balancing algorithm

---

**Function** BALANCE ( $a, f$ ):

**if**  $|balance(a)| > 1$  **then**  
     $\Delta =$  appropriated rotation  
    **return** BALANCE ( $a, \Delta(a, f)$ )

---

Where a rotation returns the node which is now the father of  $a$ .

Let  $n$  be the number of element in our tree. The complexity of each phase is :

1.  $O(\log(n))$  because it a research into a binary search tree
2.  $O(\log(n))$  because it is in  $O(1)$  or we dive into the right subtree
3.  $O(\log(n))$  because we need to dive into a subtree at most once
4.  $O(\log(n))$  because the number of created "unbalancement" is bounded by  $9.\log(n)$

### **3. Numerical experiments**

Numerical instances:

- Dataset 1: random; clusters  
( $\#S=100\ 000$ , 5 values for the parameter “quality of distribution”)
- Dataset 2: random; classic  
( $\#S=100\ 000$ , 5 values for the parameter “quality of distribution”)
- Dataset 3: 2TSP, PLS  
( $\#S=100\ 000$ )
- Dataset 4:  $S$  composed by only (supported) non-dominated points  
( $\#S=100\ 000$  to  $1\ 000\ 000$ , 10 runs where  $S$  is randomly shaken)

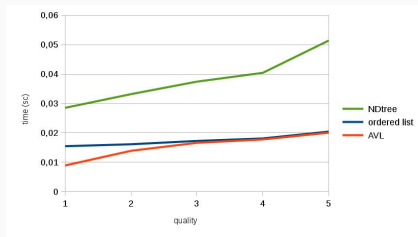
Procedures:

- implemented in C and Julia programming languages,  
Julia version of AVL-ND is integrated to vOptSolver
- C/C++ versions of SL, ND, AVL used here for comparisons needs

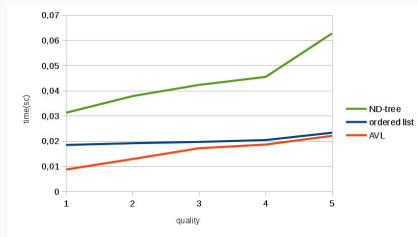
Datasets 1~3, ND-tree: from [Jaszkiewicz and Lust 2016]

# Results

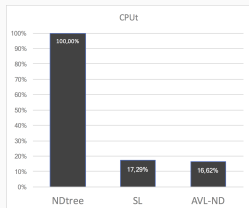
## Dataset 1: (clusters)



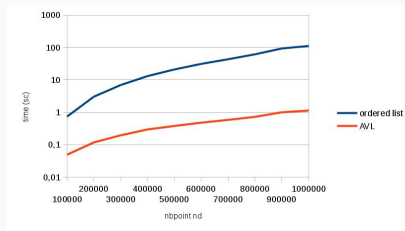
## Dataset 2: (classic)



## Dataset 3: (PLS)

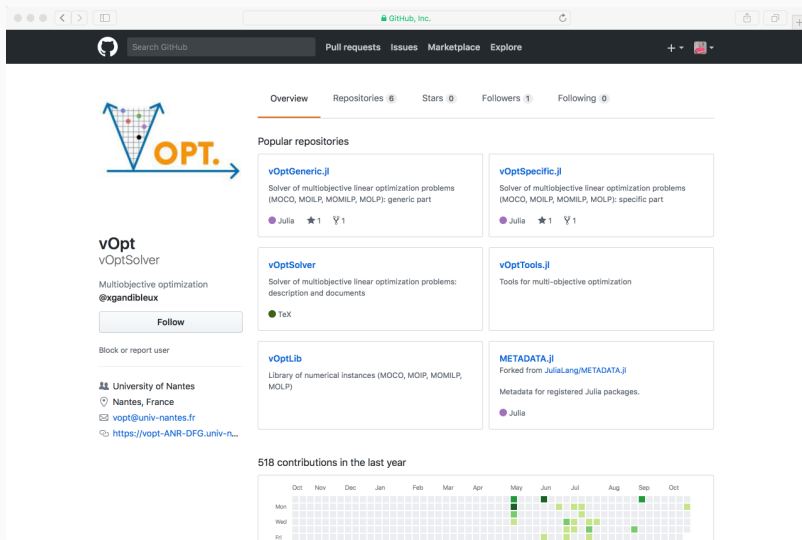


## Dataset 4: ( $S_N$ )





# AVL-ND available within vOptSolver



The screenshot shows the GitHub profile page for vOpt. The header includes the GitHub logo, a search bar, and navigation links for Pull requests, Issues, Marketplace, and Explore. The profile section on the left features the vOpt logo (a blue triangle with a grid of colored dots and an orange arrow pointing right with the text 'OPT.'), the name 'vOpt vOptSolver', the description 'Multiobjective optimization', the username '@xgandibleux', a 'Follow' button, and contact information for the University of Nantes. The 'Popular repositories' section lists four repositories: vOptGeneric.jl, vOptSpecific.jl, vOptSolver, and vOptTools.jl, each with a brief description and language tags. The 'vOptLib' repository is also listed. At the bottom, a contribution graph shows 518 contributions in the last year, with a grid of green squares indicating activity across months and days.

Overview Repositories 6 Stars 0 Followers 1 Following 0

**vOpt**  
vOptSolver  
Multiobjective optimization  
@xgandibleux  
Follow  
Block or report user  
University of Nantes  
Nantes, France  
vopt@univ-nantes.fr  
https://vopt-ANR-DFG.univ-n...

**Popular repositories**

- vOptGeneric.jl**  
Solver of multiobjective linear optimization problems (MOCO, MOILP, MOMILP, MOLP): generic part  
Julia ★ 1 🍴 1
- vOptSpecific.jl**  
Solver of multiobjective linear optimization problems (MOCO, MOILP, MOMILP, MOLP): specific part  
Julia ★ 1 🍴 1
- vOptSolver**  
Solver of multiobjective linear optimization problems: description and documents  
TeX
- vOptTools.jl**  
Tools for multi-objective optimization
- vOptLib**  
Library of numerical instances (MOCO, MOIP, MOMILP, MOLP)
- METADATA.jl**  
Forked from JuliaLang/METADATA.jl  
Metadata for registered Julia packages.  
Julia

518 contributions in the last year

	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct
Mon													
Tue													
Wed													
Thu													

<http://voptsolver.github.io/vOptSolver/>

# Follow/join/contribute to vOptSolver

Homepage of vOptSolver:

<http://voptsolver.github.io/vOptSolver/>

Repository of vOptSolver:

<http://github.com/vOptSolver>

Contact concerning vOptSolver:

[vopt@univ-nantes.fr](mailto:vopt@univ-nantes.fr)

Follow vOptSolver on Twitter:

@vOptSolver

## 4. Discussion

## Conclusion and ongoing works

- A new approach in this context
- Introduction of a new rotation, allowing a quicker balancing of a possibly very unbalanced tree
- Best known theoretical complexity in the worst case

### PROS:

- very good running time
- in practice, the algorithm is not sensitive to the distribution of  $y \in S$ ; no pathologic case of the tree may appear
- not difficult to implement (codes in Julia and C are available)

### CONS:

- up to now, limited to  $p = 2$

### NOW:

- Careful analyse of the existing literature about *Dynamic Algorithms in Computational Geometry*
- to study the case when  $p \geq 3$

# Main references

1. G. Adelson-Velskii and E. Landis: “An algorithm for organization of information”. *Soviet Mathematics doklady*, 6:1259–1263, 1962.
2. H. T. Kung, F. Luccio, and F. P. Preparata: “On finding the maxima of a set of vectors”. *Journal of the ACM*, 22 (4): 469–476, 1975.
3. Walter Habenicht: “Quad trees, a datastructure for discrete vector optimization problems”. In *Lecture Notes in Economics and Mathematical Systems*, volume 209, pages 136–145. Springer Berlin Heidelberg, 1983.
4. Minghe Sun and Ralph E. Steuer: “Quad-trees and linear lists for identifying nondominated criterion vectors”. *INFORMS Journal on Computing*, 8(4):367–375, 1996.
5. Dorian Dumez: “Etude, mise en oeuvre et comparaison d’algorithmes de filtrage et maintien d’ensembles de points non dominés”. Summer internship, Université de Nantes, 2016.
6. Andrzej Jaszkievicz and Thibaut Lust: “Nd-tree: a fast online algorithm for updating a pareto archive and its application in many-objective pareto local search”. *CoRR*, *abs/1603.04798*, 2016.
7. Xavier Gandibleux, Gauthier Soleilhac, Anthony Przybylski, Stefan Ruzika. “vOptSolver: an open source software environment for multiobjective mathematical optimization”. IFORS2017: 21st Conference of the International Federation of Operational Research Societies. July 17-21, 2017. Quebec City (Canada).
8. Homepage of vOptSolver: <http://voptsolver.github.io/vOptSolver/>
9. Dorian Dumez, Xavier Gandibleux, Irena Rusu. “Datastructure for Filtering and Storing Non-Dominated Points”. Technical report, Université de Nantes, 2017.