A Software Library for Archiving Nondominated Points

Duarte M. Dias
CISUC, University of Coimbra
Coimbra, Portugal
duartedias@student.dei.uc.pt

Alexandre D. Jesus
CISUC, University of Coimbra
Coimbra, Portugal
ajesus@dei.uc.pt

Luís Paquete
CISUC, University of Coimbra
Coimbra, Portugal
paquete@dei.uc.pt

ABSTRACT

The need to efficiently maintain nondominated points in an unbounded archive arises in many exact and heuristic approaches to multiobjective combinatorial optimization problems. In this work, we present a C++ library, nondLib, which allows to perform several operations to extract nondominated points from an archive for any number of dimensions. In addition, we discuss the practical impact of the number of nondominated points in the archive in the run-time of these implementations.

CCS CONCEPTS

• **Theory of computation** → *Computational geometry*; **Evolutionary algorithms**; • **Applied computing** → **Multi-criterion optimization and decision-making**.

KEYWORDS

Multiobjective Evolutionary Algorithms, Dimension Sweep, Multidimensional Divide & Conquer

ACM Reference Format:
Duarte M. Dias, Alexandre D. Jesus, and Luís Paquete. 2021. A Software Library for Archiving Nondominated Points. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3449726.3462737

1 INTRODUCTION

Finding a set of nondominated points from a larger point set is a recurrent task that arises in several heuristic approaches such as Pareto Local Search [7] and implicit enumeration algorithms [5] for multiobjective combinatorial optimization problems. Typically, these approaches use an archive of *best* solutions found that is continuously updated during the search process. It is usually required that this set contains only images of solutions in the objective space (as d-dimensional points) that are mutually nondominated. For the case of unbounded archiving strategies, the archive may grow considerably and operations that remove dominated solutions become a bottleneck in terms of running time.

We present a C++ library, nondLib, that performs update operations on an archive to maintain only nondominated points. This library is publicly available at [2] and it can easily be used by programmers with only basic knowledge of C++ and C++ Standard Template Library (STL). Currently, it requires this archive to be passed as a reference to an `std::vector<C>` where type C denotes a point in the objective space implemented as a container of at least size d. Exactly which containers are allowed depends on the exact library function but random access containers are always valid, e.g. `std::vector`, `std::array`, and `std::deque`.

This library allows to perform two operations on an archive of n points: 1) *filter*, which returns the points that are nondominated in the archive, assuming that a certain fraction of it may contain dominated points, and 2) *update*, which returns the points that are nondominated in the archive once a new point is inserted, assuming that the former contains only (mutually) nondominated points. Since maintaining an unbounded archive of nondominated points may lead to large memory requirements, we consider two different implementations of the operations above: 1) *in-place*, in which the operations are performed in-place, and 2) *not-in-place*, in which the nondominated points are copied to another data structure. Although the latter implies memory duplication, it leads to better running times since the removal of dominated points is not performed, and may allow for different use cases for the library.

The algorithms implemented in nondLib are able to deal with any number of objective functions (d). In the case of the *filter* operation, for $d = 2$ and $d = 3$, it uses $O(n \log n)$ dimension-sweep algorithms described in [4], whereas, for $d \geq 4$, it provides two alternatives: 1) a classical $O(dn^2)$-time algorithm that performs pairwise comparisons, and 2) an $O(n \log^{d-2} n)$ multidimensional divide-and-conquer [4]. Note that, these two algorithms may also be used for smaller dimensions, and in very specific cases, the quadratic version may exhibit linear behavior, e.g. when the first point in the list dominates all others, and as such, be preferable to the dimension-sweep implementations. For the *update* operation, the library implements an $O(dn)$-time algorithm for any d.

2 EXPERIMENTAL ANALYSIS

We conducted an experimental analysis of the in-place *filter* operation algorithms implemented in nondLib for $d \in \{2, 3, 4, 6, 8, 10\}$. For $d \geq 4$, we consider both the quadratic-time algorithm (quad) and the multidimensional divide-and-conquer (d&c).

In order to understand the performance of our approaches with respect to n, d and the fraction of nondominated points in the archive (m/n), we generated n d-dimensional points in which a probability p defines whether a given point is dominated or not. First, a point is generated uniformly on the surface of the first quadrant of an unit radius d-dimensional sphere using Muller's method [6]. Then, if $p > m/n$, the point coordinates are multiplied by a value drawn from an uniform distribution in the interval $[0, 1)$. We considered $n = 10^i$, for $i = 3, \ldots, 7$, and $m/n = j/5$ for $j = 1, \ldots, 5$, and generated 30 point sets for each combination of n, m/n and d.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
GECCO '21 Companion, July 10–14, 2021, Lille, France
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8351-6/21/07.
https://doi.org/10.1145/3449726.3462737

Table 1: Average and standard deviation of CPU-time in seconds

n	m/n	$d = 2$	$d = 3$	$d = 4$		$d = 6$		$d = 8$		$d = 10$	
				quad	d&c	quad	d&c	quad	d&c	quad	d&c
10^3	1/5	0.00±0.00	0.00±0.00	**0.00±0.00**	**0.00±0.00**	**0.00±0.00**	0.01±0.00	**0.01±0.00**	**0.01±0.00**	**0.01±0.00**	**0.01±0.00**
	2/5	0.00±0.00	0.00±0.00	**0.00±0.00**	**0.00±0.00**	**0.01±0.00**	**0.01±0.00**	**0.01±0.00**	**0.01±0.00**	**0.01±0.00**	**0.01±0.00**
	3/5	0.00±0.00	0.00±0.00	0.01±0.00	**0.00±0.00**	**0.01±0.00**	**0.01±0.00**	**0.01±0.00**	**0.01±0.00**	**0.01±0.00**	**0.01±0.00**
	4/5	0.00±0.00	0.00±0.00	0.01±0.00	**0.00±0.00**	**0.01±0.00**	**0.01±0.00**	**0.01±0.00**	0.02±0.00	**0.01±0.00**	0.02±0.00
	5/5	0.00±0.00	0.00±0.00	0.02±0.00	**0.00±0.00**	0.02±0.00	**0.01±0.00**	**0.02±0.00**	**0.02±0.00**	**0.02±0.00**	**0.02±0.00**
10^4	1/5	0.00±0.00	0.00±0.00	0.06±0.00	**0.01±0.00**	0.10±0.00	**0.05±0.00**	0.15±0.00	**0.10±0.00**	0.21±0.00	**0.13±0.00**
	2/5	0.00±0.00	0.00±0.00	0.16±0.01	**0.02±0.00**	0.21±0.00	**0.07±0.00**	0.27±0.01	**0.13±0.00**	0.33±0.00	**0.17±0.00**
	3/5	0.00±0.00	0.00±0.00	0.33±0.01	**0.02±0.00**	0.37±0.00	**0.09±0.01**	0.44±0.01	**0.17±0.00**	0.50±0.00	**0.21±0.00**
	4/5	0.00±0.00	0.01±0.00	0.56±0.01	**0.02±0.00**	0.61±0.01	**0.16±0.01**	0.67±0.01	**0.22±0.00**	0.72±0.00	**0.26±0.01**
	5/5	0.00±0.00	0.01±0.00	0.86±0.02	**0.02±0.00**	0.90±0.01	**0.37±0.01**	0.95±0.01	**0.27±0.00**	0.99±0.00	**0.31±0.01**
10^5	1/5	0.02±0.00	0.03±0.00	4.76±0.07	**0.16±0.01**	8.84±0.39	**0.76±0.01**	17.26±1.64	**2.05±0.01**	33.91±0.00	**3.22±0.02**
	2/5	0.02±0.00	0.04±0.00	20.09±0.72	**0.19±0.01**	28.91±3.02	**1.15±0.01**	50.01±4.99	**3.09±0.04**	81.29±0.00	**4.59±0.02**
	3/5	0.02±0.00	0.04±0.00	65.54±2.67	**0.21±0.01**	85.34±6.07	**1.60±0.01**	124.13±5.82	**4.33±0.02**	157.63±0.00	**6.24±0.02**
	4/5	0.02±0.00	0.05±0.00	149.37±3.62	**0.24±0.01**	186.28±9.99	**2.07±0.02**	222.58±13.72	**5.70±0.02**	246.99±0.00	**8.11±0.04**
	5/5	0.02±0.00	0.05±0.00	271.76±5.23	**0.26±0.01**	305.82±9.80	**2.55±0.05**	339.68±8.64	**7.04±0.02**	364.07±0.00	**9.91±0.04**
10^6	1/5	0.86±0.07	0.95±0.06	-	**3.19±0.10**	-	**12.31±0.14**	-	**41.27±0.27**	-	**78.83±0.28**
	2/5	0.80±0.06	0.94±0.06	-	**3.44±0.13**	-	**20.14±0.13**	-	**69.65±0.73**	-	**125.16±0.51**
	3/5	0.84±0.04	1.02±0.06	-	**3.69±0.05**	-	**29.03±0.16**	-	**101.68±0.40**	-	**180.67±0.75**
	4/5	0.79±0.04	1.02±0.05	-	**4.15±0.09**	-	**38.52±0.18**	-	**137.15±0.67**	-	**242.87±0.97**
	5/5	0.67±0.09	0.94±0.08	-	**4.34±0.17**	-	**47.61±0.72**	-	**173.76±0.85**	-	**310.14±1.90**
10^7	1/5	14.26±2.47	14.51±2.98	-	**49.63±1.17**	-	**201.88±3.96**	-	-	-	-
	2/5	13.93±3.26	15.50±3.55	-	**55.00±2.08**	-	**359.94±4.38**	-	-	-	-
	3/5	11.95±3.45	14.74±3.27	-	**59.89±0.79**	-	**539.32±8.35**	-	-	-	-
	4/5	11.70±3.39	14.26±3.03	-	**67.71±0.69**	-	**718.65±10.30**	-	-	-	-
	5/5	10.05±3.52	14.42±3.18	-	**67.46±6.40**	-	-	-	-	-	-

The experiments were conducted on a computer cluster with two Intel Xeon Silver 4210R 2.4G processors with 10 cores / 20 threads and 13.75M cache, two 32GB RDIMM, operating system Debian GNU/Linux 10 (buster). The library was compiled with GNU g++ version 8.3.0 under the C++14 standard and using optimization flag O3. We defined a cut-off time of 720 seconds for each run.

Table 1 presents the average and standard deviation of the CPU-times in seconds taken by our approaches for different values of n, d and fraction of nondominated points (column m/n). The symbol "-" means that the algorithm was not able to terminate within the cut-off limit. The boldface indicates which of the two approaches for $d \geq 4$ performed better in terms of average running-time. Note that for $d = 2$ and $d = 3$ the dimension-sweep algorithms take less than 0.1 seconds for $n = 10^5$ and less than 20 seconds for $n = 10^7$. In addition, they are not strongly affected by the ratio m/n. Differently, for the case of $d \geq 4$, both quadratic-time and multidimensional divide-and-conquer algorithms require more time as the ratio m/n increases. This effect is stronger for the quadratic-time algorithm, which is not able to solve instances for $n \geq 10^6$ and $d \geq 4$. The multidimensional divide-and-conquer is always faster than the quadratic time for $n \geq 10^4$. For $n = 10^3$, both present comparable run-times, which may be due to the initial setup overhead of the multidimensional divide-and-conquer. Preliminary experiments for $d = 20$ indicate that the multidimensional divide-and-conquer is still faster than the quadratic-time algorithm, except for small n and small m/n, although with small difference.

3 FURTHER WORK

Further work consists of considering the case in which the archive is already (lexicographically) sorted, which should give further improvements in the running-time. We also plan to integrate the data structures mentioned in [3] for the update operation as well as the operation of returning the nondominated points from the union of several archives, each of which containing only mutually nondominated points, which typically arises on dynamic programming algorithms for multiobjective combinatorial optimization problems [1, 5]. Finally, it would also be interesting to further generalize the library to accept other containers.

ACKNOWLEDGMENTS

D.M. Dias acknowledges a research fellowship from CISUC, University of Coimbra. This work was funded by national funds through the FCT - Foundation for Science and Technology, I.P., within the scope of the project CISUC - UID/CEC/00326/2020 and by European Social Fund, through the Regional Operational Program Centro 2020.

REFERENCES

[1] R. Beier and B. Vöcking. 2011. The Knapsack Problem. In *Algorithms Unplugged*, B. Vöcking et al. H. Alt, M. Dietzfelbinger, R. Reischuk, C. Scheideler, H. Vollmer, and D. Wagner (Eds.). Springer, 375–381.

[2] D.M. Dias, A.D. Jesus, and L. Paquete. 2021. nondLib (Version v0.2.0). http://doi.org/10.5281/zenodo.4733027.

[3] A. Jaszkiewicz and T. Lust. 2018. ND-Tree-Based Update: A Fast Algorithm for the Dynamic Nondominance Problem. *IEEE Trans. Evol. Comput.* 22, 5 (2018), 778–791.

[4] H.T. Kung, F. Luccio, and F.P. Preparata. 1975. On Finding the Maxima of a Set of Vectors. *J. ACM* 22, 4 (1975), 469–476.

[5] A. Liefooghe, L. Paquete, and J.R. Figueira. 2013. On Local Search for Bi-objective Knapsack Problems. *Evol. Comput.* 21, 1 (2013), 179–196.

[6] M.E. Muller. 1959. A Note on a Method for Generating Points Uniformly on N-Dimensional Spheres. *Commun. ACM* 2 (1959), 19–20.

[7] L. Paquete, T. Schiavinotto, and T. Stützle. 2007. On Local Optima in Multiobjective Combinatorial Optimization Problems. *Ann. Oper. Res.* 156, 1 (2007), 83–97.