Framing Algorithms for Approximate Multicriteria Shortest Paths

Nicolas Hanusse, David Ilcinkas, Antonin Lentz

Framing Algorithms for Approximate Multicriteria Shortest Paths

Nicolas Hanusse
LaBRI, CNRS & Université de Bordeaux, France

David Ilcinkas
LaBRI, CNRS & Université de Bordeaux, France

Antonin Lentz
LaBRI, Université de Bordeaux, France

Abstract

This paper deals with the computation of d-dimensional multicriteria shortest paths. In a weighted graph with arc weights represented by vectors, the cost of a path is the vector sum of the weights of its arcs. For a given pair consisting of a source s and a destination t, a path P dominates a path Q if and only if P's cost is component-wise smaller than or equal to Q's cost. The set of Pareto paths, or Pareto set, from s to t is the set of paths that are not dominated. The computation time of the Pareto paths can be prohibitive whenever the set of Pareto paths is large.

We propose in this article new algorithms to compute *approximated Pareto paths* in any dimension. For $d = 2$, we exhibit the first approximation algorithm, called Frame, whose output is guaranteed to be always a subset of the Pareto set. Finally, we provide a small experimental study in order to confirm the relevance of our Frame algorithm.

2012 ACM Subject Classification Theory of computation → Shortest paths; Applied computing → Multi-criterion optimization and decision-making

Keywords and phrases Pareto set, multicriteria, shortest paths, approximation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.11

Funding This study has been supported by the ANR project DESCARTES (ANR-16-CE40-0023).

1 Introduction

1.1 Context and Motivation

Computing a shortest path is a classical problem and it has been widely studied for one criterion. However, in a transportation network for example, one is often interested in finding a path minimizing several criteria like the duration, the financial cost, or the physical effort. The list of potentially interesting criteria gets even larger with the development of multimodal and public transportation systems, when a traveler can walk, take a taxi, a plane, a train within a journey. For instance, the number of connections [6] matters especially when time tables are uncertain. Even time might have different facets: in temporal graphs, it is different to minimize arrival time and traveling time [8, 25]. More generally, people want to get personalized answers taking simultaneously into account several criteria, that is handling several cost functions. For a given path of cost $(c_1, c_2, \ldots, c_d)$, the first natural approach consists in computing a linear combination of the costs, that is $\sum_{1 \leq i \leq d} \alpha_i c_i$, for some coefficients α_i. Then any algorithm dedicated to shortest path computation for one criterion can be used. This approach has several drawbacks: *how to set up the α_i's? Does such a formula have a semantic meaning?*

A first immediate property drops whenever a multiple cost function is considered: the "smallest" cost is no more unique. Taking a helicopter to reach a destination is much quicker than walking but it is also much more expensive! We can also think of other paths with

other transportation vehicles that are all incomparable for the two criteria time and price. A set of paths representing all incomparable "best" costs is called a *set of Pareto paths*[1] and reflects the variety of smallest costs. A Pareto set can be exponentially large even for bounded degree graphs and two criteria [10]. As a consequence, the computation may take a lot of time and require an significant amount of space. Besides, in practical settings, users do not want to get thousands of propositions. To reduce the size of a Pareto set, the notion of $(1+\varepsilon)$-*Pareto set*[1] has been proposed and proved to always exist even with the constraint of having a polynomial size in n [18].

Dijkstra-based algorithms for multiple criteria, called in this paper MC Dijkstra, also called *Multicriteria Label Setting* (MLS)[2], have been proposed in order to compute exact Pareto sets for two [10] or more dimensions [15]. Whenever the criteria are correlated and the distance between the source and the destination is small, Pareto sets tend to be small. For instance, for $10K$ vertices, using as criteria time and distance, the existing solutions are practical.

However, these algorithms are not scalable in practice: without any preprocessing, it takes a few seconds to solve a query in a network of 18 millions vertices modeling western Europe [2] for queries with only one criterion. Informally, even for a city like Prague with 65K nodes, for a given pair of source and destination, an exact Pareto set often contains thousands of paths for three criteria [11] and its computation may take around 10 minutes. Since a query can require to store all the incomparable paths for one source, the amount of memory can be a thousand times larger than the storage of the graph itself.

In order to compute queries on large graphs and to limit the number of optimal paths proposed to users, the approximation of Pareto sets is promising. The main difficulty is that, even if the output may be quite small, the existing algorithms require a large working memory, and very little is known about the time and the memory of computing $(1+\varepsilon)$-approximations of Pareto sets. To speed up the queries for one criterion, preprocessing algorithms are presented in the survey [2] but it is not obvious that all of these techniques can be efficient for multicriteria queries.

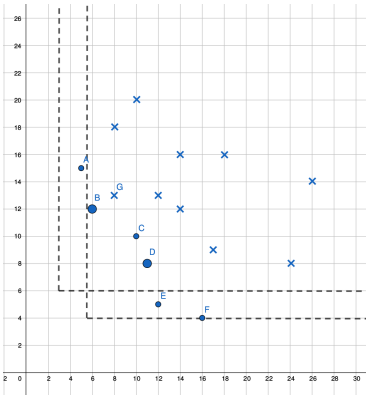
1.2 Problem Description and State of the Art

1.2.1 Exact and Approximated Pareto Sets

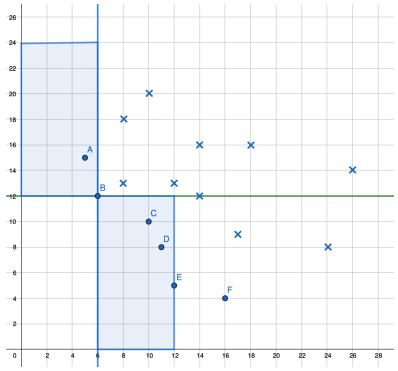
The input of our problem is a weighted directed graph $G = (V, A)$ of n vertices and m arcs defined on d criteria, and a source vertex s. The graph may contain multiple arcs and loops. The weight $w(a)$ of an arc a is a d-dimensional vector whose values belong to the range $\{0\} \cup [1, C]$, the components of the arc weights being normalized and bounded by some common value C. The cost $c(P) = (P_1, \ldots, P_d)$ of a k-hop path $P = a_1, \ldots, a_k$ is the vector sum $\sum_{1 \leq i \leq k} w(a_i)$.

A path P *dominates* a path P' if $P_i \leq P'_i$ for every $i \in \{1, \ldots, d\}$. A Pareto set of a set $\mathcal{T}$ of paths is a set of incomparable[3] paths from $\mathcal{T}$, that are not dominated by any other path from $\mathcal{T}$ with a different cost, and which is maximal by inclusion. In particular, if several paths of $\mathcal{T}$ have the same cost, then at most one is kept in a Pareto set of $\mathcal{T}$. Notice that if $\mathcal{S}$ is a Pareto set of some set $\mathcal{T}$, then the Pareto set of $\mathcal{S}$ is $\mathcal{S}$ itself. The *Multicriteria*

[1] A formal definition will be given in Section 1.2.1.
[2] The letters M and S may also stand for "Multiobjective" and "Scheme" respectively.
[3] w.r.t. dominance



(a) $\{B, D\}$ is a 2-Pareto set.



(b) Regions containing the incomparable paths 2-covering B.

Figure 1 Pareto sets and Covering.

Shortest Path problem consists in finding, for each vertex $v \in V$, a Pareto set $\mathcal{S}_v$ of the set of all paths from s to v. We use the notations $S_v = |\mathcal{S}_v|$ and $S = \sum_{v \in V} S_v$. The values S_v and S do not depend on the actual choices of the sets $\mathcal{S}_v$, since these values derive from the size of the unique Pareto set of the path costs.

A path P $(1+\varepsilon)$-*covers* a path P' if $P_i \leq (1+\varepsilon)P'_i$ for every $i \in \{1, \ldots, d\}$. A $(1+\varepsilon)$-Pareto set of a set $\mathcal{T}$ is a set $\mathcal{S}_\varepsilon$ of incomparable paths from $\mathcal{T}$, such that any path of $\mathcal{T}$ is $(1+\varepsilon)$-covered by a path in $\mathcal{S}_\varepsilon$. In particular, a 1-Pareto set is a Pareto set and vice versa. Then, the $(1+\varepsilon)$-*approximated Multicriteria Shortest Path problem* consists in finding, for each vertex $v \in V$, a $(1+\varepsilon)$-Pareto set $\mathcal{S}_{v,\varepsilon}$ of the set of all paths from s to v.

A solution $(\mathcal{S}_{v,\varepsilon})_{v \in V}$ to the $(1+\varepsilon)$-approximated Multicriteria Shortest Path problem is said to be *Pareto compatible* if and only if $\mathcal{S}_{v,\varepsilon}$ is a subset of a Pareto set $\mathcal{S}_v$, for every vertex v. This property is useful since it guarantees that the size S_ε of the output of an approximation algorithm is always at most S. In Fig. 1a, $\mathcal{S} = \{A, B, C, D, E, F\}$ is a Pareto set of all the paths, whereas $\{B, D\}$ is a 2-Pareto set. The two quadrants bounded by the dashed lines represent the areas 2-covered by B and D. Note that there may be various $(1+\varepsilon)$-Pareto sets when $\varepsilon > 0$. For example the set $\{G, D\}$ is also a 2-Pareto set even though $G \notin \mathcal{S}$.

To solve the Multicriteria Shortest Paths problem, Hansen [10] proposes a generalization of Dijkstra's algorithm with two criteria. This algorithm has then been generalized to any number of criteria in [15]. The bicriteria algorithm proposed by Hansen operates in $O(mnC \log(nC))$ time. In [3], it is proved that the standard MC Dijkstra for the one-to-all query in dimension d has time complexity $O(nS^2)$ and uses $O(nS)$ space when there are no multiple arcs. Although S can reach $\Theta(n(nC)^{d-1})$, it is very unlikely in practice to get such a size.

It is also interesting to observe that exact Pareto sets are not always large in practice, especially if the criteria are correlated. In [17], Pareto sets sizes are often smaller than 100 for real graphs and synthetic graphs with a random weight assignment. However, when the number of criteria grows and some are negatively correlated, Pareto set sizes can be unpractical. Some examples can be found in [1]. An experimental comparison of methods are presented in [19] on grids and road networks up to 300K nodes. It does not exhibit which algorithm is the best in practice for exact Pareto sets.

Papadimitriou and Yannakakis show that for any multiobjective optimization problem, there exists a $(1+\varepsilon)$-Pareto set $(\mathcal{S}_{v,\varepsilon})_{v\in V}$ of polynomial size in n even if C is exponential in n. In our context, they show that $S_{v,\varepsilon}$ can be in $O\left(\left(\frac{\log(nC)}{\varepsilon}\right)^{d-1}\right)$. It means that the output can be quite small but the difficulty is still to limit the time and the memory space during the computation. For $d=2$, Hansen [10] proposes a solution applying m times MC Dijkstra on the initial graph. In a similar fashion, Warburton [24] gives an algorithm for any d, calling an exact algorithm several times. This algorithm could require less MC Dijkstra iterations than Hansen's, but this number is still claimed in [4] to be too huge in order to be competitive in practice. Wang et al. develop in [23] a new algorithm called α-Dijkstra, pruning path with a variable severity, depending on the number of best paths kept at a certain stage of the algorithm. This algorithm is limited to $d=2$. Tsaggouris and Zariolagis [21] propose a Bellman-Ford-based algorithm TZ operating in $O\left(nm\left(\frac{n\log(nC)}{\varepsilon}\right)^{d-1}\right)$ time. Inspired from TZ, Breugem *et al.*[3] proposed a Dijkstra-based algorithm, called Hydrid, running in $O\left(n^3\left(\frac{n\log(nC)}{\varepsilon}\right)^{2d-2}\right)$ time. They made an experimental comparison between the two approximated Pareto sets computations and the standard MC Dijkstra. The new hybrid algorithm is efficient and sometimes outperforms MC Dijkstra whenever Pareto sets are very large. It is also interesting to notice that TZ does not prune a lot of explored paths. It means that it can be much worse than MC Dijkstra for small Pareto sets. An attempt to unify Djikstra and Bellman-Ford-based algorithms is addressed by Bökler et al. in [4], containing TZ, Hydrid and new variants.

If we allow a light preprocessing, NAMOA* [13, 14] is a generalization of the well-known A* search algorithm to the multicriteria setting, meaning that it is dedicated to one-to-one requests. The difficulty here is the estimation of a guaranteed lower bound $h(v)$ for the d dimensions. For large and real graphs, the computation time of the algorithms with guarantee can be too long. Some heuristics have been proposed and speed up drastically the computation time [11] but without any guarantee.

Other attempts have been done to summarize Pareto sets [1, 20]. A *linear path skyline*, defined as a subset of conventional Pareto sets, is a set of paths optimal under a linear combination of their cost values. Multicriteria being especially relevant in a multimodal setting, a different approximation definition has been proposed in [5]. This paper proposes to summarize a Pareto set by the paths such that their projection on two specific criteria (arrival time and number of trips) are additively not far from an optimal one.

1.3 Contributions

In this article, we propose two algorithms, called Sector and Frame, computing guaranteed $(1+\varepsilon)$-Pareto sets $\mathcal{S}_\varepsilon = \bigcup_{v\in V}\mathcal{S}_{v,\varepsilon}$. Frame is a variant of Sector optimized in dimension 2. It guarantees the Pareto compatibility property and thus outputs a set which cannot be larger than the exact Pareto set size, while having a worst case time complexity lower than or equal to MC Dijkstra's one.

In Table 1, we focus on the *one-to-all* query in simple graphs, and the computation time is expressed in the output sensitive complexity, Δ being the maximal degree.

In approximation algorithms, S_ε denotes the size of the output, which is a $(1+\varepsilon)$-Pareto set. It can be much larger than S^*_ε, the minimum cardinality of a $(1+\varepsilon)$-Pareto set. However, starting from $\mathcal{S}_\varepsilon$, a linear time algorithm can output $\mathcal{S}'_\varepsilon \subseteq \mathcal{S}_\varepsilon$ such that $S'_\varepsilon = O(S^*_\varepsilon)$.

Table 1 Our results.

	Output sensitive complexity $O(\cdot)$	Pareto compatible	Ref.
MC Dijkstra	ΔS^2	✓	[10, 3]
Sector	$\Delta S_\varepsilon \log^{d-1}(\Delta S_\varepsilon)$		Theorem 8
TZ	$n\Delta S_\varepsilon$		[21]
Hydrid	?		[3]
MC Dijkstra ($d = 2, 3$)	$\Delta S \log(\Delta S)$	✓	Proposition 1
Frame ($d = 2$)	$\Delta S_\varepsilon \log(\Delta S_\varepsilon)$	✓	Theorem 18
Hydrid ($d = 2$)	$nS_\varepsilon^2 \leq n^2 S^3$		[3]

Since Frame is Pareto compatible, we have $S_\varepsilon \leq S$ for that algorithm. Thus we can hope that its computation time is in practice significantly smaller than the one of the best MC Dijkstra algorithm in 2D. Hybrid [3] and TZ [21] are not Pareto compatible. However, for $d = 2$, $S_\varepsilon(\text{Hydrid}) \leq nS$. More generally, for $d \geq 3$, it is a priori impossible to claim what is the smallest output among $\mathcal{S}_\varepsilon(\text{Sector})$, $\mathcal{S}_\varepsilon(\text{Hydrid})$, $\mathcal{S}_\varepsilon(\text{TZ})$ and $\mathcal{S}(\text{MC Dijkstra})$.

For integer arc weights, the output size S_ε of Sector is in $O\left(d(nC)^{d-1}\log_{1+\varepsilon}(nC)\right)$. We can observe that whenever C is moderate, Sector provides smaller upper bounds on the time complexity than TZ. To make a simple comparison with $\Delta = \Theta(1)$, if $C \leq \left(\frac{n}{d^2\varepsilon^{d-2}}\right)^{\frac{1}{d}}$ then Sector has a smaller known upper bound on its time complexity than TZ. For instance, if $d = 2$, it is the case if $C = O(\sqrt{n})$. Furthermore, if $C = \Theta(1)$, then TZ upper bound is $\Omega(n^2)$ times Sector's one.

2 Preliminaries

Notations and Remarks

A *path* is a sequence of arcs $a_1, \ldots, a_k$ such that, for all $1 \leq i < k$, the destination of a_i is the source of a_{i+1}. The *source* of a path $P = a_1, \ldots, a_k$ is the source of a_1 and its *destination* is that of a_k. In this paper, all paths have the same source s. Notice that if $P = a_1, \ldots, a_k$ is a path and a_{k+1} is an arc whose source is the destination of a_k, notation $P \cdot a_{k+1}$ stands for the path $a_1, \ldots, a_k, a_{k+1}$, defined by the extension of P by a_{k+1}.

For a path P of cost $c(P) = (P_1, P_2, \ldots, P_d)$, its *rank* is defined as $\texttt{rank}(P) = \sum_{1 \leq i \leq d} P_i$. For legibility reasons, each arc rank is strictly positive in our algorithms descriptions.

Let $P = a_1, \ldots, a_k$ and $P' = a'_1, \ldots a'_{k'}$ be two paths sharing the same source and destination. If $\texttt{rank}(P) > \texttt{rank}(P')$ then P cannot dominate P'. Depending on $\varepsilon > 0$, P could however $(1+\varepsilon)$-cover P'. In Fig. 1a, $\texttt{rank}(G) = 21$ and $\texttt{rank}(B) = 18$ but G 2-covers B.

Pareto Set Computation

Depending on the context, maximal or minimal vectors, Pareto sets (mathematics) or Skylines (data-mining) are different names of the same notion. In the *offline* setting, the whole set of n points on which we want to compute a Pareto set is given at the beginning. If S is the Pareto set size, the computation can be done in $O(nS)$ time and can drop to $O(n \log n)$ for $d = 2$ and $O(n \log^{d-2} n)$ for $d > 2$ [12]. However, these methods cannot be used in an *online* setting, i.e., if the points are processed one by one. As explained later in Section 3.1, it means that these methods are not relevant for MC Dijkstra.

Domination and Covering Checking

Checking if a given point P is $(1+\varepsilon)$-covered by a point in a set $\mathcal{S}$, not necessarily being a Pareto set, can be done using *range queries* in dimension d.

Given a cartesian product of intervals $\mathcal{I} = [x_1, x_1'] \times [x_2, x_2'] \times \ldots \times [x_d, x_d']$ and a point set $\mathcal{S}$, $\texttt{RangeQuery}(\mathcal{I}, \mathcal{S})$ reports every point Q in $\mathcal{S} \cap \mathcal{I}$. We use such queries to test $(1+\varepsilon)$-coverings or finer properties. Note that in our case, we do not require to report every point in the subspace specified by the intervals but just to learn if there is at least one point. A point set $\mathcal{S}$ of n points can be preprocessed in $O(n \log^{d-1} n)$ time so that any range query and thus any $(1+\varepsilon)$-covering (or similar) checking can be done in $O\left(\left(\frac{\log n}{\log \log n}\right)^{d-1}\right)$ time [16].

3 General Algorithms

3.1 Reminder on MC Dijkstra

MC Dijkstra overview

The MC Dijkstra algorithm follows Dijkstra's one, adapted to the case of multiple criteria. In that case, the goal is to obtain a Pareto set from s to v for each vertex v. For this reason, the algorithm maintains a set $\mathcal{T}$ of paths rather than vertices. This set is initialized with the empty path from s to s. Also, for each vertex v, the algorithm maintains a candidate Pareto set $\mathcal{S}_v$, initialized to the empty set.

Similarly as in the single-criterion case, MC Dijkstra selects at each step the minimum of $\mathcal{T}$. More precisely, MC Dijkstra selects the path P in $\mathcal{T}$ which has the lexicographically minimum cost. If v is the destination of P, then P is added to the set $\mathcal{S}_v$. Again similarly, all paths P' which consist of P plus one arc from the destination of P are considered. Let w be the destination of P'. If P' is dominated by a path in $\mathcal{S}_w$ or by a path in $\mathcal{T}$ with the same destination, P' is discarded. Otherwise, P' is added to $\mathcal{T}$, and any path $P'' \in \mathcal{T}$ with the same destination as P' which is dominated by it is removed from $\mathcal{T}$.

The algorithm terminates when $\mathcal{T}$ is empty at the end of a step. At that time, the sets $\mathcal{S}_v$ contain Pareto sets from s to every vertex v. The following proposition is more or less an agglomeration of existing results, with small adjustments in order to obtain a consistent statement.

MC Dijkstra pseudo code

A more formal description of MC Dijkstra is given in Algorithm 1. In this algorithm, we use the following two functions:

- $\texttt{IsNotDominated}(P, \mathcal{S})$ takes a path P and a Pareto set $\mathcal{S}$ as input. It returns `True` if the path P is not dominated by any path in $\mathcal{S}$, and `False` otherwise.
- $\texttt{InsertAndClean}(P, \mathcal{S})$ takes a path P and a Pareto set $\mathcal{S}$ as input and returns a Pareto set of $\mathcal{S} \cup \{P\}$.

Correctness and complexity

MC Dijkstra algorithm solves the Multicriteria Shortest Paths problem (see [15] and [9]). Its complexity depends in particular heavily on the parts removing dominated paths, i.e. on the functions `IsNotDominated` and `InsertAndClean`. Nevertheless, existing papers simply use a naive algorithm for these functions, except for dimension 2, for which [10] claims a

Algorithm 1 MC Dijkstra overview.

Input: Graph $G = (V, A)$ with V the vertices, A the arcs, $s \in V$ the source vertex
Output: Sets $\mathcal{S}_u$ for every vertex u

```
1  begin Initialization
2      foreach u ∈ V do
3          S_u ← ∅ ; T_u ← ∅ ;
4      T_s ← {empty path from s to s} ;
5  while ⋃_{u∈V} T_u ≠ ∅ do
6      let P of destination v be the lexmin of ⋃_{u∈V} T_u ;
7      T_v ← T_v \ {P} ;
8      S_v ← S_v ∪ {P} ;
9      foreach (v, w) ∈ A do
10         if IsNotDominated(P · (v, w), S_w) then
11             T_w ← InsertAndClean(P · (v, w), T_w) ;
```

logarithmic complexity. In order to lower the complexity of MC Dijkstra, we may use the algorithms described in Section 2 to remove paths that are dominated. For $d = 2$ and $d = 3$, we can use online algorithms since MC Dijkstra processes elements in lexicographic order.

▶ **Proposition 1.** *[partially from [10] and [3]] Let μ be the maximal number of parallel arcs between a pair of vertices, and S be the Pareto set size. The output-sensitive time complexity of MC Dijkstra is $O(\Delta S \log(\Delta S))$ for $d \leq 3$, and $O(\mu \Delta S^2)$ for $d > 3$.*

Proof. In all cases, the size of a set $\mathcal{T}_u$ (the subset of paths from $\mathcal{T}$ having the same destination u) is upper-bounded by μS, since any path is an extension of an optimal one (a path in some $\mathcal{S}_v$), and there exists at most μ extensions of a path having the same destination. The same reasoning leads to the fact that the union of all the sets $\mathcal{T}_u$ has cardinality at most ΔS.

Besides, the repeated application of Line 6 requires to efficiently store the sets $\mathcal{T}_u$. The used data structure keeps the elements in $\bigcup_{u \in V} \mathcal{T}_u$ sorted. This hidden sorting in Lines 7 and 11 leads to a complexity in $O(\log(\Delta S))$ when inserting or removing a vertex.

Therefore, in each of the at most ΔS iterations of the while loop, the time complexity is upper-bounded by $O(\log(\Delta S))$ (the sorting time) plus the time needed to execute the functions `IsNotDominated` and `InsertAndClean`.

For $\mathbf{d = 2}$, the proof is essentially the same as in [10]. Since in MC Dijkstra the path P is lexicographically larger than any element in $\mathcal{S}$, the function `IsNotDominated`$(P, \mathcal{S})$ can be computed in constant time with the algorithm in [12], instead of time $O(\log(\mu S))$ by using a tree as proposed in [10]. However, the function `InsertAndClean`$(P, \mathcal{T})$ has an amortized complexity of $O(\log |\mathcal{T}|)$ to keep the structure sorted, amortized since it may remove a lot of paths during one call but a path can be removed only once. Anyway, the complexity in this case is dominated by the sorting time, leading to the overall complexity $O(\Delta S \log(\Delta S))$.

For $\mathbf{d = 3}$, using the algorithm proposed in [12] and the same reasoning as in the $d = 2$ case, the functions `IsNotDominated` and `InsertAndClean` can be computed in logarithmic time, leading to the same overall complexity as in the case $d = 2$.

For $\mathbf{d > 3}$, we extend the proof for $\mu = 1$ (simple graph) given in [3]: the dominance relation of the current path is iteratively tested with each element of the sets $\mathcal{S}_u$ and $\mathcal{T}_u$ for some u. The latter being upper-bounded by μS, we obtain the overall time complexity in $O(\mu \Delta S^2)$. ◀

3.2 Meta Rank Algorithm

Unfortunately, the efficient methods from Section 2 are not suitable in dimensions larger than 3, since those are offline. Yet, if the paths are processed in subsets, we could apply an offline method to each subset. For this purpose, we may group paths by rank, allowing to have several paths with the same destination in the same group. We will then process groups in increasing rank order, so that we keep the nice property that the "smallest" elements of $\mathcal{T}$ incorporated in $\mathcal{S}$ cannot be dominated by paths that are discovered later. This idea to process paths in increasing rank order is already used in [22] to compute Pareto sets.

Using this method, it is much easier to test dominance when paths are leaving the set $\mathcal{T}$ rather than when they enter it, because the paths are leaving the set $\mathcal{T}$ in increasing rank order, while this is not the case for their entering. Furthermore, we may take advantage of this dominance pruning step by group to also remove some optimal paths in order to output a smaller approximated Pareto Set. In order to implement this versatility, we propose a meta-algorithm Meta Rank (see Algorithm 2) which uses a blackbox function called `Sample`. If this function simply removes paths dominated by permanent solutions, Meta Rank solves the exact Multicriteria Shortest Paths problem. In the following, additional properties on `Sample` are defined in order to ensure that Meta Rank solves the $(1+\varepsilon)$-approximated Multicriteria Shortest Paths problem. Later on, instanciations of `Sample` are provided.

Algorithm 2 Meta Rank overview.

Input: Graph $G = (V, A)$ with V the vertices, A the arcs, $s \in V$ the source vertex
Output: Sets $\mathcal{S}_v$ for every vertex v

Initialization: $(\forall u \in V\ \mathcal{S}_u \leftarrow \emptyset\ ;\ \mathcal{T}_u \leftarrow \emptyset\ )\ ;\ \mathcal{T}_s \leftarrow \{\text{empty path from } s \text{ to } s\}\ ;$
while $\bigcup_{u\in V} \mathcal{T}_u \neq \emptyset$ **do**
  let r be the minimum rank in $\bigcup_{u\in V} \mathcal{T}_u$;
  foreach $v \in V$ **do**
    let $\mathcal{R}$ be the paths of destination v and of rank r in $\bigcup_{u\in V} \mathcal{T}_u$;
    $\mathcal{R}' \leftarrow \texttt{Sample}(\mathcal{R}, \mathcal{S}_v, \varepsilon)$;
    $\mathcal{T}_v \leftarrow \mathcal{T}_v \setminus \mathcal{R}\ ;\ \mathcal{S}_v \leftarrow \mathcal{S}_v \cup \mathcal{R}'$;
    foreach $P \in \mathcal{R}'$ **do**
      foreach $(v, w) \in A$ **do**
        $\mathcal{T}_w \leftarrow \mathcal{T}_w \cup \{P \cdot (v, w)\}$;

The following theorem gives the complexity of Meta Rank, depending on `Sample`'s one.

▶ **Theorem 2.** *Let S_ε be the size of Meta Rank's output and $C_{Sample}(n, S_\varepsilon, \Delta)$ be the complexity of the repeated usage of* `Sample` *during Meta Rank. Then Meta Rank time complexity is $C_{Sample}(n, S_\varepsilon, \Delta) + O(\Delta S_\varepsilon \log(\Delta S_\varepsilon))$. If the weights are in $[\![1, C]\!]$, the complexity is $C_{Sample}(n, S_\varepsilon, \Delta) + O(\Delta dn(nC)^{d-1} \log(\Delta nC))$.*

Proof. In order to justify precisely the claimed complexity, we provide details about the chosen data structures. For a better legibility, we introduce the notations $\mathcal{T}^{(r)}$ (resp. $\mathcal{T}_u^{(r)}$) as the subset of $\mathcal{T}$ (resp. $\mathcal{T}_u$) of paths having a rank r.

- The set $\mathcal{T}$ is a priority queue and its elements are the sets $\mathcal{T}^{(r)}$. The priority is given by r (the smaller r, the higher priority). We use a strict Fibonacci heap, guaranteeing a constant time complexity for insertion and a $O(\log(\Delta S_\varepsilon))$ complexity to remove the highest priority element.

- For a given rank r, $\mathcal{T}^{(r)}$ is an array. In order to do that, a unique identifier $[\![0, n-1]\!]$ is given to each vertex. If the identifier of u is i_u, $\mathcal{T}^{(r)}[i_u] = \mathcal{T}_u^{(r)}$, guaranteeing a constant *worst case* time complexity for accessing or removing a $\mathcal{T}_u^{(r)}$ set. Whereas this implementation is interesting in a theoretical point of view, a hash table would be more relevant in practice for memory purpose, since $\mathcal{T}$ may contains only a fragment of V at the same time. This choice would only guarantee a constant *mean* time complexity. A key would be a vertex and the associated value to a key u would be $\mathcal{T}_u^{(r)}$.
- The sets $\mathcal{T}_u^{(r)}$ are represented as chained lists in order to obtain a constant time insertion.
- $\mathcal{S}$ is also an array and the sets $\mathcal{S}_v$ are chained lists.

Given these data structures, the lines 10 and 11 (Alg. 2) are in $O(\log(\Delta S_\varepsilon))$, thus their repetition are in $O(\Delta S_\varepsilon \log(\Delta S_\varepsilon))$. Line 13 has an overall $O(S_\varepsilon)$ complexity. The repetition of the loop at line 14 has an overall complexity of $O(\Delta S_\varepsilon)$ since the number of added path in some T_w is upper-bounded by ΔS_ε. ◀

3.3 Algorithms Based on Sectors

3.3.1 Elimination Criterion

It turns out that the framework provided by Algorithm Meta Rank (Alg. 2) can compute $(1+\varepsilon)$-Pareto paths, by defining an appropriate `Sample` function. To guarantee Algorithm Meta Rank to output a $(1+\varepsilon)$-approximated Pareto set, we require the following ε-weak framing property.

▶ **Definition 3** (ε-Weak framing property). *A function* `Sample` *outputting* $\mathcal{R}' \subseteq \mathcal{R}$ *on input* $(\mathcal{R}, \mathcal{S}, \varepsilon)$ *satisfies the* ε-weak framing property *if, for every path* $P \in \mathcal{R} \setminus \mathcal{R}'$*, there exists* d *representative paths* $Q^{(1)}, \ldots, Q^{(d)}$ *in* $\mathcal{S} \cup \mathcal{R}'$ *such that, for every* i*,* $Q_i^{(i)} \leq (1+\varepsilon)P_i$ *and* $\forall j \neq i, Q_j^{(i)} \leq P_j$*. Furthermore,* $\mathcal{S} \cup \mathcal{R}'$ *is a set of incomparable paths.*

Notice that if $P \in \mathcal{R}$ is dominated by $Q \in \mathcal{S}$, it is sufficient to set $Q^{(i)} = Q$ for all i. Overall, this ε-weak property guarantees that the output of Meta Rank is a $(1+\varepsilon)$-Pareto set.

▶ **Theorem 4.** *With a function* `Sample` *satisfying the* ε*-weak framing property,* Meta Rank *algorithm (Alg. 2) solves the* $(1+\varepsilon)$*-approximate Multicriteria Shortest Paths problem.*

Proof. Let $\mathcal{S}$ be a Pareto set and $\mathcal{S}_a$ be the output of the algorithm. It is sufficient to show that for any path $P \in \mathcal{S}$, there exists a path $Q \in \mathcal{S}_a$ such that P is $(1+\varepsilon)$-covered by Q and $\texttt{rank}(Q) \leq \texttt{rank}(P)$. By contradiction, let $P' \in \mathcal{S}$ be a minimal rank path not $(1+\varepsilon)$-covered by any $Q \in \mathcal{S}_a$ such that $\texttt{rank}(Q) \leq \texttt{rank}(P')$. P' cannot be an empty path since the only one the algorithm can process is the one from the source to itself, and being the first one to leave $\mathcal{T}$, it is inserted in S. Thus, we can write $P' = P \cdot e$, with P a path and e the last arc of P'. P having an inferior rank than $P \cdot e$, there exists a path $Q \in S_a$ $(1+\varepsilon)$-covering P. If P is kept in $\mathcal{S}_a$, then $P \cdot e$ is inserted in $\mathcal{T}$ and is either kept in $\mathcal{S}_a$ or removed because of some representatives. In either cases, it is $(1+\varepsilon)$-covered, which is absurd. Otherwise, P is not kept in $\mathcal{S}_a$ and in particular, $P \neq Q$. Since $\texttt{rank}(Q) \leq \texttt{rank}(P)$, there exists a dimension i such that $P_i \leq Q_i$. Furthermore, $Q \in \mathcal{S}_a$ implies that it is extended and that $Q \cdot e$ is inserted in $\mathcal{T}$. However, Q $(1+\varepsilon)$-covers P, thus $Q \cdot e$ $(1+\varepsilon)$-covers $P \cdot e$ and:

$$\begin{cases} Q_i + e_i \leq P_i + e_i \\ \forall j \neq i, Q_j + e_j \leq (1+\varepsilon)(P_j + e_j) \end{cases}$$

That is why $Q \cdot e$ cannot be in $\mathcal{S}_a$. This means that $Q \cdot e$ is removed because of some representative paths, among which a path $R \in \mathcal{S}_a$, with $\texttt{rank}(R) \leq \texttt{rank}(Q \cdot e)$, that satisfies:

$$\begin{cases} R_i \leq (1+\varepsilon)(Q_i + e_i) \\ \forall j \neq i, R_j \leq Q_j + e_j \end{cases}$$

Then:

$$\begin{cases} R_i \leq (1+\varepsilon)(Q_i + e_i) \leq (1+\varepsilon)(P_i + e_i) \\ \forall j, R_j \leq Q_j + e_j \leq (1+\varepsilon)(P_j + e_j) \end{cases}$$

Which means that $P \cdot e$ is $(1+\varepsilon)$-covered by R. Since R is in $\mathcal{S}_a$, we obtain a contradiction. ◀

Two caracteristics of `Sample` are of particular interest: the time complexity and the number of paths the function removes. Naive greedy algorithms are not efficient for either of these metrics. Thus, we propose a sample algorithm guaranteeing the ε-weak framing property, achieving a good tradeoff for the two caracteristics. Given a d-dimensional space, we define d *sectors* for every path P.

▶ **Definition 5.** *The* i-th sector *of* P *contains every point* Q *with* $Q_j \leq P_j$ *for* $j \neq i$. *Given* ε, *the boolean function* $coverSector(P, Q, i, \varepsilon)$ *is* `True` *if* Q *belongs to the* i*-th sector and* $(1+\varepsilon)$*-covers* P.

In Figure 1b, the two rectangles represent the incomparable part of the two sectors 2-covering B, i.e., the points Q not dominating B satisfying $coverSector(B, Q, 1, 1) = True$ (for instance C, D and E), and $coverSector(B, Q, 2, 1) = True$ respectively (such as A). For three criteria, the Figure 2 depicts the three sectors covering a point P.

3.3.2 Sample Sector

We propose Sample Sector, an algorithm implementing the `Sample` function. It considers each dimension i independently to compute a set of paths $\mathcal{R}'_i \subseteq \mathcal{R}$ and the output of the algorithm is $\mathcal{R}' = \bigcup_{i=1}^{d} \mathcal{R}'_i$.

Let r be the rank of all paths in $\mathcal{R}$, and let i be a dimension. We partition $\mathcal{R}$ into *strips* $\mathcal{R}_i^{(l)}$, for $l \in [\![0, \lceil \log_{1+\varepsilon} r \rceil + 1]\!]$. $\mathcal{R}_i^{(0)}$ (resp. $\mathcal{R}_i^{(1)}$) contains the paths such that $P_i = 0$ (resp. $P_i = 1$). For $l \geq 2$, $P \in \mathcal{R}$ belongs to $\mathcal{R}_i^{(l)}$ if its i-th coordinate P_i is in $\left((1+\varepsilon)^{l-2}, (1+\varepsilon)^{l-1}\right]$. Our algorithm Sample Sector proceeds as follows: $\mathcal{R} \cup \mathcal{S}$ is first preprocessed to answer quickly range queries. Then, for every path $P \in \mathcal{R}_i^{(l)}$, we add P to $\mathcal{R}'_i$ if P is not $(1+\varepsilon)$-covered in its i-th sector by a path of $\mathcal{R} \cup \mathcal{S}$ in the same strip $\mathcal{R}_i^{(l)}$. This can be done using $\texttt{RangeQuery}([0, P_1] \times [0, P_2] \times \cdots \times [0, P_{i-1}] \times [P_i, (1+\varepsilon)^{l-1}] \times [0, P_{i+1}] \times \cdots \times [0, P_d], \mathcal{R} \cup \mathcal{S})$.

In Figure 2, the grey z-strip contains only 6 points, the other one in the sector cannot be used to represent P since it is outside the grey zone.

▶ **Definition 6.** *Algorithm Sector is the Meta Rank algorithm (Alg. 2) using Sample Sector.*

As mentioned in the introduction, Sector solves the $(1+\varepsilon)$-Multicriteria shortest path problem. Combined with Theorem 4, the following theorem confirms that.

▶ **Theorem 7.** *Sample Sector satisfies the* ε*-weak property when* $\mathcal{R}$ *and* $\mathcal{S}$ *are both Pareto sets such that any path of* $\mathcal{R}$ *has a larger rank than any path of* $\mathcal{S}$.

Proof. In both `Sample` functions, we have to prove that if a path P of rank r has been removed, $\mathcal{S} \cup \mathcal{R}'$ contains d paths guaranteeing the ε-weak framing property. Let us focus on one dimension i.

If the range query returns a non empty set $\mathcal{Q}$ for the P's i-th sector of its strip, we have two cases: (1) the corresponding subspace contains at least a permanent path in $\mathcal{S}$ or (2) only contains paths of same rank. In the first case, we are sure that path P will have a representative path in its i-th sector whereas in the second case, these paths might be not kept in $\mathcal{R}'_i$. This case is not possible since the path in $\mathcal{Q}$ with the highest value for its i-th coordinate is added in $\mathcal{R}'_i$. In both cases, if a path does not belong to $\mathcal{R}'_i$, then there is at least one path in $\mathcal{R}'_i \cup \mathcal{S}$ that $(1+\varepsilon)$-covers P in its i-th sector.

By construction, any path P kept in Sample Sector has no representative path in at least one of its sector in the same strip. ◀

The following theorem states the output-sensitive time complexity of Sector given in Table 1, along with the space complexity and the time complexity in the special case where weights are integers. In order to conclude, it is sufficient to compute the sum of the complexities of the Sample Sector calls in Sector, and then to use Theorem 2.

▶ **Theorem 8.** *If the arc weights are integers, the output $\mathcal{S}_\varepsilon$ of Sector is of size $S_\varepsilon = O\left(dnC(nC)^{d-2}\log_{1+\varepsilon}(nC)\right)$. The time complexity of Sector is $O(\Delta S_\varepsilon \log^{d-1}(\Delta S_\varepsilon))$ and the space complexity is $\Theta(\Delta S_\varepsilon \log^{d-1}(\Delta S_\varepsilon))$.*

Proof. Assume first that the weights are integers. Given a current rank r and a strip $\mathcal{R}_i^{(l)}$, Sample Sector stores at most one path for every $x \in \mathbb{Z}^{d-2}$. Thus for every i, $|\mathcal{R}_i'^{(l)}| = O(r^{d-2})$. Since we have at most $\lceil 2 + \log_{1+\varepsilon} r \rceil$ strips and d dimensions, $|\mathcal{R}'|$ is smaller than or equal to $d(r+1)^{d-2}(\lceil 2 + \log_{1+\varepsilon} r \rceil)$. Since we have dnC ranks, $S_\varepsilon = O(d(dnC)^{d-1}\log_{1+\varepsilon}(dnC))$.

To get bounds on $C_{\text{Sample Sector}}$, we have to build data structures dedicated to range queries. The number of insertions to do before the queries is bounded by $O(\Delta\mathcal{S}_\varepsilon)$. Each of these insertions takes $O(\log^{d-1}\Delta\mathcal{S}_\varepsilon)$ and a range query takes $O\left(\left(\frac{\log \mathcal{S}_\varepsilon}{\log\log \mathcal{S}_\varepsilon}\right)^{d-1}\right)$ [16]. Then the number of range queries is at most $d\Delta\mathcal{S}_\varepsilon$. Thus $C_{\text{Sample Sector}} = O(\Delta\mathcal{S}_\varepsilon \log^{d-1}\Delta\mathcal{S}_\varepsilon)$.

From Theorem 2, we have to add $O(\Delta S_\varepsilon \log(\Delta S_\varepsilon))$ time steps to get the complexity of both algorithms assuming d is constant. Whenever the arc weights are integers we also have $\Delta S_\varepsilon \leq dnC$. ◀

4 Frame (dimension 2)

4.1 Elimination Criterion

Sector could potentially return non optimal solutions. In order to guarantee the Pareto compatibility property, we introduce a stronger property, based on the idea that the representatives of a path have to cover themselves too. However, we will restrict the definition for $d = 2$ because it is not giving satisfying results in higher dimensions.

We start by giving the formal definition of what we call being framed between two paths. This definition is commented and illustrated afterwards.

▶ **Definition 9** (Frame). *For any paths A, P, B s.t. $\textit{rank}(A) \leq \textit{rank}(P)$ and $\textit{rank}(B) \leq \textit{rank}(P)$, we say that A and B* frame *P, or that P is* framed *between A and B if:*

(i) $A_1 \leq P_1$ (iii) $A_2 \leq (\textit{rank}(P) - B_1)(1+\varepsilon)$
(ii) $B_2 \leq P_2$ (iv) $B_1 \leq (\textit{rank}(P) - A_2)(1+\varepsilon)$

We will note this property $\textit{frame}(A, P, B, \varepsilon)$.

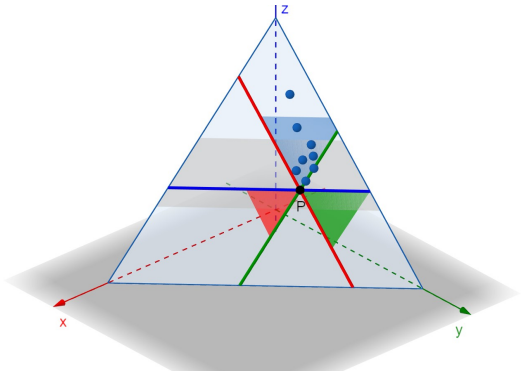


Figure 2 In 3D, the three sectors covering P at distance at most $(1+\varepsilon)$ are depicted in green, red and blue. Only 6 points are within the grey z-strip of P.

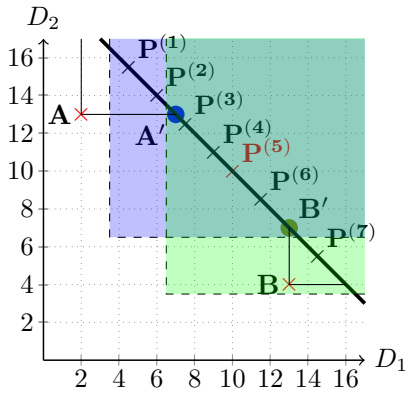


Figure 3 Sample Frame. The paths $P^{(3)}, P^{(4)}, P^{(5)}$ and $P^{(6)}$ are framed by A and B for $\varepsilon = 1$ (see colored regions) but not for $\varepsilon = 0.5$. In this latter case, the algorithm keeps the middle point $P^{(5)}$.

In the particular case where the two paths A and B have the same rank as the path P, if $\texttt{frame}(A, P, B, \varepsilon)$, then A and B match the $Q^{(1)}$ and $Q^{(2)}$ representatives of P in the ε-weak framing property, with the additional constraint that A and B $(1+\varepsilon)$-cover each other. This definition is extended for A and B having lower ranks than $\texttt{rank}(P)$, projecting those into the line of paths having the same rank as P. A is projected on the second dimension, B on the first one. These projections of A and B are depicted in Figure 3 as A' and B'. The blue (resp. green) zone corresponds to the paths 2-covered by A' (resp. B'). Notice that the frame property requires the projections A' and B' to cover each other, but not necessarily A and B. Thus, in this example, for $3 \leq i \leq 6$, $\texttt{frame}(A, P^{(i)}, B, \varepsilon)$ since $\texttt{frame}(A', P^{(i)}, B', \varepsilon)$.

We define the ε-strong framing property as a particular case of the ε-weak framing property for which the representatives of a path are framing it according to Def. 9.

▶ **Definition 10** (ε-Strong framing property). *A function* `Sample` *outputting* $\mathcal{R}'$ *on input* $(\mathcal{R}, \mathcal{S}, \varepsilon)$ *satisfies the* ε-strong framing property *if:*

- $\forall P \in \mathcal{R} \setminus \mathcal{R}',\ \exists A, B \in \mathcal{S} \cup \mathcal{R}',\ \textit{frame}(A, P, B, \varepsilon)$,
- $\mathcal{R}'$ *is minimal by inclusion,*
- $\mathcal{S} \cup \mathcal{R}'$ *is a Pareto set.*

As the name suggests, the strong property is stronger than the weak one since it requires some conditions between the representatives, as well as the minimality of the output.

▶ **Proposition 11.** *The ε-strong framing property implies the ε-weak framing property.*

Proof. Let `Sample` verifying the ε-strong framing property on inputs $(\mathcal{R}, \mathcal{S}, \varepsilon)$. Let $P \in \mathcal{R} \setminus \mathcal{R}'$. There exists $A, B \in \mathcal{S} \cup \mathcal{R}'$ such that $\texttt{frame}(A, P, B, \varepsilon)$. Since $\mathcal{S} \cup \mathcal{R}'$ is a Pareto set, we only need to show that there exists two representatives $Q^{(1)}, Q^{(2)} \in \mathcal{S} \cup \mathcal{R}'$, such that:

$$\begin{cases} Q_1^{(1)} \leq (1+\varepsilon)P_1 \\ Q_2^{(1)} \leq P_2 \end{cases} \quad \begin{cases} Q_2^{(2)} \leq (1+\varepsilon)P_2 \\ Q_1^{(2)} \leq P_1 \end{cases}$$

Unfortunately, setting $Q^{(1)} = B$ and $Q^{(2)} = A$ is not always sufficient. We consider three cases:

- if $A_2 \leq P_2$, then $Q^{(1)} = Q^{(2)} = A$ is correct,

- if $B_1 \leq P_1$, then $Q^{(1)} = Q^{(2)} = B$ is correct,
- otherwise, we take $Q^{(1)} = B$ and $Q^{(2)} = A$. Indeed:
  - $Q_1^{(2)} = A_1 \leq P_1$ and $Q_2^{(1)} = B_2 \leq P_2$ by (i) and (ii) (cf Def. 9),
  - $Q_1^{(1)} = B_1 \leq (1+\varepsilon) \cdot (\texttt{rank}(P) - A_2) = (1+\varepsilon) \cdot (P_1 + P_2 - A_2) \leq (1+\varepsilon)P_1$ by (iv)
  - $Q_2^{(2)} \leq (1+\varepsilon)P_2$ using symmetric arguments. ◀

The following theorem is a direct corollary of Theorem 4 and the previous proposition.

▶ **Theorem 12.** *With a function* `Sample` *satisfying the ε-strong framing property, Meta Rank algorithm (Alg. 2) solves the $(1+\varepsilon)$-approximate Multicriteria Shortest Paths problem.*

Proof. Corollary of Theorem 4 and Proposition 11. ◀

4.2 Pareto Compatible Property

During a Meta Rank execution, a path P could be framed, then removed. Furthermore, the extensions of the representatives could be themselves framed and removed, and so on. We show that the extensions of P are nevertheless still framed by kept paths in the ε-strong setting.

▶ **Lemma 13.** *Let $\mathcal{S}_\varepsilon$ be the output of Meta Rank (Alg. 2) using a* `Sample` *function satisfying the ε-strong property. Any path P is framed by some paths $A, B \in \mathcal{S}_\varepsilon$.*

Proof. For paths A, B and P, if P is framed by A and B, we note: $\alpha(P) = A, \beta(P) = B$ (beware that α and β are not functions, A and B not being necessarily unique). By contradiction, let us assume that there exist paths in the Pareto Set that are not framed by the output. Let P' be such a path of minimal rank. If P' is an empty path, then it is the first path seen by the algorithm, and it is kept, giving directly a contradiction. Otherwise, we can write $P' = P \cdot e$, with e being the last arc of P'. We have $\texttt{rank}(P) < \texttt{rank}(P \cdot e)$, thus P is framed by two paths $\alpha(P), \beta(P) \in \mathcal{S}_\varepsilon$ framing P. Notice that if P is kept, we can say that P is framed by (P, P). We will note: $A = \alpha(P) \cdot e$ and $B = \beta(P) \cdot e$. Since $\alpha(P)$ and $\beta(P)$ are kept, A and B will be considered by the algorithm but not necessarily kept.

We consider three cases:

1. If the algorithm keeps both A and B, then they frame $P \cdot e$, since they have inferior ranks and:
   (i) $A_1 = \alpha(P)_1 + e_1 \leq P_1 + e_1$
   (ii) $B_2 = \beta(P)_2 + e_2 \leq P_2 + e_2$
   (iii)
   $$\begin{aligned} A_2 &= \alpha(P)_2 + e_2 \\ &\leq (1+\varepsilon)(\texttt{rank}(P) - \beta(P)_1) + e_2 \\ &\leq (1+\varepsilon)(\texttt{rank}(P) - \beta(P)_1) + (1+\varepsilon)e_2 \\ &\leq (1+\varepsilon)(\texttt{rank}(P) - \beta(P)_1) + (1+\varepsilon)(\texttt{rank}(e) - e_1) \\ &\leq (1+\varepsilon)(\texttt{rank}(P) + \texttt{rank}(e) - \beta(P)_1 - e_1) \\ &\leq (1+\varepsilon)(\texttt{rank}(P \cdot e) - B_1) \end{aligned}$$
   (iv) $B_1 \leq (\texttt{rank}(P \cdot e) - A_2)(1+\varepsilon)$ by a reasoning similar to (iii)
2. The algorithm keeps only one. W.l.o.g., we can consider that A is kept. B being removed, it is framed by $\alpha(B)$ and $\beta(B)$.
   - Either $\alpha(B)_1 \leq P_1 + e_1$, in which case, P' is framed by $\alpha(B)$ and $\beta(B)$ too. Indeed, we have $\beta(B)_2 \leq B_2 \leq P_2 + e_2 = P'_2$ giving (ii). And (iii), (iv) are given by the fact that $\texttt{rank}(B) \leq \texttt{rank}(P')$.
   - Otherwise A and $\alpha(B)$ frame P'. Indeed,
     (i) $A_1 = \alpha(P)_1 + e_1 \leq P_1 + e_1 = P'_1$

(ii) $\alpha(B)_2 = \texttt{rank}(\alpha(B)) - \alpha(B)_1 \leq \texttt{rank}(P \cdot e) - (P_1 + e_1) \leq P_2 + e_2 = P'_2$
(iii) $A_2 \leq (1+\varepsilon)(\texttt{rank}(P) - \beta(P)_1) + e_2 \leq (1+\varepsilon)(\texttt{rank}(P \cdot e) - B_1) \leq (1+\varepsilon)(\texttt{rank}(P \cdot e) - \alpha(B)_1)$
(iv) $\alpha(B)_1 \leq B_1 \leq (1+\varepsilon)(\texttt{rank}(P) - \alpha(P)_2) + e_1 \leq (1+\varepsilon)(\texttt{rank}(P \cdot e) - A_2)$

3. The last case corresponds to removing both A and B. As in the previous case, if $\alpha(B)_1 \leq P_1 + e_1$, P is framed by $\alpha(B)$ and $\beta(B)$. Otherwise, A and $\alpha(B)$ frame P' and we can use the same reasoning than before, replacing B by $\alpha(B)$.

We have proved that P' is framed, leading to a contradiction. ◀

The idea is, by contradiction, to consider, among the paths not framed, one with minimum rank. This path cannot be empty, thus it can be written $P \cdot e$, with P a path and e an arc. By definition of $P \cdot e$, P is framed. Using paths A and B framing P, we can show that their extentions $A \cdot e$ and $B \cdot e$ are framing $P \cdot e$. These extentions are either kept in $\mathcal{S}_\varepsilon$ or in turn framed by some paths of $\mathcal{S}_\varepsilon$ framing $P \cdot e$ too.

It can be deduced from this lemma that the ε-strong property implies the Pareto compatibility.

▶ **Theorem 14.** *Meta Rank (Alg. 2) using a* `Sample` *function satisfying the ε-strong property is Pareto compatible property.*

Proof. By contradiction, we assume that some $P \in \mathcal{S}_\varepsilon$ is dominated by some path Q. If $Q \in \mathcal{S}_\varepsilon$, then P cannot be kept since it is processed after Q and is dominated. Therefore, $Q \notin \mathcal{S}_\varepsilon$. According to Lemma 13, there exists $A, B \in \mathcal{S}_\varepsilon$ framing Q. Thus, A, B frame P, which would mean that P is not kept since $\mathcal{S}_\varepsilon$ is minimal. ◀

4.3 Frame Algorithm

We provide an efficient algorithm for `Sample`: Sample Frame. The algorithm is first presented in a simplified version, which is generalized afterwards. Let $\mathcal{R} = \{P^{(1)}, \cdots, P^{(k)}\}$ be a set of paths of rank r. We assume the paths $P^{(i)}$ to be sorted in lexicographic order.

The simplified algorithm consists in finding the maximal index j such that $P^{(1)}$ and $P^{(j)}$ cover each other. Then, $\forall\ 1 < i < j$, $\texttt{frame}(P^{(1)}, P^{(i)}, P^{(j)}, \varepsilon)$ holds, and those paths in-between are removed. Next, the algorithm is repeated recursively on $\mathcal{R}' = \{P^{(j)}, \cdots, P^{(k)}\}$ until $\mathcal{R}'$ contains at most two paths. The output of the simplified algorithm consists of the set of paths from $\mathcal{R}$ that were not removed. See Alg. 3 for a more formal description of the simplified algorithm.

Algorithm 3 Sample Frame Same Rank.

Input: k paths $(P^{(1)}, \cdots, P^{(k)})$ sorted in lexicographic order, $\varepsilon > 0$

```
1 i_min ← 1 ;
2 for i = 2 to k − 1 do
3     if frame(P^(i_min), P^(i), P^(i+1), ε) then
4         Remove P^(i) ;
5     else
6         i_min ← i ;
```

In order to improve the pruning capability, paths from lower ranks are actually used to frame current rank paths. Assume that A and B are two paths of rank lower than r such that $\forall P \in \mathcal{R}, A_1 \leq P_1 \leq B_1$ and $B_2 \leq P_2 \leq A_2$. Then Sample Frame performs the following three steps:

1. Paths from $\mathcal{R}$ dominated by A are removed.
2. Let $A' = (r - A_2, A_2)$ and $B' = (B_1, r - B_1)$ be projections of A and B on the current rank r. If $P^{(i)}, \cdots, P^{(j)}$ are the paths from $\mathcal{R}$ non dominated by A or B, and sorted in lexicographic order, then the simplified algorithm is applied on $\{A', P^{(i)}, \cdots, P^{(j)}, B'\}$.
3. Paths from $\mathcal{R}$ dominated by B are removed.

An example of this case is depicted in Figure 3 for $\epsilon = 0.5$. The first step removes $P^{(1)}$ and $P^{(2)}$ since they are dominated by A. Then the second step computes the fact that A' and $P^{(5)}$ cover each other but not A' and $P^{(6)}$. Thus, $P^{(3)}$ and $P^{(4)}$ are removed too. Since $P^{(5)}$ and B' cover each other, $P^{(6)}$ is removed. Finally, during the third step, $P^{(7)}$ is removed because B dominates it. Sample Frame's output is $\{P^{(5)}\}$.

Sample Frame Algorithm

In a general setting, an unordered set $\mathcal{R} = \{P^{(1)}, \cdots, P^{(k)}\}$ of paths of rank r is given as input to Sample Frame, along with a Pareto set $\mathcal{S}$ of paths of rank lower than r. Algorithm Sample Frame proceeds as follows. First, $\mathcal{R}$ is sorted in lexicographic order. Then, let $A = \arg\max_{Q \in \mathcal{S}} \{Q_1 | Q_1 \leq P_1^{(1)}\}$ and $B = \arg\min_{Q \in \mathcal{S}} \{Q_1 | Q_1 > P_1^{(1)}\}$. Note that B is the path following A in $\mathcal{S}$ in lexicographic order. Let j be the maximal index such that $P_1^{(j)} < B_1$. Intuitively, the paths $P^{(1)}, \cdots, P^{(j)}$ are the paths between A and B as in the previously described situation. Sample Frame applies the corresponding three steps to these paths. Then, this algorithm is recursively applied on $\{P^{(j+1)}, \cdots, P^{(k)}\}$.

If A is not defined, then $A' = P^{(1)}$ and the algorithm is applied to $\mathcal{R} = \{P^{(2)}, \cdots, P^{(k)}\}$. Symmetrically, if B is not defined, then $B' = P^{(k)}$ and the algorithm is applied to $\mathcal{R} = \{P^{(1)}, \cdots, P^{(k-1)}\}$.

To search A and B among $\mathcal{S}$ efficiently, $\mathcal{S}$ is a balanced search tree allowing a logarithmic time search. Similarily to Sector using Sample Sector, we can now define our algorithm Frame using Sample Frame.

▶ **Definition 15.** *Algorithm Frame is the Meta Rank algorithm (Alg. 2) using Sample Frame.*

In order to confirm that Frame is Pareto compatible, it is sufficient to verify that Sample Frame satisfies the ε-strong property thanks to Theorem 14. Intuitively, one can see on the example depicted in Figure 3 that any removed path is either between two consecutive (in lexicographic order) kept paths, or dominated, thus framed by the dominating path.

▶ **Theorem 16.** *Sample Frame algorithm satisfies the ε-strong framing property.*

Proof. Deleted paths are always framed by kept paths. Furthermore, the output is minimal since the algorithm is framing the largest interval possible. Finally, for A and B fixed, steps 1 and 3 remove dominated paths, guaranteeing to have a Pareto Set as output. ◀

Sample Frame$(\mathcal{S}, \mathcal{R}, \varepsilon)$ is efficient since it processes sequencially the paths from $\mathcal{R}$, and potentially for each one of those, performs a logarithmic search through $\mathcal{S}$.

▶ **Proposition 17.** *Let R (resp. S) be the number of paths of rank r (resp. inferior to r). The complexity of the Sample Frame algorithm is $O(R(\log R + \log S)\})$.*

Proof. Paths of rank r are sorted in $O(R \log R)$ time. Then these paths are considered only once and each one may require to search for A and B in $O(\log S)$ time. ◀

With the previous proposition and Theorem 2, the time complexity of Frame, claimed in Table 1, is computable by summing the complexities of each call to Sample Frame.

▶ **Theorem 18.** *Let S_ε be the size of the output of Frame. The time complexity of Frame is in $O(\Delta S_\varepsilon \log(\Delta S_\varepsilon))$.*

Proof. For each vertex u and rank r, let T_u^r be the size of the first parameter of `Sample`, and $S_u^{<r}$ be the size of the second parameter of `Sample`. Then the complexity of `Sample` using Sample Frame is $O(T_u^r(\log S_u^{<r} + \log T_u^r))$ which is in $O(T_u^r(\log(\Delta S_\varepsilon)))$ since $T_u^r \leq \Delta S_\varepsilon$. Repeating this operation over each vertex and rank gives $C_{Sample}(n, S_\varepsilon, \Delta) = O(\Delta S_\varepsilon \log(\Delta S_\varepsilon))$. Furthermore, recall that adding an optimal path to the set of permanent paths costs $O(\log S_\varepsilon)$, therefore the overall complexity for the line 13 of Meta Rank (Alg. 2) is $O(S_\varepsilon \log S_\varepsilon)$. Applying Theorem 2 allows us to conclude. ◀

5 Is the Pareto-compatible property practically relevant ?

Although Frame is Pareto compatible, it is interesting to check whenever S_ε given by Frame is really smaller than S in practice. We run shortest path queries for $d = 2$ for two types of graphs: small synthetic graphs but with large exact Pareto sets and large real-life graphs, up to 1 millions arcs with relatively small exact Pareto sets. For these experiments, we take $\varepsilon = 1$. Then, we study the impact of the variation of ε on the size of $\mathcal{S}_\varepsilon$. S is computed using an optimized version of MC Dijkstra dedicated to $d = 2$.

Algorithms have been implemented in C++, using data structures which guarantee the desired complexities for dimension 2. Temporary and permanent solution sets ($\mathcal{T}_u$ and $\mathcal{S}_u$) are implemented using *std::set* class template. For MC Dijkstra, a global temporary solution is used to store the minimum path of each $\mathcal{T}_u$. It is also a *set*, and the priority list of Meta Rank is implemented using *std::map* class template. The program is compiled with g++-8 and the option -o2, since the used space can be huge. It is executed on a computer running Ubuntu 18.04.3, having 16GB RAM and an Intel Core i7-6700 processor.

Oriented complete graphs. We use the graphs construction proposed by Breugem *et al.* (see [3] for the exact description) to get oriented complete graphs $\overrightarrow{K_n}$ with large exact Pareto sets (2^{n-2} for n vertices), and, for given n ($n = 19$ for us), to generate intermediate graphs between $\overrightarrow{K_n}$ and the standard Erdös-Renyi random graphs. Parameter p defines the closeness to these two extreme graphs: every arc of $\overrightarrow{K_n}$ is changed (removed or redirected) with probability p. Whenever $p = 0$, we get $\overrightarrow{K_n}$, and for $p = 1$, we have a pure random graph.

For this extreme case, S_ε is much smaller than S for small values of p. Figure 6 shows that Frame is at least 10^5 times quicker than MC Dijkstra for $\overrightarrow{K_{19}}$ ($p = 0$). For $p < 0,5$, Frame is still several orders of magnitude faster than MC Dijkstra. However, MC Dijkstra performance improves whenever p increases and that of Frame remains stable. This is explained by S being small for p close to 1.

Real-life graphs. The previous graphs are small and dense. We now study the impact of the number of vertices for sparse graphs. We take the graphs given by the 9^{th} challenge of DIMACS [7]. It offers bicriteria (distance and edge traversal time) datasets on road networks for different USA states. On these graphs, 100 shortest path queries are performed randomly and we report the average Pareto set size $\bar{S}_u$ for a random destination u. We remark that for small $\bar{S}_u$, Frame and MC Dijkstra performs similarly (Tab. 2), whereas for larger $\bar{S}_u$, Frame has a gain of 30%.

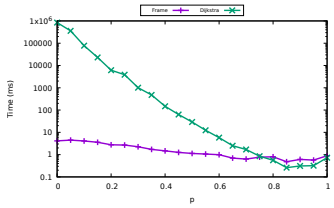


Figure 4 Time for $\overrightarrow{K_{19}}$ variations.

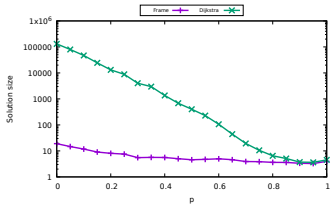


Figure 5 S_ε for $\overrightarrow{K_{19}}$ variations.

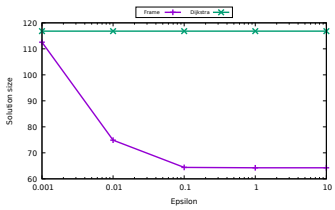


Figure 6 The impact of ε on $S_{u,\varepsilon}$.

Table 2 MC Dijkstra vs Frame on DIMACS (time in ms).

Graph	Vertices	Arcs	MC Dijkstra Time	Frame Time	$\tilde{S}_u$	$\tilde{S}_{u,1}$(Frame)
DC	9559	14909	79.34	76.02	4.84	4.22
RI	53658	69213	154.78	148.49	5.24	4.37
WY	253077	304014	309.18	253.28	7.73	5.31
NM	467529	567084	1333.5	1209.93	22.09	14.92
VA	630639	714809	10943.98	7475.28	62.87	48.84
NC	887630	1009846	25206.34	17637.98	66.78	49.93

Impact of ε. Up to now, we set up ε to 1. We now introduce the variation of $\varepsilon = 10^k$ with $k \in [\![-3, 1]\!]$ on square grids of 10000 sommets. The arcs weights are randomly drawn between 1 and 100. The sources and the destinations are also randomly chosen. We observe in Figure 6 that whenever ε goes to 0, the output of Frame converges to $\mathcal{S}$. For ε larger than 1, $\mathcal{S}_\varepsilon$ is almost constant (around 60) whereas two paths are enough to cover $\mathcal{S}$. It shows the limitation of the Pareto compatibility property of Frame.

6 Conclusion

In the current description of Meta Rank, we assume that the rank of each edge is non-null. We can easily handle this limitation: in order to be able to consider at once all paths having the same rank, we can add a step before applying Sample. It consists simply in extending recursively every path with null rank arcs whenever it is possible.

In this article, we get the first approximated algorithm being Pareto compatible. It would be interesting to provide other algorithms with this property but in dimension ≥ 3. Moreover, Frame and Sector are promising from a practical point of view. Experiments comparing them with the best exact and approximated algorithms would be an interesting future work. In our experiments, we observed that Frame is always competitive with respect to MC Dijkstra in various situations. The bigger the Pareto set, the better Frame. However, even if $S_\varepsilon < S$, it can be far from S^*_ε. We let open the question of getting a constant approximation of S^*_ε with a polynomial time algorithm whenever C is bounded. Another question is to get an efficient algorithm in 3 dimensions. Algorithm Sector is promising but is not Pareto compatible, limiting the theoritical gain.

References

1 Florian Barth, Stefan Funke, and Sabine Storandt. Alternative multicriteria routes. In *2019 Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 66–80. SIAM, 2019.

2 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route Planning in Transportation Networks. In *Algorithm Engineering*, volume 9220 of *Lecture Notes in Computer Science*, pages 19–80. Springer, Cham, 2016. doi:10.1007/978-3-319-49487-6_2.

3 Thomas Breugem, Twan Dollevoet, and Wilco van den Heuvel. Analysis of FPTASes for the multi-objective shortest path problem. *Computers & Operations Research*, 78(Supplement C):44–58, February 2017. doi:10.1016/j.cor.2016.06.022.

4 Fritz Bökler and Markus Chimani. Approximating Multiobjective Shortest Path in Practice. In *2020 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, Proceedings, pages 120–133. Society for Industrial and Applied Mathematics, December 2019. doi:10.1137/1.9781611976007.10.

5 Daniel Delling, Julian Dibbelt, and Thomas Pajor. Fast and exact public transit routing with restricted pareto sets. In *2019 Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 54–65. SIAM, 2019.

6 Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-Based Public Transit Routing. *Transportation Science*, 49(3):591–604, October 2014. doi:10.1287/trsc.2014.0534.

7 Camil Demetrescu, Andrew Goldberg, and David Johnson. 9th DIMACS Implementation Challenge: Shortest Paths. URL: http://users.diag.uniroma1.it/challenge9/.

8 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly Simple and Fast Transit Routing. In *Experimental Algorithms*, Lecture Notes in Computer Science, pages 43–54. Springer, Berlin, Heidelberg, June 2013. doi:10.1007/978-3-642-38527-8_6.

9 Matthias Ehrgott. *Multicriteria optimization*. Springer, Berlin ; New York, 2nd ed edition, 2005.

10 Pierre Hansen. Bicriterion Path Problems. In Günter Fandel and Tomas Gal, editors, *Multiple Criteria Decision Making Theory and Application*, Lecture Notes in Economics and Mathematical Systems, pages 109–127. Springer Berlin Heidelberg, 1980.

11 J. Hrnčíř, P. Žilecký, Q. Song, and M. Jakob. Practical Multicriteria Urban Bicycle Routing. *IEEE Transactions on Intelligent Transportation Systems*, 18(3):493–504, March 2017. doi:10.1109/TITS.2016.2577047.

12 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, October 1975. doi:10.1145/321906.321910.

13 E. Machuca and L. Mandow. Multiobjective heuristic search in road maps. *Expert Systems with Applications*, 39(7):6435–6445, June 2012. doi:10.1016/j.eswa.2011.12.022.

14 Lawrence Mandow and José. Luis Pérez De La Cruz. Multiobjective A* search with consistent heuristics. *Journal of the ACM*, 57(5):1–25, June 2010. doi:10.1145/1754399.1754400.

15 Ernesto Queirós Vieira Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2):236–245, May 1984. doi:10.1016/0377-2217(84)90077-8.

16 Christian Worm Mortensen. Fully Dynamic Orthogonal Range Reporting on RAM. *SIAM Journal on Computing*, 35(6):1494–1525, January 2006. Publisher: Society for Industrial and Applied Mathematics. doi:10.1137/S0097539703436722.

17 Matthias Müller-Hannemann and Karsten Weihe. On the cardinality of the Pareto set in bicriteria shortest path problems. *Annals of Operations Research*, 147(1):269–286, October 2006. doi:10.1007/s10479-006-0072-1.

18 C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of Web sources. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 86–92, 2000. doi:10.1109/SFCS.2000.892068.

19 Andrea Raith and Matthias Ehrgott. A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research*, 36(4):1299–1331, April 2009. doi:10.1016/j.cor.2008.02.002.

20 Michael Shekelyan, Gregor Josse, and Matthias Schubert. Linear path skylines in multicriteria networks. In *2015 IEEE 31st International Conference on Data Engineering*, pages 459–470, Seoul, South Korea, April 2015. IEEE. doi:10.1109/ICDE.2015.7113306.

21 George Tsaggouris and Christos Zaroliagis. Multiobjective Optimization: Improved FPTAS for Shortest Paths and Non-Linear Objectives with Applications. *Theory of Computing Systems*, 45(1):162–186, June 2009. doi:10.1007/s00224-007-9096-4.

22 Chi Tung Tung and Kim Lin Chew. A multicriteria Pareto-optimal path algorithm. *European Journal of Operational Research*, 62(2):203–209, October 1992. doi:10.1016/0377-2217(92)90248-8.

23 Sibo Wang, Xiaokui Xiao, Yin Yang, and Wenqing Lin. Effective indexing for approximate constrained shortest path queries on large road networks. *Proceedings of the VLDB Endowment*, 10(2):61–72, October 2016. doi:10.14778/3015274.3015277.

24 Arthur Warburton. Approximation of Pareto Optima in Multiple-Objective, Shortest-Path Problems. *Operations Research*, 35(1):70–79, February 1987. doi:10.1287/opre.35.1.70.

25 Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. Path Problems in Temporal Graphs. *Proc. VLDB Endow.*, 7(9):721–732, May 2014. doi:10.14778/2732939.2732945.