algorithms

Article

An FPTAS for Dynamic Multiobjective Shortest Path Problems

Pedro Maristany de las Casas [1,*], Ralf Borndörfer [1], Luitgard Kraus [1] and Antonio Sedeño-Noda [2]

[1] Network Optimization Department, Zuse Institute Berlin, Takustraße 7, 14195 Berlin, Germany; borndoerfer@zib.de (R.B.); kraus@zib.de (L.K.)
[2] Departamento de Matemáticas, Estadística e Investigación Operativa, Universidad de La Laguna, 38271 San Cristóbal de La Laguna, Santa Cruz de Tenerife, Spain; asedeno@ull.edu.es
* Correspondence: maristany@zib.de

Abstract: The Dynamic Multiobjective Shortest Path problem features multidimensional costs that can depend on several variables and not only on time; this setting is motivated by flight planning applications and the routing of electric vehicles. We give an exact algorithm for the FIFO case and derive from it an FPTAS for both, the static Multiobjective Shortest Path (MOSP) problems and, under mild assumptions, for the dynamic problem variant. The resulting FPTAS is computationally efficient and beats the known complexity bounds of other FPTAS for MOSP problems.

Keywords: multiobjective shortest paths; time dependent shortest paths; multiobjective approximation algorithms; flight planning problem

check for updates

Citation: Maristany de las Casas, P.; Borndörfer, R.; Kraus, L.; Sedeño-Noda, A. An FPTAS for Dynamic Multiobjective Shortest Path Problems. *Algorithms* **2021**, *14*, 43. https://doi.org/10.3390/a14020043

Academic Editor: Frank Werner
Received: 30 November 2020
Accepted: 22 January 2021
Published: 29 January 2021

1. Introduction

We consider in this paper the solution of the Dynamic Multiobjective Shortest Path (Dyn-MOSP) problem that generalizes the standard shortest path problem in two ways. First, each arc of the graph bears more than one attribute (e.g., length, duration, consumption...); this produces the (static) Multiobjective Shortest Path (MOSP) problem. Second, generalizing arc attributes to functions gives rise to the Dyn-MOSP problem. In it, each arc function is evaluated along a path in a dynamic (programming) fashion, i.e., a time dependent function is evaluated based on passed time, a consumption dependent function based on resource consumption etc. We will refer to all arc and path attributes simply as cost. The goal in MOSP problems and Dyn-MOSP problems is then to find paths from a source node to all other nodes in the input graph that are minimal w.r.t. their costs.

Our setting is motivated by Flight Planning Problems (FPP), in which optimal aircraft routes have to be determined in an airway network, considering multiple and dynamic optimization criteria. The most important scenario is the simultaneous minimization of flight time and fuel consumption. The flight time depends on the weather, in particular, the wind, which changes over time and hence depends on the flight time itself. Similarly, the fuel consumption depends on the weight of the aircraft and hence, again, on the consumption so far. Another application with similar dynamics is the routing of electric vehicles through mountainous terrains with varying traffic congestion. Here, the traffic depends on the time of day, and the battery's state of (dis)charge is non-linear.

In multiobjective optimization it is common to refer to optimal solutions using the terms efficient or Pareto optimal. Already the static MOSP problem is known to be intractable because of the possibly exponential cardinality of the solution set that, in this case, contains so called efficient paths (cf. Hansen [1]). This intrinsic difficulty of all multiobjective optimization problems can be circumvented by restricting attention to a polynomially sized subset of efficient solutions with an a priori bound on the quality loss w.r.t. the complete solution set. This idea lead to the development of Fully Polynomial Time Approximation Schemes (FPTAS) for MOSP problems in recent years (cf. Tsaggouris and Zaroliagis [2], Breugem et al. [3], or Bökler and Chimani [4]).

Our aim in this paper is to show that recently developed exact algorithms for MOSP problems can be generalized to also solve Dyn-MOSP instances, given that the arc cost functions fulfill the First In First Out (FIFO) property, i.e., a worse arrival at an arc's tail node never turns out to be beneficial. We then introduce a new FPTAS for MOSP problems. For ease of exposition, we first consider the static case for which—to the best of our knowledge—it features the currently best asymptotic run time. Afterwards, we show that our results carry over to Dyn-MOSP problems if a certain (realistic) assumption on the arc cost functions is made. As usual in the FIFO setting, the asymptotic run time is the same as that of the static version. We finally provide extensive computational evidence of the efficiency of our approach. Indeed, our FPTAS is faster than existing ones for MOSP problems. Moreover and contrary to what was observed in the computational experiments presented in recent publications about FPTAS for static MOSP problems, the new FPTAS avoids the computation of a considerable number of paths in the dynamic costs setting.

1.1. Literature Review

Multiobjective optimization problems and, in particular, MOSP problems, have been extensively investigated in the literature. Very good general introductions are provided by, e.g., Emmrich and Deutz [5], Ehrgott [6], or Ehrgott and Gandibleux [7]. The theoretical foundation and the algorithmic development of MOSP problems are reviewed in Ulungu and Teghem [8], Current and Marsh [9], Skriver [10], Tarapata [11], or Clímaco and Pascoal [12].

In the 1970s, Vincke [13] considered the MOSP problem for the first time, studying two objective functions. This Biobjective Shortest Path problem was also considered by Hansen [1], who came up with the first label-setting algorithm. Serafini [14] showed that the MOSP problem is NP-complete and Martins [15] generalized Hansen's algorithm for the general multiobjective case. Since then, Martins's algorithm has served as the benchmark for solving MOSP problems. Recently, Sedeño and Colebrook [16] and Maristany et. al. [17] devised the Biobjective and Multiobjective Dijkstra algorithms (B/MD-A): new label setting algorithms for MOSP problems that have not only superior computational complexity, but are also efficient in practice.

Turning to approximation, Papadimitriou and Yannakakis [18] set a milestone in the field of approximation algorithms for multiobjective optimization problems. They proved that for a multiobjective optimization problem with d objectives and an $\varepsilon > 0$, a $(1+\varepsilon)$ Pareto curve of size $\mathcal{O}(\frac{4B}{\varepsilon})^{d-1}$ exists. Here, B is the number of bits required to represent the values that the objective functions can take (B is assumed to be polynomially bounded). They also constructed the first general FPTAS for MOSP problems. It was superseded by the method of Tsaggouris and Zaroliagis [2], who presented an FPTAS for the MOSP problem inspired by the classical Bellman Ford algorithm for Shortest Path problems. Their main idea is to subdivide the space of possible path costs into polynomially (in the size of the input and $1/\varepsilon$) many cells and admit just one path per cell. The right choice of the subdivision guarantees that if a path is rejected because its cell is occupied, the quality loss remains bounded. This produces a $(1+\varepsilon)$-cover of the exact set of efficient paths. The idea was picked up by Breugem et al. [3] who managed to pair Martins's algorithm with the subdivision of the outcome space introduced in [2]. The result was an FPTAS for MOSP problems that is worse regarding the theoretical running time, but performs better in computation. Based on this work, Bökler and Chimani [4] recently published an extensive comparison of different label ordering and selecting strategies.

The literature considering MOSP instances with dynamic, also called time-dependent, cost functions is scarce. Kostreva and Lancaster [19] presented an algorithm for non-monotone increasing arc cost functions that does not reduce to Dynamic Programming. Disser et al [20] mention the necessity to tackle this kind of problems on train networks and use Martins's algorithm to solve them without going into details. Our label setting algorithm for Dyn-MOSP problems considers only arc cost functions that fulfill the FIFO property. An extensive analysis of the implications of this condition on the solvability of single-objective Time-Dependent Shortest Path problems is given by Foschini et al. [21].

1.2. Outline

In Section 2, we formulate the Dyn-MOSP problem, and in Section 3, we explain how the Multiobjective Dijkstra Algorithm can be used to solve Dyn-MOSP instances if the arc cost functions fulfill the FIFO property. The analysis of the algorithm's asymptotic run time is done using a black-box dominance check whose complexity varies depending on the number of objectives and the partial order used to define minimality of paths. We use this abstract representation in Section 4 to introduce the new MD-FPTAS. It divides the outcome space of MOSP and Dyn-MOSP instances into polynomially many buckets, each of them allowing the storage of at most one path. The correctness of the resulting FPTAS is first proven for the case of static arc costs. Then, we derive a condition on dynamic arc cost functions that ensures that the MD-FPTAS also works for Dyn-MOSP instances. Finally, in Section 5 we test our algorithms against a state of the art FPTAS for the MOSP problem.

2. Multiobjective Shortest Path Problems

We start this section with a combinatorial definition of the Dyn-MOSP problem. Its components will be explained immediately afterwards.

Definition 1 (Dynamic Multiobjective Shortest Path Problem)**.** *Given a directed graph $G = (V, A)$, a start node $s \in V$, an initial cost vector $\tau_0 \in \mathbb{R}^d_{\geq 0}$, a dimension $d \in \mathbb{N}$, and an arc cost function $c : A \to \mathbb{F}^d$ (see Equation (1)), the d dimensional Dynamic Multiobjective Shortest Path Problem is to find either a minimum or a maximum complete set of efficient (s, v)-paths in G for all $v \in V$ (which type of set is output depends on the partial order used to define the minimality of paths).*

Graph terminology

We refer to the end-nodes of an arc $a \in A$ as the tail and the head nodes of a. Given two nodes u, $v \in V$, a (u, v)-path in G is a sequence $(a_1, \ldots, a_k)$, $k \in \mathbb{N}$, of arcs such that u is the tail node of a_1 and v is the head node of a_k. Additionally, the head node of a_i coincides with the tail node of a_{i+1} for any $i \in \{1, \ldots, k-1\}$. We denote by $p_{|l}$ for any $l \in \{1, \ldots, k\}$ the leading subpath of p consisting of the first l arcs of p.

The arc cost function c

In a d-dimensional Dyn-MOSP instance $\mathcal{I}$, each arc $a \in A$ bears d independent real valued cost functions $c_{a,i} : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, $i \in \{1, \ldots, d\}$; these will be used to propagate costs along the paths. The propagation can be described conveniently in terms of cost propagation functions $\hat{c}_{a,i} := id + c_{a,i} : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, $i \in \{1, \ldots, d\}$. Then, given a cost vector $\tau \in \mathbb{R}^d_{\geq 0}$, its cost component τ_i is propagated along arc a to become $\hat{c}_{a,i}(\tau_i) := \tau_i + c_{a,i}(\tau_i)$, $i \in \{1, \ldots, d\}$, and along path $p = (a_1, \ldots, a_k)$ to become $c_i(p, \tau_i) = \hat{c}_{a_k,i}(\hat{c}_{a_{k-1},i}(\cdots \hat{c}_{a_2,i}(\hat{c}_{a_1,i}(\tau_i))))$, $i \in \{1, \ldots, d\}$. Throughout this paper, we will consider paths starting at a source node s with arbitrary, but fixed, starting costs $\tau_{0,i}$, $i \in \{1, \ldots, d\}$; for such paths let us omit the starting costs $\tau_{0,i}$ from the notation and abbreviate $c_i(p, \tau_{0,i})$ by $c_i(p)$.

Arc costs of this type arise in applications where multidimensional "state flows" evolve through a graph. For example, in the Flight Planning problem that we will discuss in Section 5.1, the costs τ_v at a node v of a path p are interpreted as the aircraft state, i.e., a tuple consisting of the flight time, the fuel consumption, the distance etc. from the source node until v, and $\hat{c}_{v,w}(\tau_v) = \tau_v + c_{(v,w)}(\tau_v)$ are the costs (in this example, the aircraft state) at the successor node w of v in p. We use τ to denote the costs because one of the costs is typically time. Note also that in the static case, the arc cost functions are constant and hence, we can write $c_{a,i}$ instead of $c_{a,i}(\tau)$.

If we denote by $\mathbb{F}$ the set of functions from $\mathbb{R}_{\geq 0}$ to $\mathbb{R}_{\geq 0}$, we can denote the arc cost function c and the arc cost propagation function $\hat{c}$ of $\mathcal{I}$ in vector and function notation by

$$\begin{aligned} c : A &\to \mathbb{F}^d & \hat{c} : A &\to \mathbb{F}^d \\ a &\mapsto c_a := [c_{a,i}]_{i=1}^d. & a &\mapsto \hat{c}_a := [id + c_{a,i}]_{i=1}^d. \end{aligned} \tag{1}$$

Then, given a cost vector $\tau \in \mathbb{R}^d_{\geq 0}$, the costs for traversing an arc a starting with costs τ at a's tail node are $c_a(\tau) := [c_{a,i}(\tau_i)]_{i=1}^d \in \mathbb{R}^d_{\geq 0}$, and costs τ are propagated along arc a to become $\hat{c}_a(\tau) = [\tau_i + c_{a,i}(\tau_i)]_{i=1}^d \in \mathbb{R}^d_{\geq 0}$. Let p be a (u, v)-path with k arcs that starts at u with costs $\tau \in \mathbb{R}^d_{\geq 0}$. Then the costs of any leading subpath $p_{|l}$, $l \in \{2, \ldots, k\}$, of p are recursively given by

$$\begin{aligned} c\Big(p_{|1}, \tau\Big) &:= \hat{c}_{a_1}(\tau) = \tau + c_{a_1}(\tau) \in \mathbb{R}^d_{\geq 0}, \\ c\Big(p_{|l}, \tau\Big) &:= \hat{c}_{a_l}\Big(c\Big(p_{|l-1}, \tau\Big)\Big) = c_{a_l}\Big(c\Big(p_{|l-1}, \tau\Big)\Big) + c\Big(p_{|l-1}, \tau\Big) \in \mathbb{R}^d_{\geq 0}. \end{aligned} \tag{2}$$

Finally, using $p = p_{|k}$, the d-dimensional costs of path p starting with cost τ are denoted by $c(p, \tau)$, or, if the starting state τ is clear, by $c(p)$. In the static case, the costs of p are $c(p) := \sum_{j=1}^k c_{a_j}$, where a_j denotes the jth arc of p.

Efficiency of paths

In Multiobjective Optimization, optimal feasible solutions are called efficient solutions (cf. [6]). The feasible solutions to any MOSP instance are paths and hence, we seek to find efficient (s, v)-paths with initial costs $\tau_0 \in \mathbb{R}^d_{\geq 0}$ for every $v \in V$. Efficiency is defined in terms of a (strict) partial order on $\mathbb{R}^d_{\geq 0}$ (cf. [5]). For our exact Multiobjective Dijkstra algorithm discussed in Section 3, we will use the following notion of efficiency and dominance:

Definition 2 (Dominance)**.** *Given two cost vectors $x, y \in \mathbb{R}^d_{\geq 0}$, x is said to dominate y ($x \preceq_D y$), if and only if $x_i \leq y_i$ for all $i \in \{1, \ldots d\}$. Moreover, x is called efficient, if there is no other cost vector $x' \in \mathbb{R}^d_{\geq 0}$ such that $x' \preceq_D x$.*

This definition carries over to the set of feasible paths in a natural way: given two (s, v)-paths p and q, p is said to $\preceq_D$-dominate q ($p \preceq_D q$) if and only if $c(p) \preceq_D c(q)$. An (s, v)-path is called $\preceq_D$ -efficient if it is not $\preceq_D$ -dominated by any other (s, v)-path. Note that the set of efficient cost vectors of a Dyn-MOSP instance $\mathcal{I}$ is unique. However, for every such point in $\mathbb{R}^d_{\geq 0}$, there might be multiple efficient paths. The maximum complete set of efficient paths is the set that contains all paths whose cost vector is efficient, i.e., the set may contain paths with identical costs. If the widely used strict partial Pareto order is used to define dominance, the output of a Dyn-MOSP instance would be the maximum complete set. This order is the same as the one in Definition 2 but enforces that at least one of the inequalities is strict, hence not allowing x to dominate y if $x = y$. In this paper we compute the minimum complete set of efficient paths that is induced by $\preceq_D$ and allows exactly one efficient path per efficient cost vector.

Example 1. *Figures 1 and 2 show an example of a biobjective Dyn-MOSP instance and how the costs of two paths evolve through the corresponding graph. We assume that at the nodes u and v there are paths (red and green) ending with costs $\tau = (\tau_1, \tau_2)$ and $\tau' = (\tau'_1, \tau'_2)$, respectively, and that these paths are now extended towards node x via node w. The plots to the left and in the middle show the costs for traversing the corresponding arc starting with costs τ and τ', respectively. After the extension along the arcs (u, w) and (v, w) both paths meet at node w with costs $\tau_w = \hat{c}_{a_1}(\tau)$ and $\tau'_w = \hat{c}_{a_2}(\tau')$. The plots on the right show the arc cost functions corresponding to the arc $a_3 = (w, x)$. Furthermore, on the same diagram, the dominance relationship between both paths at*

w becomes clear since $\tau'_{w,1} < \tau_{w,1}$ and $\tau'_{w,2} > \tau_{w,2}$. This means that none of the paths ending at w dominates the other one.

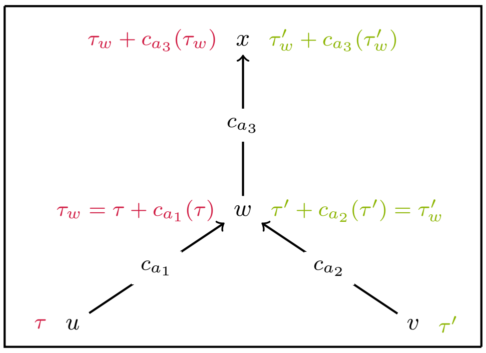

$\tau_w + c_{a_3}(\tau_w)$
x
$\tau'_w + c_{a_3}(\tau'_w)$
c_{a_3}
$\tau_w = \tau + c_{a_1}(\tau)$
w
$\tau' + c_{a_2}(\tau') = \tau'_w$
c_{a_1}
c_{a_2}
τ
u
v
τ'

Figure 1. Graph corresponding to the Dyn-MOSP instance discussed in Examples 1 and 2. The corresponding arc cost functions are shown in Figure 2.

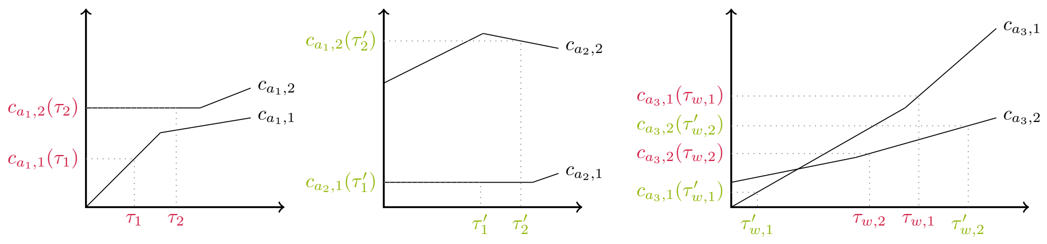


Figure 2. These plots show the two arc cost functions corresponding to the three arcs of the graph in Figure 1: the leftmost functions correspond to the arc $a_1 = (u,w)$; the functions in the middle to the arc $a_2 = (v,w)$; the rightmost functions to the arc $a_3 = (w,x)$.

3. The Multiobjective Dijkstra Algorithm for Dynamic MOSP Problems

In this section, we discuss Algorithm 1, an adaptation of the Multiobjective Dijkstra Algorithm (MD-A) presented in [16,17] that is extended here to work with dynamic costs. We will see that for the dynamics discussed in this paper, the solvability of MOSP and Dyn-MOSP instances mirrors the well known relationship from the single objective case: if a path p dominates a path p' at a node v, their extensions inherit this dominance relationship. This characteristic of the cost functions is known as the FIFO property and defined formally in Section 3.2.

3.1. Description of the Algorithm

Whenever we use arrays in our algorithms, we will use an operator $[\cdot]$ to access its elements. To be able to do this on arrays that are indexed according to the nodes or the arcs of G, we assume that these have unique ids from 0 to $|V|-1$ and from 0 to $|A|-1$, respectively. Then, if for example Δ is an array with one entry per arc in G, $\Delta[a]$ denotes the content of Δ at the position specified by the id of the arc $a \in A$.

Paths and Labels

In Algorithm 1, paths are considered according to the lexicographically increasing costs. A point $x \in \mathbb{R}^d_{\geq 0}$ is said to be lexicographically smaller than a point $y \in \mathbb{R}^d_{\geq 0}$ ($x <_{lex} y$) if and only if $x_i < y_i$ in the first dimension $i \in \{1,\dots,d\}$ in which $x_i \neq y_i$, and we say that a path p is lexicographically smaller than another path p' if $c(p) <_{lex} c(p')$. We will store paths using labels, i.e., by an implicit representation. Let p be an (s,v)-path (starting at s with costs τ_0) whose last arc is $(u,v) \in A$. The label corresponding to p is a tuple $\ell = (v, c_\ell := c(p), \ell_u)$, where ℓ_u is a pointer to the label representing the (s,u)-subpath of p. Note that $c_\ell = \hat{c}_{(u,v)}(c_{\ell_u})$, which, incurring in an abuse of notation that increases the readability, can be put as $c_\ell = \hat{c}_{(u,v)}(\ell_u)$. For every node $v \in V$ the set L_v

contains the labels corresponding to the efficient (s, v)-paths found during the algorithm. We will see that the label sets L_v will only increase; a label $\ell \in L_v$ is therefore also called a permanent label. Additionally, a lexicographically sorted priority queue Q stores at most one tentative label per node. Tentative labels correspond to paths that have been explored during the algorithm but are not yet known to be efficient. For a node v, the label stored in Q corresponds to the lexicographically minimal (s, v)-path that has not yet been made permanent and is not dominated by any label in L_v (we write $L_v \not\preceq_D \ell_v$).

Algorithm 1: Multiobjective Dijkstra Algorithm (MD-A)
Blue code only for the MD-FPTAS described in Section 4.

```
Input                     : Digraph $G = (V, A)$, Arc Costs $c : A \to \mathbb{F}^d$, Node $s \in V$, Initial costs $\tau_0 \in \mathbb{R}^d_{\geq 0}$.
Additional FPTAS Input    : Vector of approximation ratios $\varepsilon \in \mathbb{R}^{d-1}_{\geq 0}$.
// The output is a minimum complete set in the exact scenario and a $(1+\varepsilon)$-cover (see Section 4) if the algorithm is used as an
   FPTAS.
Output                    : Labels $L := \bigcup_{v \in V} L_v$.
1  Prio. Queue $Q \leftarrow \emptyset$;
2  for $v \in V$ do Permanent labels $L_v \leftarrow \emptyset$ ;
3  for $a \in A$ do $lastProcessedLabel[a] \leftarrow 0$;
4  Label $l_{init} \leftarrow (s, \tau_0, \texttt{NULL})$;                                   /* Only for MD-A. */
   Label $l_{init} \leftarrow (s, \tau_0, \texttt{NULL}, \text{pos}(\tau_0))$;                 /* Only for MD-FPTAS. */
5  $Q \leftarrow Q \cup \{l_{init}\}$;
6  $L_s \leftarrow L_s \cup \{l_{init}\}$;
7  while $Q \neq \emptyset$ do
8      Label $\ell^*_v \equiv (v, c_{\ell^*_v}, \ell_{pred}) \leftarrow$ Label with lexicographically minimal costs in $Q$;
9      $L_v \leftarrow L_v \cup \{\ell^*_v\}$ ;                                             /* Make $\ell^*_v$ permanent. */
10     $\ell^{new}_v \leftarrow nextCandidateLabel(v, lastProcessedLabel, \cup_{u \in \delta^-(v)} L_u)$ ;   /* Only for MD-A. */
       $\ell^{new}_v \leftarrow nextCandidateLabel(v, lastProcessedLabel, \cup_{u \in \delta^-(v)} L_u, \varepsilon)$;   /* Only for MD-FPTAS. */
11     if $\ell^{new}_v \neq \texttt{NULL}$ then $Q \leftarrow Q \cup \{\ell^{new}_v\}$ ;
12     for $w \in \delta^+(v)$ do
13         $Q \leftarrow propagate(\ell^*_v, w, Q, L_w)$ ;                                    /* Only for MD-A. */
           $Q \leftarrow propagate(\ell^*_v, w, Q, L_w, \varepsilon)$;                        /* Only for MD-FPTAS. */
14 return $L_v$ for all $v \in V$;
```

We present the pseudocode of Algorithm 1 in an abstract way, i.e., without specifying the partial order used to define dominance. Instead, we call a function `BLACK_BOX_DOM` that returns `TRUE` only if a newly found candidate path is dominated by the set of permanent paths at a node, i.e., if this set contains a path that dominates the candidate path. This allows us to use the same pseudocode for the exact algorithm presented in this section and for the MD-FPTAS presented in Section 4. Algorithms 2 and 3 show the function `BLACK_BOX_DOM` used for the exact MD-A.

Algorithm 2: `BLACK_BOX_DOM`(L_v, ℓ_v) for MD algorithm, $d = 2$.

```
1  return $L_v[|L_v| - 1] \preceq_D \ell_v$
```

Algorithm 3: `BLACK_BOX_DOM`(L_v, ℓ_v) for MD algorithm, $d \geq 3$.

```
1  for $\ell \in L_v$ do
2      if $\ell \preceq_D l_v$ then return TRUE;
3  return FALSE
```

Iterations

At the beginning, a tentative label $\ell_{init} = (s, \tau_0, NULL)$ is inserted into Q. The main loop of the algorithm retrieves labels from Q; makes them permanent, possibly inserting new labels into Q for the node for which a label was retrieved; propagates their costs, possibly inserting new labels into Q for the successors of this node, and ends once Q becomes empty. Iterations start with the extraction of the lexicographically minimal label

ℓ_v^* from Q, which is added to the end of L_v since it is guaranteed to correspond to an efficient path. Since this is the only way labels are added to the lists L_v, these sets are also sorted in lexicographically increasing order. The main trick in the algorithm is that the queue Q stores at any time at most one label for any node, namely, a lexicographically minimal one that is not dominated by any permanent label at the corresponding node. This makes the algorithm efficient by eliminating any search for non-dominated labels in Q–a tedious task that so far was common to all label setting MOSP algorithms in the literature.

After making ℓ_v^* permanent, each iteration pursues two main tasks. The first task is to find a new tentative label for v. This is necessary since the priority queue Q stores at most one label for node v at any time. If such a label exists,

$$l_v^{new} := \operatorname{arglexmin}_{\substack{\ell_u \in L_u,\\ u \in \delta^-(v)}} \left\{ \ell = \left(v,\, \hat{c}_{(u,v)}(\ell_u),\, \ell_u\right) \mid L_v \not\preceq_D \ell \right\} \tag{3}$$

is not dominated by any label in L_v and is lexicographically minimal among the labels gained from the extension of labels in L_u, $u \in \delta^-(v)$, along the arc (u,v) (see nextCandidateLabel). This label l_v^{new} has to be built every time a label for v is extracted from Q; this is the price for storing at most one label per node. However, it is not necessary to traverse the entirety of the lists L_u every time. Instead, labels in L_u whose extension along (u,v) were already dominated at L_v the last time a new label for v was built, do not have to be considered. To take advantage of this observation, we store the indices `lastProcessedLabel`(u,v) for any arc (u,v). These refer to the last checked position in L_u when looking for a label for v and define where a search for v's next candidate label has to start. If a new label l_v^{new} for v is found, it is added to Q. The second task in each iteration is the propagation of ℓ_v^* along the outgoing arcs of v. The algorithm builds the labels $\ell_w = (w, \hat{c}_{(v,w)}(\ell_v^*), \ell_v^*)$ for every $w \in \delta^+(v)$ and adds them to Q only if $L_w \not\preceq_D \ell_w$. If ℓ_w is lexicographically smaller than w's current label in Q, the latter is replaced by ℓ_w. In case there is no label for w in Q, ℓ_w is just inserted into Q (see propagate).

Procedure nextCandidateLabel
Blue lines only for the MD-FPTAS described in Section 4.

Input	: Node v, Indices *lastProcessedLabel*, Permanent labels L.
Additional FPTAS Input	: Vector of approximation ratios $\varepsilon \in \mathbb{R}^{d-1}_{\geq 0}$.
Output	: New lex. min., non-dominated label for v, if one exists.

```
1  Label $\ell_v^{new} \leftarrow (v, (\infty, \dots, \infty), NULL)$;
2  for $u \in \delta^+(v)$ do
3      for $k = lastProcessedLabel[(u,v)]$ to $|L_u|$ do
4          Label $\ell_u \leftarrow L_u[k]$; ;                          /* $k^{th}$ label in $L_u$ */
5          Cost $c_{new} \leftarrow \hat{c}_{(u,v)}(\ell_u)$;
6          Label $\ell \leftarrow (v, c_{new}, \ell_u)$);                  /* Only for MD-A. */
           Label $\ell \leftarrow (v, c_{new}, \ell_u, \text{pos}_\varepsilon(c_{new})))$;   /* Only for MD-FPTAS. */
           // Next time a label for $v$ is searched in $L_u$, the search starts where the current one ends.
7          $lastProcessedLabel[(u,v)] \leftarrow k$;
8          if !BLACK_BOX_DOM($L_v, \ell$) then
               // $\ell$ is non-dominated. Additionally, it has to be lex. minimal.
9              if $\ell \prec_{lex} \ell_v^{new}$ then $\ell_v^{new} \leftarrow \ell$;
10             break;
11 if $c_{\ell_v^{new}} \neq \infty$ then return $\ell_v^{new}$;
12 return $NULL$;
```

Procedure propagate
Blue lines only for the MD-FPTAS described in Section 4.

```
Input                    : Label $\ell_v$, Node $v \in \delta^+(u)$, Prio. Queue $Q$, Permanent labels $L$.
Additional FPTAS Input   : Vector of approximation ratios $\varepsilon \in \mathbb{R}^{d-1}_{\geq 0}$.
Output                   : Updated Prio. Queue $Q$.
// Cost propagation: the cost functions of the arc $(v,w)$ are evaluated at the costs of the $(s,v)$-path represented by $\ell_v$. The
   resulting $(s,w)$-path has costs $c_{new}$.
1 Cost $c_{new} \leftarrow c_{\ell_v} + c_{(v,w)}(c_{\ell_v})$;
2 Label $\ell_w \leftarrow (w, c_{new}, \ell_v)$;                                   /* Only for MD-A. */
  Label $\ell_w \leftarrow (w, c_{new}, \ell_v, \text{pos}_\varepsilon(c_{new}))$;     /* Only for MD-FPTAS. */
3 if !BLACK_BOX_DOM($L_w, \ell_w$) then
4     if there is no label for $w$ in $Q$ then
5         $Q \leftarrow Q \cup \{\ell_w\}$;
6     else if $\ell_w$ is lex. smaller than $w$'s heap label then
7         Replace $w$'s label in $Q$ with $\ell_w$; ;              /* This is a decrease_key operation on $Q$. */
8 return $Q$;
```

Example 2. *Going back to the example shown in Figure 2, assume that the labels ℓ with costs τ at u and ℓ' with costs τ' at v are in the queue Q and $\tau <_{lex} \tau'$. Then, ℓ is extracted earlier from Q and in the same iteration, Procedure propagate is called to extend ℓ along the arc $a_1 = (u, w)$ and obtain the tentative label $\ell_w = (w, \tau_w = \hat{c}_{(u,w)}(\tau), \ell)$. We assume that no label for w exists, neither in Q nor in L_w, and hence, ℓ_w is added to Q. Suppose that in the next iteration of Algorithm 1, the label ℓ' with costs τ' at v is extracted. During its extension towards w in propagate, the label $\ell'_w = (w, \tau'_w = \hat{c}_{(v,w)}(\tau'), \ell')$ is created. As we can see in the rightmost plot in Figure 2, $\tau'_{w,1} < \tau_{w,1}$ and hence, $\tau'_w <_{lex} \tau_w$. Since the conditions in Lines 3 and 6 of propagate are fulfilled, the label ℓ_w is replaced by ℓ'_w, which becomes the new label for w in Q. This however does not discard ℓ_w! Some iterations later, ℓ'_w is extracted from Q and in the same iteration, nextCandidateLabel is called to build a new tentative label for w in Q out of the permanent labels of the predecessor nodes of w. It is in this search where ℓ_w is found again and since it is not dominated (Line 8 of nextCandidateLabel) by any permanent label at w (currently L_w only contains ℓ'), it is added to Q. Indeed, as shown in the rightmost plot in Figure 2, $\tau_w \npreceq_D \tau'_w$ and $\tau'_w \npreceq_D \tau_w$ meaning that both labels have to be added to L_w. Later, the label with costs τ_w is extracted from Q and finally made permanent. The extension of both labels towards x along (w, x) works in the same way.*

3.2. Correctness

As noted in the beginning of the section, we require the arc cost functions $c_{a,i}$, to fulfill the FIFO property:

Definition 3 (First-In-First-Out Property). *The i_{th} cost function $c_{a,i}$ of an arc $a \in A$ fulfills the FIFO property if and only if for two cost components τ_i, τ'_i with $\tau_i < \tau'_i$, there holds $\hat{c}_{a,i}(\tau_i) \leq \hat{c}_{a,i}(\tau'_i)$.*

The correctness of Algorithm 1 relies on the Bellman-Principle [22] of optimality. In the context of MOSP problems, it states that an efficient (s, v)-path contains only efficient subpaths. Given that the arc cost functions are FIFO, Bellman's principle holds for efficient paths of the Dyn-MOSP problem. In particular, the following statement holds.

Lemma 1. *Given a Dyn-MOSP instance with FIFO arc cost functions, consider two (s, v)-paths p, p' such that $c(p) \preceq_D c(p')$. Then, for the extensions of p and p' along a (v, w)-path q in G there holds $c(p \circ q) \preceq_D c(p' \circ q)$.*

Proof. Since $c(p) \preceq_D c(p')$, we know that $c_j(p) \leq c_j(p')$ for any $j \in \{1, \dots, d\}$. Due to the FIFO property of the arc cost functions, this implies that after q's first arc, say $a \in A$, there holds

$$c(p \circ a) = \hat{c}_a(c(p)) \leq \hat{c}_a(c(p')) = c(p' \circ a).$$

This argument can be repeated along every arc of q until we reach the paths' end point, implying that $c(p \circ q) \preceq_D c(p' \circ q)$. □

The consequence of Lemma 1 is that during Algorithm 1, dominated labels/paths can be neglected since they will not become efficient later on. Hence, for a Dyn-MOSP instance whose arc cost function has the FIFO property in every dimension, Algorithm 1 computes a minimal complete set of efficient paths. The rest of the correctness proof proceeds along standard lines as in the static case (cf. [16,17]).

3.3. Complexity

The running time of Algorithm 1 is dominated by the running time of nextCandidate-Label and propagate, that both depend on the complexity of the oracle `BLACK_BOX_DOM`. It implements the dominance checks that are applied in the different versions of the MD-A that we discuss throughout the paper. In the exact biobjective algorithm, the dominance check (see Algorithm 2) can be implemented in constant time. The reason is that the sets L_v are sorted in lexicographically increasing order and contain only efficient labels. Indeed, the first cost component is sorted in increasing order, while the second cost component is sorted in decreasing order. Thus, a new tentative label that is lexicographically greater than all elements of L_v must only be compared with the last element. This observation seems to be not very widespread in the literature, but given the possibly exponential number of efficient paths at any node, the possibility of checking dominance in constant time has a big impact in theory and practice (cf. [16,17,23]). For $d \geq 3$ the complexity is linear in the number of labels contained in L_v, since in the worst case, the tentative label has to be compared with all existing ones (see Algorithm 3). In our analysis, we will denote the complexity of the dominance check, i.e., of the function `BLACK_BOX_DOM`, by $\mathcal{C}$. We assume that Q is a Fibonacci–Heap [24] to get constant time complexity for the update and the insertion of labels. The label extraction is performed in $\mathcal{O}(\log n)$, since the size of Q is at most n. We set $\mathcal{N}_{\max} := \max_{v \in V} |L_v|$ to be the maximal number of labels at a single node at the end of the algorithm. $\mathcal{N} := \sum_{v \in V} |L_v|$ is the total number of computed efficient labels.

Remark 1. *Note that we assume that the evaluation of the arc cost functions can be done in constant time. This is for example the case, when the functions are piecewise linear/constant and the breakpoints are distributed equidistantly. This assumption is common in the literature on Time-Dependent Shortest Paths problems. However, note that logarithmic running times have to be assumed if the breakpoints are spaced without regularity (binary search to find the neighboring breakpoints), or even worse complexities if the functions cannot be easily parametrized. Additionally, we assume that the dimension d of the problem is a constant.*

Complexity of nextCandidateLabel

For a node $v \in V$ nextCandidateLabel is called every time a label for v is made permanent, i.e., $|L_v| + 1$ times. The use of the `lastProcessedLabel` pointers for every arc $(u, v) \in A$ guarantees that the list L_u of permanent labels at each predecessor node u of v is traversed exactly once. During each call, a dominance check between the extension along (u, v) of the considered predecessor labels and L_v is performed. This results in a running time of

$$\mathcal{O}\left(\left(\sum_{u \in \delta^-(v)} |L_u| + |L_v| + 1\right)\mathcal{C}\right),$$

which summing over all nodes $v \in V$, can be put as $\mathcal{O}(m\mathcal{N}_{\max}\mathcal{C})$.

Complexity of propagate

In total, $|L_u|$ labels are propagated along an arc $(u,v) \in A$. Every time a label is propagated from u along (u,v), we have to check if the resulting label is dominated by any label in L_v. Summing over all nodes, we get an overall complexity of

$$\mathcal{O}\left(\sum_{u \in V} |\delta^+(u)||L_u|\mathcal{C}\right) = \mathcal{O}(m\mathcal{N}_{\max}\mathcal{C}).$$

Note that since Q contains at most one label per node, we can have constant time access (e.g., via a pointer) to a node's heap label. We make use of it in Lines 4 and 6 of propagate.

Algorithm 1 performs one iteration per permanent label, i.e., $\mathcal{N}$ iterations in total. In addition to nextCandidateLabel and propagate, a label is extracted in every iteration. All in all, the running time of Algorithm 1 is

$$\mathcal{O}(\mathcal{N}\log n + 2m\mathcal{N}_{max}\mathcal{C}) = \mathcal{O}(\mathcal{N}\log n + m\mathcal{N}_{max}\mathcal{C}) \subseteq \mathcal{O}(\mathcal{N}_{max}(nlog(n) + m\mathcal{C})). \quad (4)$$

In Table 1, we list the complexity $\mathcal{C}$ of the dominance check for the different variants of the MD-A that we consider during the paper. It is clear that the space consumption of Algorithm 1 is $\mathcal{O}(N+m)$ if we assume that d is fixed. More in depth discussions of these running time bounds for the exact versions of Algorithm 1 can be found in [16,17].

Table 1. Complexity $\mathcal{C}$ of the dominance checks.

	Exact MD-A (Section 3.3)	**MD-FPTAS (Section 4.2)**	**MD-FPTAS_*T* (Section 4.2.1)**
$d = 2$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
$d \geq 3$	$\mathcal{O}(L_v) \subseteq \mathcal{O}(N_{max})$	$\mathcal{O}(\mathcal{T})$	$\mathcal{O}(1)$

4. A New FPTAS for the Multiobjective Shortest Path Problem

In this section, we introduce the MD-FPTAS, a new FPTAS for MOSP problems. Basically the MD-FPTAS is Algorithm 1 with a new partial order on the cost vectors, i.e., with a new notion of efficiency. As used for the first time in [2], the main idea is to divide the outcome space into a polynomial number of cells, each of them holding at most one path. The correspondence of a path to a cell is defined via the path's cost vector. For ease of exposition, we will focus on the static version first. In Section 4.1, we show that the FPTAS also works on Dyn-MOSP instances if some extra assumptions are made about the arc cost functions.

Consider an $\alpha \in \mathbb{R}^d_{\geq 1}$ and two (s,v)-paths p, q. Then, p α-covers q if $c_j(p) \leq \alpha_j c_j(q)$ for all $j \in \{1,\ldots,d\}$. Let $\mathcal{X}$ be the set of all feasible paths of a (Dyn-)MOSP instance. A subset $\tilde{\mathcal{X}} \subseteq \mathcal{X}$ is an α-cover of $\mathcal{X}$ if and only if for any $x \in \mathcal{X}$ there is a $\tilde{x} \in \tilde{\mathcal{X}}$ that α-covers x. If we denote the all-ones vector (initially in d dimensions) by $\mathbb{1}$, a Fully Polynomial Time Approximation Scheme (FPTAS) for the (Dyn)-MOSP problem is a family of algorithms that computes, for any $\varepsilon > 0$, a $(\mathbb{1} + \varepsilon\mathbb{1})$-cover of $\mathcal{X}$. Furthermore, its space requirements and running time are polynomial in the size of the used instance as well as in $\frac{1}{\varepsilon}$ (note that we could define ε as a vector and get different approximation ratios for every component. However, this would not give deeper insights into the used techniques and is also not used in the experiments presented in Section 5). Additionally, we assume that $\varepsilon \leq 1$ such that $\ln(1+\varepsilon) = \Theta(\varepsilon)$ and set $c_i^{\min} := \min_{a \in A} c_{a,i}$, $c_i^{\max} := \max_{a \in A} c_{a,i}$, and $C_i = \frac{c_i^{\max}}{c_i^{\min}}$ for $i \in \{1,\ldots,d\}$.

As in the FPTAS presented in [2,3], ours is exact in one dimension. In our case, we choose the first cost component to be exact meaning that we compute a $(1, \mathbb{1} + \varepsilon\mathbb{1})$-cover of $\mathcal{X}$, with $\mathbb{1} \in \mathbb{R}^{d-1}$. The MD-FPTAS assigns a $(d-1)$-dimensional tensor T_v to each node $v \in V$. The entries of each tensor in the $(j-1)^{th}$ dimension, $j \in \{2,\ldots d\}$, are indexed from 0 to $\lfloor \log_r(nC_j) \rfloor$, for an approximation factor $r = (1+\varepsilon)^{\frac{1}{n-1}} \in \mathbb{R}_{\geq 1}$. At most, one

(s, v)-path is stored in each entry of T_v. With our choice of r, this guarantees the desired polynomial running time and space consumption.

The bound on the space consumption can be derived from the size of T_v: the length of $T_{v,j}$ becomes $\lfloor \frac{n}{\varepsilon} \log(nC_j) \rfloor$ for any node $v \in V$ and $j \in \{2, \dots, d\}$ such that, in total, T_v stores at most $\prod_{i=2}^{d} |T_{v,i}|$, a term that is polynomial in the size of the graph and in $\frac{1}{\varepsilon}$. If we allow our FPTAS to only store one (simple) path per entry in the vector, we guarantee the polynomial space consumption of the MD-FPTAS. The position of a (s, v)-path $p \in \mathcal{X}$ in T_v is computed using the function $\operatorname{pos} : \mathbb{R}^d_{\geq 0} \to \mathbb{N}_0^{d-1}$ which is defined componentwise as

$$\operatorname{pos}_j(c(p)) := \begin{cases} 0 & \text{, if } c_j(p) = 0 \\ 1 + \left\lfloor \log_r \frac{c_j(p)}{c_j^{\min}} \right\rfloor & \text{, else} \end{cases} \quad , j \in \{2, \dots d\}. \tag{5}$$

This justifies the size of the tensors since efficient (simple) paths have at most $(n-1)$ arcs and hence, $c_j(p) \leq (n-1)c_j^{\max}$.

We incorporate the subdivision of the outcome space into Algorithm 1 and, in particular, into the dominance checks made therein. To do so, we consider the blue parts in the pseudocodes from Section 3. In the MD-FPTAS any label ℓ representing a path p additionally stores $\operatorname{pos}(c(p)) = \operatorname{pos}(c_\ell)$. For ease of exposition, we will write $\operatorname{pos}(p)$ and $\operatorname{pos}(\ell)$ from now on. Labels continue to be sorted lexicographically in the priority queue Q but the dominance checks are done using the labels' pos values: if two distinct labels ℓ, ℓ' at a node $v \in V$ fulfill $c_{\ell,1} \leq c_{\ell',1}$ and $\operatorname{pos}_j(\ell) \leq \operatorname{pos}_j(\ell')$ for $j \in \{2, \dots d\}$, we say that ℓ pos-dominates ℓ' and write $\ell \preceq_{\text{pos}} \ell'$. The new checks are shown in Algorithms 4 and 5. No further modifications w.r.t. Algorithm 1 are needed to obtain the MD-FPTAS from the MD-A. Hence, the MD-FPTAS makes labels permanent after extracting them from the heap. It is guaranteed that at the moment of extraction of a label ℓ_v at a node $v \in V$, there is no label ℓ'_v in L_v such that $\ell'_v \preceq_{\text{pos}} \ell_v$ and thus, no two labels with the same pos values will be stored, i.e., nextCandidateLabel and propagate.

Algorithm 4: `BLACK_BOX_DOM`(L_v, ℓ_v) for MD-FPTAS, $d = 2$

```
1 return $L_v[|L_v| - 1] \preceq_{\text{pos}} l_v$
```

Algorithm 5: `BLACK_BOX_DOM`(L_v, ℓ_v) for MD-FPTAS, $d \geq 3$

```
1 for $\ell \in L_v$ do
2     if $\ell \preceq_{\text{pos}} \ell_v$ then return TRUE;
3 return FALSE
```

The following example shows how efficient labels can be rejected by the MD-FPTAS if they are pos-dominated by permanent labels.

Example 3. *Figure 3 visualizes the situation at the end of each of three subsequent iterations of MD-FPTAS. As in Examples 1 and 2, the illustrated graph is a subgraph of a larger graph with* $n = 10$ *nodes and we set* $\varepsilon = 0.5$. *To illustrate how our FPTAS works, dynamic arc cost functions are not needed. Instead, each arc has* $d = 2$ *static cost components. Labels are represented only by their cost vector; their correspondence to nodes is made clear by their positioning. The example starts with a permanent label with costs* $(80, 131)$ *at node x and tentative labels for u and v in Q with costs* $(100, 100)$ *and* $(100, 101)$*, respectively.*

In the first iteration, the label for u is extracted from Q and no new candidate label for u is found. Then, during the propagation of the label $(100, 100)$*, the resulting label with costs* $(120, 120)$ *at node w is inserted into Q. In the second iteration, the label with costs* $(101, 100)$ *is extracted from the heap and no new candidate label for v is found. This label is then propagated to node w, where the resulting tentative label with costs* $(140, 119)$ *is rejected, as w's current heap label has costs* $(120, 120)$ *and is lexicographically smaller. So far, the MD-A and the MD-FPTAS would have done exactly the same. In the third iteration, the label with costs* $(120, 120)$ *at node w is* *extracted from the heap. When nextCandidateLabel is called, the extension of v's permanent label towards w yields costs* $(140, 119)$*. Hence, it would not be dominated by the new permanent label with costs* $(120, 120)$ *at w in the exact scenario. However, in the MD-FPTAS, it is rejected since* $120 < 140$ *and* $\text{pos}(120) = 107 = \text{pos}(119)$*. The iteration continues and considers the tentative label with costs* $(130, 130)$ *at node x. It is also rejected despite it would be made permanent in the exact scenario, since* $80 < 130$ *and* $\text{pos}(130) = 109 = \text{pos}(131)$*.*

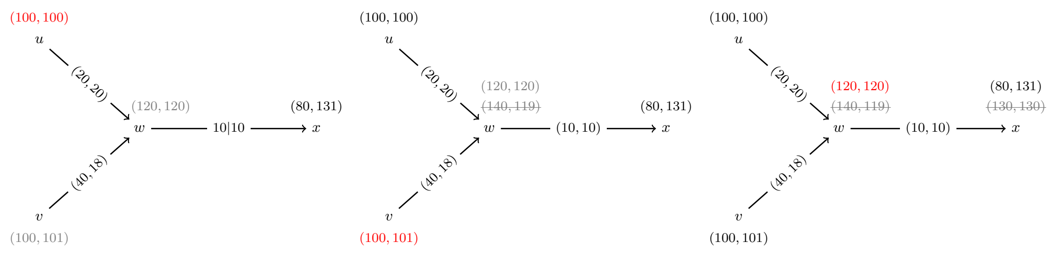


Figure 3. Three consecutive iterations of the MD-FPTAS. The extracted label ℓ^* in every iteration is marked in red, the permanent labels $\ell \in L_v$ in black, and the tentative labels generated in nextCandidateLabel or propagate in grey.

The following Lemma holds for any version of Algorithm 1 presented so far. We need it to prove the correctness of the MD-FPTAS. We denote the set of permanent labels at node v found until iteration $i \in \{1, \dots, \mathcal{N}\}$ (including the i^{th} iteration) by L_v^i. Thus, $L_v \equiv L_v^{\mathcal{N}}$ is used only to denote the final set of permanent labels at v.

Lemma 2. *A label ℓ_v for a node $v \in V$ is considered at most $|L_v| + 1$ times before it is made permanent or finally discarded. If it is discarded, there is a permanent label in L_v that (pos-)dominates ℓ_v.*

Proof. Let ℓ_u be the permanent predecessor label of ℓ_v at node $u \in \delta^-(v)$. ℓ_v is considered for the first time during a call to propagate in the iteration $i \in \{1, \dots, \mathcal{N}\}$ in which ℓ_u is extracted from H and added to L_u^i. Let $k \in \{i+1, \dots, \mathcal{N}\}$ be the next iteration wherein a label ℓ'_v for node v is extracted from Q. If $\ell'_v = \ell_v$, we are done since ℓ_v was considered just once before it is made permanent. In case $\ell'_v \neq \ell_v$, ℓ_v was either rejected in iteration i because a lex. smaller label for v existed in Q or ℓ_v was replaced in Q by a lex. smaller label for v (Line 7 of propagate) that was not (pos -)dominated by any permanent label at v. Note that multiple such updates to v's heap label could have happened until ℓ'_v is extracted and made permanent. The k^{th} iteration proceeds with a call to nextCandidateLabel, where at least one permanent label per predecessor node of v is considered. Since the current iteration is the first time that nextCandidateLabel is called for v since l_u's insertion, `lastProcessedLabel`(u, v) points at a label in L_u^k that is not after l_u. We want to prove an upper bound on the number of times that ℓ_v is considered, so we assume w.l.o.g. that ℓ_u is considered during the current search for v's new label in Q, i.e., `lastProcessedLabel`(u, v) advances at least until ℓ_u's position in L_u^k. Hence, ℓ_u is extended along the arc (u, v), generating ℓ_v as a candidate to enter Q in iteration k. According to Line 8 in nextCandidateLabel, `lastProcessedLabel`(u, v) is increased if there is a label in L_v^k that (pos-)dominates ℓ_v. If this happens, later searches for a new tentative label for v no longer consider ℓ_u as a possible predecessor label and hence, ignore ℓ_v. In case $L_v^k \not\preceq_D \ell_v$, `lastProcessedLabel`$(u, v)$ will not be altered and the next search for a new tentative label for v will consider ℓ_v again. Since such searches only happen when a label for v is made permanent, ℓ_v will be considered at most $|L_v|$ times during calls to nextCandidateLabel. $\square$

The following theorem proves the correctness of the MD-FPTAS for the static case. Its proof is similar to the one given in [3]. Recall that efficient paths can have at most $n - 1$ arcs since the arc cost functions are positive.

Theorem 1. *Consider a node $v \in V$ and an efficient (s,v)-path $p^* = (a_1, \dots, a_k)$. Then, the MD-FPTAS finds an (s,v)-path $\tilde{p}$ s.t.*

$$c(\tilde{p}) \leq r^k c(p^*).$$

Proof. We prove the statement by induction over k, the number of arcs of p^*. W.l.o.g. we assume that no parallel arcs exist in G. In the base case, we consider an efficient single-arc path $p^* = ((s,v))$. In the first iteration of Algorithm 1, the label ℓ^* corresponding to p^* will be added to Q during propagate. Consider the first iteration in which a label ℓ for node v is extracted from Q. If $\ell = \ell^*$, we are done since ℓ^* itself is made permanent. In case $\ell \neq \ell^*$, Lemma 2 implies that ℓ^* will be made permanent later or be discarded. If it is discarded, Algorithms 4 and 5 guarantee the existence of a permanent label $\tilde{\ell} \in L_v$ corresponding to an (s,v)-path $\tilde{p}$ such that

$$c_1(\tilde{p}) \leq c_1(p^*) \tag{6a}$$

and

$$\text{pos}(\tilde{p}) \leq \text{pos}(p^*). \tag{6b}$$

For $j \in \{2, \dots, d\}$, we can derive $\log_r(c_j(\tilde{p})) - 1 \leq \log_r(c_j(p^*))$ from (6b) and this in turn can be restated as $c_j(\tilde{p}) \leq r c_j(p^*)$, which, coupled with (6a), proves the statement. In the induction hypothesis, we assume that

$$c(\tilde{p}) \leq r^{k-1} c(p^*), \tag{7}$$

holds for any $k \in \{2, \dots n-1\}$ and efficient paths p^* with $k-1$ arcs.

Induction Step: Let p^* be an efficient (s,v)-path with k arcs and let (u,v) be its last arc. Due to subpath efficiency, the (s,u)-subpath p^*_u of p^* is efficient. In addition, the induction hypothesis guarantees the existence of a path $\tilde{p}_u$ with corresponding permanent label $\tilde{\ell}_u$ such that (7) holds for $\tilde{p}_u$ and p^*_u. When $\tilde{\ell}_u$ is extracted and made permanent, the label $\tilde{\ell} := (v, c_{\tilde{\ell}_u} + c_{(u,v)}, \tilde{\ell}_u)$ is analyzed in propagate. For the (s,v)-path $\tilde{p}$ corresponding to $\tilde{\ell}$, we have

$$c(\tilde{p}) = c(\tilde{p}_u) + c_{(u,v)} \overset{\text{Equation}(7)}{\leq} r^{k-1} c(p^*_u) + c_{(u,v)} \leq r^{k-1} c(p^*). \tag{8}$$

From the proof of the base case and from Lemma 2, we know that $\tilde{\ell}$ is going to either be made permanent in a later iteration or be discarded. In case $\tilde{\ell}$ is made permanent, we have $c(\tilde{p}) \leq r^{k-1} c(p^*) \leq r^k c(p^*)$ and we are done. If $\tilde{\ell}$ is discarded, there exists a permanent label $\ell \in L_v$ corresponding to an (s,v)-path p such that $c_1(p) \leq c_1(\tilde{p})$ and $\text{pos}(p) \leq \text{pos}(\tilde{p})$. The latter inequality implies $c_j(p) \leq r c_j(\tilde{p})$ for $j \in \{2, \dots, d\}$ and, for $j = 1$, $c_1(p) \leq r c_1(\tilde{p})$ is trivially given. Combining this with Euqation (8), we get

$$\frac{1}{r} c(p) \leq c(\tilde{p}) \leq r^{k-1} c(p^*) \iff c(p) \leq r^k c(p^*),$$

which finishes the proof. □

4.1. FPTAS for Dyn-MOSP Problems

From now on we restrict the set $\mathbb{F}$ of arc cost functions in Dyn-MOSP instances to contain only continuous, piecewise linear FIFO functions. In this case, the proof of Theorem 1 works if the intercepts of the affine functions describing the pieces of the arc cost functions are non-negative. Note that in the proof of Theorem 1 we needed the arc cost vectors only in Equation (8).

Using the notation for dynamic cost, what needs to hold is

$$c(\tilde{p}) = c(\tilde{p}_u) + c_{(u,v)}(\tilde{p}_u) \overset{!}{\leq} r^{k-1} c(p^*_u) + r^{k-1} c_{(u,v)}(p^*_u) = r^{k-1} c(p^*). \tag{9}$$

To prove the following Lemma, we consider a function $f \in \mathbb{F}$ with $k \in \mathbb{N}$ breakpoints and describe the affine functions that build the pieces of f by $\text{aff}_i(x) := a_i x + b_i$, $i \in \{1, \dots, k-1\}$.

Lemma 3. *Let $f \in \mathbb{F}$ be a piecewise affine function with $k \in \mathbb{N}$ breakpoints and $\alpha \in \mathbb{R}_{>1}$ a constant. Then, for points $x, y \in \mathbb{R}_{\geq 0}$ with $x \leq \alpha y$ there holds $x + f(x) \leq \alpha(y + f(y))$ if the intercepts b_i of the affine functions building f are non-negative for all $i \in \{1, \dots, k-1\}$.*

Proof. We consider three different cases to prove the statement.

Case 1: $f(x) \leq f(y)$. Since $\alpha > 1$, we have $f(x) \leq \alpha f(y)$. Together with $x \leq \alpha y$ this proves the statement.

Case 2: $x < y$ and $f(x) \geq f(y)$. In this case, the FIFO property and $\alpha > 1$ can be used to get:

$$x + f(x) \leq y + f(y) \leq \alpha y + \alpha f(y).$$

Case 3: $y < x$ and $f(x) > f(y)$. Let aff_i be the affine function with $\text{aff}_i(y) = f(y)$ and aff_j the one with $\text{aff}_j(x) = f(x)$. There holds $i \leq j$ and we define aff_l, $l \in \{i, \dots, j\}$, to be the affine function corresponding to the steepest piece of f between y and x, i.e., the one with the biggest a_l. This choice implies $f(x) \leq \text{aff}_l(x)$ and $\text{aff}_l(y) \leq f(y)$. Additionally, as for any affine function with positive intercept we have $\text{aff}_l(\alpha y) \leq \alpha(a_l y + b_l) = \alpha\, \text{aff}_l(y)$. All in all, we can conclude

$$f(x) \leq \text{aff}_l(x) \leq \text{aff}_l(\alpha y) \leq \alpha\, \text{aff}_l(y) \leq \alpha f(y).$$

Together with $x \leq \alpha y$ this proves the statement. □

Now we set $c_{(u,v),j} = f$, $c_j(\tilde{p}) = x$, $c_j(p^*) = y$, and $r^{k-1} = \alpha$ to get that Equation (9) holds under the given assumptions. This enables us to formulate our main Theorem:

Theorem 2 (FPTAS for Dyn-MOSP problems)**.** *Let $\mathcal{I}$ be a Dyn-MOSP instance with continuous piecewise linear and positive arc cost functions that fulfill the FIFO property. Additionally, for any arc $a \in A$ let the functions $c_{a,j}$, $j \in \{2, \dots, d\}$ have only non-negative intercepts. Then, the MD-FPTAS computes a $(1+\varepsilon)$-cover of the minimum complete set of efficient paths for $\mathcal{I}$ computed by the MD-A.*

Proof. The proof is analogous to that of Theorem 1 using Equation (9) instead of Equation (8) in the induction step. □

4.2. Complexity of the MD-FPTAS

In this section, we set $C := \max_{j \in \{2,\dots,d\}} C_j$. Then, each tensor T_v can store at most $\mathcal{T} := (\lfloor \frac{n}{\varepsilon} \log(nC) \rfloor)^{d-1}$ paths and when analyzing the MD-FPTAS in terms of $\mathcal{N}_{\max}$ and $\mathcal{N}$ as in Section 3, we get

$$\mathcal{N}_{max} = \left(\left\lfloor \frac{n}{\varepsilon} \log(nC) \right\rfloor \right)^{d-1} \text{ and } \mathcal{N} \leq n \left(\left\lfloor \frac{n}{\varepsilon} \log(nC) \right\rfloor \right)^{d-1}.$$

Recall that the lists L_v contain permanent labels that are sorted in lexicographically increasing order. In the biobjective case this implies that they will be sorted increasingly according to the first cost component and simultaneously, decreasingly w.r.t. their pos value (Note that for $d = 2$, $\text{pos}(\cdot)$ maps to $\mathbb{R}_{\geq 0}$ so it is well defined to talk about a label's pos value). There are two reasons for this. On one hand we have the monotonicity of the log function and on the other hand, the already discussed fact that for $d = 2$ efficient labels that are sorted lexicographically have an increasing second cost component. Hence, as in the exact case, the complexity $\mathcal{C}$ of the pos-dominance checks (Algorithms 4 and 5) is constant in the biobjective case and linear ($\mathcal{O}(\mathcal{T})$) for $d \geq 3$. Table 2 shows the time complexity of the different FPTAS for the MOSP problem that we are discussing in this paper.

Table 2. Complexities of the different state of the art FPTAS for MOSP problems. Recall that $C := \max_{j\in\{2,\dots,d\}} \frac{c_j^{\max}}{c_j^{\min}}$, and $\mathcal{T}$ denotes the size of the tensors T_v, i.e., the max. number of paths to be stored at each node.

	TZ [2]	**Martins-FPTAS [3]**	**MD-FPTAS**	**MD-FPTAS_T_v (Section 4.2.1)**
$d = 2$	$\mathcal{O}\left(\frac{n^2m}{\varepsilon}\log(nC)\right)$	$\mathcal{O}(n^3\mathcal{T}^2)$	$\mathcal{O}((n\log n + m)\mathcal{T})$	$\mathcal{O}((n\log n + m)\mathcal{T})$
$d \geq 3$	$\mathcal{O}(nm\mathcal{T})$	$\mathcal{O}(n^3\mathcal{T}^2)$	$\mathcal{O}((n\log n + m\mathcal{T})\mathcal{T})$	$\mathcal{O}((n\log n + m)\mathcal{T})$

4.2.1. Storing Tensors Explicitly

While the complexity of the MD-FPTAS is lower than that of the Martins-FPTAS, the FPTAS presented in [2] (TZ-FPTAS) is yet to be undercut. This algorithm works similar to the well known Bellman Ford algorithm for the One-to-All Shortest Path Problem and stores the tensor T_v for every node $v \in V$. In iteration $i \in \{1,\dots,n-1\}$, the algorithm computes (s,v)-paths with at most i edges and does no proper dominance check. Instead, for a newly found path p, the entry $\operatorname{pos}(p)$ in T_v is checked: if it is empty, p is added; if a path already exists, only the one with the lowest costs in the exact dimension (in [2] it is the d_{th} one) is kept. Since the dominance checks are costly, storing the tensors T_v and checking only the current position yields a great advantage when it comes to the algorithmic complexity.

We can adapt the MD-FPTAS such that at every node a tensor T_v is stored. The entries of T_v are 0 or 1 depending on whether a path with the corresponding pos value has already been stored in L_v. Let ℓ_v be a tentative label for a node v computed in Line 6 of nextCandidateLabel or in Line 2 of propagate. Instead of calling the function `BLACK_BOX_DOM`, we check if $T_v[\operatorname{pos}(\ell_v)] == 0$. In case the tensor entry is indeed set to 0, we add ℓ_v to L_v and set $T_v[\operatorname{pos}(\ell_v)] = 1$. If the tensor entry is set to 1, we neglect ℓ_v since there is a lex. smaller label in L_v with the same pos than ℓ_v, hence pos-dominating ℓ_v.

The suggested adaption increases the space complexity of the MD-FPTAS. Moreover, in general, it will compute more labels than before since checking dominance using $\preceq_{\text{pos}}$ is more restrictive than checking pos-equality. However, as in the TZ-FPTAS, the construction of the tensors T_v still guarantees that the number of paths and iterations stays polynomially bounded. As a consequence, the running time of this variant of the MD-FPTAS for $d \geq 3$ objectives becomes

$$\mathcal{O}((nlogn + m)\mathcal{T}).$$

As shown in Table 2, this is the best known running time bound for an FPTAS for MOSP problems.

5. Computational Results

In this section we provide evince the computational efficiency of the MD-FPTAS in comparison to Martins-FPTAS as presented in [3]; the latter turned out to be faster than the TZ-FPTAS of [2]. Martins-FPTAS is based on the classical label setting algorithm for the MOSP problem by Martins [15]. The data structures are similar to those of the MD-FPTAS. Instead of having at most one label per node in the priority queue, it stores all tentative labels therein until they are extracted or deleted because a label entering the queue dominates them. This iteration through the queue to possibly delete labels is more costly than the searches for a node's next candidate label performed in the MD-FPTAS. Table 2 shows the complexity of Martins-FPTAS.

5.1. Test Instances

We perform experiments considering 2 and 3 objective functions. In the biobjective static case we use the same instances that were used in [3]. The first group of such instances consists of graphs that contain only efficient paths. These graphs were first described in [1] and are suitable to check the impact of the used approximation techniques (see Figure 4).

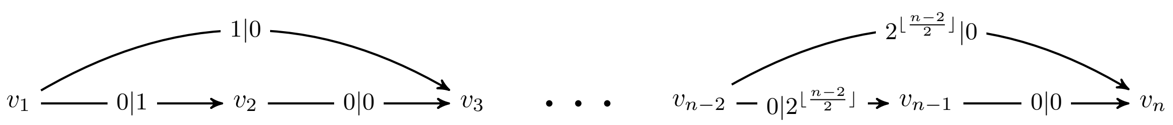

v_1
$0|1$
v_2
$0|0$
v_3
$1|0$
v_{n-2}
$0|2^{\lfloor \frac{n-2}{2} \rfloor}$
v_{n-1}
$0|0$
v_n
$2^{\lfloor \frac{n-2}{2} \rfloor}|0$

Figure 4. General EXP instance. Every path is efficient.

We call these instances EXP; the corresponding graphs have 3 to 51 nodes. The second group of instances, denoted by GRID, are 33 undirected grid graphs of varying size. All instances within GRID have a number of nodes that varies between 1202 and 40,002 and a number of arcs that varies between 4720 and 159,600. The search starts at an artificial node connected to all nodes in the first column of the grid. The costs on the arcs are generated randomly between 0 and 10. The third group of instances are 15 so called NetMaker graphs. They have 3000 nodes and between 30,000 and 80,000 arcs. The source node is always the node with id 0. Both the GRID and NET instances were first used in [25].

In the 3-dimensional case, we consider a subset of the instances used in [17]. The first set of instances are again NetMaker graphs with an extra cost component. These NET3D instances have 5000 to 15,000 nodes and 40,045 to 344,189 arcs. In total, we consider 35 such graphs. Again, the source node is always the one with id 0. We also consider grids with 3 objectives. The undirected 100×100 grid graph remains unchanged among all instances; we consider 10 different 3-dimensional arc cost functions. These instances are the only ones for which we consider a One-to-One scenario. Trying to solve the One-to-All MOSP problem on these grids was not possible without violating the time limit. Hence, we added the pruning techniques for One-to-One MOSP instances described in [17] to Algorithm 1. It is easy to see that they are compatible with the approximation techniques used in this paper. In total, the $\texttt{GRID} - \texttt{3D}$ instance set contains 300 One-to-One MOSP instances with varying L_1 norm between the source and the target node.

The last test instance is a Dyn-MOSP instance motivated by the Horizontal Flight Planning Problem (HFPP) introduced in [26,27]. The directed graph in this instance has $410,387$ nodes and $878,902$ arcs and is called an airway network. The arcs are the direct connections between pre-defined coordinates (the graph's nodes) along which commercial aircraft are allowed to fly (on www.skyvector.com Sky-Vector an airway network can be displayed). We define two cost functions on each arc. The first one encodes the duration of the traversal of an arc depending on the time point at which the tail of the arc is reached. The duration is influenced by weather conditions and we evaluate the weather information we have every 3h to get 10 data points per arc. The second function models the aircraft is fuel consumption along an arc depending on the aircraft is weight at the arc's tail node. In our model we get 171 initial weights per arc and the corresponding consumption for each weight. The difference between two consecutive weights is 500 kg. In both functions, datapoints are interpolated linearly, hence obtaining two continuous piecewise linear functions. The single pieces of the duration function can have positive or negative slopes depending on the wind but the FIFO property still holds as shown in [26]. The consumption function yields an always positive slope since clearly, a higher initial weight will cause a higher consumption. It is therefore also FIFO and, more importantly, the intercepts of its affine pieces are positive, hence fulfilling the requirements from Lemma 3. In total, we have randomly chosen 380 airports as the initial nodes s and compute the (pos)-efficient paths to all nodes reachable with the full tank of a long-haul aircraft.

5.2. Results

The experiments were ran on a machine with an Intel Xeon CPU E5-2670 v2 @ 2.50 GHz processor. It has two CPUs per node and 10 cores per CPU. The available RAM was 128 GB. All algorithms were implemented in *C* and compiled with the version 7.5 of the GCC compiler with compiler optimization level set at 03. For the priority queues, we used our own implementation of a binary heap. The only difference between the heaps used for the implementations of Martins's algorithm and those used in the implementations of the MD algorithms is that in the former we took extra care of guaranteeing fast access to a node's heap labels. This is needed because every time a label for a node v is added to Q, labels for v in Q that are dominated by the new one have to be removed. All lists of permanent labels are modelled as arrays, allowing all algorithms to share the code used for the dominance checks. We set a time limit of 5400 s for all algorithms. Whenever we report averages, we consider instances that were solved by all algorithms involved in the comparison. The min and max values consider only the results obtained by single algorithms.

5.2.1. Static Biobjective Results

Table 3 summarizes the results of the biobjective MOSP instances. On average, the BDA is 3450 times faster than Martins's algorithm in the EXP instance set. The average number of permanent labels on these instances is $338,611$ and the maximum is $1,572,862$. We computed $(1+\varepsilon)$-covers for the `EXP` instances with different values for ε and the average speedup decreases steadily as ε grows: $\times 30$ for $\varepsilon = 0.05$, $\times 2.4$ for $\varepsilon = 0.5$, and $\times 1,84$ for $\varepsilon = 1$. This is because 1.10%, 0.17%, and 0.11% of the labels from the exact solution sets are computed for the mentioned ε values.

Table 3. Results from one to all runs of biobjective Martins-FPTAS and MD-FPTAS.

			Martins				**BDA**			
		$\mathcal{N}$	**Exact $t[s]$**	**FPTAS $t[s]$**			**Exact $t[s]$**	**FPTAS $t[s]$**		
	$\varepsilon =$			0.05	0.5	1		0.05	0.5	1
EXP	avg	338,611	524.5146	0.3062	0.0077	0.0046	0.1545	0.0109	0.0032	0.0025
	min	4	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
	max	1,572,862	4213.4131	8.1309	0.1652	0.0658	36.6369	0.0811	0.0229	0.0176
GRID	avg	872,717	0.9005	1.0308	1.0323	1.0030	0.6181	0.9846	0.9891	0.9534
	min	8189	0.0009	0.0024	0.0011	0.0038	0.0015	0.0013	0.0038	0.0034
	max	5,381,078	6.8469	7.7819	7.8717	7.8638	4.2258	6.8425	6.8795	6.7666
NET	avg	597,998	2.3356	2.7183	2.7106	2.6828	0.7214	1.3948	1.3723	1.3597
	min	185,894	0.3399	0.4359	0.4480	0.4217	0.1407	0.3527	0.3260	0.3395
	max	1,260,412	5.6329	6.6300	6.6990	6.6252	1.7005	3.8076	3.7614	3.7924

Figure 5 gives a visual impression of the running times: on the left hand side we compare the BDA with the FPTAS-BDA and notice how the solution time of the exact algorithm grows exponentially with the number of computed paths. The FPTAS is not faster than the exact algorithms when the number of labels is similar, but on the bigger `EXP` graphs it saves a lot of labels. On the right hand side we compare the Martins-FPTAS and the BDA-FPTAS. We can see that the running time growth of the BDA is much slower.

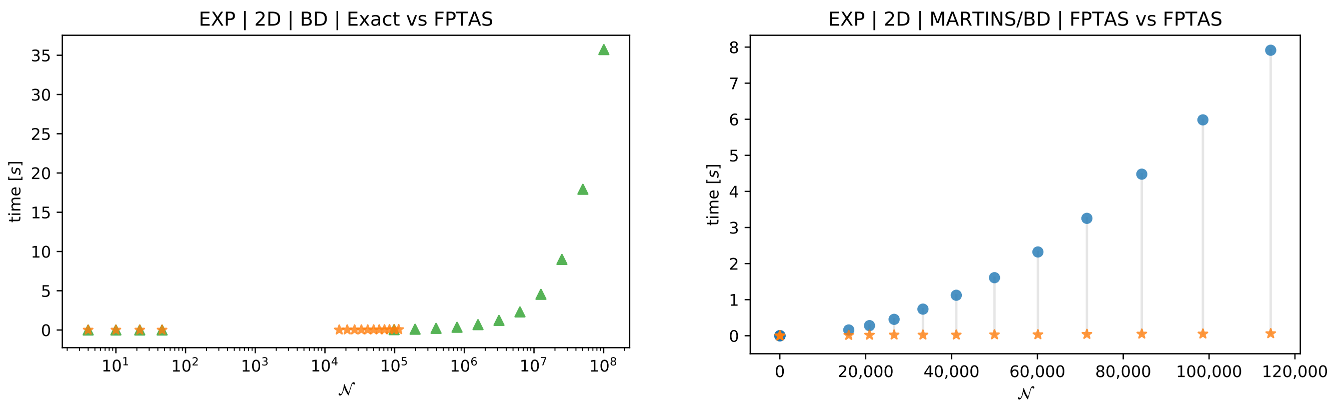


Figure 5. BDA exact (green), BD-FPTAS (yellow) and Martins-FPTAS (blue) on exponential instances. $\varepsilon = 0.05$.

The biobjective GRID and NET instances unveil the major drawback of the used approximation techniques: on graphs with a realistic/practical amount of nodes, the value of $r = (1+\varepsilon)^{\frac{1}{n-1}}$ is very close to 1 and the pos values of any two different paths are almost always distinct. Hence, the exact algorithms are faster than the FPTAS since the computation time of pos is non negligible in practice and no labels are saved. In [3] the authors overcome this problem by choosing huge values for ε. Then, they compute an a posteriori approximation that always turns out to be much better than ε but there is no guarantee for it. Instead, we focus on values for ε that are consistent with the assumption $\varepsilon \leq 1$ that we made in Section 4. The consequence is indeed that the average FPTAS speedup for $\varepsilon = 0.05$ on GRID is $\times 1.66$ and on NET instances it is $\times 3.23$, i.e., in both cases a slowdown. On both instance sets the FPTAS solutions contained almost all exact solutions and even increasing ε up to 1 did not have a noteworthy impact. All algorithms were able to solve all instances in these two sets within the time limit. Figure 6 compares both FPTAS and consolidate the impression gained from the EXP instances: the running time advantage of the MD-FPTAS gets bigger as the number of efficient paths grows.

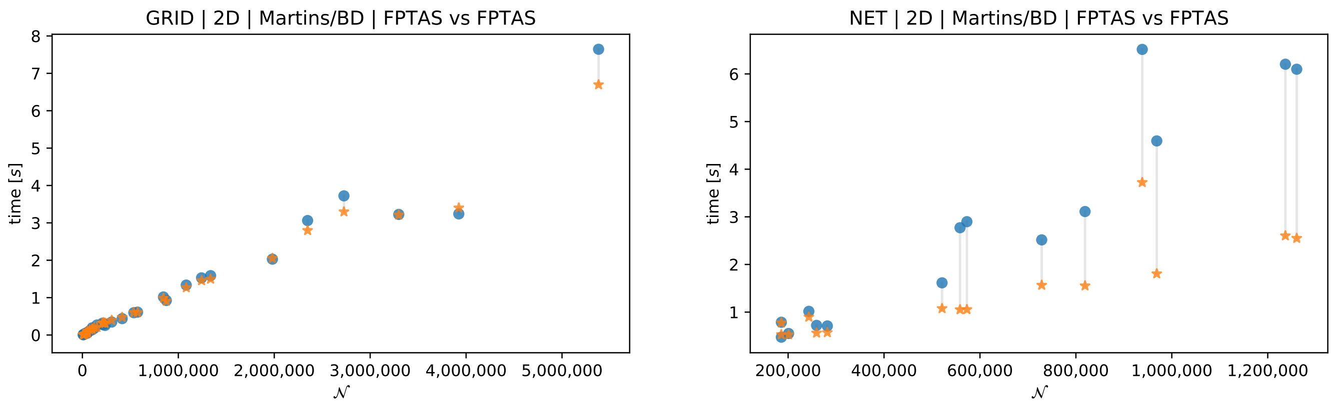


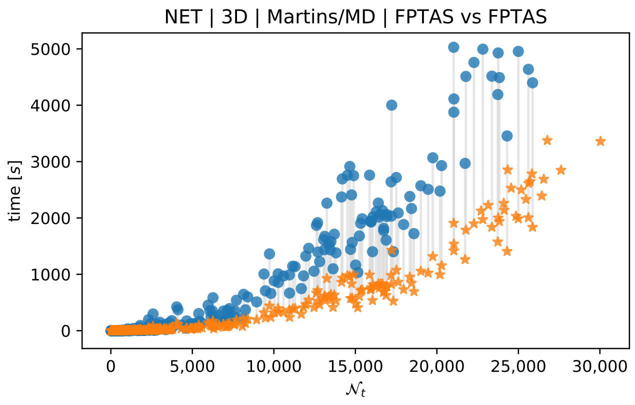
Figure 6. BD-FPTAS (yellow) and Martins-FPTAS (blue) on grid instances (**left**) and netmaker instances (**right**).

5.2.2. Static Three Objective Results

Instances with three objectives are much harder to solve. In Table 4, we summarize the results obtained from the One-to-One queries ran on GRID3D instances and from the One-to-All queries ran on NET3D instances. We observe the same behavior as in the 2*D* instances: a solid running time advantage for the MD-FPTAS on average ($\times 1.70$ on GRID3D instances and $\times 1.46$ on NET3D instances) for $\varepsilon = 0.05$ but slower running times than in the exact counterparts. In these experiments all algorithms always computed always the same amount of labels per instance. Figure 7 shows the comparison of both FPTAS' running times depending on the number of computed paths. In both instance sets the Martins-FPTAS failed to solve bigger instances within the time limit. The overall trend mirrors the biobjective results as the running time of the MD-FPTAS grows considerably slower than that of the Martins-FPTAS.

Table 4. Results obtained by 3-dimensional Martins-FPTAS and MD-FPTAS.

			Martins			**MDA**		
		$\mathcal{N}_t$(o2o)/ $\mathcal{N}$(o2a)	**Exact $t[s]$**	**FPTAS $t[s]$**		**Exact $t[s]$**	**FPTAS $t[s]$**	
	$\varepsilon =$			0.05	1		0.05	1
GRID 3D One-to-One	avg	8307	647.5844	772.9468	757.3884	439.4622	452.7110	452.4130
	min	4	0.0255	0.0200	0.0255	0.0251	0.0232	0.0261
	max	30,041	4258.0670	5027.8374	5015.9595	3338.0901	3375.1526	3408.5612
NET 3D One-to-All	avg	13,308,684	1136.8950	1271.7758	1227.7586	823.6826	872.3511	844.0173
	min	1,170,703	8.8158	43.9947	10.3916	4.0682	18.8914	5.5458
	max	38,647,047	4288.9419	4626.8179	4636.0591	4394.0124	4486.7231	4468.3011



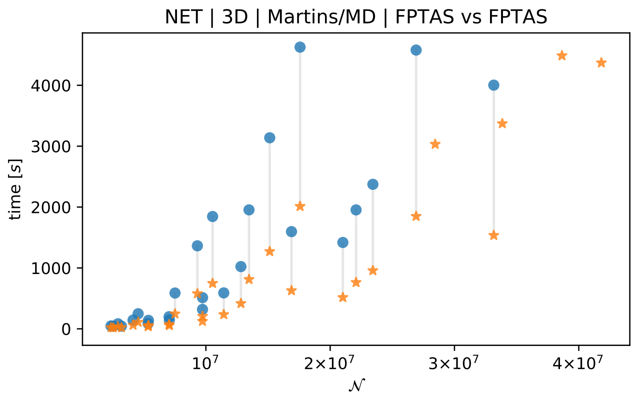


Figure 7. MD-FPTAS (yellow) and Martins-FPTAS (blue) on grid instances (**left**) and netmaker instances (**right**).

5.2.3. Our Implementation of Martins's Algorithm

Note that our implementation of Martins's (exact) Algorithm and of the Martins-FPTAS is competitive compared to other relevant publications. For example, our biobjective implementation is up to $\times 10^5$ faster than the one used in [3] when solving `EXP` instances. We believe that the reason is that-as already mentioned-we are using a constant time dominance check in the biobjective case. However, an in depth comparison of the performance of both implementations is not possible: we restricted ourselves to instances that use an approximation factor $\varepsilon \leq 1$. This matches the assumption that we made in Section 4 to ensure that $(1+\varepsilon) = \Theta(\varepsilon)$.

The direct comparison of our implementation of Martins's algorithm with the one used in [4] is even harder because the used instances do not coincide. However, the authors try to use `C`$++$ vectors whenever possible and use a lexicographically sorted binary heap to store the tentative labels. These choices are similar to ours (we use *C* arrays) and the reported running times on instances with three cost components are comparable to ours.

5.2.4. Results on Dyn-MOSP Instances

Figure 8 contains histograms showing how the labels (left) and running time (right) savings of the MD-FPTAS are distributed among the 380 considered Dyn-MOSP instances. On average, 21% of the time and 35% of the exact labels can be saved already for $\varepsilon = 0.5$.

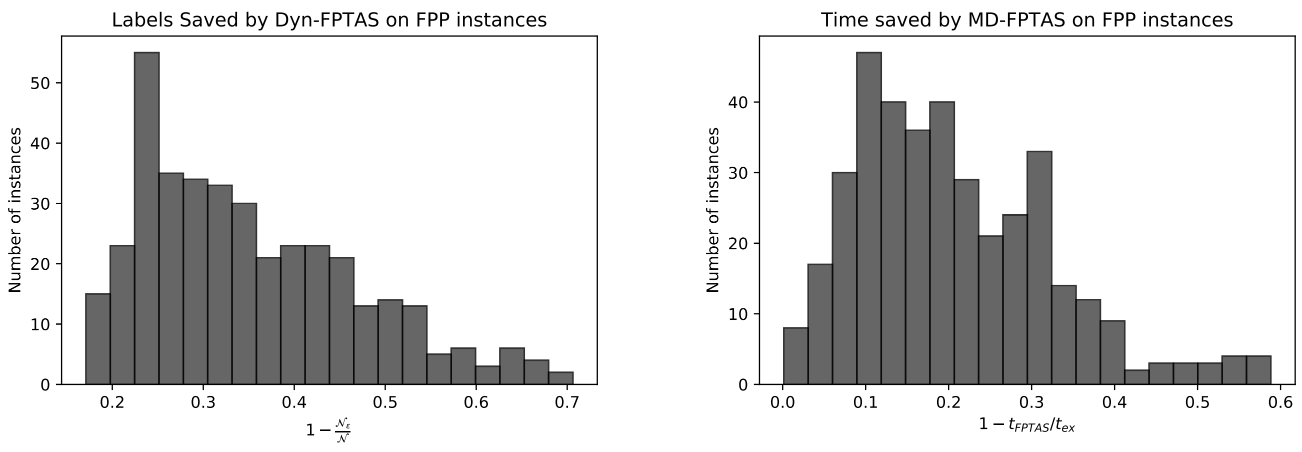


Figure 8. Distribution of percentage of labels saved by the MD-FPTAS in comparison to the exact MD-A on FPP instances.

In Table 5, we depicted some geographically distant departure airports and show the impact of the FPTAS when computing routes to all reachable nodes. We finish our computational experiments showing the least consumption and fastest routes from Berlin to Yekaterinburg in Figure 9. Even though consumption and time are correlated objectives, this example shows that both have to be considered since the routes vary considerably.

Table 5. Running times and computed permanent labels on FPP instances.

Out-Airport	Exact		$\varepsilon = 0.5$		$\varepsilon = 1$	
	$t[s]$	$\mathcal{N}$	$t[s]$	$\mathcal{N}$	$t[s]$	$\mathcal{N}$
Cape Town	1.3288	1,537,645	1.0757	917,327	1.0246	817,945
Los Angeles	11.1541	12,854,272	8.5292	8,260,426	8.4359	7,961,418
Moscow	15.8314	19,385,407	11.4957	11,458,971	11.2693	10,819,170
Berlin-Tegel	14.6621	16,815,977	12.8159	12,724,498	12.6338	12,126,247
Tenerife	6.4615	9,182,417	5.4099	6,247,958	5.2085	5,739,303

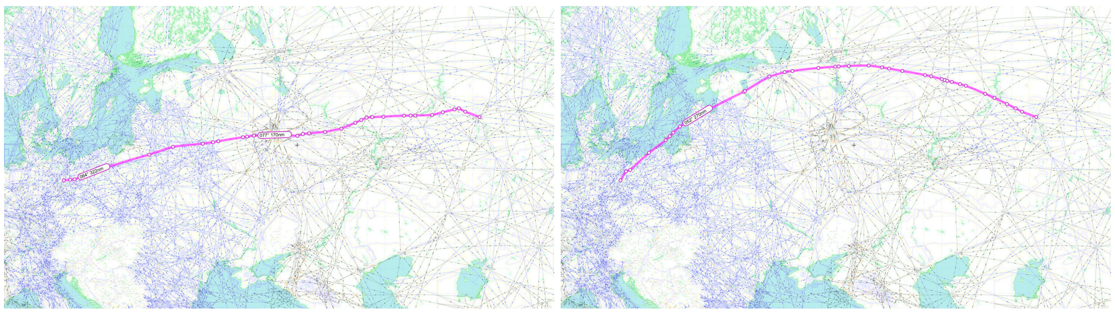


Figure 9. Least consumption (**left**) and fastest (**right**) routes from Berlin to Yekaterinburg. Source: www.skyvector.com.

6. Conclusions

We have proven that Dynamic Multiobjective Shortest Path (Dyn-MOSP) problems can be solved by a generalization of the static Multiobjective Dijkstra Algorithm (MD-A) if the arc cost functions are FIFO and have independent dynamics (e.g., weight and time; time and state of charge). Our main contribution was to adapt the techniques used in the seminal work by Tsaggouris and Zaroliagis [2] to derive a new FPTAS for MOSP problems that is based on the label setting MD-A. The running time of the resulting MD-FPTAS is the number of computed paths multiplied by the running time of the classical Dijkstra algorithm and is thus—to the best of our knowledge—the most efficient FPTAS for MOSP problems in the literature. Even better, it also works for Dyn-MOSP instances if the arc cost functions are FIFO, continuous, and piecewise linear, having only positive intercepts. These requirements are not very restrictive in practice.

We corroborated the theoretical efficiency of our algorithms computationally. On a test set of standard bidimensional and 3-dimensional instances, our MD-FPTAS was faster than the Martins-FPTAS introduced by Breugem et al. [3]. In the static case, we faced the same problem as the authors in [2,3]: the FPTAS does not avoid the computation of paths unless ε is chosen very large. The reason is that, so far, most instances used in the literature to test MOSP algorithms have integer costs, causing efficient cost vectors to lie at least one cost unit apart from each other. In Dyn-MOSP instances the evaluation of continuous, piecewise linear functions is likely to generate labels with rational cost. This is the case in the Flight Planning instances that we considered. Furthermore, indeed, using realistic values for ε, we computed $(1+\varepsilon)$-covers for these instances and saved 21% in terms of running time and 35% in terms of labels.

Author Contributions: Conceptualization: P.M.d.l.C., R.B., and A.S.-N.; methodology, P.M.d.l.C., R.B., and A.S.-N.; software, P.M.d.l.C. and L.K.; validation, P.M.d.l.C. and L.K.; formal analysis, P.M.d.l.C., R.B., L.K., and A.S.-N.; investigation, P.M.d.l.C. and L.K.; resources, P.M.d.l.C., R.B., L.K., and A.S.-N.; data curation, P.M.d.l.C. and L.K.; writing–original draft preparation, P.M.d.l.C.; writing–review and editing, P.M.d.l.C., R.B., L.K., and A.S.-N.; visualization, L.K.; supervision, P.M.d.l.C., R.B., and A.S.-N.; project administration, P.M.d.l.C., R.B., and A.S.-N.; funding acquisition, R.B. and A.S.-N. All authors have read and agreed to the published version of the manuscript.

Funding: The researches affiliated to the Zuse Institute Berlin conducted this work within the Research Campus MODAL-Mathematical Optimization and Data Analysis Laboratories-, funded by the German Federal Ministry of Education and Research (BMBF) (fund number 05M20ZBM). Antonio Sedeño-Noda was partially supported by the grant MTM2016-74877-P from the Ministerio de Economía y Competitividad.

Data Availability Statement: Data sharing not applicable.

Acknowledgments: We thank Lufthansa Systems GmbH & Co. KG for providing us with the data used to build our Dyn-MOSP test instances. We also thank Arno Kühner for the helpful and fruitful discussions related proofs and the implementations concerning Dyn-MOSP problems. The authors from [3] provided us with the biobjective Grid and Netmaker instances used in their paper, so we could compare our results with existing data. They got their instances from Andrea Raith, who we also want to thank for providing us with the 3-dimensional instances we used for our computations.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hansen, P. Bicriterion Path Problems. In *Multiple Criteria Decision Making Theory and Application*; Fandel, G., Gal, T., Eds.; Springer: Berlin/Heidelberg, Germany, 1980; pp. 109–127.
2. Tsaggouris, G.; Zaroliagis, C. Multiobjective Optimization: Improved FPTAS for Shortest Paths and Non-linear Objectives with Applications. In *Algorithms and Computation*; Asano, T., Ed.; Springer: Berlin/Heidelberg, Germany, 2006; pp. 389–398.
3. Breugem, T.; Dollevoet, T.; van den Heuvel, W. Analysis of FPTASes for the multi-objective shortest path problem. *Comput. Oper. Res.* **2017**, *78*, 44–58. [CrossRef]
4. Bökler, F.; Chimani, M. Approximating Multiobjective Shortest Path in Practice. In *2020 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2020; pp. 120–133. [CrossRef]
5. Emmerich, M.; Deutz, A. A tutorial on multiobjective optimization: Fundamentals and evolutionary methods. *Nat. Comput.* **2018**, *17*, 585–609,
6. Ehrgott, M.; Gandibleux, X. *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys*; International Series in Operations Research & Management Science; Springer: New York, NY, USA, 2006.
7. Ehrgott, M.; Gandibleux, X. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum* **2000**, *22*, 425–460. [CrossRef]
8. Ulungu, E.; Teghem, J. Multi-objective shortest path problem: A survey. In *Proceedings of the International Workshop on Multicriteria Decision Making: Methods-Algorithms-Applications at Liblice, Czechoslovakia*; Glückaufova, D., Loula, D., Cerny, M., Eds.; Institute of Economics, Czechoslovak Academy of Sciences: Prague, Czech Republic, 1991; pp. 176–188.
9. Current, J.; Marsh, M. Multiobjective transportation network design and routing problems: Taxonomy and annotation. *Eur. J. Oper. Res.* **1993**, *65*, 4–19. [CrossRef]

10. Skriver, A. A classification of Bicriterion Shortest Path (BSP) algorithms. *Asia Pac. J. Oper. Res.* **2000**, *17*, 199–212.
11. Tarapata, Z. Selected multicriteria shortest path problems: An analysis of complexity, models and adaptation of standard algorithms. *Int. J. Appl. Math. Comput. Sci.* **2007**, *17*, 269–287. [CrossRef]
12. Clímaco, J.; Pascoal, M. Multicriteria path and tree problems: Discussion on exact algorithms and applications. *Int. Trans. Oper. Res.* **2012**, *19*, 63–98. [CrossRef]
13. Vincke, P. Problemes multicriteres. *Cah. Centre d' Etudes de Rech. Oper.* **1974**, *16*, 425–439.
14. Serafini, P. Some considerations about computational complexity for multiobjective combinatorial problems. *Recent Adv. Hist. Dev. Vector Optim.* **1986**, *294*, 222–232.
15. Martins, E.Q.V. On a multicriteria shortest path problem. *Eur. J. Oper. Res.* **1984**, *16*, 236–245. [CrossRef]
16. Sedeño-noda, A.; Colebrook, M. A biobjective Dijkstra algorithm. *Eur. J. Oper. Res.* **2019**, *276*, 106–118. [CrossRef]
17. de las Casas, P.M.; Sedeno-Noda, A.; Borndörfer, R. *An Asymptotically and Computationally Improved Multiobjective Shortest Path Algorithm*; Technical Report 20–26. ZIB; Takustr: Berlin, Germany, 2020.
18. Papadimitriou, C.; Yannakakis, M. On the approximability of trade-offs and optimal access of Web sources. In Proceedings of the 41st Annual Symposium on Foundations of Computer Science, Washington, DC, USA, 12–14 November 2000. [CrossRef]
19. Kostreva, M.M.; Lancaster, L. Multiple Objective Path Optimization for Time Dependent Objective Functions. In *Multiple Objective and Goal Programming*; Trzaskalik, T., Michnik, J., Eds.; Physica-Verlag HD: Heidelberg, Germany, 2002; pp. 127–142.
20. Disser, Y.; Müller-Hannemann, M.; Schnee, M. Multi-criteria Shortest Paths in Time-Dependent Train Networks. *Exp. Algorithms Lect. Notes Comput. Sci.* **2008**, *5038*, 347–361. [CrossRef]
21. Foschini, L.; Hershberger, J.; Suri, S. On the Complexity of Time-Dependent Shortest Paths. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2011; SODA' 11, pp. 327–341.
22. Bellman, R. The theory of dynamic programming. *Bull. Am. Math. Soc.* **1954**, *60*, 503–515. [CrossRef]
23. Captivo, M.; Clímaco, J.; Figueira, J.; Martins, E.; Santos, J. Solving bicriteria 0-1 knapsack problems using a labeling algorithm. *Comput. Oper. Res.* **2003**, *30*, 1865–1886. [CrossRef]
24. Fredman, M.L.; Tarjan, R.E. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM* **1987**, *34*, 596–615. [CrossRef]
25. Raith, A.; Ehrgott, M. A comparison of solution strategies for biobjective shortest path problems. *Comput. Oper. Res.* **2009**, *36*, 1299–1331. [CrossRef]
26. Blanco, M.; Borndörfer, R.; Hoang, N.D.; Kaier, A.; Schienle, A.; Schlechte, T.; Schlobach, S. Solving Time Dependent Shortest Path Problems on Airway Networks Using Super-Optimal Wind. In *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*; Goerigk, M., Werneck, R., Eds.; OpenAccess Series in Informatics (OASIcs); Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik: Dagstuhl, Germany, 2016; Volume 54, pp. 12:1–12:15. [CrossRef]
27. Blanco, M.; Borndörfer, R.; Hoàng, N.D.; Kaier, A.; Casas, P.M.; Schlechte, T.; Schlobach, S. Cost Projection Methods for the Shortest Path Problem with Crossing Costs. In *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*; D'Angelo, G., Dollevoet, T., Eds.; OpenAccess Series in Informatics (OASIcs); Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik: Dagstuhl, Germany, 2017; Volume 59, pp. 15:1–15:14. [CrossRef]