Zuse Institute Berlin

Takustr. 7
14195 Berlin
Germany

PEDRO MARISTANY DE LAS CASAS, RALF BORNDÖRFER, LUITGARD KRAUS, AND ANTONIO SEDEÑO-NODA

An FPTAS for Dynamic Multiobjective Shortest Path Problems

ZIB Report 20-31 (December 2020)

Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

Telephone: +49 30-84185-0
Telefax: +49 30-84185-125

E-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

An FPTAS for Dynamic Multiobjective Shortest Path Problems

Pedro Maristany de las Casas* Ralf Borndörfer* Luitgard Kraus*
Antonio Sedeño-Noda†

Abstract

We propose in this paper the Dynamic Multiobjective Shortest Problem. It features multidimensional states that can depend on several variables and not only on time; this setting is motivated by flight planning and electric vehicle routing applications. We give an exact algorithm for the FIFO case and derive from it an FPTAS, which is computationally efficient. It also features the best known complexity in the static case.

1 Introduction

We consider in this paper the solution of *Dynamic Multiobjective Shortest Path* (Dyn-MOSP) problems. Our setting is motivated by *Flight Planning Problems* (FPP), in which efficient aircraft routes have to be found in an airway network, considering multiple and dynamic optimization criteria. Namely, the cost of an arc does not only depend on a single variable modeling time, but also on the aircraft's weight. Another application with similar dynamics is the routing of electric vehicles through mountainous terrains in which shorter routes consume more battery.

MOSP problems, i.e., already the static case, are known to be intractable because of the possibly exponential cardinality of the solution set of efficient paths. However, one can try to polynomially bound the size of the solution set while minimizing the quality loss, an approach that motivated the development of Fully Polynomial Time Approximation Schemes (FPTAS) for MOSP problems in recent years.

We will prove that recently developed algorithms for MOSP problems can be generalized to also solve Dyn-MOSP instances, given that the arc cost functions fulfill the *FIFO* property, i.e., a worse arrival at an arc's source node will never turn out to be beneficial. For ease of exposition, we first consisder the static case and design a new FPTAS for MOSP problems. This FPTAS features the best known time complexity bounds. We then show that our results carry over to Dyn-MOSPs if a certain (realistic) assumption on the arc cost functions is made. As usual under the FIFO assumption, the time complexity is the same. We also provide extensive computational evidence of the efficiency of our approach. While existing FPTAS for MOSP problems often do not perform well in practice, it is remarkable that the presented FPTAS avoids the computation of a considerable amount of paths for FPP instances.

*Zuse Institute Berlin, Takustraße 7, 14195, Berlin, Germany.
†Departamento de Matemáticas, Estadística e Investigación Operativa, Universidad de La Laguna, 38271 Santa Cruz de Tenerife, España.

1.1 Literature Review

Multiobjective optimization problems and, in particular, Multiobjective Shortest Path Problems, have been extensively investigated in the literature. Introductions to Multiobjective Combinatorial Optimization problems are provided by, e.g., Emmrich and Deutz [13], Ehrgott [12], or Ehrgott and Gandibleux [11]. The theoretical foundation and the algorithmic development of MOSP problems are reviewed in Ulungu and Teghem [25], Current and Marsh [8], Skriver [22], Tarapata [23], Clímaco, or Pascoal [7].

In the 70s, Vincke [26] considered the MOSP for the first time, using two objective functions. This Biobjective Shortest Path Problem (BOSP) was also considered by Hansen [15], who introduced the first label-setting algorithm for BOSP. Serafini [21] showed that the MOSP problem is NP-complete and Martins [17] generalized Hansen's algorithm for the multiobjective case. Martin's algorithm has served as state of the art MOSP algorithm since it was discovered. Recently, Sedeño and Colebrook [20] and Maristany et. al [9] introduced the Biobjective and Multiobjective Dijsktra algorithms (B/M DA): new label setting algorithms for MOSP problems that were proved to have improved theoretical running time bounds as well as to be more efficient in computational experiments.

Papadimitriou and Yannakakis [18] set a milestone in the field of approximation algorithms for multiobjective optimization problems. They proved that a for a d-dimensional optimization problem, a $(1+\varepsilon)$ pareto curve of size $\mathcal{O}\left((\frac{4B}{\varepsilon})^{d-1}\right)$ exists. Here, B is the number of bits required to represent the values that the objective functions can take and is assumed to be polynomially bounded. They also constructed the first general FPTAS for MOSP. Tsaggouris and Zaroliagis [24] presented a new FPTAS for MOSP 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, hence obtaining a $(1+\varepsilon)$-*cover* of the exact set of efficient solutions. This idea was picked up by Breugem et al. [4] who managed to use Martins' algorithm paired with the subdivision of the outcome space introduced in [24]. The result was a new FPTAS for MOSP that is worse regarding the theoretical running time but performs better computationally. Based on this work, Bökler and Chimani [5] recently published an extensive comparison of different label ordering and selecting strategies.

Literature considering MOSP instances with dynamic, also called time-dependent, cost functions is scarce. Kostreva and Lancaster [16] presented an algorithm for non-monotone increasing arc cost functions that does not reduce to Dynamic Programming. Disser et al [10] mention the necessity to tackle this kind of problems on train networks and use Martins' algorithm to solve them without going into details.

1.2 Outline

In Section 2, we formulate the Dyn-MOSP problem and in Section 3, we explain how the MD 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 running time is done using a black-box dominance check whose complexity varies depending on the number of objectives and the used definition of dominance. We use this abstract representation in Section 4 in which we explain how to divide the outcome space of 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 dynamic arc costs. Then, we derive a condition on dynamic arc cost functions that ensures that the new FPTAS also works for Dyn-MOSP instances. Finally, in Section 5 we test our algorithms against a state of the art FPTAS for MOSP.

2 Multiobjective Shortest Path Problems

The d-dimensional Dyn-MOSP problem involves a directed graph $D = (V, A)$ with $n := |V|$ nodes and $m := |A|$ arcs, a start node $s \in V$, an initial state $\tau_0 \in \mathbb{R}^d_{\geq}$, $d \in \mathbb{N}$, and d arc cost functions

$$\begin{aligned} c_i : A \times \mathbb{R}_{\geq} &\to \mathbb{R}_{\geq} \\ (a, \tau_i) &\mapsto c_i(a, \tau_i), \end{aligned} \tag{1}$$

for $i = 1, \ldots, d$. The second argument of c_i models the dynamics of the problem and represents a *state*[1] at the tail node of a. Note that each cost function c_i depends only on the state τ_i, i.e., we consider d independent dynamic cost functions. For ease of exposition, we denote by $c := [c_i]_{i=1}^d$ the multidimensional cost function consisting of all d individual costs. This setting applies to applications that involve an independent, but multidimensional "state flow". We will discuss an application in flight planning in Section 5.1.

In the static case, we can drop the states to simplify notation. The arc cost function c can be extended to 2^A to be well defined on paths in D. Given two nodes u, $v \in V$, a (u, v)-path p is a sequence $(a_1, \ldots, a_k)$, $k \in \mathbb{N}$, of arcs such that u is the source node of a_1 and v is the target node of a_k. The costs of a (u, v)-path p with initial state τ are recursively given by

$$c_\tau(a_1) := c(a_1, \tau) \tag{2}$$

$$c(p, \tau) := c(a_k, c((a_1, \ldots, a_{k-1}), \tau)) + c((a_1, \ldots, a_{k-1}), \tau). \tag{3}$$

The extension of p along a (v, w)-path q is $p \circ q$ with cost $c(p \circ q, \tau) := c(p, \tau) + c(q, c(p, \tau))$. In the static case, the costs of p are $c(p) := \sum_{i=1}^k c(a_i)$.

In Dyn-MOSP problems, we seek to find efficient (s, v)-paths with initial state τ_0 for every $v \in V$. The efficiency of paths in a multiobjective scenario is defined using a binary relation $\prec$ on the coimage $\mathbb{R}^d_{\geq}$ of c. Given two (s, v)-paths p and q and a common initial state τ_0, p is said to $\prec$-*dominate* q if and only if $c(p, \tau_0) \prec c(Q, \tau_0)$. In this case, we write $p \prec q$. An (s, v)-path is called $\prec$-*efficient* if it is not $\prec$-dominated by any other (s, v)-path p' with initial state τ_0. We can now give a formal definition of the Dynamic Multiobjective Shortest Path Problem:

Definition 1 (Dynamic Multiobjective Shortest Path Problem.)**.** Given a directed graph $D = (V, A)$, a start node $s \in V$, an initial state $\tau_0 \in \mathbb{R}^d_{\geq}$, $d \in \mathbb{N}$, an arc cost function c as defined in (1), and a binary relation $\prec$ on the coimage of c, the *Dynamic Multiobjective Shortest Path Problem* is to find the $\prec$-efficient (s, v)-paths in D for all $v \in V$.

We want to compute at most one efficient path p for every non-dominated point $y \in \mathbb{R}^d_{\geq}$ such that $c(p) = y$. We achieve this using the following partial order:

Definition 2 (Dominance)**.** Given two cost vectors x, $y \in \mathbb{R}^d_{\geq}$, x is said to *dominate* y ($x \preceq_D y$), if and only if $x_i \leq y_i$ for all $i \in \{1, \ldots d\}$.

Note that the widely used *Pareto order* would enforce at least one of the inequalities in Definition 2 to be strict. In this case, equal cost vectors do not dominate each other and the MOSP problem is to find the *maximum complete set* of efficient paths of $\mathcal{I}$. Using $\preceq_D$, as we will do for the remainder of the paper, we instead look for the *minimum complete set* of efficient paths of $\mathcal{I}$.

[1] It possibly encodes e.g., time, weight, state of charge...

3 The Multiobjective Dijkstra Algorithm for Dynamic MOSP Problems

In this section we discuss how the Multiobjective Dijsktra (MD) algorithm (Algorithm 1) presented in [20] and [9] can be adapted to work with dynamic cost. We will see that for the dynamics described in the latter section, the solvability of MOSP and Dyn-MOSP instances mirrors the well known relationship from the single objective case: if arc cost functions fulfill the FIFO property, label setting algorithms for the static case also solve the dynamic case.

3.1 Description of the Algorithm

Algorithm 1: Multiobjective Dijkstra Algorithm

Input : Directed Graph $D = (V, A)$, Arc Costs $c : A \times \mathbb{R}^d_{\geq} \to \mathbb{R}^d_{\geq}$, Node $s \in V$, Initial state $\tau_0 \in \mathbb{R}^d_{\geq}$.

Input FPTAS: Vector of approximation ratios $\varepsilon \in \mathbb{R}^{d-1}_{\geq}$.

`/* Exact: A minimum complete set. FPTAS: A` $(1+\varepsilon)$`-cover. (See Section 4) */`

Output : Labels $L := \bigcup_{v \in V} L_v$.

Prio. Queue $Q \leftarrow \emptyset$;
for $v \in V$ **do** Permanent labels $L_v \leftarrow \emptyset$;
for $a \in A$ **do** $lastProcessedLabel[a] \leftarrow 0$;
Label $l_{init} \leftarrow (s, \tau_0, \texttt{NULL}, \mathrm{pos}\,(\tau_0))$;
$Q \leftarrow Q.\texttt{insert}(l_{init})$;
$L_s \leftarrow L_s \cup \{l_{init}\}$;
while $Q \neq \emptyset$ **do**
  Label $\ell_v^* \leftarrow Q.\texttt{extract\_lexmin}()$;
  Node $v \leftarrow \ell_v^*.\texttt{node}$;
  $L_v \leftarrow L_v \cup \{\ell_v^*\}$;
  $\ell_v^{new} \leftarrow nextCandidateLabel(v, lastProcessedLabel, \cup_{u \in \delta^-(v)} L_u, \varepsilon)$;
  if $\ell_v^{new} \neq \texttt{NULL}$ **then** $Q.\texttt{insert}(\ell_v^{new})$;
  for $w \in \delta^+(v)$ **do** $Q \leftarrow propagate(\ell_v^*, w, Q, L_w, \varepsilon)$;
return L_v for all $v \in V$;

Paths, Labels, and Datastructures. In Algorithm 1, paths are considered in *lexicographically increasing order*. A point $x \in \mathbb{R}^d_{\geq}$ is said to be lexicographically smaller than a point $y \in \mathbb{R}^d_{\geq}$ ($x <_{lex} y$) if and only if $x_i < y_i$ in the first dimension $i \in \{1, \ldots, d\}$ in which $x_i \neq y_i$. If $c(P, \tau_0) <_{lex} c(Q, \tau_0)$ for paths p, p', we say that p is lexicographically smaller than p'. Let P be an (s, v)-path whose last arc is $(u, v) \in A$. We will store paths using *labels*, i.e., an implicit representation. The label corresponding to p is a tuple $\ell = (v, c_\ell := c(p, \tau_0), \ell_u)$, where ℓ_u is the label representing the (s, u)-subpath of p. For every node $v \in V$ the set L_v contains the labels corresponding to the efficient (s, v)-paths found during the algorithm. Additionally, a $\prec_{lex}$ -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 $L_v \not\preceq_D \ell_v$).

Iterations. At the beginning, a label $\ell_{init} = (s, \tau_0, NULL)$ is inserted into Q. The main loop of the algorithm 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. After making ℓ_v^* permanent, each iteration pursues two main tasks.

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

```
1 return L_v[|L_v| − 1] ⪯_D ℓ_v
```

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

```
1 for ℓ ∈ L_v do
2 |   if ℓ ⪯_D l_v then return TRUE;
3 return FALSE
```

The first is to find the label

$$l_v^{new} := \text{arglexmin}_{\substack{\ell_u \in L_u,\\ u \in \delta^-(v)}} \{\ell = (v, c_{\ell_u} + c((u,v), c_{\ell_u}), \ell_u) \mid L_v \not\preceq_D \ell\}, \tag{4}$$

if it exists. l_v^{new} is the lexicographically minimal and non-dominated label for v that can be built out of the permanent labels in L_u, $u \in \delta^-(v)$ (see nextCandidateLabel) and the costs of the connecting arc (u, v). It is not necessary to traverse the entirety of each list L_u for every attempt made to generate a new label for v. Instead, labels in any list 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. `lastProcessedLabel`(u, v) stores the last checked position in L_u and defines where a search for v's next candidate label has to start. If a new label 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 $\ell_w = (w, c_{\ell_v^*} + c((v, w), \ell_v^*), \ell_v^*)$ 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).

3.2 Correctness

As noted in the beginning of the section, we require the arc cost function c to fulfill the *FIFO* property in every dimension: for an $i \in \{1, \ldots, d\}$, an arc $a \in A$, and two states τ, τ' such that $\tau_i < \tau_i'$, we have $\tau_i + c_i(a, \tau_i) < \tau_i' + c_i(a, \tau_i')$. The correctness of Algorithm 1 relies on the *Bellman-Principle* [1] 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 solutions of the Dyn-MOSP problem. In particular, the following statement holds.

Lemma 1. *Given a Dyn-MOSP instance with FIFO arc cost, consider two (s, v)-paths p, p' with equal initial state τ_0 such that $c(p, \tau_0) \preceq_D c(p', \tau_0)$. Then, for the extensions of p and p' along a (v, w)-path q in D there holds $c(p \circ q, \tau_0) \preceq_D c(p' \circ q, \tau_0)$.*

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

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

```
1  Label ℓ_v^new ← (v, (∞, ..., ∞), NULL);
2  for u ∈ δ⁺(v) do
3  |  for k = lastProcessedLabel[(u, v)] to |L_u| do
4  |  |  ℓ_u ← L_u[k];
5  |  |  Cost c_new ← c_{ℓ_u} + c((u, v), c_{ℓ_u});
6  |  |  Label ℓ ← (v, c_new, ℓ_u, pos_ε(c_new)));
   |  |  /* Next time a label for v is searched in L_u, the search starts where the
   |  |     current one ends.  */
7  |  |  lastProcessedLabel[(u, v)] ← k;
8  |  |  if !BLACK_BOX_DOM(L_v, ℓ) then
   |  |  |  // ℓ is non-dominated.  Additionally, it has to be lex.  minimal.
9  |  |  |  if ℓ ≺_lex ℓ_v^new then ℓ_v^new ← ℓ;
10 |  |  |  break;
11 if c_{ℓ_v^new} == ∞ then return NULL;
12 return ℓ_v^new;
```

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

Input : Label ℓ_v, Node $v \in \delta^+(u)$, Prio. Queue Q, Permanent labels L.
Input FPTAS: Vector of approximation ratios $\varepsilon \in \mathbb{R}^{d-1}_{\geq}$.
Output : Updated Prio. Queue Q.

```
1 Cost c_new ← c_{ℓ_v} + c((v, w), ℓ_v);
2 Label ℓ_w ← (w, c_new, ℓ_v, pos_ε(c_new));
3 if !BLACK_BOX_DOM(L_w, ℓ_w) then
4 |  if !Q.contains(w) then
5 |  |  Q.insert(ℓ_w);
6 |  else if ℓ_w ≺_lex Q.getLabel(w) then
7 |  |  Q.update(Q.getLabel(w), ℓ_w);/* Replace current heap label with ℓ_w.   */
8 return Q;
```

Proof. Since $c(p, \tau_0) \preceq_D c(p', \tau_0)$, we know that $c_j(p, \tau_0) \leq c_j(p', \tau_0)$ for any $j \in \{1, \dots, d\}$. Due to the FIFO property of the arc cost function, this implies that after q's first arc, say $(v, v') \in A$, we will have

$$c_j(p \circ ((v, v')), \tau_0) = c_j(p, \tau_0) + c_j((v, v'), c(p, \tau_0)) \leq$$
$$c_j(p', \tau_0) + c_j((v, v'), c(p', \tau_0)) = c_j(p' \circ ((v, v')), \tau_0).$$

This argument can be repeated along every arc of q until we reach the path's end point, implying that $c(p \circ q, \tau_0) \preceq_D c(p' \circ q, \tau_0)$. □

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 solutions. Moreover, the correctness can be proven as in the static case (cf. [20], [9]).

3.3 Complexity

The running time of Algorithm 1 is characterized by the running time of nextCandidateLabel and propagate, both depending on the complexity of the function `BLACK_BOX_DOM`. This abstract function contains the dominance check applied in the different versions of the MD algorithm that we discuss throughout the paper. In the exact biobjective scenario, the dominance check (see Algorithm 2) is performed in constant time. The reason is that the sets L_v are sorted in lexicographically increasing order and contain only efficient labels. Thus, the first cost component is sorted increasingly, while the second cost component is sorted decreasingly. For a new tentative label that is lexicographically greater than all elements of L_v it is enough to be compared with the last element only. This observation is not very widespread in literature but has a remarkable impact in theory and practice (cf. [6], [20], [9]). 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 [14] to get constant running time 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.

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

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

	Exact MD-A (Sec. 3.3)	MD-FPTAS (Sec. 4.1)	MD-FPTAS_T (Sec. 4.1.1)
$d = 2$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
$d \geq 3$	$\mathcal{O}(L_v) \subseteq \mathcal{O}(N_{max})$	$\mathcal{O}(\mathcal{T})$	$\mathcal{O}(1)$

dominated by any label in L_v. Summing over all nodes, we get an overall complexity of

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

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 propagate and nextCandidateLabel, a label is extracted in every iteration. All in all, the running time of Algorithm 1 is

$$\mathcal{O}\left(\mathcal{N}\log n + 2m\mathcal{N}_{max}\mathcal{C}\right) = \mathcal{O}\left(\mathcal{N}\log n + m\mathcal{N}_{max}\mathcal{C}\right) \overbrace{\subseteq}^{\mathcal{N} \leq n\mathcal{N}_{max}} \mathcal{O}\left(\mathcal{N}_{max}(nlog(n) + m\mathcal{C})\right). \quad (5)$$

In Table 1, we list the complexity $\mathcal{C}$ of the dominance check for the different variants of the MD algorithm 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 [20] and [9].

4 A new FPTAS for the Multiobjective Shortest Path Problem

In this section we introduce a new FPTAS for MOSP problems. The FPTAS is based on Algorithm 1 and the idea presented in [24] of dividing the outcome space into a polynomial number of cells, each of them holding at most one path's cost vector.

Consider a vector $\alpha \in \mathbb{R}^d_{\geq 1}$ and two (s, v)-paths p, q. We say that 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 solutions 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. A fully polynomial time approximation scheme (FPTAS) for the (Dyn)-MOSP problem is a family of algorithms such that for any $\vec{\varepsilon} \in \mathbb{R}^d_{>}$ there is an algorithm whose running time is polynomial in the size of the used instance as well as in $\frac{1}{\vec{\varepsilon}_i}$ that computes a $(\vec{1}+\vec{\varepsilon})$-cover of $\mathcal{X}$. The size of the computed cover is also required to be polynomial in the input size and in $1/\varepsilon$. As in the FPTAS presented in [24] and in [4], ours is exact in one dimension. We choose this dimension to be the first one and hence, compute an α-cover with $\alpha = [1, \vec{1} + \vec{\varepsilon}] \in \mathbb{R}^d_{\geq 1}$. In this section, we assume w.l.o.g. that $\vec{\varepsilon}_i = \vec{\varepsilon}_j$ for all i, $j \in \{1, \ldots, d\}$ and speak of a $(1+\varepsilon)$-*cover*, $\varepsilon \in \mathbb{R}_{>}$. Additionally, we assume that $\varepsilon \leq 1$ such that $\ln(1+\varepsilon) = \Theta(\varepsilon)$. We set $c_i^{min} := min_{a \in A} c_i(a)$, $c_i^{max} := max_{a \in A} c_i(a)$, and $C_i = \frac{c_i^{max}}{c_i^{min}}$ for $i \in \{1, \ldots, d\}$.

The new *MD-FPTAS* is based on Algorithm 1 but we need to get rid of the exponential number of efficient paths that a (Dyn)-MOSP instance can have. Let $\vec{r} \in \mathbb{R}^d_{\geq 1}$ be a vector of approximation ratios with $\vec{r}_1 = 1$. 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 $\lfloor \log_{\vec{r}_j}(nC_j) \rfloor$. As with $\vec{\varepsilon}$, we assume w.l.o.g. that all entries of $\vec{r}$ are equal

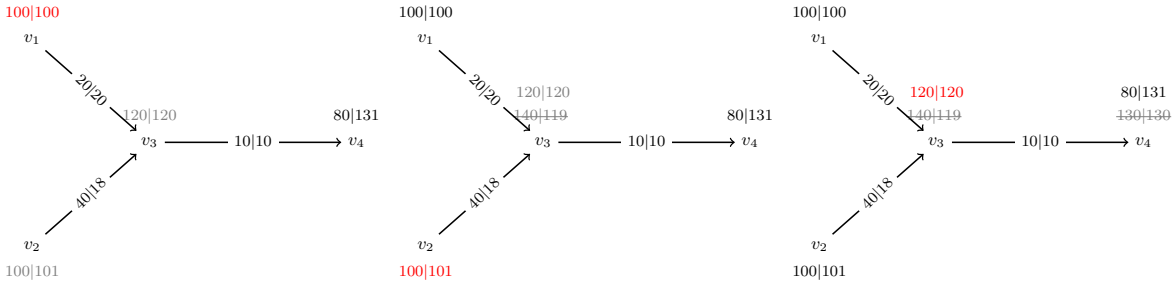

Figure 1: Three consecutive iterations of the MD-FPTAS. The extracted label ℓ^* in every iteration is marked in red, the permanent labels $\ell \in L_v$ in black, and the tentative labels generated in propagate or nextCandidateLabel in grey.

and use r as the approximation ratio in what follows. To obtain a runtime that is polynomial in the input size and in $1/\varepsilon$, we will set $r = (1+\varepsilon)^{\frac{1}{n-1}}$ and store at most one path per tensor entry. With this choice of r, the length of $T_{v,j}$ becomes $\left\lfloor \frac{n}{\varepsilon} \log(nC_j) \right\rfloor$ for any node $v \in V$ and $j \in \{2, \dots, d\}$ such that, in total, T_v stores at most $\prod_{i=2}^{d} |T_{v,j}|$, which is a polynomial upper bound on the number of computed permanent paths. The position of a (s,v)-path $p \in \mathcal{X}$ in T_v is computed using the function $\text{pos} : \mathbb{R}^d_{\geq} \to \mathbb{N}_0^{d-1}$ which is defined component wise as

$$\text{pos}_j(c(p)) := \begin{cases} 0 & \text{, if } c_j(p) = 0 \\ 1 + \left\lfloor \log_r \frac{c_j(p)}{c_j^{min}} \right\rfloor & \text{, else.} \end{cases} \quad , j \in \{2, \dots d\}. \tag{6}$$

The size of the tensors is well defined since efficient (elemental) paths have at most $(n-1)$ arcs and hence, $c_j(p) \leq (n-1)c_j^{\max}$.

For the MD-FPTAS we extend the label ℓ representing a path p by $\text{pos}(c(p)) = \text{pos}(c_\ell)$. 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 $\text{pos}_j(c_\ell) \leq \text{pos}_j(c_{\ell'})$ for $j \in \{2, \dots d\}$, we say that ℓ pos*-dominates* ℓ' and write $\ell \preceq_{\text{pos}} \ell'$. No further modifications w.r.t. Algorithm 1 are needed. 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. The MD-FPTAS computes at most one path per entry in T_v. Algorithms 4 and 5 show the pos-dominance tests to be embedded in propagate and nextCandidateLabel.

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

```
1 return L_v[|L_v| − 1] ⪯_pos l_v
```

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

```
1 for l ∈ L_v do
2 |   if l ⪯ pos l_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 1. *Figure 1 visualizes the situation at the end of each of three subsequent iterations*

of the MD-FPTAS. The shown graph is a subgraph of a larger graph with $n = 10$ nodes and we set $\varepsilon = 0.5$. Labels are represented only by their cost; their correspondence to nodes is made clear by their positioning. The example starts with a permanent label at node v_4 and labels for v_2 (100|101) and v_1 (100|100) in Q.

*In the **first iteration**, the latter is extracted and no new candidate label for v_1 is found. Then, the label (100|100) is propagated to node v_3 and the resulting label (120|120) is inserted into Q. In the **second iteration**, the label (101|100) is extracted from the heap and again, no new candidate label for v_2 is found. This label is then propagated to node v_3, where the resulting tentative label (140|119) is rejected, as v_3's current heap label, (120|120), is lexicographically smaller. In the **third iteration**, the label (120|120) at node v_3 is extracted from the heap. When nextCandidateLabel is called, the extension (140|119) of v_2's permanent label would be efficient in the exact scenario but is rejected yet again by the MD-FPTAS, since $120 < 140$ and $\text{pos}(120) = 107 = \text{pos}(119)$. The tentative label (130|130) at v_4 is then generated by propagation along the arc (v_3, v_4) but also rejected despite being efficient, since $80 < 130$ and $\text{pos}(130) = 109 = \text{pos}(131)$.*

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 $i \in \{1, \dots, N\}$ (including the i^{th} iteration) by L_v^i. Thus, $L_v = L_v^N$ is used only for 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 $i \in \{1, \dots, N\}$ in which ℓ_u is extracted from H and added to L_u^i. Let $k \in \{i+1, \dots, N\}$ 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. □

The following theorem proves the correctness of the MD-FPTAS for the static case. Its proof is similar to the one given in [4]. 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 $p^* = (s = v_0, v_1, \dots, v_k = v)$. Then, the MD-FPTAS finds an (s,v)-path $\tilde{p}$ s.t.*

$$c_j(\tilde{p}) \leq r^k c_j(p^*),\ \forall j \in \{1,\dots,d\}.$$

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 D. 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 $\tilde{\ell} \in L_v$ such that

$$c_1(\tilde{\ell}) \leq c_1(\ell^*) \quad \text{(7a)} \qquad \text{and} \qquad pos(\tilde{\ell}) \leq pos(\ell^*). \quad \text{(7b)}$$

For $j \in \{2,\dots,d\}$, we can derive $\log_r(c_j(\tilde{\ell})) - 1 \leq \log_r(c_j(\ell^*))$ from (7b) and this in turn can be restated as $c_j(\tilde{\ell}) \leq r c_j(\ell^*)$, which, coupled with (7a), proves the statement. In the **induction hypothesis**, we assume that

$$c_j(\tilde{p}) \leq r^{k-1} c_j(p^*),\ j \in \{1,\dots,d\}, \tag{8}$$

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 subpath p^*_{su} of p^* with $k-1$ arcs is efficient. In addition, the induction hypothesis guarantees the existence of a path $\tilde{p}_{su}$ with corresponding permanent label $\tilde{\ell}_{su}$ such that (8) holds for $\tilde{p}_{su}$ and p^*_{su}. When $\tilde{\ell}_{su}$ is extracted and made permanent, the label $\tilde{\ell} := (v, c_{\tilde{\ell}_{su}} + c((u,v)), \tilde{\ell}_{su})$ is analyzed in propagate. For the (s,v)-path $\tilde{p}$ corresponding to $\tilde{\ell}$, we have

$$c_j(\tilde{p}) = c_j(\tilde{p}_{su}) + c_j((u,v)) \leq r^{k-1} c_j(p^*_{su}) + c_j((u,v)) \leq r^{k-1} c_j(p^*),\ j \in \{1,\dots,d\}. \tag{9}$$

From the proof of the base case and the statement in 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 $c_j(\tilde{p}) \leq r^{k-1} c_j(p^*) \leq r^k c_j(p^*)$ for all $j \in \{1,\dots,d\}$ and we are done. If $\tilde{\ell}$ is discarded, there exists a permanent label $\ell \in L_v$ such that $c_1(\ell) \leq c_1(\tilde{\ell})$ and $pos(\ell) \leq pos(\tilde{\ell})$. The latter inequality implies $c_j(\ell) \leq r c_j(\tilde{\ell})$ for $j \in \{2,\dots,d\}$ and for $j = 1$, $c_1(\ell) \leq r c_1(\tilde{\ell})$ is trivially given. Combining this with (9) and considering the paths corresponding to the used labels, we get

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

which finishes the proof. □

From now on we assume that the arc cost functions of Dyn-MOSP instances are 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 positive. Note that in the proof of Theorem 1 we needed the arc cost vectors only in (9).

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

$$c_j(\tilde{p}) = c_j(\tilde{p}_{su}) + c_j(a, c(\tilde{p}_{su})) \overset{!}{\leq} r^{k-1} c_j(p^*_{su}) + r^{k-1} c_j(a, c(p^*_{su})) = r^{k-1} c_j(p^*). \tag{10}$$

To prove the following Lemma, we switch to a less cluttered notation. We assume that $f : \mathbb{R}_{\geq} \to \mathbb{R}_{\geq}$ is a continuous piecewise linear function 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 $f : \mathbb{R}_{\geq} \to \mathbb{R}_{\geq}$ be as described above and $\alpha \in \mathbb{R}_{>1}$ a constant. If f fulfills the FIFO property and the intercepts b_i of the affine functions building f are non-negative for all $i \in \{1, \dots, k-1\}$, then for points $x, y \in \mathbb{R}_{\geq}$ with $x \leq \alpha y$ there holds $x + f(x) \leq \alpha(y + f(y))$.*

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 $c_j((u, v), \cdot) = f$, $x = c_j(\tilde{p})$, $y = c_j(p^*)$, and $r^{k-1} = \alpha$ to get that (10) holds and formulate the following theorem:

Theorem 2 (FPTAS for Dyn-MOSP)**.** *Let $\mathcal{I}$ be a Dyn-MOSP instance with continuous piecewise linear and positive arc cost functions that fulfill the FIFO property. Additionally, let the functions $c_j((u, v), \cdot)$, $j \in \{2, \dots, d\}$ 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 algorithm.*

Proof. The proof is analogous to the one of Theorem 1 using (10) instead of (9) in the induction step. □

4.1 Complexity of the MD-FPTAS.

In this section, we set $C := \max_{j \in \{2, \dots, d\}} C_j$ and, as before, assume that the approximation ratio ε_j is equal in every dimension. 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 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 [2]. This follows directly from the monotonicity of the log function and 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 running time of the different FPTAS for MOSP that we are discussing in this paper.

[2]Note that for $d = 2$, $\text{pos}(\cdot)$ maps to $\mathbb{R}_{\geq}$ so it is well defined to talk about a label's pos value.

Table 2: Complexities of the different state of the art FPTAS for MOSP problems.

	TZ [24]	Martins FPTAS [4]	MD-FPTAS	MD-FPTAS_T_v (Sec. 4.1.1)
$d=2$	$\mathcal{O}\left(\frac{n^2m}{\varepsilon}\log(nC)\right)$	$\mathcal{O}\left(n^3\mathcal{T}^2\right)$	$\mathcal{O}\left((n\log n+m)\mathcal{T}\right)$	$\mathcal{O}\left((n\log n+m)\mathcal{T}\right)$
$d\geq 3$	$\mathcal{O}\left(nm\mathcal{T}\right)$	$\mathcal{O}\left(n^3\mathcal{T}^2\right)$	$\mathcal{O}\left((n\log n+m\mathcal{T})\mathcal{T}\right)$	$\mathcal{O}\left((n\log n+m)\mathcal{T}\right)$

4.1.1 Storing Tensors Explicitly.

While the complexity of the MD-FPTAS is lower than the one of the Martins-FPTAS, the FPTAS presented in [24] (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 $\mathrm{pos}(p)$ in T_v is checked: if it is empty, p is added; if a path exists already, only the one with the lowest cost in the exact dimension (in [24] 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 $T_v[\mathrm{pos}(\ell_v)] == 0$. In case the tensor entry is indeed set to 0, we add ℓ_v to L_v and set $T_v[\mathrm{pos}(\ell_v)] = 1$. 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 storage space of the MD-FPTAS. Moreover, in general, it will compute more labels than before since checking dominance using $\preceq_{\mathrm{pos}}$ is more restrictive than checking pos-equality. However, as in the TZ-FPTAS, the construction of the tensors T_v guarantees that the number of paths and iterations stays polynomially bounded. As a consequence, the running time of the MD-FPTAS for $d \geq 3$ objectives becomes

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

5 Computational Results

In this section we provide computational evidence of the improved running time of the MD-FPTAS when compared with the Martins-FPTAS presented in [4] that turned out to be faster than the TZ-FPTAS from [24].

Martins-FPTAS is based on the classical label setting algorithm for MOSP by Martins [17]. The data structures are similar to the ones 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 necessity to iterate 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 the 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 [4]. The first group of such instances consists of graphs that contain only efficient paths. These graphs were first described in [15] and are

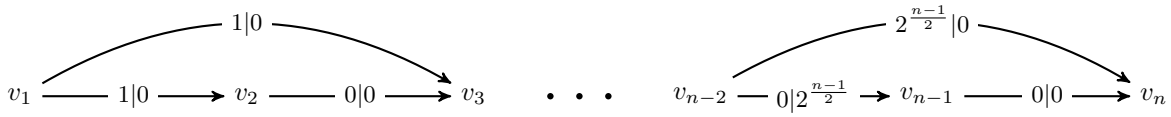

Figure 2: General EXP instance. Every path is efficient.

suitable to check the impact of the used approximation techniques (see Figure 2). We call these instances EXP and consider the corresponding graphs with 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 [19].

In the 3-dimensional case we consider a subset of the instances used in [9]. 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 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 on these grids was not possible without violating the time limit. Hence, we endorsed Algorithm 1 with the pruning techniques for MOSP instances described in [9]. It is easy to see that they are compatible with the approximation techniques used in this paper. In total, the GRID − 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 [3] and [2]. The directed graph in this instances 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 aircrafts are allowed to fly[3]. 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's consumption along an arc depending on the aircraft's 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 $500kg$. 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 [3]. 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.

[3]On Sky-Vector an airway network can be displayed.

Table 3: Results from one to all runs of bidimensional 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

5.2 Results

The experiments were ran on a machine with an Intel Xeon CPU E5-2670 v2 @ 2.50GHz processor. It has 2 CPUs per node and 10 cores per CPU. The available RAM was 128GB. 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 the Martins' based algorithms 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 $5400s$ 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 Bidimensional Results.

Table 3 summarizes the results of the biobjective MOSP instances. The running time advantage of the exact BDA on the `EXP` instances is remarkable. On average, it is 3450 times faster than Martins algorithm. 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 solutions are computed for the mentioned ε values.

Figure 3 gives a visual impression of the running times: on the left hand side we compare the BDA with the FPTAS-BDA and appreciate how the solution time of the exact algorithm grows exponentially with the number of computed solutions. 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.

The bidimensional `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 $r = (1+\varepsilon)^{1/n-1}$ is very close to 1 and the pos values of any two different paths are almost

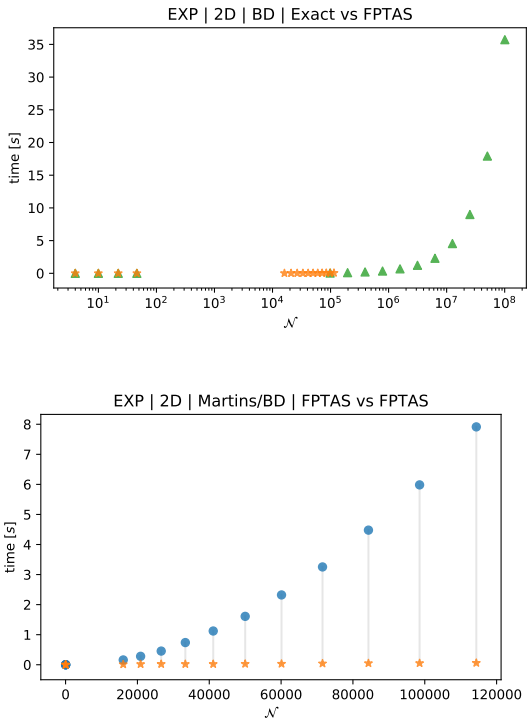

Figure 3: BDA exact (green), BD-FPTAS (yellow) and Martins FPTAS (blue) on exponential instances. $\varepsilon = 0.05$.

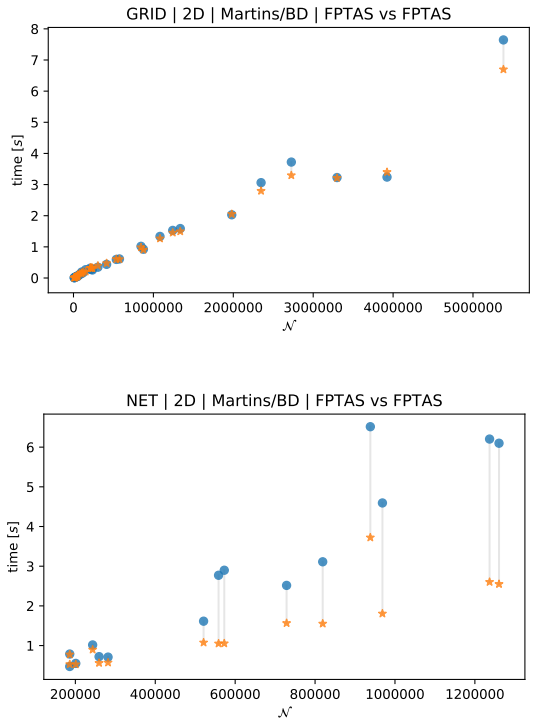

Figure 4: BD-FPTAS (yellow) and Martins FPTAS (blue) on grid instances (left) and netmaker instances (right)

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 [4] they overcome this problem by choosing huge values for ε and they compute an a posteriori approximation that always turns out to be much better than ε. We focus on realistic values for ε instead and see that the average FPTAS speedup on `GRID` instances and $\varepsilon = 0.05$ is $\times 1.66$ and on `NET` instances and $\varepsilon = 0.05$ it is $\times 3.23$. On both instance sets the FPTAS solutions contained almost all exact solutions and even increasing $\varepsilon = 0.05$ 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 4 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 solutions grows.

5.2.2 Static Three Dimensional Results

Instances with 3 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 $2D$ 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.

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

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 5 shows the comparison of both FPTAS' running times depending on the number of computed solutions. 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.

5.2.3 Dyn-MOSP results

Figure 6 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. Considering that we have chosen $\varepsilon = 0.5$, the efficiency of the FPTAS on these instances is remarkable. Additionally, in real world flight planning problems, the computation of every label is very expensive, hence giving the known FPTAS techniques and in particular the new MD-FPTAS a new impulse.

In Table 5 we depicted some geographically distant departure airports and show the impact of the FPTAS when computing routes to all possibly reachable nodes. We finish our computational experiments showing the less consumption and fastest routes from Berlin to Yekaterinburg in Figure 7. Even though consumption and time are correlated objectives, this example shows that both have to be considered since the routes vary considerably.

6 Conclusion

We have proven that Dynamic Multiobjective Shortest Path problems (Dyn-MOSP) can be solved by a generalization of the static Multiobjective Dijkstra (MD) algorithm if the arc cost

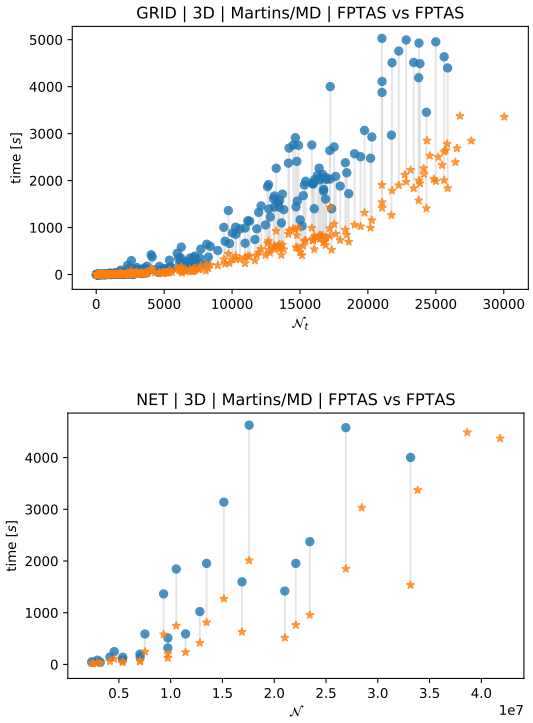

Figure 5: MD-FPTAS (yellow) and Martins FPTAS (blue) on grid instances (left) and netmaker instances (right)

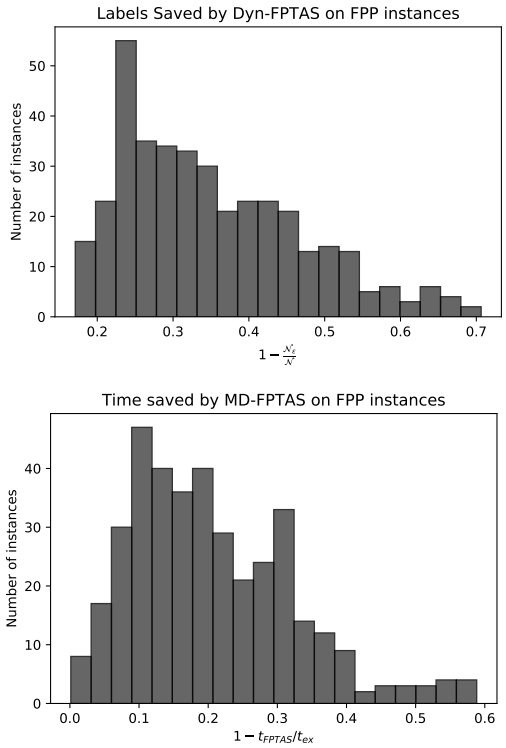

Figure 6: Distribution of percentage of labels saved by the MD-FPTAS in comparison to the exact MDA on FPP instances

Figure 7: Less consumption (left) and fastest (right) routes from Berlin to Yekaterinburg.

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 [24] to derive a new FPTAS for MOSP problems that is based on the label setting MD algorithm. The running time of the resulting MD-FPTAS is the number of computed solutions 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 three-dimensional instances, our MD-FPTAS was faster than the Martins-FPTAS introduced by Breugem et al. [4]. In the static case, we faced the same problem as the authors in [24] and [4]: 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. And indeed, using realistic values for ε, we computed $(1+\varepsilon)$-covers for these instances and saved 30% in terms of running time and 45% in terms of labels.

References

[1] Richard Bellman. The theory of dynamic programming. *Bull. Amer. Math. Soc.*, 60(6):503–515, 11 1954.

[2] Marco Blanco, Ralf Borndörfer, Nam Dung Hoàng, Anton Kaier, Pedro M. Casas, Thomas Schlechte, and Swen Schlobach. Cost Projection Methods for the Shortest Path Problem with Crossing Costs. In Gianlorenzo D'Angelo and Twan Dollevoet, editors, *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASIcs)*, pages 15:1–15:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[3] Marco Blanco, Ralf Borndörfer, Nam-Dung Hoang, Anton Kaier, Adam Schienle, Thomas Schlechte, and Swen Schlobach. Solving Time Dependent Shortest Path Problems on Airway Networks Using Super-Optimal Wind. In Marc Goerigk and Renato Werneck, editors, *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54 of *OpenAccess Series in Informatics (OASIcs)*, pages 12:1–12:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[4] Thomas Breugem, Twan Dollevoet, and Wilco van den Heuvel. Analysis of fptases for the multi-objective shortest path problem. *Comput. Oper. Res.*, 78:44–58, 2017.

[5] Fritz Bökler and Markus Chimani. *Approximating Multiobjective Shortest Path in Practice*, pages 120–133. Society for Industrial and Applied Mathematics, 2020.

[6] M.E. Captivo, J. Clímaco, J. Figueira, E. Martins, and J.L. Santos. Solving bicriteria 0-1 knapsack problems using a labeling algorithm. *Computers and Operations Research*, 30(12):1865–1886, 2003.

[7] J.C.N. Clímaco and M.M.B. Pascoal. Multicriteria path and tree problems: Discussion on exact algorithms and applications. *International Transactions in Operational Research*, 19(1-2):63–98, 2012.

[8] J. Current and M. Marsh. Multiobjective transportation network design and routing problems: Taxonomy and annotation. *European Journal of Operational Research*, 65(1):4–19, 1993.

[9] Pedro Maristany de las Casas, Antonio Sedeno-Noda, and Ralf Borndörfer. An asymptotically and computationally improved multiobjective shortest path algorithm. Technical Report 20-26, ZIB, Takustr. 7, 14195 Berlin, 2020.

[10] Yann Disser, Matthias Müller-Hannemann, and Mathias Schnee. Multi-criteria shortest paths in time-dependent train networks. *Exp. Algorithms Lecture Notes Comput. Sci*, 5038:347–361, 05 2008.

[11] M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum*, 22(4):425–460, 2000.

[12] Mathias Ehrgott and Xavier Gandibleux. *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys*. International Series in Operations Research & Management Science. Springer US, 2006.

[13] Michael Emmerich and André Deutz. A tutorial on multiobjective optimization: fundamentals and evolutionary methods. *Natural Computing*, 17, 05 2018.

[14] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, jul 1987.

[15] Pierre Hansen. Bicriterion path problems. In Günter Fandel and Tomas Gal, editors, *Multiple Criteria Decision Making Theory and Application*, pages 109–127, Berlin, Heidelberg, 1980. Springer Berlin Heidelberg.

[16] Michael M. Kostreva and Laura Lancaster. Multiple objective path optimization for time dependent objective functions. In Tadeusz Trzaskalik and Jerzy Michnik, editors, *Multiple Objective and Goal Programming*, pages 127–142, Heidelberg, 2002. Physica-Verlag HD.

[17] Ernesto Queiros Vieira Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2):236–245, May 1984.

[18] Christos Papadimitriou and Mihalis Yannakakis. On the approximability of trade-offs and optimal access of Web sources. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*. IEEE Comput. Soc, 2000.

[19] Andrea Raith and Matthias Ehrgott. A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research*, 36(4):1299–1331, apr 2009.

[20] Antonio Sedeño-noda and Marcos Colebrook. A biobjective dijkstra algorithm. *European Journal of Operational Research*, 276(1):106–118, jul 2019.

[21] P. Serafini. Some considerations about computational complexity for multiobjective combinatorial problems. *Recent advances and historical development of vector optimization*, 294:222–232, 1986.

[22] A.J.V. Skriver. A classification of bicriterion shortest path (bsp) algorithms. *Asia-Pacific Journal of Operational Research*, 17(2):199–212, 2000.

[23] Z. Tarapata. Selected multicriteria shortest path problems: An analysis of complexity, models and adaptation of standard algorithms. *International Journal of Applied Mathematics and Computer Science*, 17(2):269–287, 2007.

[24] George Tsaggouris and Christos Zaroliagis. Multiobjective optimization: Improved fptas for shortest paths and non-linear objectives with applications. In Tetsuo Asano, editor, *Algorithms and Computation*, pages 389–398, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[25] E.L. Ulungu and J Teghem. Multi-objective shortest path problem: A survey. *In D. Glückaufova, D. Loula, and M. Cerny, editors, Proceedings of the International Workshop on Multicriteria Decision Making: Methods - Algorithms - Applications at Liblice, Czechoslovakia. Institute of Economics, Czechoslovak Academy of Sciences, Prague*, pages 176–188, 1991.

[26] P. Vincke. Problemes multicriteres. *Cahiers du Centre d' Etudes de Recherche Operationelle*, 16:425–439, 1974.