Analysis of FPTASes for the Multi-Objective Shortest Path Problem

Thomas Breugem, Twan Dollevoet, Wilco van den Heuvel

Econometric Institute, Erasmus School of Economics
Erasmus University Rotterdam, Rotterdam, the Netherlands
breugem@ese.eur.nl, dollevoet@ese.eur.nl, wvandenheuvel@ese.eur.nl

Econometric Institute Report Series - EI2016-03

Abstract

We propose a new FPTAS for the multi-objective shortest path problem. The algorithm uses elements from both an exact labeling algorithm and an FPTAS proposed by Tsaggouris and Zaroliagis (2009). We analyze the running times of these three algorithms both from a theoretical and a computational point of view. Theoretically, we show that there are instances for which the new FPTAS runs an arbitrary times faster than the other two algorithms. Furthermore, for the bi-objective case, the number of approximate solutions generated by the proposed FPTAS is at most the number of Pareto-optimal solutions multiplied by the number of nodes. By performing a set of computational tests, we show that the new FPTAS performs best in terms of running time in case there are many dominated paths and the number of Pareto-optimal solutions is not too small.

Keywords: Shortest Path Problem, Multi-Objective Optimization, FPTAS, Complexity Analysis

1 Introduction

Multi-objective combinatorial optimization (MOCO), also called multi-criteria optimization, is a well-studied branch of optimization. The goal is to find optimal solutions based on multiple criteria. For example, consider the problem of designing a communication network between certain cities in which we do not only take into account cost, but also the failure probability of the network. This problem can be formulated as a bi-objective minimum spanning tree problem. Ulungu and Teghem (1994) discuss numerous multi-objective variants of well-known combinatorial optimization problems. The survey of Marler and Arora (2004) considers a more general setting, by also considering non-linear multi-objective problems.

Intuitively, it is clear that the presence of multiple objective functions leads to a more complex problem. Indeed, for a lot of MOCO problems it is shown that their corresponding decision versions are NP-complete (see, for example, Ehrgott (2000)), i.e., it is NP-complete to check whether a feasible solution exists which does not exceed a particular value for each objective (in case of minimization). For example, it can be shown that the multi-objective versions of well-known problems such as the shortest path and the minimum spanning tree problem are NP-hard, while efficient algorithms exist for the single-objective variants. This even holds for the multi-objective version of an unconstrained combinatorial optimization problem (i.e., the problem where the only constraint is that the variables are binary), for which the single-objective variant is trivial.

Besides the complexity of a MOCO problem, another issue is the size of a solution, which consists of all non-dominated points, known as the Pareto-optimal frontier. Typically, the cardinality of this set of points is exponential, and therefore these MOCO problems are called intractable (Ehrgott, 2000). For example, it can be shown that the multi-objective versions of the shortest path and the minimum spanning tree problem are intractable, even if we consider only two objective functions.

Due to the NP-completeness and intractability results for MOCO problems, one may consider alternative approaches for solving them, such as approximation algorithms. Approximation algorithms have the advantage over heuristics that the approximation ratio is guaranteed (i.e., the algorithm is constructed in such a way that a certain approximation ratio is always achieved). Papadimitriou and Yannakakis (2000) show that under general conditions a fully polynomial time approximation scheme (FPTAS) exists for a class of MOCO problems. They construct such an algorithm by a clever partition of the solution space, combined with solving the exact version of the single-objective variant (i.e., find a solution which has exactly a given objective value). However, the running times of such FPTASes are typically polynomials of high degree, and hence they may be slower than exact approaches when applied to relatively small instances.

The approximation results to multi-objective optimization problems have applications for single-objective non-linear optimization. Mittal and Schulz (2008) show that FPTASes for numerous combinatorial optimization problems with non-linear objectives can be constructed, provided there exists an FPTAS for the multi-objective variant with linear objectives. For example, if we want to find a path that is not only short but that also has a low probability of delay, then we can consider an objective function of the form $c_1(P) \cdot c_2(P)$, where $c_1(P)$ and $c_2(P)$ are the length and delay probability of path P, respectively. The results in Mittal and Schulz (2008) imply that there exists an FPTAS for this problem. A different type of approximation algorithm for such a problem, based on quadratic programming, is given in Kern and Woeginger (2007). However, the latter method cannot be generalized to problems with more than two objectives.

In this paper we propose a new FPTAS for the multi-objective shortest path (MOSP) problem. The shortest path problem is well-known and has many applications ranging from pure navigational problems to, for example, determining the structure of social networks. Deo and Pang (1984) give an overview of applications of the shortest path problem. For the linear single-objective version of the problem efficient algorithms exist. The most famous one is Dijkstra's labeling algorithm (Dijkstra, 1959), which solves the shortest path problem on n vertices with non-negative arc cost in $\mathcal{O}\left(n^2\right)$ time. Ahuja et al. (1990) discuss how Dijksta's algorithm can be implemented more efficiently using different data structures.

The multi-objective shortest path problem, and in particular the bi-objective version, is well-studied. Hansen (1980) proves the intractability of the bi-objective version. In Section 2.1 we show a similar result by providing a MOSP instance on a complete digraph in which every path is Pareto optimal, implying that the MOSP problem is intractable. In general, such a worst case bound is often not achieved. In this light, interesting results can be found in Müller-Hannemann and Weihe (2006), who derive conditions under which the cardinality of the Pareto-optimal frontier is polynomial. Serafini (1987) shows that the decision version of the MOSP problem is NP-complete. The results of Hansen (1980) and Serafini (1987) imply that the Pareto-optimal frontier of large MOSP instances cannot be constructed in a reasonable amount of time. A parametric approach to the bi-objective problem is given in Mote et al. (1991), and in Sedeño-Noda and Raith (2015). The latter focuses on finding all extreme supported non-dominated points (i.e., the extreme points on the convex hull of the Pareto-optimal set), which is done using a Dijkstra-like algorithm. Mulmuley and Shah (2001) show that the number of breakpoints in the parametric shortest path problem can be sub-exponential, implying that the number of extreme supported non-dominated points is sub-exponential in the worst case. This shows that the problem of finding extreme supported non-dominated points is still intractable.

A general overview of algorithms for the MOSP problem is given in Skriver (2000). Paixão and Santos (2013) conduct a comparative computational study among different labeling algorithms. Tarapata (2007) provides an extensive review of recent results, including results on approximating the Pareto-optimal frontier. Among those is the algorithm of Tsaggouris and Zaroliagis (2009), on which we will build our proposed algorithm (which we discuss in Section 3.2.2). As we will see, the running time of the FPTAS of Tsaggouris and Zaroliagis (2009) does not depend on the cardinality of the Pareto-optimal frontier, in contrast to, for example, the exact approach as mentioned in Ehrgott (2000). As a result, the FPTAS may be slower than the exact approach in case of instances with a relatively small Pareto-optimal frontier, which is typically observed for practical or randomly generated instances. By combining both algorithms we are able to obtain an FPTAS for which the running time depends on the cardinality of the Pareto-optimal frontier, and hence is faster than the FPTAS of Tsaggouris and Zaroliagis (2009) in the case of such instances.

Summarizing, the contribution of our paper is as follows: (i) we provide a MOSP instance having the maximum number of Pareto-optimal points (with respect to the number of nodes), (ii) to the best of our knowledge, we are the first to identify the running time of an exact label setting algorithm with respect to the cardinality of the Pareto frontier, (iii) we propose a new FPTAS with the feature that its running time depends on the number of Pareto-optimal points, (iv) we perform a detailed running time analysis in which we compare the proposed FPTAS to an exact algorithm and the FPTAS of Tsaggouris and Zaroliagis (2009), (v) we prove that, in the bi-objective case, the number of paths generated by our FPTAS is bounded by the number of Pareto-optimal points times the number of nodes, and (vi) to the best of our knowledge, we are the first to test FPTASes for the MOSP problem in a computational study.

The remainder of the paper is organized as follows. In Section 2, we introduce the basic concepts of MOCO problems, define the MOSP problem, discuss its intractability, and introduce the concept of approximation algorithms for MOCO problems. In Section 3, we turn to the main part of the paper, in which we propose a new FPTAS for the MOSP problem. Since our FPTAS builds on the exact labeling method described in Ehrgott (2000) and the FPTAS proposed by Tsaggouris and Zaroliagis (2009), we discuss these algorithms in some detail first. In Section 4, we conduct a computational study to test the performance of the three algorithms. To the best of our knowledge, no FPTASes for the MOSP problem have been tested before. We generate instances that are in the spirit of three types of complexity analysis: worst case, average case and smoothed complexity. The former two types are well-known, while the latter is relatively new (see, for example, Spielman and Teng (2009)). The paper is concluded in Section 5.

2 Problem Description and Definitions

2.1 Multi-objective optimization problems

Before we discuss solution approaches for MOCO problems in detail, we formalize a multi-objective optimization problem, and a solution to such a problem. In the remainder of the paper, we consider multi-objective optimization problems defined as follows (see also Ehrgott (2000)).

Definition 2.1. *An instance π of a multi-objective optimization problem Π is given by d objective functions, denoted by $f_1, \ldots, f_d$. Every f_i maps the set of feasible solutions, denoted as X, to the positive real numbers, i.e., $f_i : X \to \mathbb{R}_+$, for $i = 1, \ldots, d$.*

In general, there is no solution that minimizes all objective functions simultaneously. Therefore, a solution to a multi-objective optimization problem is given by a Pareto-optimal frontier.

Definition 2.2. *Let π be an instance of a multi-objective minimization problem. A Pareto-optimal frontier, denoted by $P(\pi)$, is a set of solutions such that $x \in P(\pi)$ if and only if there is no $x' \in X$ for which $f_i(x') \leq f_i(x)$, for $i = 1, \ldots, d$, with strict inequality for at least one i.*

Hence the Pareto-optimal frontier $P(\pi)$ (also called Pareto curve) is the set of all non-dominated solutions (also called efficient solutions in the literature). The concept of a Pareto-optimal frontier is visualized in Figure 1. The figure shows a bi-objective minimization problem, where the points in the plane indicate objective values corresponding to feasible solutions. The Pareto-optimal frontier consists of all the non-dominated solutions, which are depicted by a $\otimes$.

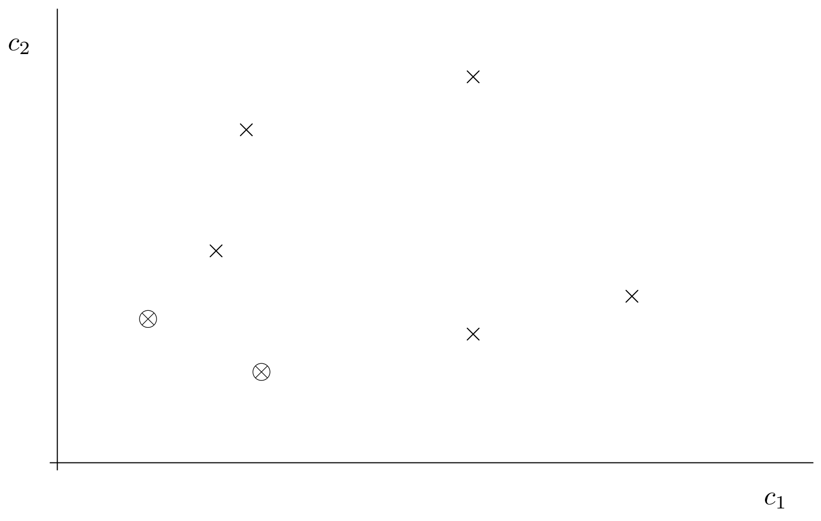

Figure 1: Example Pareto-optimal frontier

As mentioned before, in the worst case the cardinality of $P(\pi)$ is exponential, and hence the problem becomes intractable (since any algorithm should examine each solution in this set at least once).

2.2 Multi-objective shortest path problem

We now turn to the shortest path problem, one of the most well-known problems in combinatorial optimization. Given a directed graph $G = (V, A)$ with vertex set V and arc set A, the problem is to find a path between two given nodes s and t that minimizes a given cost function c. In this paper we consider the case where the arc costs are non-negative integers, i.e., $c : A \to \mathbb{Z}_+$. This naturally generalizes to the MOSP problem by considering a cost function $c : A \to \mathbb{Z}_+^d$, for some positive integer d.

This version of the shortest path problem is also known as the one-to-one or shortest s, t-path problem and is one among several variations. One of those variations is the one-to-all or single-source shortest path problem. In this problem a node s is given, and we are asked to determine all shortest s, v-paths, for each $v \in V$. Although one can simply take only the s, t-path from the returned solution of a single-source shortest path problem to obtain a solution for the shortest s, t-path problem, it is important for comparing running times which one of the two variants is solved. In this paper we consider algorithms for the single-source shortest path problem.

We will use the following notation throughout the paper. For a general directed graph $G = (V, A)$, define $n \equiv |V|$ as the number of vertices and $m \equiv |A|$ as the number of arcs. We extend the cost function to paths by defining $c(p) \equiv \sum_{a \in p} c(a)$ as the cost of a path p. Hence, $c_i(p)$ is the cost of path p in terms of the i^{th} objective function.

Unfortunately, as mentioned before, the shortest path problem is among the combinatorial optimization problems for which the generalization to multiple dimensions leads to intractability. Hansen (1980) shows that for a bi-objective shortest path instance π, the cardinality of the Pareto-optimal frontier $P(\pi)$ can be $2^{\frac{n}{2}} = (\sqrt{2})^n$. We strengthen this result with respect to the number of nodes n by showing that a number of 2^{n-2} Pareto-optimal paths can be achieved. We note that the instance in Hansen (1980) has $\frac{3}{2}(n-1)$ arcs while our instance has $\frac{1}{2}n(n-1)$ arcs, and hence the instance of Hansen (1980) can be viewed as stronger with respect to the number of arcs.

Theorem 2.1. *There exists a MOSP instance π with $|P(\pi)| = 2^{n-2}$.*

Proof. Consider the sets $V = \{0, 1, \ldots, n\}$ and $A = \{(i, j) \in V \times V : i < j\}$. Then the graph

$G = (V, A)$ is a directed and acyclic graph. Note that the graph G contains $n + 1$ nodes. Define two cost functions on A as

$$w_1(i, j) = 2^{j-1} \quad \text{and} \quad w_2(i, j) = 2^{j-1} - 2^i.$$

It then holds for all arcs $(i, j) \in A$ that

$$(w_1 + w_2)(i, j) = 2^{j-1} + 2^{j-1} - 2^i = 2^j - 2^i.$$

For any path p from 0 to n, it follows, that

$$(w_1 + w_2)(p) = 2^n - 1. \tag{1}$$

Considering only the first objective, it holds for every number $2^{n-1} \leq q < 2^n$ that there is a path p such that $w_1(p) = q$. This can be seen by considering the binary expansion of q. For every number $2^{n-1} \leq q < 2^n$ there is a unique set $V_q \subseteq \{1, \ldots, n\}$, such that

$$q = \sum_{j \in V_q} 2^{j-1}.$$

Recalling that $q \geq 2^{n-1}$, it holds that $n \in V_q$ for all q under consideration. Define now the path p from 0 to n containing exactly all vertices $j \in V_q$. Then, it holds that

$$w_1(p) = \sum_{(j',j) \in p} 2^{j-1} = \sum_{j \in V_q} 2^{j-1} = q.$$

From (1), we conclude that $w_2(p) = 2^n - 1 - q$. The cost for this path is thus equal to $(w_1(p), w_2(p)) = (q, 2^n - 1 - q)$. As a path can be constructed for all $q \in \{2^{n-1}, \ldots, 2^n - 1\}$, we find $2^n - 2^{n-1} = 2^{n-1}$ different values for the objective. This shows that there are 2^{n-1} Pareto-optimal paths. Given that there are 2^{n-1} different $0, n$-paths in this graph, we conclude that all $0, n$-paths in the graph are Pareto-optimal. Finally, the theorem follows by considering an equivalent graph defined on n nodes instead of $n + 1$ nodes as in the proof. □

The intuition of the proof is that for every node j in $V \setminus \{0, n\}$ we decide whether it is in the path or not. If vertex j is in the path, we add 2^{j-1} to the first objective; if vertex j is not in the path we add 2^{j-1} to the second objective. Selecting an arc (i, j) in the path means that vertex j is selected, and the nodes between i and j are not. Hence, we add 2^{j-1} to w_1 and

$$\sum_{k=i+1}^{j-1} 2^{k-1} = \sum_{k=0}^{j-1} 2^{k-1} - \sum_{k=0}^{i} 2^{k-1} = (2^{j-1} - 1) - (2^i - 1) = 2^{j-1} - 2^i$$

to w_2.

The objective values of the paths in the worst case instance are not symmetric: The first objective ranges between 2^{n-1} and $2^n - 1$, whereas the second one ranges between 0 and $2^{n-1} - 1$. In order to obtain a symmetric instance, we subtract 2^{n-1} from the first objective for all paths. This can be realized by redefining the arc costs as $w_1(i, n) = 0$ for all $i \in \{0, \ldots, n - 1\}$. In the remainder of this paper, we assume that the arc costs in the worst case instances have been redefined.

2.3 Approximation Schemes

By the results of Hansen (1980) and Section 2.1, it is clear that the MOSP is intractable. To circumvent the issue of intractability that is common among many multi-objective optimization problems, one may consider an approximate Pareto-optimal frontier. Such a solution approach is based on the approximation schemes well-known for single-dimensional optimization problems.

Informally, for a given instance of an optimization problem, an approximation scheme or approximation algorithm returns an 'almost optimal' solution, i.e., a solution of which the objective never differs more than a certain factor from the optimal objective value. Such algorithms provide alternative approaches for NP-complete problems which cannot be solved efficiently (assuming $\mathsf{P} \neq \mathsf{NP}$). By efficient we mean that the algorithm runs in time polynomial in the size of the instance (i.e., the size of the input to the algorithm). Clearly, one wants to know how large the difference between the optimal solution and the approximate solution is, and therefore approximation algorithms are classified based on their difference in worst case ratio.

Formally, let Π be a minimization (maximization) problem. We say that algorithm $\mathcal{A}$ is an ε-approximate algorithm for Π if, for every instance π of Π, $\mathcal{A}$ returns a solution of which the objective value is at most $1+\varepsilon$ (at least $1-\varepsilon$) times that of the optimal solution, for some fixed $\varepsilon > 0$. A well-known example of such an algorithm is the $\frac{1}{2}$-approximation algorithm for the metric TSP (that is, the Traveling Salesman Problem with triangle inequality) given in Christofides (1975). Obviously, such algorithms are only relevant when they run substantially faster than any (known) exact algorithm.

When for an optimization problem there exists an efficient ε-approximation algorithm for every $\varepsilon > 0$, we say the problem admits a polynomial time approximation scheme (PTAS). Formally, a PTAS for an optimization problem Π is a family of algorithms such that for every $\varepsilon > 0$ there is an algorithm $\mathcal{A}_\varepsilon$ which is an ε-approximate algorithm for Π and runs in time polynomial in the input size. If the running time is also polynomial in $1/\varepsilon$, this family of algorithms is called a fully polynomial time approximation scheme (FPTAS). FPTASes are of much practical importance, since they allow a trade-off between solution quality and resources such as computation time and memory. For a lot of well-known problems, such as the knapsack problem, FPTASes are known. Woeginger (1999) gives a general framework for designing FPTASes for single-objective problems based on dynamic programming techniques. One can see the existence of an FPTAS as the best possible approximation result for an NP-complete problem (again, assuming $\mathsf{P} \neq \mathsf{NP}$).

Approximate solutions generalize to the multi-objective case as follows.

Definition 2.3. *Let π be an instance of a multi-objective minimization problem. For $\varepsilon > 0$, an ε-approximate Pareto-optimal frontier, denoted by $P_\varepsilon(\pi)$, is a set of solutions, such that for all $x \in X$, there is an $x' \in P_\varepsilon(\pi)$ for which $f_i(x') \leq (1+\varepsilon)f_i(x)$, for all $i = 1, \ldots, d$.*

Loosely speaking, $P_\varepsilon(\pi)$ is a set of solutions which almost dominate all solutions. An FPTAS for a multi-objective optimization problem is now an algorithm that finds an ε-approximate Pareto-optimal frontier to a problem in time polynomial in both the input size and $1/\varepsilon$. Note that the cardinality of an approximate Pareto-optimal frontier obtained by any FPTAS should be polynomial in both the input size and $1/\varepsilon$ as well, since each element in the approximate Pareto-optimal frontier should be considered at least once in the FPTAS. Papadimitriou and Yannakakis (2000) show that under general conditions $P_\varepsilon(\pi)$ can be constructed in polynomial time.

Figure 2 gives an example of an approximate Pareto-optimal frontier (the approximation factor ε is chosen arbitrarily and slightly larger than 1). The point depicted by a $\otimes$ is the only solution in the approximate Pareto-optimal frontier. The highlighted area indicates which solutions are 'covered' by this solution x, i.e., the area contains all points (a_1, a_2) with $f_i(x) \leq (1+\varepsilon)a_i$, for $i = 1, 2$. We note that the set $P_\varepsilon(\pi)$ is not unique: the set consisting of the two Pareto-optimal points (the points depicted by a $\otimes$ in Figure 1) qualifies as well. Furthermore, as Figure 2 illustrates, the approximate Pareto-optimal frontier can consist entirely of dominated solutions.

3 Solution Methods

In this section, we present solution methods for the MOSP problem. We first discuss an exact algorithm (see Ehrgott (2000)). We show that the running time of the exact algorithm depends quadratically on the number of Pareto-optimal paths. Then, we explain the main ideas of the FPTAS proposed in Tsaggouris and Zaroliagis (2009). In turn, we develop a new FPTAS for the MOSP, that combines the benefits of the other two algorithms. Finally, we perform a comparative analysis between the running times of the exact algorithm and the two FPTASes.

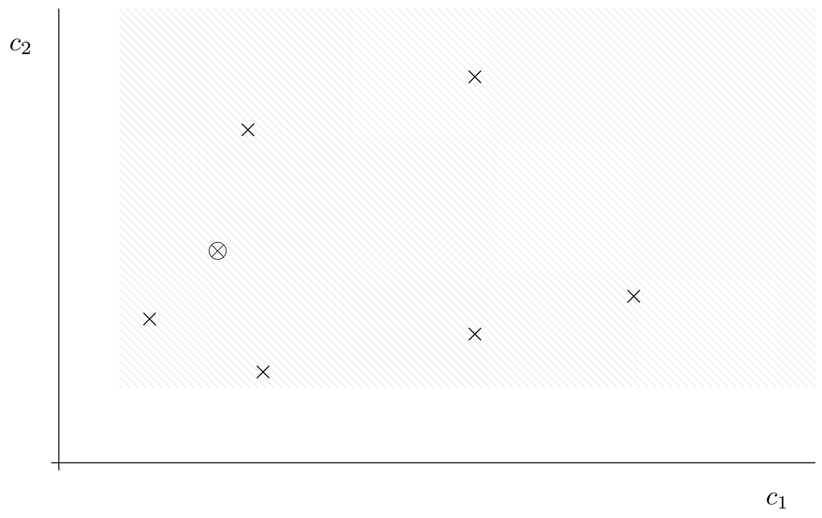

Figure 2: Example approximate Pareto-optimal frontier

3.1 Exact Solution Method

The MOSP problem can be solved using a labeling algorithm similar to Dijkstra's algorithm for the shortest path problem with a single objective. In the algorithm, every path p is represented by a label $(c(p), v(p), a(p), pred(p))$. Here, $c(p)$ denotes the d-dimensional cost of path p. The last vertex and arc on path p are denoted by $v(p)$ and $a(p)$, respectively. Finally, $pred(p)$ is a pointer to the label of the path that is obtained by removing $v(p)$ and $a(p)$ from p. From here on, we use the names path and label interchangeably. An overview of the exact algorithm is given in Algorithm 1.

Algorithm 1: An exact algorithm for the MOSP

```
Data: A graph $(V, A)$, a vertex $s \in V$, a $d$-dimensional cost function $c : A \to \mathbf{Z}_+^d$
Result: The Pareto-optimal $s, v$-paths for all $v \in V$
Init: Define $\mathcal{L}_T = \{(0, s, -, -)\}$ and $\mathcal{L}_P = \emptyset$;
while $\mathcal{L}_T$ is not empty do
    Remove the lexicographically smallest label $w \in \mathcal{L}_T$;
    Add $w$ to $\mathcal{L}_P$;
    for all outgoing arcs $a = (v(w), u)$ of $v(w)$ do
        Generate the label $w' = (c(w) + c(a), u, a, w)$ by extending $w$ along the arc $a$;
        if $w'$ is not dominated by any label in $\mathcal{L}_P$ or $\mathcal{L}_T$ then
            Add $w'$ to $\mathcal{L}_T$;
        end
        if $w'$ dominates any labels $w''$ in $\mathcal{L}_T$ then
            Remove $w''$ from $\mathcal{L}_T$;
        end
    end
end
```

The algorithm maintains two sets of labels; one set, denoted by $\mathcal{L}_T$, contains the temporary labels, while the other set, denoted by $\mathcal{L}_P$, contains all permanent labels. As long as the set $\mathcal{L}_T$ is not empty, the algorithm takes a lexicographically smallest label from the set, say label w. It then removes this label w from the set of temporary labels and adds it to the set of permanent labels. Furthermore, for each outgoing arc a of $v(w)$, a new label w' is generated by expanding w along a. This new label w' is added to $\mathcal{L}_T$, unless w' is dominated by one of the labels already generated. Moreover, if any temporary label w'' is dominated by the newly created label w', we delete w''

from $\mathcal{L}_T$. When checking for dominance, we only consider labels that end at the same node: a path w dominates a path w' if and only if $v(w) = v(w')$, $c(w) \leq c(w')$ and $c_i(w) < c_i(w')$ for at least one index $i = 1, \ldots, d$. The algorithm finds all Pareto-optimal s, v-paths, for all $v \in V$. For a more detailed description and a proof of correctness of the algorithm, we refer to Ehrgott (2000). An example of the Pareto-optimal frontier obtained by this algorithm is depicted in the left graph in Figure 3. Here, we have $d = 3$ and assume, for graphical purposes, that all paths w satisfy $c_3(w) = 1$. The dominated paths are depicted by a $\times$, the Pareto-optimal paths by a $\otimes$.

Recall that the Pareto-optimal frontier possibly has an exponential size. Hence, the worst case performance of any exact algorithm is exponential. We now show that the running time of the algorithm is quadratic in the number of Pareto-optimal paths. In order to do so, consider a MOSP instance π. Note that the algorithm finds the Pareto-optimal paths from s to v for all $v \in V$, and hence solves the single-source shortest path problem. Therefore, we want to express the running time as a function of the sum of the number of Pareto-optimal paths over all $v \in V$. Due to the intractability results from Section 2.2, the worst case performance of the algorithm is exponential, and hence running time analysis for this algorithm is often omitted. In fact, to the best of our knowledge, no exact running time analysis has been done in the literature. We often observed, however, small Pareto-optimal frontiers and a good performance of the algorithm, which led to developing a new FPTAS based on this labeling algorithm. We therefore state the running time explicitly as a function of the number of Pareto-optimal paths. We assume here that the labels are grouped per end vertex, and thus that a dominance check only involves paths that correspond to the same end vertex.

Lemma 3.1. *Let K_v be the number of Pareto-optimal s, v-paths for $v \in V$. Furthermore, let K be the total number of Pareto-optimal paths, i.e., $K \equiv \sum_{v \in V} K_v$. The algorithm finds the Pareto-optimal frontier in $\mathcal{O}\left(nK^2\right)$ time.*

Proof. Since the total number of Pareto-optimal paths is K, we add K labels to $\mathcal{L}_P$. At most n arcs are incident to every vertex $v \in V$. Hence, for each label that is added to $\mathcal{L}_P$, we create at most n new labels. Hence we create $\mathcal{O}(nK)$ labels in total. When creating a new label we have to check whether it dominates any label in $\mathcal{L}_T$ or is dominated by any label in $\mathcal{L}_P$ or $\mathcal{L}_T$. Observe that there can be at most K labels in $\mathcal{L}_T$ with the same end node. This observation follows from the fact that we make K labels permanent and that every label is extended at most once to a certain vertex $v \in V$. It follows that the dominance check for a newly created label involves $\mathcal{O}(K + K) = \mathcal{O}(K)$ steps: we check all permanent labels with the same end node (which is bounded by K) and all temporary labels with the same end node (which is also bounded by K). While checking for dominance, we can simultaneously keep the set $\mathcal{L}_T$ in lexicographic order. Since the dominance check is done for all created labels, the total running time becomes $\mathcal{O}\left(nK^2\right)$. $\square$

The conclusions in Lemma 3.1 are valid for any label-setting algorithm. The correctness of such an algorithm requires an ordering of the temporary labels such that the first label in the list (which is made permanent) is guaranteed to be a Pareto-optimal solution. In our case the lexicographical ordering is used, but an ordering based on the sum, norm or max function work as well. Paixão and Santos (2013) compare label setting algorithms using many different orderings.

3.2 FPTASes for the MOSP

We now turn to discussing two FPTASes for the MOSP problem. The first one was introduced in Tsaggouris and Zaroliagis (2009), which we will refer to as the TZ FPTAS. In the second one, we incorporate the dominance checks from the exact approach into the first FPTAS and thereby reduce the running time for graphs with relatively small Pareto-optimal frontiers.

3.2.1 Preliminaries FPTASes

The first FPTAS that we discuss computes the approximate Pareto-optimal frontier for all s, v-walks of size at most $n-1$, instead of all s, v-paths. Not all the walks returned by FPTASes are necessarily paths, due to the fact that we consider an *approximate* Pareto optimal frontier. We denote by $P^i(s,v)$ and $W^i(s,v)$ the set of all s, v-paths and s, v-walks, respectively, containing at most i arcs. Note that in a path, a vertex is only visited once, while in a walk a vertex may be visited several times. The set of all s, v-paths and s, v-walks is denoted as $P(s,v)$ and $W(s,v)$, respectively. Note that $P(s,t)$ is a subset of $W^{n-1}(s,t)$. Furthermore, any walk can be truncated to obtain a path whose cost is never higher, because the arc costs are assumed non-negative. Hence, we obtain an approximate Pareto-optimal frontier for the original problem by truncating all walks in a post-processing step. We note that the FPTAS we propose in Section 3.2.3, although concerning walks, always returns a set of paths, due to the dominance checks.

For convenience, we introduce some additional notation in this section. Given an instance of the MOSP as discussed in Section 2.2, define $C_j = \max_{a \in A} c_j(a)$ for $1 \leq j \leq d$ and $C^{\max} \equiv \max_{1 \leq j \leq d} C_j$.

3.2.2 FPTAS of Tsaggouris and Zaroliagis (2009)

In this section, we discuss the TZ FPTAS in detail. Their algorithm resembles the Bellman-Ford labeling algorithm (see Ehrgott (2000)), and is remarkably elegant. The main idea of the algorithm is to store all labels in arrays of polynomial size by relaxing the assumption of non-dominated solutions. That is, by looking at $P_\varepsilon(\pi)$ instead of $P(\pi)$, the number of labels that need to be stored is polynomial. The algorithm determines the approximate Pareto-optimal frontier for the s, v-walks for all $v \in V$.

Similarly as in Section 3.1, we represent a walk w by a label $(c(w), v(w), a(w), \mathit{pred}(w))$, that contains the cost vector $c(w)$, the last vertex $v(w)$, the last arc $a(w)$ and a pointer $\mathit{pred}(w)$ to the label of the walk obtained by removing $v(w)$ and $a(w)$ from w.

Consider now a multi-objective shortest path instance π. In order to store all relevant labels, we consider $(d-1)$-dimensional arrays of labels, denoted by $\mathcal{L}_v^i$, for $i = 0, \ldots, n-1$ and $v \in V$, with size

$$1 + \lfloor \log_r ((n-1)C_j) \rfloor$$

in the j^{th} dimension, for $j = 1, \ldots, d-1$. Here r is a parameter that depends on the approximation factor and will be determined later. The arrays $\mathcal{L}_v^i$ will contain the labels of the approximately Pareto-optimal s, v-walks with at most i arcs. We first define the function pos_j for $j = 1, \ldots, d-1$ by

$$\mathit{pos}_j(w) = \begin{cases} 0 & \text{if } c_j(w) = 0 \\ 1 + \lfloor \log_r c_j(w) \rfloor & \text{otherwise.} \end{cases}$$

Here we assume w.l.o.g. that $\log_r c_j(w) \geq 0$ whenever $c_j(w) > 0$. For a walk w, we then define the index in the arrays as

$$\mathit{pos}(w) = [\mathit{pos}_1(w), \ldots, \mathit{pos}_{d-1}(w)] \,.$$

It is clear that the size of the arrays is sufficient to store any $w \in W^{n-1}$, where $W^{n-1} \equiv \bigcup_{v \in V} W^{n-1}(s,v)$.

The algorithm proceeds as follows. Initially, the arrays $\mathcal{L}_v^0$ contain no labels if $v \in V - \{s\}$, while $\mathcal{L}_s^0$ contains only the trivial, empty path. In each iteration $1 \leq i \leq n-1$ we compute $\mathcal{L}_v^i$ for all v as follows. First we set $\mathcal{L}_v^i$ equal to $\mathcal{L}_v^{i-1}$. Then, for every arc $(u,v) \in A$, we extend all labels in $\mathcal{L}_u^{i-1}$ with the arc (u,v), i.e., for every label $w \in \mathcal{L}_u^{i-1}$ we create a new label $w' = (c(w) + c(u,v), v, (u,v), w)$. We then put this label at position $\mathit{pos}(w')$ in $\mathcal{L}_v^i$, unless there is a label w'' at that position for which $c_d(w'') \leq c_d(w')$. In particular, if there is no label yet, we always add w'.

The next lemma shows how the algorithm can be used to obtain a $(1+\varepsilon)$-approximation (see Tsaggouris and Zaroliagis (2009) for the proof).

Lemma 3.2 (Tsaggouris and Zaroliagis (2009, Lemma 1))**.** *For all $i \leq n-1$ and $v \in V$, after the i^{th} round $\mathcal{L}_v^i$ is an r^i-approximate Pareto frontier for $W^i(s,v)$.*

Clearly, by setting $r = (1+\varepsilon)^{1/n-1}$ we obtain the desired $(1+\varepsilon)$-approximation. Lemma 3.2 does not only hold for a one parameter approximation, i.e., the same approximation factor holds in each dimension, but it can easily be generalized to different approximation factors for each dimension (again, we refer to Tsaggouris and Zaroliagis (2009) for the details).

Furthermore, note that the algorithm is exact in one dimension (in this case, the d^{th} dimension). This means that for every path w, there exists a path w' in the approximate Pareto-optimal frontier, such that $c_j(w') \leq (1+\varepsilon)c_j(w)$ for all $j = 1, \dots, d-1$ and $c_d(w') \leq c_d(w)$. In other words, the approximation factor $1+\varepsilon$ reduces to 1 in the d^{th} dimension.

The idea of the algorithm is illustrated in Figure 3. Note that the algorithm is exact in the c_3-dimension. Therefore, for each position value (corresponding to a dashed rectangle), the point with lowest c_3 is added. We should stress that the algorithm does not consider dominance when creating the labels, as we observe from the selected points in the right graph in Figure 3. Here, in every dashed rectangle, exactly one path is selected, even if the entire rectangle is dominated by other points.

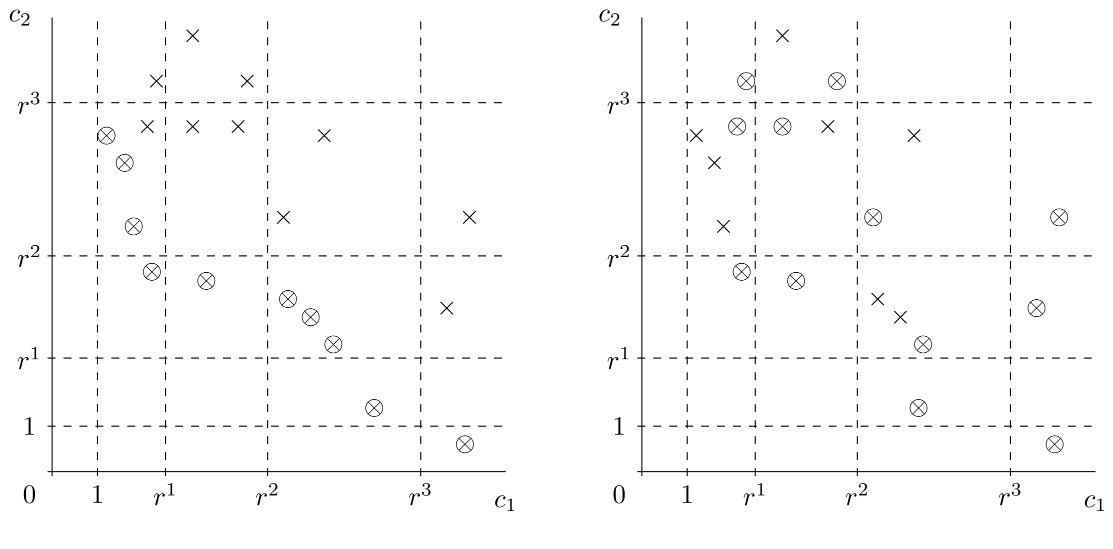

Figure 3: The frontiers obtained by the exact algorithm (on the left) and by the TZ FPTAS (on the right). In both cases, the frontiers contain only the points denoted by $\otimes$. We assume that $d = 3$ and that $c_3(w) = 1$ for all paths w.

We now consider the running time of this algorithm. The following lemma states the running time of the TZ FPTAS. Again, for the proof, we refer to Tsaggouris and Zaroliagis (2009).

Lemma 3.3 (Tsaggouris and Zaroliagis (2009, Corollary 1))**.** *The algorithm constructs an ε-approximate Pareto-optimal frontier for π in $\mathcal{O}\left(nm\left(n\log(nC^{max})/\varepsilon\right)^{d-1}\right)$ time.*

3.2.3 Hybrid FPTAS

In this section, we propose an FPTAS that combines features of both the exact MOSP algorithm (see Section 3.1) and the TZ FPTAS (see Section 3.2.2) and hence we refer to it as the hybrid FPTAS. More precisely, we propose an algorithm which combines the position function used by Tsaggouris and Zaroliagis (2009) with the dominance checks and lexicographic ordering used by the exact labeling algorithm. As a consequence, our hybrid FTPAS is also exact in the d^{th} dimension.

The main motivation for this approach is that one often observes small Pareto-optimal frontiers in numerical experiments (see, for example, Paixão and Santos (2013)). For such instances with small Pareto-optimal frontiers, our FPTAS takes advantage of intermediate dominance checks in order to keep the set of temporary labels small. This contrasts the TZ FPTAS, which only considers the position function when adding labels, and hence profits little from small Pareto-optimal frontiers. On the other hand, when the Pareto-optimal frontier is large, the use of the position function keeps the running time of our FPTAS polynomial in n and $1/\varepsilon$. For our new algorithm, we have to be careful how to apply the position function, order the labels and perform dominance checks in order to assure a correct approximation of the Pareto-optimal frontier and a fully polynomial running time.

Algorithm 2: A new FPTAS for the MOSP

```
Data: A graph $(V, A)$, a vertex $s \in V$, a $d$-dimensional cost function $c : A \to \mathbf{Z}_+^d$
Result: An approximate Pareto-optimal frontier for all $v \in V$
Init: Define $\mathcal{L}_T = \{(0, s, -, -)\}$ and $\mathcal{L}_P = \emptyset$;
while $\mathcal{L}_T$ is not empty do
    Remove the lexicographically smallest label $w \in \mathcal{L}_T$;
    if no label $w' \in \mathcal{L}_P$ satisfies $pos(w') \leq pos(w)$ then
        Add $w$ to $\mathcal{L}_P$;
        for all outgoing arcs $a = (v(w), u)$ of $v(w)$ do
            Generate the label $w' = (c(w) + c(a), u, a, w)$ obtained by extending $w$ along the arc $a$;
            if $w'$ is not dominated by any label in $\mathcal{L}_P$ or $\mathcal{L}_T$ then
                Add $w'$ to $\mathcal{L}_T$;
            end
            if $w'$ dominates any labels $w''$ in $\mathcal{L}_T$ then
                Remove $w''$ from $\mathcal{L}_T$;
            end
        end
    end
end
```

The pseudo-code for our FPTAS is given in Algorithm 2. We follow the same procedure as the algorithm of Section 3.1, with one crucial difference. Whenever a label w is to be made permanent, we do not add it to $\mathcal{L}_P$ immediately. Instead, we check whether there is already a label $w' \in \mathcal{L}_P$, with the same end vertex, such that $pos(w') \leq pos(w)$ (note that $c_d(w') \leq c_d(w)$ due to the lexicographic ordering). If this is true, we do not add w to $\mathcal{L}_P$ but continue with the next label. In this case, we say that w is *approximately* dominated by w'. Note that the algorithm still includes an exact dominance check when a new label is created, i.e., we do not consider $pos(\cdot)$ in this check.

We now first prove correctness of the above algorithm. The proof is similar to the proof of Lemma 3.2 as given in Tsaggouris and Zaroliagis (2009). Then, we consider the running time of our FPTAS.

Lemma 3.4. *For each $1 \leq i \leq n-1$ and $w \in W^i(s, v)$ with $v \in V$, the algorithm finds a path w^* such that $c_j(w^*) \leq r^i c_j(w)$ for $j = 1, \ldots, d$.*

Proof. It suffices to prove that for all Pareto-optimal walks $w \in W^i(s, v)$, there is a label $w^* \in W(s, v)$ that is made permanent and satisfies $c_j(w^*) \leq r^i c_j(w)$ for $j = 1, \ldots, d$. We prove this claim by induction on i.

For $i = 1$, consider a walk $w \in W^1(s,v)$ which is Pareto-optimal for an arbitrary $v \in V$. By definition, any walk $w \in W^1(s,v)$ contains at most one arc and is therefore given by (s,v). It is clear that we add w to $\mathcal{L}_T$ when making the trivial path permanent. Since w is Pareto-optimal, it is never deleted from $\mathcal{L}_T$ (recall that this dominance check is exact). Therefore, at some point, this label is lexicographically smallest, and is thus selected to (possibly) be made permanent. If w is made permanent itself, take $w^* = w$. Otherwise, there is another label $w^* \in W(s,v)$ that approximately dominates w and is already made permanent. Given that w^* approximately dominates w, it holds that (i) $pos(w^*) \leq pos(w)$ and (ii) $c_d(w^*) \leq c_d(w)$. We now show that $c_j(w^*) \leq rc_j(w)$ for all $j = 1, \ldots, d-1$. Assume first that $c_j(w) = 0$. In that case, (i) implies that $c_j(w^*) = 0$ as well. Otherwise, $c_j(w) \neq 0$. If $c_j(w^*) = 0$, we are done. Otherwise, (i) implies that

$$\lfloor \log_r c_j(w^*) \rfloor \leq \lfloor \log_r c_j(w) \rfloor ,$$

which implies that

$$\log_r c_j(w^*) - 1 \leq \log_r c_j(w).$$

This shows that $c_j(w^*) \leq rc_j(w)$. Together with (ii) this implies that $c_j(w^*) \leq rc_j(w)$ for $j = 1, \ldots, d$.

For the induction step, we consider a walk $w \in W^i(s,v)$ for $1 < i \leq n-1$ which is Pareto-optimal. Removing the last arc of w, say (u,v), results in a new Pareto-optimal walk $q \in W^{i-1}(s,u)$. By the induction hypothesis, there is a label $q^* \in W(s,u)$ that has been made permanent, such that $c_j(q^*) \leq r^{i-1}c_j(q)$ for $j = 1, \ldots, d$. When q^* was made permanent, the label $w' = (c(w'), v, (u,v), q^*)$, corresponding to a walk $w' \in W(s,v)$, was generated. It holds for $j = 1, \ldots, d$ that

$$c_j(w') = c_j(q^*) + c_j(u,v) \leq r^{i-1}c_j(q) + c_j(u,v) \leq r^{i-1}(c_j(q) + c_j(u,v)) = r^{i-1}c_j(w). \quad (2)$$

Now, either w' is considered to be made permanent, or it is deleted from $\mathcal{L}_T$. The latter can only occur if we find a label $w_1 \in W(s,v)$ that dominates w'. On its turn, w_1 is either considered to be made permanent, or dominated by another label w_2. Repeating this argument, we ultimately end up with a label $w'' \in W(s,v)$ that is considered to be made permanent and dominates w':

$$c_j(w'') \leq c_j(w') \quad j = 1, \ldots, d. \quad (3)$$

Now, similarly as for $i = 1$, either w'' itself is made permanent, in case we set $w^* = w''$ and we are done, or there is another label w^*, such that (iii) $pos(w^*) \leq pos(w'')$ and (iv) $c_d(w^*) \leq c_d(w'')$. We then obtain

$$c_j(w^*) \leq rc_j(w'') \quad j = 1, \ldots, d. \quad (4)$$

Combining (2), (3) and (4) we obtain $c_j(w^*) \leq r^i c_j(w)$ for $j = 1, \ldots, d$. This completes the proof. □

We now consider the running time of our FTPAS. Recall that the lexicographic ordering of the cost vectors first considers the d^{th} dimension. This small fact is important, as the following lemma shows.

Lemma 3.5. *For every value $pos(\cdot)$ can take and for each $v \in V$, we add at most one label to $\mathcal{L}_P$.*

Proof. We prove this lemma by contradiction. Assume that there are two labels w' and w, ending in the same vertex $v = v(w) = v(w')$ and having $pos(w) = pos(w')$, that are both added to $\mathcal{L}_P$. Suppose that we first add w and then w'. Because w' is added, it must hold that $c_d(w') < c_d(w)$. This implies that $w' <_{lex} w$ (recall that lexicographic order was established starting at dimension d). This can only hold if w' was not yet contained in $\mathcal{L}_T$ when w was made permanent. Therefore, w' must have been added to $\mathcal{L}_T$ after w was made permanent. At that moment, all labels $w'' \in \mathcal{L}_T$ satisfied $w'' \geq_{lex} w$. Because the cost function is non-negative, the labels generated in all later iterations will satisfy this inequality as well. This contradicts the claim that $w' <_{lex} w$. □

Theorem 3.6. *The algorithm constructs an ε-approximate Pareto-optimal frontier for π in*

$$\mathcal{O}\left(n^3\left(\left(n\log(nC^{max})/\varepsilon\right)^{d-1}\right)^2\right)$$

time.

Proof. From Lemma 3.4 it is clear that for $r = (1+\varepsilon)^{1/(n-1)}$, the set of permanent labels obtained by Algorithm 2 is the desired ε-approximate Pareto frontier. Note that the total number of values the position function can take is proportional to $(\log_r(nC^{max}))^{d-1}$. Using the fact that $\log_r(x) = \log(x)/\log(r)$ and that $\log(r) \approx \varepsilon/(n-1)$ for small ε shows that this can be bounded by

$$\left(n\log(nC^{max})/\varepsilon\right)^{d-1}.$$

From Lemma 3.5 we know that the number of permanent labels returned by the algorithm for each node is at most the number of different values $pos(\cdot)$ can take. Hence, by multiplying the above number of values for $pos(\cdot)$ by the number of nodes n, we obtain an upper bound on the size of the set $\mathcal{L}_P$ after executing the algorithm:

$$|\mathcal{L}_P| \in \mathcal{O}\left(n\left(n\log(nC^{max})/\varepsilon\right)^{d-1}\right).$$

For every label made permanent we create at most n new labels. Hence, the total number of labels created during the execution of the algorithm is bounded by

$$|\mathcal{L}_P| + |\mathcal{L}_T| \in \mathcal{O}\left(n^2\left(n\log(nC^{max})/\varepsilon\right)^{d-1}\right). \tag{5}$$

For each of the created labels, we need to check dominance. As discussed in the proof of Lemma 3.1, there can be at most $|\mathcal{L}_P|$ temporary labels with the same end node. Therefore, checking dominance for one label requires at most

$$\mathcal{O}\left(n\left(n\log(nC^{max})/\varepsilon\right)^{d-1}\right) \tag{6}$$

comparisons. Note that the additional check when considering to make a label permanent can be seen as checking dominance for one additional label. Hence, the running time is bounded by the number of comparisons. This number is obtained by multiplying (5) and (6). This proves the claim. □

Although the asymptotic running time of the algorithm seems substantially higher than that of the TZ FPTAS, we emphasize that this only occurs in the worst possible scenarios with hardly any dominated paths. However, if there are many dominated labels and if $pos(p) \neq pos(p')$ for each pair p, p' of s, v-paths, for $v \in V$ (i.e., the position function is irrelevant), then the running time of the TZ FPTAS might be much higher than that of the exact algorithm, because the FPTAS does not check for dominance. On the contrary, the hybrid FPTAS is almost as fast as the exact algorithm of Section 3.1 on those instances.

3.3 Theoretical Comparison of the Algorithms

In this section we will analyze the running times of the three algorithms in more detail. First, we derive an explicit expression for the running time of the hybrid FPTAS based on the total number of Pareto-optimal paths for the bi-objective case. We also prove that there is a class of instances for which the hybrid FPTAS outperforms both the exact algorithm and the TZ FPTAS, with a factor that can become arbitrarily large (when the number of nodes increases).

3.3.1 Additional Running Time Analysis Hybrid FPTAS

In the bi-objective case, the running time of the hybrid FPTAS is bounded by a function of the number of Pareto-optimal paths. In order to prove this, we first derive an upper bound on the number of labels generated by the hybrid FPTAS.

Theorem 3.7. *Let $d = 2$ and K be the total number of Pareto-optimal paths as in Lemma 3.1. The approximate Pareto-optimal frontier generated by the hybrid FPTAS contains at most nK labels.*

Proof. It suffices to prove that for each Pareto-optimal path p that we do not make permanent, we make at most n paths permanent that were dominated by p. Recall that the algorithm is exact in the d^{th} dimension, i.e., for every label w we find a label w' such that $c_1(w') \leq (1+\varepsilon)c_1(w)$ and $c_2(w') \leq c_2(w)$.

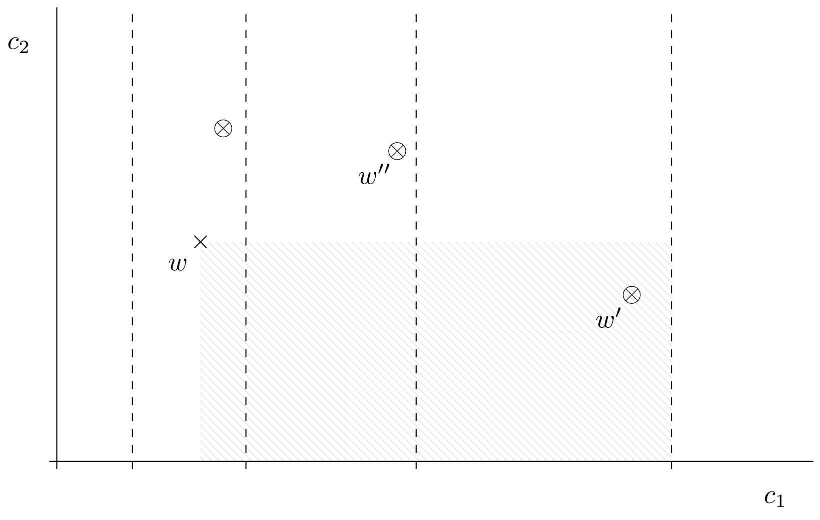

Figure 4: Example of the found paths when a Pareto-optimal path is not made permanent. For this instance we used $n = 3$.

First, it holds that $c_1(w') \leq (1+\varepsilon)c_1(w)$ only if $pos(w') - pos(w) \leq n - 1$. This is easily seen, since if $pos(w') - pos(w) \geq n$, then $c_1(w')/c_1(w) > r^{n-1}$, by definition of the position function, and thus $c_1(w') > (1+\varepsilon)c_1(w)$.

The above idea is illustrated in Figure 4. For illustrative purposes we consider a small instance with $n = 3$. The Pareto-optimal path w is marked by a $\times$, and the paths picked by the algorithm by a $\otimes$. The highlighted area represents all paths possibly picked by the algorithm that approximately dominate w. Note the 'cut-off' at $pos(w') > pos(w) + 3$. We denote the path found in this region by w' from hereon.

Let w'' be a label we make permanent, but which is dominated by w. Due to the lexicographic ordering and dominance check it is clear that if we make w'' permanent, it is not dominated by w'. This implies that

$$pos(w) \leq pos(w'') < pos(w'), \tag{7}$$

as w'' is dominated by w but not by w'. The strict inequality follows directly from Lemma 3.5, as we make at most one label permanent for each $pos(\cdot)$ value. Note that it is allowed to have $pos(w'') = pos(w)$ (if w'' is the leftmost dominated point in the figure), because w was not made permanent. Again applying Lemma 3.5 we conclude that, for each of the $n-1$ allowed $pos(\cdot)$ values following from (7), we make at most one dominated label permanent. Finally, note that it is not excluded that w' is dominated by w, as it might hold that $c_2(w') = c_2(w)$, thus we make at most n dominated labels permanent. This in turn implies we make at most nK labels permanent in total. □

Using Theorem 3.7, the next result is now immediate.

Corollary 3.8. *Let $d = 2$ and K be the total number of Pareto-optimal paths as in Lemma 3.1. The algorithm constructs an ε-approximate Pareto frontier for π in $\mathcal{O}\left(n\bar{K}^2\right)$ time, where $\bar{K} = \min\left\{n^2 \log(nC^{max})/\varepsilon, nK\right\}$.*

3.3.2 Relative Performance Hybrid FPTAS

Next, we show that the relative decrease in running time when using the hybrid FPTAS (compared to the exact algorithm and the TZ FPTAS) can become arbitrarily large when n increases.

We first introduce some notation. Let G_n be the graph on n nodes defined in Theorem 2.1. Let $G_q \circ G_n$ be the graph obtained by putting G_n after G_q and adding arcs (v_0, v_i), with $i = q, \dots, q+n-2$, with cost $c(v_0,v_i) = (0, 0)$. As an example, $G_3 \circ G_4$ is shown in Figure 5.

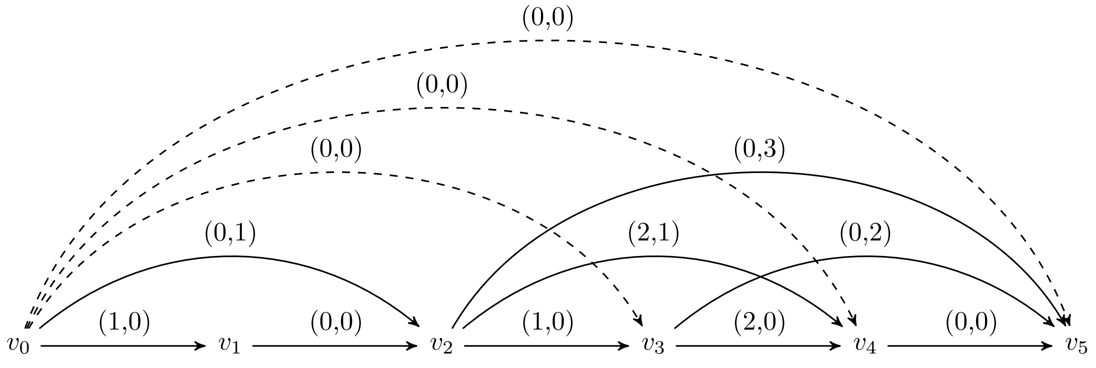

Figure 5: The graph $G_3 \circ G_4$.

Let $\theta(n)$ be the maximum ratio between the running time of the Exact Algorithm and the hybrid FPTAS on graphs on n nodes. Similarly let $\phi(n)$ be the maximum ratio between the TZ FPTAS and the hybrid FPTAS on graphs on n nodes. We will show that even in two dimensions these two ratios can become arbitrarily large in terms of n.

Theorem 3.9. *Consider $d = 2$ and let $k > 1$ be a fixed integer. For a given $\varepsilon > 0$, we have*

$$\theta(n) \in \Omega\left(\frac{\varepsilon^2 2^{n^{1/k}}}{n^{1+5/k}}\right),$$

and

$$\phi(n) \in \Omega\left(\varepsilon n^{1-5/k}\right).$$

Proof. The main idea of the proof is that we construct a class of graphs that have a large set of Pareto points (hence a high running time for the Exact Algorithm), but that also have a lot of 'redundant' paths (and hence the dominance check gives a great speed-up). Our asymptotic analysis shows that for a class of graphs with this structure, our hybrid FPTAS outperforms both algorithms.

The outline of the proof is as follows. We first give a lower bound for the running time of both the Exact Algorithm and the TZ FPTAS. We do this by giving a lower bound on the number of paths the algorithms find. We then give an upper bound on the running time of the hybrid FPTAS by considering the number of comparisons that are performed during execution of the algorithm. Let us now formalize this idea.

We consider the graph $G_q \circ G_{q^k-q}$ for some integer q. Note that this graph has $n = q + q^k - q - 1 = q^k - 1$ nodes (clearly, this is not restrictive, as our analysis is asymptotic). For this graph, there are 2^{i-1} Pareto-optimal v_0, v_i-paths, for $i = 1, \dots, q-1$. This follows directly from the proof of Theorem 2.1. Furthermore, there is exactly one Pareto-optimal v_0, v_i-path for $i = q, \dots, q^k - 2$, namely the path (v_0, v_i) with cost (0,0).

We first consider the TZ FPTAS. We know from the construction of G_q, combined with the proof of Theorem 2.1, that the costs of the v_0, v_i-paths, for $i = 1, \dots, q-1$, in G_q achieve all integer values ranging from 0 to $2^{i-1} - 1$ in both dimensions (recall our remark about symmetric costs at the end of Theorem 2.1). A similar result holds for G_{q^k-q}, i.e., the costs of the v_{q-1}, v_{q-1+i} paths, for $i = 1, \dots, q^k - q - 1$, achieve all integer values ranging from 0 to $2^{i-1} - 1$ in both dimensions. Furthermore, the path (v_0, v_{q-1}) has cost $(0, 2^{q-2} - 1)$ and can be extended with all paths in the graph G_{q^k-q}. As a consequence, for all $i = 1, \dots, q^k - q - 1$ and any integer value between 0 and $2^{i-1} - 1$, there exists a v_0, v_{q-1+i}-path for which the first objective is equal to that value.

Recall that the $pos(\cdot)$ function was defined, for a given walk w, as

$$pos_j(w) = \begin{cases} 0 & \text{if } c_j(w) = 0 \\ 1 + \lfloor \log_r c_j(w) \rfloor & \text{otherwise.} \end{cases}$$

Furthermore, we know that

$$\lfloor \log_r(2^{i-1}) \rfloor \approx \frac{(n-1)(i-1)}{\varepsilon}. \tag{8}$$

Hence, the v_0, v_i-paths are divided over approximately $\frac{(n-1)(i-1)}{\varepsilon}$ position values. This implies that the total number of paths found by the TZ FPTAS is at least

$$\sum_{i=1}^{q-1} \frac{(n-1)(i-1)}{\varepsilon} + \sum_{i=1}^{q^k-q-1} \frac{(n-1)(i-1)}{\varepsilon} = \sum_{i=1}^{q-1} \frac{(n-1)(i-1)}{\varepsilon} + \sum_{i=1}^{n-q} \frac{(n-1)(i-1)}{\varepsilon}.$$

Note that the second part of this sum grows fastest in terms of n: it follows that the running time of the TZ FPTAS is $\Omega(n^3/\varepsilon)$ for this instance.

Next, we consider the running time of the exact algorithm. The exact algorithm finds the 2^{i-1} Pareto-optimal v_0, v_i-paths, with $i = 1, \dots, q-1$. As each of the Pareto-optimal v_0, v_{q-1}-paths is extended to all nodes in G_{q^k-q} unequal to v_{q-1}, we know the algorithm creates at least $(q^k - q - 1)2^{q-2}$ temporary labels. It follows that the running time of the algorithm is $\Omega\left(q^k 2^q\right)$. Using that $n = q^k - 1$ we obtain the final expression $\Omega\left(n 2^{n^{1/k}}\right)$ for the running time of the exact algorithm on this instance.

Finally, we consider the running time of the hybrid FPTAS on this instance. The general idea is that we bound the number of comparisons that are perform by analyzing the number of temporary and permanent labels at each node. By doing that, we can give an upper bound on the number of comparisons that are necessary to perform the dominance checks.

First, we consider the nodes $v_0, \dots, v_{q-1}$. As argued above, the v_0, v_i-paths, for $i = 1, \dots, q-1$, are divided over approximately $\frac{(n-1)(i-1)}{\varepsilon}$ position values. Hence, we make at most $\frac{(n-1)(q-2)}{\varepsilon} \leq \frac{nq}{\varepsilon}$ labels permanent for each of these $q-1$ nodes.

Each of these permanent labels is extended along all arcs to the other nodes in G_q. It follows that for each node v_i, with $i = 0, \dots, q-1$, we have at most $\frac{nq^2}{\varepsilon}$ temporary labels at v_i. Therefore, by taking the dominance checks into account, we need at most $\frac{n^2q^4}{\varepsilon^2} + \frac{n^2q^3}{\varepsilon^2}$ comparisons per node (as every temporary label is compared to all temporary and permanent labels), and thus at most $\mathcal{O}\left(\frac{n^2q^5}{\varepsilon^2}\right)$ comparisons in total for the first q nodes.

For the nodes v_i, with $i = q, \ldots, q^k - 1$, the analysis is slightly easier. As mentioned, for each of these nodes there is exactly one Pareto-optimal path (namely the path (v_0, v_i) with zero cost). Due to the lexicographic ordering, this path is the first one to be considered to be made permanent at v_i. Since all permanent labels at v_{q-1} are extended to all nodes of G_{q^k-q}, we have, for each of the nodes $v_q, ..., v_{q^k-2}$, at most $\frac{nq}{\varepsilon}$ labels that are extended from v_{q-1} and 1 label extended from each other node in G_{q^k-q}. Checking dominance takes constant time, since all labels are dominated by the only Pareto-optimal label, which has been made permanent first, and thus it follows that the number of comparisons we need for the last $q^k - q - 1$ nodes is at most $(q^k - q - 1)\left(\frac{nq}{\varepsilon} + q^k - q - 1\right)$. Combining these two results, and substituting n for q^k, the total number of steps the hybrid FPTAS performs is at most

$$\frac{n^2q^4}{\varepsilon^2} + \frac{n^2q^5}{\varepsilon^2} + (n - q - 1)\left(\frac{nq}{\varepsilon} + n - q - 1\right),$$

which is $\mathcal{O}\left(\frac{n^2q^5}{\varepsilon^2}\right)$, or solely in terms of n, $\mathcal{O}\left(\frac{n^{2+5/k}}{\varepsilon^2}\right)$.

The bounds on $\theta(n)$ and $\phi(n)$ follow now directly from the three running time expressions. □

Note that k can be seen as a 'trade-off' parameter between $\theta(n)$ and $\phi(n)$, i.e., for fixed n we have that $\phi(n)$ increases when k increases, while $\theta(n)$ decreases and vice versa.

4 Computational Results

In this section, we test the performance of the exact algorithm and the FPTASes on various instances. First, we consider the class of worst case instances that we introduced in the proof of Theorem 2.1. Then, we perform an empirical analysis that is inspired by the theory on smoothed complexity. Finally, we consider practical instances, which have been used as a benchmark in several other papers on the bi-objective shortest path problem.

4.1 Worst Case Analysis

Worst case analysis is the most common way of analyzing the performance of algorithms. It is the form of analysis on which, for example, the famous $\mathsf{P} = \mathsf{NP}$ question is based. In this type of analysis, algorithms are evaluated on the *worst* possible input. Arora and Barak (2009) give a detailed treatise on the subject and related topics.

We evaluate the worst case performance based on the class of instances used to prove intractability in Theorem 2.1. We justified that these instances have the largest number of Pareto-optimal paths, for a given number of nodes in a directed graph. We solve instances with $n = 6, 7, \ldots, 20$ nodes. For the FPTASes, we use an approximation factor $\varepsilon = 0.05$. The running times are shown in Figure 6. To avoid fluctuations in running time, we report the average running times over 100 repetitions of the algorithm.

The vertical axis in Figure 6 has a logarithmic scale. We observe a trend in the running time of the exact algorithm that is at least linear. This is to be expected, due to the results from Theorem 2.1 and Lemma 3.1. The running times of the two FPTASes also increase, but have a much lower and decreasing slope. Furthermore, we see that the TZ FPTAS is faster for these instances than the hybrid FPTAS for larger values of n. This is to be expected as well: the dominance checks that are performed by the hybrid FPTAS are costly, since the set is large, but lead to no improvement, since all paths are Pareto-optimal. The TZ FPTAS omits these steps and is therefore able to solve the instances more efficiently.

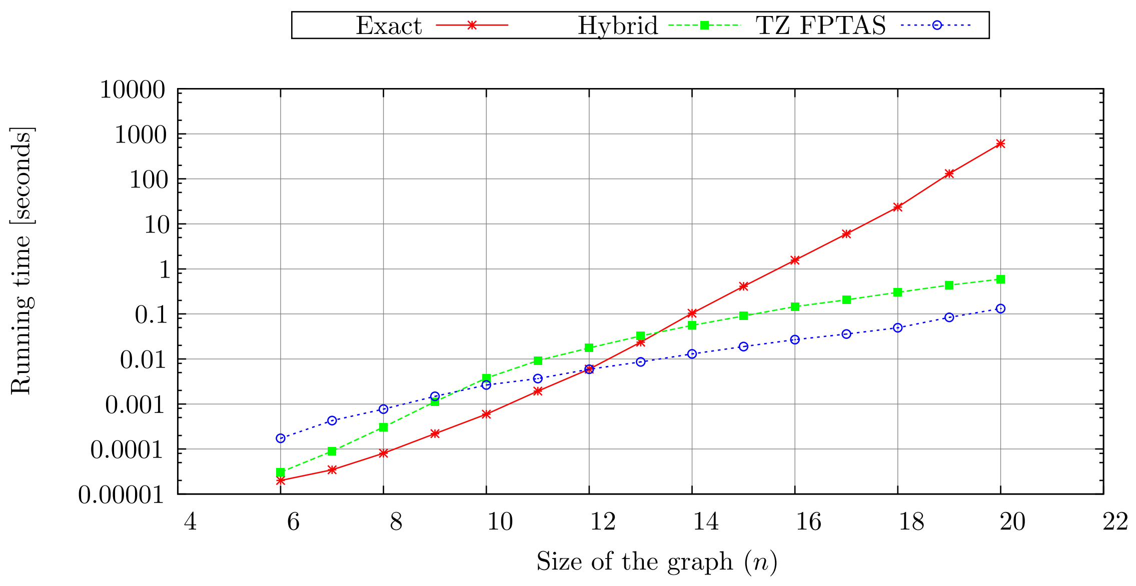

Figure 6: Worst case analysis

4.2 Smoothed Analysis

The second type of analysis that we consider is based on smoothed complexity. This type of analysis is relatively new, and is based on small 'perturbations' around worst case instances. Spielman and Teng (2009) give an in depth treatise on the subject. Interesting results are shown in Spielman and Teng (2004); they show that the smoothed complexity of the well-known simplex algorithm (see Papadimitriou and Steiglitz (1982)) is polynomial (under a certain pivot rule) in the input size and the inverse of the standard deviation of the perturbations. This provides new insights in the fact that the simplex algorithm is fast in practice, although the worst case running time is exponential.

In recent years, numerous authors have analyzed the smoothed complexity of bi-objective optimization problems. Beier et al. (2007) consider a very general class of bi-objective optimization problems, and derive upper bounds on the expected number of Pareto-optimal paths, assuming a certain distribution for the perturbations. To the best of our knowledge, all upper bounds depend on the product $\mu\phi$, where μ is the maximum expected cost, and ϕ the supremum of the probability density function of the random perturbations. For example, in case of a Gaussian distribution with standard deviation σ, we have $\phi \sim 1/\sigma$. This implies that the upper bound is exponential when the average costs are exponential and the standard deviation is relatively small (this holds, for instance, if σ is not exponential in the number of nodes). We note that intuitively this is clear: if $\sigma \ll \mu$ then the perturbations have little effect.

We consider two different frameworks for analyzing the algorithms. In the first framework, we perturb the arc costs of the graph. Here, we analyze the case where σ is relatively small empirically. In the second framework, we also consider perturbations of the topology of the graph.

4.2.1 Cost perturbations

The first approach is based on random perturbations of the arc costs. We consider the worst case scenario used in Theorem 2.1 and analyze what happens if we add independent, normally distributed perturbations to the arc costs (rounding to the nearest integer to keep costs integer). By using different standard deviations in the normal distribution, we vary the sizes of the perturbations that are applied to the instance.

The results of this method are shown in Figure 7, for the instance with $n = 17$ nodes and $\varepsilon = 0.05$.

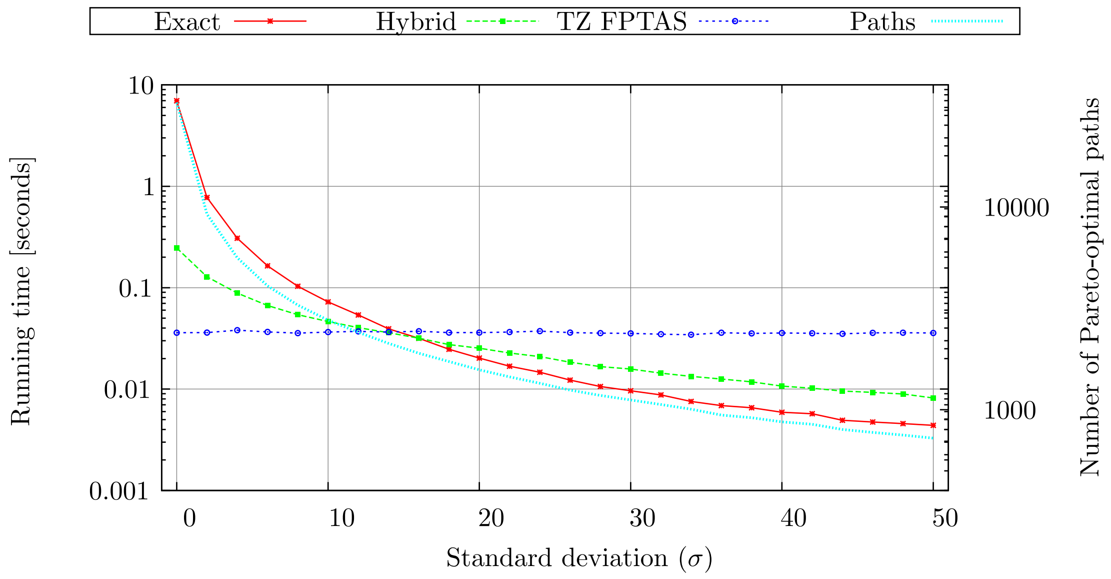

Figure 7: Smoothed analysis based on perturbations of the cost structure

For each value of $\sigma \in \{0, 2, \dots, 50\}$, we have solved 1000 instances. In the figure, we show the average running times for all three solution methods. Furthermore, we report the number of Pareto-optimal paths, also using a logarithmic scale.

As can be observed in Figure 7, even small perturbations of the arc costs decrease the running times of the exact algorithm and hybrid FTPAS drastically. This is a consequence of the decreasing number of Pareto-optimal paths. In fact, for $\sigma \approx 14$, the running times of the FPTASes are similar to that of the exact method. Note that the running time of the TZ FPTAS is independent of the standard deviation. This is to be expected, as this FPTAS does not include dominance checks and, as a consequence, does not profit from smaller Pareto-optimal frontiers. We note that all values of σ are small compared to the arc costs in the graph: the largest cost is $2^{15} \approx 3.3 \cdot 10^4$.

4.2.2 Perturbing the structure of the graph

The second framework we consider is somewhat more involved. Because a MOSP instance is determined both by the structure of the graph and the cost vectors, we also consider changes in terms of the graph structure. The idea of the method is that we simulate the *transition* of a random instance to a worst case instance. We refer to this approach as a transition analysis. We now first describe how to simulate this transition.

Consider the worst case instance $G = (V, A)$ where $V = \{1, \dots, n\}$. We consider a random graph $G' = (V, A')$, with the same set of vertices, but with different arcs and costs. The random graph is determined as follows. First we define a density $\rho \in (0, 1]$. For every ordered pair of vertices $a = (i, j)$ with $1 \leq i < j \leq n$, we generate a Bernoulli random variable with success probability ρ. If the realization is 1, the arc (i, j) is present in A'. If the realization is 0, the arc is not included in A'. If the arc is present, both arc costs are uniformly distributed random variables between 0 and 2^n. Note that we obtain a complete directed graph on V if $\rho = 1$. Furthermore, the expected density of the graph is equal to ρ.

In order to simulate the transition from the worst case instance G to the random graph G', we define a family of graphs G_θ. Here, θ is a parameter between 0 and 1. For a given value of θ, the graph $G_\theta = (V, A_\theta)$ is defined as follows. For each arc $(v_1, v_2) \in A$, we generate a Bernoulli random variable with success probability θ. If the realization is 0, we add the arc $a \in A$ to A_θ. Otherwise, if the realization is 1, we add the arc $a \in A'$ to A_θ if this arc is present in G'. By

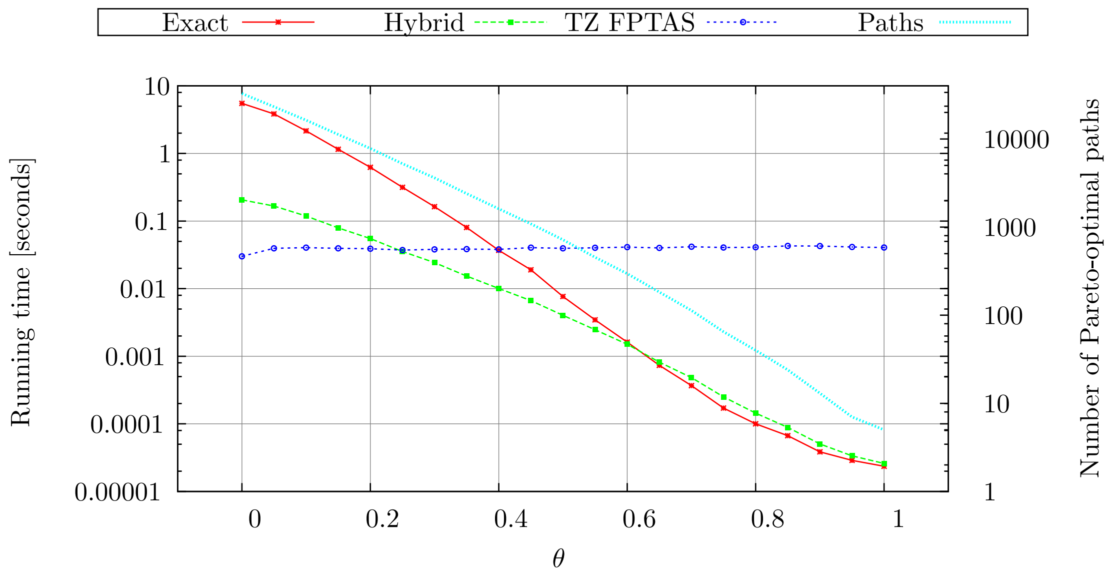

Figure 8: Smoothed analysis based on perturbations of the graph structure

definition, it holds that G_0 is equal to the worst case instance. Similarly, for $\theta = 1$, G_1 is equal to a random instance.

In Figure 8, the results are depicted for $n = 17$ and $\rho = 0.8$. The FPTASes use an approximation factor $\varepsilon = 0.05$. For each value of $\theta \in \{0, 0.05, 0.1, \ldots, 1\}$, we have generated 1000 graphs G_θ. In the figure, we plotted the average computation time over these 1000 instances for all three solution methods. We also report the number of Pareto-optimal paths in the graph. We observe that the running times of the exact method and of the hybrid FPTAS decrease rapidly if θ increases. This is a consequence of the decreasing number of Pareto-optimal paths in the graph. On the contrary, the running time of the TZ FPTAS is roughly constant. As the TZ FPTAS does not use any dominance checks, it cannot exploit the decreasing number of Pareto-optimal paths. Remarkably, for this particular instance, the exact algorithm is orders of magnitude faster than the TZ FPTAS for $\theta \geq 0.6$. The running time of the hybrid FPTAS is comparable to that of the exact algorithm for these instances.

4.3 Practical instances

In order to assess the performance of the algorithms on real-world instances, we apply them to a set of instances from the literature. These instances have been introduced as benchmarks for bi-objective shortest path algorithms in Raith and Ehrgott (2009), and have subsequently been used in many other computational evaluations of bi-objective shortest path algorithms.

A problem when applying the FTPASes in practice is that they cannot cope with large node sets. Recall that $r = (1+\varepsilon)^{\frac{1}{n-1}}$, where n is the number of nodes in the graph. For values of n that are typically observed in practice (e.g., $n \approx 10{,}000$), r is very close to 1. This means that $pos(w) \neq pos(w')$ for all pairs of labels w and w'. It follows that no labels are approximately dominated. In this situation, both FPTASes are slower than the exact algorithm.

To circumvent the situation described above, we do not use the value for r that is proposed in Theorem 3.6. Instead, we select a value of r that is much larger. We should stress that by doing so, we cannot guarantee that a (reasonable) approximation factor is obtained. However, by comparing the approximate Pareto-optimal frontier to the exact one, we can compute the approximation factor *ex post*.

We have tested our algorithms on three classes of instances. We consider 33 grid networks and 15

	Exact		Hybrid (1.001)		Hybrid (1.01)		Hybrid (1.1)		TZ FPTAS (1.1)	
Instance	Time	Paths	Time	Factor	Time	Factor	Time	Factor	Time	Factor
Grid	71.5	220	66.9	1.002	18.0	1.04	2.32	1.30	2,116	1.22
small	0.39	56	0.43	1	0.33	1.02	0.12	1.23	36.1	1.17
medium	22.7	260	9.98	1.003	1.94	1.05	0.38	1.27	370	1.20
large	209	179	222	1.000	61.5	1.03	7.57	1.39	6,881	1.27
Netmaker	33.4	5	36.7	1	12.9	1.001	2.0	1.008	906	1.008
Road, DC	9.1	76	7.50	1.002	2.88	1.035	0.61	1.19	1,097	1.19
Road, RI	811	92	257	1.001	82.4	1.016	15	1.15	44,942	1.149

Table 1: The running times of the exact algorithm and FPTASes for practical instances and the ex post approximation factors of the FPTASes. For the FPTASes, we report the value of r between brackets.

NetMaker instances that are generated by Raith and Ehrgott (2009). Firstly, each grid instance contains a grid of nodes, in which each node is connected to its left, right, upper, and lower neighbor whenever these are present. Furthermore, there is a source node that is connected to all nodes in the first column and a sink node that is connected to all nodes in the last column in the grid. The number of nodes in these graphs varies between 1,202 and 40,002; the number of arcs varies between 4,720 and 159,600. We have subdivided the grid instances into three classes. We consider 3 small instances with up to 3,000 nodes; 9 large instances with over 10,000 nodes and 21 medium instances with about 5,000 nodes. Secondly, we consider 15 NetMaker instances. These instances contain 3,000 nodes and between 30,000 and 80,000 arcs. These artificially generated instances should represent networks that are more difficult to solve. Finally, we consider two road networks. The road networks have been researched extensively in the 9th DIMACS implementation challenge. We consider the road networks of Washington, DC and of Rhode Island. These instances contain 9,559 and 53,658 nodes, respectively.

In Table 1, we report the results of our analysis. For the grid and NetMaker instances, we report the average running time and the average approximation factor. The averages are computed over all instances in a class. We also give the average results for the subclasses of small, medium, and large grid instances. For the road networks, we report the results for each instance individually.

We first compare the exact algorithm and the hybrid FPTAS. We observe that good approximations are obtained for $r = 1.001$. The ex post approximation ratios are all at most 1.003. We also observe that the running times of the hybrid FPTAS with $r = 1.001$ and the exact algorithm are comparable. Only for the largest instance, the road network of RI, the running time is reduced considerably by using the hybrid algorithm. If we use $r = 1.01$, the approximation ratios are still small. All approximation ratios observed in the table are at most 1.05. Especially for larger instances, the running times are reduced considerably. Finally, the lowest running times are obtained for $r = 1.1$. However, we generally obtain large approximation factors for this setting. For the large grid instances, the average approximation factor is 1.39. Note that all approximation factors observed are orders of magnitude smaller than the ones that can theoretically be guaranteed.

Table 1 also reports the results for the TZ FPTAS. For practical instances, the running times of TZ FPTAS are very large already for $r = 1.1$. This finding confirms the theoretical running time in Lemma 3.3. For this value of r, the approximation ratios are large. This indicates that the TZ FPTAS does not perform well on practical instances, which can be explained by the fact that many paths in the graph are dominated: in contrast to the hybrid FTPAS, the TZ FPTAS does not exploit this dominance.

Among the medium grid instances, there are 19 instances that all have approximately 4,900 nodes. These instances vary in the shape of the grid. The grids range from very wide and low, to very narrow and high. In the narrow instances, the paths are very short, whereas the wide instances allow for very long paths. The running time of the exact algorithm increases with the number of columns in the graph: wider instances require much more computation time than narrow

Grid instance	Exact		Hybrid (1.001)		Hybrid (1.01)		Hybrid (1.1)	
(id: rows × columns)	Time	Paths	Time	Factor	Time	Factor	Time	Factor
15: 2450 × 2	0.21	6	0.26	1	0.23	1	0.22	1
16: 1225 × 4	0.25	6	0.26	1	0.26	1	0.30	1
17: 612 × 8	0.35	10	0.34	1	0.37	1	0.32	1.10
18: 288 × 17	0.66	15	0.62	1	0.57	1	0.45	1.10
19: 196 × 25	1.24	18	1.10	1	1.14	1.01	0.45	1.11
20: 140 × 35	1.66	32	1.67	1	1.56	1.01	0.49	1.18
21: 111 × 44	2.16	54	2.38	1	1.74	1.01	0.60	1.15
22: 92 × 53	2.50	53	2.77	1	2.13	1.01	0.54	1.27
23: 79 × 62	3.35	77	3.52	1	2.20	1.02	0.47	1.18
24: 70 × 70	5.27	93	5.54	1	3.06	1.02	0.56	1.23
25: 62 × 79	6.01	95	6.02	1	2.72	1.02	0.48	1.39
26: 53 × 92	5.81	93	6.46	1	3.01	1.02	0.45	1.29
27: 44 × 111	7.59	137	8.86	1	3.07	1.04	0.47	1.31
28: 35 × 140	11.6	209	13.92	1	3.65	1.02	0.41	1.35
29: 25 × 196	15.7	244	19.12	1.00	3.34	1.05	0.39	1.52
30: 17 × 288	29.3	371	30.71	1.00	3.28	1.09	0.34	1.68
31: 8 × 612	72.4	819	46.57	1.01	2.67	1.21	0.17	1.62
32: 4 × 1225	143	1383	37.59	1.02	1.47	1.27	0.09	1.48
33: 2 × 2450	161	1594	15.18	1.04	0.57	1.20	0.07	1.24

Table 2: The running times of the exact algorithm and the hybrid FPTAS for some of the grid instances and the ex post approximation factors. For the FPTAS, we report the value of r between brackets.

instances. In Table 2, we report the running times and approximation factors for these instances. It is interesting to note that the instances which are most difficult to solve exactly, turn out to be relatively easy to approximate. For example, the exact algorithm needs 161 seconds to solve the instance with 2 rows and 2,450 columns of nodes. Using $r = 1.001$, this instance can be solved by our hybrid FPTAS within 15 seconds. The ex post approximation factor for this instance is 1.04. This shows that the hybrid FPTAS might work well for those practical instances that are most difficult to solve to optimality.

5 Conclusion and Further Research

In this paper we have proposed a new FPTAS for the multi-objective shortest path problem. The algorithm, referred to as the hybrid FPTAS, uses elements from both an exact labeling algorithm and an FPTAS proposed by Tsaggouris and Zaroliagis (2009), referred to as the TZ FPTAS. We have analyzed the running times of the algorithms both from a theoretical and a computational point of view.

First of all, we show that, in the worst case, the number of Pareto-optimal solutions is exponential (with base 2) in the number of nodes of the graph. Furthermore, we show that the running time of the exact labeling algorithm is quadratic in the number of Pareto-optimal solutions. These results suggest that finding the whole Pareto-optimal frontier cannot be performed in a reasonable amount of time in case the number of Pareto-optimal solutions is relatively high. Therefore, FPTASes are a good alternative to approximate the Pareto-optimal frontier.

The running time of the TZ FPTAS mainly depends on the size of the arc costs and it is almost insensitive to the number of Pareto-optimal solutions. This means that the running time can be much higher compared to an exact approach in case the Pareto-optimal frontier is small. This drawback is not present in our hybrid FPTAS, as it exploits the dominance of solutions, which reduces the running time in case of small Pareto-optimal frontiers. In fact, in case of two objectives, we prove that the number of approximate solutions is at most the number of Pareto-optimal solutions multiplied by the number of nodes in the graph. Moreover, we prove that there exist instances for which our hybrid FPTAS is an arbitrary times faster than both the exact labeling algorithm and the TZ FPTAS.

Besides these theoretical properties, we have tested the practical running times of the algorithms under different settings. In the first set of tests, we tested the algorithms on the earlier mentioned worst case instances. As expected, the running time of the exact algorithm explodes if the number of nodes is sufficiently large. Our hybrid FPTAS performs slightly worse than the TZ FPTAS due to unnecessary dominance checks, but much better than the exact algorithm.

Next, we tested the algorithms on two sets of instances inspired on the theory of smoothed complexity analysis. In the first class only the arc costs were perturbed with a random amount, whereas in the second class also the topology of the graph is changed with a certain probability. In both sets of instances we see a similar behavior: As an instance deviates more from the worst case instance, the number of Pareto-optimal solutions gets smaller rapidly. As a result, the TZ FPTAS is the fastest in case of small perturbations, our hybrid algorithm outperforms the other algorithms if the perturbations get larger, while the exact algorithm has the lowest computation time in case the perturbations are large enough (due to a small Pareto-optimal frontier).

The last set of tests contains large practical instances that have been used in other papers. Unfortunately, the hybrid FPTAS has a high running time if directly used due to (i) the small required approximation factor between iterations, (ii) the large number of nodes in the instances, and (iii) the high degrees of the polynomials in the running time. However, it turns out that even when setting the desired precision low, the true precision (observed *ex post*) is much higher and seems acceptable for practical use. If our hybrid algorithm is applied in this way, it is competing with and often faster than the exact algorithm, and much faster than the TZ FPTAS.

When considering all tests performed, the algorithms seem to complement each other: when one expects a high number of Pareto-optimal paths with hardly any dominated paths, then the TZ FPTAS seems to be the best choice. If the number of dominated paths gets larger and the Pareto-optimal frontier gets smaller, the hybrid FPTAS is recommended. In turn, the exact algorithm performs best, if only a few Pareto-optimal paths exist. Finally, since our hybrid FPTAS never performs much worse than the best algorithm, we can conclude that it is the most robust algorithm among the three in terms of running time (of course, if one is satisfied with an approximate Pareto-optimal frontier).

Since the running times strongly depend on the number of Pareto-optimal solutions, a direction for further research is to determine bounds on the number of Pareto-optimal solutions, or even stronger, to derive a (functional) relation between the number of Pareto-optimal solutions and the standard deviation of the arc costs (as in smoothed complexity analysis). If a good estimate of the number of Pareto-optimal solutions is available, one can make a well-informed decision on which algorithm to use. Another obvious direction for future research is to derive properties to reduce the time complexity of the hybrid FPTAS.

Acknowledgments

The authors thank Dr. Andrea Raith from the University of Auckland for providing us with the grid instances and the NetMaker instances.

References

Ahuja, R. K., K. Mehlhorn, J. Orlin, and R. E. Tarjan (1990). Faster algorithms for the shortest path problem. *Journal of the ACM 37*(2), 213–223.

Arora, S. and B. Barak (2009). *Computational Complexity: A Modern Approach* (1st ed.). New York, NY, USA: Cambridge University Press.

Beier, R., H. Röglin, and B. Vöcking (2007). The smoothed number of Pareto optimal solutions in bicriteria integer optimization. In M. Fischetti and D. P. Williamson (Eds.), *Integer Programming and Combinatorial Optimization: 12th International IPCO Conference, Ithaca, NY, USA, June 25-27, 2007. Proceedings*, Berlin, Heidelberg, pp. 53–67. Springer Berlin Heidelberg.

Christofides, N. (1975). Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA.

Deo, N. and C.-Y. Pang (1984). Shortest-path algorithms: Taxonomy and annotation. *Networks 14*(2), 275–323.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik 1*(1), 269–271.

Ehrgott, M. (2000). *Multicriteria optimization.* Lecture Notes in Economics and Mathematical Systems. Springer-Verlag.

Hansen, P. (1980). Bicriterion path problems. In G. Fandel and T. Gal (Eds.), *Multiple Criteria Decision Making Theory and Application*, Volume 177 of *Lecture Notes in Economics and Mathematical Systems*, pp. 109–127. Springer Berlin Heidelberg.

Kern, W. and G. J. Woeginger (2007). Quadratic programming and combinatorial minimum weight product problems. *Mathematical Programming 110*(3), 641–649.

Marler, R. T. and J. S. Arora (2004). Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization 26*(6), 369–395.

Mittal, S. and A. S. Schulz (2008). A General Framework for Designing Approximation Schemes for Combinatorial Optimization Problems with Many Objectives Combined into One. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, Volume 5171 of *Lecture Notes in Computer Science*, pp. 179–192. Springer Berlin Heidelberg.

Mote, J., I. Murthy, and D. L. Olson (1991). A parametric approach to solving bicriterion shortest path problems. *European Journal of Operational Research 53*(1), 81–92.

Müller-Hannemann, M. and K. Weihe (2006). On the cardinality of the Pareto set in bicriteria shortest path problems. *Annals of Operations Research 147*(1), 269–286.

Mulmuley, K. and P. Shah (2001). A lower bound for the shortest path problem. *Journal of Computer and System Sciences 63*, 253–267.

Paixão, J. M. and J. L. Santos (2013). Labeling methods for the general case of the multi-objective shortest path problem – a computational study. In A. Madureira, C. Reis, and V. Marques (Eds.), *Computational Intelligence and Decision Making: Trends and Applications*, pp. 489–502. Dordrecht: Springer Netherlands.

Papadimitriou, C. H. and K. Steiglitz (1982). *Combinatorial Optimization: Algorithms and Complexity.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Papadimitriou, C. H. and M. Yannakakis (2000). On the approximability of trade-offs and optimal access of web sources (Extended Abstract). In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pp. 86–92.

Raith, A. and M. Ehrgott (2009). A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research 36*(4), 1299 – 1331.

Sedeño-Noda, A. and A. Raith (2015). A Dijkstra-like method computing all extreme supported non-dominated solutions of the biobjective shortest path problem. *Computers & Operations Research 57*, 83–94.

Serafini, P. (1987). Some considerations about computational complexity for multi objective combinatorial problems. In J. Jahn and W. Krabs (Eds.), *Recent Advances and Historical De-*

velopment of Vector Optimization, Volume 294 of *Lecture Notes in Economics and Mathematical Systems*, pp. 222–232. Springer Berlin Heidelberg.

Skriver, A. J. V. (2000). A classification of bicriterion shortest path (BSP) algorithms. *Asia Pacific Journal of Operational Research 17*(2), 199–212.

Spielman, D. A. and S.-H. Teng (2004). Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM 51*(3), 385–463.

Spielman, D. A. and S.-H. Teng (2009). Smoothed analysis: An attempt to explain the behavior of algorithms in practice. *Communications of the ACM 52*(10), 76–84.

Tarapata, Z. (2007). Selected Multicriteria Shortest Path Problems: An Analysis of Complexity, Models and Adaptation of Standard Algorithms. *International Journal of Applied Mathematical and Computer Science 17*(2), 269–287.

Tsaggouris, G. and C. Zaroliagis (2009). Multiobjective optimization: Improved FPTAS for shortest paths and non-linear objectives with applications. *Theory of Computing Systems 45*(1), 162–186.

Ulungu, E. L. and J. Teghem (1994). Multi-objective combinatorial optimization problems: A survey. *Journal of Multi-Criteria Decision Analysis 3*(2), 83–104.

Woeginger, G. J. (1999). When Does a Dynamic Programming Formulation Guarantee the Existence of an FPTAS? In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '99, Philadelphia, PA, USA, pp. 820–829. Society for Industrial and Applied Mathematics.