Bachelor Thesis

A Parametric View on Robust Graph Problems

Christoph Hertrich

July 29, 2016

Supervised by
Prof. Dr. Sven O. Krumke

Department of Mathematics
Technische Universität Kaiserslautern

Contents

1. Introduction

1.1. The General Parametric Problem

Given a (directed or undirected) graph $G = (V, E)$, a nonnegative cost function $c\colon E \to \mathbb{R}_{\geq 0}$ and a set X of feasible subsets of E, a typical problem is to find a feasible subset $S \subseteq E$ of edges with minimum cost $c(S) = \sum_{e \in S} c(e)$. This formulation covers, for instance, the shortest path problem by choosing X as the set of all paths between two vertices, or the minimum spanning tree problem by choosing X to be the set of all spanning trees. For formal definitions refer to the appendix.

In this bachelor thesis, the cost functions of the edges additionally depend on a parameter $\lambda \in \mathbb{R}_{\geq 0}$. We replace our cost function c by c_λ and consider the problem

$$(P(\lambda)) \qquad \min\{c_\lambda(S) \mid S \in X\}.$$

Given nominal costs $c\colon E \to \mathbb{R}_{\geq 0}$, growth rates $d\colon E \to \mathbb{R}_{\geq 0}$ and upper bounds $u\colon E \to \mathbb{R}_{\geq 0} \cup \{+\infty\}$ we consider the following kind of bounded affine growth:

$$c_\lambda(e) = \min\{c(e) + \lambda d(e), u(e)\}.$$

Throughout this thesis we assume without loss of generality that $u(e) \geq c(e)$. Otherwise just set $c(e)$ to $u(e)$ without changing anything (the value of $d(e)$ does not matter).

Our formulation includes some special cases we sometimes investigate separately:

1. Proportional growth: $c_\lambda(e) = c(e)(1 + \lambda)$
2. Constant growth: $c_\lambda(e) = c(e) + \lambda$
3. Affine growth: $c_\lambda(e) = c(e) + \lambda d(e)$

and the bounded variants of those three growth types.

1.2. Our Approach

In contrast to standard nonparametric graph problems, for our parametric problem $(P(\lambda))$ it is a priori not clear, what a solution should look like. If we still want to have a single solution $S \in X$, we have to accept that it might not be possible to find one which is optimal for all $\lambda \in \mathbb{R}_{\geq 0}$. Instead of that, we could, for instance, be interested in finding a most robust solution, in the sense that for every $\lambda \in \mathbb{R}_{\geq 0}$ our robust solution is not too far from the optimal solution. However, it turns out that this might not be a really good idea, if λ is allowed to take all values in $\mathbb{R}_{\geq 0}$: A solution which is still robust for very large λ may be very bad for small values of λ and vice versa.

Another possibility is to abandon the requirement of finding a single solution $S \in X$. In order to have an optimal solution for all $\lambda \in \mathbb{R}_{\geq 0}$, we could allow a solution to consist of a set $R \subseteq X$ with the constraint that for all $\lambda \in \mathbb{R}_{\geq 0}$ there is an optimal $S \in R$. This leads to the definition of a representation system:

Definition 1.1. For $(P(\lambda))$ let $R \subseteq X$ be a set of feasible solutions such that for every $\lambda \in \mathbb{R}_{\geq 0}$ there is an optimal solution $S(\lambda) \in R$. Then we call R a **representation system**.

Such a system can be used in practice to solve the problem $(P(\lambda))$ efficiently: If λ is unknown one can already compute a representation system in a preprocessing step. After obtaining the concrete value of λ, only the solutions in R have to be reconsidered.

Obviously, X itself is a representation system. However, this is not really useful, since X might be very large (e.g. not polynomial in the size of the graph). For practical purposes we are looking for a representation system which is as small as possible. However, there are not always representation systems which are small enough for being efficient in practice. Hence, this approach is not always useful, even if it really is in some cases.

This motivates us to think about taking a mixture of both strategies: We could look for a set $R \subseteq X$ which, on one hand, is reasonably small, and, on the other hand, satisfies that $\min_{S \in R} c_\lambda(S)$ and $\min_{S \in X} c_\lambda(S)$ are not too far from each other for all $\lambda \in \mathbb{R}_{\geq 0}$.

1.3. A Two-Phase Approach

Our approach to solve the parametric problem by finding a representation system is a special variant of a two phase algorithm as introduced in [CFLY10]: In the first phase λ is unknown, but we already know the structure of the parametric problem. The goal in the first phase is to prepare for the second phase, where we know a concrete value of λ and want to find the specific solution for this λ. Obviously this only makes sense if the computational time in phase two is lower than the time needed for solving the nonparametric problem.

In [CFLY10] this division into two phases is analyzed in greater detail for the parametric shortest path problem. For example, if all cost functions are affine and not necessarily positive, they introduced a method, how an appropriate preprocessing in phase one can reduce the time complexity of phase two to $\mathcal{O}(m + n \log n)$, which is faster than any known shortest path algorithm with negative costs.

In our case we only look at positive costs. Then the mentioned running time can already be achieved by Dijkstra's algorithm using a Fibonacci heap, see e.g. [KN05].

1.4. Outline

The remainder of this thesis is divided into three chapters:

In Chapter 2 we find upper bounds for the size of a minimum representation system under different assumptions. This leads to a pseudo-polynomial bound in the case of general bounded affine growth. An example of Carstensen in her PhD thesis [Car83] shows that this can not be improved to a proper polynomial bound. However, in the remainder of the chapter we prove strongly polynomial bounds for several special cases and, in particular, for the minimum spanning tree problem.

While the results of Chapter 2 are not always constructive, Chapter 3 fills this gap by providing some algorithms for finding representation systems for the shortest path problem under different assumptions. Here we adapt and extend an existing algorithm introduced in [KO81] and improved in [YTO91].

Finally, Chapter 4 relaxes our definition of a representation system. We introduce the notion of an approximation system which only requires for every λ to have a solution which is optimal up to a constant factor. We prove that leaving the requirement of exact optimality leads to polynomial systems, as long as the approximation factor remains constant. We also give an algorithm how to find such an approximation system.

In the appendix one can find basic definitions, notations and conventions used in this thesis.

2. The Size of Representation Systems

The goal in this chapter is to investigate how large a representation system has to be under different assumptions. We summarize our general setting for later reference:

Assumption 2.1. *$G = (V, E)$ is a (directed or undirected) graph, $\emptyset \neq X \subseteq 2^E$ is a set of feasible subsets of E, $c_\lambda \colon E \to \mathbb{R}_{\geq 0}$ is a cost function dependent on a parameter $\lambda \in \mathbb{R}_{\geq 0}$. We write $c(e) = c_0(e)$ and consider the problem*

$$(P(\lambda)) \qquad \min \left\{ c_\lambda(S) = \sum_{e \in S} c_\lambda(e) \;\middle|\; S \in X \right\}.$$

Moreover, $K = \{|S| \mid S \in X\}$ is the set of different sizes of feasible solutions, $K_{\max} = \max K$ the maximum of those sizes and $k = |K|$ the number of different sizes.

2.1. Existence of Pseudo-Polynomial Sized Systems

In this section we show that there is always a representation system of pseudo-polynomial size, if for every edge $e \in E$ the parametric cost function $\lambda \mapsto c_\lambda(e)$ is piecewise linear and concave.

Definition 2.2. We call a function $f \colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ **piecewise linear**, if there exist finitely many $0 = x_0 < x_1 < \cdots < x_l = \infty$ such that f is linear on $[x_{i-1}, x_i] \setminus \{\infty\}$ for $i = 1, \ldots, l$.

Observe that our definition of piecewise linearity implies continuity: For points in the interior of a linear piece this follows by the linearity on this piece. For a breakpoint this is ensured by the fact that we claim the linearity on the closed intervals: This gives us even local Lipschitz continuity around a breakpoint by choosing the Lipschitz constant as the maximum of the absolute values of the slopes of the previous and the next linear piece.

We also need a lemma about concave functions:

Lemma 2.3. *Let $f_i \colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, $i = 1, \ldots, l$ be l concave functions. Then $f(x) = \min_{i=1,\ldots,l} f_i(x)$ is also concave.*

Proof. For all $\mu \in [0,1]$, all $x, y \in \mathbb{R}_{\geq 0}$ and all $i = 1, \ldots, l$ we have by concavity of f_i:

$$f_i(\mu x + (1-\mu)y) \geq \mu f_i(x) + (1-\mu) f_i(y) \geq \mu f(x) + (1-\mu) f(y)$$

The claim follows by taking the i for which the minimum is attained at $\mu x + (1-\mu)y$ on the left hand side. □

Theorem 2.4. *Suppose Assumption 2.1 holds and for all edges $e \in E$ the function $\lambda \mapsto c_\lambda(e)$ is concave, piecewise linear and all the slopes of the linear pieces are integer values in $\{0, 1, \ldots, d_{\max}\}$. Then there is a representation system of cardinality at most $K_{\max} d_{\max} + 1$.*

Proof. For every $S \in X$, also $f_S(\lambda) = c_\lambda(S)$ is piecewise linear and concave in λ as a sum of at most $K_{\max}$ such functions. Moreover, the slope of each linear piece is an integer value between 0 and $K_{\max} d_{\max}$. This lets us conclude that also $f(\lambda) = \min_{S \in X} f_S(\lambda)$ is piecewise linear and concave as a minimum of finitely many such functions using Lemma 2.3. Moreover, the minimum preserves the property that all occurring slopes are integer values between 0 and $K_{\max} d_{\max}$. The concavity guarantees that the slope of every piece is less than the slope of the piece before. Hence, there are at most $K_{\max} d_{\max} + 1$ linear pieces.

We show that for every linear piece there is a solution S such that $c_\lambda(S)$ is minimum on this piece. This implies the claim, since then the solutions corresponding to the linear pieces form a representation system of cardinality at most $K_{\max} d_{\max} + 1$.

Let $[a, b]$ be a maximal interval on which f is linear. Let X_a be the set of all solutions $S \in X$ such that S is optimal for $\lambda = a$. For each $S \in X_a$ let $[a_S, b_S]$ be a maximal interval such that f_S is linear on it and $a_S \leq a < b_S$. In other words, a_S is the largest breakpoint of f_S smaller or equal to a and b_S is the smallest breakpoint larger than a. Let d_S be the slope of f_S on $[a_S, b_S]$. Choose $S^* \in X_a$ with minimum d_S. We show that S^* is optimal on $[a, b]$.

First, observe that by continuity there is an $\varepsilon > 0$ such that $f_{S^*}(\lambda) < f_S(\lambda)$ for all $\lambda \in [a, a+\varepsilon]$ and all $S \in X \setminus X_a$. Moreover, we can choose $\varepsilon < \min_{S \in X_a} b_S - a$ and $\varepsilon < b - a$. Then, by the choice of S^* as the solution in X_a with minimum slope, S^* is optimal in $[a, a+\varepsilon]$.

Hence, $f(\lambda) = f_{S^*}(\lambda)$ for $\lambda \in [a, a+\varepsilon]$. Assume there is a $\lambda' \in]a+\varepsilon, b]$ such that $f(\lambda') < f_{S^*}(\lambda')$. Choose $\mu \in [0, 1[$ such that $a + \varepsilon = \mu a + (1-\mu)\lambda'$. Then we have by concavity of f_{S^*} and linearity of f on $[a, b]$:

$$f(a+\varepsilon) = f_{S^*}(a+\varepsilon) \geq \mu f_{S^*}(a) + (1-\mu) f_{S^*}(\lambda') > \mu f(a) + (1-\mu) f(\lambda') = f(a+\varepsilon)$$

which is a contradiction. Hence, S^* is optimal on $[a, b]$, which is what we wanted to prove. □

Corollary 2.5. *Suppose the assumptions of Theorem 2.4 are satisfied. Then:*

(i) There is a representation system of size $\mathcal{O}(md_{\max})$.

(ii) If X is the set of all spanning trees or the set of all paths between two vertices, then there is a representation system of size $\mathcal{O}(nd_{\max})$.

Proof. This follows from Theorem 2.4 by the following observations:

(i) $K_{\max} \leq m$.

(ii) Since all spanning trees have size $n-1$ and since there is always a shortest path which is simple due to non-negativity of the costs we have $K_{\max} \leq n-1$ in both cases.

□

We observe that the bounded affine growth mentioned in the introduction is a special case of the assumptions of the previous results. Hence, we get the following corollary:

Corollary 2.6. *Let c_λ from Assumption 2.1 be given by $c_\lambda(e) = \min\{c(e) + \lambda d(e), u(e)\}$ where $d(e) \in \{0, 1, \ldots, d_{\max}\}$ for all $e \in E$. Then there is a representation system of size $\mathcal{O}(md_{\max})$. In the case of minimum spanning trees and shortest paths we even get $\mathcal{O}(nd_{\max})$.*

Proof. This follows from Corollary 2.5 by the fact that $\lambda \mapsto \min\{c(e) + \lambda d(e), u(e)\}$ is piecewise linear and concave with slope in $\{0, 1, \ldots, d_{\max}\}$ for every linear piece. □

Remark 2.7. The results we have so far do not tell us how to find such a representation system. We will investigate this separately for the different problems at a later point of time.

2.2. A Non-Polynomial Example

In her dissertation [Car83] Carstensen provided an example where no polynomial representation system exists. In her example, all costs have the form $c_\lambda(e) = c(e) + \lambda d(e)$ with nonnegative values $c(e)$ and $d(e)$. Then the optimal cost curve for $\lambda \in \mathbb{R}_{\geq 0}$ may in fact have $n^{\mathcal{O}(\log n)}$ breakpoints and is concave. Since all solutions have an affine cost curve, this means that any representation system is at least that large. This is clearly not polynomial in the number of vertices. Moreover, her graph is simple, which means that the smallest representation system is also not polynomial in the number of edges.

However, the number of breakpoints in her example is not exponential either. Moreover, the existence of pseudo-polynomial systems proven in the previous section shows that such an example is only possible, if the arithmetic values of the input are non-polynomial in the size of the graph. Hence, not all hope is lost that we are able to find quite small representation systems in many cases.

2.3. Smaller Systems for Special Cost Functions

2.3.1. Proportional Growth

In this subsection we consider the special case of a proportional growing cost function $c_\lambda(e) = c(e)(1+\lambda)$. It turns out that, in this case, solving the parametric problem is as easy as solving the nominal problem:

Theorem 2.8. *If in Assumption 2.1 we additionally assume that $c_\lambda(e) = c(e)(1+\lambda)$, then any optimal solution $S(0)$ to $(P(0))$ remains optimal for all $(P(\lambda))$. In particular, $R = \{S(0)\}$ is a representation system.*

Proof. Let $S \in X$ be an arbitrary feasible subset of E. Then it follows from the optimality of $S(0)$ for $(P(0))$:

$$c_\lambda(S) = (1+\lambda)c(S) \geq (1+\lambda)c(S(0)) = c_\lambda(S(0)).$$

Hence, $S(0)$ is also optimal for $(P(\lambda))$. □

2.3.2. Bounded Constant Growth

Now we consider cost functions of the type $c_\lambda(e) = \min\{c(e)+\lambda, u(e)\}$. Essentially we already have done all the hard work to prove the main result of this subsection:

Corollary 2.9. *Let c_λ in Assumption 2.1 be given by $c_\lambda(e) = \min\{c(e)+\lambda, u(e)\}$. Then there is a representation system of cardinality $\mathcal{O}(m)$. In the case of minimum spanning trees or shortest paths, we even get $\mathcal{O}(n)$.*

Proof. This follows by Corollary 2.6 and the fact that $d_{\max} = 1$. □

We will see how to find such a system for minimum spanning trees in Section 2.4 and for shortest paths in Section 3.3.

2.3.3. Constant Growth

We now investigate what happens if we additionally drop the upper bounds resulting in cost functions of the form $c_\lambda(e) = c(e) + \lambda$. The important observation to deal with this case is that for growing λ all solutions with the same number of edges grow by the same amount. In order to make this precise, we introduce some notations:

As in Assumption 2.1, let $K = \{|S| \mid S \in X\}$ be the set of different cardinalities of sets in X and $k = |K|$ the number of those cardinalities. For each $l \in K$ we write $X_l = \{S \in X \mid |S| = l\}$. Now we can divide $(P(\lambda))$ into k subproblems

$$(P_l(\lambda)) \qquad \min \left\{ c_\lambda(S) = \sum_{e \in S} c_\lambda(e) \;\middle|\; S \in X_l \right\}.$$

This leads to the following theorem:

Theorem 2.10. *If c_λ in Assumption 2.1 is given by $c_\lambda(e) = c(e) + \lambda$, then there exists a representation system of cardinality at most k.*

Proof. First we show that for each $l \in K$ there is a single solution S_l which solves $(P_l(\lambda))$ for all $\lambda \in \mathbb{R}_{\geq 0}$. In order to see this let S_l be a solution to $(P_l(0))$. Then, for all $S \in X_l$ and for all $\lambda \in \mathbb{R}_{\geq 0}$ we have $c_\lambda(S) = c(S) + l\lambda \geq c(S_l) + l\lambda = c_\lambda(S_l)$, which implies the optimality of S_l for all $(P_l(\lambda))$.

Since $\{X_l \mid l \in K\}$ is a partition of X, we have

$$\min_{S \in X} c_\lambda(S) = \min_{l \in K} \min_{S \in X_l} c_\lambda(S) = \min_{l \in K} c_\lambda(S_l).$$

This implies that always one of the S_l is optimal. Hence, $\{S_l \mid l \in K\}$ is a representation system with cardinality at most k. □

The following corollary shows how to express this in terms of m and n for different types of problems:

Corollary 2.11. *Let c_λ in Assumption 2.1 be given by $c_\lambda(e) = c(e) + \lambda$.*

(i) If X is an arbitrary nonempty subset of 2^E, then there is a representation system of cardinalty at most $m + 1$

(ii) If X is the set of all spanning trees, then there is a representation system of cardinality 1.

(iii) If X is the set of all paths between two distinct vertices, then there is a representation system of cardinality at most $n - 1$.

Proof. This follows directly from Theorem 2.10 by the following observations:

(i) $k = |K| \leq |\{0, 1, 2, \dots m\}| = m + 1$.

(ii) Since all spanning trees have exactly $n-1$ edges, we have $k = |K| = |\{n-1\}| = 1$.

(iii) Due to non-negativity of all costs, there is always a shortest path which is also simple. Hence, with the notation of Theorem 2.10, $\{S_l \mid l \in K, 1 \leq l \leq n-1\}$ is already a representation system of size less than or equal to $n-1$.

□

Remark 2.12. The magnitudes in Corollary 2.11 (i) and (iii) can also be obtained by Corollary 2.9 and the fact that constant growth is a special case of bounded constant growth by setting $u(e) = \infty$. However, the result in (ii) cannot be derived this way.

Remark 2.13. Another advantage of the proof of Theorem 2.10 is that it also suggests how to find the representation system: One only has to solve the k nonparametric problems $(P_l(0))$. In case of minimum spanning trees this is only one usual minimum spanning tree computation with respect to the nominal costs. How this works for shortest paths is explained in Section 3.1.

2.4. Strongly Polynomial Sized Systems for Minimum Spanning Trees

While determining a minimum spanning tree, only the order of the edge costs is important, not the specific values of the costs. This observation makes it possible to find strongly polynomial sized representation systems even in a quite general setting which includes (potentially bounded) affine growth.

Definition 2.14. In the setting of Assumption 2.1, we call $\lambda' \in \mathbb{R}_{\geq 0}$ a **changing point**, if there are two edges $e_1, e_2 \in E$ and an $\varepsilon > 0$ with the property that $c_{\lambda'}(e_1) = c_{\lambda'}(e_2)$ and $c_\lambda(e_1) < c_\lambda(e_2)$ either for any $\lambda \in]\lambda', \lambda' + \varepsilon[$ or for any $\lambda \in]\lambda' - \varepsilon, \lambda'[$. Moreover, we call λ' a **strong changing point**, if there is even a pair of edges $e_1, e_2 \in E$ and an $\varepsilon > 0$ such that $c_{\lambda'}(e_1) = c_{\lambda'}(e_2)$ and $c_\lambda(e_1) < c_\lambda(e_2)$ for any $\lambda \in]\lambda', \lambda' + \varepsilon[$.

In other words, a changing point is either an intersection point of two edge cost curves or a boundary point of an interval where two cost curves are identical. From all changing points we call only those strong changing points which are intersection points or right boundary points of an interval with equal cost curves.

We do our analysis of representation systems for minimum spanning trees under the following assumption:

Assumption 2.15. *Assume that Assumption 2.1 holds, X is the set of all spanning trees and the functions $\lambda \mapsto c_\lambda(e)$ are piecewise linear (in the sense of Definition 2.2) consisting of at most p pieces for all $e \in E$.*

Lemma 2.16. *Assume that Assumption 2.15 holds. Then there are at most* $\binom{m}{2}(4p-2)$ *changing points of which only* $\binom{m}{2}(2p-1)$ *are strong changing points.*

Proof. We prove this by looking at each single pair $e_1, e_2 \in E$. Since $\lambda \mapsto c_\lambda(e)$ has at most p linear pieces, there are at most $p-1$ breakpoints excluding 0 and ∞. Since this holds for e_1 and e_2, there are at most $2p-2$ points which are breakpoints for one of the functions. Hence, we can divide $\mathbb{R}_{\geq 0}$ into $2p-1$ intervals, such that $\lambda \mapsto c_\lambda(e_1)$ and $\lambda \mapsto c_\lambda(e_2)$ are linear on each interval. Consider one of those intervals. Either both functions are identical on this interval or they have at most one intersection point. In the first case, only the two boundary points of the interval (if they are not ∞) can be changing points and only the right boundary point can be a strong one. In the second case, only the intersection point can be a changing point. In both cases, there are at most two changing points per interval of which at most one is strong. Hence, for every pair, there are at most $4p-2$ changing points of which at most $2p-1$ are strong. The claim follows since we have $\binom{m}{2}$ pairs of edges. □

Lemma 2.17. *Suppose Assumption 2.15 holds. Let* $[a, b] \subseteq \mathbb{R}_{\geq 0}$ *be an interval with* $b > a$ *such that there is no strong changing point in the interior of the interval. Then there is a spanning tree* $S \in X$ *which is minimum for all* $\lambda \in [a, b]$.

Proof. By Lemma 2.16, there are only finitely many changing points. Hence, either there is a smallest changing point $\lambda'' \in]a, b[$ or there is no changing point at all in $]a, b[$. In the second case, set $\lambda'' = b$.

Set $\lambda' = \frac{a+\lambda''}{2}$. Sort the edges in E such that $c_{\lambda'}(e_1) \leq c_{\lambda'}(e_2) \leq \cdots \leq c_{\lambda'}(e_m)$. We show that the same order applies to all $\lambda \in [a, b]$: Assume this is not the case. Then there are indices $i < j$ and a $\lambda \in [a, b]$ such that $c_\lambda(e_i) > c_\lambda(e_j)$.

Case 1: $\lambda < \lambda'$. Define $\lambda^* = \inf\{x \mid c_x(e_i) \leq c_x(e_j), x \geq \lambda\}$. Due to continuity of the cost functions, this infimum is taken over a closed set which is bounded from below and not empty, because λ' is contained in it. Hence, the infimum is actually a minimum. Moreover, since λ is not in this set, we have $\lambda^* \in]\lambda, \lambda']$. But then, choosing $\varepsilon = \lambda^* - \lambda$, λ^* satisfies the conditions of being a changing point in contradiction to the assumption that there is no changing point in $]a, \lambda''[\supseteq]\lambda, \lambda'] \ni \lambda^*$.

Case 2: $\lambda > \lambda'$. Define $\lambda^* = \sup\{x \mid c_x(e_i) \leq c_x(e_j), x \leq \lambda\}$. Analogously to the first case we get $\lambda^* < \lambda$ and can conclude that λ^* is a strong changing point in contradiction to the fact that there is no strong changing point in $]a, b[$.

Hence, we have $c_\lambda(e_1) \leq c_\lambda(e_2) \leq \cdots \leq c_\lambda(e_m)$ for all $\lambda \in [a, b]$.

Recall how Kruskal's algorithm for finding a minimum spanning tree works (see e.g. [KN05]). The first step is to sort the edges with respect to the costs. We just showed that there is a common sorting for all $\lambda \in [a, b]$. If we take this sorting, the remaining algorithm works independently from the specific edge costs. Hence we get the same minimum spanning tree for all $\lambda \in [a, b]$. □

Theorem 2.18. *Assume that Assumption 2.15 holds. Then there is a representation system of size $\mathcal{O}(pm^2)$.*

Proof. By Lemma 2.16, we can divide $\mathbb{R}_{\geq 0}$ into $\mathcal{O}(pm^2)$ intervals such that there are no strong changing points in the interior of these intervals. By Lemma 2.17, it suffices to have one spanning tree in our representation system for each of these intervals. This already completes the proof. □

Corollary 2.19. *If Assumption 2.1 holds, X is the set of all spanning trees and $c_\lambda(e) = \min\{c(e) + \lambda d(e), u(e)\}$, then there is a representation system of size $\mathcal{O}(m^2)$.*

Proof. This follows from Theorem 2.18, because in the case of bounded constant growth we have $p = 2$. □

Remark 2.20. The proof of Theorem 2.18 is in some sense constructive: In order to find the representation system one only has to find all changing points. Then, for every strong changing point λ' Kruskal can be applied on a point which is in the interior of the interval between λ' and the next changing point. The resulting spanning trees form a representation system.

However, this idea can even be improved by the fact that the minimum spanning tree does not change totally when λ traverses a strong changing point, but can be updated:

Suppose we have a strong changing point λ' and a minimum spanning tree for $(P(\lambda'))$. Let e_1 and e_2 be the pair of edges such that $c_{\lambda'}(e_1) = c_{\lambda'}(e_2)$ and $c_\lambda(e_1) < c_\lambda(e_2)$ for $\lambda \in]\lambda', \lambda' + \varepsilon[$ according to Definition 2.14. Assume for the moment that there is only one such pair.

The only change in the order of the edge costs which can be induced by the changing point is the change of the order of e_1 and e_2. Hence, the only necessary change on the minimum spanning tree might be to replace e_2 by e_1. We test whether e_2 is in the spanning tree and e_1 not. If yes, we also test whether replacing e_2 by e_1 is possible without destroying the tree. This can be done by adding e_1 to the tree and testing whether e_2 is on the unique cycle produced by adding e_1. If yes, do the replacement, if no, do nothing.

If there are multiple pairs fulfilling the requirement for the changing point, just apply this procedure sequentially for all those pairs.

Hence, in the setting of Corollary 2.19, computing the representation system takes a total time of $\mathcal{O}((n + \log m)m^2)$: Finding the changing points can be done in constant time for each of the $\mathcal{O}(m^2)$ pairs. Sorting them takes $\mathcal{O}(m^2 \log m)$ time. The initial tree can clearly be found in the claimed total running time (e.g. using Kruskal, compare [KN05]). Each of the $\mathcal{O}(m^2)$ updates takes $\mathcal{O}(n)$ time for the cycle check described above, which can be settled by a common search strategy (e.g. breadth first search) on the current tree, which has only $\mathcal{O}(n)$ edges.

Remark 2.21. During the algorithm described in the previous remark, each change of the spanning tree at a strong changing point results in a decrease of the slope of the cost function. Comparing with the proof of Theorem 2.4 and Corollary 2.5 this means that the resulting representation system is in fact one of size $\mathcal{O}(nd_{\max})$. In particular, in the case of bounded constant growth, the described algorithm is a way to find a representation system of size $\mathcal{O}(n)$ as claimed in Corollary 2.9.

2.5. Summary

The following table summarizes the upper bounds on the size of a minimum representation system proven in this chapter. The columns determine the type of problem we consider (X is either a general subset of 2^E, the set of all paths between two vertices or the set of all spanning trees). The rows determine the type of dependence on λ.

	General	SP	MST
(Bounded) Affine Growth	$\mathcal{O}(K_{\max}d_{\max}) \subseteq \mathcal{O}(md_{\max})$	$\mathcal{O}(nd_{\max})$	$\mathcal{O}(m^2)$
Bounded Constant Growth	$\mathcal{O}(K_{\max}) \subseteq \mathcal{O}(m)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Constant Growth	$\mathcal{O}(k) \subseteq \mathcal{O}(m)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$
Proportional Growth	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

3. Finding Representation Systems for Shortest Paths

Due to non-negativity of all costs we have that, if there is a shortest path between two vertices, it can be assumed to be a simple one. This means that we obtain the parametric shortest path problem from $(P(\lambda))$ by choosing X to be the set of all simple paths between two vertices s and t.

Starting from there, this chapter derives more specific algorithms for the parametric shortest path problem for different types of cost functions.

3.1. Constant Growth

Let c_λ be given by $c_\lambda(e) = c(e) + \lambda$. In Corollary 2.11 we already found out that there is a representation system of size at most $n - 1$.

Recall from the proof of Theorem 2.10 that we can find this system by solving the k nonparametric problems

$$(P_l(0)) \qquad \min \{c(S) \mid S \in X_l\}$$

where $X_l = \{S \in X \mid |S| = l\}$. This can be done recursively: The idea is that any shortest path from s to an arbitrary node v using exactly l edges consists of a shortest path from s to a predecessor of v using exactly $l - 1$ edges and an l^{th} edge from this predecessor to v.

Formally, we write $d_l(v)$ for the length of the shortest path from s to a node v using exactly l edges. Moreover, we write $\pi_l(v)$ for the predecessor of v in the shortest path from s to v using exactly l edges. Then we have

$$d_0(v) = \begin{cases} 0, & v = s \\ +\infty, & v \neq s \end{cases}$$

and

$$\pi_0(v) = \text{nil}.$$

Now, one can compute d_l from d_{l-1} using the observation described above by the formula

$$d_l(v) = \min_{e=(u,v)\in\delta^-(v)} (c(e) + d_{l-1}(u))$$

where $\delta^-(v)$ is the set of incoming edges at v.

If this minimum is finite and attained by edge (u, v), we set $\pi_l(v) = u$. Otherwise we set $\pi_l(v) = \text{nil}$.

In a final step, we can find our representation system by backtracking π: We observe that there is a path from s to t using exactly l edges, if and only if $\pi_l(t) \neq \text{nil}$. If we write $v_l^{(l)} = t$ and $v_{i-1}^{(l)} = \pi_i(v_i^{(l)})$ until $v_0^{(l)} = s$, then the edges of the shortest of those paths are given by

$$S_l = \left\{ \left(v_i^{(l)}, v_{i+1}^{(l)} \right) \;\middle|\; 0 \leq i \leq l-1 \right\}.$$

Hence, our representation system is

$$R = \{S_l \mid 1 \leq l \leq n-1, \pi_l(t) \neq \text{nil}\}.$$

For finding the d and π values, we have to consider all edges and all vertices exactly once for all $l \in \{1, 2, \ldots, n-1\}$. This yields a total number of $\mathcal{O}(n(n+m))$ computations. The backtracking takes $\mathcal{O}(n^2)$ time. Hence, our total time is $\mathcal{O}(n(n+m))$.

3.2. Partial Constant Growth

Partial constant growth means that some of the edge costs grow with a constant growth rate while others remain constant. For this kind of growth, Karp and Orlin presented two parametric shortest path algorithms in [KO81]. Each of them computes a sequence of rooted trees such that for each λ one of them is a shortest path tree. In this section we want to reformulate the second of those two algorithms for finding a representation system. The setting in this section will be:

Assumption 3.1. *Suppose that Assumption 2.1 holds. Moreover, s and t are two distinct vertices of G such that there is a path from s to all other vertices in G. X is the set of all simple paths from s to t. The set E of edges is partitioned into two sets E_1 and E_2 and the cost functions c_λ are given by:*

$$c_\lambda(e) = \begin{cases} c(e), & e \in E_1 \\ c(e) + \lambda, & e \in E_2 \end{cases}$$

The assumption that all vertices are reachable from s can be made without loss of generality: If not, just remove all unreachable vertices from G and all the results of this section apply for the smaller graph. In particular, all running times and cardinalities become only smaller by the removal. The removal itself takes $\mathcal{O}(n + m)$ time using a common search strategy, e.g. breadth-first search.

First we observe that we already know an upper bound for the size of a representation system:

Corollary 3.2. *In the setting of Assumption 3.1 there is a representation system of cardinality $\mathcal{O}(n)$.*

Proof. This follows by Corollary 2.5 and the fact that $d_{\max} = 1$. □

The idea of the algorithm is to compute the rooted shortest path tree for $\lambda = 0$ and then increase λ from 0 to $+\infty$ and calculate where and how the shortest path tree changes.

In the setting of [KO81] the edges in E_2 have a subtractive constant instead of an additive one. They let λ increase from $-\infty$ to the greatest value λ^* for which the graph does not contain a negative cycle. If we wanted to do this analogously, we would have do decrease λ from $+\infty$ to 0. Since we assume non-negativity of all costs, we never have a negative cycle in this range. However, the same idea works, if we do it the other way around and increase λ from 0 to $+\infty$.

We introduce some notations we need for the algorithm: For an edge $e \in E$ we write $c_1(e) = c(e)$ and

$$c_2(e) = \begin{cases} 0, & e \in E_1 \\ 1, & e \in E_2 \end{cases}$$

Then, $c_\lambda(e) = c_1(e) + \lambda c_2(e)$. Moreover, given a rooted spanning tree T of G with root s, we write $d_i(T, v)$ for the distance of v from s in T with respect to the costs c_i for $i = 1, 2$. Then, $d_2(T, v)$ is exactly the number of edges from E_2 in the unique path from s to v in T. For an edge $e = (u, v)$ we write $\Delta_i(T, e) = d_i(T, u) + c_i(e) - d_i(T, v)$ for $i = 1, 2$. Finally, we denote $P(T, v) = \{w \in V \mid \text{there is a path from } v \text{ to } w \text{ in } T\}$, which is exactly the set of vertices of the subtree of T with root v.

Now we are ready to formulate the algorithm:

Algorithm 1 Modified Karp-Orlin Algorithm

Input: An instance satisfying Assumption 3.1
Output: A representation system for $(P(\lambda))$

1: Compute a shortest path tree T_1 with root s for $\lambda = \lambda_0 := 0$, e.g. using Dijkstra's algorithm.
2: Let R be the set consisting of the path from s to t in T_1.
3: For each vertex $v \in V$ compute $d_1(T_1, v)$ and $d_2(T_1, v)$.
4: For each edge $e = (u, v) \in E$ calculate

$$p(e) := \begin{cases} \infty, & \Delta_2(T_1, e) \geq 0 \\ -\frac{\Delta_1(T_1,e)}{\Delta_2(T_1,e)}, & \Delta_2(T_1, e) < 0 \end{cases}$$

5: $j := 1$.
6: **while** $\lambda_j := \min_{e \in E} p(e) < \infty$ **do**
7: Let $e' = (u, v) \in E$ be the edge for which the minimum is attained.
8: Obtain T_{j+1} from T_j by replacing the edge ending at v by e'.
9: **if** $t \in P(T_j, v)$ **then**
10: Add the path from s to t in T_{j+1} to R.
11: **end if**
12: Calculate for all $w \in V$ and $i = 1, 2$:

$$d_i(T_{j+1}, w) := \begin{cases} d_i(T_j, w), & w \notin P(T_j, v) \\ d_i(T_j, w) + \Delta_i(T_j, e'), & w \in P(T_j, v) \end{cases}$$

13: For all $e \in E$ with either a starting point or an end point in $P(T_j, v)$ update:

$$p(e) := \begin{cases} \infty, & \Delta_2(T_{j+1}, e) \geq 0 \\ -\frac{\Delta_1(T_{j+1},e)}{\Delta_2(T_{j+1},e)}, & \Delta_2(T_{j+1}, e) < 0 \end{cases}$$

14: $j := j + 1$.
15: **end while**
16: **return** R

The following proof of the correctness and running time uses the basic ideas from [KO81], but is completely reformulated, more detailed and adapted to our scenario.

Lemma 3.3. *For all T_j in Algorithm 1, it holds:*

(i) T_j is in fact a rooted spanning tree with root s.

(ii) The formula in line 12 correctly computes $d_i(T_j, v)$ for $i = 1, 2$ and all $v \in V$.

Proof. We show both claims simultaneously by induction over the iteration counter j. T_1 is obviously a rooted spanning tree since it is a shortest path tree for s. Also, the

$d_i(T_1, v)$ values are computed from scratch and hence correct. This settles the induction basis.

Now let us assume that T_j is a tree as claimed and all $d_i(T_k, v)$ values have been computed correctly for $k = 1, \ldots, j$. We show that this also holds for T_{j+1}.

Let $e' = (u, v)$ be the chosen edge with minimum $p(e)$ value. We denote the unique path from s to v in T_j by P_v and the unique path from s to u in T_j by P_u.

We show that v does not lie on P_u. Let k be the last iteration in which we changed $p(e')$. If this was before we entered the while loop, set $k = 0$. Since $p(e')$ is finite, the formulas in line 4 and line 13 ensure that $\Delta_2(T_{k+1}, e') < 0$. Since $p(e')$ was not updated after iteration k, by construction of line 13 and 12 we obtain that $\Delta_2(T_j, e')$ is still equal to $\Delta_2(T_{k+1}, e')$ and hence smaller than zero. Enrolling the definition of Δ_2 yields

$$d_2(T_j, v) > d_2(T_j, u) + c_2(e) \geq d_2(T_j, u).$$

We observe that the left hand side of this inequality is exactly the number of edges of E_2 in P_v and the right hand side the number of edges of E_2 in P_u. The inequality shows that P_v is not a subpath of P_u and by the uniqueness of the paths we get that v does not lie on P_u.

This lets us conclude that the edge removed in line 8 is not part of P_u. Hence, we can still reach v and all other vertices in $P(T_j, v)$ after the replacement of the edges. Moreover, we can reach all other vertices in G via the old paths. The number of edges did not change by the replacement. Hence, T_{j+1} is again a rooted spanning tree, which proves (i).

By this proof we also see that for any vertex w the unique path from s to w changes if and only if $w \in P(T_j, v)$. In this case, P_u and e' are used instead of P_v. This means that

$$d_i(T_{j+1}, w) - d_i(T_j, w) = d_i(T_j, u) + c_i(e') - d_i(T_j, v) = \Delta_i(T_j, e').$$

This settles the proof for (ii). □

Lemma 3.4. *In the setting of Algorithm 1, if T_j is optimal for $\lambda_{j-1} \in \mathbb{R}_{\geq 0}$, then it holds:*

(i) $\lambda_j \geq \lambda_{j-1}$

(ii) *T_j is optimal for the whole interval* $[\lambda_{j-1}, \lambda_j]$.

(iii) *T_{j+1} is optimal for* λ_j.

Proof. (i) We have to show that for all $e = (u, v) \in E$ it holds that $p(e) \geq \lambda_{j-1}$ at the beginning of iteration j. Again, we observe that if the Δ_i values change for e, also

the corresponding $p(e)$ value is updated. Hence, by line 4 and line 13 we have at the beginning of the j^{th} iteration:

$$p(e) = \begin{cases} \infty, & \Delta_2(T_j, e) \geq 0 \\ -\frac{\Delta_1(T_j,e)}{\Delta_2(T_j,e)}, & \Delta_2(T_j, e) < 0 \end{cases}$$

If we are in the first case, then $p(e) \geq \lambda_{j-1}$ is clearly satisfied. Assume now, that we are in the second case, i.e. $p(e) = -\frac{\Delta_1(T_j,e)}{\Delta_2(T_j,e)}$ and $\Delta_2(T_j, e) < 0$.

Let P be the unique path from s to v in T_j and P^* the path from s to v which consists of the unique path from s to u in T_j and the edge e. By optimality of T_j for λ_{j-1} we get $c_{\lambda_{j-1}}(P) \leq c_{\lambda_{j-1}}(P^*)$. This yields

$$d_1(T_j, v) + \lambda_{j-1} d_2(T_j, v) \leq d_1(T_j, u) + \lambda_{j-1} d_2(T_j, u) + c_1(e) + \lambda_{j-1} c_2(e).$$

By solving this for λ_{j-1}, plugging in the definition of Δ_1 and Δ_2 and using the fact that $\Delta_2(T_j, e) < 0$ we obtain

$$\lambda_{j-1} \leq -\frac{\Delta_1(T_j, e)}{\Delta_2(T_j, e)} = p(e),$$

which is what we wanted to prove.

(ii) Assume for the sake of a contradiction that there is a $\lambda' \in [\lambda_{j-1}, \lambda_j]$ and a vertex w such that $\text{dist}_{c_{\lambda'}}(s, w, G) < \text{dist}_{c_{\lambda'}}(s, w, T_j)$, where $\text{dist}_{c'}(s', t', G')$ stands for the length of the shortest path from s' to t' in G' with respect to the costs c'. By optimality of T_j for λ_{j-1} we get $\lambda' \in]\lambda_{j-1}, \lambda_j]$. Let P^* be a shortest path from s to w in G with respect to $c_{\lambda'}$. Since $\text{dist}_{c_{\lambda'}}(s, s, G) = 0 = \text{dist}_{c_{\lambda'}}(s, s, T_j)$, there is a first vertex $v \neq s$ on P^* with the property that $\text{dist}_{c_{\lambda'}}(s, v, G) < \text{dist}_{c_{\lambda'}}(s, v, T_j)$. Let u be the predecessor of v on P^* and $e = (u, v)$ the edge between u and v on P^*. By choice of v and the fact that the subpath from s to u of P^* is also a shortest path with respect to $c_{\lambda'}$ in G, we get that $\text{dist}_{c_{\lambda'}}(s, u, G) = \text{dist}_{c_{\lambda'}}(s, u, T_j)$. So we can assume without loss of generality that P^* without e lies completely in T_j. This lets us conclude that

$$\begin{aligned} d_1(T_j, v) + \lambda' d_2(T_j, v) &= \text{dist}_{c_{\lambda'}}(s, v, T_j) \\ &> \text{dist}_{c_{\lambda'}}(s, v, G) \\ &= c_{\lambda'}(e) + \text{dist}_{c_{\lambda'}}(s, u, G) \\ &= c_{\lambda'}(e) + \text{dist}_{c_{\lambda'}}(s, u, T_j) \\ &= c_1(e) + \lambda' c_2(e) + d_1(T_j, u) + \lambda' d_2(T_j, u). \end{aligned}$$

Rearranging this and plugging in the definitions of Δ_1 and Δ_2 yields

$$\Delta_1(T_j, e) < -\lambda' \Delta_2(T_j, e).$$

Case 1: $\Delta_2(T_j, e) < 0$. Then we have $\lambda_j \geq \lambda' > -\frac{\Delta_1(T_j,e)}{\Delta_2(T_j,e)} = p(e)$, which contradicts to the choice of λ_j in line 6.

Case 2: $\Delta_2(T_j, e) \geq 0$. We observe that by optimality of T_j for λ_{j-1} we have

$$\begin{aligned} c_{\lambda_{j-1}}(P^*) &\geq \mathrm{dist}_{c_{\lambda_{j-1}}}(s, v, T_j) \\ &= \mathrm{dist}_{c_{\lambda'}}(s, v, T_j) + (\lambda_{j-1} - \lambda')d_2(T_j, v). \end{aligned}$$

On the other hand we have:

$$\begin{aligned} c_{\lambda_{j-1}}(P^*) &= c_{\lambda'}(P^*) + (\lambda_{j-1} - \lambda')(c_2(e) + d_2(T_j, u)) \\ &< \mathrm{dist}_{c_{\lambda'}}(s, v, T_j) + (\lambda_{j-1} - \lambda')(c_2(e) + d_2(T_j, u)). \end{aligned}$$

Combining the last two inequalities yields

$$(\lambda_{j-1} - \lambda')\Delta_2(T_j, e) > 0,$$

which is a contradiction because $\lambda_{j-1} - \lambda' < 0$ and $\Delta_2(T_j, e) \geq 0$.

(iii) Part (ii) implies that T_j is optimal for λ_j. For $w \notin P(T_j, v)$ the unique path from s to w remains unchanged while switching from T_j to T_{j+1}. For $w \in P(T_j, v)$ we have:

$$\begin{aligned} \mathrm{dist}_{\lambda_j}(s, w, T_{j+1}) &= \mathrm{dist}_{\lambda_j}(s, w, T_j) + \Delta_1(T_j, e') + \lambda_j \Delta_2(T_j, e') \\ &= \mathrm{dist}_{\lambda_j}(s, w, T_j) + \Delta_1(T_j, e') - \frac{\Delta_1(T_j, e')}{\Delta_2(T_j, e')}\Delta_2(T_j, e') \\ &= \mathrm{dist}_{\lambda_j}(s, w, T_j). \end{aligned}$$

In both cases, the distances from s to w with respect to c_{λ_j} are the same in T_j and T_{j+1}. Hence, T_{j+1} is optimal for λ_j, too.

□

Lemma 3.5. *Algorithm 1 stops after $\mathcal{O}(n^2)$ iterations.*

Proof. We show that $\sum_{w \in V} d_2(T_j, w)$ decreases by at least one in every step. This yields the claim because the sum is always positive and at the beginning the term for each vertex is at most $n - 1$.

We observe that in every iteration we have $v \in P(T_j, v)$. Moreover, as argued in the proof of the previous lemma, $\Delta_2(T_j, e') < 0$. Since this is integer, we have that $d_2(T_{j+1}, v) \leq d_2(T_j, v) - 1$ and for all other $w \in V$ we have $d_2(T_{j+1}, w) \leq d_2(T_j, w)$. Hence, the considered sum becomes smaller by at least one. □

Theorem 3.6. *Algorithm 1 computes finite sequences $0 = \lambda_0, \lambda_1, \ldots, \lambda_{j_{\max}} = \infty$ and $T_1, \ldots, T_{j_{\max}}$ such that the first one is increasing and T_j is a shortest path tree with respect to c_λ for all $\lambda \in [\lambda_{j-1}, \lambda_j] \setminus \{\infty\}$.*

Proof. The finiteness of the sequences follows by Lemma 3.5 and the other properties by induction over j where our initial choice of T_1 settles the induction basis and Lemma 3.4 the induction step. □

Theorem 3.7. *Algorithm 1 returns in fact a representation system of size $\mathcal{O}(n)$ for $(P(\lambda))$.*

Proof. By Theorem 3.6 we get a representation system by extracting the unique s-t-path from all T_j. We observe that this unique s-t-path changes during an iteration j, if and only if t is in $P(T_j, v)$. Similarly to the proof of Lemma 3.5, this happens at most $\mathcal{O}(n)$ times, since in this case we always reduce $d_2(T_j, v)$ by at least one. Hence, it suffices to add a new path to R, if and only if the condition in line 9 is satisfied and this happens at most $\mathcal{O}(n)$ times. □

Theorem 3.8. *A basic implementation of Algorithm 1 runs in $\mathcal{O}(n^2m)$ time.*

Proof. We can find T_1 with a simple implementation of Dijkstra's algorithm in $\mathcal{O}(n^2)$ (see e.g. [KN05]). Initializing the d_i values can be done using a common strategy for traversing a tree, e.g. breadth-first search, in $\mathcal{O}(n+m)$. Initializing the $p(e)$ values takes $\mathcal{O}(m)$ time. We run through the while loop $\mathcal{O}(n^2)$ times due to Lemma 3.5. In each iteration we have to:

- Find the edge with minimum $p(e)$ value, which takes $\mathcal{O}(m)$ time.
- Find $P(T_j, v)$, which can again be done by breadth-first search starting at v in $\mathcal{O}(m + n)$ time.
- Potentially add a new path to R, which can be done in $\mathcal{O}(n)$ steps.
- Update the d_i values, which takes $\mathcal{O}(n)$ steps.
- Update the $p(e)$ values, which takes $\mathcal{O}(m)$ steps.

In total, this yields the claimed running time of $\mathcal{O}(n^2m)$. □

Remark 3.9. The implementation and running time estimation described in Theorem 3.8 can be improved as follows:

As observed in the proof of Lemma 3.5, the d_i values of each vertex are updated at most $\mathcal{O}(n)$ times. Moreover, since the $p(e)$ value of an edge e is only updated if either its starting point's or its end point's d_i values have been updated, the updating of the $p(e)$ values takes a total time of at most $\mathcal{O}(nm)$. If we additionally organize the $p(e)$ values in a data structure for priority queues, we obtain better running times, depending on the type of the data structure:

- In [KO81] Karp and Orlin used a balanced tree for this purpose, where we can add and delete an entry and extract the minimum entry in $\mathcal{O}(\log n)$ time. Then the total running time reduces to $\mathcal{O}(nm \log n)$.

- Young, Tarjan and Orlin improved this to $\mathcal{O}(nm + n^2 \log n)$ in [YTO91] using a Fibonacci heap as priority queue.

3.3. Bounded Constant Growth

The goal of this section is to give an efficient algorithm for finding a representation system for the single pair shortest path problem with bounded constant growth. That means our setting is:

Assumption 3.10. *Suppose that Assumption 2.1 holds and the cost functions are given by* $c_\lambda(e) = \min\{c(e) + \lambda, u(e)\}$. *Moreover,* s *and* t *are two distinct vertices of* G *such that there is a path from* s *to all other vertices in* G *and* X *is the set of all simple paths from* s *to* t.

A first simple idea is the following: We observe that each of the m cost functions $\lambda \mapsto c_\lambda(e)$ is piecewise linear and has at most one breakpoint in the interior of its domain $[0, \infty[$. This leads to a total amount of m breakpoints such that between two successive breakpoints all cost functions are piecewise linear with slope zero or one. Thus, we get $m + 1$ subproblems and each of them is a valid instance for Algorithm 1. The union of all $m+1$ resulting systems, each of which has size $\mathcal{O}(n)$, is a representation system for the original problem with bounded constant growth. However, as an upper bound for the size we only get $\mathcal{O}(nm)$.

Another simple and even better idea needs only a single call of Algorithm 1 by modifying the input graph in a preprocessing step: For every edge $e \in E$ which has in fact a breakpoint, replace e by two parallel edges e_1 and e_2. Set $c_\lambda(e_1) = c(e) + \lambda$ and $c_\lambda(e_2) = u(e)$. Denote the resulting graph by $G' = (V, E')$. Apply Algorithm 1 on G'. From the resulting representation system R' for G' we get R by substituting each edge $e' \in E' \setminus E$ by the edge which was replaced by e'. Then, R is a representation system for G. This approach yields the same asymptotic running time and size of R as Algorithm 1, since we at most doubled the number of edges.

Now we are able to combine both methods to obtain a third and most efficient way to handle bounded constant growth. The idea is to calculate the breakpoints as described in the first method. Then we start increasing λ as in Algorithm 1, pretending we have unbounded (partial) constant growth. Whenever we reach a breakpoint, we replace the edge with growing costs by one with constant costs and do all necessary updates.

Before we give the pseudocode for the algorithm, we have to extend our notation: There are no longer sets E_1 and E_2 which also define c_2, d_2, Δ_2 and $p(e)$. However, we can still use the same notation as in Algorithm 1 by updating the sets E_1, E_2 as well as the values c_2, d_2, Δ_2 and $p(e)$ with growing λ.

Algorithm 2 Extended Karp-Orlin Algorithm

Input: An instance satisfying Assumption 3.10
Output: A representation system for $(P(\lambda))$

Compute a shortest path tree T_1 with root s for $\lambda = \lambda_0 := 0$.
Let R be the set consisting of the path from s to t in T_1.
Compute the set $B := \{u(e) - c(e) > 0 \mid e \in E\} \cup \{\infty\}$ of positive breakpoints.
Store for each $b \in B \cup \{0\}$: $E(b) := \{e \in E \mid u(e) - c(e) = b\}$.
Set $E_1 := E(0)$ and $E_2 := E \setminus E(0)$.
For each vertex $v \in V$ compute $d_1(T_1, v)$ and $d_2(T_1, v)$ with respect to E_1 and E_2.
For each edge $e = (u, v) \in E$ calculate

$$p(e) := \begin{cases} \infty, & \Delta_2(T_1, e) \geq 0 \\ -\frac{\Delta_1(T_1,e)}{\Delta_2(T_1,e)}, & \Delta_2(T_1, e) < 0 \end{cases}$$

$j := 1$.
while $B \neq \emptyset$ **do**
    Extract the minimum b from B.
    while $\lambda_j := p(u, v) := p(e') := \min_{e \in E} p(e) < b$ **do**
        Obtain T_{j+1} from T_j by replacing the edge ending at v by e'.
        if $t \in P(T_j, v)$ **then**
            Add the path from s to t in T_{j+1} to R.
        end if
        Calculate for all $w \in V$ and $i = 1, 2$:

$$d_i(T_{j+1}, w) := \begin{cases} d_i(T_j, w), & w \notin P(T_j, v) \\ d_i(T_j, w) + \Delta_i(T_j, e'), & w \in P(T_j, v) \end{cases}$$

        For all $e \in E$ with either a starting point or an end point in $P(T_j, v)$ update:

$$p(e) := \begin{cases} \infty, & \Delta_2(T_{j+1}, e) \geq 0 \\ -\frac{\Delta_1(T_{j+1},e)}{\Delta_2(T_{j+1},e)}, & \Delta_2(T_{j+1}, e) < 0 \end{cases}$$

        $j := j + 1$
    end while
    Set $E_1 := E_1 \cup E(b)$ and $E_2 := E_2 \setminus E(b)$.
    for each $e' = (u, v) \in E(b)$ **do**
        For all $w \in P(T_j, v)$: $d_2(T_j, w) := d_2(T_j, w) - 1$.
        For all $e \in E$ with either a starting point or an end point in $P(T_j, v)$ update:

$$p(e) := \begin{cases} \infty, & \Delta_2(T_j, e) \geq 0 \\ -\frac{\Delta_1(T_j,e)}{\Delta_2(T_j,e)}, & \Delta_2(T_j, e) < 0 \end{cases}$$

    end for
end while
return R

Theorem 3.11. *Theorem 3.6 also holds for Algorithm 2.*

Proof. Between two breakpoints, Algorithm 2 behaves exactly like Algorithm 1. Hence, we only have to show that the breakpoints do not destroy the correctness of the algorithm. Formally, this means we have to show that:

(i) In line 23, $p(e) \geq b$.

(ii) T_j is optimal for $\lambda \in [\lambda_{j-1}, \lambda_j]$, even if there is a $b \in B$ in the interior of this interval.

These two claims are consequences of the following observations:

(i) This can be shown as in the proof of Lemma 3.4 (i), if we replace λ_{j-1} by b.

(ii) We divide the interval $\lambda \in [\lambda_{j-1}, \lambda_j]$ into subintervals such that in the interior of the subintervals there is no point in B, but each boundary point is either in B or λ_{j-1} or λ_j. Then the induction step proven in Lemma 3.4 can be done similarly applied on the subintervals. Hence, Algorithm 2 computes the correct trees.

□

Theorem 3.12. *Algorithm 2 returns a representation system of size $\mathcal{O}(n)$. A basic implementation runs in $\mathcal{O}(m(n^2 + m))$ time.*

Proof. The total number of iterations of the inner while loop as well as the size of the returned representation system can be bounded with the same arguments as in the analysis of Algorithm 1. The outer while loop is called at most $\mathcal{O}(m)$ times.

We observe:

- Decreasing d_2 can happen at most $\mathcal{O}(n)$ times for every vertex, and hence, at most $\mathcal{O}(n^2)$ times in total, no matter whether this happens in the inner while loop or in the for each loop.
- Updating $p(e)$ happens at most $\mathcal{O}(n)$ times for each edge, since this only happens if the d_2 value of one of its end points is decreased. Hence, this takes a total time of $\mathcal{O}(nm)$.

These observations already imply a total time of $\mathcal{O}(n(n + m))$ for the for each loop, because inside this loop only such updates happen.

Extracting the minimum of B and updating E_1 and E_2 takes at most $\mathcal{O}(m)$. Since this happens in each iteration of the outer loop, we get a total time of $\mathcal{O}(m^2)$ for these operations.

The initialization before we enter the outer while loop can easily be achieved within the claimed running time. The rest of the algorithm is exactly the same as Algorithm 1, which means that $\mathcal{O}(m(n^2 + m))$ time suffices. □

Remark 3.13. If G is a simple graph (which is a valid assumption in this setting since if there are parallel edges, just replace them by one edge with minimum $c(e)$ and minimum $u(e)$ value), then the running time of Algorithm 2 is not worse than the time of Algorithm 1. Again, the running time can be improved by sharper analysis and the use of data structures for priority queues.

3.4. Bounded Affine Growth

We consider even weaker assumptions, this time the setting of bounded affine growth:

Assumption 3.14. *Suppose that Assumption 2.1 holds and the cost functions are given by* $c_\lambda(e) = \min\{c(e) + \lambda d(e), u(e)\}$*, where* $d(e) \in \{0, 1, \ldots, d_{\max}\}$*. Moreover,* s *and* t *are two distinct vertices of* G *such that there is a path from* s *to all other vertices in* G *and* X *is the set of all simple paths from* s *to* t*.*

A simple idea to handle this case is to modify the input graph again: Replace an edge e with $d(e) > 1$ by a series of $d(e)$ edges $e_1, \ldots, e_{d(e)}$. Define $d(e_j) = 1$, $c(e_j) = \frac{c(e)}{d(e)}$ and $u(e_j) = \frac{u(e)}{d(e)}$ for $j = 1, \ldots, d(e)$. Then we obtain a valid instance for Algorithm 2. However, this transformation increases the number of vertices to $\mathcal{O}(n + md_{\max})$ and hence the n in running time and size of the resulting representation system has to be replaced by this number. Therefore, this approach yields only a pseudo-polynomial representation system. On the other hand, this is all we can expect due to Carstensen's example in [Car83] mentioned in Section 2.2.

Again, this approach can be improved by a similar idea as in the previous section, where we adapted Algorithm 1 to obtain Algorithm 2: We can adapt Algorithm 2 such that it works like it would work on the modified graph instead of actually replacing each edge by at most $d_{\max}$ edges. One only has to ensure that the d_2 values are initialized and updated accordingly. However, the running time and the size of the resulting representation system still increase by the factor $d_{\max}$, since the initial values of d_2 increase by this factor, and they determine the number of steps calculated in Lemma 3.5.

4. Approximation Systems

As we saw earlier, there may not always be a polynomial representation system for $(P(\lambda))$. In the introduction we mentioned that we are going to fix this issue by abandoning the goal of having an optimal solution for every $\lambda \in \mathbb{R}_{\geq 0}$. Instead of that, we try to find a smaller (the goal will be polynomial) system which only ensures that for every $\lambda \in \mathbb{R}_{\geq 0}$ there is a member in the system which is an approximately optimal solution. Formally, we define:

Definition 4.1. For $(P(\lambda))$ and a real number $\alpha > 1$ let $R \subseteq X$ be a set of feasible solutions with the property that for all $\lambda \in \mathbb{R}_{\geq 0}$:

$$\frac{\min_{S \in R} c_\lambda(S)}{\min_{S \in X} c_\lambda(S)} \leq \alpha.$$

Then we call R an **approximation system** of ratio α.

In terms of robust optimization, an approximation system ensures a min max relative regret of α.

4.1. A General Algorithm

In this section we want to explore how to find an approximation system under the following assumption:

Assumption 4.2. *Suppose Assumption 2.1 holds. Moreover, the cost functions $\lambda \mapsto c_\lambda(e)$ are nondecreasing and concave. Finally, $\alpha > 1$ is a real number.*

Observe that this implies that $\lambda \mapsto c_\lambda(S)$ is nondecreasing and concave for all $S \in X$. In addition, by Lemma 2.3 also $\lambda \mapsto \min_{S \in X} c_\lambda(S)$ is nondecreasing and concave.

The key idea, how we want to find polynomial approximation systems, is the following lemma:

Lemma 4.3. *Suppose Assumption 4.2 holds. If S^* is an optimal solution for a particular $\lambda^* \in \mathbb{R}_{\geq 0}$, then we have for all $\lambda \in [\frac{\lambda^*}{\alpha}, \alpha\lambda^*]$:*

$$\frac{c_\lambda(S^*)}{\min_{S \in X} c_\lambda(S)} \leq \alpha.$$

Proof. First, let $\lambda \in [\lambda^*, \alpha\lambda^*]$ be arbitrary. Then, since the optimal cost curve and each single cost function are nondecreasing as observed above, we have

$$\min_{S \in X} c_\lambda(S) \geq \min_{S \in X} c_{\lambda^*}(S) = c_{\lambda^*}(S^*)$$

and

$$c_\lambda(S^*) \leq c_{\alpha\lambda^*}(S^*).$$

Those two inequalities yield

$$\frac{c_\lambda(S^*)}{\min_{S \in X} c_\lambda(S)} \leq \frac{c_{\alpha\lambda^*}(S^*)}{c_{\lambda^*}(S^*)}.$$

It remains to show that the right hand side is less than or equal to α. By concavity and non-negativity of the cost functions we have

$$c_{\lambda^*}(S^*) \geq (1 - \frac{1}{\alpha})c_0(S^*) + \frac{1}{\alpha}c_{\alpha\lambda^*}(S^*) \geq \frac{1}{\alpha}c_{\alpha\lambda^*}(S^*).$$

Dividing by the left hand side and multiplying with α completes the proof for $\lambda \in [\lambda^*, \alpha\lambda^*]$.

Now suppose $\lambda \in [\frac{\lambda^*}{\alpha}, \lambda^*]$. This time we use the concavity and non-negativity of $\lambda \mapsto \min_{S \in X} c_\lambda(S)$:

$$\min_{S \in X} c_{\frac{\lambda^*}{\alpha}}(S) \geq (1 - \frac{1}{\alpha}) \min_{S \in X} c_0(S) + \frac{1}{\alpha} \min_{S \in X} c_{\lambda^*}(S) \geq \frac{1}{\alpha} \min_{S \in X} c_{\lambda^*}(S) = \frac{1}{\alpha}c_{\lambda^*}(S^*).$$

This yields

$$\alpha \geq \frac{c_{\lambda^*}(S^*)}{\min_{S \in X} c_{\frac{\lambda^*}{\alpha}}(S)} \geq \frac{c_\lambda(S^*)}{\min_{S \in X} c_\lambda(S)}$$

where the last step follows from the fact that numerator and denominator are nondecreasing in λ. □

Lemma 4.3 justifies the following algorithm, because it allows us to increase λ in exponentially growing steps, which yields the polynomial running time in the size of the input data:

Algorithm 3 Finding an Approximation System

Input: An instance satisfying Assumption 4.2
Output: An approximation system of ratio α for $(P(\lambda))$

1: Find initial solutions S_0 and S_∞ and real numbers $0 < q \leq Q < \infty$ with the property that for $\lambda \in [0, q]$:

$$\frac{c_\lambda(S_0)}{\min_{S \in X} c_\lambda(S)} \leq \alpha$$

and for $\lambda \in [Q, \infty[$:

$$\frac{c_\lambda(S_\infty)}{\min_{S \in X} c_\lambda(S)} \leq \alpha.$$

2: $j_{\max} := \lceil \frac{1}{2} \log_\alpha \frac{Q}{q} \rceil$.
3: **for** $j := 1$ **to** $j_{\max}$ **do**
4: $\quad \lambda_j := q\alpha^{2j-1}$.
5: $\quad$ Calculate S_j as an optimal solution to $(P(\lambda_j))$.
6: **end for**
7: **return** $R := \{S_0, S_1, \ldots, S_{j_{\max}}, S_\infty\}$.

We prove that the algorithm works as we wish:

Theorem 4.4. *Algorithm 3 returns an approximation system of ratio α of size $\mathcal{O}\left(\frac{\log Q/q}{\log \alpha}\right)$. If the corresponding nonparametric problem can be solved in $\mathcal{T}(n, m)$ steps, then the algorithm runs in $\mathcal{O}\left(\frac{\log Q/q}{\log \alpha}\mathcal{T}(n, m)\right)$ time, provided that finding the initial solutions in line 1 does not take longer.*

Proof. The algorithm clearly terminates and returns a subset R of X.

In order to show that R is an approximation system, let $\lambda \in \mathbb{R}_{\geq 0}$ be arbitrary. Since $j_{\max} \geq \frac{1}{2} \log_\alpha \frac{Q}{q}$, we have by monotonicity of the exponential function that $q\alpha^{2j_{\max}} \geq q\frac{Q}{q} = Q$. Hence, λ is either contained in one of the intervals $[0, q]$ and $[Q, \infty[$, or there is a $j \in \{1, 2, \ldots j_{\max}\}$ such that $\lambda \in [q\alpha^{2j-2}, q\alpha^{2j}]$. In the first case, one of the solutions S_0 and S_∞ is an approximate solution for $(P(\lambda))$ by the choice in line 1. In the second case, Lemma 4.3 ensures that S_j is an approximate solution for $(P(\lambda))$. Hence, R is in fact an approximation system.

The size of R follows by $j_{\max} \in \mathcal{O}(\log_\alpha Q/q) = \mathcal{O}\left(\frac{\log Q/q}{\log \alpha}\right)$ and the running time by the fact that apart from line 1, the only thing which requires more than a constant amount of steps is calculating the $j_{\max}$ solutions to the λ_j values. □

Theorem 4.4 shows that Algorithm 3 can be used to find a small approximation system, provided there is a proper way to settle the first step with constants q and Q which are not too far from each other. The next section deals with this question.

4.2. Finding Initial Solutions

In order to obtain useful results we have to strengthen our assumptions and consider again the bounded affine growth:

Assumption 4.5. *Suppose Assumption 2.1 holds. Moreover, the functions $c_\lambda(e)$ are given by $c_\lambda(e) = \min\{c(e) + \lambda d(e), u(e)\}$, where*

$$\begin{aligned} c(e) &\in \{0, 1, \dots, c_{\max}\}, \\ d(e) &\in \{0, 1, \dots, d_{\max}\}, \\ u(e) &\in \{0, 1, \dots, u_{\max}\} \cup \{\infty\}. \end{aligned}$$

Finally, $\alpha > 1$ is a real number.

We see that Assumption 4.5 implies Assumption 4.2. Additionally, recall that $K_{\max} = \max K = \max\{|S| \mid S \in X\}$ is the largest number of edges in a feasible set. We investigate how we can choose q and Q in Algorithm 3:

Lemma 4.6. *If Assumption 4.5 holds, then for any $\lambda \in \mathbb{R}_{\geq 0}$, any $\delta \in \mathbb{R}_{\geq 0}$ and any $S \in X$ it holds that*

$$c_{\lambda+\delta}(S) - c_\lambda(S) \in [0, \delta d_{\max} K_{\max}].$$

Proof. This follows since the slope of $\lambda \mapsto c_\lambda(S)$ is always nonnegative and smaller than $d_{\max} K_{\max}$. □

Lemma 4.7. *If the input of Algorithm 3 satisfies Assumption 4.5, then we can choose $q = \frac{\alpha}{d_{\max} K_{\max}}$ in line 1.*

Proof. Denote $q' = \frac{1}{2 d_{\max} K_{\max}}$. We choose S_0 to be an optimal solution to $(P(q'))$. First we prove that S_0 is optimal for $(P(0))$. By Lemma 4.6 and optimality of S_0 at q' we can conclude for any feasible $S \in X$:

$$\begin{aligned} c_0(S_0) &\leq c_{q'}(S_0) \\ &\leq c_{q'}(S) \\ &\leq c_0(S) + q' d_{\max} K_{\max} \\ &= c_0(S) + \frac{1}{2}. \end{aligned}$$

Since $c_0(S_0)$ and $c_0(S)$ are both integer, we even get $c_0(S_0) \leq c_0(S)$, which implies optimality of S_0 for $(P(0))$.

Now we show that S_0 is even optimal for all $\lambda \in [0, 2q']$. Again, let $S \in X$ be an arbitrary feasible solution. We distinguish into two cases:

Case 1: $c_0(S) = c_0(S_0)$. We explore for which $\lambda \in \mathbb{R}_{\geq 0}$ the function $\lambda \mapsto c_\lambda(S)$ may have a breakpoint. This can only happen, if for one of the edges $e \in S$ the function $\lambda \mapsto c_\lambda(e)$ has a breakpoint, i.e. if $d(e) > 0$ and $u(e) = c(e) + \lambda d(e)$, which is equivalent to $\lambda = \frac{u(e)-c(e)}{d(e)}$. Hence, by integrality the smallest possible positive breakpoint is at least $\frac{1}{d(e)} \geq 2p'$. From this it follows that both $c_\lambda(S_0)$ and $c_\lambda(S)$ are linear for $\lambda \in [0, 2q']$. Since $c_0(S) = c_0(S_0)$ and $c_{p'}(S) \geq c_{p'}(S_0)$ by optimality of S_0 at q', the linearity implies optimality of S_0 on the whole interval $[0, 2q']$.

Case 2: $c_0(S) > c_0(S_0)$. Then, by integrality, we even get $c_0(S) \geq c_0(S_0) + 1$ and hence for $\lambda \in [0, 2q']$ by Lemma 4.6:

$$\begin{aligned} c_\lambda(S_0) &\leq c_0(S_0) + \lambda d_{\max} K_{\max} \\ &\leq c_0(S) - 1 + \lambda d_{\max} K_{\max} \\ &\leq c_0(S) - 1 + 2q' d_{\max} K_{\max} \\ &= c_0(S) - 1 + 1 \\ &= c_0(S) \\ &\leq c_\lambda(S). \end{aligned}$$

Both cases together yield that S_0 is optimal for the whole interval $[0, 2q']$. Lemma 4.3 implies that S_0 is even an approximate solution for $[0, 2q'\alpha] = [0, q]$, which is what we wanted to prove. □

Lemma 4.8. *If the input of Algorithm 3 satisfies Assumption 4.5, then we can choose* $Q = \frac{K_{\max} \max\{c_{\max}, u_{\max}\}}{\alpha}$ *in line 1.*

Proof. Denote $Q' = K_{\max} \max\{c_{\max}, u_{\max}\} + 1$. We choose S_∞ to be an optimal solution to $(P(Q'))$.

First we show that there can not be any breakpoint of $\lambda \mapsto c_\lambda(e)$ for $\lambda \in]\alpha Q, \infty[$. In order to do so, observe that $\alpha Q = K_{\max} \max\{c_{\max}, u_{\max}\} \geq u_{\max}$. As seen before, a breakpoint can only be of the form $\lambda = \frac{u(e)-c(e)}{d(e)} \leq u_{\max}$ where the inequality holds because $d(e)$ is integer and hence at least one and because $c(e) \geq 0$. Plugging the first inequality into the second one yields that there are no breakpoints in $]\alpha Q, \infty[$. Hence, the cost functions are linear on $[\alpha Q, \infty[$. More precisely, they have the form

$$c_\lambda(e) = \begin{cases} u(e), & u(e) < \infty, d(e) > 0 \\ c(e) + \lambda d(e), & \text{else} \end{cases}$$

on this interval. This motivates to write

$$c'(e) = \begin{cases} u(e), & u(e) < \infty, d(e) > 0 \\ c(e), & \text{else} \end{cases}$$

and

$$d'(e) = \begin{cases} 0, & u(e) < \infty, d(e) > 0 \\ d(e), & \text{else} \end{cases}$$

in order to obtain $c_\lambda(e) = c'(e) + \lambda d'(e)$ for $\lambda \in [\alpha Q, \infty[$.

We prove that S_∞ is optimal to $(P(\lambda))$ for $\lambda \in [\alpha Q, \infty[$. In order to do so, let $S \in X$ be an arbitrary feasible solution. Since $Q' = \alpha Q + 1 \in [\alpha Q, \infty[$ and since S_∞ is optimal for $(P(Q'))$, we get

$$\begin{aligned} d'(S_\infty) &= \frac{c_{Q'}(S_\infty) - c'(S_\infty)}{Q'} \\ &\leq \frac{c_{Q'}(S) - c'(S_\infty)}{Q'} \\ &= d'(S) + \frac{c'(S) - c'(S_\infty)}{Q'} \\ &\leq d'(S) + \frac{K_{\max} \max\{c_{\max}, u_{\max}\} - 0}{Q'} \\ &< d'(S) + 1. \end{aligned}$$

Hence, by integrality, $d'(S_\infty) \leq d'(S)$. We distinguish into two cases:

Case 1: $d'(S_\infty) = d'(S)$. Then, by optimality of S_∞ for $(P(Q'))$, we get $c'(S_\infty) \leq c'(S)$ and hence $c_\lambda(S_\infty) \leq c_\lambda(S)$ for $\lambda \in [\alpha Q, \infty[$.

Case 2: $d'(S_\infty) \leq d'(S) - 1$. Then we get for $\lambda \in [\alpha Q, \infty[$:

$$\begin{aligned} c_\lambda(S_\infty) &= c'(S_\infty) + \lambda d'(S_\infty) \\ &\leq K_{\max} \max\{c_{\max}, u_{\max}\} + \lambda d'(S_\infty) \\ &= \alpha Q + \lambda d'(S_\infty) \\ &\leq \lambda(1 + d'(S_\infty)) \\ &\leq \lambda d'(S) \\ &\leq c_\lambda(S). \end{aligned}$$

Both cases together show the optimality of S_∞ for $\lambda \in [\alpha Q, \infty[$. Then, Lemma 4.3 shows that S_∞ is an approximate solution for $\lambda \in [Q, \infty[$. □

For the sake of simplicity we write $I_{\max} = \max\{c_{\max}, d_{\max}, u_{\max}, K_{\max}, 2\}$. Here, we added the 2 in order to avoid obtaining zero when taking the logarithm. Now we formulate the main theorem of this section:

Theorem 4.9. *Suppose the corresponding nonparametric problem to $(P(\lambda))$ can be solved in $\mathcal{T}(n, m)$ time. Then Algorithm 3 returns an approximation system of ratio α, which has a size of $\mathcal{O}\left(\frac{\log I_{\max}}{\log \alpha}\right)$, in $\mathcal{O}\left(\frac{\log I_{\max}}{\log \alpha} \mathcal{T}(n, m)\right)$ time. In particular, if $\mathcal{T}(n, m)$ is polynomial in n and m and α is constant, then this is polynomial in the size of the input data.*

Proof. Combining Theorem 4.4 and the two preceding Lemmas yields that the algorithm in fact returns an approximation system of size

$$\mathcal{O}\left(\frac{\log Q/q}{\log \alpha}\right) \subseteq \mathcal{O}\left(\frac{\log(d_{\max} K_{\max}^2 \max\{c_{\max}, u_{\max}\})}{\log \alpha}\right) = \mathcal{O}\left(\frac{\log I_{\max}}{\log \alpha}\right),$$

where the first subset relation follows because $Q/q \leq \alpha^2 Q/q = d_{\max} K_{\max}^2 \max\{c_{\max}, u_{\max}\}$ and the second equality by basic logarithmic identities. Since the initialization in line 1 basically consists of solving the nonparametric problem two more times, Theorem 4.4 implies that the whole algorithm runs in $\mathcal{O}\left(\frac{\log I_{\max}}{\log \alpha} \mathcal{T}(n,m)\right)$ time. □

Remark 4.10. The algorithm and its analysis also cover the unbounded cases by setting $u(e)$ to ∞ for all edges. Then we can just omit $u_{\max}$ in the running time and size of the system.

A. Appendix

We define a graph similar to [KN05]:

Definition A.1. A **directed graph** is a quadruple $G = (V, E, \alpha, \omega)$ such that V is a finite nonempty set of vertices which is disjoint to the finite set E of edges. $\alpha\colon E \to V$ and $\omega\colon E \to V$ are mappings which determine the start and end vertex of an edge.

Definition A.2. An **undirected graph** is a triple $G = (V, E, \gamma)$ such that V is a finite nonempty set of vertices which is disjoint to the finite set E of edges.

$$\gamma\colon E \to \{X \mid X \subseteq V \text{ with } 1 \leq |X| \leq 2\}$$

is a mapping which determines the end edges of an edge.

In this thesis we often handle the cases of directed and undirected graphs at once. For simplicity, we just write $G = (V, E)$, bearing in mind that this notation is just an abbreviation for the formal definitions given above. We also write $e = (u, v)$ for an edge between the vertices u and v, knowing that this is not a unique notation if the graph contains parallels. In the directed case this means $\alpha(e) = u$, $\omega(e) = v$, in the undirected case $\gamma(e) = \{u, v\}$. Finally, we write $n = |V|$ and $m = |E|$ throughout this thesis.

For formal definitions and elementary properties of paths and (spanning) trees refer to [KN05]. However, we identify a path with the set of edges contained in it as well as a spanning tree with the set of edges contained in the subgraph which forms the spanning tree. This makes our life easier when talking about a general problem formulation.

If we talk about shortest paths we always assume our graph to be directed. In the case of minimum spanning trees we work with undirected graphs.

Bibliography

[Car83] Patricia June Carstensen. *The Complexity of Some Problems in Parametric Linear and Combinatorial Programming.* PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1983.

[CFLY10] Sourav Chakraborty, Eldar Fischer, Oded Lachish, and Raphael Yuster. Two-phase algorithms for the parametric shortest path problem. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science*, pages 167–178, 2010.

[KN05] Sven O. Krumke and Hartmut Noltemeier. *Graphentheorische Konzepte und Algorithmen.* B.G. Teubner, 1 edition, 2005.

[KO81] Richard M. Karp and James B. Orlin. Parametric Shortest Path Algorithms with an Application to Cyclic Staffing. *Discrete Applied Mathematics*, 3:37–45, 1981.

[YTO91] Neal E. Young, Robert E. Tarjan, and James B. Orlin. Faster Parametric Shortest Path and Minimum-Balance Algorithms. *Networks*, 21:205–221, 1991.