On Flows, Paths, Roots, and Zeros

A dissertation submitted towards the degree
Doctor of Natural Science

of the

Faculty of Mathematics and Computer Science of Saarland University

by Ruben Becker

Saarbrücken, Germany, 2017

Colloquium Details

Day of Colloquium: July 23, 2018

Dean of the Faculty: Prof. Dr. Sebastian Hack

Chair of the Committee: Prof. Dr. Gerhard Weikum

Examiners:

Dr. Andreas Karrenbauer
Prof. Dr. Michael Sagraloff
Prof. Dr. Dr. h.c. mult. Kurt Mehlhorn
Fabrice Rouillier, HDR

Academic Assistant: Dr. Antonios Antoniadis

Abstract

This thesis has two parts; in the first of which we give new results for various network **flow** problems. (1) We present a novel dual ascent algorithm for min-cost flow and show that an implementation of it is very efficient on certain instance classes. (2) We approach the problem of numerical stability of interior point network flow algorithms by giving a path following method that works with integer arithmetic solely and is thus guaranteed to be free of any numerical instabilities. (3) We present a gradient descent approach for the undirected transshipment problem and its special case, the single source shortest **path** problem (SSSP). For distributed computation models this yields the first SSSP-algorithm with near-optimal number of communication rounds.

The second part deals with fundamental topics from algebraic computation. (1) We give an algorithm for computing the complex **roots** of a complex polynomial. While achieving a comparable bit complexity as previous best results, our algorithm is simple and promising to be of practical impact. It uses a test for counting the roots of a polynomial in a region that is based on Pellet's theorem. (2) We extend this test to polynomial systems, i.e., we develop an algorithm that can certify the existence of a k-fold **zero** of a zero-dimensional polynomial system within a given region. For bivariate systems, we show experimentally that this approach yields significant improvements when used as inclusion predicate in an elimination method.

Kurzfassung

Im ersten Teil dieser Dissertation präsentieren wir neue Resultate für verschiedene Netzwerkflussprobleme. (1) Wir geben eine neue Duale-Aufstiegsmethode für das Min-Cost-Flow-Problem an und zeigen, dass eine Implementierung dieser Methode sehr effizient auf gewissen Instanzklassen ist. (2) Wir behandeln numerische Stabilität von Innere-Punkte-Methoden für Netwerkflüsse, indem wir eine solche Methode angeben die mit ganzzahliger Arithmetik arbeitet und daher garantiert frei von numerischen Instabilitäten ist. (3) Wir präsentieren ein Gradienten-Abstiegsverfahren für das ungerichtete Transshipment-Problem, und seinen Spezialfall, das Single-Source-Shortest-Problem (SSSP), die für SSSP in verteilten Rechenmodellen die erste mit nahe-optimaler Anzahl von Kommunikationsrunden ist.

Der zweite Teil handelt von fundamentalen Problemen der Computeralgebra. (1) Wir geben einen Algorithmus zum Berechnen der komplexen Nullstellen eines komplexen Polynoms an, der eine vergleichbare Bitkomplexität zu vorherigen besten Resultaten hat, aber vergleichsweise einfach und daher vielversprechend für die Praxis ist. (2) Wir erweitern den darin verwendeten Pellet-Test zum Zählen der Nullstellen eines Polynoms auf Polynomsysteme, sodass wir die Existenz einer k-fachen Nullstelle eines Systems in einer gegebenen Region zertifizieren können. Für bivariate Systeme zeigen wir experimentell, dass eine Integration dieses Ansatzes in eine Eliminationsmethode zu einer signifikanten Verbesserung führt.

Acknowledgements

I would like to thank Andreas Karrenbauer for advising me and guiding me in exploring the research topic of network flows. I am happy that I was a part of his discrete optimization group and I enjoyed the prevailing atmosphere.

I want to thank Michael Sagraloff for his guidance. I was very fortunate to work with him on research topics that he was already exploring for years. His patience and his devotedness to specific questions both impressed and motivated me.

I furthermore want to thank Kurt Mehlhorn both for having been an amazing scientific role model and for being "the wise old man" when requested. During all my years at the Max Planck Institute for Informatics, I knew that I could always approach him with my concerns.

Lastly, I want to thank all the other people with whom I have collaborated on the projects that this thesis is based on. Here, I especially want to mention Christoph Lenzen.

Preface

This thesis has two parts. Part I deals with a very powerful class of optimization problems, the so-called network flow problems. We give new algorithms for several different variants of network flow problems (in different models of computation). This part is based on the following three papers: [BFK16; BKM16; Bec+17].

[BFK16]	Ruben Becker, Maximilian Fickert, and Andreas Karrenbauer. "A Novel Dual Ascent Algorithm for Solving the Min-Cost Flow Problem". In: *Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2016, Arlington, Virginia, USA, January 10, 2016*. Ed. by Michael T. Goodrich and Michael Mitzenmacher. SIAM, 2016, pp. 151–159.
[BKM16]	Ruben Becker, Andreas Karrenbauer, and Kurt Mehlhorn. "An Integer Interior Point Method for Min-Cost Flow Using Arc Contractions and Deletions". In: *CoRR* abs/1612.04689 (2016). Abstract and Talk at International Network Optimization Conference 2017 (INOC).
[Bec+17]	Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger, and Christoph Lenzen. "Near-Optimal Approximate Shortest Paths and Transshipment in Distributed and Streaming Models". In: *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*. Ed. by Andréa W. Richa. Vol. 91. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, 7:1–7:16.

Part II is concerned with topics from symbolic-numeric computation. We present new algorithms for the computation of the complex roots of a univariate complex polynomial and for counting the number of zero-dimensional polynomial systems within a given region. The second part is based on the following two manuscripts: [Bec+18; BS17].

[Bec+18]	Ruben Becker, Michael Sagraloff, Vikram Sharma, and Chee Yap. "A near-optimal subdivision algorithm for complex root isolation based on the Pellet test and Newton iteration". In: *J. Symb. Comput.* 86 (2018), pp. 51–96.
[BS17]	Ruben Becker and Michael Sagraloff. "Counting Solutions of a Polynomial System Locally and Exactly". In: *CoRR* abs/1712.05487 (2017).

Besides the above publications, during my PhD, that I have started in October 2013, I published the following articles: [BK14; Bec+16; Wim+17].

[BK14]	Ruben Becker and Andreas Karrenbauer. "A Simple Efficient Interior Point Method for Min-Cost Flow". In: *Algorithms and Computation - 25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15-17, 2014, Proceedings*. Ed. by Hee-Kap Ahn and Chan-Su Shin. Vol. 8889. Lecture Notes in Computer Science. Springer, 2014, pp. 753–765.
[Bec+16]	Ruben Becker, Michael Sagraloff, Vikram Sharma, Juan Xu, and Chee Yap. "Complexity Analysis of Root Clustering for a Complex Polynomial". In: *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2016, Waterloo, ON, Canada, July 19-22, 2016*. Ed. by Sergei A. Abramov, Eugene V. Zima, and Xiao-Shan Gao. ACM, 2016, pp. 71–78.
[Wim+17]	Ralf Wimmer, Andreas Karrenbauer, Ruben Becker, Christoph Scholl, and Bernd Becker. "From DQBF to QBF by Dependency Elimination". In: *SAT*. Vol. 10491. Lecture Notes in Computer Science. Springer, 2017, pp. 326–343.

Note that [BK14] contains the results of my master's thesis.

Contents

Part I

Flows and Paths

Chapter 1

Introduction to Part I

Networks appear literally everywhere in today's world. Whenever a discrete set of entities get into contact with each other, a network is inherent. If their goal is to transport some sort of good through the network, a network optimization problem appears. The first part of this thesis deals with a very fundamental and classical class of network optimization problems, namely network flow problems. The work of mathematicians on this class of problems goes back to the first half of the previous century [AMO93]. In fact, the so-called transportation problem, a special case of the min-cost flow problem that we will consider later on in this thesis, was already studied independently by mathematicians on different continents [Hit41; Koo51; Kan60] around 1950. Kantorovich and Koopmans later received the Nobel Prize in Economic Sciences for their work on this problem. The more general minimum-cost flow problem was first studied algorithmically by Dantzig [Dan63] and Ford and Fulkerson [FF62] around 1960. Their work can be considered as the basis of what we consider the classic literature on network flow problems today. Besides this, it had a great influence on the evolution of the whole field of linear optimization [AMO93]. In addition to this great impact on mathematics, economics, and computer science, network flow problems have always been of great interest also because they offer such a vast set of applications in many different fields [Ahu+95]. Besides applications in other fields of computer science and mathematics (see, e.g., [Tam87; II86]), network flows have applications in physics (see, e.g., [Rie98]), in telecommunications (see, e.g., [BOR80]), in biology (see, e.g., [Wat89]), in image processing (see, e.g., [SG82; Cos98]), and many more. In fact, many real-world problems that admit polynomial-time algorithms in the first place can be turned into network flow problems [Ahu+95].

Naturally, many different variants of the abstract description of transporting a good through a given network are conceivable. In what follows, we describe the most important ones in the context of this thesis. The question of how to get from one point in a network to another point in the shortest manner, i.e., the famous *shortest path problem*, is the simplest version of a network flow problem. Here, shortest may just refer to the number of links or, in the weighted version, lengths are associated to the links in the network and the path of shortest length is called for. Shortest path problems are of huge importance in logistics and routing and by now many of us use algorithms to compute shortest paths on a daily basis – namely, whenever we use a route planner like Google Maps or any other navigation system. This problem has received a lot of attention both from theoreticians and practitioners [Dij59; Tho99; Bas+07; DGJ09]. From a theoretical point of view, the problem can be considered as solved in the standard RAM model of computation. However, as nowadays computer systems are more and more distributed and data becomes larger and larger, it becomes necessary to consider different, more realistic models of computation such as distributed and streaming models. In this thesis, we will present a result in this scope for the shortest path problem. Another variant of network flow problems that has also received plenty of attention by researchers is the *max-flow problem* and its dual the min-cut problem [Kar73; ET75; Kar00; KT15; Mąd13; HRW17]. In the max-flow problem the question is how to send a maximum amount of flow from one point in the graph to another, while respecting capacity constraints on the links.

The already mentioned min-cost flow problem generalizes both the shortest path and the max-flow problem. The problem is described as follows: Given a network with costs and capacities on the links, the goal is to send a good in the cheapest possible way from some set of sources, i.e., nodes having surplus, to some set of sinks, i.e., nodes having deficit, while respecting the capacity constraints on the edges. This problem is known as the *capacitated min-cost flow problem* or *min-cost flow problem* in the case where no capacities are given. We will see that there is a very close relation between the capacitated and the non-capacitated version of the problem, see Section 1.3. Each of the described problems can be thought of in the setting of a directed graph, where flow is only allowed to be sent in one direction over the edges or in undirected graphs, where flow can travel in both directions. In the setting of non-capacitated undirected graphs, the min-cost flow problem is also referred to as the *undirected transshipment problem*.

We proceed with more history on network flow problems before giving an overview of the main results of this part of the thesis.

1.1 Related Work

The literature on network flows is extremely vast. A good overview can be found in standard books on the topic [AMO93; Sch03]. We will focus on the most relevant contributions for the topics of this thesis. The algorithmic work on the problem starts with the seminal works of Dantzig [Dan63] and Ford and Fulkerson [FF62]. Dantzig, in his book on linear programming, focuses on the simplex method and its interpretation for network flow problems. Ford and Fulkerson's work can be understood as the starting point of an extremely fruitful line of research: the primal-dual combinatorial algorithms. Edmonds and Karp [EK01] were the first to give a weakly-polynomial time algorithm for the min-cost flow problem, i.e., an algorithm with a run-time that is polynomial in the size of the graph, as well as in the encoding length of the numbers in the input representing costs, capacities, and demands. The first strongly-polynomial time algorithm, i.e., an algorithm with polynomial time complexity in the number of nodes and edges, but independent of the size of the numbers in the input, was given by Tardos [Tar85]. There have been many further contributions since then. The strongly-polynomial time algorithms peak in the algorithm of Orlin [Orl88], which refines the algorithm of Edmonds and Karp. It solves the uncapacitated min-cost flow problem by computing a sequence of $O(n \log(n))$ many shortest path problems, which leads to a run-time of $O(n \log(n)(m + n \log n))$, where n is the number of nodes and m the number of arcs. Using a standard reduction that we will review in Section 1.3, a run-time of $O(m \log(n)(m + n \log n))$ can be achieved for the capacitated min-cost flow problem. In terms of weakly-polynomial time algorithms, the scaling algorithms [GT90] finally lead to the bound of $O(nm \log \log U \log(nC))$ that is due to Ahuja et al. [Ahu+92]. Here C denotes an upper bound on the cost and U an upper bound on the capacities and demands.

More recently, a different approach has been very fruitful. It dates back to the work of Vaidya [Vai89] who was the first to give a tailored interior point method for network flow problems. Although his algorithm did not improve over the best combinatorial methods at that time, the general approach of using continuous optimization techniques for network flow problems has become extremely successful in the last decade. The first very prominent result goes back to Daitch and Spielman [DS08]. They gave a randomized $\tilde{O}(m^{3/2} \log^2 U)$-time algorithm[1] that heavily relies on nearly-linear time equation solvers [ST04; KMP14; Kel+13]. This general approach of using nearly-linear time equation solvers as a subroutine

[1]We use the $\tilde{O}(\cdot)$-notation to hide logarithmic factors, i.e., $\tilde{O}(f) := O(f \log^c(f))$ for any constant $c \geq 0$.

has proven successful also for various other optimization problems. Daitch and Spielman's method was the first to achieve a sub-quadratic running time in sparse graphs (assuming that the costs, capacities, and demands are polynomially bounded in n). The same running time (up to logarithmic factors) was achieved in [BK14] for the classical min-cost flow problem with a simple potential reduction algorithm. Even more recently, the breakthrough result of Lee and Sidford [LS14] yields a very intricate randomized interior point method of complexity $\tilde{O}(m\sqrt{n}\log^{O(1)} U)$ for the min-cost flow problem. Most recently, Cohen et al. developed a new framework to analyze interior point methods [Coh+17]. They obtain a bound of $\tilde{O}(m^{10/7}\log C)$ for the special case of min-cost flows with unit capacities. The very same bound has been achieved before by Mądry for the max-flow problem in directed graphs with unit capacities [Mąd13].

For network flow problems in undirected graphs, the picture is slightly different. On the one side, it is rather easy to see, confer Section 1.3, that the undirected setting can be reduced to the directed one by introducing two directed arcs for every undirected edge and thus all the previously mentioned asymptotic bounds also apply for the undirected case. However, the current state of research is that the undirected case actually allows for better results in some settings. Namely, when allowing for some approximation, i.e., searching for a solution that is within a factor of $1 + \varepsilon$ of the optimum, both the undirected max-flow as well as the undirected transshipment problem can be solved in close to linear time. For the max-flow problem the first results in this scope were simultaneously given by Sherman [She13] and Kelner et al. [Kel+14]. Peng improved their run-time from almost linear in m, i.e., $O(m^{1+o(1)})$, to nearly-linear in m, i.e., $O(m \operatorname{polylog}(n)) = \tilde{O}(m)$. For the undirected transshipment problem without capacity constraints a comparable result, namely an almost-linear time $(1 + \varepsilon)$-approximation algorithm is due to Sherman [She17].

These advances in theoretical run-time bounds, however, did not yet translate into faster implementations in practice. In fact, the fastest implementations for solving network flow instances in practice are still based on the classical combinatorial approaches, see for example [GK91; Gol97] or [KK12] for a comparison study. It is one of the ambitions of this thesis to help narrowing this gap.

1.2 Main Results of Part I

Main Result 1.1. Chapter 2 introduces a fully-combinatorial dual ascent algorithm for the capacitated min-cost flow problem. We formally prove the correctness of the approach and show further properties. After presenting a greedy variant of the algorithm's dual step, we report on an implementation of the algorithm and several variants of it. We evaluate these variants experimentally and show that our approach is competitive with recent third-party implementations of well-known algorithms for the min-cost flow problem and even outperforms them on certain realistic instance classes such as road networks.

Main Result 1.2. Although the current record bounds for the min-cost flow problem are achieved by interior point methods, the combinatorial algorithms are still the methods of choice in practice. In fact, we are not aware of any interior point method that has been reported superior to combinatorial algorithms for network flow problems in practice. The striking efficiency of the combinatorial algorithms is partly due to their straightforward arithmetic. Interior point methods, however, are formulated for real arithmetic and bridging the gap between real and finite-precision arithmetic requires tedious error analysis. In Chapter 3, we approach this problem by presenting an interior point method for the min-cost flow problem that uses integer arithmetic only. The use of arc contractions and deletions avoids having to deal with huge and small numbers simultaneously, and an initial scaling guarantees that data stays integral as the algorithm visits only *integer* lattice points in the vicinity of the central path of the primal-dual polytope. We provide explicit bounds on the size of all numbers appearing through the computation. We moreover show that our algorithm runs in time $\tilde{O}(m^{3/2})$ time with high probability and thus, in terms of time complexity, dominates over the classical combinatorial methods. By eliminating one of the drawbacks of interior point methods for network flow problems, we hope to narrow the gap between practically and theoretically efficient algorithms for the min-cost flow problem.

Main Result 1.3. In Chapter 4, we turn to undirected flow problems and make use of one big opportunity that continuous optimization techniques have to offer for becoming relevant in practice, namely concurrency. That is, in addition to the standard RAM model, we also approach the topic from the perspective of distributed and streaming models of computation, more precisely the Broadcast Congest model, the Broadcast Congested Clique model, as well as the Multipass Streaming model. We present a tailored gradient descent technique for computing a solution to the undirected transshipment problem that is optimal up to a $(1 + \varepsilon)$-approximation factor. Besides an interesting result in the RAM model that improves over the work of Sherman [She17] in certain settings, we obtain the first non-trivial results for undirected transshipment in the aforementioned distributed models of computation. Maybe even more interestingly, when applied to the special case of single source shortest path, our method is the first to achieve a near-optimal number of communication rounds in the Broadcast Congest and Broadcast Congested Clique models. We would like to note that the previous best results for distributed single source shortest path follow the framework of sparse hop sets and that the recent lower bounds by Abboud et al. [ABP17] had shown that it is impossible to achieve near-optimality when using approaches based on this framework.

We continue by introducing some basic notation and then proceed with formally defining the described network flow problems.

1.3 Preliminaries

1.3.1 Basics

In what follows, we define $[n] := \{1, \dots, n\}$ for any natural number $n \in \mathbb{N}_{\geq 1}$. For any set S and integer $k \leq |S|$, we denote with $\binom{S}{k}$ the set of all subsets of S that have size k. For a vector $x \in \mathbb{R}^n$ and a subset $S \subseteq [n]$ of the indices, we denote $x(S) := \sum_{i \in S} x_i$. With $\mathbb{1}$ we denote the vector of suitable dimension with all entries being one, while for any index i, the vector $\mathbb{1}_i$ is the vector that is 1 at position i and zero everywhere else. For any vector $x \in \mathbb{R}^d$ by $\|x\|_1 = \sum_{i=1}^d |x_i|$ we denote its 1-norm, by $\|x\|_2 = (\sum_{i=1}^d x_i^2)^{1/2}$ we denote its 2-norm, and by $\|x\|_\infty = \max\{|x_i| : i \in [d]\}$ its ∞-norm. We will also use $\|x\|_W = (\sum_{i=1}^d w_i x_i^2)^{1/2}$ for the weighted 2-norm with a square and positive weight matrix $W \in \mathbb{R}^{n \times n}_{>1}$. Moreover, we use $(x)_{\max} := \max\{x_i : i \in [d]\}$ to denote the maximum entry in the vector x. We will use $\log(\cdot)$ to denote the logarithm with base 2 and $\ln(\cdot)$ to denote the natural logarithm. We write $\operatorname{poly} f$ and $\operatorname{polylog} f$ for $O(f^c)$ and $O(\log^c(f))$ for some constant c, respectively. As stated in a footnote before, we use the $\tilde{O}(\cdot)$-notation to hide poly-logarithmic factors, i.e., $\tilde{O}(f) := O(f \operatorname{polylog} f)$.

1.3.2 Graphs

We will deal with *undirected* as well as with *directed graphs*. An *undirected graph* G consist of two sets (V, E): a set of nodes V and a set of edges E. We will, w.l.o.g., assume that $V = [n]$. The set of edges E is a set of unordered sets of size two, i.e., $E \subseteq \binom{V}{2}$. A *directed graph* consists of a set of nodes V and a set of arcs A, where $A \subseteq V^2$ is a set of ordered pairs. Whenever an undirected or directed graph G is given, n denotes the number of nodes and m the number of edges or arcs of that graph.

For an undirected graph G, we denote with $\delta_G(v) := \{e \in E : v \in e\}$, the set of incident edges to v in G. For a directed graph, we distinguish between the set of *ingoing* arcs $\delta_G^{\text{in}}(v) := \{a \in A : \text{ there is } u \in V \text{ with } a = (u,v)\}$ and the set of *outgoing* arcs $\delta_G^{\text{out}}(v) := \{a \in A : \text{ there is } w \in V \text{ with } a = (v,w)\}$. In the directed case, we let $\delta_G(v) := \delta_G^{\text{in}}(v) \cup \delta_G^{\text{out}}(v)$.

Given a directed graph $G = (V, A)$, after fixing some order $\sigma : [m] \to A$ of the arcs, we associate the so-called *node-arc incidence matrix* $A \in \{-1, 0, 1\}^{n \times m}$ with G. It is defined as

$$A_{v,j} = \begin{cases} 1, & \text{if } \sigma(j) \in \delta_G^{\text{in}}(v) \\ -1, & \text{if } \sigma(j) \in \delta_G^{\text{out}}(v) \\ 0 & \text{otherwise.} \end{cases}$$

Instead of writing $A_{v,j}$ for the entry corresponding to v and j, we just write $A_{v,a}$ with $a = \sigma(j)$ from now on, i.e., we will assume the order implicitly. Note that the matrix $-A$ corresponds to a graph on the same set of nodes and with the same number of arcs, but the direction of each arc is flipped. Thus, we will use $-A = \{-a = (w,v) : a = (v,w) \in A\}$ for the set of reversed arcs analogously.

For an undirected graph $G = (V, E)$, we frequently fix an orientation $E^{\to} \subseteq V^2$ of the edges E. We then call $e \in E^{\to}$ a forward edge and $-e$ a backward edge. After fixing the orientation $E^{\to}$ of E, we can associate a directed graph $G^{\leftrightarrow} = (V, A) = (V, E^{\to} \cup -E^{\to})$ consisting of the forward and backward edges with G. We call $G^{\leftrightarrow}$ the bidirected graph corresponding to G. With a slight abuse of notation, we write $\delta_G^{\text{in}}(v)$ instead of $\delta_{G^{\leftrightarrow}}^{\text{in}}(v)$ and $\delta_G^{\text{out}}(v)$ instead of $\delta_{G^{\leftrightarrow}}^{\text{out}}(v)$ for a node $v \in V$. To an undirected graph G we associate the node-arc incidence matrix $A \in \{-1, 0, 1\}^{n \times 2m}$ corresponding to its bidirected graph $G^{\leftrightarrow}$. If the graph G is clear from the context, we will sometimes entirely omit the subscript and just write $\delta^{\text{in}}(v)$ and $\delta^{\text{out}}(v)$.

1.3.3 Network Flow Problems

Undirected network flow problems. As described above, there are many different variants of network flow problems, both in directed as well as in undirected graphs. One of the most fundamental ones in undirected graphs is the **undirected transshipment problem**. It is defined by a tuple (G, b, w), where $G = (V, E)$ is an undirected graph, $b \in \mathbb{Z}^n$ is a demand vector with one entry per node that satisfies $\mathbb{1}^T b = 0$, and $w \in \mathbb{N}_{\geq 1}^m$ is a weight or cost vector with one entry per edge. Let us fix an arbitrary orientation $E^{\to}$ of the edge set E. The task is to find a *flow* vector $x \in \mathbb{R}_{\geq 0}^{2m}$, i.e., x has an entry x_a for every $a \in A = E^{\to} \cup -E^{\to}$, that is *feasible* and of *minimum cost*. We call a flow x feasible, if it fulfills flow conservation, i.e., $x(\delta_G^{\text{in}}(v)) - x(\delta_G^{\text{out}}(v)) = b_v$ holds at every node $v \in V$, or equivalently, $Ax = b$ in vector-matrix notation. The cost of a flow x is defined as $\mathbb{1}^T W x = \sum_{a \in A} w_a x_a$, where $W = \operatorname{diag}(w, w) \in \mathbb{N}_{\geq 1}^{2m}$ is the diagonal matrix given by the weights w and we define $w_{-e} = w_e$ for any $e \in E^{\to}$. The undirected transshipment problem can then be written as the following minimization problem with non-negativity and linear equality constraints:

$$\min\{\mathbb{1}^T W x : Ax = b, x \geq 0\}. \tag{1.1}$$

We remark that every optimal solution x^* of the above problem satisfies $x_e^* \cdot x_{-e}^* = 0$ for $e \in E^{\to}$ as $w > 0$. Assume that this is not the case for some e, then $x^* - \varepsilon(\mathbb{1}_e + \mathbb{1}_{-e})$ is feasible for a sufficiently small $\varepsilon > 0$ and has a smaller objective value.

The **asymmetric transshipment problem** is defined exactly as the undirected transshipment problem with the sole difference being that the weight w_e^+ for a forward edge $e \in E^{\to}$

can be different from the weight w^-_e for the corresponding backward edge. That is, the edges have different cost depending on the direction that the flow travels along them. W.l.o.g., we assume that $w^+ \geq w^-$, i.e., that we picked the $E^{\rightarrow}$ such that the forward weight is at least the backward weight. The asymmetric transshipment problem can then be written as before with the definition $W = \operatorname{diag}(w^+, w^-) \in \mathbb{N}^{2m}_{\geq 1}$. Clearly, if $w^+ = w^-$, then the asymmetric transshipment problem reduces to the symmetric or standard undirected transshipment problem.

Note that the **undirected single-source-shortest path problem**, where we want to compute the shortest path from one source node to all other nodes, is the special case of the undirected transshipment problem with $b = \mathbb{1} - n\mathbb{1}_s$. Also for the shortest path problem, we may allow the weights to be asymmetric.

Directed network flow problems. We now turn to network flow problems in directed graphs. The **min-cost flow problem** is defined by a tuple (G, b, c), where $G = (V, A)$ is a directed graph, $b \in \mathbb{Z}^n$ as before is a demand vector with one entry per node that satisfies $\mathbb{1}^T b = 0$, and $c \in \mathbb{N}^m_{\geq 1}$ is a cost vector with an entry for every arc. The task in the min-cost flow problem is to find a *flow* $x \in \mathbb{R}^m_{\geq 0}$ that is feasible and of minimum cost. Note the difference to the undirected problem above: an arc $a = (v, w) \in A$ only permits flow to travel in one direction from v to w, while an edge $e = \{v, w\}$ results in two arcs (v, w) and (w, v) in the bidirected graph and thus permits flow to travel in both directions.[2] Again, we call the flow x feasible, if it fulfills flow conservation, i.e., in vector-matrix notation, if $Ax = b$ is satisfied. The weight of a flow $x \geq 0$ is defined as $c^T x$. Clearly, the min-cost flow problem can be written as the following linear program:

$$\min\{c^T x : Ax = b, x \geq 0\}. \tag{1.2}$$

In a variant of the problem, the so-called **capacitated min-cost flow problem**, in addition a capacity vector $u \in \mathbb{N}^m_{\geq 1}$ is given, i.e., an instance is described by a tuple (G, b, c, u). In this problem, there is the additional constraints that $x \leq u$, i.e., in form of a linear program the problem writes as

$$\min\{c^T x : Ax = b, 0 \leq x \leq u\}. \tag{1.3}$$

The directed single-source-shortest path problem is a special case of the min-cost flow problem. We sometimes refer to the standard min-cost flow problem as the *uncapacitated* min-cost flow problem if we want to emphasize that there are no capacity constraints.

1.3.4 Relations Between the Problems

We proceed by giving relations between the above introduced problems. It is clear that the undirected and asymmetric transshipment problems are actually special cases of the min-cost flow problem with cost vector $c = \left[\begin{smallmatrix} w \\ w \end{smallmatrix}\right]$ and $c = \left[\begin{smallmatrix} w^+ \\ w^- \end{smallmatrix}\right]$, respectively:

Lemma 1.1. *The asymmetric and undirected transshipment problem on a graph with n nodes and m edges can be solved by solving a min-cost flow problem on a graph with n nodes and $2m$ arcs.*

Now let (G, b, c, u) be an instance of the capacitated min-cost flow problem. There is a straightforward, well-known reduction to the min-cost flow problem. Recall the LP-formulation of the capacitated min-cost flow problem from (1.3) and consider the LP resulting

[2]We remark that some authors use the term directed transshipment problem for what we define as the min-cost flow problem. In this thesis, we will reserve the term transshipment for the undirected case in order to have a clearer differentiation between the directed and undirected variant of the problem.

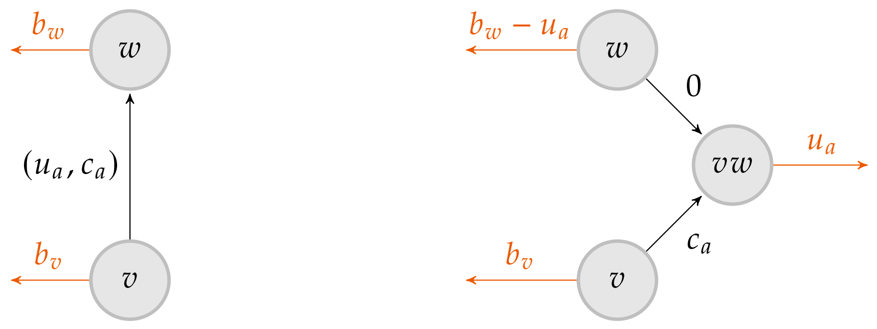

Figure 1.1: The transformation that, when done for every arc $a = (v, w)$, leads to a min-cost flow problem from a capacitated min-cost flow problem.

from introducing slack variables s for the capacity constraints $x \leq u$:

$$\min \left\{ \begin{bmatrix} c \\ 0 \end{bmatrix}^T \begin{bmatrix} x \\ s \end{bmatrix} : \begin{bmatrix} A & 0 \\ I & I \end{bmatrix} \begin{bmatrix} x \\ s \end{bmatrix} = \begin{bmatrix} b \\ u \end{bmatrix}, x, s \geq 0 \right\}.$$

We will now argue how to transform this LP into the form of a min-cost flow problem again. For this, we need to achieve the form of a node-arc incidence matrix for the constraint matrix $B = \left[\begin{smallmatrix} A & 0 \\ I & I \end{smallmatrix}\right] \in \{-1, 0, 1\}^{(n+m)\times 2m}$. Note that every column B_a for $a = (v, w) \in A$ in the first half of B has three non-zero entries (two +1s and one −1) and every column in the second half has only one non-zero entry. In order to obtain a node-arc incidence matrix from B, we subtract the a'th row of $[\begin{smallmatrix} I & I \end{smallmatrix}]$ from the w'th row of $[\begin{smallmatrix} A & 0 \end{smallmatrix}]$. In other words the matrix $A' := \left[\begin{smallmatrix} A & 0 \\ I & I \end{smallmatrix}\right] - \sum_{a=(v,w)\in A} \left[\begin{smallmatrix} e_w \\ 0 \end{smallmatrix}\right] \left[\begin{smallmatrix} e_a^T & e_a^T \end{smallmatrix}\right]$ is again a node-arc incidence matrix. Doing the equivalent transform for the right-hand side vector $\left[\begin{smallmatrix} b \\ u \end{smallmatrix}\right]$ leads to the vector $b' := \left[\begin{smallmatrix} b \\ u \end{smallmatrix}\right] - \sum_{a=(v,w)\in A} \left[\begin{smallmatrix} e_w \\ 0 \end{smallmatrix}\right] u_a$ as new demand vector. Together with $c' = \left[\begin{smallmatrix} c \\ 0 \end{smallmatrix}\right]$, we obtain the problem

$$\min\{c'^T x : A'x = b', x \geq 0\},$$

which corresponds to an *uncapacitated* min-cost flow problem as in (1.2) with $m + n$ nodes and $2m$ arcs. The applied transformation is equivalent to doing the graph transformation from Figure 1.1 for every arc in the graph. We remark that this is a well-known construction, see for example [AMO93, Section 2.4]. We obtain the following lemma.

Lemma 1.2. *The capacitated min-cost flow problem on a graph with n nodes and m arcs can be solved by solving an uncapacitated min-cost flow problem on a graph with $n + m$ nodes and $2m$ arcs.*

Note that depending on the application or the algorithm used, this cost of introducing a node for every arc might be undesirable, e.g., assume the graph to be dense, i.e., $m = \Theta(n^2)$ and the algorithm's run-time, in terms of n and m, to be $\tilde{O}(\sqrt{n}m)$. In this case, the construction would lead to a run-time of $\tilde{O}(n^3)$ instead of $\tilde{O}(n^{5/2})$. Avoiding this blow-up is one of the contributions of Lee and Sidford in [LS14].

Chapter 2

A Dual Ascent Algorithm for Capacitated Min-Cost Flow

2.1 Introduction

In this chapter, we present an algorithm for the capacitated min-cost flow problem. This algorithm is inspired by the crossover technique described in [BK14] in the context of an interior point methods for min-cost flow. In [BK14], the crossover technique was used in order to round a given fractional dual solution to an optimum integral solution, provided that the fractional solution is sufficiently close to the optimum solution. In the work that we report on in this chapter, we notice that even if the given dual solution is far from being optimal, this technique returns an integer solution that is not worse (and in many cases much better) than the given one. This observation leads to the idea of using the crossover procedure as a subroutine in a min-cost flow algorithm that iteratively improves a dual feasible solution until it becomes optimal. In order to certify optimality of the dual solution, the algorithm maintains complementary slackness of the dual solution together with some primal variables. The primal variables always satisfy the property that they are a pseudo-flow , i.e., a vector x with $0 \leq x \leq u$ that satisfies the capacity constraints but may violate flow conservation. As complementary slackness is maintained as an invariant, it holds that as soon as x becomes feasible, i.e., satisfies flow conservation, we actually have optimum primal and dual solutions. This abstract procedure is not different from other dual algorithms such as *successive shortest path*, but the novelty is the way we perform the dual update; namely, we replace the shortest path routine by the crossover technique. In fact, our method can be considered a generalization of the computation of a shortest path tree. While the shortest path tree is directed either towards or away from the root, we grow an undirected tree, i.e., with some arcs pointing towards and some away from the root. To this end, we also take the residual demand of the current pseudo-flow into account to decide whether to extend the current tree and the subgraph it spans by an ingoing or an outgoing arc. We will offer a precise description in Section 2.2. We remark that this dual update is not only different from the one in the successive shortest path method [Tom71; EK70], but also substantially different from the primal-dual method in [Has83] and the dual heuristic improvements in [Gol97] – as we take into account the sign of the deficit of the nodes during the dual step, our primal steps are closely tied to the tree resulting from the dual step, and we improve node potentials along $n - 1$ cuts in each iteration.

Although this approach does not achieve a record complexity bound, it is interesting from a practical point of view, as we will see that its implementation turns out to be very efficient on certain instance classes.

Structure of the Remainder of this Chapter. The rest of this chapter consists of three sections. In Section 2.2, we first introduce the dual ascent algorithm, prove that it is correct and fulfills certain properties. We then present a greedy variant of the algorithm. In

Section 2.3, we describe some implementation choices and details. In Section 2.4, we report on the experimental evaluation. We evaluate several variants of the dual ascent technique. These include the choice of the starting node, the premature termination of the dual step if primal progress is already guaranteed. The premature termination of the dual step is similar to stopping the shortest path search as soon as a sink is reached when starting from a source. Moreover, we compare our implementation with recent third-party implementations on instances that have been commonly used for benchmarks in previous works. These include randomly generated artificial instances and realistic instances. We conclude in Section 2.5.

2.2 The Dual Ascent Algorithm

2.2.1 Preliminaries

We have seen in the previous chapter that the capacitated min-cost flow problem can be written as a linear program. Moreover, when written as a primal-dual pair, the problem takes the form

$$\min\{c^T x : Ax = b \text{ and } 0 \le x \le u\} = \max\{b^T y - u^T z : A^T y - z \le c \text{ and } z \ge 0\},$$

where the dual variables $y \in \mathbb{R}^n$ are called *node potentials*. We proceed by introducing some further related notation:

Definition 2.1 (Residual Network, etc.). *Let $G = (V, A)$ be a directed graph.*

1. *A vector $x \in \mathbb{R}^m$ that satisfies $0 \le x \le u$ is called a pseudo-flow.*
2. *For a pseudo-flow $x \in \mathbb{R}^m$, we define* residual capacities $u_a^x = u_a - x_a$ *and* $u_{-a}^x = x_a$ *for all $a \in A$. We call $G^x = (V, A^x)$, where $A^x := \{a \in A \cup -A : u_a^x > 0\}$, the* residual network.
3. *For a pseudo-flow $x \in \mathbb{R}^m$, we call $b^x := b - Ax$ the* residual demands.
4. *For node potentials $y \in \mathbb{R}^n$, we call $c_a^y := c_a + y_v - y_w$ the* reduced costs *of an arc $a = (v, w) \in A^x$. Moreover, we define $z_a^y := \max\{0, -c_a^y\}$.*

For any given y, the dual constraint of an arc a can always be satisfied by setting $z_a = z_a^y$, which is the optimal choice of z for fixed y.

2.2.2 Algorithm

The algorithm that we present in this chapter is a primal-dual technique in the sense that it maintains primal as well as dual variables. The algorithm maintains the following *invariants*:

(a) The dual variables, i.e., node potentials y and corresponding variables z are dual feasible.

(b) The primal vector x is a pseudo-flow.

(c) The primal and dual variables x, y fulfill *complementary slackness*, i.e., for all $a \in A$:

$$c_a^y > 0 \implies x_a = 0, \tag{2.1}$$

$$c_a^y < 0 \implies x_a = u_a, \text{ and} \tag{2.2}$$

$$0 < x_a < u_a \implies c_a^y = 0. \tag{2.3}$$

The algorithm, see Algorithm 2.1 for a pseudo-code formulation, performs a dual and a primal step in each iteration. The dual step can be understood as calling the crossover

Algorithm 2.1: DualAscent

```
Input  : directed graph $G = (V, A)$, $b \in \mathbb{Z}^n$ with $\mathbb{1}^T b = 0$, $c \in \mathbb{Z}^m_{\geq 0}$, and $u \in \mathbb{Z}^m_{\geq 1}$
Output: optimal flow $x \in \mathbb{Z}^m_{\geq 0}$ and node potentials $y \in \mathbb{Z}^n$
$x := 0$ and $y := 0$
while $\|b^x\|_1 > 0$ do
    Choose starting node $s$ with minimal label such that $b^x(s) \neq 0$
    $y, T' \leftarrow$ DualStep $(G^x, s, b^x, y)$
    $x \quad \leftarrow$ PrimalStep $(x, y, A \cap (T' \cup -T'))$
return flow $x$ and potentials $y$
```

Algorithm 2.2: DualStep

```
Input  : directed graph $G' = (V, A')$, starting node $s \in V$, deficits $b'$, and potentials $y$
Output: node potentials $y$ and spanning tree $T' \subseteq A'$
$S^1 := \{s\}$.
for $k = 1, \dots, n-1$ do
    if $b^x(S^k) < 0$ or $\delta^{\text{in}}_{G'}(S^k) = \emptyset$ then
        $a^k = (v^k, w^k) := \text{argmin}\{c^y_a : a \in \delta^{\text{out}}_{G'}(S^k)\}$ and $\Delta^k := c^y_{a^k}$.
    else
        $a^k = (w^k, v^k) := \text{argmin}\{c^y_a : a \in \delta^{\text{in}}_{G'}(S^k)\}$ and $\Delta^k = -c^y_{a^k}$.
    $y_v \leftarrow y_v + \Delta^k$ for all $v \in V \setminus S^k$
    $S^{k+1} \leftarrow S^k \cup \{w^k\}$ and $T' \leftarrow T' \cup \{a^k\}$
return potentials $y$, spanning tree $T'$.
```

Algorithm 2.3: PrimalStep

```
Input  : pseudoflow $x$, potentials $y'$, and spanning tree $T \subseteq A$
Output: pseudoflow $x'$ such that $\|b^{x'}\|_1 < \|b^x\|_1$
Compute $f \in \mathbb{Z}^m \setminus \{0\}$ such that
    1. $f_a = 0$ for all $a \in A \setminus T$,
    2. $0 \leq x + f \leq u$, and
    3. $\min\{0, b^x(v)\} \leq b^{x+f}(v) \leq \max\{0, b^x(v)\}$ for all $v \in V$,
or return Infeasible if no such flow exists.
```

procedure from [BK14] on the residual network G^x with the current iterate y. For self-containment and because of the slightly different setting, we will repeat the description of the algorithm here, see Algorithm 2.2. It iteratively constructs a spanning tree of the graph on which the primal update will take place. The idea is to shift the potentials along a cut in each iteration in such a way that the dual objective does not decrease. Each iteration makes the reduced cost of at least one arc crossing this cut zero, and one of these arcs enters the spanning tree. In a primal step, the flow variables are updated by a routing over the constructed spanning tree. If after the primal step all residual demands are zero, the algorithm terminates; otherwise, a new iteration starts.

2.2.3 Correctness and Further Properties

The invariants of the algorithm imply correctness as soon as termination is established. Termination of the algorithm follows by showing that $\|b^x\|_1$ strictly decreases in every iteration (see Theorem 2.6). We will now formally show that the invariants are maintained.

The following claim is a direct consequence of the minimal choice of Δ among the reduced costs of all in- or outgoing arcs, respectively.

Claim 2.2. *Let x, y be the iterates at the beginning of a specific iteration of Algorithm 2.1, and let y' be the dual iterate after the call of* DualStep. *If $c_a^y \geq 0$ for all arcs $a \in A^x$, then also $c_a^{y'} \geq 0$ for all arcs $a \in A^x$. Moreover, if $c_a^y = 0$, then also $c_a^{y'} = 0$.*

The following lemma shows that the complementary slackness conditions (2.1) and (2.2) are maintained by the algorithm.

Lemma 2.3. *Let x, y be the iterates at the beginning of an iteration of Algorithm 2.1, and let x', y' be the iterates after that iteration. Assume that complementary slackness holds for x, y. Then:*

1. *If $c_a^{y'} < 0$, then $x'_a = u_a$ for any $a \in A$.*
2. *If $c_a^{y'} > 0$, then $x'_a = 0$ for any $a \in A$.*

Proof. Let $a \in A$, and let T' be the tree returned by DualStep. If either $a \in T'$ or $-a \in T'$, then $c_a^{y'} = 0$, and there is nothing to show. Hence, assume that neither is the case. Then, f_a is set to 0 in step 1 of PrimalStep, and thus, $x'_a = x_a + f_a = x_a$.

1. Let $c_a^{y'} < 0$, and assume for the sake of contradiction that $x_a < u_a$. It follows that $a \in G^x$. By complementary slackness of x, y, we get that $c_a^y \geq 0$. Applying Claim 2.2 yields $c_a^{y'} \geq 0$, which is a contradiction. Hence, $x_a = u_a$, and thus, $x'_a = u_a$ holds.
2. Let $c_a^{y'} > 0$, and assume for the sake of contradiction that $x_a > 0$. It follows that $-a \in G^x$. By complementary slackness of x, y, we get that $c_a^y \leq 0$, or equivalently, $c_{-a}^y \geq 0$. Applying Claim 2.2 to $-a$ yields $c_{-a}^{y'} \geq 0$, and thus, $c_a^{y'} \leq 0$, which is a contradiction. Hence, $x_a = 0$, and thus, $x'_a = 0$ holds. □

Lemma 2.4. *At the beginning of the while-loop of Algorithm 2.1, the invariants hold, i.e., (a) y, z are feasible, (b) x is a pseudo-flow, and (c) the complementary slackness conditions* (2.1) *to* (2.3) *hold.*

Proof. Part (a) follows from the definition of z^y: Let $a = (v, w) \in A$ be any arc, then $y_w - y_v - z_a^y \leq y_w - y_v + c_a^y = c_a$. Part (b) is an immediate consequence of step 2 in PrimalStep. For (c), note that conditions (2.1) and (2.2) follow together with Lemma 2.3. For condition (2.3), apply the contrapositions of the two statements in Lemma 2.3. □

Let us denote with $G_{s,t}^x = (V', A')$ the graph with $V' := V \dot\cup \{s, t\}$ and $A' := T \cup -T \cup A_{s,t}$, where

$$A_{s,t} = \{(s, v) : b_v^x < 0\} \cup \{(v, t) : b_v^x > 0\}.$$

Define capacities u' on $G_{s,t}^x$ as

$$u'_a = \begin{cases} u_a^x & \text{if } a \in A \cup -A \\ -b_v & \text{if } a = (s, v) \in A_{s,t} \\ b_v & \text{if } a = (v, t) \in A_{s,t}. \end{cases}$$

Lemma 2.5. *If the value of a maximum s-t-flow in the graph $G^x_{s,t}$ is 0, then the problem is infeasible; otherwise,* PrimalStep *returns a flow. Moreover, any non-zero feasible s-t flow restricted to the arcs $A \cup -A$ satisfies the conditions in* PrimalStep.

Proof. If the max-flow value is 0, then there is an s-t-cut in $G^x_{s,t}$ with 0 capacity by the max-flow/min-cut theorem. Let $\{s\} \cup S$ with $S \subseteq V$ be such a min-cut. Since $||b^x||_1 > 0$, the min-cut cannot be just $\{s\}$ itself. Thus, $S \neq \emptyset$ and contains all nodes $v \in V$ with $b^x_v < 0$. For the same reason, $S \neq V$, and moreover, it must not contain any $v \in V$ with $b^x_v > 0$. Moreover, there are no outgoing arcs from S on the min-cut because they would have a non-zero residual capacity. Let $a = (v, w) \in T \cup -T$ be the arc with $w \in S$ and $v \notin S$ that has first been added to the tree in the dual step. Let $S^k \subset V$ denote the set of visited nodes at that iteration in the dual step. Note that $S^k \subseteq S$ if $b^x(S^k) < 0$ and $S^k \subseteq V \setminus S$ if $b^x(S^k) > 0$. This implies that either $u(\delta^{\text{out}}(S^k)) < -b(S^k)$ in the former case or $u(\delta^{\text{in}}(S)) < b(S^k)$ in the latter case, certifying infeasibility of the min-cost flow problem. If the max-flow value is at least 1, any non-zero feasible s-t-flow restricted to the arcs in $A \cup -A$ satisfies the conditions in Algorithm 2.3 by construction. Moreover, any flow f returned by the algorithm can be extended to a feasible s-t-flow by setting the flows on the arcs incident to s and t to the corresponding value $|b^{x+f}_v|$. □

Theorem 2.6. *Algorithm 2.1 terminates after at most $||b||_1/2$ iterations and returns a correct result.*

Proof. Let x, y be the iterates at the beginning of an iteration of Algorithm 2.1, and let x', y' be the iterates afterwards. We will show that $||b^{x'}||_1 < ||b^x||_1$. To this end, we argue that there are two nodes s and t, s.t. $b^x(s) < b^{x'}(s) \leq 0$ and $b^x(t) > b^{x'}(t) \geq 0$. This is sufficient, due to the requirement of PrimalStep that $\min\{0, b^x(v)\} \leq b^{x'}(v) \leq \max\{0, b^x(v)\}$ for all $v \in V$.

Let the nodes be numbered in the order in which they join S, i.e., $S_i = [i]$ for $i = 1, \dots, n$. Assume that $b^x(s) < 0$, the case $b^x(s) > 0$ is symmetric. Let k be the number of the first iteration in which $b^x(S^k) \geq 0$. It is clear that such k exists since $\mathbb{1}^T b^x = 0$, and it is also clear that $b^x(w^k) > 0$. It follows that the tree $T \subseteq A$ contains a directed path $P = a_1, \dots, a_\ell$ from s to w^k with $u^x_{a_i} > 0$ for all $i \in [1, k]$, and thus, together with Lemma 2.5, it follows that at least one unit of flow is sent along P, implying that $||b^{x'}||_1 < ||b^x||_1$. Optimality of the output solution directly follows from the fact that the algorithm terminates with x, y satisfying the invariants (dual feasibility, complementary slackness and the property that x is a pseudoflow) and $||b^x||_1 = 0$, which implies that x is a feasible flow. □

The above proof only gives a very pessimistic estimate for the complexity of DualAscent, since it only guarantees that $||b^x||_1$ decreases by 2 in every step. Note that a standard scaling approach for the demands and the capacities can be used to achieve a bound that is polynomial in the size of the graph and in the encoding length of the data. It is, however, not evident at first glance whether such a scaling approach would yield to a good behavior in practice, especially in view of the overhead of multiple *phases* that would have to be carried out. In Section 2.4.3, we will give an experimental estimate for the running time of DualAscent. We proceed with the below theorem that justifies to call the algorithm a *dual ascent* algorithm. The running time analysis presented above relies on convergence in primal feasibility, whereas the following theorem concerns the dual progress of the algorithm.

Theorem 2.7. *Let y, z^y be the dual iterates before a specific iteration k of* DualStep, *let x be the current primal iterate, and let $y', z^{y'}$ be the iterates after iteration k. Then, $b'^T y' - u^T z^{y'} \geq b'^T y - u^T z^y$.*

Proof. Let us denote the current set of nodes $S^k \subseteq V$ with S, and let $\Delta = \Delta^k$. Define

$$Z^{\text{in}} := \{a \in \delta^{\text{in}}(S) : c^y_a < 0\} \quad \text{and} \quad Z^{\text{out}} := \{a \in \delta^{\text{out}}(S) : c^y_a < 0\}.$$

First, note that in the case where $\Delta = 0$, the statement is trivially fulfilled because equality holds. Otherwise, by non-negativity of the reduced costs of arcs in the residual network, we obtain that $\Delta > 0$. Let us consider the case where $\Delta = \min\{c_a^y : a \in \delta_{G^x}^{\text{out}}(S)\}$; the other case is symmetric. Then, $\Delta \leq c_a^y$ for $a \in \delta^{\text{out}}(S) \setminus Z^{\text{out}}$. Moreover, for every arc $a \in Z^{\text{in}}$, it follows that $x_a = u_a$, and thus, $-a \in \delta_{G^x}^{\text{out}}(S)$, and thus, $\Delta \leq c_{-a}^y = -c_a^y$ for all $a \in Z^{\text{in}}$. It follows that

$$\begin{aligned} b^T y' - u^T z^{y'} &= b^T y + \Delta \cdot b(V \setminus S) - \sum_{a \in A \setminus \delta(S)} u_a z_a^y \\ &\quad - \sum_{a \in \delta^{\text{in}}(S) \setminus Z^{\text{in}}} u_a \underbrace{\max\{0, -c_a^y - \Delta\}}_{=0=z_a^y} - \sum_{a \in Z^{\text{in}}} u_a \underbrace{\max\{0, -c_a^y - \Delta\}}_{=z_a^y - \Delta} \\ &\quad - \sum_{a \in \delta^{\text{out}}(S) \setminus Z^{\text{out}}} u_a \underbrace{\max\{0, -c_a^y + \Delta\}}_{=0=z_a^y} - \sum_{a \in Z^{\text{out}}} u_a \underbrace{\max\{0, -c_a^y + \Delta\}}_{=z_a^y + \Delta} \\ &= b^T y - u^T z^y + \Delta[b(V \setminus S) + u(Z^{\text{in}}) - u(Z^{\text{out}})]. \end{aligned}$$

Note that if $\Delta = \min\{c_a^y : a \in \delta_{G^x}^{\text{out}}(S)\}$, then either $b^x(S) \leq 0$ or $\delta_{G^x}^{\text{in}}(S) = \emptyset$ and $b^x(S) > 0$. In the latter case, the instance is infeasible, and in the former case, the definition of b^x yields $b(V \setminus S) \geq x(\delta^{\text{out}}(S)) - x(\delta^{\text{in}}(S))$, and thus, we obtain

$$\begin{aligned} b^T y' - u^T z^{y'} &\geq b^T y - u^T z^y + \Delta[x(\delta^{\text{out}}(S)) - u(Z^{\text{out}}) - (x(\delta^{\text{in}}(S)) - u(Z^{\text{in}}))] \\ &= b^T y - u^T z^y + \Delta[x(\delta^{\text{out}}(S) \setminus Z^{\text{out}}) - x(\delta^{\text{in}}(S) \setminus Z^{\text{in}})]. \end{aligned}$$

Assume for the purpose of contradiction that there exists an arc $a \in \delta^{\text{in}}(S) \setminus Z^{\text{in}}$ with $x_a > 0$. Then, it follows by complementary slackness that $c_a^y \leq 0$. Since $a \notin Z^{\text{in}}$, it follows that $c_a^y = 0$, and together with $x_a > 0$, this yields that $-a \in \delta_{G^x}^{\text{out}}(S)$ with $c_{-a}^y = 0$, a contradiction to $\Delta > 0$. We conclude that $x(\delta^{\text{in}}(S) \setminus Z^{\text{in}}) = 0$. This yields the result. □

2.2.4 Greedy Dual Ascent

In this section, we describe an extension of the dual ascent algorithm that we call *greedy dual ascent*. The underlying observation is the following: Given any cut $S \subset V$ and a scalar $\Delta \in \mathbb{R}$, consider the potential update $y_v \longleftarrow y_v + \Delta$ for all $v \in V \setminus S$. This is exactly what is done in DualStep with Δ set to the minimum reduced cost of all arcs in $\delta_{G^x}^{\text{out}}(S)$, if $b^x(S) < 0$ or $\delta_{G^x}^{\text{in}}(S) = \emptyset$ or to minus the minimum reduced cost of all arcs in $\delta_{G^x}^{\text{in}}(S)$ otherwise. Here we take a different approach of choosing Δ. In fact, we propose to choose Δ in a way that greedily maximizes the improvement of the dual objective function.

More precisely, let us define the function $f_S : \mathbb{R} \longrightarrow \mathbb{R}$ that for a given value $\Delta \in \mathbb{R}$ evaluates the change in the dual function value under the above change in potentials from y to $y' = y + \Delta \mathbb{1}_{V \setminus S}$:

$$\begin{aligned} f_S(\Delta) :&= b^T y' - u^T z^{y'} - b^T y + u^T z^y \\ &= -\Delta \cdot b(S) + \sum_{a \in \delta^{\text{out}}(S)} u_a \min\{0, c_a^y - \Delta\} + \sum_{a \in \delta^{\text{in}}(S)} u_a \min\{0, c_a^y + \Delta\}. \end{aligned}$$

We obtain the following lemma.

Lemma 2.8. *Let $a_1, \ldots, a_k$ be an ordering of $\delta^{\text{in}}(S) \cup \delta^{\text{out}}(S)$ such that $\kappa(a_1) \leq \kappa(a_2) \leq \ldots \leq \kappa(a_k)$, where we call*

$$\kappa(a) = \begin{cases} c_a^y & \text{if } a \in \delta^{\text{out}}(S) \\ -c_a^y & \text{if } a \in \delta^{\text{in}}(S) \end{cases}$$

the event-points.

1. *The slope of $f_S(\Delta)$ on $(\kappa(a)-1, \kappa(a))$ is, for all $a \in \delta^{\text{out}}(S) \cup \delta^{\text{in}}(S)$, described by the function*

$$g_S(\Delta) := -b(S) - u(\{e \in \delta^{\text{out}}(S) : \kappa(e) < \Delta\}) + u(\{e \in \delta^{\text{in}}(S) : \kappa(e) \geq \Delta\}).$$

2. *If the problem is feasible, there exists a Δ such that $g_S(\Delta) \leq 0$.*

3. *Let i be smallest such that $\kappa(a_i) \leq 0$, then $\hat{\Delta} = \kappa(a_{i-1})$ is a maximum of f_S.*

Proof. 1. It is clear that the function $f_S(\Delta)$ is a piecewise linear function whose slope only changes at the points $\kappa(a)$ for $a \in \delta^{\text{out}}(S) \cup \delta^{\text{in}}(S)$. We compute the values of

$$\begin{aligned} f_S(\Delta) - f_S(\Delta-1) = -b(S) &+ \sum_{a \in \delta^{\text{out}}(S)} u_a[\min\{0, c_a^y - \Delta\} - \min\{0, c_a^y - \Delta + 1\}] \\ &+ \sum_{a \in \delta^{\text{in}}(S)} u_a[\min\{0, c_a^y + \Delta\} - \min\{0, c_a^y + \Delta + 1\}] = g_S(\Delta). \end{aligned}$$

2. Consider the arc a_k with maximal $\kappa(a_k)$. It follows that $g_S(\kappa(a_k)+1) = b(V \setminus S) - u(\delta^{\text{out}}(S)) \leq 0$, since for every cut S, we must have $b(S) \geq u(\delta^{\text{out}}(S))$, provided that the instance is feasible.

3. This follows because, in addition to being piecewise linear, f_S is also unimodal, since $g_S(\kappa(a_1)) \geq g_S(\kappa(a_2)) \geq \ldots \geq g_S(\kappa(a_k))$. □

We conclude that we can maximize $f_S(\Delta)$ by evaluating $g_S(\Delta) = f(\Delta) - f(\Delta-1)$ at all event-points $\kappa(a)$ for $a \in \delta^{\text{out}}(S) \cup \delta^{\text{in}}(S)$ and choose $\Delta = \kappa(a_{i_0-1})$ for a_{i_0} being the first event-point for which $g(\kappa(a_{i_0})) \leq 0$.

Algorithm 2.4: GreedyDualStep

```
Input  : starting node $s$,
         pseudo-flow $x$ and
         potentials $y$
Output: potentials $y$ and spanning
         tree $T \subseteq G$
$S^1 := \{s\}$.
for $k = 1, \ldots, n-1$ do
    $\Delta^k \leftarrow \text{argmax}\{f_{S^k}(\Delta) : \Delta \in \mathbb{R}\}$
      and let $a^k = (v^k, w^k) \in \delta(S^k)$
      such that $\Delta^k = c^y_{a^k}$
    $y_v \leftarrow y_v + \Delta^k$ for all $v \in V \setminus S$
    $S^{k+1} \leftarrow S^k \cup \{w^k\}$ if $v^k \in S^k$ and
      $S^{k+1} \leftarrow S^k \cup \{v^k\}$ otherwise,
    $T = T \cup \{a^k\}$
return $y, T$.
```

Algorithm 2.5: PrimalStep′

```
Input  : pseudo-flow $x$, potentials $y'$
         and spanning tree $T \subseteq A$
Output: pseudo-flow $x'$ such that
         $||b^{x'}||_1 < ||b^x||_1$
Compute $f \in \mathbb{Z}^m \setminus \{0\}$ such that
    1. $f_a = 0$ for all $a \in A \setminus T$ with $c_a^{y'} > 0$,
    2. $f_a = u_a^x$ for all $a \in A \setminus T$ with $c_a^{y'} < 0$,
    3. $0 \leq x + f \leq u$, and
    4. $\min\{0, b^x(v)\} \leq b^{x+f}(v) \leq$
       $\max\{0, b^x(v)\}$ for all $v \in V$.
or return Infeasible if no such flow
  exists.
return $x' \leftarrow x + f$.
```

This leads to the greedy dual ascent algorithm. The pseudo-code of the main routine of this algorithm is identical to the one of Algorithm 2.1, with the calls to the dual and primal step routines substituted calls to the routines described in Algorithm 2.4 and 2.5.

We remark that setting Δ greedily as described above may cause violated complementary slackness constraints that possibly have to be repaired in the subsequent primal step by saturating and de-saturating arcs, respectively, which in turn may yield a smaller reduction in $\|b^x\|_1$ as with the standard non-greedy dual ascent version. It is however not obvious, whether this gain in the dual objective value compared with the possible loss in primal feasibility will yield to a faster convergence of the algorithm in theory or practice. We will investigate the latter at the end of the following section.

2.3 Implementation Details and Variants

We describe some implementation choices and details.

Lazy Potential Update. The update of the potentials in DualStep should not be done in a naive way, since this would directly imply quadratic running time in n for a single iteration. Instead, the potentials can be updated lazily, as already described in [BK14], in such a way that the potential y_v of a node v is only set once in each call of DualStep, namely at the time when v enters S^k for some k.

Choosing the Starting Node. We implemented different ways of choosing the starting node $s \in V$ from which the search in DualStep takes off. The first variant is to start with the first node and, as soon as this one has deficit 0, move over to the second one, and so on. This is what is proposed in Section 2.2 as well as in the pseudo-code; we refer to it as `snbal`. The second variant is to always choose the node with maximal absolute deficit; we call it `maxdef`. A third variant is a combination of the previous two: Always choose the node with maximal absolute deficit and start with this node until it is balanced. Then, go over to the node with maximal absolute deficit, and so on. We call this variant `snbaldef`.

Different Tree Updates in Primal Step. The primal update described in the pseudo-code in Section 2.2 can be implemented efficiently as follows: Let T' be the spanning tree in the residual network G^x (rooted at the starting node s) that is returned by DualStep, and let $T = A \cap (T' \cup -T')$ be the corresponding spanning tree in G that is given to PrimalStep. Compute f as follows: Traverse the tree from bottom to top – at a node $v \in T$, let $a = \pm(v, w) \in A$ be an arc between v and a node w on the level above v: Now, send that amount of flow f_a along the arc a that minimizes $|b^{x+f}(v)|$, respecting the capacity and non-negativity constraints corresponding to a. More formally, let

$$f_a \longleftarrow \max\{0, \min\{u_a^x, |b_v^{x+f}|\}\}, \text{ and set } x_a \longleftarrow x_a + f_a.$$

Then, update the deficits at v and w, accordingly. Thereafter, traverse the tree from top to bottom and at each node v, for any arc $a = \pm(v, w)$ between v and a child w, change f as follows: Reduce f_a such that $\min\{0, b^x(v)\} \leq b^{x+f}(v) \leq \max\{0, b^x(v)\}$ holds. The idea of this approach is to first greedily send the maximum amount of deficit up the tree, and in the second traversal, correct the flows such that the absolute deficit of the nodes never increases and its sign stays the same.

We implemented another variant of the algorithm that we obtain by using a different primal update. The idea of this alternative approach is not to traverse the tree back down, but to stay with the greedily chosen flows f after the first traversal of the tree from bottom to top. Note that the difference between this and the original approach is that for an arc that goes up in T from v to w, we ignore the change in $|b^{x+f}(w)|$ while choosing f_a. In particular, it could be the case that $|b^{x'}(w)| > |b^x(w)|$ for some nodes w. We call this update version `defupt` for **def**icit **up t**ree, and the original version we call `rbn` for **r**estore **b**alanced **n**odes. The latter name originates from the fact that in this version all nodes with deficit zero will stay at zero deficit after the primal update.

Breaking out of the Dual Step. Another variant of the algorithm is obtained by breaking out of DualStep after fewer than $n - 1$ steps. We experimented with several different approaches here. The first stops as soon as a node with a deficit having a different sign than the starting node is found. Note that this approach strongly resembles successive shortest path, with the only difference being that the primal update is still done on a tree and not only on the path from s to that node. This version did not show to be very efficient on the instances tested; however, a relaxed version of it brings great advantages. The idea is to only break out of DualStep in the iteration k when the deficit *of the set* $b^x(S^k)$ changes its sign, or even only when the deficit hits exactly zero, i.e., $b^x(S^k) = 0$. We call these two variants `sc` and `def0` for *sign changes* and *deficit zero*, respectively. Note that in the lazy potential update version, the potentials of $v \in V \setminus S^t$ have to be shifted by the last Δ in order to maintain the invariants. The advantage of these approaches is that they do not search through the whole graph in each iteration. This, of course, comes at the cost of a poorer dual update, but might be beneficial when large portions of the graph already have optimal node potentials.

Priority Queue. From a theoretical point of view, the in- and outgoing arcs of S^k should be kept in a priority queue such as a Fibonacci heap [FT87]. In this case, the complexity of DualStep is $O(m + n \log n)$, similar to Dijkstra's algorithm [Dij59]. We implemented the following two approaches. First, we implemented our algorithm using the STL `priority_queue` that is built into C++. Second, we implemented what we call a `hybrid_queue`. It can be seen as a hybrid of a bucket queue and the standard priority queue. The `hybrid_queue` stores a normal array called `bucket`, a priority queue Q (we again used the STL implementation), and a variable called `lower_bound`. It provides two functions called `push()` and `top()`. If `push()` is called with an element v with key $k =$ `lower_bound`, it is stored in `bucket`. If v, however, has key $k >$ `lower_bound`, it is stored within Q. If $k <$ `lower_bound`, then all elements in `bucket` are moved to Q, v is added to `bucket`, and `lower_bound` is set to k. If `top()` is called, then an element is taken from `bucket` as long as it is non-empty. If `bucket` is empty, the minimum element of Q is returned, and `lower_bound` is set to the key of that element. In order to explain the idea underlying this approach, let us assume that the same starting node s is chosen for several iterations. Then, it is likely that there are a lot of arcs with reduced costs 0 spanning from s into the graph and these arcs can be taken out of the bucket at smaller cost using this approach

Greedy Dual Ascent Update. Clearly, the evaluation of g_S at all event-points would yield quadratic run-time for the dual step. However, the arcs on the cut can be kept in a balanced binary search tree, see for example [AL62] or [GS78], with the keys being the event-points $\kappa(a)$. In addition, for every $a \in \delta^{\text{out}}(S) \cup \delta^{\text{in}}(S)$, we store a *prefix-sum* and a *postfix-sum*. The *prefix-sum* corresponds to the sum of the capacities of all out-going arcs b with $\kappa(b) < \kappa(a)$ and the *postfix-sum* is equal to the sum of the capacities of all in-going arcs b with $\kappa(b) \geq \kappa(a)$. The update of the balanced binary search tree can be done in logarithmic time in this way, as well as the computation of the maximum of f_S. We use AVL-trees [AL62] for the cut edges and complement our own implementation with the ability of computing the prefix- and postfix-sums as described above. We denote the greedy dual ascent algorithm with `min_cost_flow_gda`. We remark that the greedy dual ascent algorithm in the the greedy dual step in the form as it is described in the pseudo-code implementation above, does not take any information about the primal iterate into account. An alternative approach, that takes the primal iterates into account, is to run the greedy dual step on the residual network instead of the original network. We call this variant `min_cost_flow_gdar`.

Since the ambition of `min_cost_flow_gda` and `min_cost_flow_gdar` is to yield a better progress in terms of dual objective value at the cost of progress in the primal feasibility, it might also be interesting to combine iterations of greedy dual ascent with iterations of the standard dual ascent method. We tried out the following two variants `min_cost_flow_gda_aq` and `min_cost_flow_gdar_aq`. In these two methods, the `aq` stands for *alternating queue*, i.e., we use the `gda` and `gdar`, respectively, for every second iteration and the default variant of the dual ascent algorithm for every other iteration.

Experimental Setting and Evaluation Details. We performed experiments on a compute server with 32 Intel (R) Xeon (R) E5-2680 2.70GHz cores and a total of 256 GB RAM running Debian GNU/Linux 7 with kernel 3.10.60. The code was compiled with `gcc` version 4.7.2 using the `-O3` flag. In all plots in this chapter, the results are averages over 25 runs, 5 runs each on 5 graphs of that size. The error bars indicate the 95%-confidence intervals of the estimated means over 25 runs.

2.4 Experimental Evaluation

2.4.1 Instances

We ran experiments on several different network instances. To this end, we used our own test instances (`feas_grid`) as well as third-party instances[1] that were generated for the study in [Pet15]. In the following, we briefly characterize the test set.

`netgen_8` These instances are generated using the Netgen generator [KNS74]. Netgen instances are random instances. In the case of `netgen_8`, the networks are sparse, i.e., $m = 8n$, and the capacities and costs are chosen uniformly at random between 1 and $1,000$ and 1 and $10,000$, respectively. There are roughly $\sqrt{n}$ sinks and sources and the total demand is $1,000\sqrt{n}$.

`netgen_sr` Same as `netgen_8`, but the networks are denser, in fact $m \approx n\sqrt{n}$.

`goto_8` They are generated using the Goto generator [GK91]. Goto instances are grid instances on tori. They are known to be rather hard instances. In the case of `goto_8`, the networks are sparse, i.e., $m = 8n$, and the capacities and costs are chosen uniformly at random between 1 and 1000 and 1 and 10000, respectively. There is one source and one sink node, and the supply is adjusted to the capacity [Pet15].

`road_paths` These instances are road networks from seven different states in the USA, where the edge cost is set to the travel time along the edge. All capacities are set to one, and there are $\approx \sqrt{n}/10$ randomly selected sources and sinks, each with a demand that depends on the maximum flow that can be sent between these sources and sinks.

`road_flow` Same as above but the capacities are not set to 1, but depending on the category of the road, to either 40, 60, 80 or 100.

`feas_grid` We created these instances using our own generator: We generate a 2D grid in which every node has a forward and a backward arc to its neighbors. We then sample capacities uniformly at random between 1 and U and costs uniformly at random between $-C$ and C. Initially, all demands are set to zero. Then, arcs with negative

[1]Kovács, Péter: Benchmark Data for the Minimum-Cost Flow Problem, website retrieved on August, 26, 2015, `https://lemon.cs.elte.hu/trac/lemon/wiki/MinCostFlowData`

costs are saturated, and in this way, demands are generated. More precisely, for an arc $a = (v, w)$ with $c_a < 0$, the arc $-a = (w, v)$ is introduced with cost $c_{-a} = -c_a$, capacity u_a. Then, b_v is increased by u_a, and b_w is reduced by u_a. This always generates feasible instances. We generate such networks for increasing C and U in order to investigate the dependence of the running time of our algorithm on u, c and b.

2.4.2 Running Time Hypotheses

We formulate hypotheses on the dependence between the running time of the algorithm and different parameters of the input. These hypotheses are tested in the next subsection. The first hypothesis concerns the behavior of the dependence between the running time and the one-norm of the demand. The second one is related to the magnitude of the costs.

Hypothesis 2.9. *The running time of the algorithm with the* `defupt` *primal update and a starting node with maximum absolute deficit scales polynomially in* $\|b\|_1$.

Hypothesis 2.10. *The maximal cost C does not have a significant impact on the overall running time.*

2.4.3 Testing the Hypotheses

We created the `feas_grid` instances for testing the hypotheses from Subsection 2.4.2. To this end, we generated grids of size 250 times 250, with increasing values of C and U. In order to test Hypothesis 2.9 that concerns the running time dependence on $\|b\|_1$, we first investigate the dependence of $\|b\|_1$ and U for these instances. We use this detour to generate demands via saturation of negative cost arcs because randomly sampled demands lead to a large portion of instances being infeasible, which makes rejection sampling impractical. Moreover, the randomized construction used suggests a linear behavior between these two quantities. Indeed, a correlation coefficient of more than 0.9999 and a significant linear regression clearly support this claim.

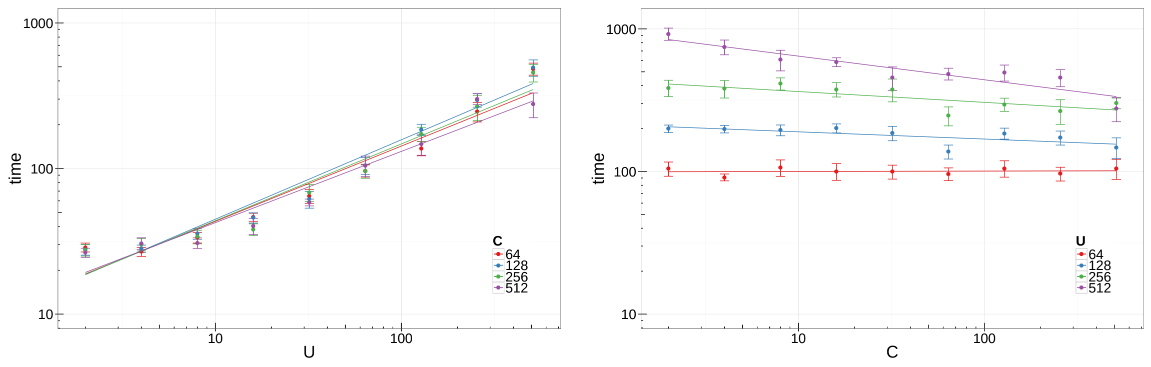

FIGURE 2.1: **On the left:** The dependence of the running time (in seconds) on the maximal capacity U for increasing values of C $(64, 128, 256, 512)$. **On the right:** The dependence of the running time on the maximal cost C. The depicted plots have increasing maximal capacity U $(64, 128, 256, 512)$.

On the left in Figure 2.1, the running times are on the y-axis and the maximal capacity U is on the x-axis. A regression yields the exponents 0.52 ± 0.10, 0.55 ± 0.10, 0.53 ± 0.10, and 0.49 ± 0.10 for $C = 64, 128, 256$, and 512, respectively. The *coefficients of determination* are above 0.94 in all cases. That is, roughly 94% of the variance of the data can be explained by the fitted power model. Since the "magical" value of 1/2 is within the 95%-confidence

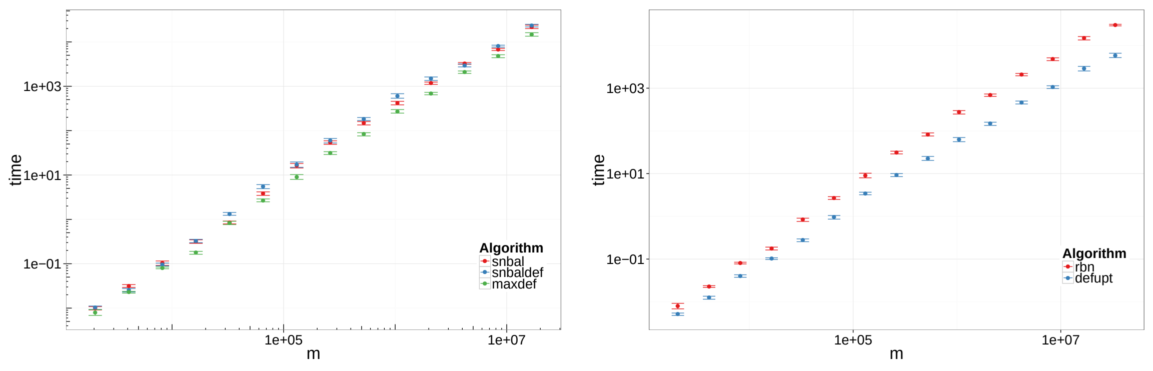

Figure 2.2: **On the left:** Comparison of three different ways of choosing the starting node. The red circles describe the running time for choosing the same node until it is balanced and then switching over to the next ones in the order of their labels. The green circles correspond to choosing the node that has maximal absolute deficit in each iteration. The blue circles correspond to a combination of the two – choose the node with maximum absolute deficit, and stay at that one until it is balanced. **On the right:** Running time comparisons of two different primal update routines, both with choosing a node with maximum absolute deficit as the starting node. Here, `rbn` (red circles) stands for the variant described in PrimalStep and `defupt` (blue circles) for the variant where the deficits are only sent up the tree, from the leaves to the root, greedily. Both experiments used the `netgen_8` instances.

intervals for the exponents in all cases, we may conjecture the running time to scale with the square root of $\|b\|_1$. In order to test Hypothesis 2.10, we also measured the running time of the algorithm on grid instances of size 250 times 250 for different values of $U = 64, \ldots, 512$. As indicated in the plot on the right in Figure 2.1, we cannot refute Hypothesis 2.10.

2.4.4 Evaluation of Different Variants

In this section, we evaluate the running time of the different variants of our algorithm described in Section 2.3. Figure 2.2 contains two plots. On the right, we evaluate the impact of the method of **choosing the starting node** on the running time. We conclude that the choice does not have a significant impact (on random graphs, here `netgen_8`). There is, however, a small advantage to using the node with the largest absolute deficit as the starting node. Figure 2.2 on the right, shows running times of two **different primal update methods** on the `netgen_8` test set. The greedy variant `defupt` that only traverses the tree once has a big advantage over the `rbn` variant – it is faster by almost a factor of 10 on the considered test set and also gives a better asymptotic behavior.

On the left in Figure 2.3, we can see the impact of **breaking out of the dual step**. The plot shows that in practice, not growing a tree of size $n-1$ in every step brings a huge advantage on some instances. In particular, the version where we interrupt the dual step as soon as the deficit of the set S^k becomes zero gives an asymptotic running time advantage over the standard version. On the right in Figure 2.3, we can see the effect of the **hybrid queue**. It seems that the impact is not too large; however, there is a slight advantage to using the hybrid queue.

2.4.5 Greedy Dual Ascent

We test the following hypothesis concerning the greedy dual ascent variant:

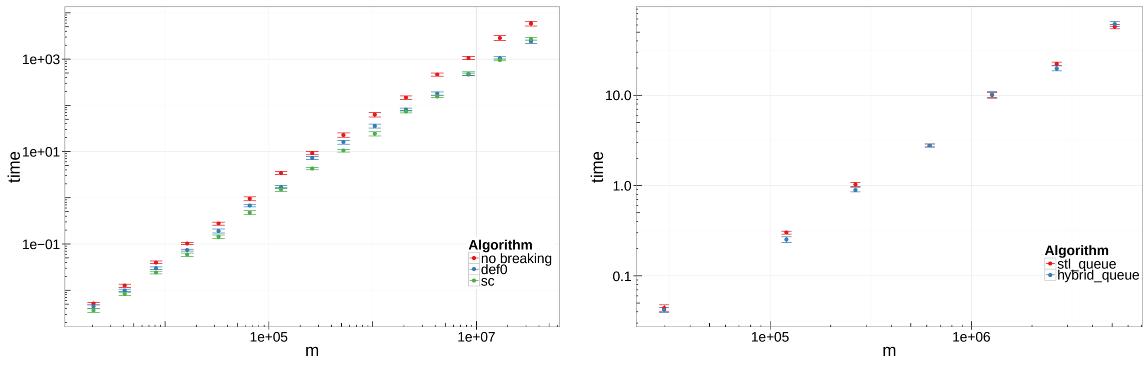

FIGURE 2.3: **On the left:** Experiments with breaking out of the dual step loop with different criteria. The red circles are without breaking out, the blue ones are with breaking out if $b^x(S)$ becomes zero, and the green ones correspond to breaking out when the sign of $b^x(S)$ changes for the first time. The instances used for these runs are `netgen_8` instances. **On the right:** The effect of using the hybrid queue. Both methods do not break out of the dual step. The red circles correspond to the version using the STL `priority queue`, whereas the blue circles correspond to the variant with the hybrid queue. Here, in both variants, the starting node with the largest absolute deficit is used. The instances are `road_paths` instances.

Hypothesis 2.11. *The greedy dual step yields to a faster convergence to the optimal dual objective value, thus to a smaller number of iterations, and thus to better asymptotic dependence of the run-time on m.*

Figure 2.4 shows the scaling behavior of the running times of the 4 `gda`-variants described above and the default variant of the dual ascent algorithm, respectively, on the `netgen_8` graph class. We can see a much worse scaling behavior of all 4 `gda`-variants as opposed to the default dual ascent method. In fact a linear regression of $\log(T_{\texttt{gda}}) - \log(T_{\texttt{mcf}})$ versus $\log(m)$ reveals that we shall reject Hypothesis 2.11 at a significance level $p < 1.5 \cdot 10^{-13}$.

In summary, we can observe that the gain in dual objective value that is achieved by the greedy variant does not seem to compensate the loss in primal feasibility that it is caused by saturating or de-saturating arcs whose reduced cost becomes negative.

2.4.6 Comparison with Other Algorithms

Similar to comparing algorithms by asymptotic running times, in this section, we evaluate the scaling behavior of our implementation and third-party implementations of known algorithms w.r.t. graph size, i.e., number of edges m. Our choice of candidates is based on the recent study [Pet15] that claims that today's fastest min-cost flow code is contained in the LEMON Graph Library[2]. Based on their findings, we compare our implementation (`our`) with their implementations of three different algorithms: the network simplex algorithm (`ns`), the cost scaling algorithm (`cos`), and the successive shortest path method (`ssp`). According to [Pet15], the fastest algorithms on most instances (such as `netgen_8`, `goto_8`, and `netgen_sr`) are `ns` and `cos`. We include `ssp` because we consider our algorithm to be a generalization of this approach.

The exponents for the different implementations and graph classes can be found in Table 2.1. The bold numbers indicate the best exponent for the dependence of the running

[2]`https://lemon.cs.elte.hu/trac/lemon`

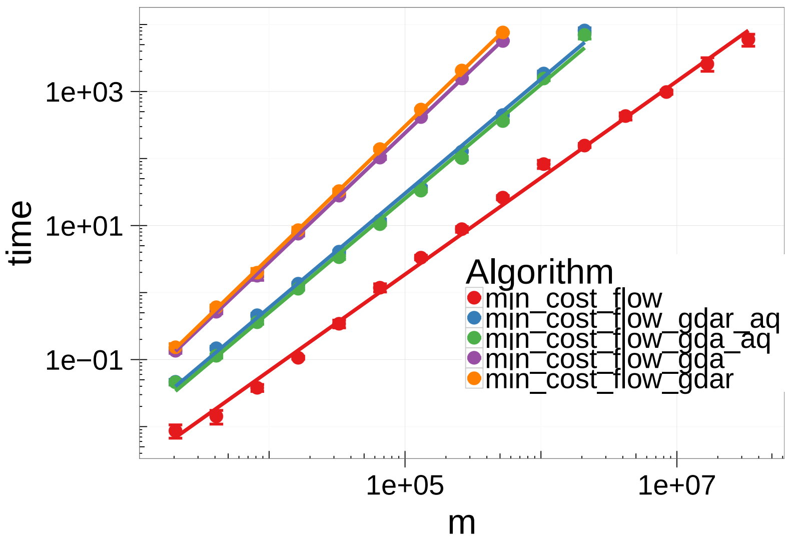

FIGURE 2.4: Running time dependence on m of `gda` variants and the default variant on the `netgen_8` graph class.

	`our`	`ssp`	`ns`	`cos`
`netgen_sr`	1.34 ± 0.02	1.42 ± 0.01	1.25 ± 0.02	**1.17** ± **0.02**
`netgen_8`	1.37 ± 0.02	1.80 ± 0.02	1.69 ± 0.02	**1.22** ± **0.01**
`goto_8`	1.66 ± 0.04	2.03 ± 0.02	2.17 ± 0.03	**1.53** ± **0.02**
`road_paths`	**1.33** ± **0.03**	1.40 ± 0.04	1.74 ± 0.03	1.50 ± 0.02
`road_flow`	**1.42** ± **0.03**	1.43 ± 0.04	1.76 ± 0.04	1.46 ± 0.03

TABLE 2.1: Fitted exponents describing the scaling behavior w.r.t. the number of arcs together with the 95%-confidence intervals.

time on the number of edges for the graphs from the corresponding class. Observe that our implementation beats the other three on the road networks tested. Moreover, our exponents are better than the ones of network simplex, except for the dense `netgen` graphs. Last but not least, we obtain a better scaling behavior than successive shortest path on all classes, which is interesting as our approach can be seen as a generalization of successive shortest path. The coefficients of determination for the corresponding regressions show that the running time data is well-described by a scaling behavior of the form m^c, where c is the corresponding fitted exponent.

We remark that all the data underlying the regression was generated with the following version of the algorithm. The potentials are updated lazily. We always choose the node with maximal absolute deficit as starting node for the dual step. The `defupt` primal update version is activated. We use the `def0` variant for the termination of the dual step, and we use the hybrid queue as priority queue.

The source code, the instances, and additional material are available for download on the project page.[3]

[3]`http://resources.mpi-inf.mpg.de/networkflow/`

2.5 Conclusion

In this chapter, we have presented a simple fully-combinatorial algorithm for the capacitated min-cost flow problem that is competitive with recent third-party implementations of well-known algorithms and even outperforms them on certain realistic instance classes such as road networks. We have formally proven correctness of the algorithm and have evaluated its run-time experimentally. We have seen that this very simple combinatorial approach leads to extremely fast implementations that solve min-cost flow instances with millions of arcs within seconds, although its theoretical worst-case run-time bound is not competitive with the most efficient methods in theory. As described in the introduction, the currently most efficient methods in theory are based on methods from continuous optimization, more precisely, on interior point methods. Such an approach and its practicability will be topic of the following chapter.

Chapter 3

An Integer Interior Point Method for Min-Cost Flow

3.1 Introduction

We have seen a combinatorial method for min-cost flow in the previous section and have seen that an implementation of this method is very fast in practice although its theoretical time complexity lacks behind the currently best known bounds. We have already mentioned the contributions based on interior point methods[1] that achieve the record bounds in the introduction, we recap the most important results: The bound of $\tilde{O}(m^{3/2} \log^2 U)$ was obtained by Daitch and Spielman [DS08] and later (up to log factors) the same bound was shown in [BK14] with a simple potential reduction algorithm.[2] The breakthrough result of Lee and Sidford [LS14] yields the bound of $\tilde{O}(m\sqrt{n} \operatorname{polylog} U)$.

Although the interior point methods achieve these impressive record bounds, this efficiency did not yet transfer to practical implementations. As a matter of fact, there is no implementation of an interior point method that was reported to be competitive with the combinatorial approaches on real world problems. There are many reasons for this. First, combinatorial algorithms are rather easy to implement and necessary subroutines are readily available in standard libraries by now. The ingredients needed to achieve the better theoretical results are however rather involved and use complicated subroutines such as low-stretch spanning trees [AN12] or spectral vertex sparsifiers [Kyn+16] that are already difficult to implement efficiently on their own. Second, interior point methods are formulated for real arithmetic and implementations require high-precision floating point numbers in order to avoid numerical instabilities, even if all the input data is given by integer numbers. Addressing this second issue of interior point methods' implementations is the central topic of this chapter. By giving an interior point method that solely uses integer arithmetic on numbers of polynomial length, we obtain an algorithm that is guaranteed to be free of any numerical instabilities while still improving over the classical combinatorial methods in terms of worst-case time complexity.

We proceed by highlighting why the challenge of numerical stability appears in interior point methods, more precisely in path following methods. First, notice that the (uncapacitated) min-cost flow problem given by an instance (G, b, c) can be written as the primal-dual linear programming pair

$$\min\{c^T x : Ax = b \text{ and } x \geq 0\} = \max\{b^T y : A^T y + s = c \text{ and } s \geq 0\}. \tag{3.1}$$

We remark that the meaning of the slack variables $s = c - A^T y$ is the exact same as the reduced costs c^y in the previous chapter. Recall that we had defined $c_a^y := c_a + y_v - y_w$ for every arc

[1]See for example [MS16], for a gentle introduction to the topic.

[2]We recall that n is the number of nodes, m the number of arcs, and U an upper bound on the maximal capacity.

$a \in A$. In the previous chapter, we also have already made use of the following property: A pair of primal/dual optimal solutions to (3.1) will satisfy complementary slackness, see (2.1) to (2.3). In the setting without capacity constraints, the complementary slackness conditions reduce to $x_a s_a = 0$ for all $a \in A$ or equivalently to $x^T s = c^T x - b^T y = 0$. We remark that $x^T s$ is called the *duality gap* as it measures the gap between primal and dual objective value.

The intuition behind path following interior point methods is to relax this optimality condition $x_a s_a = 0$. More precisely, path following methods maintain a pair of primal/dual feasible solutions x and (y, s) such that $x_a s_a \approx \mu$ for all $a \in A$ for a positive parameter μ, which they drive to zero. They thereby follow the so-called *central path*, which is the set of all points satisfying $x_a s_a = \mu$ for all $a \in A$. As a consequence of μ approaching zero, the duality gap $x^T s \approx \mu m$ goes to zero. It may however happen, that either x_a or s_a becomes tiny while the other becomes huge. This behavior is the source of numerical issues as it requires to simultaneously deal with huge and tiny numbers.

Our way to resolve this issue is to only consider the arcs with sufficiently large values of x_a and s_a. That is, whenever x_a becomes too small, we delete the arc a from the network and whenever s_a becomes too small, we contract the arc a. We thereby obtain a *reduced problem* in form of a minor of the original network. We do not revoke any of these operations until termination of the algorithm. We show that a near-optimal solution for the reduced problem can be lifted efficiently to an optimal solution of the original network. Such arc deletions and contractions have been used in combinatorial optimization before [Tar85; Orl84], but, to our knowledge, not in combination with path-following methods for network flow problems.

For the reduced problem, we can show that one can stay sufficiently close to the central path by iterating over points in the vicinity of the central path that lie on a lattice that is such that the lattice points are numbers with not too large representation. More precisely, we show that if the greatest common divisor (gcd) of the demands $\beta := \gcd(b)$ and the gcd of the costs $\gamma := \gcd(c)$ are sufficiently large, we can perform all arithmetic operations on integers. The requirement on the gcds can be easily achieved by an initial scaling of the demands, capacities and costs. Such scaling comes at a very low price because the expected running time of $\tilde{O}(m^{3/2})$ of our algorithm only depends logarithmically on these scaling factors. We remark that it appears obvious that integer arithmetic can be achieved by scaling with a huge number that is, say, doubly exponential in m, n. However, we show that scaling with numbers that are polynomial in n is sufficient. Moreover, we manage to make the exact numerical requirements of our algorithm explicit by stating absolute bounds on the appearing numbers, without hiding anything in O-notation. We prove the following theorem.

Theorem 3.1. *Optimal primal and dual solutions for a min-cost flow problem can be computed in expected $\tilde{O}(m^{3/2})$ time using only integer arithmetic on numbers bounded by $2^{31} m^{10} U^2 C^2$, where U and C are bounds on the maximum capacity and cost, respectively.*

Note that the running time of our algorithm matches the time bound from [DS08], which dominates the known bounds for combinatorial algorithms in many settings. For a better comparison of the two results, note that [DS08] gives a bound on the bit length of the numbers appearing in their algorithms, as well: The methods work on numbers of bit length at most $O(\log(nU/\varepsilon))$, where $\varepsilon = (12m^2U^3)^{-1}$ for solving the standard min-cost flow problem exactly[3]. If the hidden constant in the O-notation can be made explicit to be, say, k, then this method could be used with integer arithmetic, as well, the resulting bound on the numbers there will then be $12^k n^k m^{2k} U^{4k}$, which is larger than our bound already for small k.

[3]Note that in the case of generalized min-cost flow, the algorithms of Daitch and Spielman only give ε-approximate solutions in general and thus the numerical requirements will also depend on the desired ε. For the standard min-cost flow problem they show that an ε of the size as described above is sufficient.

Note that for robust implementations, it is advantageous to know the exact precision requirements, as an appropriate scaling can be performed in the beginning without having to worry about numerical issues or overflows in the course of the algorithm. Alternatively, one can still use adaptive precision, check the correctness of the output, and repeat the computation with higher precision, if necessary. However, this approach usually requires changes to the implementation as insufficient precision may lead to diverging behavior of sub-routines, see for example [Ket+08]. Adding a timer is no solution either as it turns diverging behavior into worst-case behavior. However, such precautions are mandatory and the only choice, if the bounds are not known explicitly.

For the potential-reduction method from [BK14], only a bound on the number of arithmetic operations is given and a strict bound on the size of the emerging numbers is non-trivial to obtain, if not even impossible. It is essential for our proof for the method presented in this chapter to exploit arc contractions and deletions and to use a path-following method instead of a potential-reduction method. It is, furthermore, interesting to note that our lower bound on the value of primal variables does not depend on the costs, capacities, or demands in the network, but only on n and m. This might be of interest in the context of strongly polynomial time algorithms for the capacitated min-cost flow problem, where the current record bound is $\tilde{O}(m^2)$, due to Orlin [Orl88].

Finally, we want to remark that we consider this work rather as a first important step in the direction of making numerical requirements of interior point methods for combinatorial problems apparent, than as giving the ultimate final answer to practitioners. We feel that the work in this chapter, despite the fact that it treats all numerical details, is still fairly elegant.

Structure of the Remainder of this Chapter. The rest of this chapter is structured as follows. We proceed with preliminaries in Section 3.2. In Section 3.3, we introduce the path following algorithm, Section 3.4 contains details about implementing the centering step with integer arithmetic. We conclude in Section 3.5.

3.2 Preliminaries

3.2.1 Arc Contraction and Deletion

We have already discussed the interpretation of the parameter $\mu > 0$. We will measure the closeness to the central path w.r.t. μ by the relative deviation in the 1-norm $||\sigma||_1$, where $\sigma \in \mathbb{R}^m$ is defined as $\sigma_a := x_a s_a/\mu - 1$. A sufficiently small deviation, more precisely, $||\sigma||_1 \leq \delta$ for some constant $\delta < 1$, will guarantee that both x_a and s_a remain positive. As mentioned before, this small deviation however does not prohibit that either x_a or s_a becomes tiny, which we want to avoid. We deal with this issue in a radical way: That is, whenever x_a becomes too small, we delete the arc from the network and whenever s_a becomes too small, we contract the arc a. We thereby obtain a minor of the original network. This is captured formally by the following definition.

Definition 3.2. *Let $G = (V, A)$ be a directed graph with m arcs. For $\varepsilon > 0$ and $x, s \in \mathbb{R}^m_{\geq 0}$, we call the minor H of G obtained by deleting the arcs in $D := \{a \in A : x_a < \frac{\varepsilon}{m}\}$ and contracting the arcs in $C := \{a \in A : s_a < \frac{\varepsilon}{m}\}$ the ε-x-s-minor of G.*

This approach has its foundation in classical min-cost flow theory [AMO93, Section 10.6]. We review a well-known fact from general LP duality.

Lemma 3.3. *Let the pair*

$$\min\{c^T x : Ax = b, x \geq 0\} = \max\{b^T y : A^T y + s = c, s \geq 0\} \tag{3.2}$$

be bounded and feasible.

- *A primal optimal solution x^* of* (3.2) *is also an optimal solution of* $\min\{\bar{s}^T x : Ax = b, x \geq 0\}$ *for any dual feasible solution $\bar{y}, \bar{s}$ of* (3.2).
- *A dual optimal solution y^*, s^* of* (3.2) *is also an optimal solution of* $\min\{\bar{x}^T s : A^T y + s = c, s \geq 0\}$ *for any primal feasible solution $\bar{x}$ of* (3.2).

Proof.

- Let y^*, s^* be dual optimal solutions to (3.2). Define $y' := y^* - \bar{y}$ and $s' := s^* = c - A^T y^*$, then $A^T y' + s' = A^T y^* - A^T \bar{y} + s^* = c - A^T \bar{y} = \bar{s}$ and hence y', s' are feasible for $\max\{b^T y : A^T y + s = \bar{s}, s \geq 0\}$. Moreover, it holds that $x^{*T} s' = x^{*T} s^* = 0$. It thus follows that x^*, y', s' are primal and dual optimal for the pair
$$\min\{\bar{s}^T x : Ax = b, x \geq 0\} = \max\{b^T y : A^T y + s = \bar{s}, s \geq 0\}.$$
This shows the claim.
- Suppose for contradiction that y^*, s^* is not optimal. Then, since $A\bar{x} = b$,
$$\begin{aligned}\bar{x}^T s^* &> \min\{\bar{x}^T s : A^T y + s = c, s \geq 0\} = \min\{\bar{x}^T (c - A^T y) : A^T y + s = c, s \geq 0\} \\ &= c^T \bar{x} - \max\{b^T y : A^T y + s = c, s \geq 0\} = c^T \bar{x} - b^T y^* = \bar{x}^T s^*,\end{aligned}$$
which is a contradiction. □

The following lemma shows the rationale for deleting arcs with tiny flow x and thus huge reduced costs s in the vicinity of the central path (respectively, contracting arcs with tiny reduced costs and huge flow).

Lemma 3.4. *Let $x, s \in \mathbb{R}^m$ be primal/dual feasible solutions of (G, b, c) with $b \in \mathbb{Z}^n$ and $c \in \mathbb{Z}^m$.*

- *If $s_a > x^T s$ for some $a \in A$, then $x^*_a = 0$ for every optimal solution x^*.*
- *If $x_a > x^T s$ for some $a \in A$, then $s^*_a = 0$ for every optimal solution s^*.*

Proof. Due to the total unimodularity of the node-arc incidence matrix A, all basic solutions are integral. Thus, it suffices to show the claims for optimal integral basic solutions. All other optimal solutions are convex combinations of these.

- Assume for contradiction that $x^*_a > 0$ in an optimal integral solution, i.e., $x^*_a \geq 1$. Then $x^{*T} s \geq x^*_a s_a > x^T s$, this shows that x^* is not optimal for $\min\{s^T x : Ax = b, x \geq 0\}$, contradicting Lemma 3.3.
- Assume for contradiction that $s^*_a > 0$ in an optimal integral solution, i.e., $s^*_a \geq 1$. Then $x^T s^* \geq x_a s^*_a > x^T s$, this shows that s^* is not optimal for $\min\{x^T s : A^T y + s = c, s \geq 0\}$, contradicting Lemma 3.3. □

Note that the central path condition $\sum_{a \in A} |\frac{x_a s_a}{\mu} - 1| \leq \delta$ implies $x_a s_a \in [(1-\delta)\mu, (1+\delta)\mu]$ for any arc $a \in A$. Now assume that for some arc $a \in A$, the value of x_a becomes tiny, i.e., $x_a < \frac{\varepsilon}{m}$. Then $s_a \geq \frac{(1-\delta)\mu}{x_a} > \frac{(1-\delta)\mu m}{\varepsilon} \geq (1+\delta)\mu m \geq x^T s$ follows for $\varepsilon := \frac{1-\delta}{1+\delta} < 1$. Thus Lemma 3.4 applies and we can conclude that $x^*_a = 0$ in every optimum primal solution x^*. A completely symmetric approach shows that, if $s_a < \frac{\varepsilon}{m}$ for an arc $a \in A$, then $s^*_a = 0$ in every optimal dual solution s^*.

The analyses of classical combinatorial algorithms that use arc deletion or arc contraction, respectively, are based on an argument that termination is guaranteed because at least one arc has to be deleted or contracted after a certain amount of iterations and there are only m

arcs. However, we have a different termination criterion that mainly depends on the duality gap. In [BK14], it is shown that it suffices to compute a pair of primal/dual interior points with duality gap strictly less than 1 to efficiently perform a crossover to an optimum integral basic feasible solution. Here, we will need to extend that result from [BK14] by showing that proximity to the optimum solution w.r.t. a minor suffices. To this end, we first formally define what proximity means in our context.

Definition 3.5. *A pair of primal and dual feasible solutions x, s of (G, b, c) with $b \in \mathbb{Z}^n$ and $c \in \mathbb{Z}^m$ is called a* proxy *for the optimum of (G, b, c) if $\sum_{a \in A_H} x_a s_a < (1-\varepsilon)^2$ where A_H is the arc-set of an ε-x-s-minor of G for some $\varepsilon < 1$.*

3.2.2 Perturbation

Suppose, for example, that we have an ε-x-s-minor H that differs from G just by the deletion of a single arc $\hat{a}$. Hence, the set of nodes of H coincides with the one of G. However, projecting x to x_H, s to s_H, and c to c_H by removing the entry corresponding to $\hat{a}$ does not yield a pair of primal/dual feasible solutions x_H, s_H for (H, b, c_H), but it is rather a feasible solution to $(H, \tilde{b}, c_H)$ where $\tilde{b} = b - A\mathbb{1}_{\hat{a}}\mathbb{1}_{\hat{a}}^T x$. Note that the two demands vectors b and $\tilde{b}$ are almost identical because $x_{\hat{a}}$ is tiny. We first define formally what we mean by almost identical instances and then draw the connection to minors.

Definition 3.6. *An instance $(G, \tilde{b}, \tilde{c})$ is called ε-*perturbed *instance of (G, b, c), if $\|b - \tilde{b}\|_1 \leq 2\varepsilon$ and $\|c - \tilde{c}\|_1 \leq \varepsilon$.*

We introduce the notation $I_S := \sum_{a \in S} \mathbb{1}_a \mathbb{1}_a^T$ for some $S \subseteq A$ and continue by showing that the instance constructed by contracting arcs of small reduced cost and deleting arcs of small flow is in fact an ε-perturbed instance of the input instance (G, b, c).

Lemma 3.7. *Let x, s be a pair of primal/dual feasible solutions of (G, b, c) let $C := \{a \in A_G : s_a < \frac{\varepsilon}{m}\}$ and $D := \{a \in A_G : x_a < \frac{\varepsilon}{m}\}$ for some $0 < \varepsilon < 1$. Then $(G, \tilde{b}, \tilde{c})$ with $\tilde{b} = b - AI_D x$ and $\tilde{c} = c - I_C s$ is an ε-perturbed instance of (G, b, c).*

Proof. The proof is a simple calculation:

$$\|b - \tilde{b}\|_1 = \|AI_D x\|_1 \leq 2\|I_D x\|_1 \leq 2\varepsilon \quad \text{and} \quad \|c - \tilde{c}\|_1 = \|I_C c\|_1 \leq \varepsilon. \qquad \square$$

The following theorem shows the equivalence of the original and the perturbed problem. For a given instance (G, b, c) and a given spanning tree T of G, we call $x \in \mathbb{R}^m$ a *primal tree solution*, if $Ax = b$ and $x_a = 0$ for all $a \in A \setminus T$ and we call $y, s \in \mathbb{R}^{n+m}$ a *dual tree solution* if $A^T y + s = c$ and $s_a = 0$ for all $a \in T$.

Theorem 3.8. *Let $(G, \tilde{b}, \tilde{c})$ be an ε-perturbed instance of (G, b, c) for some $0 < \varepsilon < 1$ and let T be a spanning tree of G.*

a) If the unique tree solution w.r.t. T in $(G, \tilde{b}, \tilde{c})$ is primal feasible, then the unique primal tree solution w.r.t. T in (G, b, c) is feasible.

b) If the unique tree solution w.r.t. T in $(G, \tilde{b}, \tilde{c})$ is dual feasible, then the unique dual[4] tree solution w.r.t. T in (G, b, c) is feasible.

c) If the spanning tree T is optimal for $(G, \tilde{b}, \tilde{c})$, then it is optimal for (G, b, c).

[4]Here *unique dual* refers only to the reduced costs because $A^T(y + t\mathbb{1}) = A^T y$ for all $t \in \mathbb{R}$.

Proof. a) Let x_T and $\tilde{x}_T$ denote the unique primal tree solutions corresponding to T in (G, b, c) and $(G, \tilde{b}, \tilde{c})$, respectively, i.e., $A_T x_T = b$ and $A_T \tilde{x}_T = \tilde{b}$ holds. Similarly let d be the unique primal tree solution corresponding to T for the right hand side $b - \tilde{b}$, i.e., $A_T d = b - \tilde{b}$. Note that in order to show feasibility of x_T, it suffices to show that $x_T \geq 0$. It follows that

$$A_T d = b - \tilde{b} = A_T(x_T - \tilde{x}_T)$$

and thus by the uniqueness of d, we conclude $d = x_T - \tilde{x}_T$. It follows that $\|x_T - \tilde{x}_T\|_\infty = \|d\|_\infty \leq \|b - \tilde{b}\|_1/2 \leq \varepsilon$, using that $(G, \tilde{b}, \tilde{c})$ is an ε-perturbed instance. However, since $b \in \mathbb{Z}^n$ also $x_T \in \mathbb{Z}^m$ and since $\varepsilon < 1$ it follows that $\tilde{x}_T \geq 0$ implies $x_T \geq 0$.

b) Let y_T, s_T and $\tilde{y}_T, \tilde{s}_T$ denote the unique dual tree solutions corresponding to T in (G, b, c) and $(G, \tilde{b}, \tilde{c})$, respectively, i.e., $A^T y_T + s_T = c$, $s_T(a) = 0$ for $a \in T$ and $A^T \tilde{y}_T + \tilde{s}_T = \tilde{c}$, $\tilde{s}_T(a) = 0$ for $a \in T$ holds. Similarly let p, d be the unique dual tree solution corresponding to T for the right hand side $c - \tilde{c}$, i.e., $A^T p + d = c - \tilde{c}$. Note that in order to show feasibility of y_T, s_T, it suffices to show that $s_T \geq 0$. It follows that

$$A^T p + d = c - \tilde{c} = A^T(y_T - \tilde{y}_T) + s_T - \tilde{s}_T$$

and thus by the uniqueness of d, we conclude $d = s_T - \tilde{s}_T$. Furthermore for $a \in T$, we have $d(a) = 0$ and for $a \in A \setminus T$, it holds that $|d_a| = |c_a - \tilde{c}_a - (p_w - p_v)| = |c_a - \tilde{c}_a - \sum_{a' \in P(v,w)} c_{a'} - \tilde{c}_{a'}| \leq \|c - \tilde{c}\|_1$. It follows that $\|s_T - \tilde{s}_T\|_\infty = \|d\|_\infty \leq \|c - \tilde{c}\|_1 \leq \varepsilon$, using that $(G, \tilde{b}, \tilde{c})$ is an ε-perturbed instance. However, since $c \in \mathbb{Z}^m$ also $s_T \in \mathbb{Z}^m$ and since $\varepsilon < 1$ it follows that $\tilde{s}_T \geq 0$ implies $s_T \geq 0$.

c) Assume the tree T is optimal for $(G, \tilde{b}, \tilde{c})$, i.e., the tree is primal and dual feasible (complementary slackness). Together with part a and b, we conclude that T is also optimal for (G, b, c). □

We note that in the non-degenerate case, the reverse statements of a and b hold as well. As our algorithm will output a tree solution to the perturbed problem, the above theorem is sufficient to show that we can use the crossover algorithm from [BK14, Section 2] on the perturbed instance and obtain a tree that is also an optimal tree for the original problem.

3.3 Integer Interior Point Algorithm

This section is structured as follows. We will first describe how to obtain initial primal/dual solutions. We remark that this is an easy application of a previous result from [BK14]. We will then describe the main algorithm and the final crossover step. In the section thereafter, we will give details on the implementation of the centering step.

3.3.1 Finding Arbitrarily Central Initial Points

In this section, we show how to construct an auxiliary instance and corresponding feasible interior solutions that are arbitrarily close to the central path. We are actually going to give a construction that works for the more general capacitated min-cost flow problem. In fact, we can use a vert similar technique as used in [BK14] for a potential reduction method. For the sake of completeness, we nevertheless repeat the method here. Let us denote with (G^0, b^0, c^0, u) a given input instance of a min-cost flow problem with capacity constraints. Let V^0 denote the node set of G^0 and let A^0 denote its arc set. We are going to describe a method that constructs an auxiliary instance (G, b, c). The auxiliary instance is an uncapacitated min-cost flow problem and the method will give corresponding primal and dual *integer* interior solutions for it.

The first part of the construction was already presented in Chapter 1 and is illustrated in Figure 1.1: For each $a = (v, w) \in A^0$, insert the two nodes v, w into G as well as a new node vw together with arcs $\acute{a} = (v, vw)$ and $\grave{a} = (w, vw)$, read "*a up*" and "*a down*". Define the new costs by $c_{\acute{a}} := c_a$ and $c_{\grave{a}} := 0$. The new demand vector is given by $b_v := b_v^0 - u(\delta^{\text{in}}(v))$ for nodes $v \in V^0$ and $b_{vw} := u_{(v,w)}$ for the newly inserted nodes.

Our goal now is to find values x, y, s that are primal and dual feasible, integral, as well as close to the central path, more precisely $\|\sigma\|_1 = \sum_{a \in A} |x_a s_a / \mu^{(0)} - 1| \le \delta$ for some parameters $\mu^{(0)} > 0$ and $\delta \ge 0$. We show that the technique from [BK14] can actually achieve this. With the right choice of parameters, the technique yields closeness to the central path for arbitrary δ. We proceed as follows:

1. We compute an integral (not necessarily feasible) tree solution z in G^0 for an arbitrary spanning tree T and set $x_{\grave{a}} = x_{\acute{a}} = u_a/2$.[5]

2. We introduce the additional arc $\hat{a} = \text{sign}(z_a - u_a/2)(v, w)$ (if $z_a - u_a/2 = 0$, we do not introduce the arc $\hat{a}$) and set the flow on $\hat{a}$ to $x_{\hat{a}} = |z_a - u_a/2|$ and its cost to $c_{\hat{a}} = \lceil t/|z_a - u_a/2| \rceil$ for some t.[6]

3. The dual variables are set as $y_v, y_w = 0$ and $y_{vw} = -\lfloor 2t/u_a \rfloor$, which determines $s_{\acute{a}} = c_a + \lfloor 2t/u_a \rfloor$, $s_{\grave{a}} = \lfloor 2t/u_a \rfloor$ and $s_{\hat{a}} = c_{\hat{a}} = \lceil t/|z_a - u_a/2| \rceil$.

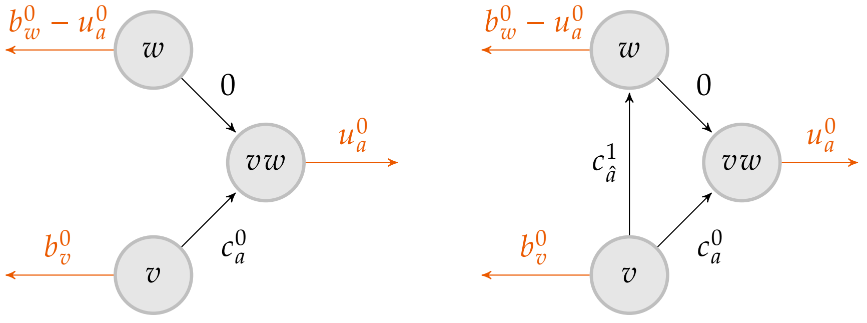

FIGURE 3.1: In order to balance the $x_a s_a$, we introduce the arc $\hat{a} = (v, w)$ with high cost $c_{\hat{a}}$ and reroute flow along it. The direction of $\hat{a}$ depends on a tree solution z in the original graph. It is flipped if $z_a \le u_a/2$. All arcs in the middle and the right graph have infinite capacity. Only the costs are shown.

We refer the reader to Figure 3.1 for a visualization of this construction. Note that the above routine can be implemented in time $O(m)$. We can now show that x, y, s are arbitrarily close to the central path.

Lemma 3.9. *Let x, y, s be as described above. Then, it holds:*

1. *For any $t > 0$, it holds that $x_a s_a \in [t - \beta\gamma UC, t + \beta\gamma UC]$ for all $a \in A$.*

2. *Let $t := 3m\beta\gamma UC/\delta$ and $\mu_0 := t - \beta\gamma UC$, then $\|\sigma\|_1 \le \delta$.*

[5]We remark that after scaling u with a factor of 2, we can ensure that x is integral.

[6]Note that a sufficiently large choice of t guarantees that no flow is routed across $\hat{a}$ in an optimal solution. More precisely, for $t \ge 2\beta\gamma mUC$, we obtain $c_{\hat{a}} \ge t/(2\beta U) \ge m\gamma C \ge \mathbb{1}^T c^0$, where we used that $z_a \le \|b^0\|_1/2$, and thus the cost of $\hat{a}$ is at least the cost of any path in G^0 and introducing $\hat{a}$ does not change the optimum of the problem.

Proof. 1. Let $a \in A_0$ be any arc in G_0, then $x_{\acute{a}} = x_{\grave{a}} = u_a/2$. It follows that,

$$x_{\acute{a}} s_{\acute{a}} = \frac{u_a}{2}\Big(c_{\acute{a}} + \Big\lfloor \frac{2t}{u_a} \Big\rfloor\Big) \le t + \frac{u_a c_a}{2} \le t + \frac{\beta\gamma UC}{2}, \text{ and } x_{\acute{a}} s_{\acute{a}} \ge t + \frac{u_a c_a}{2} - \frac{u_a}{2} \ge t;$$

$$x_{\grave{a}} s_{\grave{a}} = \frac{u_a}{2}\Big(c_{\grave{a}} + \Big\lfloor \frac{2t}{u_a} \Big\rfloor\Big) \le t, \text{ and } x_{\grave{a}} s_{\grave{a}} = \frac{u_a}{2}\Big(c_{\grave{a}} + \Big\lfloor \frac{2t}{u_a} \Big\rfloor\Big) \ge t - \frac{u_a}{2} \ge t - \beta U$$

For the newly introduced arc $\hat{a}$, we obtain

$$x_{\hat{a}} s_{\hat{a}} \ge \Big|z_a - \frac{u_a}{2}\Big| \frac{t}{|z_a - \frac{u_a}{2}|} = t \quad \text{and} \quad x_{\hat{a}} s_{\hat{a}} \le \Big|z_a - \frac{u_a}{2}\Big| \Big(\frac{t}{|z_a - \frac{u_a}{2}|} + 1\Big) \le t + \beta U.$$

2. For the sake of readability, let us denote $\alpha := \beta\gamma UC$. From the first part, it follows that $\frac{x_a s_a}{\mu_0} \in [1, (t+\alpha)/(t-\alpha)] = [1, 1 + 2\alpha/(t-\alpha)]$ for $a \in A$. This yields $\frac{x_a s_a}{\mu_0} \in [1, 1 + \delta/m]$ with $t \ge 3m\alpha/\delta$ and hence

$$\|\sigma\|_1 = \sum_{a \in A} \Big|\frac{x_a s_a}{\mu_0} - 1\Big| \le \sum_{a \in A} \frac{\delta}{m} = \delta. \qquad \square$$

The dual variables defined above are integral, whereas the primal variables are half-integral. Integrality can be achieved via a simple scaling of b and u with factor 2. From the above lemma and the lower bound on t that ensures that the optimum is not changed due to the introduction of the new arcs $\hat{a}$, we obtain that choosing $\mu^{(0)} = t - \beta\gamma UC \le 9m_0\beta\gamma UC/\delta$ is feasible.

3.3.2 Integer Interior Point Algorithm

We have seen how to construct (in linear time) initial interior points for an auxiliary instance with the same optimum that fulfill the central path condition $\|\sigma\|_1 \le \delta$ for $\mu_0 \le 3m\beta\gamma UC/\delta$. The idea of the path following method is to decrease the current μ by roughly a factor of $1 - \delta/\sqrt{m}$, more precisely, we set $\mu \leftarrow \lceil (1-\tau)\mu \rceil$, where $\tau = \delta/\sqrt{m}$, as long as we have not achieved a solution that is a proxy for the optimum of (G, b, c) yet. Then, we restore closeness to the central path w.r.t. the new μ and the minor in the so-called *centering step*. Observe that closeness to the central path together with the absence of tiny x_a and s_a in the minor implies that x_a and s_a are bounded by $(1+\delta)\mu/\varepsilon$. Thus the size of μ_0 yields an upper bound on the numbers that have to be considered in this algorithm. A pseudocode implementation can be found in Algorithm 3.1.

The following theorem is a more precise formulation of Theorem 3.1 and the rest of this chapter is dedicated to prove it. Recall that $\beta := \gcd(b)$ and $\gamma := \gcd(c)$ and observe that if a given input instance does not satisfy the assumption in the theorem of having sufficiently large gcd's, scaling down the instance by dividing by $\gcd(b)$ and $\gcd(c)$, respectively, and scaling them up by sufficiently large numbers that satisfy the above theorem is a feasible approach. This shows that Theorem 3.1 follows from Theorem 3.10. Moreover, note that the running time of our algorithm only scales logarithmically with β and γ and thus up-scaling to meet the condition from the theorem can be achieved at low cost.

Theorem 3.10. *Let (G, b, c) be a min-cost flow instance with $b \in \beta\mathbb{Z}^n$ and $c \in \gamma\mathbb{Z}^m$. There is a randomized algorithm to compute $x, s \in \mathbb{Z}^m$ s.t. $x/\beta, s/\gamma$ is a proxy for the optimum of $(G, b/\beta, c/\gamma)$ in $\tilde{O}(m^{3/2}\log(UC))$ time with high probability using only integer arithmetic on numbers bounded by $2^8 m^3 \gamma UC$, provided that $\beta \ge 2^8 m^3$ and $\gamma \ge 2^{15} m^4 \beta UC$.*

Algorithm 3.1: PathFollow

```
Input  : $(G, b, c)$ with $\gcd(b) = \beta$ and
         $\gcd(c) = \gamma$, feasible interior
         $x, s \in \mathbb{Z}^m$, and $\mu > 0$ s.t.
         $\|\sigma\|_1 := \sum_{a \in A} |\frac{x_a s_a}{\mu} - 1| \leq \delta$.
Output: Vectors $x, s$ s.t. $x/\beta$ and $s/\gamma$ yield a
         proxy for the optimum of
         $(G, b/\beta, c/\gamma)$.

repeat
    $H = $ Minor $(G, x, s)$
    $\mu \leftarrow \lceil (1-\tau)\mu \rceil$, where $\tau = \delta/\sqrt{m}$
    $x, s = $ CenteringStep $(H, x, s, \mu)$
until $\sum_{a \in A_H} x_a s_a < (1-\varepsilon)^2 \beta\gamma$
return $x, s$
```

Algorithm 3.2: Minor (G, x, s)

```
Let $\varepsilon := \frac{1-\delta}{1+\delta}$.
for $a \in A$ do
    if $x_a < \frac{\varepsilon\beta}{m}$ then remove $a$
     from $G$
    if $s_a < \frac{\varepsilon\gamma}{m}$ then contract $a$ in
     $G$
return $G$
```

Note that $\mu \geq \frac{x^T s}{m(1+\delta)} \geq \frac{(1-\varepsilon^2)\beta\gamma}{m(1+\delta)} \geq \frac{\beta\gamma}{2m}$ for $\delta \leq 1/8$ and $\varepsilon \leq 1/2$ holds during the execution of the algorithm. Thus

$$\mu' := \left\lceil \left(1 - \frac{\delta}{\sqrt{m}}\right) \mu \right\rceil \leq \left(1 - \frac{\delta}{\sqrt{m}}\right) \mu + 1 \leq \left(1 - \frac{\delta}{\sqrt{m}}\right) \mu + \frac{2m\mu}{\beta\gamma} \leq \left(1 - \frac{\delta}{2\sqrt{m}}\right) \mu$$

provided that $\beta\gamma \geq 4m^{3/2}/\delta$, which is satisfied for β, γ fulfilling the conditions of the theorem. Let us denote with $\mu^{(0)}$ the initial value of μ and likewise let $\mu^{(f)}$ be the final value of μ. Then after at most $t := \lceil 2\sqrt{m} \log(\mu^{(0)}/\mu^{(f)})/\delta \rceil$ many iterations, it holds that $\mu^{(0)}$ is reduced to $\mu^{(f)}$, since for the value t' for which $(1 - \frac{\delta}{2\sqrt{m}})^{t'} \mu^{(0)} = \mu^{(f)}$, we have that

$$t' = \log\left(\frac{\mu^{(f)}}{\mu^{(0)}}\right) \left[\log\left(1 - \frac{\delta}{2\sqrt{m}}\right)\right]^{-1} \leq \left\lceil \log\left(\frac{\mu^{(0)}}{\mu^{(f)}}\right) \frac{2\sqrt{m}}{\delta} \right\rceil = t.$$

Since $\mu^{(0)}/\mu^{(f)} \leq 2m^2 UC/\delta$, we can thus conclude that Algorithm 3.1 terminates after at most $O(\sqrt{m}\log(nUC))$ many iterations.

It thus remains to describe the following: (1) How to implement the centering step in nearly-linear time with high probability using only integer arithmetic given the above assumption on the gcds? This will be discussed in detail in Section 3.4. For now it is only important that it maintains closeness to the central path w.r.t. the arcs in the minor. For the sake of presentation, we did not maintain the y-variables that can be used to restore the reduced costs on the arcs that have been deleted. But this can be easily achieved. Moreover, one can extend the cycles mentioned above from the minor through the contracted nodes to cycles in the original graph such that one can also keep track of the updates to maintain a feasible solution in the original graph. (2) How to implement the crossover step? Notice that from Algorithm 3.1, we obtain a proxy for the optimum of (G, b, c). However, the variables of the arcs in the minor describe an interior point of a slightly perturbed instance. We describe how to handle the situation in Theorem 3.11.

3.3.3 Crossover

We repeat the crossover technique from [BK14, Section 2]. We note that we have used a similar approach in Chapter 2 as dual step. The crossover takes an instance $(G, \tilde{b}, \tilde{c})$ and primal/dual feasible points $\tilde{x}, (\tilde{y}, \tilde{s})$. Given that $\tilde{c}$ is integral, it outputs an integral dual tree solution y^*, s^* with $b^T y^* \geq b^T \tilde{y}$. In a nutshell, the crossover algorithm works as follows. It iteratively constructs nested cuts $\{s\} = S^1 \subset S^2 \subset \ldots \subset S^n = V$, starting with an arbitrary node s. During iteration i, the algorithm modifies the potentials y along the cut S^i ensuring that $b^T y$ does not decrease. This is done by incrementing the potentials of all nodes in S^i, if $b(S^i) \geq 0$, and by decrementing them otherwise. The potentials are increased (decreased) until the reduced cost of at least one arc a on the cut $S, V \setminus S$ becomes 0. Then, the node adjacent to S^i through a is being added to S^i. We note that the amount by which the potentials inside of S^i are changed relatively to the nodes outside of S^i, is exactly equal to the reduced cost of the arc a.

Theorem 3.11. *Let (G, b, c) be a min-cost flow instance with $b \in \beta\mathbb{Z}^n$ and $c \in \gamma\mathbb{Z}^m$. Given $x, s \in \mathbb{Z}^m$ s.t. $x/\beta, s/\gamma$ is a proxy for the optimum of $(G, b/\beta, c/\gamma)$, a pair of primal-dual optimum solutions x^*, s^* of (G, b, c) can be computed in $O(m^{3/2} \log \frac{n^2}{m} \log U)$ time.*

Proof. Let $\bar{x} = x/\beta, \bar{s} = s/\gamma$ as well as $\bar{b} = b/\beta$ and $\bar{c} = c/\gamma$. Consider the projection $\tilde{x}$ of $\bar{x}$ to the subspace with $\tilde{x}_a = 0$ for all $a \in D$ and the projection $\tilde{s}$ of $\bar{s}$ to the subspace with $\tilde{s}_a = 0$ for all $a \in C$. Note that these projections are primal/dual feasible for the ε-perturbation $(G, \tilde{b}, \tilde{c})$ of (G, b, c), where $\tilde{b} = \bar{b} - AI_D x$ and $\tilde{c} = \bar{c} - I_C s$ and their duality gap satisfies $\tilde{x}^T \tilde{s} = \sum_{a \in A_H} \tilde{x}_a \tilde{s}_a < (1 - \varepsilon)^2$. Given feasible $\bar{s}$, we can compute corresponding feasible $\bar{y}$ in linear time by propagation from the root to the leaves of an arbitrary spanning tree. The same holds for $\tilde{y}$ and $\tilde{s}$. If we perform the crossover procedure with $\tilde{y}, \tilde{s}$ in $(G, \tilde{b}, \tilde{c})$, we obtain a tree solution w.r.t. some spanning tree T. By Theorem 3.8, this spanning tree also yields a dual feasible solution for (G, b, c). Moreover, the two admissible networks are combinatorially the same. If the admissible network yields a primal feasible solution w.r.t. b, we obtain a pair of primal/dual feasible solutions satisfying complementary slackness and hence the sought optimum solutions x^*, s^*. Suppose the contrary for a contradiction. Then, there is a cut $S \subset V$ s.t. $b(S) > 0$ and there are no ingoing arcs with vanishing reduced costs, i.e., all ingoing arcs have reduced costs of at least 1. Thus, y_v could be safely increased by 1 for all $v \in S$. Moreover, $\tilde{y}_v$ could be safely increased by $1 - \varepsilon$. Thus, the dual objective $\tilde{b}^T \tilde{y}$ would increase by at least $(1 - \varepsilon)^2$ because $\mathbb{1}_S^T \tilde{b} = \mathbb{1}_S^T b - \mathbb{1}_S AI_D x \geq 1 - \varepsilon$. But this contradicts $\tilde{x}^T \tilde{s} < (1 - \varepsilon)^2$. The crossover can be computed in $O(m + n \log n)$ and the transshipment problem in the admissible network that needs to be solved in order to obtain the corresponding primal solution can be solved by a max-flow computation in $O(m^{3/2} \log \frac{n^2}{m} \log U)$ using the algorithm of Goldberg and Rao [GR97]. □

Note that the additional max-flow computation can be avoided, by using the isolation lemma [MVV87] in order to make the optimal solution unique [DS08, Lemma 3.12]. However, the perturbation of the cost vector yields non-integral values and, hence, an additional scaling would be necessary in order to make the input integral. Furthermore, this strategy would introduce a new source of randomization that we prefer to avoid.

We summarize what we have established: For an input instance (G, b, c), using Algorithm 3.1, we can obtain primal/dual feasible solutions x, s to an ε-perturbed solution $(G, \tilde{b}, \tilde{c})$ of (G, b, c) that can be used in order to compute an optimal primal/dual solution pair for the input instance.

3.4 Centering Step

In this section, we describe how to implement the centering step by using a variant of the electrical flow solver from [Kel+13]. We would like to stress the fact that we do not rely on the numerical stability analysis of [Kel+13] that would involve an additional scaling of the current sources, instead the current sources that we use can be directly defined as integer values dependent on the iterates x, s.

The centering step takes as input the minor H of G, the variables x, s and the parameter $\mu' = \lceil (1-\tau)\mu \rceil$, with x, s, μ fulfilling $\|\sigma\|_1 \leq \delta$, where $\sigma_a = x_a s_a/\mu - 1$ for each $a \in A_H$. For simplicity, we will denote the vectors x and s as vectors over A_H, instead of A. The goal of CenteringStep is to update x and s to $x + \Delta x, s + \Delta s$ such that $\|\sigma'\|_1 \leq \delta$ holds, where $\sigma'_a = x'_a s'_a/\mu' - 1$. Note that s uniquely defines feasible potentials y whose update we denote with Δy. Since we want the new iterates to be interior solutions again, we require

$$A_H \Delta x = 0,\ A_H^T \Delta y + \Delta s = 0 \text{ and } x + \Delta x > 0 \text{ as well as } s + \Delta s > 0. \tag{3.3}$$

Moreover, since we want $\|\sigma'\|_1 \leq \delta$, the idea is to require that $(x + \Delta x)(s + \Delta s) \approx \mu'$, more precisely we will require $\eta := \frac{1}{\mu'}\left[S\Delta x + X\Delta s + Xs\right] - \mathbb{1}$ to be small, i.e., we omit the quadratic term $\Delta x \Delta s$. Here $X = \operatorname{diag}(x)$ and $S = \operatorname{diag}(s)$, respectively.

Such $\Delta x, \Delta s$ satisfying (3.3) with $\eta = 0$ can be obtained from the solution of a linear equation system. This can be seen as follows: Multiplying the equations $\eta = 0$ by $\mu' A_H S^{-1}$ from the left yields $A_H \Delta x + A_H S^{-1} X \Delta s + A_H x - \mu' A_H S^{-1} \mathbb{1} = 0$. Using $A_H \Delta x = 0$ and $A_H x = b'$ yields $A_H S^{-1} X \Delta s = \mu' A_H S^{-1} \mathbb{1} - b'$.[7] Multiplying $A_H^T \Delta y + \Delta s = 0$ by $A_H X S^{-1}$ from the left and plugging in the above equation for $A_H S^{-1} X \Delta s$ yields

$$A_H X S^{-1} A_H^T \Delta y = -A_H X S^{-1} \Delta s = b' - \mu' A_H S^{-1} \mathbb{1},$$

which is an equation system with n_H equations in n_H variables Δy. Note that the values of Δx and Δs can be reconstructed from the solutions Δy using the equations $A_H^T \Delta y + \Delta s = 0$ and $\eta = 0$. The matrix $A_H X S^{-1} A_H^T$ is the weighted graph laplacian of H with weights XS^{-1}. Solving such an equation system is equivalent to computing a so-called electrical flow in the network H with current sources $\mu' A_H S^{-1} \mathbb{1} - b' \in \mathbb{R}^{n_H}$ and resistances $X^{-1}s \in \mathbb{R}^{m_H}_{>0}$.

We will take exactly this approach, but with the following two crucial differences. (1) We will have to be satisfied with fulfilling the equation system approximately, i.e., not having $\eta = 0$ but only $\|\eta\|_1 \leq \delta/4$. We will achieve this by using an approximate electrical flow computation. (2) We want to compute with integer arithmetic only, hence defining the resistances and current sources as described above is not feasible, instead we will use the rounded resistances $r_a = \lceil s_a/x_a \rceil$ and the rounded current sources $A_H \varphi^0$, where $\varphi^0_a = x_a - \lceil \mu'/s_a \rfloor$ for all $a \in A_H$. We will denote $R := \operatorname{diag}(r)$. Note that, due to the assumption on the size of β and γ and the closeness to the central path, it holds that $s_a/x_a \geq 1$, see the proof of Lemma 3.15 for a precise derivation.

We need some additional notation concerning electrical flows.[8] Let φ be a given flow and let T be a spanning tree of H. For any $a = (v, w) \in A_H \setminus T$, we define $C_a := \{a\} \cup P(w, v)$, where $P(w, v) \subseteq A \cup -A$ is the unique (undirected) path in T between w and v. Similarly, $\chi(a) \in \{-1, 0, 1\}^m$ is the corresponding characteristic vector of the cycle C_a, i.e., $\chi(a)_b = 1$, if $b \in A \cap C_a$, $\chi(a)_b = -1$, if $b \in -A \cap C_a$, and $\chi(a)_b = 0$ otherwise. Moreover, $r(C_a) := \sum_{a' \in C_a} r_{a'}$ and $\operatorname{tcn}(T) := \sum_{a \in A_H \setminus T} \frac{r(C_a)}{r_a}$ is the *tree condition number of* T. The *tree induced voltages* are defined as $\pi_v := \sum_{a \in P(v_0, v)} \varphi_a r_a$, where v_0 is an arbitrary root of the spanning tree T.

As previously mentioned, we use an adaption, of the electrical flow solver from [Kel+13], where the authors show how to compute an approximate electrical flow φ and corresponding node voltages π in nearly linear time. See Algorithm 3.3 for a pseudo-code implementation and note that up to the choice of the termination criterion, the choice of the initial current, and the rounded update value our algorithm is identical to the one from [Kel+13]. In the

[7] Note that $b' \in \mathbb{R}^{n_H}$ is a perturbed and restricted version of b due to the contraction and deletion of arcs from G that lead to H.

[8] For a more detailed depiction of electrical flows see for example [Bol98; Kel+13].

algorithm, we need to compute a spanning tree and in order to guarantee fast convergence this spanning tree needs to be of low *stretch*:

Definition 3.12. *Given a weighted undirected graph $G = (V, E, r)$ and a spanning tree $T \subseteq E$ of G. For two nodes $v, w \in V$, let $\mathrm{dist}_T(v, w) = \sum_{e \in P_{v,w}} r_e$ denote the length of the unique path $P_{v,w}$ from v to w in T. The stretch of an edge $e = (v, w)$ with respect to T is defined by $\mathrm{str}_T(e) = \mathrm{dist}_T(v, w)/r_e$. The stretch of the tree T is defined as $\mathrm{str}(T) = \sum_{e \in A \setminus T} \mathrm{str}_T(e) = \sum_{e \in A \setminus T} \mathrm{dist}_T(v, w)/r_e$.*

The current record bounds concerning low stretch spanning trees is achieved by the algorithm of Abraham and Neiman [AN12]. They show how to compute a tree of stretch $O(m \log n \log \log n)$ in $O(m \log n \log \log n)$ time. We furthermore remark that, as in the algorithm from [Kel+13], the flow and voltage updates should be performed using a special tree data structure [Kel+13, Section 5], which allows updating the flow in $O(\log n)$. Moreover, the update of the dual variables s should be performed only after every m iterations, resulting in $O(1)$ amortized time per iteration of the centering step for the update of the dual variables. Thus, in total, every iteration takes $O(\log n)$ amortized time.

Algorithm 3.3: CENTERINGSTEP (H, x, s, μ)

```
Define $r_a := \lceil \frac{s_a}{x_a} \rceil$, $\varphi_a^0 := x_a - \lceil \frac{(1-\tau)\mu}{s_a} \rfloor$ for $a \in A_H$.
Let $T$ :=spanning tree and $p_a := \frac{r(C_a)}{\mathrm{tcn}(T) r_a} \forall a \in A_H \setminus T$
$\varphi := \varphi^0$, $s' = s$ and $x' = x$
**while** $\sum_{a \in A_H} |x'_a s'_a - \mu| \geq \delta\mu$ **do**
    Sample $a \in A_H \setminus T$ according to $p$
    $\alpha(a) := \lceil \frac{-\sum_{a' \in C_a} r_{a'} \varphi_{a'}}{r(C_a)} \rfloor$, $x' \leftarrow x' + \alpha(a) \cdot \chi(a)$, $\varphi \leftarrow \varphi + \alpha(a) \cdot \chi(a)$
    occasionally: compute tree induced voltages $\pi$ for $\varphi$, $s' := s - A_H^T \pi$
**return** $x', s'$
```

We will assume δ to be set to $1/8$, $\mu^{(0)} \leq 24 m\beta\gamma UC$ as we show to be valid in Section 3.3.1, and we will denote with x, s, μ the values of the variables at the beginning of the CENTERINGSTEP and the update values we define by $\Delta x := \varphi - \varphi^0$ and $\Delta s := -A_H^T \pi$. We remark that, for our choice of $\delta = 1/8$, evaluating the condition of the while-loop can be done easily with integer arithmetic (by comparing the two integral values resulting from multiplying the inequality by 8).

Moreover, define τ' by $(1 - \tau')\mu = \lceil (1 - \tau)\mu \rceil = \mu'$ and recall that we set $\varepsilon := \frac{1-\delta}{1+\delta}$. Note that from arc contraction and deletion as well as closeness to the central path, we obtain lower and upper bounds on x and s for arcs in the minor, respectively:

$$x_a \in \left[\frac{\varepsilon\beta}{m}, \frac{(1+\delta)\mu m}{\varepsilon\gamma} \right] \quad \text{and} \quad s_a \in \left[\frac{\varepsilon\gamma}{m}, \frac{(1+\delta)\mu m}{\varepsilon\beta} \right] \quad \text{for all } a \in A_H. \tag{3.4}$$

We first of all show that the updates are feasible moves. Primal feasibility follows since Δx is a circulation. The update for potentials can be computed in such a way that the dual constraints, except non-negativity of s, are fulfilled as well. For non-negativity of x and s, consider the following lemma – in fact the closeness to the central path already implies non-negativity for x and s, respectively.

Lemma 3.13. *If $\|\sigma'\|_1 < \delta$, then $x_a + \Delta x_a > 0$ and $s_a + \Delta s_a > 0$ for all $a \in A_H$.*

Proof. If $\|\sigma'\|_1 < \delta$, we must have $|(x_a + \Delta x_a)(s_a + \Delta s_a)/\mu' - 1| < \delta$ for all $a \in A_H$. Thus $(x_a + \Delta x_a)(s_a + \Delta s_a) > 0$ for all $a \in A_H$. Assume $x_a + \Delta x_a \leq 0$ and $s_a + \Delta s_a \leq 0$ for some $a \in A_H$. Then

$$0 \geq s_a(x_a + \Delta x_a) + x_a(s_a + \Delta s_a) = \mu' + s_a x_a > 0, \quad \text{a contradiction.} \qquad \square$$

In what follows, we will denote with $\varphi^{(t)}$ the value of φ in iteration t of CenteringStep and with $\text{gap}^{(t)}$ the corresponding value of the gap. We proceed by upper bounding the energy of the initial flow. Since the energy of any flow $\varphi^{(t)}$ is upper bounded by the initial energy, see [Kel+13, Lemma 4.2], the upper bound follows for the energy of all flows.

Lemma 3.14. *It holds that* $\|\varphi^{(t)}\|_R^2 \leq 10\mu m_H$.

Proof. Since the energy of the initial flow φ^0 is at least the energy of $\varphi^{(t)}$ for any t, we obtain

$$\begin{aligned}\|\varphi^{(t)}\|_R^2 \leq \|\varphi^0\|_R^2 &\leq \sum_{a\in A_H} \frac{2s_a}{x_a}\left[x_a^2 - 2x_a\left\lceil\frac{\mu'}{s_a}\right\rfloor + \left\lceil\frac{\mu'}{s_a}\right\rfloor^2\right] \leq \sum_{a\in A_H} 2x_a s_a - 2\mu' + \frac{8\mu'^2}{x_a s_a} \\ &\leq m_H\left[2(1+\delta)\mu - 2(1-\tau')\mu + \frac{8\mu(1-\tau')^2}{(1-\delta)}\right] \leq m_H\left[\frac{1}{2}\mu + \frac{8\mu(1-\tau')^2}{(1-\delta)}\right] \leq 10\mu m_H,\end{aligned} \tag{3.5}$$

where we used $x_a s_a/\mu \geq 1-\delta$, $\delta \leq \frac{1}{8}$, $\tau' \leq \tau = \delta/\sqrt{m} \leq \delta$, and $1-\tau' \leq 1$ for the enumerator. □

In the following lemma we show that, assuming sufficiently large β and γ, as well as small $\text{gap} := \varphi^T R\varphi - 2\pi^T A_H\varphi^0 + \pi^T A_H R^{-1} A_H^T \pi$ of the electrical flow and voltages yields small one-norm of η.

Lemma 3.15. *Assume* $\beta \geq \frac{2^4 m^2}{\delta}$ *and* $\gamma \geq \frac{2^{17} m^5 \beta U C}{\delta}$. *Let* $\delta \leq 1/8$ *and* $\varphi \in \mathbb{R}^{m_H}$ *and* $\pi \in \mathbb{R}^{n_H}$ *such that* $\text{gap} < \frac{2^{-8}\delta^2\mu}{m_H}$. *Then it holds that* $\|\eta\|_1 \leq \frac{\delta}{4}$, *where* $\eta := \frac{1}{\mu'}[S\Delta x + X\Delta s + Xs] - \mathbb{1}$.

Proof. Let us denote with $D_a := \lceil\frac{\mu'}{s_a}\rfloor - \frac{\mu'}{s_a}$ and $E_a := \lceil\frac{s_a}{x_a}\rceil - \frac{s_a}{x_a}$ the errors introduced by the rounding in the algorithm. Then,

$$\begin{aligned}\|\eta\|_1 &= \sum_{a\in A_H}\frac{x_a}{\mu'}\left|\frac{s_a}{x_a}\Delta x_a + \Delta s_a + s_a - \frac{\mu'}{x_a}\right| \\ &= \sum_{a\in A_H}\frac{x_a}{\mu'}\left|\frac{s_a}{x_a}(\varphi_a - \varphi_a^0) - (\pi_w - \pi_v) + s_a - \frac{\mu'}{x_a}\right| \\ &= \sum_{a\in A_H}\frac{x_a}{\mu'}\left|r_a\varphi_a - (\pi_w - \pi_v) + \frac{s_a}{x_a}D_a - E_a\varphi_a\right| \\ &\leq \frac{\|X(R\varphi - A^T\pi)\|_1}{\mu'} + \frac{\|s\|_1}{2\mu'} + \sum_{a\in A_H}\frac{x_a|\varphi_a|}{\mu'},\end{aligned} \tag{3.6}$$

since $|E_a| < 1$ and $|D_a| \leq 1/2$. Moreover, note that using the upper and lower bounds on x and s in (3.4) and the fact that $\tau' \leq \tau \leq \delta$ yields the following multiplicative errors introduced by the rounding:

$$r_a = \left\lceil\frac{s_a}{x_a}\right\rceil \in \left[\frac{s_a}{x_a}, 2\frac{s_a}{x_a}\right] \quad\text{and}\quad \left\lceil\frac{\mu'}{s_a}\right\rfloor \in \left[\frac{\mu'}{2s_a}, \frac{2\mu'}{s_a}\right], \tag{3.7}$$

where we need that $s_a/x_a \geq 1$, which is guaranteed by $\gamma \geq 2^6 m^3 \beta UC$ and $\beta \geq 4m$. Let us now first show how to bound the last summand in (3.6): Note that

$$\sum_{a\in A_H} \frac{x_a|\varphi_a|}{\mu'} \leq \frac{1}{\mu'} \sum_{a\in A_H} x_a \sqrt{\frac{x_a}{s_a}} \cdot \sqrt{\frac{s_a}{x_a}} |\varphi_a| \leq \frac{1}{\mu'} \sqrt{\sum_{a\in A_H} \frac{x_a^3}{s_a}} \sqrt{\sum_{a\in A_H} \frac{s_a}{x_a} |\varphi_a^2|}$$
$$\leq \frac{1}{\mu'} \sqrt{\sum_{a\in A_H} \frac{x_a^3}{s_a}} \|\varphi\|_R.$$

We have shown in Lemma 3.14 that $\|\varphi\|_R^2 \leq 10\mu m_H$. The upper bound for x and the lower bound for s in (3.6) yield $\sqrt{\sum_{a\in A_H} x_a^3/s_a/\mu'} \leq \sqrt{(1+\delta)\mu m^5}/(\varepsilon^3\gamma^2)$, using $\tau' \leq \delta$. Putting the two bounds together, we obtain $\sqrt{10(1+\delta)}\mu m^3/(\varepsilon^3\gamma^2)$ as a bound on the third summand in (3.6). Now, for the first summand in (3.6), note that gap $= \sum_{a\in A_H\setminus T}[r_a\varphi_a - (\pi_w - \pi_v)]^2/r_a$, as shown in [Kel+13, Lemma 4.4]. Hence,

$$\frac{\|X(R\varphi - A^T\pi)\|_1}{\mu'} = \sum_{a\in A_H\setminus T} \frac{|r_a\varphi_a - (\pi_w - \pi_v)|}{\sqrt{r_a}} \frac{\sqrt{r_a}x_a}{\mu'} \leq \frac{\sqrt{\text{gap}\cdot\|X^2 r\|_1}}{\mu'} \leq \frac{2\sqrt{\text{gap}\cdot m_H}}{\sqrt{\mu}}$$

using the Cauchy-Schwarz-Inequality, $x_a s_a/\mu \leq 1+\delta$ and again $\tau' \leq \delta$. Finally, for the second summand in (3.6), note that $\frac{\|s\|_1}{2\mu'} \leq \frac{(1+\delta)\mu m^2}{2\mu'\varepsilon\beta} \leq \frac{m^2}{2\varepsilon^2\beta}$. Hence, plugging all bounds together into (3.6) yields

$$\|\eta\|_1 \leq \frac{2\sqrt{\text{gap}}\, m}{\sqrt{\mu}} + \frac{m^2}{2\varepsilon^2\beta} + \frac{\sqrt{10(1+\delta)}\mu m^3}{\varepsilon^3\gamma^2} \leq \frac{\delta}{8} + \frac{\delta}{16} + \frac{\delta}{16} \leq \frac{\delta}{4},$$

with $\beta \geq \frac{2^4 m^2}{\delta}$, $\gamma \geq \frac{2^{12} m^4 \beta UC}{\delta}$, gap $\leq \frac{2^{-8}\delta^2\mu}{m_H}$ and the upper bound on $\mu \leq \mu^{(0)} \leq 24m\beta\gamma UC$. □

Recall the definition of $\eta := \frac{1}{\mu'}[S\Delta x + X\Delta s + Xs] - \mathbb{1}$ from the previous lemma. We next show that a small one-norm of η actually implies the closeness to the central path of the new iterates x' and s', i.e., implies $\|\sigma'\|_1 \leq \delta$, where $\sigma'_a := \frac{x'_a s'_a}{\mu'} - 1$ for $a \in A_H$.

Lemma 3.16. *Let $\delta \leq 1/8$. If $\|\eta\|_1 \leq \frac{\delta}{4}$, then $\|\sigma'\|_1 \leq \delta$ where $\sigma'_a := \frac{x'_a s'_a}{\mu'} - 1$ for $a \in A_H$.*

Proof. By definition of η, it holds that $(1+\eta_a)\mu' = \Delta x_a s_a + x_a \Delta s_a + x_a s_a$, which yields

$$\Delta x_a \Delta s_a = \tfrac{1}{2x_a s_a}\left[((1+\eta_a)\mu' - x_a s_a)^2 - (\Delta x_a s_a)^2 - (x_a \Delta s_a)^2\right] \quad \text{for } a \in A_H. \tag{3.8}$$

For the one-norm of σ', we get the following estimate

$$\|\sigma'\|_1 = \sum_{a\in A_H} \left|\frac{x'_a s'_a}{\mu'} - 1\right| = \sum_{a\in A_H} \left|\frac{\Delta x_a \Delta s_a}{\mu'} + \eta_a\right| \leq \sum_{a\in A_H} \left|\frac{\Delta x_a \Delta s_a}{\mu'}\right| + \|\eta\|_1.$$

Using (3.8) and the triangle inequality, we obtain for the first summand that

$$\sum_{a\in A_H} \left|\frac{\Delta x_a \Delta s_a}{\mu'}\right| \leq \sum_{a\in A_H} \frac{[(1+\eta_a)\mu' - x_a s_a]^2}{2x_a s_a \mu'} + \frac{(\Delta x_a s_a)^2 + (x_a \Delta s_a)^2}{2x_a s_a \mu'}$$
$$= \sum_{a\in A_H} \frac{[(1+\eta_a)\mu' - x_a s_a]^2}{x_a s_a \mu'},$$

because Δx and Δs are orthogonal and thus the sum over all arcs in (3.8) yields the last equality. Now let τ' be such that $\mu' = (1-\tau')\mu$. Factoring out μ, plugging in the definition of μ' and σ_a and using the lower bound on $\frac{x_a s_a}{\mu}$ yields

$$\sum_{a\in A_H}\left|\frac{\Delta x_a \Delta s_a}{\mu'}\right| \leq \sum_{a\in A_H}\frac{\mu^2\left(\frac{(1+\eta_a)\mu'}{\mu}-\frac{x_a s_a}{\mu}\right)^2}{x_a s_a \mu'} \leq \sum_{a\in A_H}\frac{[(1-\tau')(1+\eta_a)-1-\sigma_a]^2}{(1-\tau')(1-\delta)}.$$

Expanding the square, re-grouping the terms and using $\mathbb{1}^T\sigma \leq \|\sigma\|_1 \leq \delta$, $-\mathbb{1}^T\eta \leq \|\eta\|_1$, the equivalence of norms and the Cauchy-Schwarz-Inequality for $-\eta^T\sigma \leq \|\eta\|_2\|\sigma\|_2$, yields

$$\sum_{a\in A_H}\left|\frac{\Delta x_a \Delta s_a}{\mu'}\right| \leq \frac{(1-\tau')^2\|\eta\|_1^2 + [2(1-\tau')\tau' + 2\delta(1-\tau')]\|\eta\|_1 + \delta^2 + 2\delta\tau' + \tau'^2 m}{(1-\tau')(1-\delta)}.$$

Using $\tau' \leq \tau = \frac{\delta}{\sqrt{m}} \leq \delta$ and $1-\tau' \leq 1$ yields

$$\|\sigma'\|_1 \leq \frac{\|\eta\|_1^2 + 4\delta\|\eta\|_1 + 4\delta^2 + (1-\tau')(1-\delta)\|\eta\|_1}{(1-\tau')(1-\delta)} \leq \frac{\|\eta\|_1^2 + (4\delta+1)\|\eta\|_1 + 4\delta^2}{(1-\delta)^2}$$

and $\|\eta\|_1 \leq \frac{\delta}{4}$ gives the result for any $\delta \leq \frac{1}{8}$. □

Note that Lemmas 3.15 and 3.16 together yield that the value of the gap is lower bounded by $2^{-8}\delta^2\mu/m_H$, if the termination criterion of CenteringStep does not hold. It remains to argue the convergence behavior of CenteringStep. This is achieved by the following lemma: The expected decrease in energy is a fraction of the gap provided that the gap fulfills the mentioned lower bound. The lemma can be seen as the equivalence of [Kel+13, Lemma 4.5], with the difference that we obtain an additional factor of 3/4 due to the rounding to the nearest integer in the definition of $\alpha(a)$. Again, in order for the rounding to not have a significant influence, we need to ensure β and γ to be sufficiently large.

Lemma 3.17. *It holds that* $\mathrm{E}[\|\varphi^{(t+1)}\|_R^2 - \|\varphi^{(t)}\|_R^2 | \operatorname{gap}^{(t)}] \leq \frac{-3\operatorname{gap}^{(t)}}{4\operatorname{tcn}(T)}$*, provided that* $\beta \geq 2^5 m^3/\delta$*,* $\gamma \geq 2^6 m^3 \beta U C$ *and* $\operatorname{gap}^{(t)} \geq 2^{-8}\delta^2\mu/m_H$*.*

Proof. Let a be the arc that is sampled in iteration t and let $\Lambda_a := \sum_{a'\in C_a} r_{a'}\varphi_{a'}^{(t)} = r_a\varphi_a^{(t)} - (\pi_w - \pi_v)$. Then $\varphi^{(t+1)} = \varphi^{(t)} - \alpha(a)\cdot\chi(a)$, where $\alpha(a) := \lceil\frac{\Lambda_a}{r(C_a)}\rfloor$ and $\chi(a)$ are as in the algorithm. The change in energy due to this update is $\|\varphi^{(t)} - \alpha(a)\cdot\chi(a)\|_R^2 - \|\varphi^{(t)}\|_R^2 = -2\alpha(a)\cdot\Lambda_a + \alpha(a)^2\cdot r(C_a)$. Thus, using the definition of the probabilities $p_a = \frac{r(C_a)}{\operatorname{tcn}(T) r_a}$ for $a \in A_H \setminus T$, it follows that

$$\begin{aligned}\mathrm{E}\left[\|\varphi^{(t+1)}\|_R^2 - \|\varphi^{(t)}\|_R^2 | \operatorname{gap}^{(t)}\right] &= \sum_{a\in A_H\setminus T}\frac{r(C_a)^2\alpha(a)^2}{\operatorname{tcn}(T) r_a} - \frac{2r(C_a)\alpha(a)\Lambda_a}{\operatorname{tcn}(T) r_a}\\ &= \sum_{a\in A_H\setminus T}\frac{\left[\Lambda_a - \alpha(a)r(C_a)\right]^2 - {\Lambda_a}^2}{\operatorname{tcn}(T) r_a}.\end{aligned}$$

Now, using $\operatorname{gap}^{(t)} = \sum_{A_H\setminus T}{\Lambda_a}^2/r_a$ and $r(C_a) \leq \|r\|_1$ for all $a \in A_H \setminus T$ for the right term in the enumerator and $|\Lambda_a - \alpha(a)r(C_a)| \leq r(C_a)/2$ as the rounding error for the left term, yields

$$\mathrm{E}\left[\|\varphi^{(t+1)}\|_R^2 - \|\varphi^{(t)}\|_R^2 | \operatorname{gap}^{(t)}\right] \leq \frac{1}{4\operatorname{tcn}(T)}\|r\|_1 \operatorname{tcn}(T) - \frac{\operatorname{gap}^{(t)}}{\operatorname{tcn}(T)} \leq \frac{1}{4\operatorname{tcn}(T)}\|r\|_1 nm - \frac{\operatorname{gap}^{(t)}}{\operatorname{tcn}(T)}.$$

Here, we used that $\operatorname{tcn}(T) \leq nm$. This bound follows from the fact that we can always use a minimum weight spanning tree as our choice for T. Using $r_a \leq 2s_a/x_a$, which follows from $\gamma \geq 2^6 m^3 \beta UC$, see (3.7) and the upper and lower bounds on x, s in (3.4), yields

$$\|r\|_1 nm \leq \frac{2(1+\delta)nm^4\mu}{\varepsilon^2\beta^2} \leq \frac{2(1+\delta)nm^4\mu}{\varepsilon^2}\frac{\delta^2}{2^{10}m^6} \leq \frac{(1+\delta)}{2\varepsilon^2}\frac{\delta^2\mu}{2^8 m} \leq \frac{\delta^2\mu}{2^8 m} \leq \operatorname{gap}^{(t)},$$

where we used $\beta \geq 2^5 m^3/\delta$. This yields $\mathrm{E}\left[\|\varphi^{(t+1)}\|_R^2 - \|\varphi^{(t)}\|_R^2 \,|\, \operatorname{gap}^{(t)}\right] \leq \frac{-3\operatorname{gap}^{(t)}}{4\operatorname{tcn}(T)}$. □

By applying a so-called multiplicative drift theorem, see for example [DG13; DJW10], we obtain the following result.

Lemma 3.18. *Let $\delta = 1/8$ and assume that $\beta \geq 2^5 m^3/\delta$, $\gamma \geq 2^6 m^3 \beta UC$, then the expected number of iterations that* CenteringStep *takes is $\tilde{O}(m)$. Moreover,* CenteringStep *terminates after $O(m \log^2 n \log\log n)$ many iterations with probability $1 - m^{-c}$ for any constant $c > 0$.*

Proof. Let $\varphi^{\text{opt}} = \operatorname{argmin}\{\|\varphi\|_R^2 : A_H\varphi = \chi\}$, where $\chi = A_H\varphi^0$. Consider the random process of sampling non-tree arcs from CenteringStep and let S be the set of values $\|\varphi\|_R^2 - \|\varphi^{\text{opt}}\|_R^2$ where φ is the flow resulting from any sequence of non-tree arc samples. Define $S' := \{s \in S : s \geq \rho\}$, where $\rho := 2^{-8}\delta^2\mu/m_H$ and note that S' is finite and moreover the minimum $s_{\min}$ of S' is lower bounded by ρ. Now, for $t \in \mathbb{Z}_{\geq 0}$, define

$$X^{(t)} := \begin{cases} \|\varphi^t\|_R^2 - \|\varphi^{\text{opt}}\|_R^2 & \text{if } \operatorname{gap}^{(t)} \geq \rho \\ 0 & \text{otherwise.} \end{cases}$$

as a random variable over $S' \cup \{0\}$ and let T be the random variable that denotes the first time $t \in \mathbb{Z}_{\geq 0}$ where $X^{(t)} = 0$. Note that the random variable T is an upper bound on the number of iterations of CenteringStep, since $\operatorname{gap}^{(t)} < \rho$ implies that the algorithm terminates, due to Lemmata 3.15 and 3.16. We distinguish two cases in order to bound $\mathrm{E}[X^{t+1} - X^t | X^t = s]$ for any $s \in S'$. First assume $X^{(t+1)} = 0$, then $\mathrm{E}[X^{t+1} - X^t | X^t = s] = \mathrm{E}[-X^t | X^t = s] = -s$. Second, if $X^{(t+1)} \neq 0$, then $\mathrm{E}[X^{t+1} - X^t | X^t = s] \leq \frac{-3}{4\operatorname{tcn}(T)}s$, since $X^t \leq \operatorname{gap}^{(t)}$ using Lemma 3.17.

In summary, $\mathrm{E}[X^{t+1} - X^t | X^t = s] \leq \frac{-3}{4\operatorname{tcn}(T)}s$ for all $s \in S'$ and hence applying the multiplicative drift theorem [DJW10, Theorem 3] yields

$$\mathrm{E}[T | X^{(0)} = s_0] \leq \tfrac{1+\ln(s_0/\rho)}{3/(4\operatorname{tcn}(T))} = O(\operatorname{tcn}(T)\log(n)) = O(m \log^2 n \log\log n),$$

where we used that $s_0 \leq 10\mu m_H$, see Lemma 3.14. In addition, using [DG13, Theorem 5], we obtain the tail bound $\Pr[T \geq \frac{\lambda+\ln(s_0/\rho)}{3/(4\operatorname{tcn}(T))}] \leq \exp(-\lambda)$ for any λ. Hence, we conclude that $\Pr[T = \omega(m \log^2 n \log\log n)] \leq m^{-c}$ for any $c > 0$. □

We summarize the results so far: Algorithm 3.1 takes $O(\sqrt{m}\log(nUC))$ many iterations in each of which it calls CenteringStep that needs $O(m \log^2 n \log\log n)$ iterations with probability $1 - O(m^{-c})$ each of which takes polylog n time. Thus, after applying the union bound over all iterations, we obtain that Algorithm 3.1 runs in $\tilde{O}(m^{3/2}\log(UC))$ time with high probability.

3.5 Conclusion

We conclude by deducing the precise bound on the size of the numbers that the algorithm has to deal with: Note that the voltages, the values of the α, as well as the currents φ are each bounded in absolute value by $\|R\varphi\|_1$. By using the Cauchy-Schwarz-Inequality, we obtain $\|R\varphi\|_1 \leq \sqrt{\|r\|_1} \cdot \|\varphi\|_R$. The first factor can be bounded by $\sqrt{2(1+\delta)\mu_0 m^3/\varepsilon\beta}$ using the bounds from (3.7) and (3.4). The second, analogously to the estimation in (3.5), can be bounded by $\sqrt{10\mu m}$. Together with $\mu \leq \mu_0 = 3m\beta\gamma UC/\delta$ and $\delta = 1/8$, we obtain $2^8 m^3\gamma UC$ as the final bound on all numbers appearing in the centering step. The values of x and s are bounded by $(1+\delta)\mu m/(\varepsilon\gamma)$ and $(1+\delta)\mu m/(\varepsilon\beta)$, respectively, which yields the bounds of $2^6 m^2\beta UC$ and $2^6 m^2\gamma UC$. This shows Theorem 3.10. Finally, an arbitrary given min-cost flow instance can first be scaled down, i.e., divided by its gcd, and then scaled up again with β and γ of size as described in Theorem 3.10. It follows that we can always guarantee the bound on the size of numbers in Theorem 3.1.

In this chapter, we have seen an interior point method for the min-cost flow problem that solely works with integer arithmetic and in this way is guaranteed to be free of any numerical instabilities. We feel that our approach, despite the fact that it treats all numerical details, is still fairly elegant. This is due to the fact that we compute in integers, and by this, avoid the tediousness of floating point error analysis. Although we believe that this is not the ultimate answer to practitioners, i.e., this approach should not be implemented exactly as it is described here, we hope that this work can be a first step towards tackling the issue of numerical stability of network flow interior point methods.

Chapter 4

A Gradient Descent Approach to Transshipment and Shortest Path

4.1 Introduction

In this chapter we turn to an undirected network flow problem, namely the asymmetric transshipment problem that we have defined in Section 1.3. We have seen previously that combinatorial methods for network flows are extremely fast in practice and that, although methods from continuous optimization give superior worst-case run-time bounds, their efficiency has not yet been transferred to actual implementations. In this chapter, we are going to make use of one big opportunity that continuous optimization techniques have to offer in order to become relevant for network flows in practice. That is, we are going to make use of their eligibility for concurrent implementation. More specifically, we are going to present a gradient descent framework for the asymmetric transshipment problem that allows for efficient implementation in distributed and streaming models of computation, particularly, in the so-called Broadcast Congest, the Broadcast Congested Clique, and the Multipass Streaming model of computation. See Section 4.3.1 for a precise description of these models. What all these computation models have in common is that they do not assume that the computing entity has knowledge of the complete input graph during the execution of the algorithm, be it because it only has a local understanding (distributed models) or be it because the graph is too large to be stored completely (streaming model). Such computation models become increasingly interesting as computing systems become more and more spread and data becomes larger and larger.

Although there has been a lot of research interest in distributed graph algorithms already for some decades now, the classical network flow problems, such as the max-flow and the min-cost flow problem, have not received too much attention until recently. The first non-trivial results for the undirected max-flow problem in the Congest model of computation (a less restrictive modification of the Broadcast Congest model) only appeared a few years ago [Gha+15]. The technique that this result is based on, goes back to previous work by Sherman [She13] and Kelner et al. [Kel+14] for computing almost maximum flows in close to linear time in the standard RAM model of computation.

In this chapter, we report on the first non-trivial result for computing $(1+\varepsilon)$-approximate solutions to the (undirected) min-cost flow or transshipment problem in distributed and streaming models of computation. See Table 4.1 for the results that we obtain. For the asymmetric variant the bounds on the number of rounds and the time complexity include an additional factor of λ^2, where $\lambda > 1$ is the maximum ratio of a forward and backward weight.

We note that the reason for this recent sudden development of distributed methods for network flow problems is that these methods are based on the continuous optimization approaches that have only been applied to these problems in the last decade. Implementing the classical combinatorial methods in distributed computation models seems to be more

Computation Model	Our Results
Broadcast Congest	$n\varepsilon^{-2}$ polylog n rounds
Broadcast Congested Clique	ε^{-2} polylog n rounds
Multipass Streaming	ε^{-2} polylog n passes $O(n \log n)$ space
RAM	$\tilde{O}(\varepsilon^{-2}(m + n^{3/2}))$ time

Table 4.1: New results for undirected transshipment in different distributed and streaming models of computation, as well as in the standard RAM model.

challenging or even impossible. In fact, both these techniques, the one for the max-flow problem from [Gha+15] and the one presented here for the transshipment problem, are based on gradient descent.

The single source shortest path problem, which is a special case of the transshipment problem (with demand vector $b = \mathbb{1} - n\mathbb{1}_s$ for some source node s), has received a lot of attention of researchers both in standard computation models as well as in distributed, parallel, and streaming models of computation. For the standard RAM model, thanks to sophisticated data structures, it has been known already for a longer time how to solve the single source shortest path problem within (near-)optimal time complexity [FT87; Tho99]. The picture for the PRAM model and also for distributed and streaming models of computation is different. Certainly, there have been advances in exact single-source shortest path algorithms [Spe97; KS97; Coh97; BTZ98; SS99; Cen+15; Gal16; Elk17], and also in methods that compute $(1 + \varepsilon)$-approximate solutions [Coh00; Mil+15; HKN16; EN16]. Nevertheless, near-optimal bounds (up to log-factors) have not been known until the work that we report on in this chapter. Our results and previous best results are summarized in Table 4.2.

Computation model	Previous results	Our results
Broadcast Congest	$(\sqrt{n} + D) \cdot 2^{O(\sqrt{\log n \log(\varepsilon^{-1} \log n)})}$ rounds [HKN16]	$(\sqrt{n} + D)\varepsilon^{-2}$ polylog n rounds
Broadcast Congested Clique	$2^{O(\sqrt{\log n \log(\varepsilon^{-1} \log n)})}$ rounds [HKN16]	ε^{-2} polylog n rounds
Multipass Streaming	$(2 + \varepsilon^{-1})^{O(\sqrt{\log n \log\log n})}$ passes $O(n \log^2 n)$ space [EN16]	ε^{-2} polylog n passes $O(n \log n)$ space

Table 4.2: Previous results and new results for single source shortest path in different distributed and streaming models of computation. Here D denotes the diameter of the graph.

Again, for the asymmetric variant, the bounds on the number of rounds and passes include an additional factor of λ^2. We remark that in the Congest model and also in the more restrictive Broadcast Congest model a lower bound of $\Omega(\sqrt{n}/\log n + D)$ is known [Sar+12], where D denotes the diameter of the input graph. Thus our result is the first to reach this lower bound up to logarithmic factors. It is also worthwhile to note that the previous fastest results for $(1 + \varepsilon)$-approximate single source shortest path in these models of computation [HKN16; EN16] follow a similar framework based on sparse hop set. The recent lower bounds by Abboud et al. [ABP17] exclude the possibility of an approach based on this framework that achieves near-optimality like ours does. Also in the Multipass Streaming model, we match a lower bound: By setting ε small enough, we can compute distances up to the value $\log n$ exactly in integer-weighted graphs using $\operatorname{polylog} n$ passes and $O(n \log n)$ space. Thus, up to poly-logarithmic factors in n, our result matches the lower bound of $n^{1+\Omega(1/p)}/\operatorname{poly} p$ space for all algorithms that decide in p passes if the distance between two fixed nodes in an unweighted undirected graph is at most $2(p+1)$ for any $p = O(\log n/\log\log n)$ [GO16].

Structure of the Remainder of this Chapter. The rest of this chapter is structured as follows: We introduce the gradient descent framework that achieves the above described results in Section 4.2 and present how to implement this framework in the different models of computation in Section 4.3. We conclude in Section 4.4.

4.2 Gradient Descent

Recall the asymmetric transshipment problem as defined in (1.1). Recall that we denoted with $A \in \{-1, 0, 1\}^{n \times 2m}$ the node-arc-incidence matrix of the bidirected graph. As before, we pick the orientation $E^{\rightarrow}$ such that $w^+ \geq w^- \geq \mathbb{1}$[1] and denote analogously to the node-arc incidence matrix with $A := E^{\rightarrow} \cup -E^{\rightarrow}$ the set of directed both forward and backward arcs, and with $G^{\leftrightarrow} = (V, A)$ the bidirected graph corresponding to $G = (V, E)$. Moreover, we define $\lambda := \max\{w_e^+/w_e^- : e \in E\} \geq 1$ as the maximum ratio over all arcs between a forward and a backward weight and let $W = \operatorname{diag}(w^+, w^-)$ denote the $2m \times 2m$ diagonal matrix containing the forward and backward weights. Thus, (1.1) leads to the primal/dual problem pair

$$P^{\text{asym}}(G, w, b): \quad \min\{\mathbb{1}^T W x : Ax = b, x \geq 0\} = \max\{b^T y : (W^{-1} A^T y)_{\max} \leq 1\}. \qquad (4.1)$$

Here, the dual program asks for potentials y such that for each arc $(v, w) \in E$, $y_v - y_w \leq w_{vw}$, maximizing $b^T y$. In the special case of single source shortest path with source $s \in V$, we have $b = \mathbb{1} - n\mathbb{1}_s$. An optimal primal solution x^* is given by routing, for each $s \neq v \in V$, one unit of flow along a shortest path from s to v, and optimal potentials y^* are given by setting y_v^* to the distance from s to v.

4.2.1 Gradient Descent Overview

We will employ gradient descent on a suitable potential function to converge to a near-optimal solution of the asymmetric transshipment problem. Our hope is to require few (poly-logarithmically many) iterations of the gradient descent that allow for efficient implementation in the aforementioned distributed and streaming models of computation. Our (unconstrained, adaptive step size, minimizing) gradient descent algorithm works as follows:

1. Pick a differentiable convex potential function that reflects the objective well and does not change too quickly.

2. Find a reasonable starting solution (a poor approximation typically suffices).

3. Proceed in iterations. In each iteration:

[1]Note that we can exclude 0 as an edge weight, since this is only a mild restriction: Given weights w, we can always generate new weights w' with $w'_e = 1 + \lceil n/\varepsilon \rceil \cdot w_e$ while preserving at least one of the shortest paths between each pair of nodes as well as $(1+\varepsilon)$-approximations. As we assume edge weights to be integer we can assume that $\varepsilon \geq 1/(n\|w\|_\infty)$ (as otherwise it is required to compute an exact solution) and thus our asymptotic running time bounds are not affected by this modification.

(a) Determine the gradient of the potential function at the current solution and a direction for the update in which the gradient indicates that the potential reduces quickly.
(b) Choose a large step size, under the constraint that the gradient does not change too much along the way, and update the solution according to direction and step size.
(c) Check whether the solution is sufficiently close to the optimum. If yes, terminate and return the current solution. Otherwise, proceed with the next iteration.

Since the primal and dual program in (4.1) both have constraints and we have to handle this issue somehow (there are several ways of doing this), we rephrase the (dual) problem so that it includes a single equality constraint.

This strategy requires to overcome a number of challenges, which we discuss before proceeding to presenting the high-level algorithm. First, we show that a sparse spanner gives rise to an efficient *oracle* yielding approximate solutions to transshipment problems in Section 4.2.2. This oracle provides a sufficiently good starting solution and, more importantly, is our tool for determining good update directions for the gradient descent steps. We then proceed to rephrasing the problem and defining a suitable potential in Section 4.2.3. We also analyze its gradient to establish the key properties needed for showing that the update steps guarantee fast progress and, at termination, a close-to-optimal solution. In Section 4.2.4, we state the pseudocode of the algorithm and prove its correctness and running time. As the algorithm computes a dual solution only, we will then show how to also obtain a primal solution from one additional call to the oracle, under the condition that the oracle is capable of providing not only dual, but also primal solutions. The oracle that we will introduce in Section 4.2.2 will meet this criterion.

4.2.2 An Efficient Oracle for $O(\log n)$-approximate Solutions

We have mentioned that we need an oracle that provides approximate solutions to the transshipment problem and variants of it, both for the initial solution as well as for determining a good update direction. We introduce a specific implementation of such an oracle based on *sparse spanners* in this section.

Definition 4.1 (Spanner)**.** *Given an undirected graph $G = (V, E)$ with (symmetric) edge weights w and $\alpha \geq 1$, an α-*spanner *of G is a subgraph $(V, E', w|_{E'})$, $E' \subseteq E$, in which distances are at most by factor α larger than in G.*

In other words, a spanner removes edges from G while approximately preserving distances. It is well-known that for every undirected graph we can efficiently compute an α-spanner of size $O(n \log n)$ with $\alpha = O(\log n)$. We will discuss derandomized implementations of the Baswana-Sen spanner construction [BS07] suitable for the computational models under consideration in Section 4.3. In all considered models of computation, the sparse representation of the approximate distance structure of the graph can be kept "available," i.e., in the Broadcast Congested Clique and Broadcast Congest models, we can make a spanner globally known, and in the Multipass Streaming model we may keep a spanner in memory, see Section 4.3.

Note that in the asymmetric case, an undirected α-spanner construction cannot be directly applied to G. However, we can instead consider the *symmetrized problem*

$$P^{\text{sym}}(G, w, b): \quad \min\{\mathbb{1}^T W_- x : Ax = b, x \geq 0\} = \max\{b^T y : (W_-^{-1} A^T y)_{\max} \leq 1\}, \tag{4.2}$$

where $W_- = \operatorname{diag}(w^-, w^-)$. Note that this is just the undirected transshipment problem with symmetric weights w^-.

Observation 4.2. *A feasible primal-dual pair x, y of $P^{\text{sym}}(G, w, b)$ is a feasible primal-dual pair of $P^{\text{asym}}(G, w, b)$, with $\mathbb{1}^T W_- x \leq \mathbb{1}^T W x \leq \lambda \mathbb{1}^T W_- x$, i.e., the primal objective increases by a factor between 1 and λ and the dual objective value is the same.*

We thus obtain the following lemma.

Lemma 4.3. *Given an α-spanner S for the undirected graph $G = (V, E)$ with weights w^- and any demand vector $\tilde{b} \in \mathbb{R}^n$ with $\tilde{b}^T \mathbb{1} = 0$. Let x, y be an optimal primal-dual pair for $P^{\text{asym}}(S, w, \tilde{b})$. Then $x, \alpha^{-1} y$ is a feasible primal-dual pair in $P^{\text{asym}}(G, w, \tilde{b})$ such that $\mathbb{1}^T W x \leq \alpha \tilde{b}^T(\alpha^{-1} y)$.*

Proof. Observe that any feasible primal solution on the spanner S (padded with 0-entries for edges not in S) is feasible to the asymmetric transshipment problem on G. We show that $\alpha^{-1} y$, whose objective is by factor α smaller than that of y, is a feasible dual solution for the asymmetric transshipment problem on G. As the objectives of x and y on the spanner and the padded x on G are identical, this proves the claim.

Let $\{v, w\} = e \in E$ be arbitrary. By definition of a spanner, there must be a path p_e of weight $\sum_{e' \in p_e} w^-_{e'}$ at most αw_e in S. We have

$$(W_-^{-1} A^T y)_e \leq \frac{|y_w - y_v|}{w^-_e} \leq \frac{\sum_{\{v',w'\} \in p_e} |y_{w'} - y_{v'}|}{w^-_e} = \frac{\sum_{e' \in p_e} w^-_{e'} \frac{|(A^T y)_{e'}|}{w^-_{e'}}}{w^-_e} \leq \frac{\sum_{e' \in p_e} w^-_{e'}}{w^-_e} \leq \alpha.$$

Thus, $(W_-^{-1} A^T (\alpha^{-1} y))_{\max} \leq 1$, i.e., $\alpha^{-1} y$ is feasible in G. □

As mentioned before, we need to restrict our updates to maintain a certain constraint. This constraint is that update steps must be orthogonal to b. Thus, for any demand vector $\tilde{b}$, instead of (4.2), we will consider the primal/dual problem pair

$$P^{\text{sym}\perp}(G, w, \tilde{b}): \quad \begin{aligned} &\min\{\mathbb{1}^T W_- x : Ax + zb = \tilde{b}, x \geq 0\} \\ &\quad = \max\{\tilde{b}^T y : (W_-^{-1} A^T y)_{\max} \leq 1 \text{ and } b^T y = 0\}, \end{aligned} \tag{4.3}$$

where $\perp$ symbolizes that orthogonality with respect to b is enforced in the dual. This additional constraint is reflected in the primal by relaxing the equality constraints to $Ax = \tilde{b} - zb$, i.e., shifting the demands by an arbitrary multiple of b. Note that feasible primal solutions of (4.3) on a spanner are still feasible on G (after padding), and scaling dual solutions does not affect whether $b^T y = 0$ or not. Hence the same arguments as before yield that an α-approximate pair can be computed based on an α-spanner of G and an optimal solution on it.

Corollary 4.4. *Given an α-spanner S for $G = (V, E)$ with weights w^- and any demand vector $\tilde{b} \in \mathbb{R}^n$ with $\tilde{b}^T \mathbb{1} = 0$. Let x, y be an optimal primal-dual pair for $P^{\text{sym}\perp}(S, w, \tilde{b})$. Then $x, \alpha^{-1} y$ is a feasible primal-dual pair in $P^{\text{sym}\perp}(G, w, \tilde{b})$ with $\mathbb{1}^T W_- x \leq \alpha \tilde{b}^T(\alpha^{-} 1 y)$.*

4.2.3 Potential Function for the Gradient Descent

In what follows, we consider a fixed instance of the asymmetric transshipment problem, i.e., G, w and b are fixed, and denote by y^* an optimal solution of the dual of $P^{\text{asym}}(G, w, b)$, i.e., $b^T y^* = \max\{b^T y : (W^{-1} A^T y)_{\max} \leq 1\}$. We relate the dual program to another linear program that normalizes the objective to 1 and seeks to minimize $(W^{-1} A^T y)_{\max}$ instead:

$$P^{\text{reci}}(G, w, b): \quad \min\{(W^{-1} A^T \pi)_{\max} : b^T \pi = 1\}. \tag{4.4}$$

Let us denote by π^* an optimal solution to $P^{\text{reci}}(G, w, b)$. There is a straightforward correspondence between feasible solutions of the dual of $P^{\text{asym}}(G, w, b)$ and feasible solutions of $P^{\text{reci}}(G, w, b)$. This correspondence is captured by the following lemma.

Lemma 4.5. 1. *If π is feasible for $P^{\text{reci}}(G,w,b)$ and satisfies $(W^{-1}A^T\pi)_{\max} > 0$, then $f(\pi) := \pi/(W^{-1}A^T\pi)_{\max}$ defines a feasible solution of the dual of $P^{\text{asym}}(G,w,b)$. Feasible solutions y of the dual of $P^{\text{asym}}(G,w,b)$ that satisfy $b^Ty > 0$ are mapped to feasible solutions of $P^{\text{reci}}(G,w,b)$ via $g(y) := y/b^Ty$.*

2. *The map $f(\cdot)$ preserves the approximation ratio. Namely, for any $\gamma \geq 1$, if π is a solution of of $P^{\text{reci}}(G,w,b)$ within factor γ of the optimum, i.e., $(W^{-1}A^T\pi)_{\max} \leq \gamma \cdot (W^{-1}A^T\pi^*)_{\max}$, then $f(\pi)$ is feasible for the dual in of $P^{\text{asym}}(G,w,b)$ and within factor γ of the optimum, i.e., $b^Tf(\pi) \geq \gamma^{-1}b^Ty^*$.*

Proof. 1. Let π be feasible for $P^{\text{reci}}(G,w,b)$, i.e., $b^T\pi = 1$, with $(W^{-1}A^T\pi)_{\max} > 0$. Then clearly $(W^{-1}A^Tf(\pi))_{\max} = (W^{-1}A^T\frac{\pi}{(W^{-1}A^T\pi)_{\max}})_{\max} = 1$ and thus $f(\pi)$ is feasible for the dual in $P^{\text{asym}}(G,w,b)$. For the other direction, let y be feasible for the dual in $P^{\text{asym}}(G,w,b)$, i.e., $(W^{-1}A^Ty)_{\max} \leq 1$, and let y satisfy $b^Ty > 0$. Then $b^Tg(y) = b^T\frac{y}{b^Ty} = 1$ and thus $g(y)$ is feasible for $P^{\text{reci}}(G,w,b)$.

2. Recall that $b \neq 0$ and $b^T\mathbb{1} = 0$. As $b/(W^{-1}A^Tb)_{\max}$ is feasible for the dual in $P^{\text{asym}}(G,w,b)$ and $b^Tb > 0$, we have that $b^Ty^* > 0$. Thus, $y^* \neq r \cdot \mathbb{1}$ for all $r \in \mathbb{R}$, i.e., there are $v, w \in V$ with $y^*_v \neq y^*_w$. As G is connected, this entails that $(W^{-1}A^Ty^*)_{\max} > 0$. Hence, $g(y^*)$ is feasible for $P^{\text{reci}}(G,w,b)$ and has positive objective, i.e., $(W^{-1}A^Tg(y^*))_{\max} = (W^{-1}A^Ty^*)_{\max}/b^Ty^* > 0$. In particular, we have that $(W^{-1}A^T\pi^*)_{\max} > 0$. Accordingly, if π is a γ-approximation of $P^{\text{reci}}(G,w,b)$, then $(W^{-1}A^T\pi)_{\max} > 0$ and

$$\begin{aligned} b^Tf(\pi) &= \frac{b^T\pi}{(W^{-1}A^T\pi)_{\max}} \geq \frac{1}{\gamma(W^{-1}A^T\pi^*)_{\max}} \geq \frac{1}{\gamma(W^{-1}A^Tg(y^*))_{\max}} \\ &= \frac{b^Ty^*}{\gamma(W^{-1}A^Ty^*)_{\max}} \geq \frac{b^Ty^*}{\gamma}. \end{aligned}$$ □

In other words, it is sufficient to determine a $(1+\varepsilon)$-approximation to $P^{\text{reci}}(G,w,b)$ in order to obtain a $(1+\varepsilon)$-approximation to the dual in $P^{\text{asym}}(G,w,b)$. We have now translated the dual maximization problem of $P^{\text{asym}}(G,w,b)$ in (4.1), which has inequality constraints, into a minimization problem with a single equality constraint $b^T\pi = 0$. This would enable us to employ projected gradient descent in the $b^T\pi = 0$ plane, if it was not for the fact that the objective $(W^{-1}A^T\pi)_{\max}$ is not differentiable. To overcome this issue, we use the standard approach of "smoothing" the gradient by approximating the objective by a differentiable function. For the maximum value of a vector, a suitable candidate is given by the log-sum-exponent (or softmax) function. For vectors $v \in \mathbb{R}^d$, it is defined as

$$\operatorname{lse}_\beta(v) := \frac{1}{\beta}\ln\left(\sum_{i\in[d]} e^{\beta v_i}\right),$$

where the parameter $\beta > 0$ determines the trade-off between accuracy of approximation and "smoothness". More precisely, observe that

$$(v)_{\max} \leq \operatorname{lse}_\beta(v) \leq \frac{1}{\beta}\ln\left(\sum_{i\in[d]} e^{\beta(v)_{\max}}\right) = \frac{1}{\beta}\ln\left(de^{\beta(v)_{\max}}\right) = (v)_{\max} + \frac{\ln d}{\beta}, \tag{4.5}$$

because both ln and e are increasing. Thus for $\beta \to \infty$, the approximation error tends to zero. Furthermore, the 1-norm of the gradient $\nabla \operatorname{lse}_\beta(\cdot)$ is β-Lipschitz continuous w.r.t. the

∞-norm (see, e.g., [She13]), i.e.,

$$\forall z, z' \in \mathbb{R}^d \colon \|\nabla \operatorname{lse}_\beta(z) - \nabla \operatorname{lse}_\beta(z')\|_1 \leq \beta \|z - z'\|_\infty. \tag{4.6}$$

Recall that our goal is to find $\pi \in \mathbb{R}^n$ that is a $(1+\varepsilon)$-approximate feasible solution to (4.4). Accordingly, we define the *potential function*

$$\Phi_\beta(\pi) := \operatorname{lse}_\beta\left(W^{-1}A^T\pi\right).$$

Note that $\Phi_\beta(\cdot)$ is convex for any β, as it is constructed by composing $\operatorname{lse}_\beta(\cdot)$, which is convex for any β with linear functions. We will vary β over the course of the algorithm to balance the requirement of a sufficiently accurate approximation of $(W^{-1}A^T\pi)_{\max}$ with the need for small β, i.e., a smooth gradient.

Concerning the approximation guarantee, our target of a $(1+\varepsilon)$-approximation entails that the potential must be a more accurate approximation to the objective. Accordingly, we will ensure that

$$\Phi_\beta(\pi) \leq \left(1 + \frac{\varepsilon}{4}\right)(W^{-1}A^T\pi)_{\max}. \tag{4.7}$$

This follows directly by (4.5) and $4\ln(2m) \leq \varepsilon\beta(W^{-1}A^T\pi)_{\max}$, which will be an invariant in the algorithm. For later use, we also record another property this approximation guarantee entails. As $\Phi_\beta(\cdot)$ is convex, it follows that

$$\pi^T\nabla\Phi_\beta(\pi) \geq \Phi_\beta(\pi) - \Phi_\beta(0) = \Phi_\beta(\pi) - \frac{\ln(2m)}{\beta} > \left(1 - \frac{\varepsilon}{4}\right)\Phi_\beta(\pi). \tag{4.8}$$

To understand the effect of β on the progress of the gradient descent, we examine the gradient of the potential function. As W^{-1} and A^T are linear functions,

$$\nabla\Phi_\beta(\pi) = AW^{-1}\nabla \operatorname{lse}_\beta\left(W^{-1}A^T\pi\right). \tag{4.9}$$

We can now show the following lemma. Recall that $W_- = \operatorname{diag}(w^-, w^-)$.

Lemma 4.6. *For all $\pi, h \in \mathbb{R}^n$ it holds that $|\nabla\Phi_\beta(\pi)^T h - \nabla\Phi_\beta(\pi - h)^T h| \leq \beta(W_-^{-1}A^T h)^2_{\max}$.*

Proof. The proof is an easy calculation that uses Hölder's inequality[2] and the Lipschitz continuity of the gradient from (4.6):

$$\begin{aligned}
\left|\nabla\Phi_\beta(\pi)^T h - \nabla\Phi_\beta(\pi - h)^T h\right| &\overset{(4.9)}{=} \left|\left(\nabla \operatorname{lse}_\beta\left(W^{-1}A^T\pi\right) - \nabla \operatorname{lse}_\beta\left(W^{-1}A^T(\pi - h)\right)\right)^T W^{-1}A^T h\right| \\
&\leq \left\|\nabla \operatorname{lse}_\beta\left(W^{-1}A^T\pi\right) - \nabla \operatorname{lse}_\beta\left(W^{-1}A^T(\pi - h)\right)\right\|_1 \left\|W^{-1}A^T h\right\|_\infty \\
&\overset{(4.6)}{\leq} \beta\|W^{-1}A^T h\|_\infty^2 = (W_-^{-1}A^T h)^2_{\max},
\end{aligned}$$

where the last step uses that $w_e^+ \geq w_e^-$ for all $e \in E$. □

Intuitively, we decomposed the gradient into the β-Lipschitz continuous part given by the $\operatorname{lse}_\beta(\cdot)$ function and the derivative AW^{-1} of the "inner" function. This means that the "length" of an update step h will be measured in terms of $(W_-^{-1}A^T h)_{\max}$. Using this bound, for a given step direction h we can now determine the step size that maximizes the progress in direction of h as a function of $\nabla\Phi_\beta(\pi)$.

[2]It holds that $|z^T z'| \leq \|z\|_p + \|z'\|_q$ for any $p, q \geq 1$ satisfying $p^{-1} + q^{-1} = 1$ (where $\infty^{-1} = 0$).

Lemma 4.7 (Additive Decrement of Φ_β). *Suppose $\pi, h \in \mathbb{R}^n$ satisfy $\nabla\Phi_\beta(\pi)^T h > 0$ and $(W_-^{-1}A^T h)_{\max} \leq 1$. Then, for $\delta := \nabla\Phi_\beta(\pi)^T h$ it holds that*

$$\Phi_\beta\left(\pi - \frac{\delta h}{2\beta}\right) \leq \Phi_\beta(\pi) - \frac{\delta^2}{4\beta}.$$

Proof. Let us denote $\tilde{h} := \frac{\delta h}{2\beta}$. The proof again uses that $\Phi_\beta(\cdot)$ is convex and applies Lemma 4.6:

$$\begin{aligned}\Phi_\beta(\pi - \tilde{h}) - \Phi_\beta(\pi) &\leq -\nabla\Phi_\beta\big(\pi - \tilde{h}\big)^T \tilde{h} + \nabla\Phi_\beta(\pi)^T\tilde{h} - \nabla\Phi_\beta(\pi)^T\tilde{h} \\ &\leq \beta(W_-^{-1}A^T\tilde{h})^2_{\max} - \nabla\Phi_\beta(\pi)^T\tilde{h} = \frac{\delta^2(W_-^{-1}A^T h)^2_{\max}}{4\beta} - \frac{\delta^2}{2\beta} \leq -\frac{\delta^2}{4\beta}. \qquad \square\end{aligned}$$

As this lemma shows, we make β small in order to ensure large progress. However, progress with respect to $\Phi_\beta(\cdot)$ is meaningless if it does not provide a sufficiently accurate approximation of the true objective $(W_-^{-1}A^T(\cdot))_{\max}$, so we will increase β only when it becomes necessary to ensure (4.7). Upper bounding β turns the additive guarantee into a relative one.

Corollary 4.8 (Multiplicative Decrement of Φ_β). *Suppose $\pi, h \in \mathbb{R}^n$ satisfy $\nabla\Phi_\beta(\pi)^T h > 0$, $(W_-^{-1}A^T h)_{\max} \leq 1$, and $\varepsilon\beta\Phi_\beta(\pi) \leq 10\ln(2m)$. Then, for $\delta := \nabla\Phi_\beta(\pi)^T h$ it holds that*

$$\Phi_\beta\left(\pi - \frac{\delta h}{2\beta}\right) \leq \left(1 - \frac{\varepsilon\delta^2}{40\ln(2m)}\right)\Phi_\beta(\pi).$$

4.2.4 Generic Algorithm

We now present the pseudocode of the generic gradient descent algorithm for a $(1+\varepsilon)$-approximate solution to the dual of $P^{\text{asym}}(G, w, b)$ in (4.1). As mentioned before, the algorithm actually computes a $(1+\varepsilon)$-approximate solution π to $P^{\text{reci}}(G, w, b)$ in (4.4) and then returns $\pi/(W^{-1}A^T\pi)_{\max}$. The algorithm is generic in the sense that the performed computations need to be implemented in model-specific ways, which is discussed in Section 4.3.

Algorithm 4.1: GradientTransship (G, b, ε)

```
1  compute $(\alpha\lambda)$-approximation $\pi$ to (4.4) // use oracle, Obs. 4.2, and Lemma 4.5
2  set $\varepsilon' := 1$
3  repeat
4      set $\varepsilon' \leftarrow \frac{\varepsilon'}{2}$
5      set $\beta := \frac{8\ln(2m)}{\varepsilon'(W^{-1}A^T\pi)_{\max}}$
6      repeat
7          compute $\alpha$-approximate solution $h$ to
             $\max\{\nabla\Phi_\beta(\pi)^T\tilde{h} : (W_-^{-1}A^T\tilde{h})_{\max} \leq 1 \text{ and } b^T\tilde{h} = 0\}$.
             // use oracle with weights $w^-$ and demands $\tilde{b} = \nabla\Phi_\beta(\pi)$ (Cor. 4.4)
8          set $\delta := \nabla\Phi_\beta(\pi)^T h$
9          if $\delta > \frac{\varepsilon'}{8\alpha\lambda}$ then $\pi \leftarrow \pi - \frac{\delta h}{2\beta}$
10         while $\beta < \frac{4\ln(2m)}{\varepsilon'(W^{-1}A^T\pi)_{\max}}$ do $\beta \leftarrow 2\beta$    // happens only when $\varepsilon' = \frac{1}{2}$
11     until $\delta \leq \frac{\varepsilon'}{8\alpha\lambda}$          // current $\pi$ is a $(1+\varepsilon')$-approximate solution
12 until $\varepsilon' \leq \varepsilon$
13 return $\pi/(W^{-1}A^T\pi)_{\max}$                                          // Lemma 4.5
```

The pseudo-code is given in Algorithm 4.1. The algorithm first computes a starting solution. This can be done by calling the oracle on the symmetrized problem (4.2), which by Observation 4.2 yields an $(\alpha\lambda)$-approximation to (4.1), and then rescaling according to Lemma 4.5. As trying to enforce a too precise approximation initially slows down progress, the algorithm initializes ε' as a constant (the choice of 1 is, neglecting technicalities, arbitrary). Then it starts an outer loop decreasing ε' exponentially. It sets β to a suitable value and then starts the inner loop, which performs gradient descent steps until the step size becomes too small to ensure fast progress, i.e., $\delta \leq \frac{\varepsilon'}{8\alpha\lambda^2}$. As in the first iteration of the outer loop, the potential may decrease dramatically (we go from an α-approximation to a $\frac{1}{2}$-approximation), Line 10 adjusts β if necessary. As we will show, this implies that at termination of the inner loop, the current solution is a $(1+\varepsilon')$-approximation.

Each gradient descent step consists of the following steps:

1. Compute $\nabla\Phi_\beta(\pi)$.
2. Compute an α-approximate solution h to

   $$\max\left\{\nabla\Phi_\beta(\pi)^T\tilde{h} : (W_-^{-1}A^T\tilde{h})_{\max} \leq 1 \text{ and } b^T\tilde{h} = 0\right\},$$

   i.e., of (4.3) with demands $\nabla\Phi_\beta(\pi)$. This is done by calling the oracle (for the constrained problem). This optimization problem maximizes the rate of progress "in units of $(W_-^{-1}A^T\tilde{h})_{\max}$," which in turn determines the step size, see Corollary 4.8.
3. Determine the step size in accordance with Corollary 4.8 and check whether δ is small enough to prove that the current π is a $(1+\varepsilon')$-approximation. If yes, the inner loop terminates; if not, the step is executed.
4. Scale up β if it became too small.

Returning a primal solution. As given, the algorithm relies on a dual oracle only and also returns only a dual solution. For a primal solution, we require the oracle to provide an α-approximate primal-dual pair to the constrained problem (4.3) (with demands $\nabla\Phi_\beta(\pi)$).[3] Denote by π the dual solution that yields the output $\pi/(W^{-1}A^T\pi)_{\max}$ of Algorithm 4.1.

Observation 4.9. *The vector* $\tilde{x} = W^{-1}\nabla\operatorname{lse}_\beta\left(W^{-1}A^T\pi\right)$ *satisfies*

$$\tilde{x} \geq 0, \quad A\tilde{x} = \nabla\Phi_\beta(\pi), \quad \textit{and} \quad \mathbb{1}^T W\tilde{x} = 1. \tag{4.10}$$

Proof. By (4.9), $A\tilde{x} = \nabla\Phi_\beta(\pi)$. For the other two properties, we let $v = W^{-1}A^T\pi$ and observe that for any $a \in A$, it holds that

$$\tilde{x}_a = \frac{1}{w_a}\nabla\operatorname{lse}_\beta(v)_a = \frac{1}{\beta w_a}\frac{\beta e^{\beta v_a}}{\sum_{a\in A} e^{\beta v_a}} > 0 \quad \text{and} \quad \mathbb{1}^T W\tilde{x} = \frac{1}{\beta}\sum_{a\in A}\frac{\beta e^{\beta v_a}}{\sum_{a\in A} e^{\beta v_a}} = 1. \qquad \square$$

Conveniently, this $\tilde{x}$ is a complimentary byproduct of evaluating $\nabla\Phi_\beta(\pi)$ in the last iteration of the algorithm, by storing the intermediate result before the final multiplication with A, see (4.9). Recall that in its final iteration, the algorithm computes an α-approximate dual solution h to $P^{\text{sym}\perp}(G, w, \nabla\Phi_\beta(\pi))$, see (4.3), where π is such that $\pi/(W^{-1}A^T\pi)_{\max}$ is the return value of the algorithm. I.e., h is an α-approximate dual solution to the program

$$\begin{aligned}\min\{&\mathbb{1}^T W_-\tilde{f} : A\tilde{f} + zb = \nabla\Phi_\beta(\pi), \tilde{f} \geq 0\}\\ &= \max\{\nabla\Phi_\beta(\pi)^T\tilde{h} : (W_-^{-1}A^T\tilde{h})_{\max} \leq 1 \text{ and } b^T y = 0\},\end{aligned}$$

[3]Actually, the primal solution is exclusively required on termination to generate a primal solution to (4.1).

where we use $\tilde{f}$ and $\tilde{h}$ to denote the variables for the sake of distinction from the original problem. An $\tilde{x}$ as above is useful for constructing a primal solution, as we can use it to cancel out the demands in the primal program and then rescale according to z.

Observation 4.10. *For any given $\beta \in \mathbb{R}_{>0}$, $\pi \in \mathbb{R}^n$ with $b^T\pi = 1$, primal solution $(f, z) \in \mathbb{R}^{2m}_{\geq 0} \times \mathbb{R}$ to $P^{\text{sym}\perp}(G, w, \nabla\Phi_\beta(\pi))$ satisfying $z > 0$, and $\tilde{x} \in \mathbb{R}^{2m}$ satisfying* (4.10)*, write $f = [\begin{smallmatrix} f^- \\ f^+ \end{smallmatrix}]$ with f^+ and f^- being of dimension m, respectively. Then a primal solution to $P^{\text{asym}}(G, w, b)$, see* (4.1)*, is given by $x := (\tilde{x} + [\begin{smallmatrix} f^- \\ f^+ \end{smallmatrix}])/z$.*

Proof. Note that we will see in the proof of Lemma 4.11 that $z > 0$ holds for any feasible primal solution of $P^{\text{sym}\perp}(G, w, \nabla\Phi_\beta(\pi))$. Thus all of $\tilde{x}$, f, and z are non-negative, and the same follows for x. It holds that $Ax = b$, since $A[\begin{smallmatrix} f^- \\ f^+ \end{smallmatrix}] = -Af = -\nabla\Phi_\beta(\pi) + zb$, as for each $e \in E$ we have a forward and a backward arc. $\square$

The intuition regarding why this should be a "good" primal solution to (4.1) is as follows. Any π' minimizing $\Phi_\beta(\cdot)$ must satisfy that $\nabla\Phi_\beta(\pi')^T h = 0$ for any h with $b^T h = 0$. Thus, $\nabla\Phi_\beta(\pi')$ is parallel to b. For a near-optimal solution π we have that $\nabla\Phi_\beta(\pi)$ is *almost* parallel to b, as the gradient restricted to the $h^T b = 0$ plane is small. Accordingly, the "cost" of the correction, given by $\mathbb{1}^T W f$ (which relates to $\mathbb{1}^T W_- f$), is small. The role of the scaling factor z is to undo the normalization of (4.1) we performed by transitioning to problem (4.4).

Proof of correctness. We show the approximation guarantee of the algorithm by arguing that for any $\tilde{x}$ satisfying (4.10) and (f, z), h as above, the pair (x, y), where $y := \pi/(W^{-1}A^T\pi)_{\max}$ is the return value of the algorithm, is a $(1 + \varepsilon)$-approximate pair, i.e., $\mathbb{1}^T W x \leq (1 + \varepsilon) b^T y$. In particular, both x and y are $(1 + \varepsilon)$-approximate solutions. We stress, however, that Algorithm 4.1 does not need to compute (f, z); we merely exploit its existence for proving the approximation guarantee. The advantage of this approach is that we also learn that the above recipe for constructing a primal solution works. Thus, to determine a primal solution in case this is required, all we need is that the oracle indeed can provide such an (f, z). Note that, by Corollary 4.4, the oracle we use throughout this work has this capability.

Lemma 4.11 (Correctness of Algorithm 4.1)**.** *Let $0 < \varepsilon \leq 1$,*

- *$\pi \in \mathbb{R}^n$ be such that $y := \pi/(W^{-1}A^T\pi)_{\max}$ is the return value of Algorithm 4.1,*
- *$(f, z) \in \mathbb{R}^{2m}_{\geq 0} \times \mathbb{R}$ and $h \in \mathbb{R}^n$ be an α-approximate pair for $P^{\text{sym}\perp}(G, w, \nabla\Phi_\beta(\pi))$, where β is the value of β in the algorithm at termination,*
- *$\tilde{x} \in \mathbb{R}^{2m}_{\geq 0}$ satisfying* (4.10)*, and*
- *$x := (\tilde{x} + [\begin{smallmatrix} f^- \\ f^+ \end{smallmatrix}])/z$.*

Then (x, y) is a $(1 + \varepsilon)$-approximate pair for $P^{\text{asym}}(G, w, b)$, i.e., it holds that $Ax = b$, $x \geq 0$, $(W^{-1}A^T y)_{\max} \leq 1$, and $\mathbb{1}^T W x \leq (1 + \varepsilon) b^T y$.

Proof. Due to termination of the algorithm, we have that $\varepsilon' \leq \varepsilon$ and $\delta \leq \frac{\varepsilon}{8\alpha\lambda}$, yielding that

$$\nabla\Phi_\beta(\pi)^T h \leq \frac{\varepsilon}{8\alpha\lambda} \leq \frac{\varepsilon}{8\alpha}. \tag{4.11}$$

Note that, as $f \geq 0$, $\|W_- f\|_1 = \mathbb{1}^T W_- f$. As $Af + zb = \nabla\Phi_\beta(\pi)$, we get $z\pi^T b = \pi^T \nabla\Phi_\beta(\pi) - \pi^T Af$ and since $b^T\pi = 1$, it holds that

$$\begin{aligned} z &= \pi^T\nabla\Phi_\beta(\pi) - \pi^T Af \overset{(4.8)}{\geq} \left(1 - \frac{\varepsilon}{4}\right)\Phi_\beta(\pi) - \pi^T A W_-^{-1} W_- f \\ &\overset{(4.5)}{\geq} \left(1 - \frac{\varepsilon}{4}\right)(W_-^{-1}A^T\pi)_{\max} - (W_-^{-1}A^T\pi)^T W f \end{aligned}$$

Using Hölder's inequality and $\|W_-^{-1}A^T\pi\|_\infty = (W_-^{-1}A^T\pi)_{\max}$, we get

$$\begin{aligned} z &\geq \left(1 - \frac{\varepsilon}{4}\right)(W_-^{-1}A^T\pi)_{\max} - \|W_-^{-1}A^T\pi\|_\infty \|W_- f\|_1 \\ &= \left(1 - \frac{\varepsilon}{4} - \mathbb{1}^T W_- f\right)(W_-^{-1}A^T\pi)_{\max} \end{aligned}$$

Since $(f, z), h$ is an α-approximate pair, it holds that $\mathbb{1}^T W_- f \leq \alpha\nabla\Phi_\beta(\pi)^T h$. Together with (4.11), this leads

$$z \geq \left(1 - \frac{\varepsilon}{4} - \alpha\nabla\Phi_\beta(\pi)^T h\right)(W_-^{-1}A^T\pi)_{\max} \geq \left(1 - \frac{3\varepsilon}{8}\right)(W_-^{-1}A^T\pi)_{\max}. \tag{4.12}$$

In particular, $z > 0$ and, by Observation 4.10 and Lemma 4.5, x and y are indeed feasible primal and dual solutions of $P^{\text{asym}}(G, w, b)$, respectively.

It remains to show that $\mathbb{1}^T W x \leq (1+\varepsilon) b^T y$. We first observe that $\mathbb{1}^T W[\begin{smallmatrix} f^- \\ f^+ \end{smallmatrix}] \leq \lambda \mathbb{1}^T W_-[\begin{smallmatrix} f^- \\ f^+ \end{smallmatrix}] = \lambda\mathbb{1}^T W_- f \leq \lambda\alpha\nabla\Phi_\beta(\pi)^T h$, where the last inequality holds again since $(f, z), h$ is an α-approximate pair. Together with $\mathbb{1}^T W\tilde{x} \leq 1 + \frac{\varepsilon}{8}$ by assumption this gives

$$\begin{aligned} \mathbb{1}^T W x &\leq \frac{1 + \frac{\varepsilon}{8} + \lambda\alpha\nabla\Phi_\beta(\pi)^T h}{z} \overset{(4.11)}{\leq} \frac{1 + \frac{\varepsilon}{4}}{z} \overset{(4.12)}{\leq} \frac{1 + \frac{\varepsilon}{4}}{\left(1 - \frac{3\varepsilon}{8}\right)(W_-^{-1}A^T\pi)_{\max}} \overset{\varepsilon \leq 1}{\leq} \frac{1+\varepsilon}{(W_-^{-1}A^T\pi)_{\max}} \\ &= (1+\varepsilon) b^T y. \end{aligned}$$

□

Corollary 4.12. *Suppose* $0 < \varepsilon \leq 1$. *Whenever the inner loop of Algorithm 4.1 terminates, the current* π *is a* $(1 + \varepsilon')$*-approximate dual solution to* (4.1).

Proof. If we ran the algorithm for $\varepsilon = \varepsilon'$, it would perform exactly the same computations up to that point and then terminate. Noting that an optimal primal solution (f, z) for the problem posed to the oracle in the last iteration and, by Observation 4.10, $\tilde{x} = W^{-1}\nabla \operatorname{lse}_\beta\left(W^{-1}A^T\pi\right)$ satisfy the preconditions of Lemma 4.11, the claim follows. □

Bounding the number of iterations. We now examine how many iterations Algorithm 4.1 requires to terminate. This reduces the task of bounding the overall running time to determining the cost of implementing a single iteration, which depends on the specific model of computation. To this end, we first prove a helper statement showing that the algorithm maintains suitable values of β, and how adjusting β affects $\Phi_\beta(\pi)$.

Lemma 4.13. *Except immediately after updates of* ε' *or* π *(i.e., before adjusting* β *in the subsequent lines), Algorithm* GradientTransship *maintains the invariant that*

$$4\ln(2m) \leq \varepsilon'\beta(W^{-1}A^T\pi)_{\max} \leq \varepsilon'\beta\Phi_\beta(\pi) \leq 10\ln(2m).$$

In particular, $\Phi_\beta(\pi) \leq (1 + \frac{\varepsilon'}{4})(W^{-1}A^T\pi)_{\max}$.

Proof. First, observe that $4\ln(2m) \leq \varepsilon'\beta(W^{-1}A^T\pi)_{\max}$ follows immediately from Lines 5 and 10. By (4.5), this implies (4.7), i.e., the last statement of the lemma. As $(W^{-1}A^T\pi)_{\max} \leq \Phi_\beta(\pi)$

for any β, it remains to show that $\varepsilon'\beta\Phi_\beta(\pi) \leq 10\ln(2m)$. Note that $\varepsilon'\beta(W^{-1}A^T\pi)_{\max} \leq 8\ln(2m)$ implies that

$$\varepsilon'\beta\Phi_\beta(\pi) \leq \varepsilon'\beta\left(1+\frac{\varepsilon'}{4}\right)(W^{-1}A^T\pi)_{\max} \overset{\varepsilon'\leq 1}{\leq} 10\ln(2m).$$

Thus, the statement always holds after executing Line 5 or adjusting β according to Line 10. This leaves only the possibility that updates of π cause a violation while β is not adjusted. However, Lemma 4.7 shows that updates of π may only decrease the potential, completing the proof. □

Lemma 4.14 (Number of iterations of Algorithm 4.1)**.** *Suppose that* $0 < \varepsilon \leq 1$. *Then Algorithm* GradientTransship *performs* $O((\varepsilon^{-2} + \log\alpha + \log\lambda)\alpha^2\lambda^2\log n)$ *iterations of its inner loop.*

Proof. Denote by $\pi^{(i)}$ and β_i, $0 \leq i \leq i_{\max} := \lceil\log\varepsilon^{-1}\rceil$, the values of π and β at the beginning the $(i{+}1)$-st iteration of the outer loop, where $\pi^{(i_{\max})}$ and $\beta_{i_{\max}}$ denote the values at termination. Refer to the i-th iteration as *phase* i and denote by $\varepsilon_i = 2^{-i}$ the value of ε' during this loop iteration.

Lemma 4.13 shows that the preconditions of Corollary 4.8 are satisfied by each update of π. As $\delta \leq \frac{\varepsilon_i}{8\lambda\alpha}$ implies termination of the inner loop, as long as the inner loop does not terminate, the corollary shows that the potential decreases by a factor of at least

$$\left(1-\frac{\varepsilon_i\delta^2}{40\ln(2m)}\right) \leq \left(1-\frac{\varepsilon_i^3}{2560\alpha^2\lambda^2\ln(2m)}\right) =: q_i.$$

However, when doubling β in Line 10, the potential might increase. As, after Line 10 we have that $\Phi_\beta(\pi) \leq (1+\frac{\varepsilon_i}{4})(W^{-1}A^T\pi)_{\max}$ again by Lemma 4.13 and since, for any β, it holds that $\Phi_\beta(\pi) \geq (W^{-1}A^T\pi)_{\max}$, the increase in potential due to an update of β according to Line 10 is multiplicatively bounded by $1+\frac{\varepsilon_i}{4}$. Observe that β can be doubled no more than

$$t_i := \log\left(\frac{(W^{-1}A^T\pi^{(i-1)})_{\max}}{(W^{-1}A^T\pi^{(i)})_{\max}}\right) \leq \log\left(\frac{(W^{-1}A^T\pi^{(i-1)})_{\max}}{(W^{-1}A^T\pi^{*})_{\max}}\right)$$

times in phase i. First, note that for $i = 1$, as $\pi^{(0)}$ is an $(\alpha\lambda)$-approximation, we have that $t_1 \leq \log(\alpha\lambda)$. For all $i > 1$, by Corollary 4.12, we have that $(W^{-1}A^T\pi^{(i-1)})_{\max} \leq (1+\varepsilon_{i-1})(W^{-1}A^T\pi^*)_{\max} < 2(W^{-1}A^T\pi^*)_{\max}$ and thus $t_i < 1$, hence $t_i = 0$.[4]

Denote by β_i' the value of β when the inner loop terminates at the end of phase i and by k_i the number of iterations in this phase. Using that $\Phi_{\beta_{i-1}}(\pi^{(i-1)}) \leq (1+\frac{\varepsilon_i}{4})(W^{-1}A^T\pi^{(i-1)})_{\max}$ by Lemma 4.13, we can bound

$$\begin{aligned}(W^{-1}A^T\pi^*)_{\max} &\leq (W^{-1}A^T\pi^{(i)})_{\max} \leq \Phi_{\beta_i'}(\pi^{(i)}) \leq q_i^{k_i}\left(1+\frac{\varepsilon_i}{4}\right)^{t_i}\Phi_{\beta_{i-1}}(\pi^{(i-1)})\\ &\leq q_i^{k_i}\left(1+\frac{\varepsilon_i}{4}\right)^{t_i+1}(W^{-1}A^T\pi^{(i-1)})_{\max}.\end{aligned}$$

Now, we distinguish cases. First consider the case $i = 1$. Then $(W^{-1}A^T\pi^{(i-1)})_{\max} = (W^{-1}A^T\pi^{(0)})_{\max} \leq \alpha\lambda(W^{-1}A^T\pi^*)_{\max}$. Moreover, using that $t_1 \leq \log(\alpha\lambda)$, we conclude that $\left(1+\frac{\varepsilon_1}{4}\right)^{t_1+1} \leq 2^{t_1+1} \leq 2\alpha\lambda$. Thus

$$(W^{-1}A^T\pi^*)_{\max} < q_i^{k_i}(W^{-1}A^T\pi^*)_{\max}\cdot 2\alpha^2\lambda^2$$

[4]This qualifies the comment in Line 10 of the algorithm.

and rearranging and taking the logarithm yields $k_1 \leq \log(2\alpha^2\lambda^2)/\log q_1^{-1} = O(\alpha^2\lambda^2 \log n \cdot \log(\alpha\lambda))$. In the other case, where $i > 1$, it holds that $\pi^{(i-1)}$ was already an ε_{i-1}-approximate solution, thus $(W^{-1}A^T\pi^{(i-1)})_{\max} \leq (1+\varepsilon_{i-1})(W^{-1}A^T\pi^*)_{\max}$ and $t_i = 0$. Hence in this case

$$(W^{-1}A^T\pi^*)_{\max} \leq q_i^{k_i}\left(1+\frac{\varepsilon_i}{4}\right)(1+\varepsilon_{i-1})(W^{-1}A^T\pi^*)_{\max} < q_i^{k_i}(W^{-1}A^T\pi^*)_{\max}\cdot(1+3\varepsilon_i),$$

where the last inequality uses that $\varepsilon_i = \varepsilon_{i-1}/2$. Rearranging and taking the logarithm yields $k_i < \frac{\log(1+\varepsilon_i)}{-\log q_i} = O\left(\varepsilon_i^{-2}\alpha^2\lambda^2\ln(2m)\right)$. Summing over all phases i, the total number of iterations is bounded by

$$\sum_{i=1}^{i_{\max}} k_i = O\Big(\Big[\log(\alpha\lambda) + \sum_{i=2}^{i_{\max}} \varepsilon_i^{-2}\Big]\alpha^2\lambda^2\log(n)\Big) = O\Big(\Big[\varepsilon^{-2} + \log\alpha + \log\lambda\Big]\alpha^2\lambda^2\log(n)\Big). \square$$

The bound on the number of iterations of the algorithm readily translates to one on the specific operations the algorithm performs.

Corollary 4.15. *Algorithm* GradientTransship *can be executed using a total of $O((\varepsilon^{-2} + \log\alpha + \log\lambda)\alpha^2\lambda^2\log n)$ operations of the following types:*

- *oracle call*
- *computation of $(W^{-1}A^T\pi)_{\max}$ for a given π*
- *computation of $\nabla\Phi_\beta(\pi)$ for given β and π*
- *scalar product $\tilde{b}^T h$ for given $\tilde{b}$ and h*
- *comparisons of and multiplications with scalar values*

We summarize the results of this section in the following theorem.

Theorem 4.16. *Given an oracle that computes α-approximate dual solutions to* (4.3)*, Algorithm* GradientTransship *computes a $(1+\varepsilon)$-approximate dual solution to the asymmetric transshipment problem* (4.1) *calling the oracle $O((\varepsilon^{-2} + \log\alpha + \log\lambda)\alpha^2\lambda^2\log n)$ times.*

If, on termination of the algorithm, the oracle provides an α-approximate primal-dual pair of solutions to (4.3)*, the algorithm can compute a $(1+\varepsilon)$-approximate primal-dual pair of solutions to* (4.1) *with the same number of oracle calls.*

4.2.5 Single-Source Shortest Paths

In the special case of single-source shortest path, the main theorem of the previous section unfortunately does not provide a satisfactory result. Namely, a $(1+\varepsilon)$-approximate primal-dual pair neither provides, for every node, a distance estimate within a factor of $1+\varepsilon$, nor a path from that node to the source that is within factor $1+\varepsilon$ of the shortest path. What it does provide is only distance estimates that are good *on average* as $b^T y$ is equal to the sum of all distance estimates (can be seen by shifting y such that $y_s = 0$ for the source node s). In this section, our goal is to give a method that delivers both the above mentioned guarantees, namely, we are going to show how to compute a tree T such that the path along the tree from any node to the source is a $(1+\varepsilon)$-approximate shortest path. Moreover, the length of that path also constitutes a $(1+\varepsilon)$-approximation to the distance of the node to the source.

We start with a well-known structural result: For any transshipment problem, there is always an optimal primal solution whose edges with non-zero flow form a forest. This is a well-known result of the structure of linear programs, see for example [AMO93, Theorem 11.1] for the case of the min-cost flow problem. For completeness, we directly prove the statement here.

Lemma 4.17. *The problem $P^{\text{asym}}(G, w, b)$, see* (4.1)*, has an optimal primal solution that sends flow only along the arcs of a forest.*

Proof. Suppose $x \in \mathbb{R}^{2m}$ is an optimal primal solution and let C be any cycle such that for each $a \in C$, $x_a \neq 0$. Consistently direct C. Let $f \in \mathbb{R}^{2m}$ (not necessarily satisfying $f \geq 0$) send $\min\{x_a : a \in C\}$ units of flow in positive direction around C, so that $f_a \neq 0$ implies $x_a \neq 0$. By construction, both $x - f \geq 0$ and $x + f \geq 0$ and, as C is a cycle, $Af = 0$, i.e., $A(x - f) = A(x + f) = Ax = b$. Thus, both $x - f$ and $x + f$ are feasible. Note that $x - f$ or $x + f$ satisfies that there is an arc less on C carrying flow; assume w.l.o.g. it is $x - f$. We claim that $\mathbb{1}^T W(x - f) = \mathbb{1}^T Wx$. Assuming for contradiction that this is false, either $\mathbb{1}^T W(x - f) = \mathbb{1}^T Wx - \mathbb{1}^T Wf < \mathbb{1}^T Wx$ or $\mathbb{1}^T W(x + f) < \mathbb{1}^T Wx$, i.e., either $x - f$ or $x + f$ has smaller objective than x, contradicting its optimality. We conclude that $x - f$ is also an optimal solution. The claim now follows by inductively repeating the argument until no cycles remain. □

Our approach to compute the tree solution is based on a simple sampling procedure using Algorithm 4.1 as a subroutine. The idea, which relies on our specific choice of a spanner-based oracle, is as follows:

1. Run Algorithm 4.1 for $\varepsilon' := \frac{\varepsilon}{6}$ and denote by x the returned primal solution.

2. For each node $v \in V$, partition its incoming arcs $a \in \delta_G^{\text{in}}(v)$ with $x_a > 0$ into classes in which arc weights differ by a factor of at most 2.[5] For an ingoing arc a, denoting by f_a the sum of flows of arcs in the class of a, sample a with probability (roughly) $\min\{(2\lambda\alpha + 1)x_a/f_a, 1\}$.

3. We show that an optimal solution using only sampled and spanner arcs is a $(1 + 2\varepsilon')$-approximation with probability at least 2/3, and that we can bound the number of arcs sampled by the procedure by $O(\alpha\lambda n \log n)$ with probability at least 2/3.

4. We abort the procedure if more arcs are selected. Otherwise, we compute and return an optimum tree solution on the selected arc set. By the union bound, we thus obtain a $(1 + \varepsilon)$-approximation using only the spanner arcs and $O(\alpha\lambda n \log n)$ sampled arcs with probability at least $\frac{1}{3}$.

5. To obtain a Las Vegas algorithm, we simply repeat the procedure until a good solution is obtained; this can be checked by comparing the objective value of the obtained tree solution to the one of the dual solution returned by Algorithm 4.1.

The second last step exploits that in the computational models that are considered in this chapter for the single source shortest path problem, operations on a sparse graph are cheap; thus, we can compute an optimal tree solution on this set of arcs being of nearly linear size. Note that thus this strategy is tied to using an oracle based on a sparse graph.

In order to prove the correctness of the above procedure, we decompose $x =: x^{\not\circ} + x^{\circ}$, where the arcs with non-zero flow in $x^{\not\circ}$ form a directed acyclic graph (DAG). Such a decomposition is always feasible, which again is a standard property whose proof we give for completeness. Note that it resembles the proof of Lemma 4.17.

[5]Note that it is straightforward to ensure that there are only $O(\log n)$ classes. We will get back to this at the end of this section.

Lemma 4.18. *Any primal solution $x \in \mathbb{R}^{2m}_{\geq 0}$ to $P^{\text{asym}}(G, w, b)$ can be decomposed into a feasible solution $x^{\not\circ}$ inducing a DAG and $x^{\circ} \geq 0$ satisfying $Ax^{\circ} = 0$.*

Proof. Let C be any *consistently oriented* cycle so that for each arc $a \in C$, $x_a > 0$ (if no such C exists, $x^{\not\circ} := x$ and $x^{\circ} = 0$ meet the claim of the lemma). Let $f \in \mathbb{R}^{2m}$ denote the flow that sends $\min\{x_a : a \in C\}$ units of flow in positive direction "around" C. By construction both f and $x - f$ are feasible, and $x - f$ has one arc less carrying non-zero flow, namely the arc corresponding to the minimizer in the definition of f. The claim now follows by inductively repeating the argument until no directed cycles remain. The value of x° is the sum of the flows f from the individual steps. □

We would like to sample from $x^{\not\circ}$, but have to face the "disturbance" by x°. We will exploit that the flow x° is "cheap" to bound its effect on the sampled solution. Let us fix the following notation. Again, let $x \in \mathbb{R}^{2m}$ be the primal solution for $P^{\text{asym}}(G, w, b)$ returned by Algorithm 4.1 when called with accuracy parameter $\varepsilon' := \varepsilon/6$. Decompose $x =: x^{\not\circ} + x^{\circ}$ according to Lemma 4.18 and define $b^{\not\circ}_v := x^{\not\circ}(\delta^{\text{in}}_G(v))$.

Lemma 4.19. *Suppose that $b_v \geq 0$ for all $v \in V \setminus \{s\}$. Let each $v \in V$ with $b^{\not\circ}_v > 0$ sample one edge, where the probability to choose arc $a \in \delta^{\text{in}}_G(v)$ is $x^{\not\circ}_a / b^{\not\circ}_v$. Then the resulting arc set is a directed tree T rooted at the source s spanning all nodes with non-zero demand. Moreover, denoting by $x_T \in \mathbb{R}^{2m}$ the (unique) flow with $Ax_T = b$ and $x_{Ta} = 0$ for any arc $a \in A \setminus T$, it holds that* $\mathrm{E}[\mathbb{1}^T W x_T] = \mathbb{1}^T W x^{\not\circ}$.

Proof. Recall that $Ax^{\not\circ} = b$. As $b_v \geq 0$ for all $v \in V \setminus \{s\}$, any such v that has an outgoing arc $a \in \delta^{\text{out}}_G(v)$ carrying non-zero flow $x^{\not\circ}_a > 0$ or that satisfies $b_v > 0$ must have an incoming arc $a' \in \delta^{\text{in}}_G(v)$ carrying non-zero flow $x^{\not\circ}_{a'} > 0$. Thus, we can inductively construct a directed path ending at any such node by following the incoming arc of the (current) first node of the path, until this first node is s. Note that it is not possible that we close a directed cycle, as $x^{\not\circ}$ induces a DAG by construction, see Lemma 4.18. Hence, the sampled graph is a DAG in which each node that sampled an arc is reachable from s and as each node sampled at most one incoming arc, it is moreover a tree T rooted at s. As we observed that each v with $b_v > 0$ has an incoming edge carrying non-zero flow, each such v sampled an edge, implying that T spans the nodes of non-zero demand. Accordingly, there exists a unique flow x_T on T such that $Ax_T = b$ holds.

It remains to show that $\mathrm{E}[\mathbb{1}^T W x_T] = \mathbb{1}^T W x^{\not\circ}$. We show this by induction on n, where the base case of $n = 1$ is trivial. Assuming the claim is true for n nodes, consider $n + 1$ nodes and let v be a node so that $x^{\not\circ}_a = 0$ for all $a \in \delta^{\text{out}}(v)$. Such a node must exist, as $x^{\not\circ}$ induces a DAG. If $b_v = 0$, then $b^{\not\circ}_v = b_v = 0$ and the claim is trivially true, so suppose $b^{\not\circ}_v = b_v > 0$. We interpret the random choice of v as follows: v picks an in-going arc $a = (w, v)$ and we route b_v units of flow from w to v, changing demands b to b' accordingly, i.e., $b'_v := b_v - b_v = 0$, $b'_w = b_w + b_v$. In expectation, this induces cost

$$C := \sum_{a \in \delta^{\text{in}}(v)} w_a b_v \cdot \frac{x^{\not\circ}_a}{b^{\not\circ}_v} \overset{b^{\not\circ}_v = b_v}{=} \sum_{a \in \delta^{\text{in}}(v)} w_a x^{\not\circ}_a.$$

For all w for which there is $a = (w, v) \in \delta^{\text{in}}(v)$ with $x_a > 0$, the modified demands b' satisfy

1. $\mathrm{E}[b'_w] = b_w + \frac{x^{\not\circ}_{(w,v)}}{b^{\not\circ}_v} \cdot b_v = b_w + x^{\not\circ}_{(w,v)}$, where the equality follow from $b^{\not\circ}_v = b_v$,

2. $b'_v = 0$,

3. $b'_u = b_u$ at all other nodes u that are not adjacent to v.

Recall that for independent random variables X and Y, it holds that $\mathrm{E}[X \cdot Y] = \mathrm{E}[X]\,\mathrm{E}[Y]$. As the random variable b' does not depend on the random choices of any nodes but v, it is independent of the tree T' sampled by the node set $V \setminus \{v\}$. Denoting by X_w the cost of routing one unit of flow from s to $w \in T \setminus \{v, s\}$ on T', by linearity of expectation we thus obtain

$$\mathrm{E}\left[\mathbb{1}^T W x_T\right] = C + \mathrm{E}\Big[\sum_{w \in T\setminus\{v,s\}} b'_w X_w\Big] = C + \sum_{w \in T\setminus\{v,s\}} \mathrm{E}[b'_w X_w] = C + \sum_{w \in T\setminus\{v,s\}} \mathrm{E}[b'_w]\,\mathrm{E}[X_w].$$

Examining the sum in the final term, observe that when deleting v from the graph, the respective restriction of $x^{\not\circ}$ routes exactly demands $\mathrm{E}[b'_w] \geq b_w \geq 0$ from s to each $w \in V \setminus \{v, s\}$. Thus, induction hypothesis and linearity of expectation complete the proof

$$\sum_{w \in T\setminus\{v,s\}} \mathrm{E}\,[b'_w]\,\mathrm{E}\,[X_w] = \mathbb{1}^T W x^{\not\circ} - \sum_{a \in \delta^{\mathrm{in}}(v)} w_a x_a^{\not\circ} = \mathbb{1}^T W x^{\not\circ} - C. \qquad \square$$

Our goal is now to apply Lemma 4.19 followed by Markov's bound in order to show that sampling edges is sufficient for obtaining a good tree solution. Unfortunately, the decomposition $x = x^{\not\circ} + x^{\circ}$ is unknown, and Lemma 4.19 critically depends on the fact that $x^{\not\circ}$ is acyclic. A naive solution would be to sample sufficiently often according to x so that each arc has at least the same probability to be sampled when sampling from the "correct" distribution given by $x^{\not\circ}$. Unfortunately, it is possible that an arc a satisfies $x_a^{\not\circ} \ll x_a^{\circ}$, conflicting with the requirement of sampling few arcs. We overcome this obstacle by replacing such "bad" arc by a corresponding path in the spanner. We first make this notation formal.

Definition 4.20. *Denote $w_{\min} := \min\{w_a : a \in A\}$ and partition A into sets $A_{v,k} := \{a \in \delta^{\mathrm{in}}(v) : 2^{k-1} w_{\min} \leq w_a < 2^k w_{\min}$ and $x_a > 0\}$, where $v \in V$ and $k \in \mathbb{Z}_{>0}$. The set $A_{v,k} \neq \emptyset$ is called* bad, *if $2\alpha\lambda \sum_{a \in A_{v,k}} x_a^{\not\circ} \leq \sum_{a \in A_{v,k}} x_a^{\circ}$. We say that $a \in A$ is bad if the set $A_{v,k}$ containing a is bad.*

We show that redirecting flow on bad arcs over the spanner only incurs small cost:

Lemma 4.21. *Redirecting the flow $x_a^{\not\circ}$ of each bad arc $a = (v, w)$ over a shortest v-w path on an (undirected) α-spanner for the graph with weights w^- incurs an additional cost of at most $\varepsilon' \mathbb{1}^T W x^*$.*

Proof. As the maximum ratio between the weights of an edge in opposing directions is λ, an undirected α-spanner for weights w^- clearly allows, for any $a = (v, w) \in A$, to find a directed path of length at most $\alpha\lambda w_a$ from v to w. The cost for rerouting the flow on bad edges over the spanner is thus bounded by

$$\sum_{\mathrm{bad} A_{v,k}} \sum_{a \in A_{v,k}} \alpha\lambda w_a x_a^{\not\circ} < \sum_{\mathrm{bad} A_{v,k}} 2^k \alpha\lambda w_{\min} \sum_{a \in A_{v,k}} x_a^{\not\circ} \leq \sum_{\mathrm{bad} A_{v,k}} 2^{k-1} w_{\min} \sum_{a \in A_{v,k}} x_a^{\circ}$$
$$\leq \sum_{\mathrm{bad} A_{v,k}} \sum_{a \in A_{v,k}} w_a x_a^{\circ} \leq \mathbb{1}^T W x^{\circ}.$$

Furthermore, since x is $(1 + \varepsilon)$-approximate and $\mathbb{1}^T W x^{\not\circ} \geq \mathbb{1}^T W x^*$ by optimality of x^*,

$$\mathbb{1}^T W x^{\circ} = \mathbb{1}^T W x - \mathbb{1}^T W x^{\not\circ} \leq (1 + \varepsilon')\mathbb{1}^T W x^* - \mathbb{1}^T W x^* = \varepsilon' \mathbb{1}^T W x^*. \qquad \square$$

We now are ready to show that sampling sufficiently many arcs according to x and adding the spanner is, in expectation, almost as good as directly sampling according to $x^{\not\circ}$.

Lemma 4.22. *Suppose that $b_v \geq 0$ for all $v \in V \setminus \{s\}$. For each set $A_{v,k} \neq \emptyset$, sample arc $a \in A_{v,k}$ with independent probability*

$$p_a := \min \left\{ \frac{(2\alpha\lambda + 1)x_a}{\sum_{a' \in A_{v,k}} x_{a'}}, 1 \right\},$$

and repeat this procedure until at least one arc from $A_{v,k}$ is selected. Add an arbitrary such arc, and add all spanner arcs. Then the expected cost of an optimal solution on the induced graph is bounded by $(1 + 2\varepsilon')\mathbb{1}^T W x^*$.

Proof. By Lemma 4.19, sampling arc $a \in \delta^{\text{in}}(v)$ with probability $x_a^{\not\circ}/b_v^{\not\circ}$ at $v \in V$ with $b_v^{\not\circ} > 0$ results in a flow x_T on a tree of expected cost at most $\mathbb{1}^T W x^{\not\circ}$. Observe that if the probability to select $a \in \delta^{\text{in}}(v)$ is *at least* as large (and we are guaranteed to select at least one arc from $\delta^{\text{in}}(v)$ for each $v \in V$), the expected cost can only become smaller. Consider an arc $a \in A_{v,k}$ with $p_a < 1$ that is not bad. It satisfies that

$$p_a \geq \frac{(2\alpha\lambda + 1)x_a^{\not\circ}}{\sum_{a' \in A_{v,k}} x_{a'}} = \frac{(2\alpha\lambda + 1)x_a^{\not\circ}}{\sum_{a' \in A_{v,k}} x_{a'}^{\not\circ} + \sum_{a' \in A_{v,k}} x_{a'}^{\circ}} > \frac{(2\alpha\lambda + 1)x_a^{\not\circ}}{(2\alpha\lambda + 1)\sum_{a' \in A_{v,k}} x_{a'}^{\not\circ}} \geq \frac{x_a^{\not\circ}}{b_v^{\not\circ}},$$

i.e., the probability to select such an arc is sufficiently large. The same is trivially true if $p_a = 1$. Hence, it remains to address bad arcs.

To this end, we apply Lemma 4.21 to see that we can reroute their flow over the spanner at an additive additional cost of at most $\varepsilon' \mathbb{1}^T W x^*$. Logically, we can reflect this by replacing the weight of such an arc by the weight of a respective shortest path in the spanner and adding all bad arcs to the sample (i.e., "sampling" each bad arc with probability 1); denoting the new weights by w', we are then guaranteed that the union of the spanner and the actually sampled arcs contains a solution of expected cost at most

$$\mathrm{E}\left[\mathbb{1}^T W' x_T\right] = \mathbb{1}^T W' x \leq \mathbb{1}^T W x + \varepsilon' \mathbb{1}^T W x^* \leq (1 + 2\varepsilon')\mathbb{1}^T W x^*. \qquad \square$$

It remains to apply Markov's bound in order to get the following corollary.

Corollary 4.23. *Performing the sampling procedure of Lemma 4.22 and returning an optimum primal tree solution on the graph induced by the sampled arcs and the spanner yields a $(1 + \varepsilon)$-approximation with probability at least $\frac{2}{3}$.*

Proof. Lemma 4.22 shows that the expected cost of an optimal solution on the stated arc set is at most $(1 + 2\varepsilon')\mathbb{1}^T W x^* = (1 + \frac{\varepsilon}{3})\mathbb{1}^T W x^*$. Applying Markov's bound to the (positive) random variable that is the amount by which this cost exceeds that of an optimal solution shows that the cost is at most $(1 + \varepsilon)\mathbb{1}^T W x^*$ with probability at least $\frac{2}{3}$. Lemma 4.17 shows that there is an optimal solution that is a forest, and as there is only one source, it is a tree. $\square$

It remains to bound the number of sampled arcs, as this affects the running time of the procedure.

Lemma 4.24. *Suppose $k_{\max} \in \mathbb{Z}_{>0}$ is such that $w \leq 2^{k_{\max}} w_{\min} \mathbb{1}$. Then the number of non-spanner arcs sampled by the procedure in Lemma 4.22 is bounded by $O(\alpha\lambda k_{\max} n)$ with probability $\frac{2}{3}$.*

Proof. When sampling from $A_{v,k}$ once, in expectation at most $(2\alpha\lambda + 1)$ arcs are selected. If p is the probability to select at least one arc from $A_{v,k}$ in an attempt, we can thus bound the number of expected arcs chosen by $\sum_{i=0}^{\infty}(1 - p)^i(2\alpha\lambda + 1) = \frac{2\alpha\lambda+1}{p}$. If $p_a \geq 1$ for any $a \in A_{v,k}$, then $p = 1$. Otherwise,

$$1 - p = \prod_{a \in A_{v,k}} (1 - p_a) = \prod_{a \in A_{v,k}} \left(1 - \frac{(2\alpha\lambda + 1)x_a}{\sum_{a' \in A_{v,k}} x_{a'}}\right) < \prod_{a \in A_{v,k}} \frac{\sum_{a' \in A_{v,k} \setminus \{a\}} x_{a'}}{\sum_{a' \in A_{v,k}} x_{a'}} \le \prod_{a \in A_{v,k}} \left(1 - \frac{1}{|A_{v,k}|}\right)$$
$$= \left(1 - \frac{1}{|A_{v,k}|}\right)^{|A_{v,k}|}.$$

This final term is at most $\frac{1}{e}$ if $|A_{v,k}| > 1$, and trivially only $1 < (2\alpha\lambda + 1)$ arcs can be selected from $A_{v,k}$ in case $|A_{v,k}| = 1$. As the sets $A_{v,k}$ form a partition of A, by linearity of expectation we conclude that the expected number of selected arcs is in $O(\alpha\lambda k_{\max} n)$. By Markov's bound, the claim follows. □

While in general there is no guarantee that $k_{\max}$ is small, this can be easily fixed by a preprocessing step.

Observation 4.25. *For demands $b_v \in \{0, 1\}$ at all $v \in V \setminus \{s\}$, we may assume that $k_{\max} = O(\log(\varepsilon^{-1} n))$, at the expense of running the algorithm twice for approximation parameter $\varepsilon' \in \Theta(\varepsilon)$.*

Proof. Once an approximation x, y to $P^{\text{asym}}(G, w, b)$ is known that is at least polynomial say n^c for some constant c, i.e., $\mathbb{1}^T W x \le n^c b^T y \le n^c \mathbb{1}^T W x^*$, we can safely delete all arcs of larger weight than $\mathbb{1}^T W x$. This can be seen as follows. As by Lemma 4.17, a tree solution exists and as demands are integers, all flows in an optimal tree solution x^* are integer as well. Assume there is an arc with $w_e > \mathbb{1}^T W x$ that carries non-zero flow in x^*. Then $x^*_e \ge 1$ and thus $\mathbb{1}^T W x^* > \mathbb{1}^T W x$, which is a contradiction. Hence, arcs of weight more than $\mathbb{1}^T W x$ can be deleted from the graph without effecting any optimal solution. On the other hand, suppose that we computed a $(1 + \varepsilon/2)$-approximate (non-tree) solution for $P^{\text{asym}}(G, w, b)$ and suppose that an arc has weight less than $t := \varepsilon \mathbb{1}^T W x / (2n^3)$. Consider the weights w' that are defined as $w'_a = w_a$ if $w_a > t$ and $w'_a = t$ if $w_a \le t$. It follows that an optimal solution x^* of $P^{\text{asym}}(G, w, b)$ satisfies

$$\begin{aligned}\mathbb{1}^T W' x^* &= \sum_{a : w_a > t} w_a x^*_a + t \sum_{a : w_a \le t} x^*_a \le \sum_{a : w_a > t} w_a x^*_a + t \frac{n^3}{2} \\ &\le \sum_{a : w_a > t} w_a x^*_a + \frac{(1 + \varepsilon/2)\varepsilon}{4} \mathbb{1}^T W x^* \le \left(1 + \frac{\varepsilon}{2}\right) \mathbb{1}^T W x^*,\end{aligned} \tag{4.13}$$

where, in the first inequality, we used that $b_v \in \{0, 1\}$ for all nodes $v \in V \setminus \{s\}$ and that there are at most $n^2/2$ many arcs. Thus, we can increase the weights of light arcs, delete heavy arcs, and solve the instance $P^{\text{asym}}(G, w', b)$ with approximation guarantee $1 + \varepsilon/4$. Let us call x' the primal solution computed. We then get

$$\mathbb{1}^T W x' \le \mathbb{1}^T W' x' \le \left(1 + \frac{\varepsilon}{4}\right) \mathbb{1}^T W' x^* \overset{(4.13)}{\le} \left(1 + \frac{\varepsilon}{4}\right)\left(1 + \frac{\varepsilon}{2}\right) \mathbb{1}^T W x^* \le (1 + \varepsilon) \mathbb{1}^T W x^*.$$

The new weights satisfy $(w)_{\max}/w_{\min} \le 2n^3/\varepsilon$, which shows the claim. □

We summarize our results on constructing a primal tree solution as follows.

Theorem 4.26. *Assume that $b_v \in \{0, 1\}$ for all $v \in V \setminus \{s\}$. There is a Las Vegas algorithm using the oracle given by Corollary 4.4 that computes a $(1 + \varepsilon)$-approximate primal-dual pair, where the primal solution has non-zero flow only on the arcs of a tree. In each attempt to determine a solution, the algorithm calls the oracle $O((\varepsilon^{-2} + \log \alpha + \log \lambda)\alpha^2 \lambda^2 \log n)$ times, samples $O(\alpha\lambda n \log n)$ arcs from G, computes an optimal tree solution on the union of the sampled arcs and the spanner, and succeeds with probability at least $\frac{1}{3}$.*

Proof. By Theorem 4.16 and Corollary 4.4, we can obtain a primal-dual pair of solutions with approximation guarantee $\varepsilon' \in \Theta(\varepsilon)$ using the stated number of calls to the oracle. By Observation 4.25, we can ensure that $k_{\max} = O(\log(\varepsilon^{-1}n))$ without affecting the asymptotic bounds on the number of oracle calls. Lemma 4.24 shows that the sampling procedure of Lemma 4.22 results in the stated number of sampled arcs with probability at least $\frac{2}{3}$. If this is not the case, we simply abort and this attempt to compute a solution fails. Moreover, Corollary 4.23 states that computing an optimal tree solution using only sampled and spanner arcs yields a $(1+\varepsilon)$-approximate solution with probability at least $\frac{2}{3}$; we compute such a solution if the number of sampled arcs is indeed in $O(\alpha\lambda n \log n)$. Applying the union bound, we conclude that with probability at least $\frac{1}{3}$, both the number of sampled arcs is sufficiently small and the resulting solution is of sufficient quality. □

Let us assume that we called the gradient descent algorithm for a $(1+\varepsilon/6)$-approximate solution and denote its output solution with y, then the guarantee is that $b^T y \geq b^T y^*/(1+\varepsilon/6)$. Let us assume $y_s = 0$ and let $\bar{x}$ be the primal tree solution from the above theorem. We call $\bar{y}_v$ the length of the s-v path in the tree. Note that $\bar{y}$ does not need to be a feasible dual solution w.r.t. the whole graph. Still using the above theorem, we have in case of success that

$$b^T\bar{y} = \mathbb{1}^T W\bar{x} \leq \left(1+\frac{\varepsilon}{6}\right)b^T y^* \leq \left(1+\frac{\varepsilon}{6}\right)^2 b^T y \leq \left(1+\frac{\varepsilon}{2}\right)b^T y \quad \text{for } \varepsilon \leq 1. \tag{4.14}$$

Let us call a node *ε-good*, if $y_v \geq \bar{y}_v/(1+\varepsilon)$ and note that since $(W^{-1}A^T y)_{\max} = 1$, it follows that for an ε-good node $\bar{y}_v \leq (1+\varepsilon)y_v \leq (1+\varepsilon)y_v^*$, i.e., we know a $(1+\varepsilon)$-approximation on its distance to the source s. The following lemma shows that the good nodes constitute at least a constant fraction of $b^T y^*$.

Lemma 4.27. *Let $b_v \geq 0$ for all $v \in V \setminus \{s\}$, $y \in \mathbb{R}^n$ with $(W^{-1}A^T y)_{\max} \leq 1$, $y_s = 0$ and $b^T y \geq \frac{b^T y^*}{1+\varepsilon/6}$, and let $X := \{v \in V \setminus \{s\} : y_v < \bar{y}_v/(1+\varepsilon)\}$ for $\varepsilon < 1$. Then, $\sum_{v\in X} b_v y_v^* \leq \frac{2}{3} b^T y^*$.*

Proof. Using (4.14) and the definition of X yields

$$\begin{aligned}
\frac{\varepsilon}{2}\sum_{v\in X} b_v y_v &\leq \frac{\varepsilon}{2}\sum_{v\in X} b_v y_v + \left(1+\frac{\varepsilon}{2}\right)b^T y - b^T\bar{y} \\
&= (1+\varepsilon)\sum_{v\in X} b_v y_v + \left(1+\frac{\varepsilon}{2}\right)\sum_{v\in V\setminus X} b_v y_v - b^T\bar{y} \\
&< \sum_{v\in X} b_v \bar{y}_v + \left(1+\frac{\varepsilon}{2}\right)\sum_{v\in V\setminus X} b_v y_v - b^T\bar{y} \leq \frac{\varepsilon}{2}\sum_{v\in V\setminus X} b_v y_v,
\end{aligned}$$

where the last inequality follows from $y_v \leq y_v^* \leq \bar{y}_v$ for $v \in V \setminus X$. It thus follows that $\sum_{v\in V\setminus X} b_v y_v > \frac{1}{2} b^T y$ and hence for $\varepsilon \leq \frac{1}{2}$, we get

$$\sum_{v\in X} b_v y_v^* = b^T y^* - \sum_{v\in V\setminus X} b_v y_v^* \leq \left(1+\frac{\varepsilon}{6}\right)b^T y - \sum_{v\in V\setminus X} b_v y_v < \left(\frac{1}{2}+\frac{\varepsilon}{6}\right)b^T y \leq \frac{2}{3}b^T y^*. \quad \square$$

We conclude that by inspecting the fractions $y_v/\bar{y}_v$ for every v, we can identify a set of good nodes for which we have $(1+\varepsilon)$-approximate distance estimates. We proceed by setting their demand b_v to zero and recurse on the remaining nodes. As we have proven in the previous lemma, the good nodes constitute at least a constant fraction of $b^T y^*$ and thus setting their demand to zero will lead to a reduction in $b^T y^*$ by a constant fraction. Hence, after a logarithmic (in $b^T y^* = O(n^2\|w\|_\infty)$) number of iterations this procedure will terminate and we will know $(1+\varepsilon)$-approximate distances for every single node $v \in V$. This yields the following theorem:

Theorem 4.28. *There is an algorithm for the single source shortest path problem that, for every node $v \in V \setminus \{s\}$, computes a path P_v from v to the source node s that is at most $(1+\varepsilon)$ times longer than the shortest path from v to s by using only $\tilde{O}((\varepsilon^{-1}\alpha\lambda)^2 \log^2(n)\log(\|w\|_\infty))$ many calls to the oracle given by Corollary 4.4.*

4.3 Implementation in Various Models of Computation

It remains to describe how to implement the above framework in the different models of computation. We start by formally defining the models.

4.3.1 Models

Broadcast Congested Clique model

In the Broadcast Congested Clique model, every of the n nodes can talk to every other node, i.e., there is an underlying communication graph that is complete. Computation proceeds in rounds. In one round every node does the following three actions: (1) It sends *the same one* message of size $O(\log n)$ to all other nodes (hence the term broadcast). (2) It receives the messages from the other nodes. (3) It can perform arbitrary local computations. The complexity in this model is measured in number of rounds needed.

The *input* of the problem is distributed among the nodes as well, i.e., initially every node knows its ID, as well as the ID of its neighbors. In the case of the approximate asymmetric transshipment problem, every node knows the value of ε, its demand, as well as the forward and backward weights of its incident edges. Note that for the single source shortest path problem this means that every node knows whether it is the source or not.

At *termination*, every node needs to know its part of the output, i.e., it needs to know its potential or distance in the successive shortest path case. If we are also interested in a primal solution, in the transshipment problem, every node needs to know the primal values of its adjacent edges, and in the single source shortest path case, every node should know the next edge among its incident edges on the path to the source.

Broadcast Congest model

The Broadcast Congest model is identical to the Broadcast Congested Clique model up to one important difference. Communication is restricted to edges in the input graph G. This difference has big consequences. Clearly, denoting by D the diameter of the graph, $\Omega(D)$ rounds are necessary in this model in order to solve the transshipment problem. Furthermore, even if the diameter is logarithmically bounded in n, Das Sarma et al. showed that $\Omega(\sqrt{n}/\log n)$ are necessary [Sar+12].[6]

Multipass Streaming model

In the Multipass Streaming model, the computation proceeds in passes [HRR98]. In every pass, the edge set is given to the algorithm as a stream and the assumption is that the edge set is too large for the algorithm to store it. The goal is to solve the problem at hand with little number of passes while keeping the space requirement small.

[6]The bounds apply also for randomized algorithms and the approximate version of the problem when the approximation ratio is polynomial in n.

4.3.2 Implementation

In this section, we describe for each of the above three models and the standard RAM model of computation how our algorithm can be implemented in them. We first recall our results from Corollary 4.15: Algorithm GradientTransship can be executed using a total of $O((\varepsilon^{-2} + \log \alpha + \log \lambda)\alpha^2\lambda^2 \log n)$ operations of the following types:

1. oracle call
2. computation of $(W^{-1}A^T\pi)_{\max}$ for a given π
3. computation of $\nabla\Phi_\beta(\pi)$ for given β and π
4. scalar product $\tilde{b}^T h$ for given $\tilde{b}$ and h
5. comparisons of and multiplications with scalar values

We will thus describe for each of the models how to implement these steps. Fortunately, this is rather straightforward.

Broadcast Congested Clique model

We start with items 2 and 3. Every node v can within one round learn the values π_w for all its neighbors w. Thus, it can locally compute $z_a := (W^{-1}A^T\pi)_a$ for all arcs $a \in \delta_G^{\text{in}}(v)$. Then, every node can broadcast $\max\{z_a : a \in \delta_G^{\text{in}}(v)\}$ to all other nodes and thus every node can learn the value of $(z)_{\max} = (W^{-1}A^T\pi)_{\max}$ within a constant number of rounds. Let us recall the definition of $\Phi_\beta(\pi) := \text{lse}_\beta(W^{-1}A^T\pi)$ and the form of its gradient $\nabla\Phi_\beta(\pi) = AW^{-1}\nabla\,\text{lse}_\beta(W^{-1}A^T\pi)$. Note that for $v \in V$, the gradient takes the form

$$(AW^{-1}\nabla\,\text{lse}_\beta(z))_v = \frac{1}{\beta\sum_{a\in A} e^{\beta z_a}}\Big(\sum_{a\in\delta^{\text{in}}(v)} \frac{e^{\beta z_a}}{w_a} - \sum_{a\in\delta^{\text{out}}(v)} \frac{e^{\beta z_a}}{w_a}\Big). \tag{4.15}$$

Thus, as every node knows its incident edges, it can locally compute $e^{\beta z_a}$ for all its incident edges $a \in \delta^{\text{in}}(v) \cup \delta^{\text{out}}(v)$ and thus it can broadcast the value $\sum_{a\in\delta^{\text{in}}(v)} e^{\beta z_a}$. Then, every node can learn $\sum_{a\in A} e^{\beta z_a} = \sum_{v\in V}\sum_{a\in\delta^{\text{in}}(v)} e^{\beta z_a}$ and thus compute $(AW^{-1}\nabla\,\text{lse}_\beta(z))_v$ according to (4.15) locally. Items 4 and 5 are even more straightforward, as we can assume that the values of $\tilde{b}$ and h are globally known every node can do these computations locally for free. For item 1, i.e, the implementation of the oracle, we have already described in Section 4.2.2 that the optimal solution to $P^{\text{sym}\perp}(S, w, \nabla\Phi_\beta(\pi))$ on an α-spanner S gives an α-apprixmate solution for the problem $P^{\text{sym}\perp}(G, w, \nabla\Phi_\beta(\pi))$. It suffices to compute this α-spanner S once initially as the weights of all oracle problems are identical, namely w. Whenever, the algorithm calls the oracle, we can then make sure that every node knows $\nabla\Phi_\beta(\pi)$ and can locally compute an optimal solution to $P^{\text{sym}\perp}(S, w, \nabla\Phi_\beta(\pi))$. For the initial computation of an α-spanner, note that already Baswana and Sen gave a randomized method to construct a $2k-1$ spanner of expected size $O(kn^{1+1/k})$ within $O(k^2)$ rounds in the Broadcast Congest model [BS07, Theorem 5.1].This strategy can be made deterministic. In [CPS17], Censor-Hillel et al. show a comparable result. They show how to, in the Broadcast Congested Clique model, deterministically construct a $(2k-1)$-spanner with $O(kn^{1+1/k}\log n)$ edges in $O(k \log n)$ rounds. Thus, for $k = \log(n)$, this result directly gives a spanner that fulfills our requirements. We conclude that after an initial computation of the spanner in $O(\log^2 n)$ rounds, every iteration can be implemented within constant number of rounds. Thus, we obtain the following theorem.

Theorem 4.29 (Broadcast Congested Clique). *1. There is an algorithm that, for any $0 < \varepsilon \leq 1$, computes a $(1+\varepsilon)$-approximate primal/dual solution pair to the asymmetric transshipment problem in the* Broadcast Congested Clique *model within $\tilde{O}(\varepsilon^{-2}\lambda^2 \operatorname{polylog} n)$ rounds of communication.*

2. There is an algorithm for the asymmetric single source shortest path problem that, for any $0 < \varepsilon \leq 1$, computes a $(1+\varepsilon)$-approximate shortest path for every node in the Broadcast Congested Clique *model within $\tilde{O}(\varepsilon^{-2}\lambda^2 \operatorname{polylog}(n, \|w\|_\infty))$ rounds of communication.*

Broadcast Congest model

In order to transfer our results from the Broadcast Congested Clique to the Broadcast Congest model, we use two different previous results. The first is a classical result in distributed computing. The idea is to pipeline information over a BFS tree.

Lemma 4.30 (cf. [Pel00]). *Suppose each $v \in V$ holds $m_v \in \mathbb{Z}_{\geq 0}$ messages of $O(\log n)$ bits each. Let $M := \sum_{v \in V} m_v$. Then all nodes in the graph can receive all M messages within $O(M + D)$ rounds.*

For the single source shortest path case we use the following more recent result:

Lemma 4.31 ([HKN16]). *Given any weighted undirected graph G and source node $s \in V$, there is an $\tilde{O}(\sqrt{n})$-round deterministic distributed algorithm in the* Broadcast Congest *model*[7] *that computes an overlay network $G' = (V', E')$ with edge weights $w' : E' \to \{1, \dots, \operatorname{poly} n\}$ and some additional information for every node with the following properties.*

- *$|V'| = \tilde{O}(\sqrt{n}\varepsilon^{-1})$ and $s \in V'$.*
- *For $\varepsilon' = \Theta(\varepsilon)$, each node $v \in V$ can infer a $(1+\varepsilon)$-approximation of its distance to s from $(1+\varepsilon')$-approximations of the distances between s and each $t \in V'$.*

Using the above results, we obtain the following theorem.

Theorem 4.32 (Broadcast Congest). *1. There is an algorithm that, for any $0 < \varepsilon \leq 1$, computes a $(1+\varepsilon)$-approximate primal/dual solution pair to the asymmetric transshipment problem in the* Broadcast Congest *model within $\tilde{O}(n\varepsilon^{-2}\lambda^2)$ rounds of communication.*

2. There is a Las Vegas algorithm that, for any $0 < \varepsilon \leq 1$, computes a $(1+\varepsilon)$-approximate shortest path tree to the asymmetric single source shortest path problem in the Broadcast Congest *model in $\tilde{O}(\sqrt{n} + D)\varepsilon^{-2}\lambda^2)$ rounds.*

Proof. 1. We obtain the result by using Lemma 4.30 and simulating the broadcast congested clique in every iteration. It holds that $M = O(n \log n)$ in this case and thus the result follows from the first part of Theorem 4.29.

2. It is easy to see that the construction from Henzinger et al. in Lemma 4.31 also works in the asymmetric setting. Hence the bound follows using the second part of Theorem 4.29 on the overlay network. □

[7]All communication of the algorithm in [HKN16] meets the constraint that in each round, each node sends the same message to all neighbors (which is the difference between the Broadcast Congest and the standard Congest model).

Multipass Streaming model

The main observation is that we can apply the same approach as before with $O(n \log n)$ space as this enables us to initially compute the spanner and store it during the entire computation. Similarly, we can store the variables related to the nodes within $O(n \log n)$ space. The gradient $\nabla\Phi_\beta(\pi)$ can be evaluated in a single pass. It follows that $\varepsilon^{-2}\lambda^2 \operatorname{polylog} n$ passes suffice and $O(n \log n)$ space is enough.

Theorem 4.33 (Multipass Streaming). *1. There is an algorihtm that, for any $0 < \varepsilon \leq 1$, computes a deterministic $(1+\varepsilon)$-approximation to the asymmetric transshipment problem in the* Multipass Streaming *model in $\tilde{O}(\varepsilon^{-2}\lambda^2 \operatorname{polylog} n)$ passes with $O(n \log n)$ space.*

2. *There is a Las Vegas algorithm that, for any $0 < \varepsilon \leq 1$, computes a $(1+\varepsilon)$-approximate shortest path tree to the asymmetric single source shortest path problem in the* Multipass Streaming *model in $\tilde{O}(\varepsilon^{-2}\lambda^2 \operatorname{polylog}(n, \|w\|_\infty))$ passes with $O(n \log n)$ space.*

RAM model

We turn back to the classical RAM model of computation and conclude the first part of the thesis with this. We will see that we can apply the results from Chapter 3 for implementing the oracle for the gradient descent algorithm and obtain an interesting result for the RAM model in this way. We need one more technical lemma that shows that we can implement an oracle for $P^{\text{sym}\perp}(G, w, \nabla\Phi_\beta(\pi))$ using an oracle for $P^{\text{asym}}(G, w, P^T\nabla\Phi_\beta(\pi))$, where P is a specific projection matrix:

Lemma 4.34. *Let $\pi \in \mathbb{R}^n$ be a γ-approximate solution to $P^{\text{reci}}(G, w, b)$, i.e., $b^T\pi = 1$ and $(W^{-1}A^T\pi)_{\max} \leq \gamma(W^{-1}A^T\pi^*)_{\max}$. Let $P = (I - \pi b^T)$ and let x, h be an η-approximate primal/dual solution pair to $P^{\text{asym}}(G, w, P^T\nabla\Phi_\beta(\pi))$. Then $(x, \pi^T\nabla\Phi_\beta(\pi)), Ph/(1+\gamma)$, is an $\alpha(1+\gamma)$-approximate primal/dual solution pair to $P^{\text{sym}\perp}(G, w, \nabla\Phi_\beta(\pi))$.*

Proof. Let us first check feasibility. Clearly, $x \geq 0$ and $Ax + (\pi^T\nabla\Phi_\beta(\pi))b = P^T\nabla\Phi_\beta(\pi)$ as well as $b^TPh = 0$ by the definition of P. Moreover,

$$\begin{aligned}(W^{-1}A^TPh)_{\max} &\leq (W^{-1}A^Th)_{\max} + (W^{-1}A^T\pi)_{\max}|b^Th| \\ &= (W^{-1}A^Th)_{\max} \cdot \left(1 + \frac{(W^{-1}A^T\pi)_{\max}|b^Th|}{(W^{-1}A^Th)_{\max}}\right) \\ &\leq (W^{-1}A^Th)_{\max} \cdot \left(1 + \gamma(W^{-1}A^T\pi^*)_{\max} b^Ty^*\right) = (W^{-1}A^Th)_{\max} \cdot (1+\gamma),\end{aligned}$$

and thus also $Ph/(1+\gamma)$ is feasible. It remains to show the approximation guarantee. Note that $\mathbb{1}^TW_{-}x \leq \eta(P^T\nabla\Phi_\beta(\pi))^Th$ implies that

$$\mathbb{1}^TW_{-}x \leq \eta \cdot \nabla\Phi_\beta(\pi)^TPh = \eta(1+\gamma) \cdot \nabla\Phi_\beta(\pi)^T\frac{Ph}{(1+\gamma)}$$

and thus $(x, \pi^T\nabla\Phi_\beta(\pi)), Ph/(1+\gamma)$ is an $\eta(1+\gamma)$-approximate primal/dual pair. □

The proof of Lemma 4.14 implies that any iterate π during the run of the algorithm is a $2\alpha^2\lambda^2$-approximation. Notice that the naive thought, that any iterate should be an α-approximation as the initial solution is and the algorithm makes progress, is trappy as the algorithm guarantees progress in terms of $\Phi_\beta(\pi)$ and this does not directly imply progress in terms of $(W^{-1}A^T\pi)_{\max}$. In summary, the above lemma shows that we can get a primal/dual pair for $P^{\text{sym}\perp}(G, w, \nabla\Phi_\beta(\pi))$ with approximation ratio $\alpha\lambda(2\alpha^2\lambda^2+1) \leq 3\alpha^3\lambda^3$ by computing an α-spanner and solving $P^{\text{asym}}(S, w, P^T\nabla\Phi_\beta(\pi))$ optimally on it.

Recall that the asymmetric transshipment problem is a special case of the directed (uncapacitated) min-cost flow problem, as we have seen in Lemma 1.1 in the introduction. Thus, we may use the integer interior point method from Chapter 3 for solving $P^{\text{asym}}(S, w, P^T \nabla \Phi_\beta(\pi))$ and obtain an oracle running in $\tilde{O}(m + n^{3/2} \log(\|b, w\|_\infty))$ time. The $\tilde{O}(m)$-term is the result of the spanner computation, see for example the algorithm of Baswana and Sen [BS07] that can be used for this purpose. The term $\tilde{O}(n^{3/2} \log(\|b, w\|_\infty))$ is due to the min-cost flow computation on the sparse spanner, where $m = \tilde{O}(n)$, see Theorem 3.10. Hence, we obtain the following result.

Theorem 4.35. *There is a randomized algorithm that, for any $0 < \varepsilon \leq 1$, computes a $(1 + \varepsilon)$-approximation to the asymmetric transshipment problem in $\tilde{O}(\varepsilon^{-2} \lambda^{O(1)}(m + n^{3/2}) \log(\|b, w\|_\infty))$ time with high probability.*

We remark that in settings where $m = \Omega(n^{3/2})$, our method is thus nearly-linear in m, thus in terms of m optimal up to log-factors, and dominates over the method due to Sherman [She17] that runs in $O(m^{1+o(1)})$.

4.4 Conclusion

We have presented a gradient descent approach to the asymmetric transshipment problem and its special case the single source shortest path problem. When implemented in distributed and streaming models of computation this leads very efficient algorithms including the first non-trivial bounds for asymmetric transshipment in the Broadcast Congest and Broadcast Congested Clique models. We obtain the first algorithms or the single source shortest path problem in these models with near-optimal number of communication rounds.

This work leaves some interesting research questions open. Clearly, our result for the RAM model is interesting but also somewhat unsatisfactory. One would hope to find an algorithm with nearly linear running time in m for all settings of n and m and not only for $m = \Omega(n^{3/2})$. Another very interesting question that arises is the following: Does a similar approach work for the single source shortest path problem in the PRAM model? Finding a PRAM-algorithm for computing single source shortest paths or $(1 + \varepsilon)$-approximate shortest paths with nearly-linear work and poly-logarithmic depth would be the ultimate goal here.

References for Part I

[ABP17] Amir Abboud, Greg Bodwin, and Seth Pettie. "A Hierarchy of Lower Bounds for Sublinear Additive Spanners". In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*. Ed. by Philip N. Klein. SIAM, 2017, pp. 568–576.

[AN12] Ittai Abraham and Ofer Neiman. "Using Petal-Decompositions to Build a Low Stretch Spanning Tree". In: *STOC*. Ed. by Howard J. Karloff and Toniann Pitassi. ACM, 2012, pp. 395–406.

[AL62] G. Adelson-Velskii and E. M. Landis. "An algorithm for the organization of information". In: *Dokl.Akad.Nauk SSSR* 146.2 (1962), pp. 263–266.

[Ahu+92] Ravindra K. Ahuja, Andrew V. Goldberg, James B. Orlin, and Robert Endre Tarjan. "Finding minimum-cost flows by double scaling". In: *Math. Program.* 53 (1992), pp. 243–266.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows - Theory, Algorithms and Applications*. Prentice Hall, 1993, pp. I–XV, 1–846.

[Ahu+95] Ravindra K. Ahuja, Thomas L. Magnanti, James B. Orlin, and M.R. Reddy. "Chapter 1 Applications of network optimization". In: *Network Models*. Vol. 7. Handbooks in Operations Research and Management Science. Elsevier, 1995, pp. 1–83.

[BOR80] John J. Bartholdi, James B. Orlin, and H. Donald Ratliff. "Cyclic Scheduling via Integer Programs with Circular Ones". In: *Operations Research* 28.5 (1980), pp. 1074–1085.

[Bas+07] H. Bast, Stefan Funke, Domagoj Matijevic, Peter Sanders, and Dominik Schultes. "In Transit to Constant Time Shortest-Path Queries in Road Networks". In: *ALENEX*. SIAM, 2007.

[BS07] Surender Baswana and Sandeep Sen. "A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs". In: *Random Struct. Algorithms* 30.4 (2007). Announced at ICALP'03, pp. 532–563.

[BK14] Ruben Becker and Andreas Karrenbauer. "A Simple Efficient Interior Point Method for Min-Cost Flow". In: *Algorithms and Computation - 25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15-17, 2014, Proceedings*. Ed. by Hee-Kap Ahn and Chan-Su Shin. Vol. 8889. Lecture Notes in Computer Science. Springer, 2014, pp. 753–765.

[Bol98] B. Bollobas. *Modern Graph Theory*. Graduate Texts in Mathematics. Springer New York, 1998.

[BTZ98] Gerth Stølting Brodal, Jesper Larsson Träff, and Christos D. Zaroliagis. "A Parallel Priority Queue with Constant Time Operations". In: *J. Parallel Distrib. Comput.* 49.1 (1998). Announced at IPPS'97, pp. 4–21.

[Cen+15] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. “Algebraic Methods in the Congested Clique”. In: *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*. Ed. by Chryssis Georgiou and Paul G. Spirakis. ACM, 2015, pp. 143–152.

[CPS17] Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. “Derandomizing Local Distributed Algorithms under Bandwidth Restrictions”. In: *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*. Ed. by Andréa W. Richa. Vol. 91. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, 11:1–11:16.

[Coh97] Edith Cohen. “Using Selective Path-Doubling for Parallel Shortest-Path Computations”. In: *J. Algorithms* 22.1 (1997). Announced at ISTCS’93, pp. 30–56.

[Coh00] Edith Cohen. “Polylog-time and near-linear work approximation scheme for undirected shortest paths”. In: *J. ACM* 47.1 (2000). Announced at STOC’94, pp. 132–166.

[Coh+17] Michael B. Cohen, Aleksander Mądry, Piotr Sankowski, and Adrian Vladu. “Negative-Weight Shortest Paths and Unit Capacity Minimum Cost Flow in Õ $(m^{10/7} \log W)$ Time (Extended Abstract)”. In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*. Ed. by Philip N. Klein. SIAM, 2017, pp. 752–771.

[Cos98] Mario Costantini. “A Novel Phase Unwrapping Method Based on Network Programming”. In: *Geoscience and Remote Sensing, IEEE Transactions on* 36.3 (May 1998), pp. 813–821.

[DS08] Samuel I. Daitch and Daniel A. Spielman. “Faster approximate lossy generalized flow via interior point algorithms”. In: *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*. Ed. by Cynthia Dwork. ACM, 2008, pp. 451–460.

[Dan63] George B. Dantzig. *Linear programming and extensions*. Rand Corporation Research Study. Princeton, NJ: Princeton Univ. Press, 1963. XVI, 625.

[DGJ09] Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson, eds. *The Shortest Path Problem, Proceedings of a DIMACS Workshop, Piscataway, New Jersey, USA, November 13-14, 2006*. Vol. 74. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. DIMACS/AMS, 2009.

[Dij59] Edsger W. Dijkstra. “A Note on Two Problems in Connexion with Graphs”. In: *Numerische Mathematik* 1.1 (1959), pp. 269–271.

[DG13] Benjamin Doerr and Leslie Ann Goldberg. “Adaptive Drift Analysis”. In: *Algorithmica* 65.1 (2013), pp. 224–250.

[DJW10] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. “Multiplicative drift analysis”. In: *Genetic and Evolutionary Computation Conference, GECCO 2010, Proceedings, Portland, Oregon, USA, July 7-11, 2010*. Ed. by Martin Pelikan and Jürgen Branke. ACM, 2010, pp. 1449–1456.

[EK70] Jack Edmonds and Richard M. Karp. “Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems”. In: *Combinatorial Structures and Their Applications*. Gordon and Breach, New York, 1970, pp. 93–96.

[EK01] Jack Edmonds and Richard M. Karp. "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems". In: *Combinatorial Optimization - Eureka, You Shrink!, Papers Dedicated to Jack Edmonds, 5th International Workshop, Aussois, France, March 5-9, 2001, Revised Papers*. 2001, pp. 31–33.

[Elk17] Michael Elkin. "Distributed exact shortest paths in sublinear time". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*. 2017, pp. 757–770.

[EN16] Michael Elkin and Ofer Neiman. "Hopsets with Constant Hopbound, and Applications to Approximate Shortest Paths". In: *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*. Ed. by Irit Dinur. IEEE Computer Society, 2016, pp. 128–137.

[ET75] Shimon Even and Robert Endre Tarjan. "Network Flow and Testing Graph Connectivity". In: *SIAM J. Comput.* 4.4 (1975), pp. 507–518.

[FF62] D. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton, NJ, USA: Princeton University Press, 1962.

[FT87] Michael L. Fredman and Robert Endre Tarjan. "Fibonacci heaps and their uses in improved network optimization algorithms". In: *J. ACM* 34.3 (1987). Announced at FOCS'84, pp. 596–615.

[Gal16] François Le Gall. "Further Algebraic Algorithms in the Congested Clique Model and Applications to Graph-Theoretic Problems". In: *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*. Ed. by Cyril Gavoille and David Ilcinkas. Vol. 9888. Lecture Notes in Computer Science. Springer, 2016, pp. 57–70.

[Gha+15] Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. "Near-Optimal Distributed Maximum Flow: Extended Abstract". In: *PODC*. ACM, 2015, pp. 81–90.

[Gol97] Andrew V. Goldberg. "An Efficient Implementation of a Scaling Minimum-Cost Flow Algorithm". In: *J. Algorithms* 22.1 (1997), pp. 1–29.

[GK91] Andrew V. Goldberg and Michael Kharitonov. "On Implementing Scaling Push-Relabel Algorithms for the Minimum-Cost Flow Problem". In: *Network Flows And Matching, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, October 14-16, 1991*. Ed. by David S. Johnson and Catherine C. McGeoch. Vol. 12. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. DIMACS/AMS, 1991, pp. 157–198.

[GR97] Andrew V. Goldberg and Satish Rao. "Beyond the Flow Decomposition Barrier". In: *FOCS*. 1997, pp. 2–11.

[GT90] Andrew V. Goldberg and Robert E. Tarjan. "Finding Minimum-Cost Circulations by Successive Approximation". In: *Math. Oper. Res.* 15.3 (1990). Announced at STOC'87, pp. 430–466.

[GS78] Leonidas J. Guibas and Robert Sedgewick. "A Dichromatic Framework for Balanced Trees". In: *19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, 16-18 October 1978*. IEEE Computer Society, 1978, pp. 8–21.

[GO16] Venkatesan Guruswami and Krzysztof Onak. “Superlinear Lower Bounds for Multipass Graph Processing”. In: *Algorithmica* 76.3 (2016). Announced at CCC’13, pp. 654–683.

[Has83] Refael Hassin. “The minimum cost flow problem: A unifying approach to dual algorithms and a new tree-search algorithm”. In: *Math. Program.* 25.2 (1983), pp. 228–239.

[HRR98] Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. “Computing on data streams”. In: *External Memory Algorithms, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, May 20-22, 1998*. Ed. by James M. Abello and Jeffrey Scott Vitter. Vol. 50. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. DIMACS/AMS, 1998, pp. 107–118.

[HKN16] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. “A deterministic almost-tight distributed algorithm for approximating single-source shortest paths”. In: *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*. Ed. by Daniel Wichs and Yishay Mansour. ACM, 2016, pp. 489–498.

[HRW17] Monika Henzinger, Satish Rao, and Di Wang. “Local Flow Partitioning for Faster Edge Connectivity”. In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*. Ed. by Philip N. Klein. SIAM, 2017, pp. 1919–1938.

[Hit41] Frank L. Hitchcock. “The distribution of a product from several sources to numerous localities”. In: *J. Math. Phys. Mass. Inst. Tech.* 20 (1941), pp. 224–230.

[II86] Hiroshi Imai and Masao Iri. “Computational-geometric methods for polygonal approximations of a curve”. In: *Computer Vision, Graphics, and Image Processing* 36.1 (1986), pp. 31–41.

[Kan60] L. V. Kantorovich. “Mathematical Methods of Organizing and Planning Production”. In: *Management Science* 6.4 (1960). Translation of the original article from 1939., pp. 366–422.

[Kar00] David R. Karger. “Minimum cuts in near-linear time”. In: *J. ACM* 47.1 (2000), pp. 46–76.

[Kar73] A. V. Karzanov. “O nakhozhdenii maksimal’nogo potoka v setyakh spetsial’nogo vida i nekotorykh prilozheniyakh”. In: *Mathematicheskie Voprosy Upravleniya Proizvodstvom* (1973). Title transl.: On finding a maximum flow in a network with special structure and some applications.

[KT15] Ken-ichi Kawarabayashi and Mikkel Thorup. “Deterministic Global Minimum Cut of a Simple Graph in Near-Linear Time”. In: *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*. Ed. by Rocco A. Servedio and Ronitt Rubinfeld. ACM, 2015, pp. 665–674.

[Kel+14] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. “An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and its Multicommodity Generalizations”. In: *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*. Ed. by Chandra Chekuri. SIAM, 2014, pp. 217–226.

[Kel+13] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. "A Simple, Combinatorial Algorithm for Solving SDD Systems in Nearly-Linear Time". In: *STOC*. Ed. by Dan Boneh, Tim Roughgarden, and Joan Feigenbaum. ACM, 2013, pp. 911–920.

[Ket+08] Lutz Kettner, Kurt Mehlhorn, Sylvain Pion, Stefan Schirra, and Chee-Keng Yap. "Classroom examples of robustness problems in geometric computations". In: *Comput. Geom.* 40.1 (2008), pp. 61–78.

[KK12] Zoltán Király and P. Kovács. "Efficient implementations of minimum-cost flow algorithms". In: *CoRR* abs/1207.6381 (2012).

[KS97] Philip N. Klein and Sairam Subramanian. "A Randomized Parallel Algorithm for Single-Source Shortest Paths". In: *J. Algorithms* 25.2 (1997). Announced at STOC'92, pp. 205–220.

[KNS74] Darwin Klingman, Albert Napier, and Joel Stutz. "NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems". In: *Management Science* 20.5 (1974), pp. 814–821.

[Koo51] T.C. Koopmans. *Optimum Utilization of the Transportation System*. Cowles Commission papers : New series. Cowles Commission for Research in Economics, University of Chicago, 1951.

[KMP14] Ioannis Koutis, Gary L. Miller, and Richard Peng. "Approaching Optimality for Solving SDD Linear Systems". In: *SIAM J. Comput.* 43.1 (2014). Announced at FOCS'10, pp. 337–354.

[Kyn+16] Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman. "Sparsified Cholesky and multigrid solvers for connection laplacians". In: *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*. Ed. by Daniel Wichs and Yishay Mansour. ACM, 2016, pp. 842–850.

[LS14] Yin Tat Lee and Aaron Sidford. "Path Finding Methods for Linear Programming: Solving Linear Programs in $\tilde{O}(\sqrt{rank})$ Iterations and Faster Algorithms for Maximum Flow". In: *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*. IEEE Computer Society, 2014, pp. 424–433.

[Mąd13] Aleksander Mądry. "Navigating Central Path with Electrical Flows: From Flows to Matchings, and Back". In: *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*. IEEE Computer Society, 2013, pp. 253–262.

[MS16] Kurt Mehlhorn and Sanjeev Saxena. "A still simpler way of introducing interior-point method for linear programming". In: *Computer Science Review* 22 (2016), pp. 1–11.

[Mil+15] Gary L. Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. "Improved Parallel Algorithms for Spanners and Hopsets". In: *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13-15, 2015*. Ed. by Guy E. Blelloch and Kunal Agrawal. ACM, 2015, pp. 192–201.

[MVV87] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. "Matching is as easy as matrix inversion". In: *Combinatorica* 7.1 (1987), pp. 105–113.

[Orl88] James B. Orlin. "A Faster Strongly Polynominal Minimum Cost Flow Algorithm". In: *STOC*. Ed. by Janos Simon. ACM, 1988, pp. 377–387.

[Orl84] J.B. Orlin. *Genuinely Polynomial Simplex and Non-simplex Algorithms for the Minimum Cost Flow Problem*. Report OS / Centrum voor Wiskunde en Informatica. Stichting Mathematisch Centrum, 1984.

[Pel00] David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000.

[Pet15] Peter Kovács. "Minimum-cost Flow Algorithms: An Experimental Evaluation". In: *Optimization Methods Software* 30.1 (Jan. 2015), pp. 94–127.

[Rie98] Heiko Rieger. "Ground State Properties of Fluxlines in a Disordered Environment". In: *Phys. Rev. Lett.* 81 (20 Nov. 1998), pp. 4488–4491.

[Sar+12] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. "Distributed Verification and Hardness of Distributed Approximation". In: *SIAM J. Comput.* 41.5 (2012). Announced at STOC'11, pp. 1235–1265.

[Sch03] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and combinatorics. Springer, 2003.

[She13] Jonah Sherman. "Nearly Maximum Flows in Nearly Linear Time". In: *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*. IEEE Computer Society, 2013, pp. 263–269.

[She17] Jonah Sherman. "Generalized Preconditioning and Undirected Minimum-Cost Flow". In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*. Ed. by Philip N. Klein. SIAM, 2017, pp. 772–780.

[SS99] Hanmao Shi and Thomas H. Spencer. "Time-Work Tradeoffs of the Single-Source Shortest Paths Problem". In: *J. Algorithms* 30.1 (1999), pp. 19–32.

[SG82] Cornelis H Slump and Jan J Gerbrands. "A network flow approach to reconstruction of the left ventricle from two projections". In: *Computer Graphics and Image Processing* 18.1 (1982), pp. 18–36.

[Spe97] Thomas H. Spencer. "Time-work tradeoffs for parallel algorithms". In: *J. ACM* 44.5 (1997). Announced at SODA'91, pp. 742–778.

[ST04] Daniel A. Spielman and Shang-Hua Teng. "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems". In: *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*. Ed. by László Babai. ACM, 2004, pp. 81–90.

[Tam87] Roberto Tamassia. "On Embedding a Graph in the Grid with the Minimum Number of Bends". In: *SIAM J. Comput.* 16.3 (June 1987), pp. 421–444.

[Tar85] Éva Tardos. "A strongly polynomial minimum cost circulation algorithm". In: *Combinatorica* 5.3 (1985), pp. 247–256.

[Tho99] Mikkel Thorup. "Undirected Single-Source Shortest Paths with Positive Integer Weights in Linear Time". In: *J. ACM* 46.3 (1999), pp. 362–394.

[Tom71] N. Tomizawa. "On some techniques useful for solution of transportation network problems". In: *Networks* 1.2 (1971), pp. 173–194.

[Vai89] Pravin M. Vaidya. "Speeding-Up Linear Programming Using Fast Matrix Multiplication (Extended Abstract)". In: *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*. IEEE Computer Society, 1989, pp. 332–337.

[Wat89] Michael S. Waterman. *Mathematical Methods for DNA Sequences*. Boca Raton, FL, USA: CRC Press, Inc., 1989.

Part II

Roots and Zeros

Chapter 5

Introduction to Part II

Part II of this thesis deals with a very fundamental problem in computer algebra: the computation of the zeros of univariate polynomials and of systems of multivariate polynomials. These are very traditional problems [Pan97; Laz09]. The history of the univariate case goes back at least to the time of the Babylonians and Egyptians (approx. 2000 BC), who were already able to solve quadratic polynomial equations in one variable using what we call the "quadratic formula" today. However, a complete understanding of this formula was only attained with the distinction between real and imaginary roots which was only developed fully in the 19th century by Gauss and others. Many other classical results in mathematics can be considered milestones in the understanding of this formula. Such an example is the Pythagoreans' proof that the square root of 2 cannot be written as a fraction. In fact, the problem of solving polynomial equations has been a driving force behind the development of many classical concepts of today's mathematics, such as irrational numbers, imaginary numbers, algebraic groups, fields, and ideals [Pan97]. Indeed, the currently maybe most active research area in theoretical mathematics, namely algebraic geometry, deals with the zero sets of polynomial systems.

The *fundamental theorem of algebra* asserts that every univariate polynomial that is not a constant has at least one complex root. Clearly, using polynomial division inductively, the statement is equivalent to saying that a univariate polynomial of degree d has exactly d complex roots. The first proof of this fundamental result is usually attributed to Gauss. Together with the advent of Galois Theory, one might argue that the problem of roots of univariate polynomials is solved from a purely mathematical point of view. However, although the fundamental theorem of algebra already asserts the existence of the roots, an algorithmic approach to the problem was not given before the early 20th century by Brouwer [BL24] and Weyl [Wey24]. Many fruitful ideas have been contained within this early work. During the last decades a rich set of different approaches have proven to be successful. We point the reader to [Sma81] for a more detailed historic overview and to Chapter 6 for more details concerning the literature on univariate root finding that is most relevant to our work.

In this chapter, we will however not only be concerned with the question of determining the complex roots of a complex univariate polynomial, but we will also consider the multi-dimensional analogue of this question: Given n multi-variate polynomials, each in n variables, determine their common zeros. There is a rich literature on the task of computing solutions of such polynomial systems. Depending on whether the system is zero-dimensional or multi-dimensional, different methods apply. A historical overview is given by Lazard [Laz09]. In this thesis, we will consider the case of zero-dimensional polynomial systems, i.e., the case where the solution set is finite. We review related work on polynomial system solving in Chapter 7.

In addition to being interesting from a theoretical point of view, methods for the above problems are of great importance due to their broad applicability in various other scientific disciplines. These include robotics, coding theory, optimization, statistics, machine learning, and many others, see [Stu02] and references therein.

5.1 Main Results of Part II

For both of the above described problems, the problem of univariate root finding and the problem of solving zero-dimensional polynomial systems, the situation is somewhat unsatisfactory from the perspective of a (theoretical computer) scientist: The methods that are most frequently used in practice (e.g. MPsolve [BF00; BR14], eigensolve [For02], Bertini [Bat+13], or PHCpack [Ver99]) usually produce answers quite fast but, other than in special case, they are not guaranteed to be complete, correct, or even satisfy good worst-case run-time bounds. Methods, however, that satisfy such guarantees usually lack efficient implementations as they are either too involved or use machinery that is unlikely to be implemented efficiently. The ambition of the second part of this thesis is to provide methods to narrow the described gap for the above two problems.

Main Result 2.1. In Chapter 6, we describe a subdivision algorithm for isolating the complex roots of a polynomial $F \in \mathbb{C}[x]$ with complex coefficients. We work in the following very general model: We assume that the complex coefficients of F are provided to any absolute error bound by an oracle. For an arbitrary given input square $\mathcal{B}$ in the complex plane that contains only simple roots of F, our algorithm returns disjoint isolating discs for all roots of F in $\mathcal{B}$. We analyze the complexity of our approach: We show bounds on the absolute error to which the coefficients of F have to be approximated, the total number of iterations, and the overall bit complexity. Our analysis also shows that the complexity of the algorithm is controlled by the geometry of the roots in a near neighborhood of the input square $\mathcal{B}$, namely, by the number of roots, their absolute values and their pairwise distances. The number of subdivision steps of our method is near-optimal. For the special case, where the input polynomial has integer coefficients of bit-size less than τ and degree d, our algorithm needs $\tilde{O}(d^3+d^2\tau)$ bit operations, which is comparable to previous record bounds. It is, however, the first time that such a bound has been achieved using subdivision methods, and independent of divide-and-conquer techniques such as Schönhage's splitting circle technique, which are extremely difficult to implement in an efficient manner. Instead, our algorithm uses the quad-tree construction of Weyl [Wey24] with two key ingredients: We derive a "soft-test" to count the number of roots in a disc that is based on Pellet's Theorem and, in addition, uses Graeffe iteration. Schröder's modified Newton operator combined with bisection, in a form inspired by the quadratic interval method from Abbott [Abb14], allows us to achieve quadratic convergence towards root clusters.

Main Result 2.2. In Chapter 7, we turn to the problem of computing the solutions of polynomial systems: Assume that a numeric method is used in order to determine the solutions of a zero-dimensional polynomial system. Such methods are usually very fast and some of them are quite reliable. However, these methods usually do not come with any guarantees and thus we cannot rely on their result to be correct. This is where our new method comes into play. We propose a symbolic-numeric algorithm to count the number of solutions of a polynomial system within a local region exactly. More specifically, given a zero-dimensional system $f_1 = \cdots = f_n = 0$, with $f_i \in \mathbb{C}[x_1, \ldots, x_n]$, and a polydisc $\mathbf{\Delta} \subset \mathbb{C}^n$, our method aims to certify the existence of k solutions (counted with multiplicity) within this polydisc. In case of success, it yields the correct result under guarantee. Otherwise, no information is given. However, we show that our algorithm always succeeds if $\mathbf{\Delta}$ is sufficiently small and well-isolating for a k-fold solution $\mathbf{z}$ of the system. Our analysis of the algorithm further yields a bound on the size of the polydisc for which our algorithm succeeds under guarantee. This bound depends on local parameters such as the size and multiplicity of $\mathbf{z}$ as well as the distances between $\mathbf{z}$ and all other solutions. Efficiency of our method stems from the fact that we reduce the problem of counting the roots in $\boldsymbol{\Delta}$ of the original system to the problem of solving a truncated system of degree k. In particular, if the multiplicity k of $\mathbf{z}$ is small compared to the total degrees of the polynomials f_i, our method considerably improves upon known complete and certified methods. For the special case of a bivariate system, we report on an implementation of our algorithm, and show experimentally that our algorithm leads to a significant improvement, when integrated as inclusion predicate into an elimination method.

5.2 Notations

We continue by introducing some notation that will be used throughout this part of the thesis: When feasible, we will use bold font for multidimensional objects such as points $\mathbf{x} \in \mathbb{C}^n$.

- As in the previous part of the thesis, for any non-negative integer k, we denote by $[k]$ the set $\{1 \dots k\}$ of size k. For any set S and any non-negative integer k, we write $\binom{S}{k}$ for the set of all subsets of S of size k.
- The norm $\|\cdot\|$ will refer to the *infinity* norm $\|\cdot\|_\infty$ throughout this part of the thesis, i.e., for any $\mathbf{x} \in \mathbb{C}^n$, $\|\mathbf{x}\| := \|\mathbf{x}\|_\infty = \max_{i\in[n]} |x_i|$..
- For points $\mathbf{x}_1, \dots, \mathbf{x}_k \in \mathbb{C}^n$, we write $M(\mathbf{x}_1, \dots, \mathbf{x}_k) := \max(1, \|\mathbf{x}_1\|, \dots, \|\mathbf{x}_k\|)$
- As before $\log(\cdot) := \log_2(\cdot)$ is the binary logarithm. Moreover, for $\mathbf{x}_1, \dots, \mathbf{x}_k \in \mathbb{C}^n$, we define $\overline{\log}(\mathbf{x}_1, \dots, \mathbf{x}_k) := \lceil M(\log M(\mathbf{x}_1, \dots, \mathbf{x}_k))\rceil$.
- For $\alpha = (\alpha_1, \dots, \alpha_n)$ and $\mathbf{x} = (x_1, \dots, x_n)$, we write $\mathbf{x}^\alpha$ for $x_1^{\alpha_1} \cdots x_n^{\alpha_n}$.
- For $F(\mathbf{x}) = \sum_\alpha c_\alpha \mathbf{x}^\alpha \in \mathbb{C}[x_1, \dots, x_n] = \mathbb{C}[\mathbf{x}]$, we define $\|F\| = \max_\alpha\{|a_\alpha|\}$. We further define $\tau_F := \overline{\log}(\|F\|)$, which bounds the number of bits before the binary point in the binary representation of any coefficient of F.
- We denote the interior of a disc in the complex plane with center $m \in \mathbb{C}$ and radius $r \in \mathbb{R}_{>0}$ by $\Delta = \Delta_r(m) := \{z \in \mathbb{C} : |z - m| < r\}$. For short, we also write $\lambda \cdot \Delta$ to denote the disc $\Delta_{\lambda r}(m)$ that is centered at m and scaled by a factor $\lambda \in \mathbb{R}_{>0}$.
- For a univariate polynomial $F(x) = \sum_{i=0}^d a_i x^i$, we denote with $\mathrm{LC}(F) = a_d$ the leading coefficient of F.

As in the previous part, we will use the tilde-O notation $\tilde{O}(\cdot)$ in order to hide poly-logarithmic factors, i.e., $\tilde{O}(f) = O(f \log^c(f))$ for any constant c.

Chapter 6

Univariate Complex Root Isolation using Pellet's Test and Newton Iteration

6.1 Introduction

This chapter deals with a very fundamental and well-studied problem in computer algebra and numerical analysis: The problem of computing isolating regions for the zeros of a complex univariate polynomial. Although this can be seen as one of the most classical problems in computational algebra, it still constitutes a very active research field. This is undoubtedly due to the great practical importance of the problem, as there are many problems in different scientific disciplines that make critical use of univariate root solving. Such disciplines include engineering, computer science and mathematics in general, as well as natural sciences. Another reason for the steady research effort is that there is still a big discrepancy between methods that are used by practitioners and methods that are provably correct and achieve good theoretical run-time bounds. Methods that are most frequently used in practice are typically based on Newton's iteration, Aberth's method, Weierstrass-Durand-Kerner's method, the QR algorithm or similar [BF00; For02; BR14]. As usually, when methods are popular in practice, these methods are relatively simple, which makes it attractive to implement them. The other set of methods, namely the ones for which very good worst-case complexity bounds are known [EPT14; MSW15; Pan02], are based on a more complicated machinery, i.e., they use a series of asymptotically fast subroutines (see [Pan02, p. 702]), require a complicated precision management, and are usually presented in a non-self-contained manner, which makes it difficult for practitioners and even for researchers working in the same area to understand and implement them.

The goal of the work that this chapter is based on is to resolve the described discrepancy. We present a subdivision algorithm for complex root isolation called $\mathbb{C}$Isolate. For our method, we mainly combine simple and well-known techniques such as the classical quad-tree construction by Weyl [Wey24], Pellet's Theorem [RS02], Graeffe iteration[1] [Bes49; Hou59], and Schröder's modified Newton operator [Sch70]. The bounds on its theoretical worst-case complexity that we prove are similar to the best bounds currently known for this problem. In this context, it is remarkable that, for the complexity results, we do not require any asymptotically fast subroutines except the classical fast algorithms for polynomial multiplication and Taylor shift computation. We believe that, compared to existing asymptotically fast algorithms, our method is relatively simple and thus has the potential of being practical.

[1]We note that the method should probably not be attributed to Gräffe only [Hou59]. In fact, it was developed independently by Dandelin in 1826 and Gräffe in 1837. Also, Lobachevsky's contribution from 1834 is significant as it contains the key idea of the approach.

The previously best algorithm for complex root finding goes back to Schönhage's splitting circle method [Sch82b] and was later significantly refined by Pan [Pan02] and others [Kir98; NR96]. Pan gives an algorithm for a slightly different setting, namely for approximate polynomial factorization, where it is assumed that the coefficients of the input polynomial are complex numbers that can be accessed to an arbitrary precision. Then, for a polynomial F with roots $z_1, \dots, z_d$ contained in the unit disc and an integer $L \geq d \log d$, Pan's algorithm computes approximations $\tilde{z}_i$ of z_i with $\|F - \mathrm{LC}(F) \cdot \prod_{i \in [d]}(x - \tilde{z}_i)\|_1 < 2^{-L} \cdot \|F\|_1$ using only $\tilde{O}(d \log L)$ arithmetic operations with a precision of $O(L)$. In [MSW15], it is shown how to obtain an algorithm for computing isolating discs from the approximate factorization that is given by Pan's method. The cost for computing these isolating regions can be expressed in terms of the degree of F and the size of its coefficients, but also in terms of geometric parameters such as the location of the roots of F. In our work, we will also follow this approach of measuring the bit complexity of our algorithm in terms of the underlying geometry of the roots of the input polynomial. For special cases of the problem, namely, when the input polynomial has integer coefficients, of bit-size τ, we can translate these geometric parameters back and obtain bounds that solely depend on d, τ. The problem of isolating the complex roots of an integer polynomial is referred to as the *complex benchmark problem* by some authors. We remark that the difficulty of isolating the roots is more directly dependent on the location of the roots than on the degree d or bit-size τ and thus stating bounds in terms of the geometric parameter is actually more enlightening.

Using Pan's method [EPT14; MSW15], the complex benchmark problem can be solved with $\tilde{O}(d^2\tau)$ operations, which constitutes the current record bound for this problem. So far, this bound can only be achieved by running Pan's factorization algorithm with an L of size $\Omega(d(\tau + \log d))$, which means that $\tilde{\Theta}(d^2\tau)$ bit operations are needed for any input polynomial; see [EPT14, Theorem 3.1] for details. The adaptive algorithm from [MSW15] needs $\tilde{O}(d^3 + d^2\tau)$ bit operations, however its cost crucially depends on the aforementioned geometric parameters (e.g. the separations of its roots).

For the *real benchmark problem*, that is the isolation of the *real* roots of a polynomial F of degree d with integer coefficients of bit-size at most τ, the bound of $\tilde{O}(d^3 + d^2\tau)$ can be achieved by a subdivision algorithm that is based on Descartes' method and Newton Iteration [SM16]. Moreover, a recent implementation [KRS16] of this algorithm is competitive with the fastest existing implementations [RZ04] for real root isolation, and it shows superior performance for hard instances, where roots appear in clusters. Our method works very similar to the subdivision approach from [SM16] with the main difference being that our approach does not use Descartes' method and is thus not bound to the specialized setting of real roots. While solving the more general complex benchmark problem, our approach achieves the same bit-complexity of $\tilde{O}(d^3 + d^2\tau)$. It is interesting that this bound exactly matches the corresponding bound for the complex root isolation algorithm from [MSW15] that is based on Pan's method for approximate polynomial factorization.

Another big advantage of our method is that, in contrast to global approaches, it can be used for a local search for only the roots contained in some given region. In this case, the number of iterations as well as the cost of the algorithm adapt to geometric parameters that only depend on the roots located in some neighborhood of this region.

6.1.1 Overview and Main Results

Setting. Throughout this chapter, we assume that we are given an input polynomial

$$F(x) = \sum_{i=0}^{d} a_i x^i \in \mathbb{C}[x], \quad \text{with } d \geq 2 \text{ and } \frac{1}{4} < |a_d| \leq 1. \tag{6.1}$$

Notice that, the condition on a_d can always be achieved by a suitable scaling and that this scaling does not change the roots. As the coefficients a_i are given by arbitrary complex numbers possibly without finite representation, we have made an assumption on how the input is given. We assume that the coefficients are given to us via an oracle that, for an arbitrary integer precision L, provides dyadic approximations $\tilde{a}_i$ of a_i that coincide with a_i to L bits after the binary point. The polynomial $\tilde{F}(x) = \sum_{i=0}^{d} \tilde{a}_i x^i$ is then called an *L-bit approximation of F*. We assume that the cost for asking the oracle for such an L-bit approximation is equal to the cost of reading the input if $\tilde{F}$ was the exact input given to the algorithm. We remark that this is a very general setting, as even algebraic, or transcendental coefficients can be provided by such an oracle.

Throughout our considerations, we will call $z_1, \dots, z_d$ the distinct roots of F. We remark that as the coefficients are given approximately only, the problem of computing the roots of F is not well defined if we allow F to have multiple roots. This is due to the fact that any additional bit of a coefficient that is provided by the oracle could possibly change the situation from F having a k-fold root to F having a cluster of k roots. Besides this, we remark that our algorithm can still be used in the setting of multiple roots, however it is not guaranteed to terminate as given here and an additional stopping criteria needs to be included. We assume that, in addition to F, the input consists of a closed, axis-aligned square $\mathcal{B}$ in the complex plane. Given this input, our algorithm ℂIsolate computes isolating discs for all roots of F contained within $\mathcal{B}$. It may however, also return isolating discs for some roots of F that are contained within $2\mathcal{B} \setminus \mathcal{B}$.

Algorithm Overview. The approach underlying ℂIsolate is based on Weyl's quad tree construction as mentioned earlier, i.e., we recursively subdivide the input square $\mathcal{B}$ into smaller sub-squares. For a given sub-square, we check if it can be discarded, as it can be verified that this sub-square does not contain any root of F. We proceed as follows with the squares that cannot be excluded: We group them together into maximal connected components and test whether we can guarantee that they contain exactly one root. We do this by checking if a specific inclusion predicate holds on this maximal connected component.

The test that we use in order to implement these inclusion and exclusion predicates is based on Pellet's theorem and we use Graeffe iteration to accelerate it, we describe this in detail in Section 6.3. Here, we give a brief overview: Pellet's theorem [RS02] says that the number of roots contained within Δ equals k if the absolute value of the k-th coefficient of $F_\Delta(x) := F(m + rx)$ dominates the sum of the absolute values of all other coefficients. For $k = 0$ and $k = 1$, this has been used before in the context of the computation of roots [SY11; Yak00]. It is used there that Pellet's theorem applies if the smaller disc $d^{-c} \cdot \Delta$ contains k roots and the larger disc $d^{c'} \cdot \Delta$ contains no further root, where c and c' are suitable positive constants. In our work, we derive constants c and c' such that a corresponding result holds for any $k = 0, \dots, d$. Moreover, using the above mentioned Graeffe iterations, more precisely only $O(\log \log d)$ of them, the polynomial factors d^c and $d^{c'}$ can be replaced by constants $\rho_1 := 2\sqrt{2}/3 \approx 0.94$ and $\rho_2 := 4/3$. This is based on the observation that a Graeffe iteration on F_Δ leads to a squaring of the roots of F_Δ. In summary, we give a predicate that if it succeeds allows us to count the number of roots contained in a disc Δ, provided that $\rho_2 \cdot \Delta$ and $\rho_1 \cdot \Delta$ contain the same number of roots. As the coefficients of F are only given approximately and the described test requires exact arithmetic, we also derive a variant of this approach that uses approximate arithmetic only. This is based on the idea of so-called soft-predicates, see [YSS13]. We denote this test by $\mathbf{T}_*(\Delta, F)$. We analyze the precision demand of $\mathbf{T}_*(\Delta, F)$ and show that is directly related to the maximal absolute value that F takes on Δ.

We obtain the exclusion predicate for a square by inscribing the square within a disc Δ and testing $\mathbf{T}_*(\Delta, F)$. If the test yields 0, this square can be excluded. As said above, we group the squares that cannot be excluded together into maximal connected components. After inscribing these components into a corresponding disc Δ as well, we can apply the $\mathbf{T}_*(\Delta, F)$-test and check if it yields 1. If so, we store the disc Δ as an isolating disc. If none of the above succeeds, we further subdivide each square into four equally sized sub-squares, group them into maximal connected components, and proceed.

This basic approach leads a classical subdivision algorithm that induces (as any subdivision algorithm) a so-called subdivision tree. However, it does only lead linear convergence towards the roots of F, as, intuitively speaking, in every subdivision step one additional bit of the root gets determined. This can lead to long paths in the subdivision tree without branching. One class of instances for which this problem classically occurs are the so-called Mignotte polynomials. These polynomials have two very nearby roots and the length of such a long path in the subdivision tree can be lower bounded by $\Omega(d\tau)$. This is where the second main ingredient of our algorithm comes into play: the Newton iteration. This approach allows us to shorten the length of such paths to $O(\log(d\tau))$ and goes back to what is known as quadratic interval refinement (QIR) [Abb14]. Here the secant method was used, while Newton iteration was introduced to a similar approach in [Sag12; Sag14a; SM16]. Our approach follows these works. We combine basic bisection with Newton-iteration in the following way. We achieve a situation that leads success of Newton steps on regions where it is already known that they are isolating for a root or a cluster of roots. In order to check for such situations, we again use the previously described $\mathbf{T}_*$-test. It exactly allows us to detect such situations.

Main Result. We get the following first main theoretical result, which shows that the above described technique actually only produces a near-optimal number of squares:

Theorem 6.1. *Let F be a polynomial as in* (6.1) *and suppose that F is square-free. For isolating all complex roots of F, the algorithm* $\mathbb{C}$Isolate *produces a number of squares bounded by*

$$\tilde{O}\left(d \cdot \log(d) \cdot \log\left(d \cdot \Gamma_F \cdot \overline{\log}(\sigma_F^{-1})\right)\right),$$

where $\overline{\log}(x) := \max(1, \log|x|)$ *for arbitrary $x \in \mathbb{C}$,* $\Gamma_F := \overline{\log}(\max_{i=1}^d |z_i|)$ *is the* logarithmic root bound, *and* $\sigma_F := \min_{(i,j):i\neq j} |z_i - z_j|$ *is the* separation of F.

For the complex benchmark problem, the bound in the above theorem can be written as $O(d\log(d)\log(d\tau))$, i.e., the bound is nearly-linear in d. Note that already the output consists of d squares. When we use the algorithm on a restricted area, i.e., if the input square $\mathcal{B}$ does not cover all roots, we can achieve bounds that are adaptive w.r.t. the number of roots contained in some neighborhood of $\mathcal{B}$ as well as with respect to their geometric location. More precisely, assuming that there are only simple roots within $2\mathcal{B}$, then we may replace d, Γ_F, and σ_F in the above bound by the number of roots contained in $2\mathcal{B}$, the logarithm of the width of $\mathcal{B}$, and the minimal separation of the roots of F in $2\mathcal{B}$, respectively. For a precise result, see Theorem 6.26. Our second main result concerns the precision demand and bit complexity of the algorithm:

Theorem 6.2. *Let F be a polynomial as in (6.1) and suppose that F is square-free. For isolating all complex roots of F, the algorithm* $\mathbb{C}$Isolate *uses a number of bit operations bounded by*

$$\tilde{O}\left(\sum_{i=1}^d d \cdot (\tau_F + d \cdot \overline{\log}(z_i) + \overline{\log}(\sigma_F(z_i)^{-1}) + \overline{\log}(F'(z_i)^{-1}))\right)$$
$$= \tilde{O}(d(d^2 + d\,\overline{\log}(\mathrm{Mea}_F) + \overline{\log}(\mathrm{Disc}_F^{-1}))),$$

where $\tau_F := \lceil \widetilde{\log} \|F\| \rceil$, $\sigma_F(z_i) := \min_{j \neq i} |z_i - z_j|$ *is the* separation of z_i, $\mathrm{Mea}_F := |a_d| \cdot \prod_{i=1}^{d} \max(1, |z_i|)$ *the* Mahler Measure, *and* Disc_F *the* discriminant of F. *As input, the algorithm requires an L-bit approximation of F with*

$$L = \tilde{O}\left(\sum_{i=1}^{d} (\tau_F + d \cdot \overline{\log}(z_i) + \overline{\log}(\sigma_F(z_i)^{-1}) + \overline{\log}(F'(z_i)^{-1}))\right)$$
$$= \tilde{O}(d^2 + d\,\overline{\log}(\mathrm{Mea}_F) + \overline{\log}(\mathrm{Disc}_F^{-1})).$$

For the complex benchmark problem the above bound on the bit complexity can be simplified to $\tilde{O}(d^3 + d^2\tau)$. As above, if the algorithm is run on a restricted region $\mathcal{B}$, we can also show that the bit complexity and precision demand only depend on local parameters for some surrounding of $\mathcal{B}$.

6.1.2 Related Work

The literature regarding the computation of the real or complex roots of a univariate polynomial is very rich. In this section, we aim to give a brief overview of the work that is most related to our approach. We refer the reader to survey articles [MP12; McN02; McN07; MP13; Pan97] for a more complete overview.

Real root computation. Due to their simplicity so-called subdivision algorithms are by now the most popular methods (in theory and practice) in order to compute the real roots of a polynomial. Most computer algebra systems have incorporated a root computation method that is based on a subdivision approach. Many different approaches are known and they all roughly follow the same approach of iteratively subdividing some given initial interval I_0 into sub-intervals while applying inclusion and exclusion predicates to the sub-intervals. One can roughly group the methods into the following categories, depending on their choice of the inclusion/exclusion predicate and the way how they subdivide intervals. (1) Methods that are based on Descartes' rule of sign [CA76; Eig08; Eig+05; RZ04; Sag12; SM16; Sag14b; SB15]. Descartes' rule of signs gives information for the maximum number of roots of a polynomial as well as for the parity of the number of roots based on the number of sign changes in the coefficient sequence of the polynomial. (2) Methods that are based on the Bolzano method [Bec12; BK12; SY11]. These methods are based on a special case of Pellet's theorem, namely, they incorporate a test that can exclude the existence of a root in a given interval (as in our $\mathbf{T}_*$-test with $k = 0$). This test is then applied to the function as well as its derivative. If the test applies to the function itself, it provides an exclusion predicate. If it applies to the derivative, this shows that the function is monotone and thus if in addition a sign change of the function on the interval can be verified, this approach provides an inclusion predicate. (3) Techniques based on Sturm's method [Dav85; DSY07], and the continued fraction method [AS05; Sha08; Tsi13; TE08] are less related to our approach, thus we only mention them briefly.

For the above mentioned real benchmark problem, i.e., the problem of isolating all real roots of a polynomial of degree d with integer coefficients of bit-size at most τ, most of the above mentioned subdivision methods need $\tilde{O}(d\tau)$ subdivision steps and their bit complexity is $\tilde{O}(d^4\tau^2)$. This bound on the number of subdivision steps is due to the fact that the separation of all roots can be lower bounded by $2^{-\tilde{O}(\tau)}$ and that these subdivision methods only achieve linear convergence towards the roots. Moreover, for certain classes of polynomials with very small separation, e.g. Mignotte polynomials that have separation $2^{-\Omega(d\tau)}$, this bound is tight up to poly-logarithmic factors [Col16; Eig+05].

The above methods that simply use exact arithmetic, in each subdivision step incur $\tilde{O}(d^3\tau)$ bit operations, which is due to the fact that d arithmetic operations with a precision of $\tilde{O}(d^2\tau)$

are needed. The methods from [Sag14b; SM16] however use approximate arithmetic. More precisely, the authors show that, for the Descartes' method, it suffices to work with a precision of size $\tilde{O}(d\tau)$ in order to isolate all real roots, this yields a worst-case bit complexity of $\tilde{O}(d^3\tau^2)$. Interestingly, this has already been empirically observed in [RZ04] before. A similar result holds for the Bolzano method, as in [Bec12] it was shown that a modification of the Bolzano method that uses approximate arithmetic leads the same bit-complexity bound. Recent work [Sag12; SM16; SB15] further improves upon this by combining Descartes' method and Newton iteration. Newton iteration is the key to jump from linear to quadratic convergence in almost all iterations. This approach leads to methods that only need $O(d\log(d\tau))$ subdivision steps (this is near optimal). The methods from [Sag12; SB15] work for integer polynomials only and each computation is carried out with exact arithmetic. An amortized analysis of their cost yields the bound $\tilde{O}(d^3\tau)$ for the bit complexity. [SM16] introduces an algorithm that improves upon the methods from [Sag12; SB15] in two points. It can be used to compute the real roots of a polynomial with arbitrary real coefficients and, due to the use of approximate arithmetic, its precision demand is considerably smaller. This line of work leads the bit complexity bound of $\tilde{O}(d^3 + d^2\tau)$. This bound is achieved as follows. The method needs $\tilde{O}(d\log(d\tau))$ iterations. In each iteration, $\tilde{O}(d)$ arithmetic operations are carried out with an average precision of size $\tilde{O}(d + \tau)$.

We remark that this bound essentially matches the bounds achieved by our algorithm $\mathbb{C}$Isolate for *complex* root isolation. Our approach is similar to the one from [SM16] that has already found its way into efficient implementations, see [KRS16]. However, as we can handle complex input polynomials and output isolating regions for all complex roots, several new ideas and techniques were needed. Most prominently, we replace Descartes' method with the $\mathbf{T}_*$-test for counting the number of complex roots in a disc.

Complex root computation. For computing the complex roots, there also exist a series of subdivision methods (e.g. [CK92; MT09; MVY02; Pan00; Pin76; Ren87; SY11; Wil78; Yak05]); however, only a few algorithms have been analyzed in a way that allows a direct comparison with our method as they are not formulated for the exact same problem, namely root isolation, but for root computation in different settings.

The earliest contribution relevant to ours, namely the work by Weyl [Wey24], also treats a slightly different problem. His algorithm computes relative 2^{-b}-approximations to all roots of a given polynomial for a given precision b with $O(d^3 b\log d)$ arithmetic operations without using asymptotically fast polynomial arithmetic. Also, Weyl's algorithm, as the early real methods discussed above suffers from the problem that it only achieves linear convergence towards the roots. Renegar [Ren87] and later Pan [Pan00] achieved a faster convergence by combining subdivision methods with Newton iteration. While Renegar's algorithm achieves the bound of $O(d^2\log b + d^3\log d)$ arithmetic operations for approximating the roots within a relative 2^{-b} factor (again without using asymptotically fast polynomial arithmetic), Pan's method achieves the bound of $O((d^2\log d)\log(bd))$. From a highlevel point of view, the method by Pan and ours are quiet similar. Both use subdivision combined with Newton's iteration and even both use Graeffe iteration. The main algorithmic difference is maybe the use of Pellet's theorem which we use both for the exclusion predicate and the detection of clusters. An even more important difference between our result and Pan's result from [Pan00] is that we bound the bit complexity and Pan only derives a bound on the number of arithmetic operations.

Also Yakoubsohn's work [Yak05] has some similarities to our approach. Like us, he uses Pellet's theorem in a subdivision method based on Weyl's quad tree approach. However, again a slighly different problem is considered by Yakoubsohn, namely his method only allows to approximate the roots and does not apply for isolating them, as, in contrary to us, he does not use Pellet's theorem with $k \geq 1$ which allows us to conclude that a certain region is isolating for a root/set of roots. A similar approach was taken by Sagraloff and Yap [SY11], who have introduced the algorithm Ceval, which uses Pellet's theorem for $k = 0$ and $k = 1$ leading an exclusion and inclusion predicate, and resulting in a very simple subdivision method. As both these methods [Yak05; SY11] are based purely on subdivision, they only lead linear convergence again. The algorithm from [SY11] needs $\tilde{O}(d^2\tau)$ subdivision steps for the complex benchmark problem leading a bit complexity of $\tilde{O}(d^4\tau^2)$.

We note that there are also extensions of Pellet's theorem for the more general setting of analytic functions, see for example [Giu+05] for such an extension. There, the authors also show how to detect clusters of roots which can be used in order to achieve a setting where Schröder's modified Newton operator yields quadratic convergence to the cluster. The main difference to our method is that we use a trial-and-error approach, i.e., we always perform Schröder's modified Newton operator and check later (again using the test based on Pellet's theorem) whether the step succeeded. Another work that is similar to our approach and can be used in the more general setting of analytic functions is the work from [YSS13]. Here, an algorithm for computing ε-clusters of roots of analytic functions is given using a similar test based on Pellet's theorem. This algorithm however, does not have quadratic convergence (as no Newton iteration is used) nor is the complexity of the algorithm analyzed.

Structure of the Remainder of this Chapter. In Section 6.2, we introduce the most important definitions and further notations. We introduce the test $\mathbf{T}_*$ for counting the roots in a disc in Section 6.3. The algorithm $\mathbb{C}$Isolate is given in Section 6.5. Its analysis is split into two parts. In Section 6.6.1, we derive bounds on the number of iterations needed by our algorithms, whereas, in Section 6.6.2, we estimate its bit complexity. Some of the (rather technical) proofs are outsourced to an appendix, and we recommend to skip these proofs in a first reading of this chapter. In Section 6.7, we summarize and hint to some future research.

6.2 Definitions and a Root Bound

Let F be a polynomial as defined in (6.1) with complex roots $z_1, \ldots, z_d$. We fix the following definitions and denotations:

- As mentioned before, we assume that there is an oracle that, for an arbitrary non-negative integer L, provides dyadic approximations $\tilde{a}_i = \frac{m_i}{2^{L+1}}$ of the coefficients a_i such that $m_i \in \mathbb{Z} + i \cdot \mathbb{Z}$ are Gaussian integers and $|a_k - \tilde{a}_k| < 2^{-L}$ for all $k = 0, \ldots, d$. We say that $\tilde{a}_k$ approximates a_k to L bits after the binary point, and a corresponding polynomial $\tilde{F} = \sum_{i=0}^{d} \tilde{a}_i \cdot x^i$ with coefficients fulfilling the latter properties is called an *(absolute) L-bit approximation of F*. It is assumed that the cost for asking the oracle for such an approximation is the cost for reading the approximations.

- For a disc $\Delta_r(m)$, we denote with $F_\Delta(x)$ the shifted and scaled polynomial $F(m + r \cdot x)$.

- We let $\Gamma_F := \overline{\log}(|z_1|, \ldots, |z_d|)$ be the *logarithmic root bound of F*. For a root z_i of F, we let $\mu(z_i, F)$ denote its multiplicity. We define $\sigma_F(z_i) := \min_{j \neq i} |z_i - z_j|$ to be the *separation of the root z_i* and $\sigma_F := \min_{i=1}^{d} \sigma_F(z_i)$ the *separation of F*. For an arbitrary region $\mathcal{R} \subset \mathbb{C}$ in the complex space, we define $\sigma_F(\mathcal{R}) := \min_{i:z_i \in \mathcal{R}} \sigma_F(z_i)$, which we call the *separation of F restricted to $\mathcal{R}$*. We further denote by $\mathcal{Z}(\mathcal{R})$ the set of all roots of F that are contained in $\mathcal{R}$, and by $\operatorname{Mea}_F(\mathcal{R}) := |a_d| \cdot \prod_{z_i \in \mathcal{Z}(\mathcal{R})} M(z_i)$ the *Mahler measure of F restricted to $\mathcal{R}$*.

- A disc Δ *is isolating for a root z_i* of F if it contains z_i but no other root of F. For a set S of roots of F and positive real values ρ_1 and ρ_2 with $\rho_1 \leq 1 \leq \rho_2$, we say that a disc Δ *is (ρ_1, ρ_2)-isolating for S* if $\rho_1 \cdot \Delta$ contains exactly the roots contained in S and $\rho_2 \cdot \Delta \setminus \rho_1 \cdot \Delta$ contains no root of F.

- We will only consider squares

$$B = \{z = x + i \cdot y \in \mathbb{C} : x \in [x_{\min}, x_{\max}] \text{ and } y \in [y_{\min}, y_{\max}]\}$$

  in the complex space that are *closed, axis-aligned, and of width* $w(B) = 2^\ell$ *for some* $\ell \in \mathbb{Z}$ (i.e., $|x_{\max} - x_{\min}| = |y_{\max} - y_{\min}| = 2^\ell$), hence, for brevity, these properties are not peculiarly mentioned. Similar as for discs, for a $\lambda \in \mathbb{R}^+$, $\lambda \cdot B$ denotes the scaled square of size $\lambda \cdot 2^\ell$ centered at B.

According to Cauchy's root bound, see e.g. [Yap00], we have $|z_i| \le 1 + \max_{i=0}^{d} \frac{|a_i|}{|a_d|} < 1 + 4 \cdot 2^{\tau_F}$, and thus $\Gamma_F = O(\tau_F)$. In addition, it holds that

$$\tau_F \le \overline{\log}(2^d \cdot \mathrm{Mea}_F) \le d(1 + \Gamma_F) \le 2d\Gamma_F,$$

where we use Landau's inequality, that is, $\mathrm{Mea}_F \le \|F\|_2 \le \sqrt{d+1} \cdot \|F\|$, see e.g. [Yap00]. Following [MSW15, Theorem 1], we can compute an integer approximation $\tilde{\Gamma}_F \in \mathbb{N}$ of Γ_F with $\Gamma_F + 1 \le \tilde{\Gamma}_F \le \Gamma_F + 8 \log d + 1$ using $\tilde{O}(d^2\Gamma_F)$ many bit operations. For this, the coefficients of F need to be approximated to $\tilde{O}(d\Gamma_F)$ bits after the binary point. From $\tilde{\Gamma}_F$, we then immediately derive an integer $\Gamma = 2^\gamma$, with $\gamma := \lceil \log \tilde{\Gamma}_F \rceil \in \mathbb{N}_{\ge 1}$, such that

$$\Gamma_F + 1 \le \tilde{\Gamma}_F \le \Gamma \le 2 \cdot \tilde{\Gamma}_F \le 2 \cdot (\Gamma_F + 8 \log d + 1). \tag{6.2}$$

It follows that $2^\Gamma = 2^{O(\Gamma_F + \log d)}$ is an upper bound for the modulus of all roots of F, and hence, when interested in isolating all complex roots of F, we can always restrict ourselves to the set of all complex numbers of absolute value at most 2^Γ.

6.3 Counting Roots in a Disc

In this section, we introduce the $\mathbf{T}_*(\Delta)$-test, which constitutes our main ingredient to count the number of roots of F in a given disc Δ. Here, we briefly summarize the main properties of the $\mathbf{T}_*(\Delta)$-test. The reader willing to focus on the algorithmic details of the root isolation algorithm is invited to read the following summary and skip the remainder of this section on a first read.

- For a given polynomial F as in (6.1) and a disc Δ, the $\mathbf{T}_*(\Delta)$-test always returns an integer $k \in \{-1, 0, 1, \dots, d\}$. If $k \ge 0$, then Δ contains exactly k roots of F. If $k = -1$, no further information on the number of roots in Δ can be derived; see Lemma 6.15, part (b).

- If Δ is (ρ_1, ρ_2)-isolating for a set of k roots of F, where $\rho_1 = \frac{2\sqrt{2}}{3} \approx 0.94$ and $\rho_2 = \frac{4}{3}$, then $\mathbf{T}_*(\Delta)$ returns k, see Lemma 6.15, part (a). In particular, $\mathbf{T}_*(\Delta)$ returns 0 if $\frac{4}{3} \cdot \Delta$ (and $\frac{2\sqrt{2}}{3} \cdot \Delta$) contains no root.

- The cost for the $\mathbf{T}_*(\Delta)$-test is bounded by

$$\tilde{O}(d(\tau_F + d\,\overline{\log}(m, r) + \overline{\log}(\|F_\Delta\|^{-1}))) = \tilde{O}(d(\tau_F + d\,\overline{\log}(m, r) + \overline{\log}((\max_{z \in \Delta} |F(z)|)^{-1})))$$

  bit operations, and thus directly related to the size of Δ and the maximum absolute value that F takes on Δ. For this, the test requires an L-bit approximation of F, with

$$L = \tilde{O}(\tau_F + d\,\overline{\log}(m, r) + \overline{\log}(\|F_\Delta\|^{-1})) = \tilde{O}(\tau_F + d\,\overline{\log}(m, r) + \overline{\log}((\max_{z \in \Delta} |F(z)|)^{-1})),$$

see Lemma 6.16. Here, we used that $\max_{z\in\Delta}|F(z)| \le (d+1)\cdot\|F_\Delta\|$ as shown in (6.7) in the proof of Theorem 6.10.

6.4 Pellet's Theorem and the $\mathcal{T}_k$-Test

We introduce the so-called $\mathcal{T}_k$-test based on Pellet's theorem. Both this chapter and the following chapter heavily rely on this test – it will allow us to count the number of roots of a univariate polynomial in a given disc. Let $F \in \mathbb{C}[x]$ be a univariate polynomial of degree $d \ge 2$, let k be an integer with $0 \le k \le d = \deg F$, and let $K \in \mathbb{R}$ with $K \ge 1$.

Definition 6.3 (The $\mathcal{T}_k$-Test)**.** *For a polynomial $F \in \mathbb{C}[x]$, the $\mathcal{T}_k$-test on a disc $\Delta := \Delta_r(m)$ with parameter K holds if*

$$\mathcal{T}_k(m,r,K,F): \quad \left|\frac{F^{(k)}(m)r^k}{k!}\right| > K\cdot\sum_{i\neq k}\left|\frac{F^{(i)}(m)r^i}{i!}\right| \tag{6.3}$$

or, equivalently, if $F^{(k)}(m)\neq 0$ and

$$\mathcal{T}_k(m,r,K,F): \quad \sum_{i<k}\left|\frac{F^{(i)}(m)r^{i-k}k!}{F^{(k)}(m)i!}\right| + \sum_{i>k}\left|\frac{F^{(i)}(m)r^{i-k}k!}{F^{(k)}(m)i!}\right| < \frac{1}{K}. \tag{6.4}$$

Mostly, we will write $\mathcal{T}_k(\Delta,K,F)$ for $\mathcal{T}_k(m,r,K,F)$, or simply $\mathcal{T}_k(\Delta,K)$ or $\mathcal{T}_k(\Delta,F)$ if the omitted arguments are clear from the context. Notice that if the $\mathcal{T}_k$-test succeeds for some parameter $K = K_0$, then it also succeeds for any K with $K \le K_0$. Clearly, $\mathcal{T}_k(m,r,K,F)$ is equivalent to $\mathcal{T}_k(0,1,K,F_\Delta)$, with $F_\Delta(x) = F(m + r\cdot x)$.

The following result is a direct consequence of Rouché's theorem, and, in our algorithm, it will be crucial in order to compute the size of a cluster of roots of F; see [RS02, Section 9.2] for a proof.

Theorem 6.4. *If $\mathcal{T}_k(m,r,K,F)$ holds for some $K\in\mathbb{R}$ with $K\ge 1$ and some $k\in\{0,\dots,d\}$, then $\Delta_r(m)$ contains exactly k roots of F counted with multiplicities.*

Proof. The proof uses Rouché's theorem, see Theorem 7.17 in Chapter 7 for a very general (multidimensional) version. For two univariate polynomials ϕ and γ, Rouché's theorem yields that ϕ and γ have the same number of zeros within a disc $\Delta_r(m)$, if $|\phi(x)-\gamma(x)| \le |\gamma(x)|$ for all $x \in \partial\Delta_r(m)$. We apply the theorem to $\phi(x) = F(x)$ and $\gamma(x) = F^{(k)}(m)\cdot(m-x)^k/k!$. Note that if $\mathcal{T}_k(m,r,K,F)$ holds, it follows that $F^{(k)}(m)\neq 0$, thus $\gamma(x)$ has exactly k zeros at m. Using Taylor expansion of F around m yields that

$$\begin{aligned}|\phi(x)-\gamma(x)| &= \left|F(x) - \frac{F^{(k)}(m)}{k!}(m-x)^k\right| = \left|\sum_{i\neq k}\frac{F^{(i)}(m)}{i!}(m-x)^i\right| \le \sum_{i\neq k}\left|\frac{F^{(i)}(m)}{i!}r^i\right| \\ &\le \frac{1}{K}\left|\frac{F^{(k)}(m)}{k!}r^k\right| \le |\gamma(x)|\end{aligned}$$

for any $x\in\partial\Delta_r(m)$. We conclude that $\phi(x) = F(x)$ has the same number of roots within $\Delta_r(m)$ as γ, namely k. □

We derive criteria on the locations of the roots $z_1,\dots,z_d$ of F under which the $\mathcal{T}_k$-test is guaranteed to succeed:

Theorem 6.5. *Let k be an integer with $0 \leq k \leq d = \deg(F)$, let $K \in \mathbb{R}$ with $K \geq 1$, and let c_1 and c_2 be arbitrary real values fulfilling*

$$c_2 \cdot d \cdot \ln\left(\frac{1+2K}{2K}\right) \geq c_1 \cdot d \geq \frac{M(k)}{\ln(1+\frac{1}{8K})}. \tag{6.5}$$

For a disc $\Delta = \Delta_r(m)$, suppose that there exists a real λ with

$$\lambda \geq \max(4c_2 \cdot M(k) \cdot d^3, 16K \cdot M(k)^2 \cdot d)$$

such that Δ is $(1, \lambda)$-isolating for the roots $z_1, \ldots, z_k$ of F, then $\mathcal{T}_k(c_1 d \cdot \Delta, K, F)$ holds.

The (rather technical) proof of the above theorem is split into two separate lemmas that we prove independent of each other.

Lemma 6.6. *Let $\Delta := \Delta_r(m)$ be a disc that is $(1, 4c_2 \cdot M(k) \cdot d^3)$-isolating for the roots $z_1, \ldots, z_k$, then, for all $z \in c_2 d^2 \cdot \Delta$, it holds that $F^{(k)}(z) \neq 0$. Furthermore, $\sum_{i=k+1}^{d} \left|\frac{F^{(i)}(m)(c_1 d \cdot r)^{i-k} k!}{F^{(k)}(m) i!}\right| < \frac{1}{2K}$.*

Proof. 1. For the first part, we may assume that $k \geq 1$. Then, for the k'th derivative of F and any complex z that is not a root of F, it holds that

$$\frac{F^{(k)}(z)}{F(z)} = \sum_{J \in \binom{[d]}{k}} \prod_{j \in J} \frac{1}{z - z_j} = \prod_{i \in [k]} \frac{1}{z - z_i} + \sum_{J \in \binom{[d]}{k}, J \neq [k]} \prod_{j \in J} \frac{1}{z - z_j}.$$

Assume for contradiction that $F^{(k)}(z) = 0$ for some $z \in c_2 d^2 \cdot \Delta$. Then from the above inequality, we obtain $\prod_{i \in [k]} \frac{1}{|z - z_i|} \leq \sum_{J \in \binom{[d]}{k}, J \neq [k]} \prod_{j \in J} \frac{1}{|z - z_j|}$. We distinguish cases. First, let $k \leq d/2$. Then, we conclude

$$\begin{aligned} 1 &\leq \sum_{J \in \binom{[d]}{k}, J \neq [k]} \frac{\prod_{i \in [k]} |z - z_i|}{\prod_{j \in J} |z - z_j|} = \sum_{k'=0}^{k-1} \sum_{J \in \binom{[k]}{k'}} \sum_{J' \in \binom{[d] \setminus [k]}{k-k'}} \frac{\prod_{i \in [k]} |z - z_i|}{\prod_{i \in J} |z - z_i| \cdot \prod_{j \in J'} |z - z_j|} \\ &\leq \sum_{k'=0}^{k-1} \binom{k}{k'} \binom{d-k}{k-k'} \left(\frac{2c_2 d^2 r}{4c_2 k d^3 r - c_2 d^2 r}\right)^{k-k'} \leq \sum_{k'=0}^{k-1} \binom{k}{k'} \binom{d-k}{k-k'} \left(\frac{1}{2kd}\right)^{k-k'}. \end{aligned}$$

We can now apply crude bounds to the binomial coefficients in the above sum in order to obtain

$$1 \leq \sum_{k'=0}^{k-1} \frac{k^{k-k'}}{(k-k')!} (d-k)^{k-k'} \left(\frac{1}{2kd}\right)^{k-k'} \leq \sum_{k'=0}^{k-1} \frac{(1/2)^{k-k'}}{(k-k')!} < e^{1/2} - 1 < 1,$$

which is a contradiction. Now suppose $k > d/2$. We then have

$$\begin{aligned} 1 &\leq \sum_{J \in \binom{[d]}{k}, J \neq [k]} \frac{\prod_{i \in [k]} |z - z_i|}{\prod_{j \in J} |z - z_j|} = \sum_{k'=1}^{n-k} \sum_{J \in \binom{[k]}{k-k'}} \sum_{J' \in \binom{[d] \setminus [k]}{k'}} \frac{\prod_{i \in [k]} |z - z_i|}{\prod_{i \in J} |z - z_i| \cdot \prod_{j \in J'} |z - z_j|} \\ &\leq \sum_{k'=1}^{d-k} \binom{k}{k-k'} \binom{d-k}{k'} \left(\frac{2c_2 d^2 r}{4c_2 k d^3 r - c_2 d^2 r}\right)^{k'} < \sum_{k'=1}^{d-k} \binom{k}{k-k'} \binom{d-k}{k'} \left(\frac{2}{3kd}\right)^{k'}. \end{aligned}$$

Bounding the binomial coefficients leads

$$1 \leq \sum_{k'=1}^{d-k} \frac{k^{k'}}{k'!}(d-k)^{k'}\left(\frac{2}{3kd}\right)^{k'} \leq \sum_{k'=1}^{d-k} \frac{1}{k'!}\left(\frac{2}{3}\right)^{k'} \leq e^{2/3}-1 < 1, \text{ again a contradiction.}$$

2. Similar as above, with $z_1^{(k)}, \ldots, z_{d-k}^{(k)}$ denoting the roots of $F^{(k)}$, it holds that

$$\left|\frac{F^{(k+i)}(m)}{F^{(k)}(m)}\right| \leq \sum_{J\in\binom{[d-k]}{i}} \prod_{j\in J} \frac{1}{|m-z_j^{(k)}|} \leq \frac{\binom{d-k}{i}}{(c_2 d^2 r)^i},$$

and thus

$$\sum_{i=k+1}^{d} \left|\frac{F^{(i)}(m)(c_1 dr)^{i-k}k!}{F^{(k)}(m)i!}\right| \leq \sum_{i=1}^{d-k}\left|\frac{F^{(k+i)}(m)}{F^{(k)}(m)}\right| \frac{(c_1 dr)^i}{i!} \leq \sum_{i=1}^{d-k} \frac{\binom{d-k}{i}}{c_2^i d^{2i} r^i}\frac{(c_1 dr)^i}{i!} < \sum_{i=1}^{d-k}\left(\frac{c_1}{c_2}\right)^i \frac{1}{i!}$$
$$\leq e^{c_1/c_2}-1 \leq \frac{1}{2K},$$

where we used that $\frac{k!}{(k+i)!} \leq \frac{1}{i!}$ and (6.5) for the last inequality. □

Lemma 6.7. *Le λ be a real value with $\lambda \geq 16K \cdot M(k)^2 \cdot d$ and suppose that $\Delta := \Delta_r(m)$ is a disc that is $(1,\lambda)$-isolating for the roots $z_1, \ldots, z_k$ of F, then $\sum_{i<k} \frac{|F^{(i)}(m)|}{|F^{(k)}(m)|}\frac{(c_1 d\cdot r)^{i-k}k!}{i!} < \frac{1}{2K}$.*

Proof. We may assume that $k \geq 1$. Write $F(x) = G(x)H(x)$ with $G(x) = \prod_{i\in[k]}(x-z_i)$ and $H(x) = \prod_{j\in[d]\setminus[k]}(x-z_j)$. By induction over k, it is straightforward to show that $F^{(k)}(x) = k!\sum_{I\in\binom{[d]}{k}}\prod_{i\notin I}(x-z_i) = k!\cdot\sum_{J\in\binom{[d]}{d-k}}\prod_{j\in J}(x-z_j)$. It follows that

$$|F^{(k)}(m)| = k!\cdot|H(m)|\Big|\sum_{J\in\binom{[d]}{d-k}} \frac{\prod_{j\in J}(m-z_j)}{\prod_{i=k+1}^d |m-z_i|}\Big| \geq k!\cdot|H(m)|\Big(1-\sum_{\substack{J\in\binom{[d]}{d-k}:\\ J\neq[d]\setminus[k]}} \frac{\prod_{j\in J}|m-z_j|}{\prod_{i=k+1}^d|m-z_i|}\Big)$$
$$\geq k!\cdot|H(m)|\Big(1-\sum_{j=1}^{\min(k,d-k)} \sum_{J_1,J_2:\, J_1\in\binom{[k]}{j}\wedge J_2\in\binom{[d]\setminus[k]}{d-k-j}} \frac{\prod_{j\in J_1}|m-z_j|}{(\lambda r)^j}\Big),$$

where we used that the j distances in the denominator that are left after canceling out the common factors from the fraction are all lower bounded by λr. Moreover, using that $J_1 \subset [k]$, we can upper bound the distances in the enumerator by r and obtain

$$|F^{(k)}(m)| \geq k!\cdot|H(m)|\Big(1-\sum_{j=1}^{\min(k,d-k)}\binom{k}{j}\binom{d-k}{d-k-j}\frac{1}{\lambda^j}\Big) \geq k!\cdot|H(m)|\Big(2-\sum_{j=0}^{d} k^j\binom{d}{j}\frac{1}{\lambda^j}\Big).$$

As $\sum_{j=0}^d k^j\binom{d}{j}\lambda^{-j} \leq (1+\frac{k}{\lambda})^d \leq e^{\frac{1}{4}} \leq 3/2$. We conclude that $|F^{(k)}(m)| \geq k!|H(m)|/2$.

As $G^{(i)}(x) = i!\sum_{J\in\binom{[k]}{i}}\prod_{j\notin J}(x-z_j)$, it holds that $|G^{(i)}(m)| \leq i!\binom{k}{i}r^{k-i}$. In addition, as $|\frac{H^{(i)}(m)}{H(m)}| \leq \sum_{J\in\binom{[d]}{i}}\prod_{j\in J}\frac{1}{|m-z_j|} \leq i!\cdot\binom{d-k}{i}\frac{1}{(\lambda r)^i}$, we obtain

$$|G^{(i-j)}(m)H^{(j)}(m)| \leq |H(m)|\cdot(i-j)!j!\binom{k}{i-j}\binom{d-k}{j}\cdot\frac{1}{\lambda^j}r^{k-i}. \tag{6.6}$$

Morevoer, by induction $F^{(i)}(x) = \sum_{j=0}^{i} \binom{i}{j} G^{(i-j)}(x) H^{(j)}(x)$, thus, using (6.6), it follows that

$$\begin{aligned}\sum_{i=0}^{k-1} \frac{|F^{(i)}(m)|}{|F^{(k)}(m)|} \frac{(c_1 d r)^{i-k} k!}{i!} &\leq \sum_{i=0}^{k-1} \sum_{j=0}^{i} \frac{|H(m)|}{|F^{(k)}(m)|} (i-j)! j! \binom{k}{i-j} \binom{d-k}{j} \binom{i}{j} \frac{(c_1 d)^{i-k}}{\lambda^j} \frac{k!}{i!} \\ &\leq 2(c_1 d)^{-k} \sum_{i=0}^{k-1} \binom{k}{i} (c_1 d)^i + 2 \sum_{i=1}^{k-1} \sum_{j=1}^{i} \binom{k}{i-j} \binom{d-k}{j} \frac{(c_1 d)^{i-k}}{\lambda^j},\end{aligned}$$

where we used that $|H(m)|/|F^{(k)}(m)| \leq 2/k!$. We can upper bound the second summand by

$$2 \sum_{i=1}^{k-1} \sum_{j=1}^{i} \binom{d-k}{j} \frac{k^j}{\lambda^j} \ln^{k-i} \left(1 + \frac{1}{8K}\right) \leq \frac{k-1}{8Kk} + \sum_{i=2}^{k-1} \sum_{j=2}^{i} \frac{2}{(16Kk)^j},$$

using that $\lambda \geq 16Kk^2 \cdot d$ and $c_1 d \geq k / \ln\left(1 + \frac{1}{8K}\right)$. In summary, it follows that

$$\sum_{i=0}^{k-1} \frac{|F^{(i)}(m)|}{|F^{(k)}(m)|} \frac{(c_1 d r)^{i-k} k!}{i!} < 2 \left(\sum_{j=0}^{k} \binom{k}{j} (c_1 d)^{-j} - 1 \right) + \frac{1}{4K} \leq 2\left(e^{k/(c_1 d)} - 1\right) + \frac{1}{4K} \leq \frac{1}{2K}. \qquad \square$$

In the two next chapters, we will make use of the following Corollary 6.8, which is a consequence of Theorem 6.5 with the specific values $K := \frac{3}{2}$, $c_1 := 16$, $c_2 := 64$, $\lambda = 256d^5$, and thus $M(k)/\ln(1 + \frac{1}{8K}) \approx 12.49 \cdot M(k)$ and $\ln\left(\frac{1+2K}{2K}\right) \approx 0.29$.

Corollary 6.8. *Let Δ be a disc in the complex space that is $(\frac{1}{16d}, 16d^4)$-isolating for a set of k roots (counted with multiplicity) of F. Then, $\mathcal{T}_k(\Delta, \frac{3}{2}, F)$ holds.*

6.4.1 The $\mathcal{T}_k^G$-Test: Using Graeffe Iteration

Corollary 6.8 guarantees success of the $\mathcal{T}_k(\Delta, 3/2, F)$-test, with $k = |\mathcal{Z}(\Delta)|$, if the disc Δ is $(\frac{1}{16d}, 16d^4)$-isolating for a set of k roots. In this section, we use a well-known approach for squaring the roots of a polynomial, called Graeffe iteration [Bes49], in order to improve upon the $\mathcal{T}_k$-test. More specifically, we present a variant of the $\mathcal{T}_k$-test, which we denote $\mathcal{T}_k^G$-test, that allows us to exactly count the roots contained in some disc Δ, if Δ is (ρ_1, ρ_2)-isolating for a set of k roots, with constants ρ_1 and ρ_2 of size $\rho_1 \approx 0.947$ and $\rho_2 = \frac{4}{3}$, i.e., the Graeffe iteration is used in order to reduce the fraction of ρ_1 and ρ_2 from polynomial in d to constant.

Definition 6.9 (Graeffe Iteration). *For a polynomial $F(x) = \sum_{i=0}^{d} a_i x^i \in \mathbb{C}[x]$, write $F(x) = F_e(x^2) + x \cdot F_o(x^2)$, with*

$$\begin{aligned} F_e(x) &:= a_{2\lfloor \frac{d}{2} \rfloor} x^{\lfloor \frac{d}{2} \rfloor} + a_{2\lfloor \frac{d}{2} \rfloor - 2} x^{\lfloor \frac{d}{2} \rfloor - 1} + \ldots + a_2 x + a_0, \quad \textit{and} \\ F_o(x) &:= a_{2\lfloor \frac{d-1}{2} \rfloor + 1} x^{\lfloor \frac{d-1}{2} \rfloor} + a_{2\lfloor \frac{d-1}{2} \rfloor - 1} x^{\lfloor \frac{d-1}{2} \rfloor - 1} + \ldots + a_3 x + a_1. \end{aligned}$$

Then, the first Graeffe iterate $F^{[1]}$ of F *is defined as:* $F^{[1]}(x) := (-1)^d [F_e(x)^2 - x \cdot F_o(x)^2]$.

The first part of the following theorem is well-known (e.g. see [Bes49]), and we give its proof only for the sake of a self-contained presentation. For the second part, we have not been able to find a corresponding result in the literature. Despite the fact that we consider the result to be of independent interest, we will need it in the analysis later on.

Theorem 6.10. *Denote the roots of F by $z_1, \ldots, z_d$, then it holds that $F^{[1]}(x) = \sum_{i=0}^{d} a_i^{[1]} x^i = a_d^2 \cdot \prod_{i=1}^{d} (x - z_i^2)$. In particular, the roots of the first Graeffe iterate $F^{[1]}$ are the squares of the roots of*

F. In addition, we have

$$d^2 \cdot M(\|F\|)^2 \geq \|F^{[1]}\| \geq \|F\|^2 \cdot 2^{-4d}.$$

Proof. Notice that $a_d^{[1]} = a_d^2$ follows directly from the definition of $F^{[1]}$. Furthermore, we have

$$\begin{aligned} F^{[1]}(z_i^2) &= (-1)^d \cdot [F_e(z_i^2)^2 - z_i^2 \cdot F_o(z_i^2)^2] \\ &= (-1)^d \cdot [F_e(z_i^2) - z_i \cdot F_o(z_i^2)] \cdot [F_e(z_i^2) + z_i \cdot F_o(z_i^2)] \\ &= (-1)^d \cdot [F_e(z_i^2) - z_i \cdot F_o(z_i^2)] \cdot F(z_i) = 0. \end{aligned}$$

Going from F to an arbitrary small perturbation $\tilde{F}$ (which has only simple roots and for which $\tilde{z}_i^2 \neq \tilde{z}_j^2$ for all pairs of distinct roots $\tilde{z}_i$ and $\tilde{z}_j$ of $\tilde{F}$), we conclude that each root z_i^2 of $F^{[1]}$ has multiplicity

$$\mu(z_i^2, F^{[1]}) = \sum_{j:z_j^2=z_i^2} \mu(z_j, F).$$

Hence, the first claim follows. For the second claim, notice that the left inequality follows immediately from the fact that each coefficient of $F^{[1]}$ is the sum of at most d^2 many products of the form $\pm a_i \cdot a_j$, and each of these products has absolute value smaller than or equal to $M(\|F\|)^2$. For the right inequality, let $k \in \{0, \dots, d\}$ be such that $|z_i| < 2$ for $i \in [k]$, and that $|z_i| \geq 2$ for $i \in [d] \setminus [k]$. Let $z_{\max}$ be a point in the closure of the unit disc $\Delta_1(0)$ such that $|F(z_{\max})| = \max_{z:|z|\leq 1} |F(z)|$. Since F takes its maximum on the boundary of $\Delta_1(0)$, we must have $|z_{\max}| = 1$, and using Cauchy's Integral Theorem to write the coefficients of F in terms of an integral, we conclude that $|F(z_{\max})| \geq \|F\|$. In addition, it holds that $|F(z_{\max})| \leq \sum_{i=0}^d |a_i| \cdot |z_{\max}|^d = \sum_{i=0}^d |a_i| \leq (d+1) \cdot \|F\|$, and thus

$$\|F\| \leq |F(z_{\max})| = \max_{z:|z|\leq 1} |F(z)| \leq (d+1) \cdot \|F\|. \tag{6.7}$$

Applying the latter result to the polynomial $g(x) := \prod_{i=1}^k (z - z_i^2)$ yields the existence of a point z' with $|z'| = 1$ and $|g(z')| \geq 1$. Hence, it follows that

$$\begin{aligned} |F^{[1]}(z')| &= |a_d|^2 \prod_{i\in[k]} |z' - z_i^2| \cdot \prod_{i\in[d]\setminus[k]} |z' - z_i^2| \geq |a_d|^2 \prod_{i\in[d]\setminus[k]} |(\sqrt{z'} - z_i) \cdot (\sqrt{z'} + z_i)| \\ &\geq |a_d|^2 \cdot \prod_{i\in[k]} \frac{|z_{\max} - z_i|^2}{9} \cdot \prod_{i\in[d]\setminus[k]} \frac{|z_{\max} - z_i|^2}{9} \geq \frac{|F(z_{\max})|^2}{9^d}, \end{aligned}$$

where we used that $|x - y| < 3$ for arbitrary complex points x, y with $|x| = 1$ and $|y| < 2$, and that $|x - z| \geq \frac{|y-z|}{3}$ for arbitrary complex points x, y, z with $|x| = |y| = 1$ and $|z| \geq 2$. We conclude that

$$\|F^{[1]}\| \geq \frac{|F^{[1]}(z')|}{d+1} \geq \frac{|F(z_{\max})|^2}{(d+1) \cdot 9^d} \geq \|F\|^2 \cdot 2^{-4d}. \qquad \square$$

We can now iteratively apply Graeffe iterations in order to square the roots of a polynomial $F(x)$ several times. In this way, we can reduce the "separation factor of the $\mathcal{T}_k$-Test" from polynomial in d (namely $256d^5$) to a constant value (in our case, this constant will be ≈ 1.41) when we run N, with $N = \Theta(\log\log d)$, Graeffe iterations first, and then apply the $\mathcal{T}_k$-test; see Algorithm 6.2. From Theorem 6.5 and Theorem 6.10, we then obtain the following result:

Lemma 6.11. *Let Δ be a disc in the complex plane and $F(x) \in \mathbb{C}[x]$ a polynomial of degree d. Let $N := \lceil \log(1 + \log d) \rceil + 5$ and $\rho_1 := \frac{2\sqrt{2}}{3} \approx 0.943$ and $\rho_2 := \frac{4}{3}$. Then, $\sqrt[2^N]{\frac{1}{16d}} > \rho_1$, and the following hold:*

Algorithm 6.1: Graeffe Iteration

```
Input  : Polynomial $F(x) = \sum_{i=0}^{d} a_i x^i$, and a non-negative integer $N$.
Output: Polynomial $F^{[N]}(x) = \sum_{i=0}^{d} a_i^{[N]} x^i$. If $F$ has roots $z_1, \dots, z_d$, then $F^{[N]}$ has roots
        $z_1^{2^N}, \dots, z_d^{2^N}$, and $a_d^{[N]} = a_d^{2^N}$
$F^{[0]}(x) := F(x)$
for $i = 1, \dots, N$ do
    $F^{[i]}(x) := (-1)^d [F_e^{[i-1]}(x)^2 - x \cdot F_o^{[i-1]}(x)^2]$
return $F^{[N]}(x)$
```

Algorithm 6.2: $\mathcal{T}_k^G(\Delta, K)$-Test

```
Input  : Polynomial $F(x)$ of degree $d$, disc $\Delta = \Delta_r(m)$, real value $K$ with $1 \le K \le \frac{3}{2}$
Output: TRUE or FALSE. If the algorithm returns TRUE, $\Delta$ contains exactly $k$ roots of $F$.
Call Algorithm 6.1 with input $F_\Delta(x) := F(m + r \cdot x)$ and $N := \lceil \log(1 + \log d) \rceil + 5$,
  which returns $F_\Delta^{[N]}$
return $\mathcal{T}_k(0, 1, K, F^{[N]}(x))$
```

(a) If Δ is (ρ_1, ρ_2)-isolating for a set of k roots of F, then $\mathcal{T}_k^G(\Delta, \frac{3}{2})$ succeeds.

(b) If $\mathcal{T}_k^G(\Delta, K)$ succeeds for some $K \ge 1$, then Δ contains exactly k roots.

Proof. The lower bound on $\rho(d) := \sqrt[2^N]{\frac{1}{16d}}$ follows by a straight forward computation that shows that $\rho(d)$, considered as a function in d, is strictly increasing and that $\rho(2) \approx 0.947 > \frac{2\sqrt{2}}{3} \approx 0.943$. Now, let $F_\Delta^{[N]}$ be the polynomial obtained from F_Δ after performing N recursive Graeffe iterations. If Δ is (ρ_1, ρ_2)-isolating for a set of k roots of F, then the unit disc $\Delta' := \Delta_1(0)$ is also (ρ_1, ρ_2)-isolating for a set of k roots of F_Δ, that is, Δ' contains k roots of F_Δ and all other roots of F_Δ have absolute value larger than $\frac{4}{3}$. Hence, we conclude that $F_\Delta^{[N]}$ has k roots of absolute value less than ${\rho_1}^{2^N} < \frac{1}{16d}$, whereas the remaining roots have absolute value larger than $\rho_2^{2^N} \ge 16d^4$. From Corollary 6.8, we thus conclude that $\mathcal{T}_k(\Delta', \frac{3}{2}, F_\Delta^{[N]})$ succeeds. This shows (a). Part (b) is an immediate consequence of Theorem 6.4 and the fact that Graeffe iteration does not change the number of roots contained in the unit disc. □

Note that in the special case where $k = 0$, the failure of $\mathcal{T}_0^G(\Delta)$ already implies that $\frac{4}{3} \cdot \Delta$ contains at least one root. The following result is a direct consequence of Theorem 6.10.

Corollary 6.12. *Let F_Δ and $F_\Delta^{[N]}$ be defined as in Algorithm 6.2. Then, it holds that*

$$\overline{\log}(\|F_\Delta^{[N]}(x)\|, \|F_\Delta^{[N]}(x)\|^{-1}) = O(\log d \cdot (d + \overline{\log}(\|F_\Delta\|, \|F_\Delta\|^{-1})).$$

6.4.2 The $\tilde{\mathcal{T}}_k^G$-Test: Using Approximate Arithmetic

So far, the $\mathcal{T}_k$-test and $\mathcal{T}_k^G$-test are formulated in a way such that, in general, high-precision arithmetic, or even exact arithmetic, is needed in order to compute its output. Namely, if the two expressions on both sides of (6.3) are actually equal, then exact arithmetic is needed to decide equality. Even worse, in our general setting, we cannot even handle this case as we have only access to (arbitrary good) approximations of the coefficients of the input polynomial F and even if the two expression are different but almost equal, then we need to evaluate the polynomial F (and its higher order derivatives) with a very high precision in order to decide the inequality, which induces high computational costs. This is a typical problem that appears in many algorithms, where a sign predicate $\mathcal{P}$ is used to draw conclusions, which in turn decides which branch of the algorithm is taken. For a predicate $\mathcal{P}$, we say that $\mathcal{P}$ succeeds if it returns True, and, otherwise, we say that it fails. Suppose that, similar as for the $\mathcal{T}_k$-test (with $E_\ell = |F^{(k)}(m)| \cdot r^k/k!$ and $E_r = \sum_{i \neq k} |F^{(i)}(m)| \cdot r^i/i!$), there exist two non-negative expressions E_ℓ and E_r such that $\mathcal{P}$ succeeds if and only if $E_\ell - E_r > 0$. We further denote by $\mathcal{P}_{3/2}$ the predicate that succeeds if and only if the stronger inequality $E_\ell - \frac{3}{2} \cdot E_r > 0$ holds. Then, success of $\mathcal{P}_{3/2}$ implies success of $\mathcal{P}$; however, a failure of $\mathcal{P}_{3/2}$ does, in general, not imply that $\mathcal{P}$ fails as well. As already mentioned above for the special case, where $\mathcal{P} = \mathcal{T}_k(m, r, 1, F)$, it might be computationally expensive (or even infeasible) to determine the outcome of $\mathcal{P}$, namely in the case where the two expressions E_ℓ and E_r are equal or almost equal. In order to avoid such undesirable situations, we replace the predicate $\mathcal{P}$ by a corresponding so-called *soft-predicate* [YSS13], which we denote by $\tilde{\mathcal{P}}$. This predicate $\tilde{\mathcal{P}}$ does not only return True or False, but may also return Undecided. If it returns True or False, the result of $\tilde{\mathcal{P}}$ coincides with that of $\mathcal{P}$. However, if $\tilde{\mathcal{P}}$ returns Undecided, we may only conclude that E_ℓ and E_r are roughly equal, more precisely $\frac{2}{3} \cdot E_\ell < E_r < \frac{3}{2} \cdot E_\ell$. We briefly sketch our approach and give details in Algorithm 6.3: In the first step, we compute approximations $\tilde{E}_\ell$ and $\tilde{E}_r$ of the values E_ℓ and E_r, respectively. Then, we check whether we can already compare the exact values E_ℓ and E_r by just considering their approximations and taking into account the quality of approximation. If this is the case, we are done as we can already determine the outcome of $\mathcal{P}$. Hence, in this case $\tilde{\mathcal{P}}$ returns True or False). Otherwise, we iteratively increase the quality of approximation until we can either show that $E_\ell > E_r$, $E_\ell < E_r$, or $\frac{2}{3} \cdot E_\ell \leq E_r \leq \frac{3}{2} \cdot E_\ell$. We may consider the latter case as an indicator that comparing E_ℓ and E_r is difficult, and thus $\tilde{\mathcal{P}}$ returns Undecided in this case.

It is easy to see that Algorithm 6.3 terminates if and only if at least one of the two expressions E_ℓ and E_r is non-zero, hence we make this a requirement. In the following lemma, we further give a bound on the precision to which the expressions E_ℓ and E_r have to be approximated in order to guarantee termination of the algorithm.

Algorithm 6.3: Soft-predicate $\tilde{\mathcal{P}}$

```
Input  : A predicate $\mathcal{P}$ defined by non-negative expressions $E_\ell$ and $E_r$, with $E_\ell \neq 0$ or
         $E_r \neq 0$; i.e., $\mathcal{P}$ succeeds if and only if $E_\ell > E_r$.
Output: True, False, or Undecided. In case of True (False), $\mathcal{P}$ succeeds (fails). In case
         of Undecided, we have $\frac{2}{3} \cdot E_\ell < E_r \leq \frac{3}{2} \cdot E_\ell$.
$L := 1$
while True do
    Compute $L$-bit approximations $\tilde{E}_\ell$ and $\tilde{E}_r$ of the expressions $E_\ell$ and $E_r$,
      respectively.
    $E_\ell^{\pm} := \max(0, \tilde{E}_\ell \pm 2^{-L})$ and $E_r^{\pm} := \max(0, \tilde{E}_r \pm 2^{-L})$
    if $E_\ell^- > E_r^+$ then return True            // It follows that $E_\ell > E_r$.

    if $E_\ell^+ < E_r^-$ then return False           // It follows that $E_\ell < E_r$.

    if $\frac{2}{3} \cdot E_\ell^+ \leq E_r^- < E_r^+ \leq \frac{3}{2} \cdot E_\ell^-$, then return Undecided
                                                       // It follows that $\frac{2}{3} \cdot E_\ell \leq E_r \leq \frac{3}{2} \cdot E_\ell$.
    $L := 2 \cdot L$
```

Lemma 6.13. *Algorithm 6.3 terminates for an L that is upper bounded by*

$$L_0 := 2 \cdot (\overline{\log}(\max(E_\ell, E_r)^{-1}) + 4).$$

Proof. Suppose that $L \geq \overline{\log}(\max(E_\ell, E_r)^{-1}) + 4$. We further assume that $E_\ell = \max(E_\ell, E_r)$; the case $E_r = \max(E_\ell, E_r)$ is then treated in analogous manner. It follows that

$$E_r^+ \leq E_r + 2^{-L+1} \leq E_\ell + 2^{-L+1} \leq \frac{9}{8} \cdot E_\ell \leq \frac{3}{2} \cdot E_\ell - 2^{-L+2} \leq \frac{3}{2} \cdot E_\ell^-.$$

Hence, if, in addition, $\frac{2}{3} \cdot E_\ell^+ \leq E_r^-$, then the algorithm returns Undecided in Step 10. Otherwise, we have $\frac{9}{8} \cdot E_\ell \geq E_\ell + 2^{-L+1} \geq E_\ell^+ > \frac{3}{2} \cdot E_r^-$, and thus

$$E_\ell^- \geq E_\ell - 2^{-L+1} \geq \frac{7}{8} \cdot E_\ell \geq \frac{3}{4} \cdot E_\ell + 2^{-L+1} \geq E_r^- + 2^{-L+1} \geq E_r^+,$$

which shows that the algorithm returns True in Step 6. Since we double L in each iteration, it follows that the algorithm must terminate for an L with $L < 2 \cdot (\overline{\log}(\max(E_\ell, E_r)^{-1}) + 4)$. □

Notice that if $\tilde{\mathcal{P}}$ returns True, then $\mathcal{P}$ also succeeds. This however does not hold in the opposite direction. In addition, if $\mathcal{P}_{3/2}$ succeeds, then $E_\ell > E_r$ and E_ℓ cannot be a relative $\frac{3}{2}$-approximation of E_r, hence $\tilde{\mathcal{P}}$ must return True. We conclude that our soft-predicate is somehow located "in between" the two predicates $\mathcal{P}$ and $\mathcal{P}_{3/2}$.

We now return to the special case, where $\mathcal{P} = \mathcal{T}_k(m, r, 1, F)$, with $E_\ell = |F^{(k)}(m)| \cdot r^k/k!$ and $E_r = \sum_{i \neq k} |F^{(i)}(m)| \cdot r^i/i!$ the two expressions on the left and the right side of (6.3), respectively. Then, success of $\mathcal{P}$ implies that the disc $\Delta = \Delta_r(m)$ contains exactly k roots of F, whereas a failure of $\mathcal{P}$ yields no further information. Now, let us consider the corresponding soft predicate $\tilde{\mathcal{P}} = \tilde{\mathcal{T}}_k(\Delta, F)$ of $\mathcal{P} = \mathcal{T}_k(\Delta, F)$. If $\tilde{\mathcal{P}}$ returns True, then this implies success of $\mathcal{P}$. In addition, notice that success of $\mathcal{T}_k(\Delta, \frac{3}{2}, F)$ implies that $\tilde{\mathcal{P}}$ returns True, and thus we may replace $\mathcal{T}_k(\Delta, \frac{3}{2}, F)$ by $\tilde{\mathcal{T}}_k(\Delta, F)$ in the second part of Theorem 6.5. Similarly, in Lemma 6.11, we may also replace $\mathcal{T}_k^G(\Delta, \frac{3}{2}, F)$ by the soft-version $\tilde{\mathcal{T}}_k^G(\Delta, F)$ of $\mathcal{T}_k^G(\Delta, F)$. We give more details for the computation of $\tilde{\mathcal{T}}_k(\Delta, F)$ and $\tilde{\mathcal{T}}_k^G(\Delta, F)$ in Algorithms 6.4 and 6.5, which are essentially applications of Algorithm 6.3 to the predicates $\mathcal{T}_k(\Delta, F)$ and $\mathcal{T}_k^G(\Delta, F)$. The lemma below summarizes our results. Based on Lemma 6.13, we also provide a bound on the precision L for which Algorithm 6.4 terminates and a bound for the bit complexity of Algorithm 6.4. A corresponding bound for the bit complexity of carrying out the $\tilde{\mathcal{T}}_k^G(\Delta, F)$-test for all $k = 0, \ldots, d$ is given in Lemma 6.14.

Lemma 6.14. *For a disc $\Delta := \Delta_r(m)$ in the complex plane and a polynomial $F \in \mathbb{C}[x]$ of degree d, the $\tilde{\mathcal{T}}_k(\Delta, F)$-test terminates with an absolute precision L that is upper bounded by*

$$L(\Delta, F) := L(m, r, F) := 2 \cdot \left(4 + \overline{\log}(\|F_\Delta\|^{-1})\right). \tag{6.8}$$

If $\mathcal{T}_k(\Delta, \frac{3}{2}, F)$ succeeds, the $\tilde{\mathcal{T}}_k(\Delta, F)$-test returns True. The cost for running the $\tilde{\mathcal{T}}_k(\Delta, F)$-test for all *$k = 0, \ldots, d$ is upper bounded by*

$$\tilde{O}(d(d \cdot \overline{\log}(m, r) + \tau_F + L(\Delta, F)))$$

bit operations. The algorithm needs an $\tilde{O}(d \cdot \overline{\log}(m, r) + \tau_F + L(\Delta, F))$-bit approximation of F.

Proof. Let $\mathcal{P} := \mathcal{T}_k(\Delta, 1, F)$ be the predicate that succeeds if and only if $E_\ell > E_r$, with $E_\ell := |f_k|$ and $E_r := \sum_{i \neq k} |f_i|$. Then, $E_\ell^\pm := f_k^\pm$ and $E_r^\pm := \sum_{i \neq k} f_i^\pm$ are lower and upper bounds for E_ℓ and E_r, respectively, such that $|E_\ell^\pm - E_\ell| \leq 2^{-L+1}$ and $|E_r^\pm - E_r| \leq 2^{-L+1}$. Hence, Lemma 6.13 yields that Algorithm 6.4 terminates for an L smaller than $2 \cdot (4 + \overline{\log}(\max(E_\ell, E_r)^{-1})) \leq L(\Delta, F)$.

Algorithm 6.4: $\tilde{\mathcal{T}}_k(\Delta, F)$-test

```
Input : A polynomial $F(x)$ of degree $d$, a disc $\Delta := \Delta_r(m)$ in the complex plane, and
        an integer $k$ with $0 \le k \le d$.
Output: TRUE, FALSE, UNDECIDED. If the algorithm returns TRUE, the disc $\Delta_r(m)$
        contains exactly $k$ roots of $F$.
$L := 1$
while TRUE do
    Compute an approximation $\tilde{F}_\Delta(x) = \sum_{i=0}^d \tilde{f}_i x^i$ of the polynomial
     $F_\Delta(x) := \sum_{i=0}^d f_i \cdot x^i := F(m + r \cdot x)$ such that $\tilde{f}_i \cdot 2^{L+\lceil \log(d+1) \rceil} \in \mathbb{Z}$ and
     $|f_i - \tilde{f}_i| < 2^{-L+\lceil \log(d+1) \rceil}$ for all $i$.
                                      // $(L + \lceil \log(d+1) \rceil)$-bit approximation of $F_\Delta$.
    $f_i^- := \max(0, |\tilde{f}_i| - 2^{-L-\lceil \log(d+1) \rceil})$ for $i = 0, \dots, d$.
    $f_i^+ := |\tilde{f}_i| + 2^{-L-\lceil \log(d+1) \rceil}$ for $i = 0, \dots, d$.
                                      // lower and upper bounds for $|f_i|$.
    if $f_k^- - \sum_{i \ne k} f_i^+ > 0$ then
        return TRUE
                                      // It follows that $\mathcal{T}_k(\Delta, F)$ succeeds.
    if $\sum_{i \ne k} f_i^- - f_k^+ > 0$ then
        return FALSE
                                      // It follows that $\mathcal{T}_k(\Delta, F)$ fails.
    if $\sum_{i \ne k} f_i^- - \frac{2}{3} \cdot f_k^+ \ge 0$ and $\frac{3}{2} \cdot f_k^- - \sum_{i \ne k} f_i^+ \ge 0$ then
        return FALSE
    $L := 2 \cdot L$
```

We have already argued above that success of the predicate $\mathcal{P}_{3/2} = \mathcal{T}_k(\Delta, \frac{3}{2}, F)$ implies that $\tilde{\mathcal{P}} = \tilde{\mathcal{T}}_k(\Delta, F)$ returns TRUE. Hence, it remains to show the claim on the bit complexity for carrying out the $\tilde{\mathcal{T}}_k(\Delta, F)$-test for all $k = 0, \dots, d$. For a given L, we can compute an $(L + \lceil \log(d+1) \rceil)$-bit approximation $\tilde{F}_\Delta(x) = \sum_{i=0}^d \tilde{f}_i x^i$ of F_Δ with a number of bit operations that is bounded by $\tilde{O}(d(\tau_F + d\,\overline{\log}(m, r) + L))$; e.g. see the first part of the proof of [SM16, Lemma 17]. For a fixed k, the computation of the signs of the sums in each of the three IF clauses needs d additions of dyadic numbers with denominators of bit-size $\lceil \log(d+1) \rceil + L$ and with numerators of bit-size $O(L + d\,\overline{\log}(r) + \tau_F)$, hence the cost is bounded by $O(d(\tau_F + d\,\overline{\log}(r) + L))$ bit operations. Notice that, when passing from an integer k to a $k' \ne k$, the corresponding sums in one If-clause differ only by two terms, that is, $f_k^\pm$ and $f_{k'}^\pm$. Hence, we can decide all If-clauses *for all* k using $O(d)$ additions. Furthermore, we double the precision L in each step, and the algorithm terminates for an L smaller than $L(\Delta, F)$. Hence, L is doubled at most $\log L(\Delta, F)$ many times, and thus the total cost for all k is bounded by $\tilde{O}(d(\tau_F + d\,\overline{\log}(m, r) + L(\Delta, F)))$ bit operations. □

We now extend the above soft-variant $\tilde{\mathcal{T}}_k$ of the $\mathcal{T}_k$-test to a corresponding soft-variant of the $\mathcal{T}_k^G$-test, which we denote $\tilde{\mathcal{T}}_k^G$; see Algorithm 6.5 for details. We further combine $\tilde{\mathcal{T}}_k^G$ for all $k = 0, \dots, d$ to obtain $\mathbf{T}_*(\Delta, F)$ with

$$\mathbf{T}_*(\Delta, F) := \begin{cases} k & \text{if there exists a } k \text{ such that } \tilde{\mathcal{T}}_k^G(\Delta, F) \text{ succeeds} \\ -1 & \text{otherwise.} \end{cases} \tag{6.9}$$

Again, for brevity, we often omit F and just write $\mathbf{T}_*(\Delta)$. We say that $\mathbf{T}_*$ succeeds if it returns a non-negative value. Otherwise, it fails. Notice the difference between $\mathbf{T}_*$ and $\mathcal{T}_*$,

```
Algorithm 6.5: $\tilde{\mathcal{T}}_k^G(\Delta, F)$-Test
Input  : Polynomial $F(x) \in \mathbb{C}[x]$ of degree $d$, a disc $\Delta := \Delta_r(m)$ in the complex space.
Output: TRUE, FALSE, or UNDECIDED. If the algorithm returns TRUE, $\Delta$ contains exactly $k$
         roots of $F$.
Let $F_\Delta^{[N]}(x)$ be the $N$-th Graeffe iterate of $F_\Delta(x) := F(m + r \cdot x)$, where
  $N := \lceil \log(1 + \log d) \rceil + 5$
return $\tilde{\mathcal{T}}_k(0, 1, F_\Delta^{[N]})$
```

which we defined in Chapter 5. The $\mathbf{T}_*$-test uses both Graeffe iteration and approximate arithmetic while the $\mathcal{T}_*$-test was formulated for exact arithmetic without Graeffe iteration.

The following result, which can be considered as the "soft variant" of Lemma 6.11, can then immediately be deduced from Lemma 6.11 and Lemma 6.14:

Lemma 6.15 (Soft-version of Lemma 6.11)**.** *Let $\Delta := \Delta_r(m)$ be a disc in the complex plane, $F(x) \in \mathbb{C}[x]$ be a polynomial of degree d, and let $\rho_1 = \frac{2\sqrt{2}}{3}$ and $\rho_2 = \frac{4}{3}$. Then, the following hold:*

(a) If Δ is (ρ_1, ρ_2)-isolating for a set of k roots of F, then $\mathbf{T}_*(\Delta)$ returns k.

(b) If $\mathbf{T}_*(\Delta)$ returns a $k \geq 0$, then Δ contains exactly k roots.

For the complexity analysis of our root isolation algorithm (see Section 6.5), we provide a bound on the total cost for running the $\mathbf{T}_*$-test.

Lemma 6.16. *The total cost for carrying out the $\mathbf{T}_*(\Delta)$ is bounded by*

$$\tilde{O}(d(\tau_F + d\,\overline{\log}(m, r) + L(\Delta, F))) = \tilde{O}(d(\tau_F + d\,\overline{\log}(m, r) + \overline{\log}((\max_{z \in \Delta} |F(z)|)^{-1})))$$

bit operations. For this, we need an L-bit approximation of F with

$$L = \tilde{O}(\tau_F + d\,\overline{\log}(m, r) + L(\Delta, F)) = \tilde{O}(\tau_F + d\,\overline{\log}(m, r) + \overline{\log}((\max_{z \in \Delta} |F(z)|)^{-1})).$$

Proof. According to Lemma 6.14, the computation of $\tilde{\mathcal{T}}_k(0, 1, F_\Delta^{[N]})$ needs an L-bit approximation $\tilde{F}_\Delta^{[N]}$ of $F_\Delta^{[N]}$, with L bounded by

$$\tilde{O}(d + \tau_{F_\Delta^{[N]}} + L(0, 1, F_\Delta^{[N]})) = \tilde{O}(d + \overline{\log}(\|F_\Delta^{[N]}\|, \|F_\Delta^{[N]}\|^{-1})). \tag{6.10}$$

Given such an approximation $\tilde{F}_\Delta^{[N]}$, the cost for running the test for all $k = 0, \dots, d$ is then bounded by

$$\tilde{O}(d(d + \tau_{F_\Delta^{[N]}} + L)) \text{ bit operations.}$$

In each of the $N = O(\log \log d)$ Graeffe iterations, the size of $\overline{\log}(\|F_\Delta^{[i]}\|, \|F_\Delta^{[i]}\|^{-1})$ increases by at most a factor of two plus an additive term $4d$; see Theorem 6.10. Hence, we must have

$$\begin{aligned}\overline{\log}(\|F_\Delta^{[i]}\|, \|F_\Delta^{[i]}\|^{-1}) &= O(\log d \cdot \overline{\log}(\|F_\Delta\|, \|F_\Delta\|^{-1}) + d \log d) \\ &= \tilde{O}(d\,\overline{\log}(m, r) + \tau_F + L(\Delta, F))\end{aligned}$$

for all $i = 0, \dots, N$. We conclude that the above bound (6.10) for L can be replaced by $\tilde{O}(\tau_F + d\,\overline{\log}(m, r) + L(\Delta, F))$.

It remains to bound the cost for computing an approximation $\tilde{F}_\Delta^{[N]}$ of $F_\Delta^{[N]}$ with $\|F_\Delta^{[N]} - \tilde{F}_\Delta^{N}\| < 2^{-L}$. Suppose that, for a given $\rho \in \mathbb{N}$ we have computed an approximation $\tilde{F}_\Delta$ of F_Δ, with $\|F_\Delta - \tilde{F}_\Delta\| < 2^{-\rho}$. According to [Sch82a, Theorem 8.4] (see also [KS13, Theorem 14] and [SM16, Lemma 17]), this can be achieved using a number of bit operations bounded by $\tilde{O}(d(d\,\overline{\log}(m,r)+\tau_F+\rho))$. In each Graeffe iteration, an approximation $\tilde{F}_\Delta^{[i]}$ of $F_\Delta^{[i]}$ is split into two polynomials $\tilde{F}_{\Delta,o}^{[i]}$ and $\tilde{F}_{\Delta,e}^{[i]}$ with coefficients of comparable bit-size (and half the degree), and an approximation $\tilde{F}_\Delta^{[i+1]}$ of $F_\Delta^{[i]}$ is then computed as the difference of $\tilde{F}_{\Delta,e}^{[i]}$ and $x\cdot\tilde{F}_{\Delta,o}^{[i]}$. If all computations are carried out with fixed point arithmetic and an absolute precision of ρ bits after the binary point, then the precision loss in the i-th step, with $i=0,\dots,N$, is bounded by $O(\log d+\log\|F_\Delta^{[i]}\|)=O(2^i(\log d+\log\|F_\Delta\|))=O(\log d(\log d+\log\|F_\Delta\|))$ bits after the binary point. The cost for the two multiplications and the addition is bounded by $\tilde{O}(d(\rho+\log\|F_\Delta^{[i]}\|))$. Since there are only $N=O(\log\log d)$ many iterations, we conclude that it suffices to start with an approximation $\tilde{F}_\Delta$ of F_Δ, with $\|F_\Delta-\tilde{F}_\Delta\|<2^{-\rho}$ and $\rho=\tilde{O}(d\,\overline{\log}(m,r)+\tau_F+L(\Delta,F))$. The total cost for all Graeffe iterations is then bounded by $\tilde{O}(d\rho)$ bit operations, hence the claim follows together with the fact that $\max_{z\in\Delta}|F(z)|\le(d+1)\|F_\Delta\|$ as shown in (6.7) in the proof of Theorem 6.10. □

6.5 ℂIsolate: An Algorithm for Root Isolation

We can now formulate our algorithm, which we denote by ℂIsolate. ℂIsolate computes isolating discs for all complex roots of a polynomial F within some given closed, axis-aligned square $\mathcal{B}\subset\mathbb{C}$ under the assumption that the enlarged square $2\mathcal{B}$ contains only simple roots of F. However, ℂIsolate might also return isolating discs for some of the roots that are only contained within the enlarged square $2\mathcal{B}$. In particular, in the important special case, where F is square-free and where we start with a square $\mathcal{B}$ that is known to contain all complex roots of F, see the end of Section 6.2 for the construction of such a square, the algorithm isolates all complex roots of F. Before we give details, we need some further definitions. Thereater, we will give an overview of the algorithm before we provide details and the proof for termination and correctness.

6.5.1 Connected Components

Given a set $S=\{B_1,\dots,B_m\}$ of squares $B_1,\dots,B_m\subset\mathbb{C}$, we say that two squares $B,B'\in S$ are connected in S ($B\sim_S B'$ for short) if there exist squares $B_{i_1},\dots,B_{i_s}\in S$ with $B_{i_1}=B$, $B_{i_s}=B'$, and $B_{i_j}\cap B_{i_{j+1}}\neq\emptyset$ for all $j=1,\dots,s'-1$. This yields a decomposition of S into equivalence classes $C_1,\dots,C_k\subset S$ that correspond to maximal connected and disjoint components $\bar{C}_\ell=\bigcup_{i:B_i\in C_\ell}B_i$, with $\ell=1,\dots,k$. Notice that formally C_ℓ is defined as the set of squares B_i that belong to the same equivalence class, whereas $\bar{C}_\ell$ denotes the closed region in $\mathbb{C}$ that consists of all points that are contained in a square $B_i\in C_\ell$. We will, for simplicity, abuse notation and simply use C to denote the set of squares B contained in a component C as well as to denote the set of points contained in the closed region $\bar{C}$. Now, let $C=\{B_1,\dots,B_s\}$ be a connected component consisting of equally sized squares B_i of width w, then we define (see also Figure 6.1):

- B_C is the axis-aligned closed square in $\mathbb{C}$ of minimal width such that $C\subset B_C$ and

$$\min_{z\in B_C}\Re(z)=\min_{z\in C}\Re(z)\text{ and }\max_{z\in B_C}\Im(z)=\max_{z\in C}\Im(z),$$

  where $\Re(z)$ denotes the real part and $\Im(z)$ the imaginary part of an arbitrary complex value z. The above condition (figuratively speaking) ensures that the square B_C is maximally moved to the "south-east".

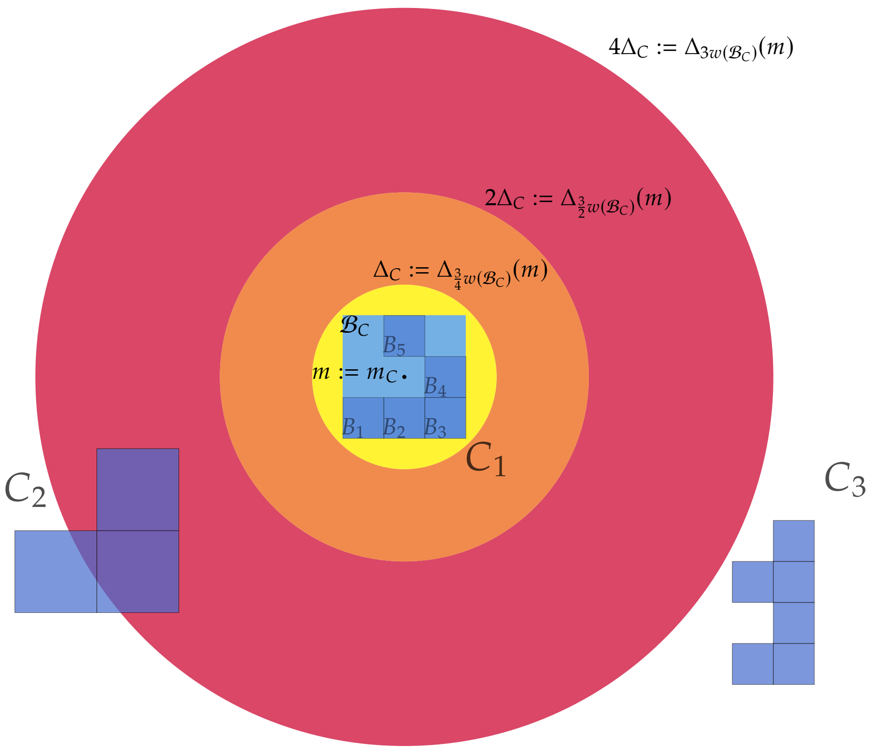
$4\Delta_C := \Delta_{3w(\mathcal{B}_C)}(m)$
$2\Delta_C := \Delta_{\frac{3}{2}w(\mathcal{B}_C)}(m)$
$\Delta_C := \Delta_{\frac{3}{4}w(\mathcal{B}_C)}(m)$
$\mathcal{B}_C$
B_5
$m := m_C$
B_4
B_1 B_2 B_3
C_1
C_2
C_3

FIGURE 6.1: A component $C_1 := C$ consisting of 5 squares $B_1, \dots B_5$, the enclosing square $\mathcal{B}_C$ with center $m := m_C$ and the discs Δ_C, $2\Delta_C$ and $4\Delta_C$. The disc $4\Delta_C$ intersects the component C_2 but does not intersect C_3.

- For a component C and corresponding square B_C, we denote m_C the center of B_C, and $\Delta_C := \Delta_{3w(B_C)/4}(m_C)$ a disc containing B_C, and thus also C and the *diameter* $w(C)$ *of the component* C we define to be the width of B_C, i.e., $w(C) := w(B_C)$, and $r(C) := w(C)/2$ is the *radius of* C.
- We define $C^+ := \bigcup_{i:B_i \in C} 2B_i$ as the union of the enlarged squares $2B_i$. Notice that C^+ is the $w/2$-neighborhood of C w.r.t. the ∞-norm.

6.5.2 The Algorithm

We start with an informal description of ℂISOLATE, where we focus on the main ideas explaining the ratio behind our choices. For the sake of comprehensibility, we slightly simplified some steps at the cost of complete formal correctness, hence, the considerations below should be taken with a grain of salt. A precise definition of the algorithm including all details is given in the pseudocode in Algorithm 6.6 and the subroutines NEWTONTEST (Algorithm 6.7) and BISECTION (Algorithm 6.8).

From a high-level perspective, our algorithm follows the classical subdivision approach of Weyl [Wey24]. That is, starting from the input square $\mathcal{B}$, we recursively subdivide $\mathcal{B}$ into smaller squares, and we remove squares for which we can show that they do not contain a root of F. Eventually, the algorithm returns regions that are isolating for a root of F. In order to discard a square B, with $B \subset \mathcal{B}$, we call the $\mathbf{T}_*(\Delta_B, F)$-test with Δ_B being the disc containing B. The remaining squares are then clustered into maximal connected components. We further check whether a component C is well-separated from all other components, that is, we test whether the distance from C to all other components is considerably larger than its diameter. If this is the case, we use the $\mathbf{T}_*$-test in order to determine the "multiplicity"

k_C of the component C, that is, the number of roots contained in the enclosing disc Δ_C; see Line 9 of Algorithm 6.6 for details and Figure 6.1 for an illustration. If $k_C = 1$, we may return an isolating disc for the corresponding unique root. Otherwise, there is a cluster consisting of two or more roots, which still have to be separated from each other. A straightforward approach to separate these roots from each other is to recursively subdivide each square into four equally sized squares and to remove squares until, eventually, each of the remaining components contains exactly one root that is well-separated from all other roots; see also Algorithm 6.8 (Bisection) and Figure 6.3. However, as described above in the overview, this approach itself yields only linear convergence to the roots, and the main idea to achieve quadratic convergence here is to consider a cluster of k roots as a single root of multiplicity k and to use a generalized Newton iteration for multiple roots to compute a better approximation of this root, see Algorithm 6.7 (NewtonTest) and Figure 6.2.

The main crux of the NewtonTest is that we never have to check in advance whether Newton iteration actually yields an improved approximation of the cluster of roots. Instead, correctness is verified independently using the $\mathbf{T}_*$-test. In order to achieve quadratic convergence in the presence of a well isolated root cluster, an integer N_C is assigned to each component C in each iteration. This speed parameter N_C may be thought of as the actual speed of convergence to the cluster of roots contained in C. In case of success of the NewtonTest, the component C is replaced by a component $C' \subset C$ of diameter $w(C') \approx w(C) \cdot N_C^{-1}$. That is the width of the component is reduced by a factor of N_C and, in this case, we "square the speed of convergence", that is, we set $N_{C'} := N_C^2$. If the NewtonTest fails, we fall back to bisection and "decrease the speed of convergence", that is, we set $N_{C'} := \sqrt{N_C}$ for all components C' into which the component C is split. Our analysis shows that the NewtonTest is the crucial ingredient for quadratic convergence. More precisely, we prove that, in the worst-case, the number s of components in each sequence $C_1, \dots, C_s$ as above becomes logarithmic in the length of such a sequence if only bisection would be used; see Lemma 6.21.

6.5.3 Termination and Correctness

We now turn to the proof of termination and correctness of the algorithm. In addition, we derive further properties, which will turn out to be useful in the analysis.

Theorem 6.17. *The algorithm* ℂIsolate *terminates and returns a correct result. In addition,* at any stage of the algorithm, *it holds that:*

(a) For any $(C, N_C) \in \mathcal{C}$, the connected component C consists of disjoint, aligned, and equally-sized squares $B_1, \dots, B_{s_C}$, each of width 2^{ℓ_C} with some $\ell_C \in \mathbb{Z}$.

(b) For any two distinct pairs $(C_1, N_{C_1}) \in \mathcal{C}$ and $(C_2, N_{C_2}) \in \mathcal{C}$, the distance between C_1 and C_2 is at least $\max(2^{\ell_{C_1}}, 2^{\ell_{C_2}})$. *In particular, the enlarged regions C_1^+ and C_2^+ are disjoint.*

(c) The union of all connected components C covers all roots of F contained in $\mathcal{B}$. In mathematical terms,

$$F(z) \neq 0 \text{ for all } z \in \mathcal{B} \setminus \bigcup_{C:(C,N_C)\in\mathcal{C}} C.$$

(d) For each square B produced by the algorithm that is not equal to the initial square $\mathcal{B}$, the enlarged square $2B$ contains at least one root of F.

(e) Each component C considered by the algorithm consists of $s_C \leq 9 \cdot |\mathcal{Z}(C^+)|$ squares. The total number of squares in all components C is at most 9-times the number of roots contained in $2\mathcal{B}$,

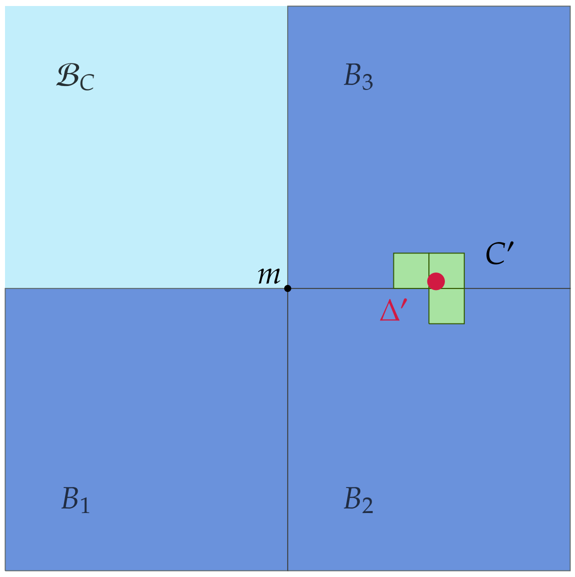

FIGURE 6.2: The NEWTONTEST: If $\mathbf{T}_*(\Delta') = k_C$, with $\Delta' := \Delta_{2^{\ell_C-3}/N_C}(\tilde{x}'_C)$, then Δ' contains exactly k_C roots of F. Since $\mathbf{T}_*(2\Delta_C) = k_C$ and $C^+ \subset 2\Delta_C$, it follows that Δ' contains all roots contained in C^+. The sub-squares $B_{i,j}$ of width $2^{\ell_C-1}/N_C$ that intersect Δ' yield a connected component C' of width at most $2^{\ell_C}/N_C \leq w(C)/N_C$. In addition, all roots that are contained in C are also contained in C'. Further notice that if $\tilde{x}'_C$ is contained in C, then Δ' intersects at most four squares $B_{i,j}$. Otherwise, it intersects at most three squares. In each case, the squares are connected with each other, and the corresponding connected component C' has width at most $2^{\ell_C}/N_C \leq w(C)/N_C$.

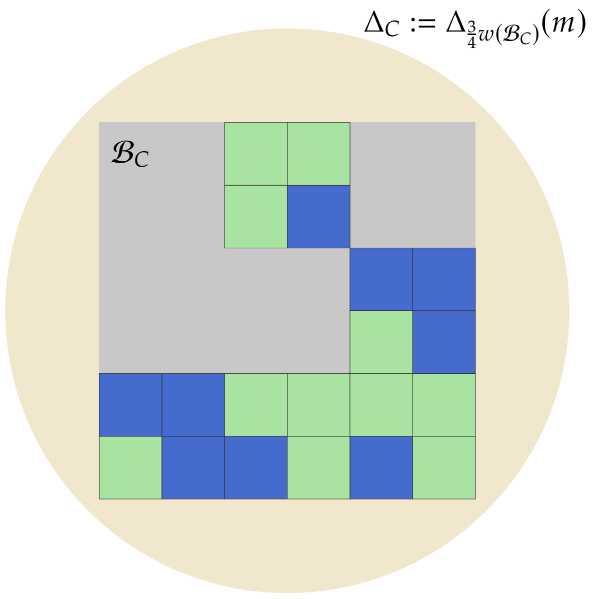

FIGURE 6.3: The BISECTION routine: The green (brighter) sub-squares are all squares B for which $\mathbf{T}_*(\Delta_B) \neq 0$. They are grouped together into three maximal connected components, which contain all roots contained in C. All other sub-squares are discarded.

Algorithm 6.6: ℂIsolate

```
Input : A polynomial $F(x) \in \mathbb{C}[x]$ as in (6.1) and a square $\mathcal{B} \subset \mathbb{C}$ of width
        $w_0 := w(\mathcal{B}) = 2^{\ell_0}$, with $\ell_0 \in \mathbb{Z}$; $F$ has only simple roots in $2\mathcal{B}$.
Output: A list $\mathcal{O}$ of disjoint discs $\Delta_1, \ldots, \Delta_s \subset \mathbb{C}$ such that, for each $i = 1, \ldots, s$, the
        disc $\Delta_i$ as well as the enlarged disc $2\Delta_i$ is isolating for a root of $F$ that is
        contained in $2\mathcal{B}$. In addition, for each root $z \in \mathcal{B}$, there exists a disc $\Delta_i \in \mathcal{O}$
        that isolates $z$.
1  $\mathcal{O} = \{\}$                      // list of isolating discs
2  $\mathcal{C} = \{(\mathcal{B}, 4)\}$      // list of pairs $(C, N_C)$, with $C$ a connected com-
                                          // ponent consisting of $s_C$ equally sized squares,
                                          // each of width $2^{\ell_C}$, where $\ell_C \in \mathbb{Z}_{\le \ell_0}$. $N_C$ is an
                                          // integer with $N_C = 2^{2^{n_C}}$ and $n_C \in \mathbb{N}_{\ge 1}$.
   // * Preprocessing *//
3  repeat
4      Let $(C, N_C)$ be the unique pair in $\mathcal{C}$
                                          // If $\bigcup_{C:(C,N_C)\in\mathcal{C}} C = \mathcal{B}$, then there exists a
                                          // unique component $C$ with $(C, N_C) \in \mathcal{C}$.
       // * linear step *//
5      $\{C'_1, \ldots, C'_\ell\}$ := Bisection$(C)$ and $\mathcal{C} = \{(C'_1, 4), \ldots, (C'_\ell, 4)\}$
6  until $\bigcup_{C:(C,N_C)\in\mathcal{C}} C \neq \mathcal{B}$
   // * Main Loop *//
7  while $\mathcal{C}$ is non-empty do
8      Remove a pair $(C, N_C)$ from $\mathcal{C}$.
9      if $4\Delta_C \cap C' = \emptyset$ for each $(C', N_{C'}) \in \mathcal{C}$ with $C' \neq C$ and there exists a $k_C \in [d]$ such that
         $k_C = \mathbf{T}_*(2\Delta_C) = \mathbf{T}_*(4\Delta_C)$ then
                                          // If the second condition holds, $k_C$ equals the
                                          // number of roots contained in $2\Delta_C$ and $4\Delta_C$.
           if $k_C = 1$ then
10             Add the disc $2\Delta_C$ to $\mathcal{O}$, continue
11         if $k_C > 1$ then
12             Let $x_C \in \mathcal{B} \setminus C$ be an arbitrary point with distance $2^{\ell_C - 1}$ from $C$ and
                 distance $2^{\ell_C - 1}$ or more from the boundary of $\mathcal{B}$.
                                          // Existence of such a point follows from the
                                          // proof of Theorem 6.17. It holds that $F(x_C) \neq 0$.
13             if NewtonTest$(C, N_C, k_C, x_C)$ = (Success, $C'$) then
                   // * quadratic step *//
14                 Add $(C', N_C^2)$ to $\mathcal{C}$, continue
       // * linear step *//
15     $\{C'_1, \ldots, C'_\ell\}$ := Bisection$(C)$.
16     Add $(C'_1, \max(4, \sqrt{N_C})), \ldots, (C'_\ell, \max(4, \sqrt{N_C}))$ to $\mathcal{C}$.
17 return $\mathcal{O}$.
```

that is,[2]

$$\sum_{C:\exists(C,N_C)\in\mathcal{C}} s_C \le 9 \cdot |\mathcal{Z}(2\mathcal{B})|.$$

[2]We will later prove that even the total number of squares produced by the algorithm in *all iterations* is near-linear in the number $\mathcal{Z}(2\mathcal{B})$ of roots contained in $2\mathcal{B}$.

```
Algorithm 6.7: NewtonTest
Input  : A tuple $(C, N_C, k_C, x_C)$: $C = \{B_1, \dots, B_{s_C}\}$ is a connected component
         consisting of equally sized and aligned squares $B_i$ contained in $\mathcal{B}$ and of size
         $2^{\ell_C}$, $N_C$ is an integer of the form $2^{2^{n_C}}$ with $n_C \in \mathbb{N}_{\geq 1}$, $k_C$ is the number of roots
         in $4\Delta_C$, and $x_C$ is a point with $F(x_C) \neq 0$.
Output: Either Failure or (Success, $C'$), where $C' \subset C$ is a connected component that
         contains all roots contained in $C$. $C'$ consists of at most 4 equally sized and
         aligned squares, each of width $\frac{2^{\ell_C - 1}}{N_C}$.
1  if Algorithm 6.3 does not return False for the input $E_\ell := 4r(C)|F'(x_C)|$ and $E_r := |F(x_C)|$
    then                              // This implies $|F(x_C)| < 6r(C)|F'(x_C)|$.
2     for $L = 1, 2, 4, \dots$ do
3        Compute $L$-bit approximations of $F(x_C)$ and $F'(x_C)$ and derive an
          $(6 - \ell_C + \log N_C)$-bit approximation $\tilde{x}'_C$ of the Newton iterate
             $x'_C := x_C - k_C \cdot \frac{F(x_C)}{F'(x_C)}$  such that $|\tilde{x}'_C - x'_C| < \frac{1}{64} \cdot \frac{2^{\ell_C}}{N_C}$.   (6.11)
                                      // For more details, see the similar computation
                                      // in [SM16, Step 2 of NewtonTest].
4     Let $\Delta' := \Delta(\tilde{x}'_C, \frac{1}{8} \cdot \frac{2^{\ell_C}}{N_C})$.
5     if $\Delta' \cap C = \emptyset$ then
6        return Failure
7
8     if $\mathbf{T}_*(\Delta') = k_C$ holds        // Then, $\Delta'$ contains all roots contained in $2\Delta_C$.
9      then
10       Decompose each square $B_i$ into $4N_C^2$ many equally sized sub-squares $B_{i,j}$
11       return (Success, $C'$), with $C'$ the unique connected component consisting of all
          squares $B_{i,j}$ of width $\frac{2^{\ell_C - 1}}{N_C}$ that intersect $\Delta'$.
12 return Failure
```

(f) *Let (C, N_C) be a pair produced by the algorithm. The sequence of ancestors of a component C produced by the algorithm is recursively defined as follows. It consists of the component C' from which C resulted followed by the ancestors of C'. We denote with* $\operatorname{anc}^*(C)$, *the first ancestor of C for which the Newton Test was successful, if such exists, otherwise* $\operatorname{anc}^*(C) = \mathcal{B}$. *Then* $w(C) \leq \frac{2w(\operatorname{anc}^*(C))}{\sqrt{N_C}} \leq \frac{2w(\mathcal{B})}{\sqrt{N_C}}$. *Moreover,*

(g) *It holds that* $\frac{\sigma_F(2\mathcal{B})^2}{2^{17} \cdot d^2 \cdot w(\mathcal{B})} \leq 2^{\ell_C} \leq w(\mathcal{B})$ *and* $4 \leq N_C \leq \left(\frac{2^9 \cdot w(\mathcal{B})}{\sigma_F(2\mathcal{B})}\right)^2$, *where as before* $\sigma_F(2\mathcal{B}) := \min_{i: z_i \in 2\mathcal{B}} \sigma_F(z_i)$ *is the separation of F restricted to $2\mathcal{B}$.*

Proof. (a) Follows immediately via induction. Namely, a component C consisting of squares of size 2^{ℓ_C} is either replaced by a single connected component consisting of (at most 4) squares of width $2^{\ell_C - 1}/N_C$ in line 14 after NewtonTest was called, or it is replaced by a set of connected components $C' \subset C$, each consisting of squares of size $2^{\ell_C - 1}$ in line 16 after Bisection was called.

(b) We can also use induction on the number of iterations. Suppose first that a component C is obtained from processing a component D in line 16. If C is the only connected

Algorithm 6.8: Bisection

```
Input  : A connected component $C = \{B_1, \dots, B_{s_C}\}$ consisting of aligned squares $B_i$,
         each of width $w(B_i) = 2^{\ell_C}$.
Output: A list of components $C'_j \subset C$, each consisting of aligned and equally sized
         squares of width $2^{\ell_C - 1}$. The union of all $C'_j$ contains all roots of $F$ that are
         contained in $C$.
1 $C' := \emptyset$
2 for each $B_i \in C$ do
3     Remove $B_i$ from $C$ and subdivide $B_i$ into four equally sized sub-squares $B_{i,j}$, with
      $j = 1, \dots, 4$, and add these to $C'$.
4 for each $B \in C'$ do
5     if $\mathbf{T}_*(\Delta_B) = 0$                    // This implies that B contains no root.
6     then
7         Remove $B$ from $C'$.
8 Compute maximal connected components $C'_1, \dots C'_\ell$ from the squares in $C'$.
9 return $C'_1, \dots C'_\ell$
```

component obtained from D, then, by the induction hypotheses, it follows that the distance to all other components C', with $C' \cap D = \emptyset$, is at least $\max(2^{\ell_D}, 2^{\ell_{C'}}) \geq \max(2^{\ell_C}, 2^{\ell_{C'}})$. If D splits into several components $C_1, \dots, C_s$, with $s > 1$, their distance to any component C', with $C' \cap D = \emptyset$, is at least $\max(2^{\ell_D}, 2^{\ell_{C'}}) \geq \max(2^{\ell_{C_i}}, 2^{\ell_{C'}})$ for all i. In addition, the pairwise distance of two disjoint components C_i and C_j is at least $2^{\ell_C - 1} = 2^{\ell_{C_i}}$ for all i. Finally, suppose that, in line 14, we replace a component D by a single component C. In this case, $C \subset D$ and C consists of squares of width $2^{\ell-1}/N_C$. Hence, the distance from C to any other component C' is also lower bounded by $\max(2^{\ell_C}, 2^{\ell_{C'}})$.

(c) Notice that in line 7 of Bisection, we discard a square B only if $\mathbf{T}_*(\Delta_B) = 0$. Hence, in this case, B contains no root of F. It remains to show that each root of F contained in C is also contained in C', where $C' \subset C$ is a connected component as produced in line 14 after NewtonTest was called. If $\mathbf{T}_*(\Delta') = k_C$, then Δ' contains k_C roots; see Lemma 6.15. Hence, since Δ' is contained in $2\Delta_C$, and since $2\Delta_C$ also contains k_C roots (as $\mathbf{T}_*(2\Delta_C) = k_C$ holds), it follows that Δ' contains all roots that are contained in C. The disc Δ' intersects no other component $C' \neq C$ as the distance from C to C' is larger than 2^{ℓ_C}, and thus, by induction, we conclude that $(\Delta' \cap \mathcal{B}) \setminus C$ contains no root of F. This shows that C' already contains all roots contained in C.

(d),(e) Any square $B \neq \mathcal{B}$ that is considered by the algorithm either results from the Bisection or from the NewtonTest routine. If a square B results from the Bisection routine, then the disc $\Delta_B = \Delta_{w(B)}(m_B)$ contains at least one root of F, and thus also $2B$ contains at least one root. If a square B results from the NewtonTest routine, then $2B$ even contains two roots or more. Namely, in this case, $\mathbf{T}_*(\Delta') = k_C$ holds for the disc $\Delta' = \Delta_{r'}(m')$, with $r' = \frac{1}{4}w(B)$ and $k_C > 1$, and thus Δ' contains k_C roots. Since $2B$ contains the latter disc, $2B$ must contain at least k_C roots. This shows (d). From (d), we immediately conclude that, for each component $C \neq \mathcal{B}$ produced by the algorithm, the enlarged component C^+ contains at least one root of F. In addition, since C^+ is contained in $2\mathcal{B}$, each of these roots must be contained in $2\mathcal{B}$. The first part in (e) now follows from the fact that, for a fixed root of F, there can be at most 9 different squares B of the same size such that

$2B$ contains this root. From part (b), it follows that, for any two distinct components C_1 and C_2, the enlarged components C_1^+ and C_2^+ do not intersect, and thus the total number of squares in all components is upper bounded by $9 \cdot |\mathcal{Z}(2\mathcal{B})|$, which proves the second part in (e).

(f) We may assume that $N_C > 4$ as, otherwise, the inequality becomes trivial. Let $C_1, \dots, C_s$ be the sequence of ancestors of C, with $C_1 := \operatorname{anc}^*(C)$, $C_s = C$, and $C_i \supset C_{i+1}$. By definition of $\operatorname{anc}^*(C)$, we have $w(C_2) \leq w(C_1)/N_{C_1} = w(C_1)/\sqrt{N_{C_2}}$, since the step from C_1 to C_2 is a quadratic step. And since $N_{C_i} = \sqrt{N_{C_{i-1}}}$ for $i = 2, \dots, s$, it follows that

$$w(C) = w(C_s) \leq w(C_{s-1}) \leq w(C_2) \leq \frac{w(C_1)}{N_{C_1}} = \frac{w(C_1)}{\sqrt{N_{C_2}}} \leq \frac{w(C_1)}{\sqrt{N_{C_s}}} = \frac{w(\operatorname{anc}^*(C))}{\sqrt{N_C}}.$$

Termination and (g): We can now show that the algorithm terminates; the inequalities in *(g)* will then follow from the proof of termination: Suppose that the algorithm produces a sequence $C_1, C_2, \dots, C_s$ of connected components, with $s \geq \log n + 6$ and $C_1 \supset C_2 \supset \cdots \supset C_s$. If, for at least one index $i \in \{1, \dots, s-1\}$, C_{i+1} is obtained from C_i via a quadratic step, then $w(C_{i+1}) \leq w(C_i)/N_{C_i} \leq w(C_i)/4$. Hence, in this case, we also have $w(C_s) \leq w(C_1)/4$. Now, suppose that each C_{i+1} is obtained from C_i via a linear step, then each square in C_i has size $2^{\ell_{C_1} - i + 1}$, and thus $w(C_s) \leq 9d \cdot 2^{\ell_{C_1} - s + 1} \leq w(C_1)/2$. This shows that, after at most $\log d + 6$ iterations, the width of each connected component is halved. Hence, in order to prove termination of the algorithm, it suffices to prove that each component C of small enough width is terminal, that is C is replaced by an isolating disc in line 10 or discarded in NewtonTest or Bisection. The following argument shows that each component C of width smaller than $w := \frac{1}{32} \cdot \sigma_F(2\mathcal{B})$ that is not discarded is replaced by an isolating disc. We have already shown that C^+ must contain a root ξ of F, and thus we have $|m_C - \xi| < 2w(C) < \sigma_F(\xi)/16$ and $r_C < \sigma_F(\xi)/16$. We conclude that the discs $2\Delta_C$ and $\Delta(m_C, 8r_C)$ are both isolating for ξ. Then, Lemma 6.15 guarantees that $\mathbf{T}_*(2\Delta_C) = 1$ and $\mathbf{T}_*(4\Delta_C) = 1$ hold. Hence, if $4\Delta_C$ intersects no other component $C' \neq C$, then the algorithm replaces C by the isolating disc $2\Delta_C$ in line 10, because the if-clause in line 9 succeeds with $k_C = 1$. It remains to show that the latter assumption is always fulfilled. Namely, suppose that $4\Delta_C$ intersects a component $C' \neq C$, and let B and B' be arbitrary squares contained in C and C', respectively. Then, the enlarged squares $2B$ and $2B'$ contain roots ξ and ξ', respectively, and ξ and ξ' must be distinct as C^+ and $(C')^+$ are disjoint. Hence, the distance between B and B', and thus also the distance δ between C and C', must be larger than $\sigma_F(2\mathcal{B}) - 2^{\ell_C} - 2^{\ell_{C'}} = 32w - 2^{\ell_C} - 2^{\ell_{C'}} \geq 31w - 2^{\ell_{C'}}$. Hence, if $2^{\ell_{C'}} \leq 25w$, then $4\Delta_C \subset \Delta(m_C, 6w)$ does not intersect C'. Vice versa, if $2^{\ell_{C'}} > 25w$, then the distance between C and C' is at least $\max(2^{\ell_C}, 2^{\ell_{C'}}) > 25w$, and thus $4\Delta_C$ does not intersect C' as well. Notice that *(g)* now follows almost directly from the above considerations. Indeed, let $C \neq \mathcal{B}$ be an arbitrary component C and let D be any component that contains C. Since D is not terminal, we conclude that $w(D) \geq w$, and thus $N_D \leq \frac{4w(\mathcal{B})}{w}$ according to (f). Since N_C is smaller than or equal to the square of the maximum of all values N_D, the second inequality in *(g)* follows. The first inequality follows from the fact that $2^{\ell_C} \geq \min_{D: C \subset D} \cdot 2^{\ell_D - 1}/N_D \geq w/(9d \cdot \max_{D: C \subset D} N_D)$.

Correctness: For correctness, we remark that each disc Δ returned by the algorithm is actually isolating for a root of F contained in $2\mathcal{B}$ and that 2Δ also isolates this root. Namely, for each component C produced by the algorithm, the enlarged component C^+ contains at least one root. Now, if the if-clause in line 9 succeeds on C with $k_C = 1$, it holds that $\mathbf{T}_*(2\Delta_C) = 1$, and thus the disc $2\Delta_C$ contains exactly one root ξ. Hence, since Δ_C contains C^+, this root must be contained in C^+. In addition, if also $\mathbf{T}_*(4\Delta_C) = 1$ holds, then the disc $4\Delta_C$ isolates ξ as well. Finally, it remains to show that the algorithm returns an isolating disc for each root ξ that is contained in $\mathcal{B}$. From (a) and (c), we conclude that there is a unique maximal sequence $\mathcal{S} = C_1, C_2, \ldots, C_s$ of connected components, with $C_1 \supset C_2 \supset \cdots \supset C_s$, such that each C_i contains ξ. Now, when processing C_s, C_s cannot be replaced by other connected components $C' \subset C_s$ as one of these components would contain ξ, and this would contradict the assumption that the sequence $\mathcal{S}$ is maximal. Since C_s contains ξ, it cannot be discarded in Bisection or NewtonTest, hence C_s is replaced by an isolating disc for ξ in line 10. This concludes the proof. □

Remark. We remark that our requirement on the input polynomial F to have only simple roots in $2\mathcal{B}$ is only needed for the termination of the algorithm. Running the algorithm on an arbitrary polynomial (possibly having multiple roots) yields isolating discs for the simple roots as well as arbitrarily small connected components converging against the multiple roots of F in $\mathcal{B}$. Namely, if $\mathcal{B}$ is not discarded in the first iteration, then the enlargement C^+ of each component C contains at least one root. Since C consists of at most $9d$ squares, each of size 2^{ℓ_C}, it holds that each point in C approximates a root of F to an error of less than $d \cdot 2^{\ell_C+4}$. In addition, the union of all components covers all roots contained in $\mathcal{B}$, and thus our algorithm yields L-bit approximations of all roots in $\mathcal{B}$ if we iterate until $\ell_C \leq -4 - \log d - L$ for all components C. In the special situation, where we run the algorithm on an input square that is known to contain all roots and if, in addition, the number k of distinct roots of F is given as input, our algorithm can be used to return isolating regions for all roots. Namely, in this situation, we may proceed until the total number of connected components C equals k. Then, each of the enlarged components C^+ isolates a root of F.

6.6 Complexity Analysis

We split the analysis into two parts. In the first part, we focus on the number of iterations that are needed to isolate the roots of F that are contained in a given square $\mathcal{B}$. We will see that this number is near-linear[3] in $|\mathcal{Z}(2\mathcal{B})|$, the number of roots contained in the enlarged square $2\mathcal{B}$. We further remark that, for any fixed non-negative constant ε, the total number of iterations is near-linear in $|\mathcal{Z}((1+\varepsilon)\cdot\mathcal{B})|$; however, for the sake of simplifying analysis, we only provide details for the special case $\varepsilon = 1$. Hence, we conclude that our algorithms performs near-optimal with respect to the number of subdivision steps if the input square $\mathcal{B}$ has the property that each root contained in $(1+\varepsilon)\cdot\mathcal{B}$ is also contained in $\mathcal{B}$; in particular, this is trivially fulfilled if $\mathcal{B}$ is chosen large enough to contain all roots of F.

In the second part of the analysis, we give bounds on the number of bit operations that are needed to process a component C. This eventually yields a bound on the overall bit complexity that is stated in terms of the degree of F, the absolute values and the separations of the roots in $\mathcal{Z}(2\mathcal{B})$, and the absolute value of the derivative F' at these roots.

6.6.1 Size of the Subdivision Tree

We consider the subdivision tree $\mathcal{T}_{\mathcal{B}}$, or simply $\mathcal{T}$, induced by $\mathbb{C}$Isolate, where $\mathcal{B}$ is the initial square. More specifically, the nodes of the tree $\mathcal{T}$ are the pairs $(C, N_C) \in \mathcal{C}$ produced by the algorithm, and two nodes (C, N_C) and $(C', N_{C'})$ are connected via an edge if and only if $C \subset C'$ (or $C' \subset C$) and there exists no other component C'' with $C \subset C'' \subset C'$ ($C' \subset C'' \subset C$). In the first case, we say that (C, N_C) is a child of $(C', N_{C'})$, whereas, in the second case, (C, N_C) is the parent of $(C', N_{C'})$. For brevity, we usually omit the integer N_C, and just refer to C as the nodes of $\mathcal{T}$. Notice that, according to Theorem 6.17, the so obtained graph is indeed a

[3]More precisely, it is linear in $|\mathcal{Z}(2\mathcal{B})|$ up to a factor that is polynomially bounded in $\log d$, $\log\overline{\log}(w(\mathcal{B}))$, and $\log\overline{\log}(\sigma_F(2\mathcal{B})^{-1})$. If $2\mathcal{B}$ contains no root, then there is only one iteration.

tree rooted at $\mathcal{B}$. A node C is called *terminal* if and only if it has no children. We further use the following definition to refer to some special nodes:

Definition 6.18. *A node $(C, N_C) \in \mathcal{T}$ is called* **special**, *if one of the following four conditions is fulfilled: (1) The node (C, N_C) is* **terminal**. *(2) The node (C, N_C) is the* **root** *of $\mathcal{T}$, that is, $(C, N_C) = (\mathcal{B}, 4)$. (3) The node (C, N_C) is the last node for which* Bisection *is called in the preprocessing phase of the algorithm. We call this node the* **base** *of $\mathcal{T}$. Notice that the first part of the tree consists of a unique path connecting the root and the base of the tree. (4) For each child D of C, it holds that $\mathcal{Z}(D^+) \neq \mathcal{Z}(C^+)$. We call such nodes* **split nodes**.

Roughly speaking, except for the root and the base of $\mathcal{T}$, special nodes either isolate a root of F or they are split into two or more disjoint clusters each containing roots of F. More precisely, from Theorem 6.17, we conclude that, for any two distinct nodes $C, D \in \mathcal{T}$, the enlarged regions $\mathcal{Z}(C^+)$ and $\mathcal{Z}(D^+)$ are either disjoint or one of the nodes is an ancestor of the other one. In the latter case, we have $C^+ \subset D^+$ or $D^+ \subset C^+$. Since, for any two children D_1 and D_2 of a node C, the enlarged regions D_1^+ and D_2^+ are disjoint, we have $\sum_{i=1}^{k} \mathcal{Z}(D_i^+) \leq \mathcal{Z}(C^+)$, where D_1 to D_k are the children of C. Hence, since each D_i^+ contains at least one root, the fourth condition in Definition 6.18 is violated if and only if C has exactly one child D and $\mathcal{Z}(C^+) = \mathcal{Z}(D^+)$. The number of special nodes is at most $2 \cdot (1 + |\mathcal{Z}(2\mathcal{B})|)$ as there is one root and one base, at most $|\mathcal{Z}(2\mathcal{B})|$ terminal nodes C with $C \neq \mathcal{B}$, and each occurrence of a special node, which fulfills the fourth condition, yields a reduction of the non-negative number $\sum_C (|\mathcal{Z}(C^+)| - 1)$ by at least one. The subdivision tree $\mathcal{T}$ now decomposes into special nodes and sequences of non-special nodes $C_1, \ldots, C_s$, with $C_1 \supset C_2 \supset \cdots \supset C_s$, that connect two consecutive special nodes. The remainder of this section is dedicated to the proof that the length s of such a sequence is bounded by some value $s_{\max}$ of size

$$s_{\max} = O\Big(\log\big(d \cdot \overline{\log}(w(\mathcal{B})) \cdot \overline{\log}(\sigma_F(2\mathcal{B})^{-1})\big)\Big). \tag{6.12}$$

For the proof, we need the following lemma, which provides sufficient conditions for the success of the NewtonTest.

Lemma 6.19 (Success of NewtonTest).

- *Let $C = \{B_1, \ldots, B_{s_C}\}$ be a non-terminal component with $\mathcal{B} \setminus C \neq \emptyset$, let $\mathcal{B}_C$ be the corresponding enclosing square of width $w(C)$ and center $m = m_C$, and let $\Delta := \Delta_C = \Delta_r(m)$, with $r := \frac{3}{4} w(C)$, be the corresponding enclosing disc.*
- *Let $z_1, \ldots, z_k$ be the roots of F contained in the enlarged component C^+, and suppose that all these roots are contained in a disc $\Delta'' := \Delta_{r''}(m'')$ of radius $r'' = 2^{-20-\log d} r / N_C$. Assume that the disc $\Delta_{2^{2\log d + 20} N_C r}(m)$ contains none of the roots $z_{k+1}, \ldots, z_d$.*

Then, the algorithm ℂIsolate *performs a quadratic step, that is, C is replaced by a single component C' of width $w(C') \leq w(C)/N_C$.*

Proof. We first argue by contradiction that 4Δ does not intersect any other component C', which implies that the first condition of the **and** in the if-clause in line 9 is fulfilled. If 4Δ intersects C', then the distance between C and C' is at most $8r$, and thus $2^{\ell_{C'}} < 8r$ as the distance between C and C' is at least $\max(2^{\ell_C}, 2^{\ell_{C'}})$. Hence, we conclude that the disc $64 \cdot \Delta$ completely contains $2B'$ for some square B' of C'. Since $2B'$ also contains at least one root and since each such root must be distinct from any of the roots $z_1, \ldots, z_k$, we get a contradiction.

According to our assumptions, each of the two discs Δ and 8Δ contains the roots $z_1, \ldots, z_k$ but no other root of F. Hence, according to Lemma 6.15, $\mathbf{T}_*(2\Delta) = \mathbf{T}_*(4\Delta) = k$ holds. Since we assumed C to be non-terminal, we must have $k \geq 2$, and thus the algorithm reaches line 11 and the NewtonTest is called. We assumed that C does not entirely cover the initial square $\mathcal{B}$, hence, in a previous iteration, we must have discarded a square of width 2^{ℓ_C} or more whose boundary shares at least one point with the boundary of C. Hence, we can choose a point in such a square as the point $x_C \in \mathcal{B} \setminus C$ in the NewtonTest such that the distance from x_C to C is equal to $2^{\ell_C - 1}$ and such that the distance from x_C to the boundary of $\mathcal{B}$ is at least 2^{ℓ_C-1}. Notice that also the distance from x_C to any other component C' is at least 2^{ℓ_C-1}, and thus the distance from x_C to any root of F is at least $2^{\ell_C-1} \geq \frac{r}{27d}$, where the inequality follows from the fact that C consists of at most $9d$ squares. Thus, for $i \leq k$, $|x_C - m''| \leq 4r$ and furthermore

$$|x_C - z_i| \geq \frac{r}{27d} \text{ for } i \leq k, \quad \text{and} \quad |x_C - z_i| \geq 2^{20} d^2 N_C r - 4r > 2^{19} d^2 N_C r \text{ for } i > k.$$

Using that $\frac{F'(x)}{F(x)} = \sum_{i=1}^d \frac{1}{x - z_i}$ for any x with $F(x) \neq 0$, we can bound the distance from the Newton iterate x'_C as defined in (6.11) to the "center" m'' of the cluster of roots:

$$\begin{aligned}
\left| \frac{1}{k} \frac{(x_C - m'')F'(x_C)}{F(x_C)} - 1 \right| &= \left| \frac{1}{k} \sum_{i=1}^k \frac{x_C - m''}{x_C - z_i} + \frac{1}{k} \sum_{i>k} \frac{x_C - m''}{x_C - z_i} - 1 \right| \\
&= \frac{1}{k} \left| \sum_{i=1}^k \frac{z_i - m''}{x_C - z_i} + \sum_{i>k} \frac{x_C - m''}{x_C - z_i} \right| \leq \frac{1}{k} \sum_{i=1}^k \frac{|z_i - m''|}{|x_C - z_i|} + \sum_{i>k} \frac{|x_C - m''|}{|x_C - z_i|} \\
&\leq \frac{r''}{r/(27d)} + \frac{d-k}{k} \frac{4r}{d^2 2^{19} N_C r} < \frac{27dr}{rd^2 2^{20} N_C} + \frac{4dr}{rd^2 2^{20} N_C} \leq \frac{1}{2^{14} d N_C}.
\end{aligned}$$

Hence, there exists an $z \in \mathbb{C}$, with $|z| < \frac{1}{2^{14} d N_C}$, such that $\frac{1}{k} \frac{(x_C - m'')F'(x_C)}{F(x_C)} = 1 + z$. This implies that $\frac{|F'(x_C)|}{|F(x_C)|} \geq \frac{1}{|x_C - m''|} \geq \frac{1}{4r}$, and thus the NewtonTest must reach line 2 as Algorithm 6.3 must return True or Undecided. With $x'_C = x_C - k \cdot \frac{F(x_C)}{F'(x_C)}$, it further follows that

$$\begin{aligned}
|m'' - x'_C| &= |m'' - x_C| \cdot \left| 1 - \frac{1}{\frac{1}{k} \frac{(x_C - m'')F'(x_C)}{F(x_C)}} \right| = |m'' - x_C| \cdot \left| 1 - \frac{1}{1+z} \right| = \left| \frac{z(m'' - x_C)}{1+z} \right| \\
&\leq \frac{4r}{2^{13} d N_C} \leq \frac{r}{2^{11} d N_C} < \frac{2^{\ell_C}}{128 N_C}.
\end{aligned}$$

We therefore obtain

$$|\tilde{x}'_C - m''| \leq |\tilde{x}'_C - x'_C| + |x'_C - m''| \leq \frac{2^{l_C}}{64 N_C} + |x'_C - m''| \leq \frac{2^{l_C}}{64 N_C} + \frac{2^{l_C}}{128 N_C} < \frac{2^{l_C}}{32 N_C}.$$

Since the distance between m'' and any of the roots $z_1, \ldots, z_k$ is also upper bounded by $r'' < 2^{l_C}/(32 N_C)$, we conclude that the disc $\Delta_{2^{l_C}/(16 N_C)}(\tilde{x}'_C)$ contains all roots $z_1, \ldots, z_k$. Hence, it follows that that $\Delta' := \Delta_{2^{l_C}/(8 N_C)}(\tilde{x}'_C)$ is $(1/2, 4/3)$-isolating for the roots $z_1, \ldots, z_k$, and thus $\mathbf{T}_*(\Delta') = k$ must hold according to Lemma 6.15. This shows that we reach line 11 and that the NewtonTest returns Success. □

In essence, the above lemma states that, in case of a well-separated cluster of roots contained in some component C, ℂIsolate performs a quadratic step. That is, it replaces the component C by a component C' of width $w(C') \leq 2^{\ell_C}/N_C \leq w(C)/N_C$, which contains all roots that are contained in C. Now, suppose that there exists a sequence $C_1, \ldots, C_s$ of non-special nodes, with $C_1 \supset \cdots \supset C_s$, such that C_s has much smaller width than C_1. Then, C_1 contains a cluster of nearby roots but no other root of F. We will see that, from a considerably small (compared to the bound in (6.12)) index on, this cluster is also well-separated from the remaining roots (w.r.t. the size of C_i) such that the requirements in the above lemma are fulfilled. As a consequence, only a small number of steps from C_i to C_{i+1} are linear, which in turn implies that the whole sequence has small length. For the proof, we need to consider a sequence $(s_i)_i = (x_i, n_i)_i$, which we define in a rather abstract way. The rationale behind our choice for s_i is that, for all except a small number of indices and a suitable choice for s_i, the sequence $(s_i)_i$ behaves similarly to the sequence $(2^{\ell_{C_i}}, \log\log N_{C_i})_i$. We remark that $(s_i)_i$ has already been introduced in [SM16]. The following lemma is shown there:

Lemma 6.20 ([SM16], Lemma 25). *Let $w, w' \in \mathbb{R}^+$ be two positive reals with $w > w'$, and let $m \in \mathbb{N}_{\geq 1}$ be a positive integer. We recursively define the sequence $(s_i)_{i\in\mathbb{N}_{\geq 1}} := ((x_i, n_i))_{i\in\mathbb{N}_{\geq 1}}$ as follows: Let $s_1 = (x_1, n_1) := (w, m)$, and*

$$s_{i+1} = (x_{i+1}, n_{i+1}) := \begin{cases} (\varepsilon_i \cdot x_i, n_i + 1) \text{ with an } \varepsilon_i \in [0, \frac{1}{N_i}], & \text{if } \frac{x_i}{N_i} \geq w' \\ (\delta_i \cdot x_i, \max(1, n_i - 1)) \text{ with a } \delta_i \in [0, \frac{1}{2}], & \text{if } \frac{x_i}{N_i} < w', \end{cases}$$

where $N_i := 2^{2^{n_i}}$ and $i \geq 1$. Then, the smallest index i_0 with $x_{i_0} \leq w'$ is bounded by $8(n_1 + \log\log\max(4, \frac{w}{w'}))$.

We are now ready to prove the bound on the length of a sequence of non-special nodes.

Lemma 6.21. *Let $\mathcal{P} = (C_1, N_1), \ldots, (C_s, N_s)$, with $C_1 \supset \cdots \supset C_s$, be a sequence of consecutive non-special nodes. Then, we have $s \leq s_{\max}$ with an $s_{\max}$ of size*

$$\begin{aligned} s_{\max} &= O\big(\log d + \log\overline{\log}(w(\mathcal{B})) + \log\overline{\log}(\sigma_F(B^+)^{-1})\big) \\ &= O\big(\log\big(d \cdot \overline{\log}(w(\mathcal{B})) \cdot \overline{\log}(\sigma_F(2\mathcal{B})^{-1})\big)\big). \end{aligned}$$

Proof. We distinguish two cases. First consider the case, where $\mathcal{P}$ is an arbitrary sub-sequence of the unique initial sequence from the child of the root to the parent of the base. If there exists no non-special node in between the root and the base of the tree, there is nothing to prove. Due to Theorem 6.17, part *(e)*, C_s consists of at most $9 \cdot |\mathcal{Z}(2\mathcal{B})|$ squares. It follows that $2^{2s} \leq 9d$ as C_i consists of at least 2^{2i} squares. This yields $s = O(\log d)$.

In the second case, we can assume that each C_i is a successor of the base of the tree. In particular, we have $\mathcal{B} \setminus C_i \neq \emptyset$. W.l.o.g., we may further assume that $z_1, \ldots, z_k$ are the roots contained in the enlarged component C_1^+. Since all C_i are assumed to be non-special, each ${C_i}^+$ contains z_1 to z_k but no other root of F. Let $w_i := w(C_i)$ be the width of the component C_i, $r_i := (3/4) \cdot w_i$ be the radius of the enclosing disc $\Delta_i := \Delta_{C_i}$, and $2^{\ell_i} := 2^{\ell_{C_i}}$ be the width of each of the squares into which C_i decomposes. Notice that, for each index i, the enlarged component C_i^+ is contained in the disc $2\Delta_i$ of radius $(3/2) \cdot w_i$, and thus the disc $2\Delta_s$ of radius $(3/2) \cdot w_s$ contains the roots z_1 to z_k. We now split the sequence $\mathcal{P}$ into three (possibly empty) subsequences $\mathcal{P}_1 = (C_1, N_1), \ldots, (C_{i_1}, N_{i_1})$, $\mathcal{P}_2 = (C_{i_1+1}, N_{i_1+1}), \ldots, (C_{i_2}, N_{i_2})$, and $\mathcal{P}_3 = (C_{i_2+1}, N_{i_2+1}), \ldots, (C_s, N_s)$, where i_1 and i_2 are defined as follows:

- We let i_1 be the first index with $2^{\ell_1} > 2^{3\log d+32} \cdot N_{i_1} \cdot 2^{\ell_{i_1}}$. If there exists no such index, we set $i_1 := s$. Further notice that, for any index i larger than i_1, we also have $2^{\ell_1} > 2^{3\log d+32} \cdot N_i \cdot 2^{\ell_i}$, which follows from induction and the fact that 2^{ℓ_i} and N_i are replaced by $2^{\ell_i}/(2N_i)$ and N_i^2 in a quadratic step.

- We define i_2 to be the first index larger than or equal to i_1 such that the step from i_2 to $i_2 + 1$ is quadratic and $2^{\ell_s} \cdot 2^{3\log d+32} \cdot N_{i_2} \geq 2^{\ell_{i_2}}$. If there exists no such index i_2, we set $i_2 := s$.

From the definition of i_2, it is easy to see that $|\mathcal{P}_3| = O(\log d)$. Namely, if $i_2 = s$, there is nothing to prove, hence we may assume that the step from i_2 to $i_2 + 1$ is quadratic and $2^{\ell_s} \geq 2^{-3\log d-32} \cdot N_{i_2}^{-1} \cdot 2^{\ell_{i_2}} = 2^{-3\log d-31} \cdot 2^{\ell_{i_2+1}}$. Hence, we conclude that $s - (i_2+1) \leq 3\log d + 31$ as ℓ_i is reduced by at least 1 in each step.

Let us now consider an arbitrary index i from the sequence $\mathcal{P}_2$. The distance from an arbitrary point in C_i^+ to the boundary of C_1^+ is at least $2^{\ell_1-1} \geq 2^{3\log d+31} \cdot N_i \cdot 2^{\ell_i} > 2^{2\log d+20} \cdot N_i \cdot r_i$, where the latter inequality follows from $r_i = (3/4)w_i \leq (3/4) \cdot 9d \cdot 2^{\ell_i}$. Since C_1^+ contains only the roots $z_1, \ldots, z_k$, this implies that the distance from an arbitrary point in C_i^+ to an arbitrary root $z_{k+1}, \ldots, z_d$ is larger than $2^{2\log d+20} \cdot N_i \cdot r_i$. Hence, the second requirement from Lemma 6.19 is fulfilled for each component C_i with $i \geq i_1$. Now, suppose that $2^{\ell_s} \cdot 2^{3\log d+32} \cdot N_i < 2^{\ell_i}$, then the roots z_1 to z_k are contained in a disc of radius $\frac{3}{2} \cdot 9d \cdot 2^{\ell_s} < 2^{-20-\log d} \cdot N_i^{-1} \cdot r_i$, and thus also the first requirement from Lemma 6.19 is fulfilled. Hence, from the definition of i_2, we conclude that the algorithm performs a quadratic step if and only if $2^{\ell_s} \cdot 2^{3\log d+32} \cdot N_i < 2^{\ell_i}$. We now define the sequence $s_i := (2^{\ell_i}, \log\log N_i)$, where i runs from i_1 to the first index, denoted i_1', for which $2^{\ell_{i_1'}} < 2^{\ell_s} \cdot 2^{-3\log d-32}$. Then, according to Lemma 6.20, it holds that $i_1' - i_1 \leq 8(m + \log\log\max(4, w/w'))$, with $w := 2^{\ell_{i_1}}$, $m := \log\log N_{i_1}$, and $w' := 2^{\ell_s} \cdot 2^{3\log d+32}$. Theorem 6.17 *(g)* yields that $m = O(\log\overline{\log}(w(\mathcal{B})) + \log\overline{\log}(\sigma_F(2\mathcal{B})^{-1})$. Hence, since $i_2 - i_1' \leq 3\log d + 32$, we conclude that $i_2 - i_1 \leq O(\log d + \log\overline{\log}(w(\mathcal{B})) + \log\overline{\log}(\sigma_F(2\mathcal{B})^{-1}))$.

It remains to show that the latter bound also applies to i_1. From the upper bound on N_i, it follows that there exists an $m_{\max}$ of size $O(\log d + \log\overline{\log}(w(\mathcal{B})) + \log\overline{\log}(\sigma_F(2\mathcal{B})^{-1}))$ such that each sequence of consecutive quadratic steps has length bounded by $m_{\max}$, and such that after $m_{\max}$ consecutive linear steps, the number N_i drops to 4. Since the number ℓ_i decreases by at least 1 in each step, there exists an index i' of size $O(\log d)$ such that $2^{\ell_{i'}} \cdot 2^{3\log d+34} < 2^{\ell_1}$. Now, if the sequence $C_{i'}, C_{i'+1}, \ldots$ starts with $m_{\max}$ or more consecutive linear steps, we must have $N_{i'+m_{\max}} = 4$, and thus $2^{\ell_{i'+m_{\max}}} \cdot 2^{3\log d+32} N_{i'+m_{\max}} < 2^{\ell_1}$. Hence, we conclude that $i_1 \leq i' + m_{\max}$ in this case. Otherwise, there must exist an index i'', with $i' \leq i'' < i' + m_{\max}$, such that the step from i'' to $i''+1$ is quadratic, whereas the step from $i''+2$ is linear. Then, it holds that

$$N_{i''+2} = \sqrt{N_{i''+1}} = N_{i''} \text{ and } 2^{\ell_{i''+2}} \leq 2^{\ell_{i''+1}} = \frac{2^{\ell_{i''}}}{2N_{i''}} < 2^{-3\log d-32}\frac{2^{\ell_1}}{N_{i''}},$$

which implies that $i_1 \leq i''+2 \leq i' + m_{\max} + 1 = O(\log d + \log\overline{\log}(w(\mathcal{B})) + \log\overline{\log}(\sigma_F(2\mathcal{B})^{-1}))$. Hence, the claimed bound on i_1 follows. □

We can now state the first main result of this section, which immediately follows from the above bound on $s_{\max}$ and the fact that there exists at most $2 \cdot (|\mathcal{Z}(2\mathcal{B})| + 1)$ special nodes:

Theorem 6.22. *The subdivision tree $\mathcal{T}$ induced by* $\mathbb{C}$I<span style="font-variant:small-caps">solate</span> *has size*

$$\begin{aligned} |\mathcal{T}| &\leq 2 \cdot (|\mathcal{Z}(2\mathcal{B})| + 1) \cdot s_{\max} \\ &= O\left(|\mathcal{Z}(2\mathcal{B})| \cdot \log\left(d \cdot \overline{\log}(w(\mathcal{B})) \cdot \overline{\log}(\sigma_F(2\mathcal{B})^{-1})\right)\right). \end{aligned} \tag{6.13}$$

If $\mathcal{B}$ contains all complex roots of F and $\overline{\log}(w(\mathcal{B})) = O(\Gamma_F + \log d)$, then the above bound writes as

$$O\big(d \cdot \log\left(d \cdot \Gamma_F \cdot \overline{\log}(\sigma_F^{-1})\right)\big). \tag{6.14}$$

Recall from Section 6.2 that we can compute a square $\mathcal{B}$ that contains all roots of F with $\tilde{O}(d^2\Gamma_F)$ bit operations. We proceed by giving simpler bounds for the special case, where the input polynomial has integer coefficients. Suppose that $f(x) \in \mathbb{Z}[x]$ has integer coefficients of bit-size less than τ. We first divide f by its leading coefficient $\mathrm{LC}(f)$ to obtain the polynomial $F := f/\mathrm{LC}(f)$, which meets the requirement from (6.1) on a_d. Then, we have $\Gamma_F = O(\tau)$ and $\sigma_F = 2^{-O(d(\log d+\tau))}$; e.g. see [Yap00] for a proof of the latter bound. Hence, we obtain the following corollary.

Corollary 6.23. *Let f be a polynomial of degree d with integer coefficients of bit-size less than τ, let $F := f/\mathrm{LC}(f)$, and let $\mathcal{B}$ be a square of width $2^{O(\Gamma_F+\log d)}$. Then, the algorithm* $\mathbb{C}$Isolate *(with input F and $\mathcal{B}$) uses*

$$O(|\mathcal{Z}(2\mathcal{B})|\cdot\log(d\tau)) = O(d\cdot\log(d\tau))$$

iterations to isolate all roots of F that are contained in $\mathcal{B}$.

The above results show that $\mathbb{C}$Isolate performs near-optimal with respect to the number of components that are produced by the algorithm. In addition, since each component consists of at most $9d$ squares, we immediately obtain an upper bound for the total number of squares produced by the algorithm that exceeds the bound from (6.14) by a factor of d. Indeed, we will see that the actual number of squares is considerably smaller, that is, of size $O(|\mathcal{Z}(2\mathcal{B})|\cdot s_{\max}\cdot\log d)$, which exceeds the bound in (6.14) only by a factor $\log d$. For the proof, we consider two mappings ϕ and ψ, where ϕ maps a component $C = \{B_1,\dots,B_{s_C}\}$ to a root $z_i \in C^+$, and ψ maps a square B_j to a root $z_i \in 4B_j \cap C^+$. The claimed bound for the total number of squares then follows from the fact that we can define ϕ and ψ in a way such that the pre-image of an arbitrary root $z_i \in 2\mathcal{B}$ (under each of the two mappings) has size $O(s_{\max}\cdot\log n)$. The rest of this section is dedicated to the definitions of ϕ and ψ and the proof of the latter claim. In what follows, we may assume that $2\mathcal{B}$ contains at least one root as, otherwise, all four sub-squares of $\mathcal{B}$ are already discarded in the first iteration of the preprocessing phase.

Definition 6.24. *For a root $\xi \in 2\mathcal{B}$, we define the* canonical path *$\mathcal{P}_\xi$ of ξ as the unique path in the subdivision tree $\mathcal{T}_\mathcal{B}$ that consists of all nodes C with $\xi \in C^+$.*

Notice that the canonical path is well-defined as, for any two nodes C_1 and C_2, either C_1^+ and C_2^+ are disjoint or one of the two components contains the other one. We can now define the maps ϕ and ψ:

Definition 6.25 (Maps ϕ,ψ). *Let $C = \{B_1,\dots,B_{s_C}\}$ be a node in the subdivision tree $\mathcal{T}_\mathcal{B}$, and let $B := B_j$ be an arbitrary square in C. Then, we define maps ϕ and ψ as follows:*

(ϕ) *Starting at C, we descend in the subdivision tree as follows: If the current node D is a non-terminal special node, we go to the child E that minimizes $|\mathcal{Z}(E^+)|$. If D is terminal, we stop. If D is non-special, then there is a unique child of D to proceed with. Proceeding this way, the number $|\mathcal{Z}(D^+)|$ is at least halved in each non-terminal special node D, except for the base node. Hence, since any sequence of consecutive non-special nodes has length at most $s_{\max}$, it follows that after at most $s_{\max}\cdot(\log\lceil(|\mathcal{Z}(C^+)|)\rceil+1) \le s_{\max}\cdot(\log d+2)$ many steps we reach a terminal node F. We define $\phi(C)$ to be an arbitrary root contained in $\mathcal{Z}(F^+)$.*

(ψ) *According to part* (d) *of Theorem 6.17, the enlarged square $2B$ contains at least one root ξ. Now, consider the unique maximal subpath $P'_\xi = C_1, C_2, \dots, C_s$ of the canonical path P_ξ that starts at $C_1 := C$. If $s \le \lceil\log(18d)\rceil$, we define $\psi(B) := \xi$. Otherwise, consider the component $C' := C_{\lceil\log(18d)\rceil}$ and define $\psi(B) := \phi(C')$.*

It is clear from the above definition that $\phi(C)$ is contained within C^+ as each root contained in the enlarged component F^+ corresponding to the terminal node F is also contained in C^+. It remains to show that $\psi(B) \in 4B \cap C^+$. If the length of the sub-path P'_ξ is $\lceil\log(18d)\rceil$ or less, then $\psi(B) = \xi \in 2B$, hence, there is nothing to prove. Otherwise, the squares in C' have width less than $\frac{w(B)}{18d}$. Since C' can contain at most $9d$ squares, we conclude that $w(C') < \frac{w(B)}{2}$, and since ξ is contained in B^+ as well as in $(C')^+$, we conclude that $(C')^+ \subset 4B$, and thus $\psi(B) = \phi(C') \in 4B \cap (C')^+ \subset 4B \cap C^+$.

Now, consider the canonical path $P_\xi = C_1,\dots,C_s$, with $C_1 := \mathcal{B}$, of an arbitrary root $\xi \in 2\mathcal{B}$. Then, a component C can only map to ξ via ϕ if $C = C_i$ for some i with $s - i \le$

$s_{\max} \cdot (\log|\mathcal{Z}(2\mathcal{B})|+1)$. Hence, the pre-image of ξ has size $O(s_{\max} \cdot \overline{\log}\,|\mathcal{Z}(2\mathcal{B})|)$. For the map ψ, notice that a square B can only map to ξ if B is contained in a component $C = C_i$ for some i with $s - i = s_{\max} \cdot (\log|\mathcal{Z}(2\mathcal{B})|+1) + \lceil\log(18d)\rceil$. Since, for each component C_i, there exist at most a constant number of squares $B' \in C_i$ with $\xi \in 4B'$, we conclude that the pre-image of ξ under ψ is also of size $O(s_{\max} \cdot \overline{\log}\,\mathcal{Z}(2\mathcal{B}))$. Hence, the total number of squares produced by $\mathbb{C}$Isolate is bounded by $O(|\mathcal{Z}(2\mathcal{B})| \cdot s_{\max} \cdot \overline{\log}\,|\mathcal{Z}(2\mathcal{B})|)$. We summarize:

Theorem 6.26. *Let $\xi \in 2\mathcal{B}$ be a root of F contained in the enlarged square $2\mathcal{B}$. Then, with mappings ϕ and ψ as defined in Definition 6.25, the pre-image of ξ under each of the two mappings has size $O(s_{\max} \cdot \overline{\log}\,|\mathcal{Z}(2\mathcal{B})|)$. The total number of squares produced by the algorithm $\mathbb{C}$Isolate is bounded by*

$$O(s_{\max} \cdot |\mathcal{Z}(2\mathcal{B})| \cdot \overline{\log}\,|\mathcal{Z}(2\mathcal{B})|) = \tilde{O}\left(d \cdot \log\left(\overline{\log}(w(\mathcal{B})) \cdot \overline{\log}(\sigma_F(2\mathcal{B})^{-1})\right)\right)$$

We can also state a corresponding result for polynomials with integer coefficients:

Corollary 6.27. *Let $f \in \mathbb{Z}[x]$ and $F := f/\mathrm{LC}(f)$ be polynomials and $\mathcal{B}$ be a square as in Corollary 6.23. Then, for isolating all roots of f contained in $\mathcal{B}$, the algorithm $\mathbb{C}$Isolate (with input F and $\mathcal{B}$) produces a number of squares bounded by*

$$O(|\mathcal{Z}(2\mathcal{B})| \cdot \log(d\tau) \cdot \log d) = O(d\log^2(d\tau)).$$

6.6.2 Bit Complexity

Weakly centered components. For our analysis, we need to introduce the notion of a square or component being *weakly centered* and *centered*:

Definition 6.28. *We say that a square B of width $w := w(B)$ is* weakly centered, *if* $\min\{|z| : z \in B\} \le 4 \cdot w$. *Similarly, we say that a square is* centered *if* $\min\{|z| : z \in B\} \le w/4$. *In addition, we define a (weakly) centered component to be a component that contains a (weakly) centered square.*

Notice that the child of a component that is *not* weakly centered can never become weakly centered. Hence, it follows that the set of weakly centered components forms a subtree $\mathcal{T}_{\text{wcent}}$ of the subdivision tree $\mathcal{T}$ that is either empty or contains the root component $\mathcal{B}$; see Figure 6.4 for an illustration.

Moreover, let C and C' be siblings in $\mathcal{T}_{\text{wcent}}$ and let w and w' be the sizes of the squares in the components C and C', respectively. We already argued that the distance between C and C' is at least $\max\{w, w'\}$. W.l.o.g. let $\min\{|z| : z \in C\} \le \min\{|z| : z \in C'\}$, then the distance of C' to the origin is at least $w'/2$, and thus C' is not centered. We further conclude that a descendant of depth 3 of C' is not weakly centered because the width of the squares in the component is at least halved in each step. It follows that each path in $\mathcal{T}_{\text{wcent}}$ consisting of only weakly centered components has length at most 3. From this observation, we conclude that the subtree $\mathcal{T}_{\text{wcent}}$ has a very special structure. Namely, it consists of only one (possibly empty) *central path* $P = C_1, \dots, C_\ell$ of all centered components, to which some trees of depth at most 4 of weakly centered components are attached, see Figure 6.4. Since there can only be a constant number of disjoint not weakly centered squares of the same size, it further holds that the degree of each node is bounded by a constant. Hence, each of the attached trees has constant size, and each node in the tree contains at most a constant number of weakly centered squares.

Notice that not weakly centered components $C \in \mathcal{T} \setminus \mathcal{T}_{\text{wcent}}$ have the crucial property that any two points in C^+ have absolute values of comparable size; see Lemma 6.29. This is not true in general for (weakly) centered components as the size of C might be very large whereas the distance from C^+ to the origin is small.

Lemma 6.29. *If B is a not weakly centered square, it holds that*

$$\max_{z\in 4B} \overline{\log}(z) \le 5 + \min_{z\in 4B} \overline{\log}\, z.$$

Moreover, if C is a not weakly centered component, then it holds that

$$\max_{z\in C^+} \overline{\log}(z) \le \log(64d) + \min_{z\in C^+} \overline{\log}\, z.$$

Proof. We first prove the claim for a not weakly centered square B. Let $\underline{z} := \operatorname{argmin}\{|z| : z \in 4B\}$ and $\overline{z} := \operatorname{argmax}\{|z| : z \in 4B\}$. By definition the distance of B to the origin is at least $4w$, where w denotes the size of B. Moreover, with $x := \operatorname{argmin}\{|z| : z \in B\}$, we get $|\underline{z}| \ge |x| - |x - \underline{z}| \ge 4w - \sqrt{12.5}w \ge w/4$. Thus

$$|\overline{z}| - |\underline{z}| \le |\overline{z} - \underline{z}| \le 4\sqrt{2}w \le 16\sqrt{2} \cdot |\underline{z}|,$$

and the statement follows.

It remains to prove the claim for a not weakly centered component. Let $\underline{z} := \operatorname{argmin}\{|z| : z \in C^+\}$, $\overline{z} := \operatorname{argmax}\{|z| : z \in C^+\}$, and let $B \subset C$ be a square in C such that $\underline{z} \in 4B$. Then, as above, it follows that $\underline{z} \ge w/4$ and since C contains at most $9n$ squares it holds that

$$|\overline{z}| - |\underline{z}| \le |\overline{z} - \underline{z}| \le \sqrt{2} \cdot (9dw + w) \le \sqrt{2} \cdot 10dw \le 57d|\underline{z}|,$$

and thus $\max_{z\in C^+} \overline{\log}(z) \le \log(64d) + \min_{z\in C^+} \overline{\log}\, z$. □

Mappings $\hat{\phi}$ and $\hat{\psi}$. In the previous section, we introduced mappings ϕ and ψ that map components C and squares B to roots contained in C^+ and $4B \cap C^+$, respectively, such that the pre-image (under each of the two mappings) of each root has size at most $O(s_{\max} \cdot \overline{\log}\,|\mathcal{Z}(2\mathcal{B})|) = O(s_{\max} \cdot \log d)$, with

$$s_{\max} = O(\log(d\, \overline{\log}(w(\mathcal{B}))\, \overline{\log}(\sigma_F(2\mathcal{B})^{-1})))$$

as shown in Lemma 6.21. The crucial idea in our analysis is to bound the cost for processing a certain component C (square B) in terms of values that depend only on the root $\phi(C)$ ($\psi(B)$), such as its absolute value, its separation, or the absolute value of the derivative F' at the root; see Lemma 6.34. Following this approach, each root in $2\mathcal{B}$ is "charged" only a small (i.e. logarithmic in the "common" parameters) number of times, and thus we can profit from amortization when summing the cost over all components (squares). For each not weakly centered component C, the width of C and the absolute value of any point in C is upper bounded by $64d \cdot |\phi(C)|$, which allows us to bound each occurring term $\overline{\log}\, w(C)$ by $O(\log d + \overline{\log}\,|\phi(C)|)$. However, for weakly centered components (squares), this does not hold in general, and thus some extra treatment is required. For this, we will introduce slightly modified mappings $\hat{\phi}$ and $\hat{\psi}$ that coincide with ϕ and ψ on all not weakly centered components and squares, respectively, but map a weakly centered component (square) to a root of absolute value that is comparable to the size of the component (square). In the next step, we will show that this can be done in a way such that the pre-image of each root is still of logarithmic size. We give details:

Let $i_1, \dots, i_s$, with $1 \le i_1 < i_2 < \dots < i_s \le \ell - 1$, be the indices of components in the central path $P = C_1, \dots, C_\ell$ such that $\mathcal{Z}(C_{i_j}^+) \supsetneq \mathcal{Z}(C_{i_j+1}^+)$ for $j = 1, \dots, s$. That is, C_{i_j} are the weakly centered components that are also special according to Definition 6.18. In addition, we say that $z_0 := \max\{|x| : x \in 2\mathcal{B}\}$ is the *pseudo-root of* $2\mathcal{B}$, and define $\mathcal{Z}^+(2\mathcal{B}) := \mathcal{Z}(2\mathcal{B}) \cup \{z_0\}$ as the set consisting of all (pseudo-) roots in $2\mathcal{B}$.

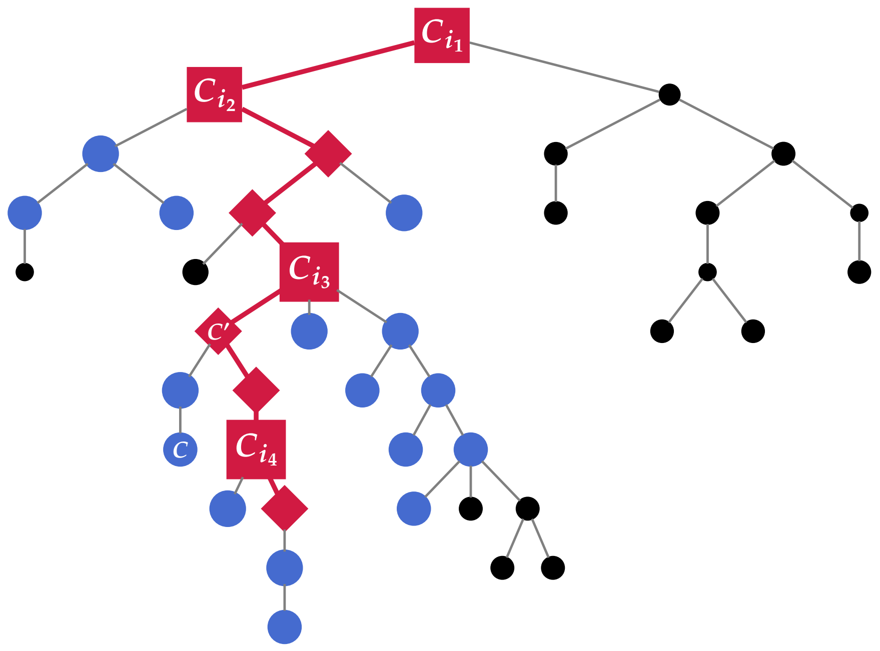

FIGURE 6.4: A possible subdivision tree of ℂISOLATE on an arbitrary input square. Red nodes (squares and diamonds) correspond to centered components, blue nodes (large circles) to weakly centered components and black nodes (small circles) to not weakly centered components. The subtree $\mathcal{T}_{\text{wcent}}$ consists of all red and blue nodes. Note that $\mathcal{T}_{\text{wcent}}$ consists of one path of centered components (marked by the red edges) and trees of depth at most 4 attached to it (consisting of blue nodes). The rectangular red nodes correspond to those components $C_{i_1}, \dots, C_{i_s}$ on the central path that are split nodes according to Definition 6.18, that is, components for which $\mathcal{Z}(C^+_{i_j}) \supsetneq \mathcal{Z}(C^+_{i_j+1})$. The difference of these two sets contains the roots that we map the (weakly) centered components to. For instance, the centered component C' is mapped to a root $\hat{\phi}(C')$ that is contained in $\mathcal{Z}(C^+_{i_3}) \setminus \mathcal{Z}(C^+_{i_3+1})$. All weakly centered components that have C' as their first centered predecessor, as for example C, are mapped to the same root $\phi(C)$.

Definition 6.30 (Maps $\hat{\phi}$ and $\hat{\psi}$)**.** *Let C be a component and $B \subset C$ be a square contained in C.*

1. *If the component C is not weakly centered, we define $\hat{\phi}(C) := \phi(C)$.*
2. *If the component C is weakly centered, let $C_i \in P$ be the first centered predecessor of C in $\mathcal{T}_{\text{wcent}}$. If there exists no such C_i or if $i \in [1, i_1]$, we define $\hat{\phi}(C) = z_0$. For $i \in (i_2, \ell]$, let j be maximal with $i_j < i$. Then, there exists a root $\xi \in \mathcal{Z}(C^+_{i_j}) \setminus \mathcal{Z}(C^+_{i_j+1})$. We define $\hat{\phi}(C) := \xi$.*
3. *If B is weakly centered, we define $\hat{\psi}(B) := \hat{\phi}(C)$. Otherwise, we define $\hat{\psi}(B) := \psi(B)$.*

We derive the first crucial property of the mappings $\hat{\phi}$ and $\hat{\psi}$:

Lemma 6.31. *It holds that*

$$\max_{z\in C^+} \overline{\log} z \le \log(64d) + \overline{\log}\, \hat{\phi}(C) \ \ \textit{and} \ \ \max_{z\in 4B} \overline{\log} z \le \log(64d) + \overline{\log}\, \hat{\psi}(B) \qquad (6.15)$$

for all components C and squares B.

Proof. For a not weakly centered component C (square $B \subset C'$), this follows directly from Lemma 6.29 and the fact that $\hat{\phi}(C) = \phi(C)$ ($\hat{\psi}(B) = \psi(B)$) and that $\phi(C) \in C^+$ ($\psi(B) \in 4B \cap C^+$).

For a centered component C, we either have $\hat{\phi}(C) = z_0$ or $\hat{\phi}(C) \notin C^+$ due to the definition of $\hat{\phi}$. The first case is trivial, hence, we may assume that $\hat{\phi}(C) \notin C^+$. Since C is centered, it contains a centered square B, and thus the distance of B to the origin is at most $w/4$. It follows that C^+ contains the disc of radius $w/4$ around the origin, hence $|\hat{\phi}(C)| \geq w/4$. Since the distance between any two points in C^+ is upper bounded by $10\sqrt{2}dw$, we conclude that $|z| \leq w/4 + 10\sqrt{2}dw \leq (1+40\sqrt{2}) \cdot |\hat{\phi}(C)| \leq 64d \cdot |\hat{\phi}(C)|$ for all $z \in C^+$. The same argument further shows that $|z| \leq 64d \cdot |\hat{\psi}(B)|$ for all centered squares and all $z \in 4B$.

It remains to show the claim for a weakly centered component C (square B) that is not centered. In this case, we either have $\hat{\psi}(C) = z_0$ or $\hat{\psi}(C) = \hat{\phi}(C')$, where C' is a centered component on the central path that contains C; see Definition 6.30. In the first case, there is nothing to prove. In the second case, we have have already shown that Inequality (6.15) holds for C', hence, it must hold for C as well. The same argument also applies to squares B in C that are weakly centered but not centered. □

Notice that, for a centered component C (square B), the image of the corresponding mapping $\hat{\phi}$ ($\hat{\psi}$) may no longer be contained in the enlarged component C^+ (enlarged square $4B$), as it is the case for the mappings ϕ and ψ. However, it still holds that the pre-image of its (pseudo-) root under each of the two mappings $\hat{\phi}$ and $\hat{\psi}$ is of small size:

Lemma 6.32. *Let $\xi \in \mathcal{Z}^+(2\mathcal{B})$. Then, the pre-image of ξ under $\hat{\phi}$ and $\hat{\psi}$ has size at most $O(s_{\max} \cdot \overline{\log}\,|\mathcal{Z}(2\mathcal{B})|) = O(s_{\max} \cdot \log d)$.*

Proof. Since $\hat{\phi}$ coincides with ϕ on all components $C \notin \mathcal{T}_{\text{wcent}}$, it suffices to show the claim for the restriction $\hat{\phi}|_{\mathcal{T}_{\text{wcent}}}$ of $\hat{\phi}$ to the components $C \in \mathcal{T}_{\text{wcent}}$ that are weakly centered. Let $\xi = \hat{\psi}(C)$, with $C \in \mathcal{T}_{\text{wcent}}$, be an arbitrary root contained in the image of $\hat{\phi}|_{\mathcal{T}_{\text{wcent}}}$, and let C_i be the first predecessor of C that is central and located on the central path. Then, there exists a $j \in \{1,\dots,\ell-1\}$ with $i_j < i \leq i_{j+1}$, and we have $\xi \in \mathcal{Z}(C^+_{i_j}) \setminus \mathcal{Z}(C^+_{i_{j+1}})$. In addition, C is connected with C_i via a path of constant length as the distance from C to the central path on $\mathcal{T}_{\text{wcent}}$ is bounded by a constant and there cannot be more than 3 consecutive components that are weakly centered but not centered. Since there exists at most $s_{\max}$ components on the central path between C_{i_j} and $C_{i_{j+1}}$, it follows that the number of components $C \in \mathcal{T}_{\text{wcent}}$ that are mapped to ξ is bounded by $O(s_{\max})$. The same argument applies to the special case, where C is mapped to the pseudo-root z_0. Also, from the same argument and the definition of $\hat{\psi}$, it further follows that there can be at most $O(s_{\max})$ many weakly centered squares that are mapped to the same (pseudo-) root, since each component contains at most constantly many weakly centered squares. □

Bounding the Bit Complexity. We can now start with the bit complexity analysis of ℂIsolate. Let $C = \{B_1,\dots,B_{s_C}\}$ be any component produced by the algorithm. When processing C, our algorithm calls the $\mathbf{T}_*$-test in up to three steps. More specifically, in line 9 of ℂIsolate the $\mathbf{T}_*(2\Delta_C)$ and the $\mathbf{T}_*(4\Delta_C)$-test are called. In the NewtonTest, the $\mathbf{T}_*(\Delta')$-test is called, with Δ' as defined in line 4 in NewtonTest. Finally, in Bisection, the $\mathbf{T}_*(\Delta_{B'})$-test is called for each of the 4 sub-squares B' into which each square B_i of C is decomposed. Our goal is to provide bounds for the cost of each of these calls. For this, we mainly use Lemma 6.16, which provides a bound on the cost for calling $\mathbf{T}_*(\Delta)$ that depends on the degree of F, the value τ_F, the size of the radius and the center of Δ, and the maximal absolute value that F takes on the disc Δ. Under the assumption that Δ has non-empty intersection with C, we may reformulate the latter value in terms of parameters (such as the absolute value, the separation, etc.) that depend on an arbitrary root contained in C^+.

Lemma 6.33. *Let C be a component, and let $\Delta := \Delta_r(m)$ be a disc that has non-empty intersection with C. If $\mathcal{Z}(C^+) \geq 1$, then it holds that $\sigma_F(z_i) < d \cdot 2^{\ell_C+6}$ and*

$$\max_{z\in\Delta} |F(z)| > 2^{-16d} \cdot \sigma_F(z_i) \cdot |F'(z_i)| \cdot (|\mathcal{Z}(C^+)| \cdot M(\lambda))^{-2d},$$

where z_i is an arbitrary root of F contained in C^+ and $\lambda := \frac{2^{\ell_C}}{r}$ the ratio of the size of a square in C and the radius of Δ. If $\mathcal{Z}(C^+) = 0$, then C is the component consisting of the single input square $\mathcal{B}$, and it holds that $\max_{z\in\Delta} |F(z)| > (2w(\mathcal{B}))^{-d-2}$.

Proof. Notice that $|\mathcal{Z}(C^+)| = 0$ is only possible if $C = \mathcal{B}$ as C^+ contains at least one root for each component C that is not equal to the single input square $\mathcal{B}$. Hence, each point $z \in \Delta \cap C$ has distance at least $w(B)/2$ to each of the roots of F, and thus $|F(z)| \geq (2w(\mathcal{B}))^{-d-2}$.

In what follows, we now assume that C^+ contains at least one root ξ. Let us first bound the separation of ξ: If there exists another root $\xi' \in C^+$, then we must have $\sigma_F(\xi) \leq |\xi - \xi'| < (9 \cdot |\mathcal{Z}(C^+)| + 1) \cdot \frac{3}{2} \cdot 2^{\ell_C} < d \cdot 2^{\ell_C+5}$, where we used that each component C consists of at most $9 \cdot |\mathcal{Z}(C^+)|$ squares, each of size 2^{ℓ_C}. Now, let $|\mathcal{Z}(C^+)| = 1$, and let C' be the direct ancestor of C. When processing C', the NewtonTest failed as its success would imply that $k := |\mathcal{Z}((C')^+)| = |\mathcal{Z}(C^+)| \geq 2$. In addition, since C' is non-terminal, the disc $8\Delta_{C'}$ must contain at least two roots as otherwise $\mathbf{T}_*(2\Delta_{C'})$ as well as $\mathbf{T}_*(4\Delta_{C'})$ would return 1. Hence, we have $\sigma_F(\xi) < d \cdot 2^{\ell_{C'}+5} = d \cdot 2^{\ell_C+6}$. Thus, for any root $\xi \in C^+$, we have $\sigma_F(\xi) < d2^{\ell_C+6}$.

In the next step, we show that there exists an $m' \in \Delta \cap C^+$ whose distance to C is at most 2^{ℓ_C-2} and whose distance to any root of F is at least $\min(2^{\ell_C-2}, r)/(2\sqrt{d})$. Namely, due to our assumption, there exists a point $p \in \Delta \cap C$. Then, the two discs $\Delta' := \Delta_{2^{\ell_C-2}}(p)$ and Δ share an area of size larger than $\min(2^{\ell_C-2}, r)^2$, and thus there must exist an $m' \in \Delta' \cap \Delta \subset C^+$ whose distance to any root of F is lower bounded by $(\min(2^{\ell_C-2}, r)^2/(\pi \cdot d))^{1/2}$.

Now, let $z_i \in C^+$ be an arbitrary but fixed root of F. If z_j is a root not contained in C^+, then $|m' - z_j| > 2^{\ell_C-2}$, and

$$\frac{|z_i - z_j|}{|m' - z_j|} \leq \frac{|z_i - m'| + |m' - z_j|}{|m' - z_j|} \leq 1 + \frac{|z_i - m'|}{|m' - z_j|} < 1 + \frac{9d2^{\ell_C+1}}{2^{\ell_C-2}} < 2^7 d. \tag{6.16}$$

If z_j is a root in C^+, then

$$\frac{|z_i - z_j|}{|m' - z_j|} \leq \frac{9d2^{\ell_C+1}}{\delta} = \frac{18 \cdot d^{3/2} \cdot 2^{\ell_C+1}}{\min(2^{\ell_C-2}, r)} \leq 2^8 \cdot d^{3/2} \cdot M(\lambda). \tag{6.17}$$

Hence, using both (6.16) and (6.17), we get

$$\begin{aligned} |F(m')| &= |a_d| \cdot \prod_{j\in[d]} |m' - z_j| = |F'(z_i)| \cdot |m' - z_i| \cdot \prod_{j\neq i} \frac{|m' - z_j|}{|z_j - z_i|} \\ &\geq |F'(z_i)| \cdot |m' - z_i| \cdot (2^8 d^{3/2} M(\lambda))^{-|\mathcal{Z}(C^+)|} \cdot (2^7 d)^{-(d-|\mathcal{Z}(C^+)|)}. \end{aligned}$$

Moreover, using that $\sigma_F(z_i) < 2^{\ell_C+6} \cdot d$, yields the result

$$\begin{aligned} |F(m')| &> |F'(z_i)| \cdot \frac{\min(2^{\ell_C-2}, r)}{2\sqrt{d}} \cdot (2^8 d^{3/2} M(\lambda))^{-d} \\ &> |F'(z_i)| \cdot \frac{\sigma_F(z_i)}{2^9 n^{3/2} M(\lambda)} \cdot (2^8 d^{3/2} M(\lambda))^{-d} \\ &> 2^{-16d} \cdot \sigma_F(z_i) \cdot |F'(z_i)| \cdot (d \cdot M(\lambda))^{-2d}. \end{aligned}$$

□

We can now bound the cost for processing a component C:

Lemma 6.34. *When processing a component $C = \{B_1, \dots, B_{s_C}\}$ with $|\mathcal{Z}(C^+)| \geq 1$, the cost for all steps outside the* NewtonTest *are bounded by*

$$\begin{aligned}&\tilde{O}(d \cdot (d \,\overline{\log}\, \hat{\phi}(C) + \overline{\log}(\sigma_F(\phi(C))^{-1}) + \overline{\log}(F'(\phi(C))^{-1}))) + \\ &\qquad\qquad \tilde{O}(d \cdot (\sum_{i=1}^{s_C} d \,\overline{\log}\, \hat{\psi}(B_i) + \tau_F + \overline{\log}\, \sigma_F(\psi(B_i))^{-1} + \overline{\log}\, F'(\psi(B_i))^{-1}))\end{aligned} \tag{6.18}$$

bit operations. The cost for the NewtonTest *is bounded by*

$$\tilde{O}(d(d \,\overline{\log}\, \hat{\phi}(C) + \overline{\log}\, \sigma_F(\phi(C))^{-1} + \overline{\log}\, F'(\phi(C))^{-1} + |\mathcal{Z}(C^+)| \cdot \log N_C)) \tag{6.19}$$

bit operations. If C^+ contains no root, then C is the component consisting of the single input square $\mathcal{B}$, and the total cost for processing C is bounded by $\tilde{O}(d(\tau_F + d \,\overline{\log}(w(\mathcal{B}), w(\mathcal{B})^{-1})))$ bit operations.

Proof. We start with the special case, where C^+ contains no root. Notice that this is only possible if $C = \mathcal{B}$ and $2\mathcal{B}$ contains no root. Hence, in this case, the algorithm performs four $\mathbf{T}_*(\Delta)$-tests in the preprocessing phase and then discards $\mathcal{B}$. Due to Lemma 6.16 and Lemma 6.33, the cost for each of these tests is bounded by $\tilde{O}(d(\tau_F + d \,\overline{\log}(w(\mathcal{B}), w(\mathcal{B})^{-1}))$ bit operations. Hence, in what follows, we may assume that C^+ contains at least one root. We first estimate the cost for calling $\mathbf{T}_*$ on a disc $\Delta = \Delta_r(m)$, where $\Delta = 2\Delta_C$, $\Delta = 4\Delta_C$, or $\Delta = \Delta_{B'}$, where B' is one of the four sub-squares into which a square B_i is decomposed. If $\Delta = 2\Delta_C$ or $\Delta = 4\Delta_C$, then $\overline{\log}(m, r) = O(\log d + \overline{\log}\, \hat{\phi}(C))$, and thus the cost for the corresponding tests is bounded by

$$\tilde{O}(d \cdot (d \,\overline{\log}\, \hat{\phi}(C) + \tau_F + \overline{\log}\, \sigma_F(\phi(C))^{-1} + \overline{\log}\, F'(\phi(C))^{-1}))$$

bit operations, where we again use Lemma 6.16 and Lemma 6.33. For $\Delta = \Delta_{B'}$, we have $\overline{\log}(m, r) = O(\log d + \overline{\log}\, \hat{\psi}(C))$, and thus Lemma 6.16 and Lemma 6.33 yields the bound

$$\tilde{O}(d \cdot (d \,\overline{\log}\, \hat{\psi}(B_i) + \tau_F + \overline{\log}\, \sigma_F(\psi(B_i))^{-1} + \overline{\log}\, F'(\psi(B_i))^{-1}))$$

for processing each of the four sub-squares $B' \subset B_i$ into which B_i is decomposed. Hence, the bound in (6.18) follows. We now consider the NewtonTest: In line 2 of the NewtonTest, we have to compute an approximation $\tilde{x}'_C$ of the Newton iterate x'_C such that $|\tilde{x}'_C - x'_C| < 2^{\ell_C}/(64N_C)$. For this, we choose a point $x_C \in \mathcal{B} \backslash C$ in line 12 of $\mathbb{C}$Isolate, whose distance to C is $2^{\ell_C - 1}$ and whose distance to the boundary of $\mathcal{B}$ is at least $2^{\ell_C - 1}$. Since the union of all components covers all roots of F that are contained in $\mathcal{B}$, and since the distance from C to any other component is at least 2^{ℓ_C}, it follows that the distance from x_C to any root of F is larger than $2^{\ell_C - 1}$. With $\Delta := \Delta_{2^{\ell_C - 3}}(x_C)$, it thus follows that $|F(x_C)| \geq 2^{-d} \cdot \max_{z \in \Delta} |F(z)|$, and using Lemma 6.33, we conclude that

$$\overline{\log}\, F(x_C)^{-1} = O(d \log d + \overline{\log}\, \sigma_F(\phi(C))^{-1} + \overline{\log}\, F'(\phi(C))^{-1}). \tag{6.20}$$

It follows that the cost for calling Algorithm 6.3 in Line 1 of the NewtonTest is bounded by

$$\begin{aligned}&\tilde{O}(d \cdot (\overline{\log}\, F(x_C)^{-1} + \tau_F + d \,\overline{\log}\, x_C)) \\ &\qquad\qquad = \tilde{O}(d \cdot (\tau_F + \overline{\log}\, \sigma_F(\phi(C))^{-1} + \overline{\log}\, F'(\phi(C))^{-1} + d \,\overline{\log}\, \hat{\phi}(C)))\end{aligned}$$

bit operations, Namely, Algorithm 6.3 succeeds with an absolute precision bounded by $O(\overline{\log}\, |F(x_C)|^{-1})$ and, within Algorithm 6.3, we need to approximately evaluate F and F' at the point x_C to such a precision; see also [KS15a, Lemma 3] for the cost of evaluating a polynomial of degree d to a certain precision. If we pass line 1, then we must have

$|F'(x_C)| > |F(x_C)|/(6r(C))$. Hence, in the for-loop of the NewtonTest, we succeed for an L of size $O(\overline{\log} F(x_C)^{-1} + d\,\overline{\log}\, w(C) - \ell_C + \log N_C)$, and thus the cost for computing $\tilde{x}'_C$ is bounded by

$$\tilde{O}(d \cdot (\overline{\log}\, \sigma_F(\phi(C))^{-1} + \overline{\log}\, F'(\phi(C))^{-1} + \overline{\log}\, \hat{\phi}(C) + \log N_C)),$$

where we again use (6.20) and the fact that $\sigma_F(\phi(C)) < d \cdot 2^{\ell_C+6}$ and $\overline{\log}\, w(C) = O(\log d + \overline{\log}\, \hat{\phi}(C))$. It remains to bound the cost for calling $\mathbf{T}_*$ on $\Delta' := \Delta_{2^{\ell_C}/(8N_C)}(\tilde{x}'_C)$ in the NewtonTest. Again, we use Lemma 6.16 and Lemma 6.33 to derive an upper bound of size

$$O(d \log d + \overline{\log}\, \sigma_F(\phi(C))^{-1} + \overline{\log}\, F'(\phi(C))^{-1} + |\mathcal{Z}(C^+)| \cdot \log N_C)$$

for $\overline{\log}(\max_{z\in\Delta'} |F(z)|)^{-1}$, and thus a bit complexity bound of size

$$\tilde{O}(d \cdot (\overline{\log}\, \hat{\phi}(C) + \overline{\log}\, \sigma_F(\phi(C))^{-1} + \overline{\log}\, F'(\phi(C))^{-1} + |\mathcal{Z}(C^+)| \cdot \log N_C))$$

for $\mathbf{T}_*(\Delta')$. This proves correctness of the bound in (6.19). We finally remark that the cost for all other (mainly combinatorial) steps are negligible. Namely, the bit-size b_C of a square in a component C is bounded by $O(\overline{\log}(w(C), w(C)^{-1}) + \log d) = O(\log d + \overline{\log}\, \sigma_F(\phi(C))^{-1} + \overline{\log}\, \hat{\phi(C)})$. Hence, each combinatorial step outside the NewtonTest, such as grouping together squares into maximal connected components in Line 8 of Bisection, needs $\tilde{O}(d)$ arithmetic operations with a precision $O(b_C)$, and thus a number of bit operations bounded by (6.18).

In the NewtonTest, we need to determine the squares $B_{i,j}$ of size $2^{\ell_C - 1}/N_C$ that intersect the disc Δ'. This step requires only a constant number of additions and multiplication, each carried out with a precision bounded by $O(\overline{\log}(w(C), w(C)^{-1}) + \log d + \log N_C)$. Hence, the cost for these steps is bounded by (6.19). □

When summing up the bound in (6.18) over all components C produced by the algorithm, we obtain the bit complexity bound

$$\begin{aligned}\tilde{O}\Big(d \cdot (d \cdot (|\mathcal{Z}(2\mathcal{B})| + \overline{\log}\, \mathrm{Mea}_F(2\mathcal{B}) + \overline{\log}(w(\mathcal{B}), w(\mathcal{B})^{-1})) + \tau_F \cdot |\mathcal{Z}(2\mathcal{B})| \\ + \sum_{z_i \in \mathcal{Z}(2\mathcal{B})} (\overline{\log}\, \sigma_F(z_i)^{-1} + \overline{\log}\, F'(z_i)^{-1}))\Big),\end{aligned} \tag{6.21}$$

for all steps outside the NewtonTest. Here, we exploit the fact that the pre-image of each (pseudo-) root in $\mathcal{Z}^+(2\mathcal{B})$ under each of the mappings $\phi, \hat{\phi}, \psi, \hat{\psi}$ has size $O(s_{\max} \cdot \log d)$, and that $|z_0| > w(\mathcal{B})/2$ for the pseudo-root $z_0 \in \mathcal{Z}^+(2\mathcal{B})$. If we now sum up the bound (6.19) for the cost of the NewtonTest over all components, we obtain a comparable complexity bound; however, with an additional term $d \cdot \sum_C |\mathcal{Z}(C^+)| \cdot \log N_C$, where the sum is only taken over the components for which the NewtonTest is called. The following considerations show that the latter sum is also dominated by the bound in (6.21).

Lemma 6.35. *Let $\mathcal{T}_{\text{New}} \subset \mathcal{T}$ be the set of all components in the subdivision tree $\mathcal{T}$ for which the NewtonTest is called. Then, $\sum_{C\in\mathcal{T}_{\text{New}}} |\mathcal{Z}(C^+)| \cdot \log N_C$ is bounded by*

$$\tilde{O}(d \cdot (\overline{\log}\, w(\mathcal{B}) + \overline{\log}\, \mathrm{Mea}_F(2\mathcal{B}) + |\mathcal{Z}(2\mathcal{B})|) + \tau_F \cdot |\mathcal{Z}(2\mathcal{B})| + \sum_{z_i\in\mathcal{Z}(2\mathcal{B})} \overline{\log}\, F'(z_i)^{-1}).$$

Proof. We define $\mathcal{T}_{\text{New}}^{=4} := \{C \in \mathcal{T}_{\text{New}} : N_C = 4\}$ and $\mathcal{T}_{\text{New}}^{>4} := \{C \in \mathcal{T}_{\text{New}} : N_C > 4\}$. Then, we have

$$\sum_{C\in\mathcal{T}_{\text{New}}^{=4}} |\mathcal{Z}(C^+)| \cdot \log N_C \le \sum_{C\in\mathcal{T}_{\text{New}}^{=4}} 2d \le \sum_{C\in\mathcal{T}} 2d \le 2d \cdot |\mathcal{Z}(2\mathcal{B})| \cdot s_{\max},$$

hence it remains to consider only the components $C \in \mathcal{T}_{\text{New}}^{>4}$. For such a component, let $\text{anc}^*(C) \in \mathcal{T}$ be the last ancestor for which the NewtonTest succeeded. According to Theorem 6.17 *(f)*, we have $N_C \leq 4 \cdot (w(\text{anc}^*(C))/w(C))^2$, and thus $|\mathcal{Z}(C^+)| \cdot \log N_C \leq 2 \cdot |\mathcal{Z}(C^+)| \cdot (1 + \log w(\text{anc}^*(C)) - \log w(C))$. In order to bound the later expression in terms of values that depend on the roots of F, let $z_i \in C^+$ be an arbitrary root in C^+. Then, assuming $w(C) \leq 1$, we obtain

$$\begin{aligned} |F'(z_i)| &= |a_d| \cdot \prod_{j \neq i : z_j \in \mathcal{Z}(2\Delta_C)} |z_i - z_j| \prod_{j : z_j \notin \mathcal{Z}(2\Delta_C)} |z_i - z_j| \\ &\leq |a_d| \cdot (2w(C))^{|\mathcal{Z}(2\Delta_C)|-1} \cdot \frac{\text{Mea}_{F(z_i - x)}}{|a_d|} \leq 2^{2d+\tau_F} w(C)^{|\mathcal{Z}(2\Delta_C)|-1} \cdot M(z_i)^d, \end{aligned}$$

where the last inequality follows from Landau's inequality [GG03, Theorem 6.31], that is, $\text{Mea}_p \leq \|p\|_2$ for any complex polynomial $p \in \mathbb{C}[x]$, and $\|F(z_i - x)\|_2 \leq \|F(z_i - x)\|_1 \leq 2^{\tau_F} M(z_i)^d 2^{d+1}$. For a more detailed derivation of the latter see [SM16, Lemma 22]. Furthermore, using that $\mathcal{Z}(2\Delta_C)$ contains at least two roots (as the NewtonTest is called) and that $C^+ \subset 2\Delta_C$, yields

$$|F'(z_i)| \leq 2^{2d+\tau_F} M(z_i)^d w(C)^{\frac{|\mathcal{Z}(2\Delta_C)|}{2}} \leq 2^{2d+\tau_F} M(z_i)^d w(C)^{\frac{|\mathcal{Z}(C^+)|}{2}}.$$

With $z_i := \phi(C)$ and $\overline{\log}\, \phi(C) \leq \log(64d) + \overline{\log}\, \hat{\phi}(C)$, we conclude that

$$-|\mathcal{Z}(C^+)| \cdot \log w(C) \leq 6d \log(64d) + 2\tau_F + 2d\, \overline{\log}\, \hat{\phi}(C) + \overline{\log}\, F'(\phi(C))^{-1}. \tag{6.22}$$

Notice that the latter inequality is trivially also fulfilled for a component C with $w(C) < 1$. In addition, it holds that

$$|\mathcal{Z}(C^+)| \cdot \log w(\text{anc}^*(C)) \leq |\mathcal{Z}(C^+)| \cdot (\overline{\log}\, \hat{\phi}(\text{anc}^*(C)) + \log(128d)). \tag{6.23}$$

The sum of the term at the right side of (6.22) over all C can be bounded by

$$\begin{aligned} &\tilde{O}(s_{\max} \cdot (|\mathcal{Z}(2\mathcal{B})| \cdot \tau_F + \sum_{z_i \in \mathcal{Z}^*(2\mathcal{B})} d\, \overline{\log}(z_i) + \sum_{z_i \in \mathcal{Z}(2\mathcal{B})} \overline{\log}\, F'(z_i)^{-1})) \\ &\quad = \tilde{O}(s_{\max} \cdot (|\mathcal{Z}(2\mathcal{B})| \cdot \tau_F + d\, \overline{\log}\, w(\mathcal{B}) + d\, \overline{\log}\, \text{Mea}_F(2\mathcal{B}) + \sum_{z_i \in \mathcal{Z}(2\mathcal{B})} \overline{\log}\, F'(z_i)^{-1})), \end{aligned}$$

as the pre-image of each (pseudo-) root $z_i \in 2\mathcal{B}$ (under ϕ and $\hat{\phi}$) has size at most $s_{\max} \log d$. We may further omit the factor $s_{\max}$ in the above bound as $\log \overline{\log}\, \sigma_F(z_i)^{-1} = O(\log(\tau_F + d\, \overline{\log}(z_i) + \overline{\log}\, F'(z_i)^{-1}))$ for an arbitrary root z_i of F, and thus

$$\begin{aligned} s_{\max} &= O(\log d + \log \overline{\log}\, w(\mathcal{B}) + \log \overline{\log}\, \sigma_F(2\mathcal{B})^{-1}) \\ &= O(\log d + \log \overline{\log}\, w(\mathcal{B}) + \log \tau_F + \log \overline{\log}\, \text{Mea}_F(2\mathcal{B}) + \log \sum_{z_i \in \mathcal{Z}(2\mathcal{B})} \overline{\log}\, F'(z_i)^{-1}). \end{aligned}$$

It remains to bound the sum of the term $|\mathcal{Z}(C^+)| \cdot \overline{\log}\, \hat{\phi}(\text{anc}^*(C))$ on the right side of (6.23) over all $C \in \mathcal{T}_{\text{New}}^{>4}$. Let $\mathcal{T}_{\text{anc}^*} := \{C^* \in \mathcal{T} : \exists C \in \mathcal{T} \text{ with } \text{anc}^*(C) = C^*\}$. Then, for a fixed $C^* \in \mathcal{T}_{\text{anc}^*}$, it holds that any $C \in \mathcal{T}_{\text{New}}^{>4}$ with $\text{anc}^*(C) = C^*$ is connected with C^* in $\mathcal{T}$ via a path of length at most $\hat{s} := \log \overline{\log}\, w(\mathcal{B}) + \log \overline{\log}\, \sigma_F(2\mathcal{B})^{-1}$. Namely, we have already shown that $\log \log N_C \leq \hat{s}$ for all $C \in \mathcal{T}$, and thus $N_C > 4$ implies that the path connecting C and C^*

must have length at most $\hat{s}$. Hence, it follows that

$$\begin{aligned}\sum_{C\in\mathcal{T}^{>4}_{\textsc{New}}:\mathrm{anc}^*(C)=C^*} |\mathcal{Z}((C)^+)|\cdot\log w(\mathrm{anc}^*(C)) &= \sum_{C\in\mathcal{T}^{>4}_{\textsc{New}}:\mathrm{anc}^*(C)=C^*} |\mathcal{Z}((C)^+)|\log w(C^*)\\ &\le (\overline{\log}\,\hat{\phi}(C^*)+\log(128d)) \sum_{C\in\mathcal{T}^{>4}_{\textsc{New}}:\mathrm{anc}^*(C)=C^*} |\mathcal{Z}(C^+)|\end{aligned}$$

As for any two components C_1, C_2 in the last sum above, either $C_1^+\cap C_2^+=\emptyset$, $C_1^+\subset C_2^+$, or $C_2^+\subset C_1^+$, it follows that this sum is upper bounded by $\hat{s}\cdot|\mathcal{Z}((C^*)^+)|\le\hat{s}\cdot|\mathcal{Z}(2\mathcal{B})|$. Thus,

$$\begin{aligned}\sum_{C\in\mathcal{T}^{>4}_{\textsc{New}}} |\mathcal{Z}(C^+)|\cdot\log w(\mathrm{anc}^*(C)) &\le \hat{s}\cdot|\mathcal{Z}(2\mathcal{B})|\cdot\sum_{C^*\in\mathcal{T}_{\mathrm{anc}^*}}(\overline{\log}\,\hat{\phi}(C^*)+\log(128d))\\ &= O(s_{\max}\cdot\hat{s}\cdot d\log d\cdot\sum_{z_i\in\mathcal{Z}^+(2\mathcal{B})}(\overline{\log}(z_i)+\log(d)).\end{aligned}$$

Now, we obtain the bound by using the definitions of $\mathrm{Mea}_F(2\mathcal{B})$ and the pseudo-root z_0, and then the definition of $\hat{s}$, as well as the bound on $s_{\max}$:

$$\begin{aligned}&\tilde{O}(s_{\max}\cdot\hat{s}\cdot d\cdot(\overline{\log}\,w(\mathcal{B})+\overline{\log}\,\mathrm{Mea}_F(2\mathcal{B})+|\mathcal{Z}(2\mathcal{B})|))\\ &\quad=\tilde{O}(d\cdot(\overline{\log}\,w(\mathcal{B})+\overline{\log}\,\mathrm{Mea}_F(2\mathcal{B})+|\mathcal{Z}(2\mathcal{B})|+\log\sum_{z_i\in\mathcal{Z}(2\mathcal{B})}\overline{\log}\,F'(z_i)^{-1})).\qquad\square\end{aligned}$$

We summarize our results in the following main theorem.

Theorem 6.36. *Let F be a polynomial as defined in (6.1) and let $\mathcal{B}\subset\mathbb{C}$ be an arbitrary axis-aligned square. Then, the algorithm $\mathbb{C}$Isolate with input $\mathcal{B}$ uses*

$$\begin{aligned}\tilde{O}\Big(d\cdot(d\cdot(|\mathcal{Z}(2\mathcal{B})|+\overline{\log}\,\mathrm{Mea}_F(2\mathcal{B})+\overline{\log}(w(\mathcal{B}),w(\mathcal{B})^{-1}))+\tau_F\cdot|\mathcal{Z}(2\mathcal{B})|\\ +\sum_{z_i\in\mathcal{Z}(2\mathcal{B})}(\overline{\log}\,\sigma_F(z_i)^{-1}+\overline{\log}\,F'(z_i)^{-1}))\Big)\end{aligned}\tag{6.24}$$

bit operations. As input, the algorithm requires an L-bit approximation of F with

$$\begin{aligned}L=\tilde{O}(d\cdot(\mathcal{Z}(2\mathcal{B})+\overline{\log}\,\mathrm{Mea}_F(2\mathcal{B})+\overline{\log}(w(\mathcal{B}),w(\mathcal{B})^{-1}))+\tau_F\cdot|\mathcal{Z}(2\mathcal{B})|\\ +\sum_{z_i\in\mathcal{Z}(2\mathcal{B})}(\overline{\log}\,\sigma_F(z_i)^{-1}+\overline{\log}\,F'(z_i)^{-1})).\end{aligned}\tag{6.25}$$

Proof. The bound in (6.24) on the bit complexity follows immediately from Lemma 6.34, Lemma 6.35, and the remark following Lemma 6.34. The bound in (6.25) on the precision demand follows directly from the proof of Lemma 6.34 and Lemma 6.16. $\square$

Notice that the above complexity bounds are directly related to the size of the input square $\mathcal{B}$ as well as to parameters that only depend on the roots located in $2\mathcal{B}$. This makes our complexity bound adaptive in a very strong sense. In contrast, one might also be interested in (probably simpler) bounds when using our algorithm to isolate all complex roots of F. In this case, we may first compute an input square $\mathcal{B}$ of width $w(\mathcal{B})=2^{\Gamma+2}$ that is centered at the origin. Here, $\Gamma\in\mathbb{N}_{\ge1}$ is an integer bound for Γ_F with $\Gamma=\Gamma_F+O(\log d)$. We have already argued in Section 6.2 that such a bound Γ can be computed using $\tilde{O}(d^2\Gamma_F)$ bit operations. Such a square $\mathcal{B}$ contains all complex roots of F, and thus running $\mathbb{C}$Isolate with input $\mathcal{B}$ yields corresponding isolating discs. Hence, we obtain the following result:

Corollary 6.37. *Let F be a square-free polynomial as in (6.1). Then, for isolating all complex roots of F, $\mathbb{C}$Isolate needs*

$$\tilde{O}(d \cdot (d^2 + d \,\overline{\log}\, \mathrm{Mea}_F + \sum_{i=1}^{d} \overline{\log}\, F'(z_i)^{-1})) = \tilde{O}(d \cdot (d^2 + d \,\overline{\log}\, \mathrm{Mea}_F + \overline{\log}\, \mathrm{Disc}_F^{-1})) \qquad (6.26)$$

bit operations, where $\mathrm{Disc}_F := |a_d|^{2d-2} \prod_{1 \le i < j \le d} (z_j - z_i)^2$ *is the* discriminant of F. *As input, the algorithm requires an L-bit approximation of F with*

$$L = \tilde{O}(d^2 + d \,\overline{\log}\, \mathrm{Mea}_F + \overline{\log}\, \mathrm{Disc}_F^{-1}) \qquad (6.27)$$

Proof. The above bounds follow directly from the bounds in (6.24) and (6.25), and the fact that $d \,\overline{\log}(w(\mathcal{B}), w(\mathcal{B})^{-1}) + d\tau_F = O(d^2 + d \,\overline{\log}\, \mathrm{Mea}_F)$ and thus

$$\sum_{i=1}^{d} \overline{\log}\, \sigma_F(z_i)^{-1} = O(d^2 + d \,\overline{\log}(\mathrm{Mea}_F) + \sum_{i=1}^{d} \overline{\log}\, F'(z_i)^{-1}).$$

The statement now follows using

$$\sum_{i=1}^{d} \overline{\log}\, F'(z_i)^{-1} = O(d^2 + d \,\overline{\log}\, \mathrm{Mea}_F + \overline{\log}\, \mathrm{Disc}_F^{-1}),$$

see, e.g. [SM16, Section 2.5] and [SM16, Theorem 31] for proofs of the latter bounds. □

Again, we provide simpler bounds for the special case, where F has integer coefficients.

Corollary 6.38. *Let $f \in \mathbb{Z}[x]$ be a square-free integer polynomial of degree d with integer coefficients of bit-size less than τ, let $F := f/\mathrm{LC}(f)$, and let $\mathcal{B} \subset \mathbb{C}$ be an axis-aligned square with $2^{-O(\tau)} \le w(\mathcal{B}) \le 2^{O(\tau)}$. Then, $\mathbb{C}$Isolate with input $\mathcal{B}$ needs $\tilde{O}(d^3 + d^2\tau)$ bit operations. The same bound also applies when using $\mathbb{C}$Isolate to compute isolating discs for all roots of f.*

Proof. The bound on the number of bit operations for $\mathbb{C}$Isolate on an input square $\mathcal{B}$ follows immediately from (6.26) and the fact that $\mathrm{Disc}_F = \mathrm{LC}(f)^{2d-2} \cdot \mathrm{Disc}_f \ge 2^{-(2d-2)\tau}$. For the second claim, we may simply run $\mathbb{C}$Isolate on a square $\mathcal{B}$ of width $2^{\tau+2}$ centered at the origin. According to Cauchy's root bound, $\mathcal{B}$ contains all roots of f, and thus $\mathbb{C}$Isolate yields corresponding isolating discs. □

6.7 Conclusion

In this chapter, we have seen a simple and efficient subdivision algorithm to isolate the complex roots of a polynomial with arbitrary complex coefficients. Our algorithm achieves complexity bounds that are comparable to the best known bounds for this problem. Previously, such bounds were achieved only by methods based on fast polynomial factorization [EPT14; MSW15; Pan02]. Compared to these methods, our algorithm is quite simple and uses only fast algorithms for polynomial multiplication and Taylor shift computation but no other, more involved, asymptotically fast subroutines. Thus, there is hope that the algorithm can be turned into an efficient implementation similar to its real counterpart from [SM16], see [KRS16] for an evaluation of its implementation. In fact, there are currently efforts to include both methods, the real and the complex one, into the computer algebra system Maple.

A possible direction of future research is to extend the current Newton-bisection technique and complexity analysis to the analytic roots algorithm in [YSS13]. See [Str12] for an alternative approach for the computation of the real roots of analytic functions obtained by composing polynomials and the functions log, exp, and arctan.

At the end of Section 6.5.3, we sketched how to use our algorithm to isolate the roots of a not necessarily square-free polynomial for which the number of distinct complex roots is given as additional input. Also, we may use our algorithm to further refine the isolating discs for the roots of a polynomial in order to compute L-bit approximations of all roots. There exist dedicated methods [KS15a; MSW15; Pan02; PT13; PT14] for refining intervals or discs, that are already known to be isolating for the roots of a polynomial. For large L, that is if L dominates other parameters, their bit complexity is $\tilde{O}(dL)$. In comparison, using $\mathbb{C}$Isolate for the refinement directly, its bit complexity would be of size $\tilde{O}(d^2L)$. We suspect that this bound can be further improved by using a proper modification of the $\mathbf{T}_*$-test, which only needs to evaluate F and its first derivative, and approximate multipoint evaluation.

Chapter 7

Counting Solutions of a Polynomial System Locally and Exactly

7.1 Introduction

In this chapter, we turn to the problem of computing the solutions of zero-dimensional polynomial systems. We propose a randomized but certified (i.e. Las-Vegas type) algorithm, denoted #PolySol, to count the number of solutions of a zero-dimensional polynomial system $\mathcal{F}$ within a given polydisc $\mathbf{\Delta} \subset \mathbb{C}^n$. More precisely, let

$$\mathcal{F} : f_1(\mathbf{x}) = \ldots = f_n(\mathbf{x}) = 0, \quad \text{with } f_i \in \mathbb{C}[\mathbf{x}] = \mathbb{C}[x_1, \ldots, x_n] \text{ for all } i = 1, \ldots, n, \tag{7.1}$$

be a zero-dimensional[1] polynomial system. As for the univariate case before, we assume that each of the coefficients $c_{i,\alpha}$ of the polynomials

$$f_i = \sum_{\alpha=(\alpha_1,\ldots,\alpha_n)} c_{i,\alpha} \cdot \mathbf{x}^\alpha = \sum_{\alpha=(\alpha_1,\ldots,\alpha_n)} c_{i,\alpha} \cdot x_1^{\alpha_1} \cdots x_n^{\alpha_n}$$

can be approximated to any desired precision. That is, for any given non-negative integer precision ρ, we can ask for a dyadic approximation $\tilde{c}_{i,\alpha} \in 2^{-\rho} \cdot (\mathbb{Z} + \mathbf{i} \cdot \mathbb{Z})$ of $c_{i,\alpha}$ with $|\tilde{c}_{i,\alpha} - c_{i,\alpha}| < 2^{-\rho}$ for the cost of reading the approximations.

Given a polydisc $\mathbf{\Delta} = \mathbf{\Delta}_r(\mathbf{m}) = \{\mathbf{z} \in \mathbb{C}^n : \|\mathbf{z} - \mathbf{m}\| < r\}$ of radius r centered at $\mathbf{m} \in \mathbb{C}^n$, we aim to compute the number of solutions of $\mathcal{F} = 0$ that are contained in $\mathbf{\Delta}$. Here, solutions are counted with multiplicity. As input, the algorithm #PolySol receives the coefficients of $\mathcal{F}$, the polydisc $\mathbf{\Delta}$, and an integer $K \in \{0, 1, \ldots, d_{\mathcal{F}}\}$, where $d_{\mathcal{F}} := \max_i d_i$ is the maximum of the degrees d_i of the polynomials f_i. As output, it returns an integer $k \in \mathbb{N} \cup \{-1\}$. There are two possible outcomes – either the algorithm fails or the algorithm succeeds. If $k = -1$, nothing can be said, that is, the algorithm fails to provide an answer to our request. If the algorithm succeeds however, it guarantees that k equals the number of solutions of $\mathcal{F} = 0$ in $\mathbf{\Delta}$. Furthermore, we show that our algorithm always succeeds under the following conditions. (1) If r is small enough. (2) The input parameter K is at least the number of solutions of $\mathcal{F}$ in $\mathbf{\Delta}$. (3) The smaller polydisc $\mathbf{\Delta}' := \mathbf{\Delta}_{r'}(\mathbf{m})$, with $r' := r/64n(K+1)^n$, contains a k-fold solution of $\mathcal{F}$. We also derive a bound on the size of r that guarantees success of our method if the other two requirements are fulfilled. The given bound is adaptive in the sense that it does not only depend on global parameters such as the degree and the size of the coefficients of the polynomials f_i, but also on solution-specific parameters, that is, the multiplicity and the size of $\mathbf{z}$ as well as the distances between $\mathbf{z}$ and the other solutions of $\mathcal{F}$. This can be seen as an analogue result to the results from the previous chapter, where we have seen that the geometry of the roots of the polynomial determines the cost of the algorithm $\mathbb{C}$Isolate for isolating the roots.

[1]There are only finitely many solution in complex projective n-space.

We proceed by stating our main result for the special case, where $\mathcal{F}$ is defined over the integers. For the more general statement, see Theorem 7.22.

Theorem 7.1. *Suppose that* $\mathbf{z}$ *is a k-fold solution of a polynomial system* $\mathcal{F}$ *as in (7.1) with polynomials* $f_i \in \mathbb{Z}[\mathbf{x}]$ *of total degree* d_i *and with* integer *coefficients* $c_{i,\alpha}$ *of bit-size less than* $\tau_{\mathcal{F}}$*. Then, for any* $K \geq k$*, there exists an* $L^* \in \mathbb{N}$ *with*

$$L^* = \tilde{O}\left(D_{\mathcal{F}} \cdot \max_{i\in[n]} \frac{d_{\mathcal{F}} + \tau_{\mathcal{F}}}{d_i} + D_{\mathcal{F}} \cdot \overline{\log}(\mathbf{z}) + \overline{\log}(\partial(\mathbf{z},\mathcal{F})^{-1}) + (K+1)^n \cdot \overline{\log}(\sigma(\mathbf{z},\mathcal{F})^{-1})\right)$$

such that, with probability at least $1/2$*, the algorithm* #PolySol($\mathcal{F},\Delta,K$) *returns* k *for any disc* $\Delta = \Delta_r(\mathbf{m})$ *with* $r \leq 2^{-L^*}$ *and* $\|\mathbf{m}-\mathbf{z}\| < r$*. Here, we use the definitions* $d_{\mathcal{F}} := \max_{i\in[n]} d_i$*,* $D_{\mathcal{F}} := \prod_{i=1}^n d_i$*,*

$$\sigma(\mathbf{z}_i,\mathcal{F}) := \min_{j\neq i} \|\mathbf{z}_i - \mathbf{z}_j\|, \text{ and } \quad \partial(\mathbf{z}_i,\mathcal{F}) := \prod_{j\neq i} \|\mathbf{z}_i - \mathbf{z}_j\|^{\mu(\mathbf{z}_j,\mathcal{F})},$$

where $\mathbf{z}_1,\dots,\mathbf{z}_N$ *denote the distinct solutions of* $\mathcal{F}$ *and* $\mu(\mathbf{z}_i,\mathcal{F})$ *the multiplicity of* $\mathbf{z}_i$*.*

Notice that the method never yields the exact multiplicity of a solution, even in the case where there is a well separated k-fold solution $\mathbf{z}$ in Δ. Instead, we only obtain the sum of the multiplicities of all solutions contained in Δ. However, in the considered computational model, where only approximations of the coefficients of the input polynomials are known, it is simply not possible to achieve a stronger result. This is due to the fact that arbitrary small perturbations of the input already destroy the multiplicity structure of non-simple roots.

There is a series of applications of our method. For instance, it can be used to verify correctness of the result provided by a numerical (non-certified) method such as homotopy methods (e.g. [Ver99; Bat+13]) or subdivision methods (e.g. [MP09; Bur+08]). Corresponding implementations of such methods (e.g. Bertini, PHCpack, axel) are available and have proven to be efficient and reliable in practice. Suppose that such a method returns an approximation ζ of a k-fold solutions $\mathbf{z}$ such that $\|\zeta-\mathbf{z}\| < 2^{-L}$, however, *without any guarantee on the correctness of the result*. Now, in order to show correctness, we may run the algorithm #PolySol with input $\mathcal{F}$, $K = k$, and $\Delta = \Delta_{64n(k+1)^n\cdot 2^{-L}}(\zeta)$. According to the above theorem, the method returns k if the claimed result is actually correct and L is large enough. Hence, we eventually succeed if the numerical solver provides a sufficiently good approximation of $\mathbf{z}$ together with the correct multiplicity. Again, we remark that the method does not provide a proof that there is exactly one root of multiplicity k, but only a proof that there are k roots counted with multiplicity in Δ.

For polynomial systems that are defined over the integers, there exist complete and certified methods (e.g. [Rou99; Laz09; BS16]) to compute isolating regions for all solutions together with the corresponding multiplicities, however, their possible application is limited in practice. In particular, if the polynomials f_i are of large degree, the running time for the necessary symbolic computations (e.g. that of a Gröbner Basis or resultants) becomes prohibitive. Combining our method with a numerical solver may instead yield a certified result on the existence of solutions in a certain region.

In Section 7.5, we report on a preliminary implementation of our method. That is, we integrated an implementation of our method into the Bisolve-routine [Ber+13; KS15b], a highly efficient algorithm for isolating the solutions of a bivariate polynomial systems with integer coefficients. There, it serves as an inclusion predicate to verify the existence of a k-fold solution of the system. Compared to the original approach in Bisolve, we observe a considerable improvement with respect to running time and precision demand.

Overview of the Algorithm. Recall Pellet's theorem that we have described in the previous chapter. It can be used for counting the number of roots of a univariate polynomial $f \in \mathbb{C}[x]$ in a disc $D_r(m) = \{x \in \mathbb{C} : |x - m| \leq r\}$ of radius r centered at a point $m \in \mathbb{C}$. Let us call $f[m](x) := f(m + x)$ the shift of f to m, see Section 7.2. The test based on Pellet's theorem works as follows. We first compute the Taylor-expansion

$$f[m](x) = \sum\nolimits_{i\geq 0} c_i \cdot x^i = \sum\nolimits_{i\geq 0} \frac{f^{(i)}(m)}{i!} \cdot x^i$$

at m and then check whether $|c_k| \cdot r^k > \sum_{i\neq k} |c_i| \cdot r^i$ for some k. Notice that the latter inequality implies that the part $c_k \cdot x^k$ of $f[m]$ of degree k dominates the remaining parts on the boundary of the disc $D_r(0)$. If this is the case, then $D_r(m)$ contains exactly k roots of f, which we have proven in Theorem 6.4. In the previous chapter, we have also given sufficient conditions on r and the locations of the roots with respect to m such that the above inequality is fulfilled. In particular, for m being a k-fold root of f, we gave a bound r_0 in terms of the degree of f and the separation of m such that Pellet's Theorem applies for any $r < r_0$. We will summarize the most important results related to Pellet's theorem for this chapter in Lemma 7.18.

Our algorithm #PolySol can be considered as an extension of Pellet's Theorem to polynomial systems. Similar as in the one-dimensional case, we make crucial use of the fact that, for a sufficiently small neighborhood $\boldsymbol{\Delta}$ of a k-fold solution $\mathbf{z}$ of $\mathcal{F}$, the system

$$\mathcal{F}[\mathbf{z}] : f_1(\mathbf{x} + \mathbf{z}) = \cdots = f_n(\mathbf{x} + \mathbf{z}) = 0$$

obtained by shifting each of the polynomials f_i by $\mathbf{z}$ is dominated by terms of degree k or less. Hence, in order to study the local behavior of $\mathcal{F}$ at $\mathbf{z}$, it should suffice to consider the truncation $\mathcal{F}[\mathbf{z}]_{\leq k}$ of $\mathcal{F}[\mathbf{z}]$, where we only consider the part $f_i[\mathbf{z}]_{\leq k} = \sum_{\alpha:|\alpha|\leq k} c'_{i,\alpha} \cdot \mathbf{x}^\alpha$ of each $f_i[\mathbf{z}] = f(\mathbf{x} + \mathbf{z}) = \sum_\alpha c'_{i,\alpha} \cdot \mathbf{x}^\alpha$ that is of degree k or less. In fact, in Corollary 7.21, we prove that, for any $K \geq k$, the system $\mathcal{F}[\mathbf{z}]_{\leq K}$ has a k-fold solution at the origin, and we give a bound on its separation in terms of the separation of $\mathbf{z}$ as a solution of the original system $\mathcal{F}$. In Theorem 7.20, we even show that if $K \geq k$, and if $\|\mathbf{m} - \mathbf{z}\| < 2^{-L}$ for a sufficiently large L, then we can work with $\mathcal{F}[\mathbf{m}]_{\leq K}$ instead of $\mathcal{F}[\mathbf{z}]$. Namely, in this case, $\mathcal{F}[\mathbf{m}]_{\leq K}$ has k solutions of norm less than $4 \cdot 2^{-L}$, whereas all remaining solutions have considerably larger norm, that is, larger than some value that does not depend on L.

We now provide an overview of our approach. For the sake of simplicity, we omit technical details and only give the main ideas. Also, we do not treat any special cases, which considerably simplifies the approach when compared to the actual algorithm as given in Section 7.3. We first define $L := \lceil \log \frac{r}{32n(K+1)^n} \rceil$ such that $\frac{r}{64n(K+1)^n} \leq 2^{-L} \leq \frac{r}{32n(K+1)^n} = r'$. Obviously, we cannot check in advance whether the above requirements on $\mathbf{m}$ and L are fulfilled, however, we can check whether $\mathcal{F}[\mathbf{m}]_{\leq K}$ has a cluster of solutions near the origin. For this, we use a complete and certified algorithm to compute isolating regions of all solutions of $\mathcal{F}[\mathbf{m}]_{\leq K}$ that are contained in the polydisc $\boldsymbol{\Delta} = \boldsymbol{\Delta}_r(0)$. Notice that if K is small compared to the degrees of the polynomials f_i, then the cost for computing the solutions of $\mathcal{F}[\mathbf{m}]_{\leq K}$ is much lower than solving the original system directly. In particular, for $K = 1$, the truncated system $\mathcal{F}[\mathbf{m}]_{\leq K}$ becomes a linear system in n variables. Now, suppose that $\boldsymbol{\Delta}$ contains k' solutions of $\mathcal{F}[\mathbf{m}]_{\leq K}$ (k' does not have to be equal to k) that are well separated from the remaining solutions, then we are left to show that $\mathcal{F}[\mathbf{m}]$ contains the same number of solutions in $\boldsymbol{\Delta}$. For this, we use a generalization of Rouché's Theorem that applies to analytic functions in n-dimensional complex space; see Theorem 7.17. This approach requires to compute a lower bound LB for $\|\mathcal{F}[\mathbf{m}]_{\leq K}(\mathbf{x})\| := \max_i |f_i[\mathbf{m}]_{\leq K}|$ on the boundary of $\boldsymbol{\Delta}$ as well as a corresponding upper bound UB on the error $\|\mathcal{F}[\mathbf{m}](\mathbf{x})_{\leq K} - \mathcal{F}[\mathbf{m}](\mathbf{x})\|$ that occurs when passing from $\mathcal{F}[\mathbf{m}]$ to the truncated system $\mathcal{F}[\mathbf{m}]_{\leq K}$. Whereas the computation of UB is straightforward (see (7.11) in Section 7.3), the computation of LB is more involved. Namely, we first compute the hidden-variable resultant $R_\ell := \operatorname{Res}(\mathcal{F}[\mathbf{m}]_{\leq K}, x_\ell) \in \mathbb{Q}[x_\ell]$ with respect to each of the variables x_ℓ; see Section 7.2 for details on the hidden variable approach. The roots of R_ℓ are the projections of the solutions of $\mathcal{F}[\mathbf{m}]_{\leq K}$ on the x_ℓ-axis, and R_ℓ is contained in the ideal given by the polynomials $f_i[\mathbf{m}]_{\leq K}$, that is, there exist $g_{\ell,1}, \dots, g_{\ell,n} \in \mathbb{Q}[\mathbf{x}]$ with

$$R_\ell = g_{\ell,1} \cdot f_i[\mathbf{m}]_{\leq K} + \cdots + g_{\ell,1} \cdot f_i[\mathbf{m}]_{\leq K}. \tag{7.2}$$

Using a recent result [DKS13] on the arithmetic Nullstellensatz, we derive upper bounds on the absolute value of the coefficients of the polynomials $g_{\ell,1}$; see Corollary 7.11 and (7.13) in Section 7.3. In addition, we use our results on Pellet's Theorem from the previous chapter in order to derive a lower bound for $|R_\ell|$ on the boundary of the disc $D_r(0) \subset \mathbb{C}$, which is the projection of the polydisc $\mathbf{\Delta}$ into one-dimensional space; see Lemma 7.18. Combining the latter two bounds then yields LB. Finally, we check whether LB > UB, in which case we conclude from Rouché's Theorem that $\mathcal{F}[\mathbf{m}]$ has the same number of solution in $\mathbf{\Delta}$ as the truncated system $\mathcal{F}[\mathbf{m}]_{\leq K}$. If UB < LB, we return -1.

In the analysis of our algorithm, we show that if $\|\mathbf{m}-\mathbf{z}\| < r/(64n(K+1)^n)$ for a sufficiently small r, then LB approximately scales like $c \cdot r^k$ for some constant C, whereas UB scales like $C' \cdot r^{K+1}$ for some constant C'. Thus, in this case, our algorithm eventually succeeds if $K \geq k$. As already mentioned, we omitted many details in the above description. In particular, for completeness, we needed to address certain special cases. In particular, this comprises the case where $\mathcal{F}[\mathbf{m}]_{\leq k}$ has distinct solutions whose projections on one of the coordinate axis are (almost) equal or solutions at infinity that yield roots of the hidden variable resultant. We show how to handle such situations by means of a random rotation of the coordinate system without harming the claimed complexity bounds.

Implementation for the Bivariate Case. For the special case of a polynomial system $\mathcal{F}$: $f_1(x_1, x_2) = f_2(x_1, x_2) = 0$ in two variables, with $f_1, f_2 \in \mathbb{Z}[x_1, x_2]$, we implemented the algorithm in Sage. As an oracle for computing an arbitrary good approximation of a solution $\mathbf{z}$ of $\mathcal{F}$, we use a subroutine of the so-called Bisolve algorithm from [Ber+13; KS15b], which currently constitutes one of the fastest exact and complete algorithms for solving bivariate systems. Bisolve is a classical elimination approach that projects the solutions of the system on each of the two coordinate axis in a first step by means of resultant computation and root isolation. This yields a set of points on a two-dimensional grid that are all possible candidates for the solutions of the system. Also, the candidates can be approximated to an arbitrary precision using root refinement for univariate polynomials. Then, in a second step, in order to check whether a certain candidate is a solution or not, Bisolve combines interval arithmetic and an inclusion test based on bounds on the cofactors g_1 and g_2 in the representation $R = g_1 \cdot f_1 + g_2 \cdot f_2$ of the resultant polynomials R as an element in the ideal $\langle f_1, f_2 \rangle$. This inclusion test is similar to the approach that we will present in this chapter, however, no truncation of the original system is considered. Also, it is tailored to the bivariate case and does not yield the multiplicity of a solution. In our experiments that we will report on in Section 7.5, we replace the original inclusion test in the Bisolve algorithm by #PolySol and compare the precision demand and the running time to that of the original variant. In our evaluation, we observe that, for a multiplicity k of $\mathbf{z}$ that is small in comparison to the degrees of the input polynomials, our novel approach outperforms the original variant. At least, for the considered instances, we observe a sub-linear dependency of the needed precision on the degrees of the input polynomials. Notice that this is not in line with the derived bounds on the precision demand, which suggest at least a quadratic dependency. However, we remark that the given bounds are just worst-case bounds. We also want to note that our implementation should rather be considered as a proof-of-concept than as an implementation that should be directly used as provided. The experiments in Section 7.5 should be considered as preliminary, nevertheless they give interesting hints and support the belief that our approach can lead to very efficient implementations.

Related Work. The literature on solving zero-dimensional polynomial systems is vast and we can only give an incomplete overview. A historical summary can be found in [Laz09]. There are roughly two different classes of methods – numeric methods and symbolic methods. In contrast to symbolic methods, the numeric methods are in general not guaranteed to be complete or correct. On the other hand they are usually efficient in practice and some of them seem to be quite reliable.

One very classic approach is Newton's method, see [Rum10, Section 13] for a general description and an approach that uses Newton's iteration with interval arithmetic. Moreover, this work presents techniques for verifying the results of Newton's iteration. Another very popular numeric approach are homotopy continuation methods. There has been also quite some implementation effort, see PHCpack [Ver99] and Bertini [Bat+13]. We also want to mention the work by Verschelde and Haegemans [VH94]. From a high-level point of view, their approach is similar to ours as it is also based on Rouché's theorem. Their method relies on finding a sparse part of the polynomial system that dominates the rest of the system on the border of a considered region and can be used as a better starting system for homotopy based techniques. The main differences to our approach are the following. First, we use our technique to directly certify the existence of a zero, not only in order to construct a starting system for a numerical method. Moreover, the system that we use in order to approximate the input system is of lower degree, more precisely our "dominating part" is always of degree k if k is the multiplicity of the zero in the given region.[2] In contrast to their result, we also show that the precision that is needed in order to do so directly depends on the arrangements of the zeros of the system. The subdivision methods [MP09; Bur+08] are usually incomplete in the sense that they only provide exclusion predicates and lack inclusion predicates. Thus they can be used in order to compute regions that are guaranteed to be free of solutions to the system but cannot ultimately guarantee that a region contains a zero. We want to stress that our work now provides an inclusion predicate that could be included in these approaches in order to turn these methods into complete methods.

A very classical approach are methods based on Groebner Bases. Their theoretical description goes back to Buchberger [Buc06]. A very efficient implementation is due to Faugère, see [Fau02]. The approach of Rational Univariate Representation (RUR) goes back to Rouillier [Rou99], and has found its way into very efficient implementations. Another quite successful approach are techniques based on resultants, see for example [BS16] for a recent approach. Other works that aim at certifying solutions of systems include the work of Hauenstein and Levandovskyy [HL17]. Their approach even applies for a more general setting, namely systems of polynomial-exponential equations and is based on Newton's iteration and Smale's α-theory.

Structure of the Remainder of this Chapter. In Section 7.2, we introduce some further notation and then work out the basic concepts that we will need later on. In Section 7.3, we present the algorithm #PolySol and describe the main rationale behind it. In Section 7.4, we analyze the algorithm and in Section 7.5, we describe the Sage implementation of our method for the bivariate case and its integration into the Bisolve-like approach.

[2]Note that it is not strictly necessary to know this parameter k, since a binary search for k can find a good enough approximation.

7.2 Preliminaries

7.2.1 Notation and Definitions

We start by introducing further frequently used notation and important definitions.

1. For a polynomial $f = \sum_\alpha c_\alpha \mathbf{x}^\alpha \in \mathbb{C}[\mathbf{x}]$, we define
$$\deg(f) := \max_{\alpha=(\alpha_1,\dots,\alpha_n):c_\alpha\neq 0} \alpha_1 + \ldots + \alpha_n$$
to be the *(total) degree of* f. Recall that $\tau_f := \lceil \log \|f\| \rceil$.

2. For a polynomial system $\mathcal{F} = (f_1,\dots,f_n)$, with $f_i \in \mathbb{C}[\mathbf{x}]$ of total degree d_i, we define
$$d_\mathcal{F} := \max_{i\in[n]} d_i, \quad D_\mathcal{F} := \prod_{i\in[n]} d_i, \quad \text{and} \quad \tau_\mathcal{F} := \max_{i\in[n]} \tau_{f_i}.$$
$D_\mathcal{F}$ is also called the *Bézout bound* in the literature. It constitutes an upper bound on the total number of solutions (counted with multiplicities) of a zero-dimensional system $\mathcal{F}$. For a system $\mathcal{F}$ with generic coefficients, it actually equals the number of solutions.

3. We further say that a polynomial $f = \sum_\alpha c_\alpha \mathbf{x}^\alpha \in \mathbb{Z}[\mathbf{x}]$ with *integer* coefficients has *magnitude* (d,τ) if $d_f \le d$ and $\tau_f \le \tau$. A system $\mathcal{F} = (f_1,\dots,f_n)$ with $f_i \in \mathbb{Z}[\mathbf{x}]$ has magnitude (d,τ) if each polynomial has magnitude (d,τ).

4. As for univariate polynomials, for a multivariate polynomial $f = \sum_\alpha c_\alpha \mathbf{x}^\alpha \in \mathbb{C}[\mathbf{x}]$ and a positive integer κ, we say that $\phi = \sum_\alpha \tilde{c}_\alpha \mathbf{x}^\alpha$ is an (absolute) κ-bit approximation of f if each $\tilde{c}_\alpha$ is a dyadic number of the form $(m + m' \cdot \mathbf{i}) \cdot 2^{-(\kappa+1)} \in \mathbb{Q} + \mathbf{i} \cdot \mathbb{Q}$, with $m, m' \in \mathbb{Z}$, and $\|f - \phi\| \le 2^{-\kappa}$. In other words, each $\tilde{c}_\alpha$ approximates c_α to κ bits after the binary point.

5. For $\mathbf{z} \in \mathbb{C}^n$ and a polynomial $f \in \mathbb{C}[\mathbf{x}]$, we define
$$f[\mathbf{z}](\mathbf{x}) := f(\mathbf{x}+\mathbf{z}) = \sum_\alpha c_\alpha (\mathbf{x}+\mathbf{z})^\alpha$$
to be the *shift of* f *to* $\mathbf{z}$. For a system $\mathcal{F} = (f_1,\dots,f_n)$, we define the *shift of* $\mathcal{F}$ *to* $\mathbf{z}$ as $\mathcal{F}[\mathbf{z}] = (f_1[\mathbf{z}],\dots,f_n[\mathbf{z}])$.

6. For $k \in [d]$, we denote with $f_{\le k} := \sum_{\alpha:|\alpha|\le k} c_\alpha \mathbf{x}^\alpha$ the *truncation of* f *of degree* k. For a system $\mathcal{F} = (f_1,\dots,f_n)$, we define the *truncation of* $\mathcal{F}$ *of degree* k as $\mathcal{F}_{\le k} = (f_{1_{\le k}},\dots,f_{n_{\le k}})$.

7.2.2 Error Bounds for Shifting, Truncation, and Rotation

We first collect some bounds on the size of $|f(\mathbf{z})|$ and $\|f[\mathbf{z}]\|$ depending on the modulus of some point $\mathbf{z} \in \mathbb{C}^n$ and the norm $\|f\|$ of some polynomial $f \in \mathbb{C}[\mathbf{x}]$. We also give bounds on the error that occurs when computing $f(\mathbf{z})$ or $f[\mathbf{z}]$ not exactly at $\mathbf{z}$ but at a nearby point ζ.

Lemma 7.2. *Let $f(\mathbf{x}) = \sum_\alpha c_\alpha \mathbf{x}^\alpha \in \mathbb{C}[\mathbf{x}] = \mathbb{C}[x_1,\dots,x_n]$ of total degree d and with $\|f\| \le 2^\tau$. Moreover, let $k \in [d] = \{1,\dots,d\}$, $\mathbf{z} \in \mathbb{C}^n$, and ζ be an approximation of $\mathbf{z}$ with $\|\zeta - \mathbf{z}\| < 2^{-L}$, then it holds:*

(a) $|f_{\le k}(\mathbf{z})| \le \binom{n+k}{k} \cdot 2^\tau \cdot M(\mathbf{z})^k$, and in particular $|f(\mathbf{z})| < \binom{n+d}{d} \cdot 2^\tau \cdot M(\mathbf{z})^d$.

(b) If $\|\mathbf{z}\| \le 1$, then $|f(\mathbf{z}) - f_{\le k}(\mathbf{z})| \le \|\mathbf{z}\|^{k+1} \cdot \binom{n+d}{d} \cdot 2^\tau$.

(c) $|f(\zeta) - f(\mathbf{z})| \leq 2^{\tau-L} \cdot \binom{n+d}{d} \cdot M(\mathbf{z})^d$.

(d) $\|f[\mathbf{z}]\| < d^n \cdot \binom{n+d}{d} \cdot 2^\tau \cdot M(\mathbf{z})^d$ *and* $\|f[\zeta] - f[\mathbf{z}]\| < 2^{\tau-L} \cdot d^n \cdot \binom{n+d}{d} \cdot M(\mathbf{z})^d$.

Proof. Part (a) and (b) follow immediately from the fact that $f_{\leq k}$ has at most $\binom{n+k}{k}$ coefficients and each occurring term $c_\alpha \cdot \mathbf{z}^\alpha$ has absolute value bounded by $2^\tau \cdot \|\mathbf{z}\|^{|\alpha|}$. Part (c) is a direct consequence of [MOS11, Theorem 12], which provides general bounds on the error when evaluating a multivariate polynomial using floating point computation. For the last claim, notice that

$$f[\mathbf{z}] = \sum_{\alpha \in \mathbb{Z}^n_{\geq 0}} \tfrac{\partial^\alpha f(\mathbf{z})}{\alpha!} \mathbf{x}^\alpha \text{ and } f[\zeta] = \sum_{\alpha \in \mathbb{Z}^n_{\geq 0}} \tfrac{\partial^\alpha f(\zeta)}{\alpha!} \mathbf{x}^\alpha,$$

where $\alpha = (\alpha_1, \ldots, \alpha_n)$, $\alpha! := \alpha_1! \cdots \alpha_n!$, and $\partial^\alpha f := \frac{\partial^{|\alpha|} f}{\partial x_1^{\alpha_1} \cdots \partial x_n^{\alpha_n}}$. The polynomials $\frac{\partial^\alpha f}{\alpha!}$ have total degree bounded by d and their norm is upper bounded by $2^\tau \cdot d^n = 2^{\tau + n \log d}$. Hence, Part (a) implies the first part of (d). The second part follows from Part (c) because, for any α, it holds that

$$|\tfrac{\partial^\alpha f(\mathbf{z})}{\alpha!} - \tfrac{\partial^\alpha f(\zeta)}{\alpha!}| \leq 2^{\tau-L} \cdot d^n \cdot \binom{n+d}{d} \cdot M(\mathbf{z})^d. \qquad \square$$

We further provide the following lemma that investigates the influence of considering only an approximation of a polynomial f when looking at shift and truncation.

Lemma 7.3. *Let $f(\mathbf{x}) \in \mathbb{C}[\mathbf{x}]$ be a polynomial of total degree d with norm $\|f\| \leq 2^\tau$, and let $\mathbf{z} \in \mathbb{C}^n$ and ζ such that $\|\zeta - \mathbf{z}\| < 2^{-L}$. Furthermore, let ϕ be an approximation of $f[\zeta]_{\leq k}$ of total degree at most k, with $k \in [d]$, such that $\|\phi - f_i[\zeta]_{\leq k}\| \leq 2^{-(k+1)L}$. Then, for any $\mathbf{x}$ with $\|\mathbf{x}\| \in [2^{-L}, 1]$, it holds*

$$|\phi(\mathbf{x}) - f[\zeta](\mathbf{x})| \leq \|\mathbf{x}\|^{k+1} \cdot d^n 2^{\tau+1} [M(\zeta) \cdot (n+d)^2]^d.$$

Proof. We first observe that using the triangle inequality, simple bounds on the number of monomials of lower ($\leq k$) and higher ($\geq k+1$) degree, and the fact that $\|\mathbf{x}\| \leq 1$ yields

$$\begin{aligned} |\phi(\mathbf{x}) - f[\zeta](\mathbf{x})| &\leq |f[\zeta](\mathbf{x}) - f[\zeta]_{\leq k}(\mathbf{x})| + |\phi(\mathbf{x}) - f[\zeta]_{\leq k}(\mathbf{x})| \\ &\leq \|\mathbf{x}\|^{k+1} \cdot \|f[\zeta] - f[\zeta]_{\leq k}\| \cdot \binom{n+d}{d} + \|\phi - f[\zeta]_{\leq k}\| \cdot \binom{n+k}{k}. \end{aligned}$$

Then, applying Lemma 7.2 part (d) to the left summand and the condition on the approximation $\|\phi - f[\zeta]_{\leq k}\| \leq 2^{-(k+1)L}$, we conclude that

$$\begin{aligned} |\phi(\mathbf{x}) - f[\zeta](\mathbf{x})| &\leq \|\mathbf{x}\|^{k+1} \cdot d^n \cdot \binom{n+d}{d}^2 \cdot 2^\tau \cdot M(\zeta)^d + 2^{-(k+1)L} \cdot \binom{n+k}{k} \\ &\leq \|\mathbf{x}\|^{k+1} \cdot [d^n \cdot 2^\tau \cdot (n+d)^{2d} \cdot M(\zeta)^d + (n+d)^d] \\ &\leq \|\mathbf{x}\|^{k+1} \cdot d^n 2^{\tau+1} [M(\zeta) \cdot (n+d)^2]^d, \end{aligned}$$

where the second to last inequality follows from $\|\mathbf{x}\| \geq 2^{-L}$. $\square$

In our algorithm, we will consider a transformation of the coordinate system induced by a rotation $\mathbf{x} \mapsto S \cdot \mathbf{x}$, where $S \in \mathrm{SO}(n)$ is a rotation matrix with rational entries. The following lemma quantifies the impact of such a rotation on the bit-size of the coefficients of a given polynomial f.

Lemma 7.4. *Let $f = \sum_\alpha c_\alpha \cdot \mathbf{x}^\alpha \in \mathbb{C}[\mathbf{x}]$ be a polynomial of total degree d and $S \in \mathrm{SO}(n)$ be a rotation matrix. Then, $f^* := f \circ S$, it holds that $\|f^*\| \leq 2^{\tau_{\mathcal{F}}} \cdot \binom{n+d}{d}^2$.*

Proof. Notice that each of the entries $a_{r,s}$ of the rotation matrix $S = (a_{r,s})_{r,s}$ has absolute value at most 1. Thus, $f^*(\mathbf{x}) = f \circ S(\mathbf{x}) = \sum_{\alpha:c_\alpha} c_\alpha \cdot [(a_{11}x_1 + \cdots + a_{1,n}x_n)^{\alpha_1} \cdots (a_{n1}x_1 + \cdots + a_{n,n}x_n)^{\alpha_n}]$ has coefficients of absolute value bounded by $2^{\tau_{\mathcal{F}}} \cdot \binom{n+d}{d}^2$ as, when expanding the product $(a_{11}x_1 + \cdots + a_{1,n}x_n)^{\alpha_1} \cdots (a_{n1}x_1 + \cdots + a_{n,n}x_n)^{\alpha_n}$ for a fixed α, there can be at most $\binom{n+d}{d}$ terms contributing to a specific monomial $\mathbf{x}^{\alpha'}$. □

7.2.3 The Hidden-Variable Approach

Let us assume that an arbitrary zero-dimensional system $\mathcal{F} = (f_1, \ldots, f_n)$ as in (7.1) is given. That is, f_i has total degree d_i, $\|f_i\| < 2^{\tau_i}$ for all i, and it is assumed that the total number of solutions of $\mathcal{F} = 0$, also at "infinity" (see the considerations below for an explanation), is finite. We now briefly describe the so-called hidden-variable approach that allows us to project the zeros of the system on an arbitrary coordinate axis. For more details, we recommend the excellent textbook [CLO05] by Cox, Little, and O'Shea.

In a first step, we consider a homogenization of the system, that is, we introduce an additional (homogenizing) variable x_{n+1} and multiply each occurring term in each f_i with a suitable power of x_{n+1} such that the so obtained polynomials $f_i^h \in \mathbb{C}[x_1, \ldots, x_{n+1}]$ are homogenous and of total degree d_i, respectively; see also the example below. Notice that each solution $(x_1, \ldots, x_n) \in \mathbb{C}$ of $\mathcal{F} = 0$ yields a solution $(x_1, \ldots, x_n, 1)$ of the homogenized system

$$\mathcal{F}^h : \quad f_1^h(x_1, \ldots, x_{n+1}) = \ldots = f_n^h(x_1, \ldots, x_{n+1}) = 0 \tag{7.3}$$

In addition, if $(x_1, \ldots, x_{n+1}) \in \mathbb{C}^{n+1}$ is a solution of $\mathcal{F}^h = 0$, then $(t \cdot x_1, \ldots, t \cdot x_{n+1})$ is a zero of $\mathcal{F}^h$ for all $t \in \mathbb{C}$. In particular, if $x_{n+1} \neq 0$, we can set $t = 1/x_{n+1}$, which yields the solution $(x_1/x_{n+1}, \ldots, x_n/x_{n+1})$ of $\mathcal{F} = 0$. It is thus preferable to consider the set S of solutions of the above homogenized system as a set of points in the n-dimensional projective space $\mathbb{P}^n$. The set S then decomposes into the set $S_{<\infty} = \{(x_1 : \ldots : x_{n+1}) \in S : x_{n+1} = 1\}$ of so-called *affine solutions*, for which $x_{n+1} = 1$, and the set $S_\infty = \{(x_1 : \ldots : x_{n+1}) : x_{n+1} = 0\}$ of *solutions at infinity*, for which $x_{n+1} = 0$. Notice that there is a one-to-one correspondence between the affine solutions of the homogenized system and the solutions of the original system (7.1).

As mentioned above, we aim to compute the projections of the solutions of $\mathcal{F} = 0$ on one of the coordinate axis, say w.l.o.g., $x = x_1$. For this, suppose that we fix some value ξ for x_1. Plugging $x_1 = \xi$ into the initial system then yields the specialized system

$$\mathcal{F}^{[\xi]} : \quad f_1^{[\xi]}(x_2, \ldots, x_n) = \ldots = f_n^{[\xi]}(x_2, \ldots, x_n) = 0,$$

with $f_i^{[\xi]}(x_2, \ldots, x_n) := f_i(\xi, x_2, \ldots, x_n)$ and the corresponding homogenized system

$$(\mathcal{F}^{[\xi]})^h : \quad (f_1^{[\xi]})^h(x_2, \ldots, x_{n+1}) = \ldots = (f_n^{[\xi]})^h(x_2, \ldots, x_{n+1}) = 0, \tag{7.4}$$

where $(f_i^{[\xi]})^h$ denotes the homogenization of $f_i^{[\xi]}$. Notice that, in general, $(f_i^{[\xi]})^h$ does not equal $(f_i^h)^{[\xi]} = f_i^h(\xi, x_2, \ldots, x_{n+1})$, that is, we cannot deduce the system in (7.4) from plugging ξ into the homogenized system in (7.3). The reason is that the total degree of f_i may become smaller for certain values for ξ, and thus homogenization does not commute with specialization.

Example. For $f := x_1x_2^3 - 2x_2^3 + x_3x_1 + x_3^2$ and $\xi = 2$, we have $f^h = x_1x_2^3 - 2x_2^3x_4 + x_3x_1x_4^2 + x_3^2x_4^2$, $f^{[\xi]} = f(2, x_2, x_3) = 2x_3 + x_3^2$, and $(f^{[\xi]})^h = 2x_3x_4 + x_3^2$, which does not equal $(f^h)^{[\xi]} =$

$f^h(2, x_2, x_3, x_4) = 2x_2^3 - 2x_2^3x_4 + 2x_3x_4^2 + x_3^2x_4^2.$

You may notice that (7.4) is a polynomial system consisting of n homogenous polynomials in n variables. If the initial homogenized system had a solution with $x_1 = \xi$, then this would yield a solution of (7.4) and vice versa. In other words, ξ would be the projection of a solution of the initial system. The following important result now gives a necessary and sufficient criteria to check whether this is actually the case.

Theorem 7.5 ([CLO05], Chapter 3, Theorems 2.3 and 3.1)**.** *Let $\mathcal{G}$ be a system of n homogeneous polynomials in n variables of total degrees $d_1, \ldots, d_n$. Then, there is a unique polynomial* [3] $\operatorname{Res}(\mathcal{G}) = \operatorname{Res}_{d_1,\ldots,d_n} \in \mathbb{Z}[\mathbf{u}]$ *in the coefficients* $\mathbf{u}$ *of $\mathcal{G}$ such that* $\operatorname{Res}(\mathcal{G}) = 0$ *if and only if* $\mathcal{G}(\mathbf{x}) = 0$ *has a non-trivial solution* $\mathbf{x} \in \mathbb{P}^{n-1}$*. The polynomial* $\operatorname{Res}(\mathcal{G})$ *is homogeneous in the variables of f_i of degree $d_1 \cdots d_{i-1} \cdot d_{i+1} \cdots d_n$ and its total degree equals $\sum_{i=1}^n d_1 \cdots d_{i-1} \cdot d_{i+1} \cdots d_n$.*

Example. The homogeneous system $\mathcal{G} : ax_1^2 + bx_1x_2 + cx_2^2 = dx_1 + ex_2 = 0$ (with general coefficients a, b, c, d, and e) has a solution in $\mathbb{P}^1$ if and only if the involved coefficients fulfill the equality $\operatorname{Res}(\mathcal{G}) = \operatorname{Res}_{2,1} = ae^2 - be + cd = 0$.

For an arbitrary polynomial system $\mathcal{G}$ consisting of $n+1$ (not necessarily homogenous) polynomials in $\mathbb{C}[x_1, \ldots, x_n]$, we simply define $\operatorname{Res}(\mathcal{G}) = \operatorname{Res}(\mathcal{G}^h)$. Since $\mathcal{G}$ has the same coefficients as $\mathcal{G}^h$, it still holds that $\operatorname{Res}(\mathcal{G})$ is a polynomial in the coefficients of $\mathcal{G}$. In addition, since there is a one-to-one correspondence between the solutions of $\mathcal{G}$ and the affine solutions of $\mathcal{G}^h$, it follows that $\operatorname{Res}(\mathcal{G}) = 0$ if and only if $\mathcal{G}^h = 0$ has a solution in $\mathbb{P}^n$.

Now, in order to compute all values ξ such that there exists a solution $(x_1, \ldots, x_n)$ of our initial system $\mathcal{F} = 0$ with $x_1 = \xi$, we aim to apply the above theorem to the system as defined in (7.4), however we now consider ξ as an indeterminate (so called *hidden variable*) rather than a fixed value. There are some subtleties with this approach. In particular, the degrees of the polynomials $f_i^{[\xi]}$ may be different for certain values for ξ, which is crucial as the definition of the resultant polynomial Res strongly depends on the degrees of the given polynomials. However, we can avoid such critical situations if we assume that the given polynomials f_i fulfill some mild prerequisites.

Lemma 7.6. *Suppose that each polynomial f_i contains a term of total degree d_i* that does not depend on x_1 *and write*

$$f_i = \sum_{\alpha=(\alpha_2,\ldots,\alpha_n)} c_{i,\alpha}(x_1) \cdot x_2^{\alpha_2} \cdots x_n^{\alpha_n} \in \mathbb{C}[x_1][x_2, \ldots, x_n]$$

as a polynomial in $x_2, \ldots, x_n$ with coefficients $c_{i,\alpha} \in \mathbb{C}[x_1]$. Furthermore, let

$$F_i := \sum_{\alpha=(\alpha_2,\ldots,\alpha_n)} c_{i,\alpha}(x_1) \cdot x_2^{\alpha_2} \cdots x_n^{\alpha_n} \cdot x_{n+1}^{d_i-\alpha_2-\ldots-\alpha_n}$$

be its corresponding homogenization (with respect to the variables $x_2, \ldots, x_n$), then it holds:

(a) For all $\xi \in \mathbb{C}$, we have $(f_i^{[\xi]})^h = F_i^{[\xi]}$ and $(f_i^{[\xi]})^h$ has total degree d_i.

(b) Each root $x_1 = \xi \in \mathbb{C}$ of $R(x_1) := \operatorname{Res}(F_1, \ldots, F_n)$ yields a solution $(x_1, \ldots, x_n) \in \mathbb{C}^n$ of $\mathcal{F} = 0$ with $x_1 = \xi$ and vice versa.

Proof. Part (a) follows directly from the fact that the total degree of $f_i^{[\xi]}$ is equal to d_i for all ξ as there exists a term of degree d_i that does not depend on ξ. For (b), we first remark that the resultant of the polynomials F_i is a polynomial in the coefficients of the F_i, and thus

[3]We remark that Res only depends on the actual degrees of the polynomials.

a polynomial in x_1. Since the degree of each f_i does not depend on the choice of $x_1 = \xi$, we also have $R(\xi) = \text{Res}(F_1|_{x_1=\xi}, \dots, F_n|_{x_1=\xi})$. Now, let $x_1 = \xi$ be a complex root of R, then according to Theorem 7.5, there must exist a solution $(\xi_2 : \ldots : \xi_{n+1}) \in \mathbb{P}^{n-1}$ of the system $(F_i|_{x_1=\xi})_{i=1,\dots,n}$. In order to prove that this solution is an affine solution (i.e. a solution of $\mathcal{F}$), we assume for contradiction that $\xi_{n+1} = 0$. Plugging $x_{n+1} = 0$ into the polynomials F_i yields

$$F_i|_{x_{n+1}=0} = \sum_{\alpha=(\alpha_2,\dots,\alpha_n):\alpha_2+\dots+\alpha_n=d_i} c_{i,\alpha}(x_1) \cdot x_2^{\alpha_2} \cdots x_n^{\alpha_n}.$$

Hence, each of the terms $c_{i,\alpha}(x_1)$ occurring in the above sum is a constant that does not depend on x_1. Since $(\xi : \xi_2 : \ldots : \xi_n)$ is a solution of the system $F_i|_{x_{n+1}=0}$, we conclude that $(x_1 : \xi_2 : \ldots : \xi_n)$ is a solution of $F_i|_{x_{n+1}=0}$ for any x_1. This contradicts our assumption that $\mathcal{F}$ has only finitely many solutions. It follows that $(\xi_2/\xi_{n+1}, \dots, \xi_n/\xi_{n+1}, 1)$ is a solution of $(F_i|_{x_1=\xi})_{i=1,\dots,n}$, and thus $(\xi, \xi_2/\xi_{n+1}, \dots, \xi_n/\xi_{n+1})$ is a solution of $\mathcal{F} = 0$. For the other direction, let $(\xi_1, \dots, \xi_n)$ be a solution of $\mathcal{F} = 0$, then $(\xi_1 : \ldots : \xi_n : 1)$ is an affine solution of the corresponding homogenized system, and thus $(\xi_2 : \ldots : \xi_n : 1)$ a solution of the system $(F_i|_{x_1=\xi})_{i=1,\dots,n} = 0$. This implies that $R(\xi_1) = 0$. □

Obviously, the above considerations apply for any coordinate (hidden-variable) x_k onto which we aim to project the solutions. The corresponding resultant polynomial $\text{Res}(\mathcal{F}, x_k) \in \mathbb{C}[x_k]$ is called the *hidden-variable resultant with respect to x_k*. The following theorem [BS16] bounds the cost for computing the hidden-variable resultant in the special case where the polynomials f_i have integer coefficients. The technique is based on a method due to Emiris and Pan [EP05] and an asymptotically fast algorithm for determinant computation due to Storjohann [Sto05].

Theorem 7.7 ([BS16, Prop. 1]). *Let $\mathcal{F} = (f_i)_{i=1,\dots,n}$ be a polynomial system with integer polynomials $f_i \in \mathbb{Z}[x_1, \dots, x_n]$ of magnitude (d, τ). There is a Las-Vegas algorithm to compute* $\text{Res}(\mathcal{F}, x_k)$ *in an expected number of bit operations bounded by*[4]

$$\tilde{O}(n^{(n-1)(\omega+1)}(d + \tau)d^{(\omega+2)n-\omega-1}).$$

We further remark that a root ξ of $\text{Res}(\mathcal{F}, x_k)$ might origin from several solutions $\mathbf{z} = (z_1, \dots, z_n)$ of $\mathcal{F} = 0$ sharing the same x_k-coordinate $x_k = \xi$. Under the requirements from Lemma 7.6, it holds that the multiplicity of ξ as a root of $\text{Res}(\mathcal{F}, x_k)$ equals the sum of the multiplicities of all these solutions $\mathbf{z}$. Also, the roots of $\text{Res}(\mathcal{F}, x_k)$ are exactly the projections of the finite solutions onto the x_k-coordinate, and vice versa. Furthermore, if there are no solutions at infinity, then $\text{Res}(\mathcal{F}, x_k)$ has degree $D_{\mathcal{F}}$ as the system has exactly $D_{\mathcal{F}}$ solutions (counted with multiplicity), which are all finite, and the roots of $\text{Res}(\mathcal{F}, x_k)$ are exactly the projections of these solutions onto the x_k-coordinate.

Lemma 7.8. *Let $\mathcal{F} = (f_i)_{i=1,\dots,n}$, with $f_i = \sum_{\alpha:|\alpha|\le d_i} c_{i,\alpha} \cdot \mathbf{x}^\alpha \in \mathbb{C}[\mathbf{x}]$ of total degree d_i, be a polynomial system in n variables $\mathbf{x} = (x_1, \dots, x_n)$ with general coefficients $c_{i,\alpha}$. Consider the decomposition*

$$f_i(\mathbf{x}) = \underbrace{\sum_{\alpha:|\alpha|=d_i} c_{i,\alpha} \cdot \mathbf{x}^\alpha}_{:=f_{i,d_i}(\mathbf{x})} + \underbrace{\sum_{\alpha:|\alpha|<d_i} c_{i,\alpha} \cdot \mathbf{x}^\alpha}_{=:f_{i,<d_i}(\mathbf{x})}$$

of each f_i into a sum of terms of degree d_i and into a sum of terms of degree less than d_i. Then, for any $k \in [n]$, it holds that the leading coefficient $\text{LC}(\text{Res}(\mathcal{F}, x_k))$ *of the (general) hidden variable resultant*

[4] ω denotes the exponent in the complexity of matrix multiplication. The current record bound for ω is $\omega \le 2.3728639$ according to [Gal14]

$\mathrm{Res}(\mathcal{F}, x_k) \in \mathbb{Z}[c_{i,\alpha}][x_k]$ *only depends on the coefficients* $c_{i,\alpha}$ *of* f_{i,d_i} *(i.e. on the coefficients* $c_{i,\alpha}$ *with* $|\alpha| = d_i$*).*

Proof. Let x_{n+1} be a homogenizing variable and

$$f_i^h = \sum_{\alpha:|\alpha|=d_i} c_{i,\alpha} \cdot \mathbf{x}^\alpha + \sum_{\alpha:|\alpha|<d_i} c_{i,\alpha} \cdot \mathbf{x}^\alpha \cdot x_{n+1}^{d_i-|\alpha|}$$

be the corresponding homogenization of f_i. For generic choice of the coefficients $c_{i,\alpha}$ with $|\alpha| = d_i$, the above system is zero-dimensional and has no solution at infinity. Namely, for $x_{k+1} = 0$, the system writes as $f_i^h(x_1, \ldots, x_n, 0) = f_{i,d_i}$, and a generic system of n homogenous polynomials in n variables has no solution. Thus, there exists no solution at infinity, which also rules out the possibility of the system being non zero-dimensional. Now, suppose that the coefficients are generically chosen such that all solutions are finite. Then, the total number of solutions equals the Bézout number $D_{\mathcal{F}}$ and the degree of $\mathrm{Res}(\mathcal{F}, x_k)$ equals $D_{\mathcal{F}}$. According to Theorem 7.5, $\mathrm{LC}(\mathrm{Res}(\mathcal{F}, x_k)) \in \mathbb{Z}[c_{i,\alpha}]$ is a polynomial in the coefficients $c_{i,\alpha}$. Now, if $\mathrm{LC}(\mathrm{Res}(\mathcal{F}, x_k))$ would depend on some coefficient $c_{i,\alpha}$ with $|\alpha| < d_i$, then, for generic choice of all other coefficients, we could choose such a $c_{i,\alpha}$ in a way such that the leading coefficient becomes zero, and thus $\deg \mathrm{Res}(\mathcal{F}, x_k) < D_{\mathcal{F}}$, a contradiction. This shows that, for generic choice of the coefficients $c_{i,\alpha}$, the leading coefficient $\mathrm{LC}(\mathrm{Res}(\mathcal{F}, x_k))$ does not depend on the coefficients of the polynomials $f_{i,<d_i}$. From this, we conclude that $\mathrm{LC}(\mathrm{Res}(\mathcal{F}, x_k))$ does not depend on the coefficients of the polynomials $f_{i,<d_i}$ in general. □

Corollary 7.9. *Let* $\mathcal{F} = (f_i)_{i=1,\ldots,n}$ *be an arbitrary polynomial system as in Lemma 7.8 with* $d_i = d$ *for all* i*, and let*

$$\bar{\mathcal{F}}: \quad \bar{f}_i := \sum_{\alpha:|\alpha|=d+1}^{n} c_{i,\alpha} \cdot \mathbf{x}^\alpha + f_i, \quad \textit{with } c_{i,\alpha} \in \mathbb{Z} \textit{ for all } \alpha \textit{ with } |\alpha| = d+1,$$

be the system obtained by adding polynomials of the form $\sum_{\alpha:|\alpha|=d+1}^{n} c_{i,\alpha} \cdot \mathbf{x}^\alpha$ *to each* f_i*. If* $\bar{\mathcal{F}}$ *does not have any solution at infinity (which is the case for generic choice of the coefficients* $c_{i,\alpha}$*), then it holds that* $\mathrm{LC}(\mathrm{Res}(\bar{\mathcal{F}}, x_k)) \in \mathbb{Z}_{\neq 0}$.

Proof. If $\bar{\mathcal{F}}$ has no solution at infinity, then $\bar{\mathcal{F}}$ is zero-dimensional and, in addition, $\mathrm{Res}(\bar{\mathcal{F}}, x_k)$ has degree $D_{\bar{\mathcal{F}}} = (d+1)^n$. From Lemma 7.8, we further conclude that $\mathrm{LC}(\mathrm{Res}(\bar{\mathcal{F}}, x_k))$ only depends on the coefficients $c_{i,\alpha}$ of the degree $(d+1)$-parts $\bar{f}_{i,d+1}$ of the polynomials $\bar{f}_i$. Hence, we have $\mathrm{LC}(\mathrm{Res}(\bar{\mathcal{F}}, x_k)) \in \mathbb{Z}_{\neq 0}$. □

Example: Let $\bar{f}_i = \sum_{j=1}^n a_{ij} \cdot x_j^{d+1} + f_i$ with $f_i \in \mathbb{C}[\mathbf{x}]_{\leq d}$ polynomials of total degree at most d. Then, it holds that

$$\mathrm{Res}(\bar{\mathcal{F}}, x_k) = \pm \det(a_{ij}) \cdot x_k^{(d+1)^n} + \cdots$$

Namely, if $\det(a_{i,j}) \neq 0$, then $\bar{\mathcal{F}}$ has no solution at infinity as each such solution would yield a non-trivial solution of the linear system $\sum_{j=1}^n a_{i,j} \cdot X_j = 0$. Thus, $\bar{\mathcal{F}}$ is zero-dimensional in this case and $\mathrm{Res}(\bar{\mathcal{F}}, x_k)$ has degree $D_{\bar{\mathcal{F}}} = (d+1)^n$. From Lemma 7.8, we further conclude that $\mathrm{LC}(\mathrm{Res}(\bar{\mathcal{F}}, x_k))$ only depends on the coefficients $a_{i,j}$ of the degree $(d+1)$-parts $\bar{f}_{i,d+1}$ of the polynomials $\bar{f}_i$. Hence, we have $\mathrm{LC}(\mathrm{Res}(\bar{\mathcal{F}}, x_k)) = \mathrm{LC}(\mathrm{Res}(\bar{f}_{1,=d+1}, \ldots, \bar{f}_{n,=d+1}, x_k))$, and using Theorem 2.3 and Theorem 3.5 in [CLO05] further shows that

$$\begin{aligned} \mathrm{Res}(\bar{f}_{1,=d+1}, \ldots, \bar{f}_{n,=d+1}, x_k) &= \det(a_{i,j})^{(d+1)^n} \cdot \mathrm{Res}((x_1^{d+1}, \ldots, x_n^{d+1}), x_k) \\ &= \pm \det(a_{i,j})^{(d+1)^n} \cdot x_k^{(d+1)^n}. \end{aligned}$$

It is also well known (e.g. this follows from Theorem 7.10 below) that $\text{Res}(\mathcal{F}, x_k)$ is contained in the ideal $\mathcal{I} := \langle f_1, \dots, f_n\rangle$ defined by the polynomials $f_1, \dots, f_n$. In particular, for polynomials $f_i \in \mathbb{Z}[x_1, \dots, x_n]$ with integer coefficients, this guarantees the existence of an integer λ, with $\lambda \neq 0$, and polynomials $g_i \in \mathbb{Z}[x_1, \dots, x_n]$ with

$$\lambda \cdot \text{Res}(\mathcal{F}, x_k) = g_1 \cdot f_1 + \cdots g_n \cdot f_n. \tag{7.5}$$

Recent work [DKS13] allows us to bound the magnitude of the polynomials g_i as well as the size of λ. For this, we first write $f_i = \sum_\alpha c_{i,\alpha}(x_k)\mathbf{x}_{\neq k}^\alpha$ as a polynomial in $\mathbf{x}_{\neq k}$ with coefficients $c_{i,\alpha} \in \mathbb{Z}[x_k]$, where $\mathbf{x}_{\neq k}$ denotes all but the k'th variable. We further introduce a variable $u_{i,\alpha}$ for every coefficient polynomial $c_{i,\alpha}$. Let $\mathbf{u}_i = (u_{i,\alpha})_\alpha$ be the variables corresponding to the polynomial f_i, and let $\mathbf{u} = (\mathbf{u}_1, \dots, \mathbf{u}_n)$ denote the variables for all polynomials. Then, $\mathcal{F}$ can be considered as a system consisting of n polynomials in $n-1$ variables $\mathbf{x}_{\neq k}$ with coefficients $\mathbf{u}$. Thus, its resultant $\text{Res}(\mathcal{F})$ is a polynomial in $\mathbb{Q}[\mathbf{u}]$, which is further contained in the ideal $\langle f_1, \dots, f_n\rangle \subset \mathbb{Q}[\mathbf{u}, \mathbf{x}_{\neq k}]$. The following theorem, which is a consequence of Theorem 4.28 in [DKS13] (see also [DKS13, pp. 6]), gives bounds on the degree and height of the polynomials in the cofactor-representation of $\text{Res}(\mathcal{F})$ in this ideal.

Theorem 7.10 ([DKS13] Consequence of Theorem 4.28)**.** *Given a polynomial system $\Phi = (\varphi_1, \dots, \varphi_{n+1})$ with $\varphi_i = \sum_\alpha u_{i,\alpha}\mathbf{x}^\alpha \in \mathbb{Z}[\mathbf{u}, \mathbf{x}]$ of total degree d_i in $\mathbf{x} = (x_1, \dots, x_n)$. Then, for any $k \in [n]$, there exists a $\lambda \in \mathbb{Z}_{\neq 0}$ and polynomials $\gamma_i \in \mathbb{Z}[\mathbf{u}, \mathbf{x}]$ such that*

$$\begin{aligned}\lambda \cdot \text{Res}(\Phi) &= \sum_{i \in [n+1]} \gamma_i \cdot \varphi_i,\\ \deg_{\mathbf{u}_j}(\gamma_i\varphi_i) &\leq \prod_{\ell \neq j} d_\ell \quad \textit{and}\\ \tau_{\lambda\,\text{Res}(\Phi, x_k)}, \tau_{\gamma_i} &\leq (6n+10)\log(n+3)D_\Phi \quad \textit{for } j \in [n] \textit{ and } i \in [n+1],\end{aligned}$$

where τ_p denotes the bit-size of a polynomial $p \in \mathbb{Z}[\mathbf{u}, \mathbf{x}]$.

We can now derive bounds on the degree and the bit-sizes of the polynomials g_i as well as on the bit-size of λ in (7.5) from the above theorem:

Corollary 7.11. *Given a zero-dimensional polynomial system $\mathcal{F} = (f_1, \dots, f_n)$ with polynomials $f_i \in \mathbb{C}[\mathbf{x}]$, we can explicitly compute (see (7.6) and (7.7)) positive integers $A_\mathcal{F}$ and $B_\mathcal{F}$, with $A_\mathcal{F} = \tilde{O}(nD_\mathcal{F})$ and $B_\mathcal{F} = \tilde{O}(n \cdot D_\mathcal{F} + \tau_\mathcal{F} \cdot \max_i \frac{D_\mathcal{F}}{d_i})$, such that there exists an integer $\lambda \in \mathbb{Z}_{\neq 0}$ and polynomials $g_i \in \mathbb{C}[\mathbf{x}]$ with*

$$\begin{aligned}|\lambda| &\leq 2^{A_\mathcal{F}},\\ \deg g_i &\leq D_\mathcal{F},\ \tau_{g_i} \leq B_\mathcal{F},\ \textit{and}\\ \lambda \cdot \text{Res}(\mathcal{F}, x_k) &= \sum_{i=1}^n g_i \cdot f_i.\end{aligned}$$

If all polynomials f_i have only integer coefficients, then we may further assume that the polynomials g_i have only integers coefficients as well.

Proof. For each $i \in [n]$, write $f_i(\mathbf{x}) = \sum_\alpha u_{i,\alpha}\mathbf{x}_{\neq k}^\alpha$ as a polynomial in the variables $\mathbf{x}_{\neq k}$ and with coefficients $u_{i,\alpha} \in \mathbb{C}[x_k]$. Theorem 7.10 now guarantees the existence of a positive $\lambda \in \mathbb{Z}_{\neq 0}$ and polynomials $g_i \in \mathbb{Z}[\mathbf{u}, \mathbf{x}_{\neq k}]$ with $\lambda \cdot \text{Res}(\mathcal{F}, x_k) = \sum_{i\in[n]} g_i \cdot f_i$. Notice that since $\mathbf{u}$ only depends on x_k, we may consider each g_i as an element in $\mathbb{C}[\mathbf{x}]$. In addition, we have $\deg_{\mathbf{u}_j}(g_if_i) \leq \prod_{\ell\neq j} d_\ell$, and thus $\deg_\mathbf{x}(g_i) \leq \deg_\mathbf{x}(g_if_i) \leq D_\mathcal{F} = d_1 \cdots d_n$ as each $u_{j,\alpha}$ has degree bounded by d_j. We can now write each polynomial g_i as $g_i = \sum_{\alpha:|\alpha|\le D_{\mathcal{F}}} P_{i,\alpha}(\mathbf{u}) \cdot \mathbf{x}_{\neq k}^{\alpha}$ with polynomials $P_{i,\alpha} = \sum_{\beta=(\beta_1,\ldots,\beta_N):|\beta|\le D_{\mathcal{F}}/d_i} c_{i,\alpha,\beta} \cdot \mathbf{u}^{\beta}$. From Theorem 7.10, we conclude that $c_{i,\alpha,\beta}$ are integers of absolute value $|c_{i,\alpha,\beta}| < 2^{A_{\mathcal{F}}}$, where

$$A_{\mathcal{F}} := \lceil (6n+4)\log(n+2)D_{\mathcal{F}} \rceil = O\big(nD_{\mathcal{F}}\log(n)\big). \tag{7.6}$$

In addition, $N \le \sum_{i=1}^{n} \binom{d_i+n}{d_i} \le n \cdot \binom{d_{\mathcal{F}}+n}{d_{\mathcal{F}}} \le n(d_{\mathcal{F}}+n)^n$ denotes the number of distinct coefficients $u_{i,\alpha}$. Further notice that, for each β, $\mathbf{u}^{\beta}$ is a product of at most $D_{\mathcal{F}}$ univariate polynomials in $\mathbb{C}[x_k]$, each of degree at most $d_{\mathcal{F}}$ and of norm bounded by $2^{\tau_{\mathcal{F}}}$. Hence, it can be written as a sum of at most $(d_{\mathcal{F}}+1)^{D_{\mathcal{F}}}$ terms, each of absolute value at most $2^{\tau_{\mathcal{F}}\cdot D_{\mathcal{F}}/d_i}$. We conclude that the norm of g_i is bounded by

$$\begin{aligned}\binom{D_{\mathcal{F}}+N}{D_{\mathcal{F}}} \cdot (d_{\mathcal{F}}+1)^{D_{\mathcal{F}}} \cdot 2^{A_{\mathcal{F}}} \cdot 2^{\tau_{\mathcal{F}}\cdot D_{\mathcal{F}}/d_i} &\le [(n(d_{\mathcal{F}}+n)^n + D_{\mathcal{F}}) \cdot (d_{\mathcal{F}}+1)]^{D_{\mathcal{F}}} \cdot 2^{A_{\mathcal{F}}} \cdot 2^{\tau_{\mathcal{F}}\cdot D_{\mathcal{F}}/d_i} \\ &\le [(n+1)(d_{\mathcal{F}}+n)^{n+1}]^{D_{\mathcal{F}}} \cdot 2^{A_{\mathcal{F}}} \cdot 2^{\tau_{\mathcal{F}}\cdot D_{\mathcal{F}}/d_i} \\ &\le [(d_{\mathcal{F}}+n)^{6n+4} \cdot 2^{A_{\mathcal{F}}} \cdot 2^{\tau_{\mathcal{F}}\cdot D_{\mathcal{F}}/d_i} \le 2^{B_{\mathcal{F}}},\end{aligned}$$

where we define

$$B_{\mathcal{F}} := 2 \cdot \lceil D_{\mathcal{F}} \cdot (6n+4)\log(d_{\mathcal{F}}+n) \rceil + \tau_{\mathcal{F}} \cdot \max_i \frac{D_{\mathcal{F}}}{d_i} = \tilde{O}(n \cdot D_{\mathcal{F}} + \tau_{\mathcal{F}} \cdot \max_i \frac{D_{\mathcal{F}}}{d_i}). \tag{7.7}$$

The final claim follows from the fact that $f_i \in \mathbb{Z}[\mathbf{x}]$ for all i implies that $u_{i,\alpha} \in \mathbb{Z}[x_k]$ for all i, α, and thus $g_j \in \mathbb{Z}[\mathbf{x}]$ for all j. □

7.2.4 Generic Position via Rotation

In the previous subsection, we have outlined how to project the solutions of a polynomial onto one of the coordinate axis. One subtlety of the approach was that certain mild conditions on the input polynomials need to be fulfilled in order to guarantee that the roots of the hidden variable resultant are exactly the projections of the (finite) solutions of the initial system; see Lemma 7.6. Another drawback of the approach is that distinct solutions might be projected onto the same point or onto two very nearby points on the coordinate axis, that is, the actual distance between distinct solutions is no longer preserved after the projection. We will show how to address these issues by using a random rotation of the coordinate system. We first start with the special case of dimension 2.

Lemma 7.12. *Let $p_\ell = (x_\ell, y_\ell) \in \mathbb{C}^2$ be N points such that $\|p_\ell\| \neq 0$ for all $\ell = 1,\ldots,N$. Let k be chosen uniformly at random from $[2^L]$. Then, with probability at least $1 - \frac{N}{2^L}$, for each point*

$$p'_\ell = \begin{pmatrix} x'_\ell \\ y'_\ell \end{pmatrix} := S_k(L) \cdot \begin{pmatrix} x_\ell \\ y_\ell \end{pmatrix}, \quad \textit{with} \quad S_k(L) := \begin{pmatrix} \frac{1-(k\cdot 2^{-L})^2}{1+(k\cdot 2^{-L})^2} & -\frac{2\cdot(k\cdot 2^{-L})}{1+(k\cdot 2^{-L})^2} \\ \frac{2\cdot(k\cdot 2^{-L})}{1+(k\cdot 2^{-L})^2} & \frac{1-(k\cdot 2^{-L})^2}{1+(k\cdot 2^{-L})^2} \end{pmatrix} \in \mathrm{SO}(2),$$

it holds that $\min(|x'_\ell|, |y'_\ell|) > 2^{-(L+2)} \cdot \|p_\ell\|$ *for all ℓ.*

Proof. Notice that each matrix $S_k(L)$ is a rotation matrix with respect to the angle $\phi_k \in [0, \pi/2]$ with $\cos\phi_k = \frac{1-(k\cdot 2^{-L})^2}{1+(k\cdot 2^{-L})^2}$ and $\sin\phi_k = \frac{2\cdot(k\cdot 2^{-L})}{1+(k\cdot 2^{-L})^2}$. We further note that the function $h(t) = (\frac{1-t^2}{1+t^2}, \frac{2t}{1+t^2})$ describes the trace of a point on the quarter-circle. Moreover, we have $\dot{h}(t) = (\frac{-4t}{(1+t^2)^2}, \frac{-2t^2+2}{(1+t^2)^2})$, and since $|\dot{h}(t)| = \frac{2}{1+t^2}$ is a decreasing function in t, it follows that the difference between two consecutive angles ϕ_{k+1} and ϕ_k is decreasing in k. We thus conclude that all differences are lower bounded by $\phi_{2^L} - \phi_{2^L-1} = \int_{1-2^{-L}}^{1} \frac{2}{1+t^2}dt \geq \int_{1-2^{-L}}^{1} dt = 2^{-L}$. Now, let $L_k \subset \mathbb{R}^2$ be the line passing through the origin and the point $(\cos\phi_k, \sin\phi_k)$, and let $L_k^\perp \subset \mathbb{R}^2$ be line that passes through the origin and is orthogonal to L_k. In addition, for each point $p_\ell = (\Re(x_\ell) + \mathbf{i}\cdot\Im(x_\ell), \Re(y_\ell) + \mathbf{i}\cdot\Im(y_\ell))$, we define

$$\bar{p}_\ell = \begin{cases} (\Re(x_\ell), \Re(y_\ell)) & \text{if } \Re(x_\ell)^2 + \Re(y_\ell)^2 \geq \Im(x_\ell)^2 + \Im(y_\ell)^2 \\ (\Im(x_\ell), \Im(y_\ell)) & \text{otherwise.} \end{cases}.$$

Then, $\bar{p}_\ell$ is a point in $\mathbb{R}^2$ with $\|\bar{p}_\ell\|_2 \geq \|p_\ell\|/\sqrt{2}$. Let $\Delta_\ell \subset \mathbb{R}^2$ be the disc centered at $\bar{p}_\ell$ of radius $r_\ell = 2^{-L-2}\cdot\|p_\ell\|$. Let $q, r \in \Delta_\ell$ be any two points in Δ_ℓ and α be the angle at the origin of the triangle given by the origin and the points q and r. Then, it holds that

$$\alpha \leq 2\cdot\arctan\left(\frac{2^{-L-2}\cdot\|p_\ell\|}{\|p_\ell\|/\sqrt{2}}\right) < 2\arctan(2^{-L-1}) < 2\cdot 2^{-L-1} \leq 2^{-L}.$$

Since the angle between any two distinct lines L_k and $L_{k'}$ is lower bounded by 2^{-L}, it thus follows that there can be at most one k such that L_k or $L_k^\perp$ intersects Δ_ℓ. Hence, if we pick a $k \in \{1, \dots, 2^L\}$ uniformly at random and choose L_k and $L_k^\perp$ as the axis of the coordinate system obtained by rotating the initial system by ϕ_k, then, with probability at least $1 - \frac{N}{2^L}$, the new coordinates $(\bar{x}'_\ell, \bar{y}'_\ell)$ of each point $\bar{p}_\ell$ will meet the condition that $\min(|\bar{x}'_\ell|, |\bar{y}'_\ell|) > 2^{-L-2}\cdot\|p_\ell\|$. Hence, the same holds true for the points $S_k(L)\cdot p_\ell$. □

We now turn to the general n-dimensional case. For integers k and L and distinct indices $i, j \in \{1, \dots, n\}$, we define

$$S_k^{[ij]}(L) := \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{1-(k\cdot 2^{-L})^2}{1+(k\cdot 2^{-L})^2} & \cdots & -\frac{2\cdot(k\cdot 2^{-L})}{1+(k\cdot 2^{-L})^2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{2\cdot(k\cdot 2^{-L})}{1+(k\cdot 2^{-L})^2} & \cdots & \frac{1-(k\cdot 2^{-L})^2}{1+(k\cdot 2^{-L})^2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} \in SO(n), \tag{7.8}$$

to be a rotation matrix that operates on the i-th and j-th coordinate only. We further define the set of rotation matrices

$$\mathcal{S}_N := \left\{ \prod_{i,j\in[n]^2 : i<j} S_{k_{ij}}^{[ij]}(L) : k_{ij} \in [2^L] \text{ for all } i, j \right\}, \text{ where } L := 4\lceil\log(2n^2N)\rceil. \tag{7.9}$$

Lemma 7.13. *Let N be a positive integer and $\mathbf{p}_\ell \in \mathbb{C}^n$ be N', with $N' \leq N$, points such that $\|\mathbf{p}_\ell\| \neq 0$ for all $\ell = 1, \dots, N'$. $\mathcal{S}_N$ and L are defined as in (7.9). Then, it holds*

(a) Choosing integers $k_{ij} \in [2^L]$ for every pair i, j uniformly at random yields, with probability at least 3/4, a rotation matrix $S \in \mathcal{S}_N$ such that, for each point $\mathbf{p}'_\ell := S(L)\cdot\mathbf{p}_\ell$, it holds that $\min_i |p'_{\ell,i}| \geq (2n^2N)^{-16n}\cdot\|\mathbf{p}_\ell\|$.

(b) There is an integer λ of bit-size $\tilde{O}(n^2\log N)$ such that the entries of λS and λS^{-1} are integer numbers of bit-size $\tilde{O}(n^2\log N)$ as well.

Proof. The proof follows almost immediately from Lemma 7.12. Namely, with probability at least $1 - N/2^L$, both entries $p'_{\ell,i}$ and $p'_{\ell,j}$ of each point $\mathbf{p}'_\ell := S^{[ij]}_{k_{ij}}(L) \cdot \mathbf{p}_\ell$ will have absolute value at least $2^{-(L+2)} \cdot \max(|p_{\ell,i}|, |p_{\ell,j}|)$. Since at least one of the coordinates of $\mathbf{p}_\ell$ has absolute value $\|\mathbf{p}_\ell\|$, we conclude that, with probability $(1-N/2^L)^{\binom{n}{2}} > (1-N/2^L)^{\frac{n^2}{2}} > 1 - \frac{n^2/2}{2^L/N} > 3/4$, each coordinate of each point $\mathbf{p}'_\ell = S(L) \cdot \mathbf{p}_\ell$ has absolute value at least

$$2^{-n\cdot(L+2)} \cdot \|\mathbf{p}_\ell\| \geq 2^{-n\cdot(4(\log(2n^2N)+1)+2)} \cdot \|\mathbf{p}_\ell\| \geq (2^{16\log(2n^2N)})^{-n} \cdot \|\mathbf{p}_\ell\| = (2n^2N)^{-16n} \cdot \|\mathbf{p}_\ell\|.$$

It remains to show the existence of an integer λ of bit-size $\tilde{O}(n^2 \log N)$ such that the entries of λS and λS^{-1} are of that bit-size as well. Each entry of a matrix $S^{[ij]}_{k_{ij}}(L)$ is rational number with denominator $2^{2L} + k^2_{ij}$ of bit-size $O(L)$. The matrix S is a product of $O(n^2)$ many such matrices, thus for $\lambda = \prod_{i,j\in[n]^2:i<j}(2^{2L}+k^2_{ij}) \leq (2^{2L+1})^{n^2} = 2^{\tilde{O}(n^2\log N)}$ it holds that λS is integer. Notice that S is contained in $\mathrm{SO}(n)$, which implies that its entries have absolute value at most 1. It thus follows that the integer entries of λS are of bit-size $\tilde{O}(n^2 \log N)$ as well. In addition, the inverse of $S^{[ij]}_{k_{ij}}(L)$ is simply given by $S^{[ij]}_{-k_{ij}}(L)$, and thus $S^{-1} = \prod_{i,j\in[n]^2:i<j} S^{[ij]}_{-k_{ij}}(L)$, which yields comparable bounds for the entries of S^{-1} as for S. □

We will later make use of the above result when considering the set of non-zero solutions of a polynomial system $\mathcal{F} = 0$. In general, some of these solutions might project (via resultant computation with respect to some variable x_k) onto zero or onto values close to zero. However, in our algorithm, we are aiming for projections that are of comparable size as the size of the corresponding solutions. In order to achieve this, we first consider a random rotation of the system given by some rotation matrix S from the set $\mathcal{S}_N$, with $N := D_{\mathcal{F}}$ the Bézout bound on the total number of solutions. This yields the "rotated system" $\mathcal{F}' := \mathcal{F} \circ S^{-1}$ whose solutions are exactly the rotations of the initial solutions by means of the rotation matrix S. Then, with high probability, each of the coordinates of the solutions of $\mathcal{F}' = 0$ are of absolute value comparable to the norm of the solutions of $\mathcal{F} = 0$. In addition, it is also likely that the rotated system fulfills the condition from Lemma 7.6 for each coordinate.

Lemma 7.14. *Let $\mathcal{F} = (f_1, \ldots, f_n)$ be a polynomial system as in* (7.1)*, $S \in \mathcal{S}_{D_{\mathcal{F}}}$ be a randomly chosen matrix, and let $\mathcal{F}' := \mathcal{F} \circ S^{-1}$ be the corresponding rotated system. Then, with probability larger than* $1/2$*, it holds:*

(a) For each $k \in [n]$, each of the polynomials $f_i \circ S^{-1} \in \mathcal{F}'$ contains a monomial of degree d_i that does not depend on x_k.

(b) For each solution $\mathbf{z} \in \mathbb{C}^n \backslash 0$ of $\mathcal{F} = 0$, it holds that $\min_i |z'_i| \geq (2n^2 D_{\mathcal{F}})^{-16n} \cdot \|\mathbf{z}\|$*, where $\mathbf{z}' = (z'_1, \ldots, z'_n) := S \cdot \mathbf{z}$ is the corresponding (rotated) solution of $\mathcal{F}' = 0$.*

Proof. Since $D_{\mathcal{F}}$ constitutes an upper bound on the number of solutions of $\mathcal{F} = 0$, it follows from Lemma 7.13 (with $N = \mathcal{D}_{\mathcal{F}}$) that, with probability at least 3/4, the inequality in (b) is fulfilled. It thus suffices to prove that, with probability larger than 2/3, the condition in (a) is fulfilled for each coordinate x_k. For this, let

$$f(\mathbf{x}) = \sum_\alpha c_\alpha \mathbf{x}^\alpha \in \mathbb{C}[x_1, \ldots, x_n]$$

be a polynomial of total degree d, and let $S(\mathbf{k})^{-1} = (a_{rs}(\mathbf{k}))_{rs}$ be the matrix depending on the values $\mathbf{k} := (k_{ij})_{i,j}$. Notice that each entry $a_{rs}(\mathbf{k})$ is a rational function in $\mathbf{k}$ with numerators and denominators of total degree (in $\mathbf{k}$) at most $2n^2$. Further notice that $S(\mathbf{k})^{-1}$ maps the point

$(1, 0, \dots, 0)$ to the first column of $S(\mathbf{k})^{-1}$ and that a full-dimensional subset T of the strictly positive part $\{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\|_2 = 1 \text{ and } \mathbf{x} > 0\}$ of the $(n-1)$-dimensional sphere $\mathbb{S}^{n-1} \subset \mathbb{R}^n$ is reached via a suitable choice of $\mathbf{k} \in \mathbb{R}^{n^2}$. Composing f and $S(\mathbf{k})^{-1}$ now yields

$$F(\mathbf{x}, \mathbf{k}) = \sum_{\alpha=(\alpha_1,\dots,\alpha_n)} c_\alpha \cdot (a_{11}(\mathbf{k}) \cdot x_1 + \cdots + a_{1n}(\mathbf{k}) \cdot x_n)^{\alpha_1} \cdots (a_{n1}(\mathbf{k}) \cdot x_1 + \cdots + a_{nn}(\mathbf{k}) \cdot x_n)^{\alpha_n},$$

and the coefficient $C(\mathbf{k})$ of the monomial x_1^d is thus given by

$$C(\mathbf{k}) = \sum_{\alpha:|\alpha|=d} c_\alpha \cdot a_{11}(\mathbf{k})^{\alpha_1} \cdots a_{n1}(\mathbf{k})^{\alpha_n}.$$

We first argue that $C(\mathbf{k})$ does not vanish identically. Let $\hat{f} := \sum_{\alpha:|\alpha|=d} c_\alpha \cdot x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ be the corresponding homogenous polynomial of degree d such that $\hat{f}(a_{11}(\mathbf{k}), \dots, a_{n1}(\mathbf{k})) = C(\mathbf{k})$. Assume that $C(\mathbf{k}) = 0$ for all $\mathbf{k}$, then this implies that $\hat{f}$ vanishes on each point in T. Since the vanishing set of any non-zero homogenous polynomial in n variables has dimension at most $n-2$, we conclude that $\hat{f}$ is the zero-polynomial, and thus $c_\alpha = 0$ for all coefficients of $\hat{f}$. This contradicts our assumption on f.

Hence, it follows that $C(\mathbf{k})$ is a non-zero rational function in $\mathbf{k}$. In addition, each term $c_\alpha \cdot a_{11}(\mathbf{k})^{\alpha_1} \cdots a_{n1}(\mathbf{k})^{\alpha_n}$ has a numerator of total degree at most $2n^2 d$ in $\mathbf{k}$ and a denominator of the form $\prod_{i,j}(2^{2L} + k_{i,j}^2)^{e_{i,j}}$, with $e_{i,j} \in \mathbb{N}$, of degree at most $2n^2 d$ in $\mathbf{k}$. This shows that $C(\mathbf{k})$ can be written as a rational function in $\mathbf{k}$ of total degree $2n^2 d + n^4 d \le 2n^4 d$ as $\prod_{i,j=1:i<j}^{n}(2^{2L} + k_{i,j}^2)^{n^2 d}$ constitutes a common denominator of all terms. According to the Schwartz-Zippel lemma, we thus conclude that choosing $k_{i,j}$ uniformly at random from $\{1, \dots, 2^{4\lceil \log(2n^2 D_{\mathcal{F}})\rceil}\}$ guarantees with probability at least $\rho := 1 - 2n^4 d \cdot 2^{-4\lceil \log(2n^2 D_{\mathcal{F}})\rceil}$ that $C(\mathbf{k}) \neq 0$. In the case where $f = f_i$ is one of the polynomials from $\mathcal{F}$, we thus obtain a probability of at least

$$\rho_i := 1 - 2n^4 d_i \cdot 2^{-4\lceil \log(2n^2 D_{\mathcal{F}})\rceil} \ge 1 - \frac{2n^4 d_i}{(2n^2 D_{\mathcal{F}})^4} \ge 1 - \frac{1}{8n^4}$$

such that f_i contains a term of the form $c \cdot x_1^{d_i}$ with a non-zero constant c. Since the same argument applies to any variable x_k and to any of the n polynomials f_i, the claim follows. □

From the above lemma, we conclude that by choosing a suitably random rotation matrix from the set $\mathcal{S}_{D_{\mathcal{F}}}$, we can ensure with high probability that there is a one-to-one correspondence between the (finite) solutions of $\mathcal{F}$ and the roots of the resultant polynomial $\operatorname{Res}(\mathcal{F}, x_k)$, which are the projections of the solutions on the x_k-axis. In addition, the absolute value of each projection compares well to the absolute value of the corresponding solution. In what follows, we will use the following definition of the set of *admissible rotation matrices with respect to a given system* $\mathcal{F}$, i.e., matrices $S \in \mathcal{S}_{D_{\mathcal{F}}}$ such that the statements (a) and (b) from the above Lemma 7.14 hold.

Definition 7.15. (Admissible Matrices) *For a given polynomial system $\mathcal{F}$ we say that a rotation matrix $S \in \mathcal{S}_{D_{\mathcal{F}}}$ is* admissible with respect to $\mathcal{F}$ *if the statements* (a) *and* (b) *from Lemma 7.14 hold. We further denote by*

$$\mathcal{S}_{\mathcal{F}} := \{S \in \mathcal{S}_{D_{\mathcal{F}}} : S \textit{ is admissible with respect to } \mathcal{F}\} \subset \mathcal{S}_{D_{\mathcal{F}}}$$

the set of admissible matrices with respect to $\mathcal{F}$.

Notice that, even though it is difficult (probably as difficult as computing all solutions of $\mathcal{F}$) to determine whether a certain matrix in $\mathcal{S}_{D_\mathcal{F}}$ is admissible with respect to $\mathcal{F}$, the previous lemma shows that at least half of the matrices in $\mathcal{S}_{D_\mathcal{F}}$ are admissible.

7.3 The Algorithm

We first sketch our algorithm #PolySol and then prove its correctness. We refer the reader to the pseudo-code in Algorithm 7.1 for details regarding #PolySol. The algorithm can be roughly split into 3 main steps:

Step 1: Shifting and Truncation. Given a polynomial system $\mathcal{F} := (f_1, \ldots, f_n)$, a polydisc $\mathbf{\Delta} = \mathbf{\Delta}_r(\mathbf{m})$, and an integer $K \in \{0, \ldots, d_\mathcal{F}\}$, we define a "precision"

$$L := \left\lceil \log \frac{r}{32n(K+1)^n} \right\rceil.$$

Then, in a first step, we compute a $(K+1) \cdot L$-bit approximation

$$\Phi'(\mathbf{x}) := (\phi'_1, \ldots, \phi'_n), \text{ with } \phi'_i \in \mathbb{Q}[\mathbf{x}]_{\leq K},$$

of $\mathcal{F}[\mathbf{m}]_{\leq K}(\mathbf{x})$, i.e., we compute ϕ'_i such that $\|\phi'_i - f_i[\mathbf{m}]_{\leq K}\| < 2^{-(K+1)L}$ and $2^{(K+1)L} \cdot \phi'_i \in \mathbb{Z}[\mathbf{x}]$ for all i. Recall that the centered polynomial system $\mathcal{F}[\mathbf{m}](\mathbf{x})$ was defined as

$$\mathcal{F}[\mathbf{m}](\mathbf{x}) := (f_1[\mathbf{m}](\mathbf{x}), \ldots, f_n[\mathbf{m}](\mathbf{x})) = (f_1(\mathbf{m}+\mathbf{x}), \ldots, f_n(\mathbf{m}+\mathbf{x})),$$

for $\mathbf{m} \in \mathbb{C}^n$ and that the truncation

$$\mathcal{F}[\mathbf{m}]_{\leq K}(\mathbf{x}) := (f_1[\mathbf{m}]_{\leq K}, \ldots, f_n[\mathbf{m}]_{\leq K}),$$

of the centered system $\mathcal{F}[\mathbf{m}]$, is defined by simply omitting all terms of $f_i[\mathbf{m}](\mathbf{x})$ of total degree more than K, as defined in Section 7.2.1. We further define

$$\Phi(\mathbf{x}) := (\phi_1, \ldots, \phi_n), \text{ with } \phi_i := x_i^{K+1} + \phi'_i,$$

the system obtained by adding the term x_i^{K+1} of degree $K+1$ to the polynomial ϕ'_i. This step seems to be odd at first sight, however, it ensures certain properties of Φ. In particular, Φ is guaranteed to have no zeros at infinity (and thus being zero-dimensional as well) according to Corollary 7.9 and our considerations in the corresponding example. This further implies that $\Phi = 0$ has exactly $D_\Phi = (K+1)^n$ finite solutions counted with multiplicity. Also, our choice of Φ allows us to bound the leading coefficient of $\operatorname{Res}(\Phi, x_\ell)$ for all $\ell = 1, \ldots, n$, which turns out to be useful in the analysis of our approach.

Remark. In practice, the latter step does not seem to be necessary in most cases, and thus we recommend to simply proceed with $\Phi := \Phi'$ and to check Φ' for being zero-dimensional. Also, when implementing our algorithms, we observed that proceeding with Φ' instead of Φ only improves the overall performance.

Step 2: Solving Φ***.*** We will later prove that, under the assumption that L is sufficiently large (or equivalently $\mathbf{\Delta}$ is sufficiently small), and $\mathbf{z}$ is a k-fold solution of the initial system with $\|\mathbf{m} - \mathbf{z}\| < 2^{-L}$, the system Φ (as well as Φ' for generic choice of its coefficients) yields a cluster of k (not necessarily distinct) solutions with norm less than $4 \cdot 2^{-L}$, whereas all other solutions have norm larger than $\delta_0 \gg 2^{-L}$. Here, δ_0 is a constant that depends on the polynomial system but not on L; see Theorem 7.20 for the exact definition of δ_0 and further

Algorithm 7.1: #PolySol($\mathcal{F}, \mathbf{\Delta}, K$)

```
Input  : Zero-dimensional system $\mathcal{F} = (f_1, \ldots, f_n)$, polydisc $\mathbf{\Delta} = \mathbf{\Delta}_r(\mathbf{m})$, and an
         integer $K \in \{0, \ldots, d_{\mathcal{F}}\}$.
Output: An integer $k \in \mathbb{N} \cup \{-1\}$. If $k \geq 0$, the polydisc $\mathbf{\Delta}$ contains exactly $k$ solutions
         of $\mathcal{F}$ (counted with multiplicity). If $k = -1$, nothing can be said.
   // * Shift and Truncation *//
1  $L := \lceil \log \frac{r}{32n(K+1)^n} \rceil$
2  Compute a $(K+1) \cdot L$-bit approximation $\Phi' = (\phi'_1, \ldots, \phi'_n)$ of
     $\mathcal{F}[\mathbf{m}]_{\leq K} = (f_1[\mathbf{m}]_{\leq K}, \ldots, f_n[\mathbf{m}]_{\leq K})$.
   // * Adding a degree $(K+1)$-perturbation ; as mentioned, this step seems to be only necessary
     in theory. In practice, we recommend to directly proceed with $\Phi := \Phi'$. *//
3  $\Phi(\mathbf{x}) := (\phi_1, \ldots, \phi_n)$, with $\phi_i := x_i^{K+1} + \phi'_i$
   // * Solving the truncated system *//
4  Compute a list $(\mathbf{\Delta}_1, k_1), \ldots, (\mathbf{\Delta}_\ell, k_\ell)$ of disjoint polydiscs $\mathbf{\Delta}_i = \mathbf{\Delta}_{r_i}(\mathbf{m}_i)$ of radius at most
     $2^{-L}$ and corresponding multiplicities $k_i$ such that each $\mathbf{\Delta}_i$ contains exactly $k_i$
     solutions of $\Phi$, and each solution of $\Phi$ is contained within one $\mathbf{\Delta}_i$.
5  $k := \sum_{i:\|\mathbf{m}_i\| < \frac{r}{2n}} k_i$
6  $k^+ := \sum_{i:\|\mathbf{m}_i\| < 2nr} k_i$
7  if $k = k^+$ then
       // * Projection step *//
8      Pick $S \in \mathcal{S}_{D_\Phi}$ uniformly at random and compute the rotated system
         $\Phi^* := \Phi \circ S^{-1} = (\phi_i \circ S^{-1})_i$.
9      for $\ell = 1, \ldots, n$ do
           $(b_\ell^-, k_\ell^-, \mathrm{LB}_\ell^-) := \mathcal{T}_*(\Delta_{\frac{r}{\sqrt{n}}}(0), \mathrm{Res}(\Phi^*, x_\ell))$
           $(b_\ell^+, k_\ell^+, \mathrm{LB}_\ell^+) := \mathcal{T}_*(\Delta_{\sqrt{n}r}(0), \mathrm{Res}(\Phi^*, x_\ell))$.
10     if $\bigwedge_{\ell \in [n]} b_\ell^- \wedge \bigwedge_{\ell \in [n]} b_\ell^+$ then
           // * Bound Computation and Comparison *//
           $\mathrm{UB}(\mathbf{m}, r) := r^{K+1} \cdot d_{\mathcal{F}}^n 2^{\tau_{\mathcal{F}}+2}[M(\mathbf{m}) \cdot (n + d_{\mathcal{F}})^2]^{d_{\mathcal{F}}}$
11         $B_{\Phi^*} := 2 \cdot \lceil D_{\Phi^*} \cdot (6n+4) \cdot \log(d_{\Phi^*} + n) \rceil + \tau_{\Phi^*} \cdot \frac{D_{\Phi^*}}{k+1}$
           $\mathrm{LB}(\mathbf{m}, r) := \min_\ell \min(\mathrm{LB}_\ell^-, \mathrm{LB}_\ell^+) \cdot \left(n \cdot \binom{D_{\Phi^*}+n}{D_{\Phi^*}} \cdot 2^{B_{\Phi^*}}\right)^{-1}$
12         if $\mathrm{UB}(\mathbf{m}, r) \leq \mathrm{LB}(\mathbf{m}, r)$ then
13             return $k$
14 return $-1$
```

details. We first check whether there exists a cluster of solutions of Φ near the origin that is well separated from all other solutions of Φ. For this, we use a certified method (e.g. [BS16]) to compute all solutions of Φ. Here, by computing all solutions, it is meant to compute a set of disjoint discs, each of size less than 2^{-L}, together with the number of solutions contained in each disc such that the union of all discs contains all complex solutions. For the more involved problem of computing isolating regions of comparable size, the following theorem applies.

Theorem 7.16. *[BS16, Thm. 9, 10] There is a Las Vegas algorithm to compute isolating regions of* *size less than $2^{-\rho}$ for all complex solutions of a zero-dimensional polynomial system $\mathcal{F} = (f_1, \dots, f_n)$, with integer polynomials $f_i \in \mathbb{Z}[\mathbf{x}]$, using*

$$\tilde{O}(n^{(n-1)(\omega+1)+1}(nd_{\mathcal{F}} + \tau_{\mathcal{F}})d_{\mathcal{F}}{}^{(\omega+2)n-\omega-1} + n \cdot d_{\mathcal{F}}{}^{n} \cdot \rho)$$

bit operations in expectation.

Since $2^{(K+1)L} \cdot \phi_i$ is a polynomial of degree $K+1$ with integer coefficients of magnitude $\tau = O(KL + n + \tau_{\mathcal{F}} + d_{\mathcal{F}} \,\overline{\log}(\mathbf{m}))$, we conclude from the above theorem that the cost for solving the system $\Phi = 0$ is bounded by

$$\tilde{O}(n^{(n-1)(\omega+1)+1}(nK + KL + \tau + d_{\mathcal{F}} \,\overline{\log}(\mathbf{m})) \cdot (K+1)^{(\omega+2)n-\omega-1}) \tag{7.10}$$

bit operations in expectation. Finally, we check whether the polydisc $\mathbf{\Delta}^- := \mathbf{\Delta}_{r/(2n)}(0)$ contains the same number k' of solutions of Φ as the enlarged polydisc $\mathbf{\Delta}^+ := \mathbf{\Delta}_{2nr}(0)$. Notice that, from the above remark, this holds true if $\mathbf{m}$ is an L-bit approximation of a k-fold zero of $\mathcal{F}$ for large enough L as then $4 \cdot 2^{-L} < r/(2n)$ and $\delta_0 > 2nr$. If the zeros of Φ do not fulfill the latter condition, we return -1. Otherwise, we proceed.

Remark. We remark that computing the solutions of Φ is typically much more affordable than computing the solutions of the initial system $\mathcal{F}$ directly, in particular, in the case where n is small and $K \ll d$. Notice that, for n of constant size, the cost for solving the initial system directly scales like $d^{(\omega+2)n-\omega-1}\tau_{\mathcal{F}}$, whereas the cost for solving the truncated system scales like $(KL + \tau_{\mathcal{F}} + d\,\overline{\log}(\mathbf{m})) \cdot (K+1)^{(\omega+2)n-\omega-1}$. Hence, for L and m of moderate size, the running times might differ by factor of size $\approx (d/(K+1))^{(\omega+2)n-\omega-1}$.

Step 3: Passing from Φ to $\mathcal{F}$. In the final step, we aim to certify that $\mathcal{F}[\mathbf{m}]$ has the same number of zeros (i.e. k counted with multiplicity) in $\mathbf{\Delta} := \mathbf{\Delta}_r(0)$ as Φ. In order to do so, we aim to apply the following generalization of Rouché's Theorem to Φ and $\mathcal{F}[\mathbf{m}]$, see [VH94, Thm. 2.1] or [Llo75, Thm. 1] for a proof.

Theorem 7.17 (Multidimensional Rouché). *Let $\mathcal{F} = (f_1, \dots, f_n)$ and $\mathcal{G} = (g_1, \dots, g_n)$, with $f_i, g_i \in \mathbb{C}[\mathbf{x}]$ for all i, define polynomial mappings from $\mathbb{C}^n$ to $\mathbb{C}^n$. If, for a given bounded domain $\mathbf{D} \subset \mathbb{C}^n$, we have*

$$\|\mathcal{F}(\mathbf{x}) - \mathcal{G}(\mathbf{x})\| < \|\mathcal{F}(\mathbf{x})\| \quad \text{for all } \mathbf{x} \in \partial\mathbf{D},$$

where $\partial\mathbf{D}$ is the boundary of $\mathbf{D}$, then $\mathcal{F}$ and $\mathcal{G}$ have finitely many zeros in $\mathbf{D}$ and the number of zeros (counted with multiplicities) of $\mathcal{F}$ and $\mathcal{G}$ in $\mathbf{D}$ is the same.

In order to apply the above theorem to $\mathcal{F} := \Phi$ and $\mathcal{G} := \mathcal{F}[\mathbf{m}]$, we derive an upper bound $\mathrm{UB}(\mathbf{m}, r)$ on the absolute error

$$\begin{aligned}\sup_{\mathbf{x}\in\mathbb{C}^n:\|\mathbf{x}\|=r} \|\mathcal{F}[\mathbf{m}](\mathbf{x}) - \Phi(\mathbf{x})\| &= \sup_{\mathbf{x}\in\mathbb{C}^n:\|\mathbf{x}\|=r} \max_{i\in[n]} |f_i[\mathbf{m}](\mathbf{x}) - \phi_i(\mathbf{x})| \\ &\leq \sup_{\mathbf{x}\in\mathbb{C}^n:\|\mathbf{x}\|=r} \max_{i\in[n]} (|f_i[\mathbf{m}](\mathbf{x}) - \phi_i'(\mathbf{x})| + |x_i|^{K+1}) \\ &\leq r^{K+1} + \sup_{\mathbf{x}\in\mathbb{C}^n:\|\mathbf{x}\|=r} \max_{i\in[n]} |f_i[\mathbf{m}](\mathbf{x}) - \phi_i'(\mathbf{x})|\end{aligned}$$

when passing from Φ to $\mathcal{F}[\mathbf{m}]$ as well as a lower bound $\mathrm{LB}(\mathbf{m}, r)$ on the norm of $\Phi(\mathbf{x})$ on the boundary of the polydisc $\mathbf{\Delta}$. The construction of $\mathrm{UB}(\mathbf{m}, r)$ is rather straightforward using Lemma 7.3. That is, we may choose

$$\mathrm{UB}(\mathbf{m}, r) := r^{K+1} \cdot d_{\mathcal{F}}{}^{n} \cdot 2^{\tau_{\mathcal{F}}+2} \cdot [M(\mathbf{m}) \cdot (n + d_{\mathcal{F}})^2]^{d_{\mathcal{F}}} \tag{7.11}$$

In contrast, the construction of $\mathrm{LB}(\mathbf{m}, r)$ is more involved: We already mentioned that if L is large enough, then there are k zeros $\mathbf{z}_1, \ldots, \mathbf{z}_k$ of Φ that have norm less than $4 \cdot 2^{-L}$, whereas all other zeros have norm $\delta_0 \gg 2^{-L}$. Hence, under this assumption, picking[5] a random rotation matrix S from $\mathcal{S}_{D_\Phi} = \mathcal{S}_{(K+1)^n}$ and considering a corresponding rotation of the coordinate system, guarantees (see Lemma 7.14), with probability larger than 1/2, that the projection of any zero of the "rotated system"

$$\Phi^* = (\phi_1^*, \ldots, \phi_n^*) := \Phi \circ S^{-1}$$

on any coordinate axis, except for the k solutions $S \cdot \mathbf{z}_i$, yields a value that is large compared to r. Hence, in this case, the hidden-variable resultant $R_\ell^* := \mathrm{Res}(\Phi^*, x_\ell)$ of Φ^* has k roots of absolute value less than r, whereas all other roots of R_ℓ^* have absolute value $\gg r$. Notice that each $\phi_j^* \in \mathbb{Q}[\mathbf{x}]$ is a polynomial of degree $K+1$ with rational coefficients, and according to Lemma 7.2 and Lemma 7.4, we have

$$\|\phi_j^*\| = 2^{\tilde{O}(n + \tau_{\mathcal{F}} + d_{\mathcal{F}} \overline{\log}(\mathbf{m}))}.$$

Lemma 7.13 further yields the existence of an integer λ of absolute value $2^{\tilde{O}(n^3 \log K)}$ with $\lambda \cdot S^{-1} \in \mathbb{Z}^{n \times n}$. Hence, we conclude that each term of degree $K+1$ of $\lambda^{K+1} \cdot \phi_j^*$ has integer coefficients, and [CLO05, Theorem 3.1] further yields that

$$\mathrm{Res}(\lambda^{K+1} \cdot \Phi^*, x_\ell) = \lambda^{n(K+1)^n} \cdot \mathrm{Res}(\Phi^*, x_\ell).$$

It thus follows that

$$|\,\mathrm{LC}(\mathrm{Res}(\Phi^*, x_\ell))| = \lambda^{-n(K+1)^n} \cdot |\,\mathrm{LC}(\mathrm{Res}(\lambda^{K+1}\Phi^*, x_\ell))| \geq \lambda^{-n(K+1)^n} = 2^{-\tilde{O}((K+1)^n)},$$

where we use Corollary 7.9 to show that $|\,\mathrm{LC}(\mathrm{Res}(\lambda^{K+1}\Phi^*, x_\ell))|$ is a positive integer, hence larger than or equal to 1. Since $\mathrm{Res}(\Phi^*, x_\ell)$ is contained in the ideal generated by the polynomials ϕ_j^*, we may write

$$\mathrm{Res}(\Phi^*, x_\ell) = g_{\ell,1} \cdot \phi_1^* + \cdots + g_{\ell,n} \cdot \phi_n^* \tag{7.12}$$

with polynomials $g_{\ell,j} \in \mathbb{Q}[\mathbf{x}]$ of total degree bounded by $D_{\Phi^*} = (K+1)^n$. Corollary 7.11 further yields the following upper bound on the size of the coefficients of the $g_{\ell,j}$'s:

$$\log \|g_{\ell,j}\| \leq B_{\Phi^*} = \tilde{O}(D_{\Phi^*} \cdot n + \tau_{\Phi^*} \cdot \frac{D_{\Phi^*}}{K+1})) = \tilde{O}((K+1)^n + (K+1)^{n-1} \cdot (\tau_{\mathcal{F}} + d\,\overline{\log}(\mathbf{m}))).$$

Using Lemma 7.2, part a, this further yields a corresponding upper bound

$$\gamma := \binom{n + D_{\Phi^*}}{D_{\Phi^*}} \cdot 2^{B_{\Phi^*}} = 2^{\tilde{O}((K+1)^n + (K+1)^{n-1} \cdot (\tau_{\mathcal{F}} + d\,\overline{\log}(\mathbf{m})))} \tag{7.13}$$

such that $\max_{\ell,j} \sup_{\mathbf{x}:\|\mathbf{x}\| \leq 1} |g_{\ell,j}(\mathbf{x})| \leq \gamma$.

Remark. The reader might wonder why we do not compute the above cofactor representation (7.12) directly and then derive bounds on the size of $\max_{i,j} \sup_{\mathbf{x}:\|\mathbf{x}\|=1} |g_{\ell,j}(\mathbf{x})|$ using interval arithmetic, but instead use Corollary 7.11? The simple reason is that, at least in practice, computing the polynomials $g_{i,j}$ turns out to be considerably more costly than computing the resultant polynomials $R_\ell^*(x) = \mathrm{Res}(\Phi^*, x_\ell)$ only. In contrast, our approach of computing

[5]In practice, we recommend to consider $S = \mathrm{id}_n$ and thus $\Phi^* = \Phi$ as the initial choice as this turns out to be sufficient in most cases.

the bound γ does not require to compute the polynomials $g_{i,j}$, and thus comes at almost no additional cost. We further remark at this point that we will use the bounds from Corollary 7.11 in our complexity analysis of the algorithm.

In the next step, we compute lower bounds LB_ℓ^- and LB_ℓ^+ for $|R_\ell(x)^*|$ on the boundary of the two discs $D^- := \Delta_{r/\sqrt{n}}(0) \subset \mathbb{C}$ and $D^+ := \Delta_{r\cdot\sqrt{n}}(0) \subset \mathbb{C}$, respectively. For this, we use the $\mathcal{T}_k$-test that we have already introduced in Chapter 6. We recap its main properties in the below lemma:

Lemma 7.18. *Let $f \in \mathbb{C}[x]$ be a uni-variate polynomial of degree d and let $\Delta := \Delta_r(0) \subset \mathbb{C}$ be the disc with radius r centered at 0. The $\mathcal{T}_k$-test returns a pair*

$$\mathcal{T}_k(\Delta, f) = (b, \mathrm{LB}) = \Big(\frac{|f^{(k)}(0)|r^k}{k!} - \frac{3}{2}\cdot\sum_{i\neq k}\frac{|f^{(i)}(0)|r^i}{i!} > 0, \frac{1}{3}\cdot\frac{|f^{(k)}(0)|r^k}{k!}\Big). \tag{7.14}$$

If $b = \textsc{True}$, we say that $\mathcal{T}_k(\Delta, f)$ succeeds. If $\mathcal{T}_k(\Delta, f)$ succeeds, Δ contains exactly k roots counted with multiplicity and

$$5\cdot\mathrm{LB} > |f(x)| > \mathrm{LB} \quad \textit{for all } x \in \partial\Delta_r(0). \tag{7.15}$$

In addition, if $\Delta_{r/(16d)}(0)$ as well as $\Delta_{16d^4r}(0)$ contain exactly k roots, then $\mathcal{T}_k(\Delta, f)$ succeeds. We further define

$$\mathcal{T}_\star(\Delta, f) = \begin{cases} \Big(\textsc{True}, k, \frac{1}{3}\cdot\frac{|f^{(k)}(0)|r^k}{k!}\Big) & \textit{if } \mathcal{T}_k(\Delta, f) \textit{ succeeds for some } k,\\ (\textsc{False}, -1) & \textit{otherwise.}\end{cases} \tag{7.16}$$

Proof. In Chapter 6, we have proven all the statements except the bounds in (7.15). These inequalities are shown by simple applications of the triangle inequality: Using Taylor expansion, $x \in \partial\Delta_r(0)$, and that $\mathcal{T}_k(\Delta, f)$ yields

$$|f(x)| = \Big|\sum_{i=0}^{d}\frac{f^{(i)}(0)x^i}{i!}\Big| \geq \frac{|f^{(k)}(0)|r^k}{k!} - \sum_{i\neq k}\frac{|f^{(i)}(0)|r^i}{i!} \geq \frac{1}{3}\cdot\frac{|f^{(k)}(0)|r^k}{k!} = \mathrm{LB}$$

and

$$|f(x)| = \Big|\sum_{i=0}^{d}\frac{f^{(i)}(0)x^i}{i!}\Big| \leq \frac{|f^{(k)}(0)|r^k}{k!} + \sum_{i\neq k}\frac{|f^{(i)}(0)|r^i}{i!} \leq \frac{5}{3}\cdot\frac{|f^{(k)}(0)|r^k}{k!} = 5\cdot\mathrm{LB}. \qquad \square$$

Now, suppose that $\mathcal{T}_*(D^-, R_\ell^*) = (b_\ell^-, k_\ell^-, \mathrm{LB}_\ell^-)$ as well as $\mathcal{T}_*(D^+, R_\ell^*) = (b_\ell^+, k_\ell^+, \mathrm{LB}_\ell^+)$ succeed for all ℓ, then $\min(\mathrm{LB}_\ell^-, \mathrm{LB}_\ell^+)$ constitutes a lower bound for $|R_\ell^*|$ on the boundary of D^- as well as D^+. From (7.12), (7.13), and the definition of $\mathrm{LB}(\mathbf{m}, r)$, we now conclude that

$$\|\Phi^*(\mathbf{x})\| > \mathrm{LB}(\mathbf{m}, r) := \frac{\min_\ell \min(\mathrm{LB}_\ell^-, \mathrm{LB}_\ell^+)}{n\gamma}, \text{ for all } \mathbf{x} \text{ with } \|\mathbf{x}\| = \frac{r}{\sqrt{n}} \text{ or } \|\mathbf{x}\| = \sqrt{n}\cdot r. \tag{7.17}$$

Since the maximum and minimum of a holomorphic function (in several variables) on a bounded domain is taken at its boundary, we further conclude that the above inequality holds for any $\mathbf{x}$ with $r/\sqrt{n} \leq \|\mathbf{x}\| \leq \sqrt{n}\cdot r$. Notice that the rotation of the system by means of the rotation matrix maintains the 2-norm $\|.\|_2$ of any point. Thus, the the norm of any point $\mathbf{x}$ differs from the norm of the rotated point $S\cdot\mathbf{x}$ by a factor that is lower and upper bounded by $r/\sqrt{n}$ and $\sqrt{n} \cdot r$, respectively. Hence, from the above bound on $\|\Phi^*\|$, we conclude that

$$\|\Phi(\mathbf{x})\| > \mathrm{LB}(\mathbf{m}, r) \quad \text{for any } \mathbf{x} \text{ with } \|\mathbf{x}\| = r. \tag{7.18}$$

Now in order to apply Rouché's Theorem to Φ and $\mathcal{F}[\mathbf{m}]$, it suffices to check whether $\mathrm{LB}(\mathbf{m}, r) > \mathrm{UB}(\mathbf{m}, r)$, in which case we have shown that Φ and $\mathcal{F}[\mathbf{m}]$ have the same number of roots in $\mathbf{\Delta}_r(0)$. Hence, we return True in this case. Otherwise, the algorithm returns False.

In the next section, we will show that, if $\mathbf{m}$ is a sufficiently good approximation (i.e. for large enough L) of a k-fold solution of $\mathcal{F}$, our algorithm succeeds. Here, we only give an informal argument: Notice that, for large L, the bound $\mathrm{UB}(\mathbf{m}, r)$ scales like $C \cdot r^{K+1}$ for some constant C. The bound γ does not depend on L, hence $\mathrm{LB}(\mathbf{m}, r)$ scales like $\min_\ell \min(\mathrm{LB}_\ell^-, \mathrm{LB}_\ell^+)$ for large enough L. However, in this situation, each R_ℓ^* has a cluster of k roots near the origin that is well separated from all of its remaining roots, and thus $\min(\mathrm{LB}_\ell^-, \mathrm{LB}_\ell^+)$ scales like $[|R_\ell^{*(k)}(0)|/(\sqrt{n}^k k!)] \cdot r^k$. Hence, we conclude that $\mathrm{LB}(\mathbf{m}, r)$ scales like $C' \cdot r^k$ for some constant C', which implies that $\mathrm{LB}(\mathbf{m}, r)$ must be smaller than $\mathrm{UB}(\mathbf{m}, r)$ for large enough L. We remark that the precise argument is slightly more involved as many subtleties need to be addressed. In particular, we need to show that $|R_\ell^{*(k)}(0)|/(\sqrt{n}^k k!)$ does not depend on r if r is small enough, even though the definition of R^* strongly depends on the choice of m, r, and the rotation matrix S. We will give details in the next section.

7.4 Analysis

We start by introducing some further notation. For a zero-dimensional polynomial system $\mathcal{F} = (f_1, \ldots, f_n)$ in n variables, let $\mathbf{z}_1, \ldots, \mathbf{z}_N$ denote its zeros. We define

$$\sigma(\mathbf{z}_i, \mathcal{F}) := \min_{j \neq i} \|\mathbf{z}_i - \mathbf{z}_j\| \quad \text{and} \quad \partial(\mathbf{z}_i, \mathcal{F}) := \prod_{j \neq i} \|\mathbf{z}_i - \mathbf{z}_j\|^{\mu(\mathbf{z}_j, \mathcal{F})}$$

to be the *separation of z_i with respect to $\mathcal{F}$* and the *geometric derivative of $\mathcal{F}$ at z_i*, respectively. We remark that these terms are derived from the interpretation of these quantities in the univariate case, where the separation of a root z_0 of a polynomial $f \in \mathbb{C}[x]$ is defined in exactly the same way, and the first non-vanishing derivative

$$\left| \frac{\partial^{\mu(z_0, f)} f}{\partial z^{\mu(z_0, f)}}(z_0) \right| = \mathrm{LC}(f) \cdot \prod_{z \neq z_0 : f(z) = 0} |z - z_0|^{\mu(z, f)}$$

of f at z_0 can be expressed as a product involving the leading coefficient of f and the distances between z_0 and the other roots. We first provide some bounds on $\|\mathbf{z}_i\|$, $\sigma(\mathbf{z}_i, \mathcal{F})$, and $\partial(\mathbf{z}_i, \mathcal{F})$ for the special case where each f_i has only integer coefficients. For similar bounds that are also adaptive with respect to the sparseness of the given system, we refer to [EMT10].

Lemma 7.19. *Let $\mathcal{F} = (f_i)_{i=1,\dots,n}$ be a zero-dimensional system with integer polynomials f_i, and let $\mathbf{z}_1,\dots,\mathbf{z}_N$ denote the zeros of $\mathcal{F}$. Then it holds:*

$$\sum_{i=1}^{N} \mu(\mathbf{z}_i,\mathcal{F}) \cdot \overline{\log}(\mathbf{z}_i) = \tilde{O}(n \cdot B_{\mathcal{F}}) = \tilde{O}(n \cdot [n \cdot D_{\mathcal{F}} + \tau_{\mathcal{F}} \cdot \max_i \frac{D_{\mathcal{F}}}{d_i}]), \text{ with } B_{\mathcal{F}} \text{ as in (7.7)}$$

$$|\log \sigma(\mathbf{z}_i,\mathcal{F})| = \tilde{O}(D_{\mathcal{F}} \cdot B_{\mathcal{F}}),$$

$$\overline{\log}(\partial(\mathbf{z}_i,\mathcal{F})^{-1}) = \tilde{O}(n \cdot D_{\mathcal{F}} \cdot [B_{\mathcal{F}} + \overline{\log}(\mathbf{z}_i)]) = \tilde{O}(n^2 \cdot D_{\mathcal{F}} \cdot B_{\mathcal{F}}), \text{ and}$$

$$\overline{\log}(\sigma(\mathbf{z}_i,\mathcal{F})^{-1}) = \tilde{O}(D_{\mathcal{F}} \cdot \overline{\log}(\mathbf{z}_i) + n \cdot B_{\mathcal{F}} + \overline{\log}(\partial(\mathbf{z}_i,\mathcal{F})^{-1})) = \tilde{O}(n^2 \cdot D_{\mathcal{F}} \cdot B_{\mathcal{F}}).$$

Proof. From Corollary 7.11, we conclude that $\mathrm{Res}(\mathcal{F},x_\ell)$ is an integer polynomial of magnitude $(D_{\mathcal{F}},B_{\mathcal{F}})$ for all $\ell = 1,\dots,n$. Since the ℓ-th coordinate $z_{i,\ell}$ of each solution of $\mathcal{F} = 0$ is a root of multiplicity at least $\mu(\mathbf{z}_i,\mathcal{F})$ of $\mathrm{Res}(\mathcal{F},x_\ell)$ and since the Mahler measure

$$\mathrm{Mea}(\mathrm{Res}(\mathcal{F},x_\ell)) = \mathrm{LC}(\mathrm{Res}(\mathcal{F},x_\ell)) \cdot \prod_{z\in\mathbb{C}:\mathrm{Res}(\mathcal{F},x_\ell)(z)=0} M(z)^{\mu(z,\mathrm{Res}(\mathcal{F},x_\ell))}$$

of $\mathrm{Res}(\mathcal{F},x_\ell)$ is upper bounded by its 2-norm $\|\mathrm{Res}(\mathcal{F},x_\ell))\|_2 \le \sqrt{D_F}\cdot 2^{B_{\mathcal{F}}}$ (e.g. see [Yap00]), it follows that

$$\sum_{i=1}^{N} \mu(\mathbf{z}_i,\mathcal{F}) \cdot \overline{\log}(\mathbf{z}_i) \le D_{\mathcal{F}} + \sum_{\ell=1}^{n} \log(\mathrm{Mea}(\mathrm{Res}(\mathcal{F},x_\ell))) = \tilde{O}(nB_{\mathcal{F}}).$$

For the second claim, notice that $\sigma(\mathbf{z}_i,\mathcal{F}) \ge \sigma(z_{i,\ell},\mathrm{Res}(\mathcal{F},x_\ell))$ for at least one ℓ (as two distinct solutions must differ in at least one coordinate), and that the separation of an integer polynomial of magnitude $(D_{\mathcal{F}},B_{\mathcal{F}})$ is lower bounded by $2^{-\tilde{O}(D_{\mathcal{F}}\cdot B_{\mathcal{F}})}$; e.g. see [MSW15] for a proof. For the bound on $\partial(\mathbf{z}_i,\mathcal{F})$, notice that

$$\partial(\mathbf{z}_i,\mathcal{F}) \ge \frac{\prod_{\ell=1}^{n} \partial(z_{i,\ell},\mathrm{Res}(\mathcal{F},x_\ell))}{\prod_{\ell=1}^{n}\prod_{z\ne z_{i,\ell}:\mathrm{Res}(\mathcal{F},x_\ell)(z)=0} M(z-z_{i,\ell})^{\mu(z,\mathrm{Res}(\mathcal{F},x_\ell))}}.$$

According to the proof of [MSW15, Thm. 5], it holds that $\partial(z_0,f) = 2^{-\tilde{O}(dL)}$ for any root z_0 of a polynomial $f \in \mathbb{Z}[x]$ of magnitude (d,L). This shows that $\partial(z_{i,\ell},\mathrm{Res}(\mathcal{F},x_\ell)) = 2^{-\tilde{O}(D_{\mathcal{F}}B_{\mathcal{F}})}$ for all ℓ. It remains to derive an upper bound on the denominator in the above fraction. For this, we define $R_\ell := \mathrm{Res}(\mathcal{F},x_\ell)[z_{i,\ell}]$. Then, it holds that

$$\prod_{\ell=1}^{n}\prod_{z\ne z_{i,\ell}:\mathrm{Res}(\mathcal{F},x_\ell)(z)=0} \max(1,|z-z_{i,\ell}|)^{\mu(z,\mathrm{Res}(\mathcal{F},x_\ell))} = \prod_{\ell=1}^{n} \frac{\mathrm{Mea}(R_\ell)}{\mathrm{LC}(R_\ell)}.$$

According to Lemma 7.2, R_ℓ is a polynomial of magnitude $(D_{\mathcal{F}},\tilde{O}(B_{\mathcal{F}} + D_{\mathcal{F}}\cdot\overline{\log}(z_{i,\ell})))$, and, in addition, it has the same leading coefficient as $\mathrm{Res}(\mathcal{F},x_\ell)$. In particular, its leading coefficient is a non-zero integer, and thus of absolute value larger than or equal to 1. Thus, we have

$$\frac{\mathrm{Mea}(R_\ell)}{\mathrm{LC}(R_\ell)} \le \|R_\ell\|_2 = 2^{\tilde{O}(B_{\mathcal{F}}+D_{\mathcal{F}}\cdot\overline{\log}(z_{i,\ell}))} = 2^{\tilde{O}(B_{\mathcal{F}}+D_{\mathcal{F}}\cdot\overline{\log}(\mathbf{z}_i))},$$

which shows that $\overline{\log}(\partial(\mathbf{z}_i,\mathcal{F})^{-1}) = \tilde{O}(nD_{\mathcal{F}}(B_{\mathcal{F}} + \overline{\log}(\mathbf{z}_i)))$.

For the last claim, notice that $\sigma(\mathbf{z}_i,\mathcal{F})$ appears as one of the factors in the definition of $\partial(\mathbf{z}_i,\mathcal{F})$. Since the product of all remaining factors is upper bounded by

$$\prod_{\mathbf{z}_j\neq\mathbf{z}_i} M(\mathbf{z}_j-\mathbf{z}_i)^{\mu(\mathbf{z}_j,\mathcal{F})} \leq \prod_{\mathbf{z}_j\neq\mathbf{z}_i}[2\cdot M(\mathbf{z}_j)\cdot M(\mathbf{z}_i)]^{\mu(\mathbf{z}_j,\mathcal{F})} \leq 2^{D_\mathcal{F}}\cdot M(\mathbf{z}_i)^{D_\mathcal{F}}\cdot\prod_{j=1}^{N} M(\mathbf{z}_j)^{\mu(\mathbf{z}_j,\mathcal{F})},$$

the claim follows directly from the bound on $\sum_{i=1}^{N}\mu(\mathbf{z}_i,\mathcal{F})\cdot\overline{\log}(\mathbf{z}_i)$ and on $\overline{\log}(\partial(\mathbf{z}_i,\mathcal{F})^{-1})$. □

We are now ready to derive one of the main results of this chapter. More specifically, the following theorem shows that, in a sufficiently small neighborhood (which we will also quantify) of a k-fold solution $\mathbf{z}=\mathbf{z}_i$ of $\mathcal{F}=0$, $\|\mathcal{F}(\mathbf{x})\|$ scales like $c\cdot\|\mathbf{x}\|^k$ with c a constant. We further argue that this implies that a sufficiently good approximation Φ of the shifted and truncated system $\mathcal{F}[\mathbf{z}]_{\leq K}$, with arbitrary $K\geq k$, has a cluster of k solutions near the origin, whereas all remaining solutions are well separated from this cluster. We also give bounds on the approximation error that involve the quantities $\sigma(\mathbf{z},\mathcal{F})$ and $\partial(\mathbf{z},\mathcal{F})$ that are intrinsic to the hardness of the given polynomial system.

Theorem 7.20. *Let $\mathcal{F}$ be a zero-dimensional system, $\mathbf{z}$ a zero of $\mathcal{F}$ of multiplicity k, and $\mathbf{m}$ be an approximation of $\mathbf{z}$ with $\|\mathbf{m}-\mathbf{z}\|<2^{-L}$. Let $K\geq k$, and $\Phi'=(\phi'_i)_{i=1,\dots,n}$ be a $(K+1)\cdot L$-bit approximation of $\mathcal{F}[\mathbf{m}]_{\leq K}$ with polynomials ϕ'_i of degree at most K, and let $a_1,\dots,a_n\in\mathbb{C}$ be arbitrary complex values of magnitude $0\leq|a_i|\leq 1$ for all i. Then, the polynomial system*

$$\Phi := (\phi'_i + a_i\cdot x_i^{K+1})_{i=1,\dots,n}$$

is zero-dimensional, and there exists an $L_0\in\mathbb{N}$ such that, for any $L\geq L_0$, Φ has exactly k zeros (counted with multiplicity) of norm smaller than $4\cdot 2^{-L}$, whereas all other zeros have norm larger than $\delta_0 := \frac{\sigma(\mathbf{z},\mathcal{F})}{(2n^2D_\mathcal{F})^{32n}}$. In the special case, where each polynomial in $\mathcal{F}$ has only integer coefficients, it holds that

$$L_0=\tilde{O}(n\cdot D_\mathcal{F}\cdot[n^3+\max_i\frac{\tau_\mathcal{F}+d_\mathcal{F}}{d_i}+\overline{\log}(\mathbf{z})]+\overline{\log}(\partial(\mathbf{z},\mathcal{F})^{-1}))).$$

Proof. We denote $\mathbf{z}_1,\dots,\mathbf{z}_N$, with $\mathbf{z}=\mathbf{z}_i$, the zeros of $\mathcal{F}$. Let $S\in\mathcal{S}_{D_\mathcal{F}}$ be an admissible rotation matrix with respect to $\mathcal{F}$ as well as with respect to the shifted system $\mathcal{F}[\mathbf{z}]$. Notice that such a matrix exists as more than half of the matrices in $\mathcal{S}_{D_\mathcal{F}}$ are admissible with respect to $\mathcal{F}$ and more than half of the matrices are admissible with respect to $\mathcal{F}[\mathbf{z}]$. Let $\mathcal{F}^*:=\mathcal{F}\circ S^{-1}$ be the corresponding "rotation" of $\mathcal{F}$ and $\mathbf{z}_1^*,\dots,\mathbf{z}_N^*$ be the zeros of $\mathcal{F}^*$ such that $\mathbf{z}_j^*=(z_{j,1}^*,\dots,z_{j,n}^*)=S\cdot\mathbf{z}_j$. Since S is admissible with respect to $\mathcal{F}[\mathbf{z}]$, Lemma 7.14 yields that

$$|z_{j,\ell}^*-z_{i,\ell}^*|\geq(2n^2D_\mathcal{F})^{-16n}\cdot\|\mathbf{z}_j-\mathbf{z}\|\geq\frac{\sigma(\mathbf{z},\mathcal{F})}{(2n^2D_\mathcal{F})^{16n}}$$

for all ℓ and $j\neq i$. In addition, since S is also admissible with respect to $\mathcal{F}$, Lemma 7.6 and Lemma 7.14 guarantees that each root of the resultant polynomial $\operatorname{Res}(\mathcal{F}^*,x_\ell)$ is the projection of a finite zero of $\mathcal{F}^*$ on the x_ℓ-coordinate. Thus, $\operatorname{Res}(\mathcal{F}^*,x_\ell)$ has a k-fold root at $z_{i,\ell}^*$, whereas all other roots $z_{j,\ell}^*$ of $\operatorname{Res}(\mathcal{F}^*,x_\ell)$ have distance at least $\frac{\sigma(\mathbf{z},\mathcal{F})}{(2n^2D_\mathcal{F})^{16n}}$ to $z_{i,\ell}^*$. Now, applying Lemma 7.18 to a disc with center $z_{i,\ell}^*$ and arbitrary radius smaller than

$$r_0^*:=\frac{\sigma(\mathbf{z},\mathcal{F})}{16D_\mathcal{F}^4\cdot(2n^2D_\mathcal{F})^{16n}},$$

yields that

$$|\operatorname{Res}(\mathcal{F}^*, x_\ell)(x)| > \frac{1}{3k!} \cdot \left|\frac{\partial^k \operatorname{Res}(\mathcal{F}^*, x_\ell)}{\partial x^k}(z^*_{i,\ell})\right| \cdot |x - z^*_{i,\ell}|^k \quad \text{for all } x \in \mathbb{C} \text{ with } |x - z^*_{i,\ell}| < r^*_0.$$

Denoting $\mathrm{LC}_\ell := \mathrm{LC}(\operatorname{Res}(\mathcal{F}^*, x_\ell))$, this further yields

$$\begin{aligned}\left|\frac{\partial^k \operatorname{Res}(\mathcal{F}^*, x_\ell)}{\partial x^k}(z^*_{i,\ell})\right| &= |\,\mathrm{LC}_\ell\,| \cdot \prod_{j\neq i} |z^*_{i,\ell} - z^*_{j,\ell}|^{\mu(z^*_{j,\ell}, \operatorname{Res}(\mathcal{F}^*, x_\ell))} \\ &\geq |\,\mathrm{LC}_\ell\,| \cdot \prod_{j\neq i} \left(\frac{\|\mathbf{z} - \mathbf{z}_j\|}{(2n^2 D_{\mathcal{F}})^{16n}}\right)^{\mu(\mathbf{z}_j, \mathcal{F})} \\ &\geq \frac{|\,\mathrm{LC}_\ell\,|}{(2n^2 D_{\mathcal{F}})^{16nD_{\mathcal{F}}}} \cdot \partial(\mathbf{z}, \mathcal{F}),\end{aligned}$$

and thus it follows that

$$\begin{aligned}|\operatorname{Res}(\mathcal{F}^*, x_\ell)(x)| &> \frac{|\,\mathrm{LC}_\ell\,|}{3k! \cdot (2n^2 D_{\mathcal{F}})^{16nD_{\mathcal{F}}}} \cdot \partial(\mathbf{z}, \mathcal{F}) \cdot |x - z^*_{i,\ell}|^k \\ &> \frac{|\,\mathrm{LC}_\ell\,|}{4D_{\mathcal{F}}! \cdot (2n^2 D_{\mathcal{F}})^{16nD_{\mathcal{F}}}} \cdot \partial(\mathbf{z}, \mathcal{F}) \cdot |x - z^*_{i,\ell}|^k \\ &> \frac{|\,\mathrm{LC}_\ell\,|}{(4n^2 D_{\mathcal{F}})^{17nD_{\mathcal{F}}}} \cdot \partial(\mathbf{z}, \mathcal{F}) \cdot |x - z^*_{i,\ell}|^k \quad \text{if } |x - z^*_{i,\ell}| < r^*_0. \qquad (7.19)\end{aligned}$$

Furthermore, $\operatorname{Res}(\mathcal{F}^*, x_\ell)$ is contained in the ideal spanned by the polynomials $\mathcal{F}^* = (f^*_1, \ldots, f^*_n)$, that is, there exist polynomials $g_{\ell,j} \in \mathbb{C}[\mathbf{x}]$ with $\operatorname{Res}(\mathcal{F}^*, x_\ell) = \sum_{j=1}^n g_{\ell,j} f^*_j$. According to Corollary 7.11, we may assume that

$$\log \|g_{\ell,j}\| \leq B_{\mathcal{F}^*} = \tilde{O}(n \cdot D_{\mathcal{F}^*} + \tau_{\mathcal{F}^*} \cdot \max_i \frac{D_{\mathcal{F}}}{d_i}) = \tilde{O}(n \cdot D_{\mathcal{F}} + (\tau_{\mathcal{F}} + d_{\mathcal{F}}) \cdot \max_i \frac{D_{\mathcal{F}}}{d_i})$$

for all ℓ, j, where the last inequality follows from Lemma 7.4. Using Lemma 7.2 then implies that

$$\gamma_{\mathcal{F}^*} := \binom{n + D_{\mathcal{F}}}{D_{\mathcal{F}}} \cdot 2^{B_{\mathcal{F}^*}} \cdot (M(\mathbf{z}) + 1)^{D_{\mathcal{F}}} \geq \sup_{\mathbf{x}:\|\mathbf{x} - \mathbf{z}^*_i\| \leq 1} |g_{i,j}(\mathbf{x})| \text{ for all } \ell, j. \qquad (7.20)$$

Now, combining (7.19) and (7.20) yields

$$\|\mathcal{F}^*(\mathbf{x})\| \geq \left[\frac{\min_\ell |\,\mathrm{LC}_\ell\,|}{n \cdot \gamma_{\mathcal{F}^*} \cdot (4n^2 D_{\mathcal{F}})^{17nD_{\mathcal{F}}}} \cdot \partial(\mathbf{z}, \mathcal{F})\right] \cdot \|\mathbf{x} - \mathbf{z}^*_i\|^k$$

for all $\mathbf{x} \in \mathbb{C}^n$ with $\|\mathbf{x} - \mathbf{z}^*_i\| < r^*_0$.

So what can we conclude about our initial (non-rotated) system? Since a rotation maintains the Euclidean distance and since the max-norm differs from the Euclidean norm by a factor of at most $\sqrt{n}$, it follows that a point $\mathbf{x}$ of max-norm $\|\mathbf{x}\|$ is rotated via S (or S^{-1}) onto a point $\mathbf{x}'$ of max-norm $\|\mathbf{x}'\| \leq \|\mathbf{x}'\|_2 = \|\mathbf{x}\|_2 \leq \sqrt{n} \cdot \|\mathbf{x}\|$. Hence, it holds that every point $\mathbf{x}^* = S\mathbf{x}$ with $\|\mathbf{x} - \mathbf{z}\| < r_0 := r^*_0/\sqrt{n}$ satisfies $\|\mathbf{x}^* - \mathbf{z}^*_i\| < r^*_0$. Thus, for all $\mathbf{x}$ with $\|\mathbf{x} - \mathbf{z}\| < r_0$, it holds that

$$\|\mathcal{F}(\mathbf{x})\| \geq \underbrace{\left[\frac{\min_\ell |\operatorname{LC}_\ell|}{n \cdot \sqrt{n}^k \cdot \gamma_{\mathcal{F}^*} \cdot (4n^2 D_{\mathcal{F}})^{17nD_{\mathcal{F}}}} \cdot \partial(\mathbf{z},\mathcal{F})\right] }_{=:c}\cdot \|\mathbf{x}-\mathbf{z}\|^k \tag{7.21}$$

Now, suppose that $L \geq \log(8/r_0)$ and thus $2^{-L} < \frac{r_0}{8}$. Since $\mathbf{m}$ is an approximation of $\mathbf{z}$ with $\|\mathbf{z}-\mathbf{m}\| < 2^{-L}$, $\mathcal{F}[\mathbf{m}]$ has exactly one solution (namely, $\hat{\mathbf{z}} := \mathbf{z}-\mathbf{m}$) of multiplicity k in $\mathbf{\Delta}_{r_0/2}(0)$ and

$$\|\mathcal{F}[\mathbf{m}](\mathbf{x})\| = \|\mathcal{F}(\mathbf{x}+\mathbf{m})\| > \frac{c}{2^k} \cdot \|\mathbf{x}\|^k \quad \text{for all } \mathbf{x} \in \mathbb{C}^n \text{ with } 2^{-L+1} < \|\mathbf{x}\| < \frac{r_0}{2},$$

as, for such $\mathbf{x}$, it holds that $\|\mathbf{x}\|/2 < \|\mathbf{x}+\mathbf{m}-\mathbf{z}\| < r_0$. Applying Lemma 7.3 to each ϕ_i and using the fact that $M(\mathbf{m}) \leq 2M(\mathbf{z})$ then shows that, for all $\mathbf{x}$ with $2^{-L+1} < \|\mathbf{x}\| < r_0/2$, it holds that

$$\begin{aligned}\|\Phi(\mathbf{x}) - \mathcal{F}[\mathbf{m}](\mathbf{x})\| &\leq 2^{-(K+1)L} + \|\mathbf{x}\|^{K+1} \cdot d_{\mathcal{F}}{}^n \cdot 2^{\tau_{\mathcal{F}}+d_{\mathcal{F}}} \cdot [M(\mathbf{z}) \cdot (n+d_{\mathcal{F}})^2]^{d_{\mathcal{F}}} \\ &\leq \|\mathbf{x}\|^{K+1} \cdot d_{\mathcal{F}}^n \cdot 2^{\tau_{\mathcal{F}}+d_{\mathcal{F}}+1} \cdot [M(\mathbf{z}) \cdot (n+d_{\mathcal{F}})^2]^{d_{\mathcal{F}}}.\end{aligned}$$

Notice that, due to the construction of Φ and Corollary 7.9, Φ is zero-dimensional. Hence, Rouché's Theorem applied to $\mathcal{F}[\mathbf{m}]$ and Φ shows that the polydisc $\mathbf{\Delta}_\rho(0)$ contains the same number of solutions of Φ and $\mathcal{F}[\mathbf{m}]$ if $2^{-L+1} < \rho < r_0/2$ and if, in addition, ρ fulfills the following inequality

$$\rho^{K+1} \cdot d_{\mathcal{F}}^n \cdot 2^{\tau_{\mathcal{F}}+d_{\mathcal{F}}+1} \cdot [M(\mathbf{z}) \cdot (n+d_{\mathcal{F}})^2]^{d_{\mathcal{F}}} < \frac{c}{2^k} \cdot \rho^k.$$

Equivalently, we must have $2^{-L+1} < \rho < r_0/2$ and

$$\rho^{K+1-k} < \frac{c}{2^k \cdot d_{\mathcal{F}}^n \cdot 2^{\tau_{\mathcal{F}}+d_{\mathcal{F}}+1}[M(\mathbf{z}) \cdot (n+d_{\mathcal{F}})^2]^{d_{\mathcal{F}}}}.$$

Hence, for

$$\begin{aligned}L > L_0 &:= \max\left[\log\frac{8}{r_0}, \log\frac{2^k \cdot d_{\mathcal{F}}{}^n 2^{\tau_{\mathcal{F}}+d_{\mathcal{F}}+1}[M(\mathbf{z}) \cdot (n+d_{\mathcal{F}})^2]^{d_{\mathcal{F}}}}{c}\right] \\ &= \tilde{O}(D_{\mathcal{F}} \cdot (n + \max_i \frac{\tau_{\mathcal{F}}+d_{\mathcal{F}}}{d_i} + \overline{\log}(\mathbf{z})) + \overline{\log}(\partial(\mathbf{z},\mathcal{F})^{-1}) + \overline{\log}(\sigma(\mathbf{z},\mathcal{F})^{-1}) \\ &\quad - |\log\min_\ell |\operatorname{LC}_\ell||).\end{aligned} \tag{7.22}$$

each polydisc $\mathbf{\Delta}_{\rho'}(\mathbf{z})$, with arbitrary radius $\rho' \in (4 \cdot 2^{-L}, r_0/4)$, contains exactly k zeros of Φ. Since $\delta_0 < r_0/4$, this proves the first part of the theorem.

It remains to prove the claimed bound on L_0 for the special case, where $\mathcal{F}$ is a polynomial system defined over the integers. For this, we need to estimate the size of the leading coefficient of $\operatorname{Res}(\mathcal{F}^*, x_\ell)$. Notice that there exists an integer λ of size $2^{\tilde{O}(n^3 \log d_{\mathcal{F}})}$ with $\lambda \cdot S^{-1} \in \mathbb{Z}^{n \times n}$, and thus

$$\mathcal{F}' : (f_1', \ldots, f_n') = (\lambda^{\deg f_1^*} \cdot f_1^*, \ldots, \lambda^{\deg f_n^*} \cdot f_n^*)$$

is a polynomial system with integer coefficients, which shows that $|\operatorname{LC}(\operatorname{Res}(\mathcal{F}', x_\ell))| \geq 1$. Using [CLO05, Thm. 2.3 and 3.5] then shows that

$$|\operatorname{LC}_\ell| = \lambda^{-nD_{\mathcal{F}}} \cdot |\operatorname{LC}(\operatorname{Res}(\mathcal{F}', x_\ell))| \geq \lambda^{-nD_{\mathcal{F}}} = 2^{-O(n^4 D_{\mathcal{F}})}.$$

Hence, the bound follows from (7.22) and the bound for $\overline{\log}\,\sigma(\mathbf{z}, \mathcal{F})^{-1}$ from Lemma 7.19. □

From the previous Theorem, we now immediately obtain the following result by setting $\mathbf{m} := \mathbf{z}$ and $\Phi := \mathcal{F}[\mathbf{z}]_{\leq K}$ for an arbitrary $K \geq k$.

Corollary 7.21. *Let* $\mathbf{z}$ *be a k-fold zero of a zero dimensional system* $\mathcal{F}$ *and* $K \geq k$. *Then,* $\mathcal{F}[\mathbf{z}]_{\leq K}$ *has a k-fold zero at the origin, and all other zeros have norm larger than* $\delta_0 := \frac{\sigma(\mathbf{z},F)}{(2n^2 D_{\mathcal{F}})^{32n}}$.[6]

We can now show that Algorithm 7.1 terminates and yields a correct result assuming that L is large enough and the oracle, which provides an approximation m of the solution $\mathbf{z}$, returns a correct answer.

Theorem 7.22. *If* #PolySol($\mathcal{F}, \mathbf{\Delta}, K$) *returns an integer* $k \geq 0$, *then the polydisc* $\mathbf{\Delta} = \mathbf{\Delta}_r(\mathbf{m})$ *contains exactly k solutions of* $\mathcal{F}$ *counted with multiplicity. Vice versa, suppose that* $\mathbf{z}$ *is a solution of* $\mathcal{F}$ *of multiplicity k and* $K \geq k$, *then there exists a positive integer* L^* *of size*

$$L^* = \tilde{O}(L_0 + (K+1)^n \cdot \log \frac{1}{\delta_0} + d_{\mathcal{F}} \cdot \overline{\log}(\mathbf{z})).$$

with L_0 *and* δ_0 *as in Theorem 7.20, such that* #PolySol($\mathcal{F}, \mathbf{\Delta}, K$) *returns k with probability at least* 1/2 *if* $r \leq 2^{-L^*}$ *and* $\|\mathbf{z} - \mathbf{m}\| < \frac{r}{64n(K+1)^n}$. *If* $\mathcal{F}$ *has only integer coefficients, it holds:*

$$\begin{aligned} L^* &= \tilde{O}(D_{\mathcal{F}} \cdot \max_i \frac{d_{\mathcal{F}} + \tau_{\mathcal{F}}}{d_i} + D_{\mathcal{F}} \cdot \overline{\log}(\mathbf{z}) + \overline{\log}(\partial(\mathbf{z}, \mathcal{F})^{-1}) + (K+1)^n \cdot \overline{\log}(\sigma(\mathbf{z}, \mathcal{F})^{-1})) \\ &= \tilde{O}((K+1)^n \cdot D_{\mathcal{F}} \cdot [D_{\mathcal{F}} + \tau_{\mathcal{F}} \cdot \max_\ell \frac{D_{\mathcal{F}}}{d_\ell}]). \end{aligned}$$

Proof. For the first part, we proceed similarly as in the proof of Theorem 7.20, however, we work with the system Φ instead of the initial system $\mathcal{F}$. From Line 7 in the algorithm, we already know that Φ has exactly k solutions with ($\|.\|$-) norm less than r/n, whereas all other solutions have norm at least nr. Now, when considering a random rotation matrix $S \in \mathcal{S}_{D_\Phi}$, the corresponding rotated system $\Phi^* = \Phi \circ S^{-1}$ has exactly k solutions with norm less than $r/\sqrt{n}$, whereas all other solutions have norm at least $\sqrt{n} \cdot r$. We may now further write

$$\operatorname{Res}(\Phi^*, x_\ell) = \gamma_{\ell,1} \cdot \phi_1^* + \cdots + \gamma_{\ell,n} \cdot \phi_n^* \tag{7.23}$$

with polynomials $\gamma_{\ell,j} \in \mathbb{C}[\mathbf{x}]$. From Part d of Lemma 7.2 and Corollary 7.11 we conclude that

$$\binom{D_{\Phi^*} + n}{D_{\Phi^*}} \cdot B_{\Phi^*} \geq \sup_{\mathbf{x}:\|\mathbf{x}\| \leq 1} |\gamma_{\ell,j}(\mathbf{x})| \text{ for all } \ell, j.$$

In addition, since $\mathcal{T}_*(\Delta_{\frac{r}{\sqrt{n}}}(0), \operatorname{Res}(\Phi^*, x_\ell)) = (\text{True}, k_\ell^-, \text{LB}_\ell^-)$ and $\mathcal{T}_*(\Delta_{\sqrt{n}r}(0), \operatorname{Res}(\Phi^*, x_\ell)) = (\text{True}, k_\ell^+, \text{LB}_\ell^+)$ for all ℓ (Line 10), we conclude from Lemma 7.18 that

$$|\operatorname{Res}(\Phi^*, x_\ell)(\mathbf{x})| > \min(\text{LB}_\ell^-, \text{LB}_\ell^+) \text{ for all } i \text{ and all } \mathbf{x} \text{ with } |x_\ell| = \frac{r}{\sqrt{n}} \text{ or } |x_\ell| = \sqrt{n}r.$$

[6]We remark that $\mathcal{F}[\mathbf{z}]_{\leq K}$ does not necessarily have to be zero-dimensional. Rouché's Theorem only guarantees that the polydisc $\mathbf{\Delta}_{\delta_0}(0)$ contains exactly k zeros of $\mathcal{F}[\mathbf{z}]_{\leq K}$.

Hence, using (7.23), this shows that

$$\|\Phi^*(\mathbf{x})\| \geq \frac{\min_{\ell=1,\dots,n} \min(\mathrm{LB}_\ell^-, \mathrm{LB}_\ell^+)}{n \cdot \binom{D_{\Phi^*}+n}{D_{\Phi^*}} \cdot B_{\Phi^*}} \text{ for all } \mathbf{x} \text{ with } \|\mathbf{x}\| = \frac{r}{\sqrt{n}} \text{ or } \|\mathbf{x}\| = \sqrt{n}r.$$

Since a holomorphic mapping cannot take its minimum or maximum in the interior of some domain, it thus follows that the above inequality even holds for any $\mathbf{x}$ with $\frac{r}{\sqrt{n}} \leq \|\mathbf{x}\| \leq \sqrt{n}r$. It thus follows that

$$\|\Phi(\mathbf{x})\| \geq \frac{\min_{\ell=1,\dots,n} \min(\mathrm{LB}_\ell^-, \mathrm{LB}_\ell^+)}{n \cdot \binom{D_{\Phi^*}+n}{D_{\Phi^*}} \cdot B_{\Phi^*}} \text{ for all } \mathbf{x} \text{ with } \|\mathbf{x}\| = r.$$

According to (7.11), $\mathrm{UB}(\mathbf{m}, r)$ constitutes an upper bound on the error $\|\mathcal{F}[\mathbf{m}](\mathbf{x}) - \Phi(\mathbf{x})\|$ for any $\mathbf{x}$ with $\|\mathbf{x}\| \leq 1$. Hence, in particular, we also have

$$\|\mathcal{F}[\mathbf{m}](\mathbf{x}) - \Phi(\mathbf{x})\| \leq \mathrm{UB}(\mathbf{m}, r) \text{ for all } \mathbf{x} \text{ with } x \text{ with } \|\mathbf{x}\| = r.$$

Hence, using Rouché's Theorem, we conclude that $\mathcal{F}[\mathbf{m}]$ and Φ have the same number of solutions in the polydisc $\mathbf{\Delta}_r(0)$. This shows the first part.

It remains to prove the second claim. For this, suppose that $\mathcal{F}$ has a k-fold solution at $\mathbf{z}$ with $\|\mathbf{m} - \mathbf{z}\| < \frac{r}{64n(K+1)^n}$ and that

$$\begin{aligned} L := \lceil \log \frac{32n(K+1)^n}{r} \rceil > L_1 &:= \max(L_0, \log \frac{1}{\delta_0} + \log[2048 \cdot n^2 \cdot (2n^2 D_{\mathcal{F}}(K+1)^n)^{32n}]) \\ &= \tilde{O}(L_0 + \log \frac{1}{\delta_0} + n^2 \log D_{\mathcal{F}}), \end{aligned} \tag{7.24}$$

with $\delta_0 = \frac{\sigma(\mathbf{z},\mathcal{F})}{(2n^2 D_{\mathcal{F}})^{32n}}$ as defined in Theorem 7.20. Let Φ be an approximation of $\mathcal{F}[\mathbf{m}]_{\leq K}$ as defined in Theorem 7.20. Then, Φ has k solutions $\mathbf{z}_1, \dots, \mathbf{z}_k$ of norm $\|\mathbf{z}_i\| < 4 \cdot 2^{-L} < r/(2n)$, whereas all remaining solutions, denoted by $\bar{\mathbf{z}}_1, \dots, \bar{\mathbf{z}}_m$, have norm $\|\bar{\mathbf{z}}_j\| > \delta_0 > 2nr$. We thus conclude that the if-condition is satisfied in Line 7. Now, when choosing a random rotation matrix $S \in \mathcal{S}_{D_\Phi}$, the solutions $\mathbf{z}_i$ near the origin are mapped to solutions $\mathbf{z}_i^* := S \circ \mathbf{z}_i$ of $\Phi^* = \Phi \circ S^{-1}$ with norm $\|\mathbf{z}_i^*\| < 4\sqrt{n} \cdot 2^{-L} \leq \frac{r}{16\sqrt{n}(K+1)^n}$, whereas the remaining solutions are mapped to solutions $\bar{\mathbf{z}}_j^*$ of Φ^* with norm $\|\bar{\mathbf{z}}_j^*\| > \delta_0/\sqrt{n} > 2\sqrt{n}r$. In addition, with probability more than $1/2$, we have

$$|z_{j,\ell}^*| \geq \|\bar{\mathbf{z}}_j\| \cdot (2n^2 D_{\Phi^*})^{-16n} > \delta_0 \cdot (2n^2(K+1)^n)^{-16n} > 16 \cdot \sqrt{n} \cdot (K+1)^{4n} r$$

for all $j \in [m]$ and all $\ell \in [n]$. This implies that each of the resultant polynomials $\mathrm{Res}(\Phi^*, x_\ell)$ has k roots of absolute value less than $\frac{r}{16(K+1)^n \sqrt{n}}$, whereas all remaining roots are of absolute value larger than $16 \cdot \sqrt{n} \cdot (K+1)^{4n} r$. Notice that each polynomial $\mathrm{Res}(\Phi^*, x_\ell)$ has degree $(K+1)^n$, and thus Lemma 7.18 guarantees success in Line 10 of the algorithm. Now, recall the lower bounds LB_ℓ^- and LB_ℓ^+ for $|\mathrm{Res}(\Phi^*, x_\ell)|$ on the boundary of $\Delta_{r/\sqrt{n}}(0)$ and $\Delta_{\sqrt{n}r}(0)$, respectively, as computed in Line 11. For arbitrary $x \in \mathbb{C}$ with $|x| = r/\sqrt{n}$, we have

$$\begin{aligned} \mathrm{LB}_\ell^- &\geq \frac{|\mathrm{Res}(\Phi^*, x_\ell)(x)|}{5} > \frac{|\mathrm{LC}(\mathrm{Res}(\Phi^*, x_\ell))|}{5} \cdot \left(\frac{r}{2\sqrt{n}}\right)^k \cdot \left(\frac{\delta_0 \cdot (2n^2(K+1)^n)^{-16n}}{2}\right)^{(K+1)^n - k} \\ &> |\mathrm{LC}(\mathrm{Res}(\Phi^*, x_\ell))| \cdot (2n^2(K+1)^n)^{-32n(K+1)^n} \cdot r^k \cdot \delta_0^{(K+1)^n}. \end{aligned}$$

Using the fact $\mathrm{LB}_\ell^+ \geq \frac{|\mathrm{Res}(\Phi^*, x_\ell)(x)|}{5}$ for all x with $|x| = 2\sqrt{n}r$, an analogous computation shows that LB_ℓ^+ fulfills the same bound, that is,

$$\mathrm{LB}_\ell^+ \geq |\,\mathrm{LC}(\mathrm{Res}(\Phi^*, x_\ell))| \cdot (2n^2(K+1)^n)^{-32n(K+1)^n} \cdot r^k \cdot \delta_0^{(K+1)^n}.$$

From Lemma 7.9 and our construction of Φ^*, the leading coefficient of each polynomial $\mathrm{Res}(\Phi^*, x_\ell)$ is a non-zero integer, hence we obtain that

$$\mathrm{LB}(\mathbf{m}, r) = \frac{\min_{\ell=1,\ldots,n} \min(\mathrm{LB}_\ell^-, \mathrm{LB}_\ell^+)}{n \cdot \binom{D_{\Phi^*}+n}{D_{\Phi^*}} \cdot 2^{B_{\Phi^*}}} \geq \underbrace{\frac{\delta_0^{(K+1)^n}}{n(2n^2(K+1)^n)^{32n(K+1)^n} \cdot \binom{D_{\Phi^*}+n}{D_{\Phi^*}} \cdot 2^{B_{\Phi^*}}}}_{=:C} \cdot r^k.$$

Notice that, for small r, $\mathrm{LB}(\mathbf{m}, r)$ scales like $C \cdot r^k$, with a constant C that does not depend on r. The upper bound

$$\mathrm{UB}(\mathbf{m}, r) = r^{K+1} \cdot \underbrace{d_{\mathcal{F}}{}^n 2^{\tau_{\mathcal{F}}+2}[M(\mathbf{m}) \cdot (n + d_{\mathcal{F}})^2]^{d_{\mathcal{F}}}}_{=:C'}$$

scales like $C' \cdot r^{K+1}$, and thus our algorithm succeeds if r fulfills the condition in (7.24) (i.e. $\lceil \log \frac{32n(K+1)^n}{r} \rceil \geq L_1$) and $r^{K-k+1} < \frac{C}{C'}$. Both condition are fulfilled if

$$\log(1/r) \geq L^* := \max(L_1, \log \frac{C'}{C}) = \tilde{O}(L_0 + (K+1)^n \cdot \log \frac{1}{\delta_0} + d_{\mathcal{F}} \cdot \overline{\log}(\mathbf{z})).$$

The claimed bound on L^* for the special case where $\mathcal{F}$ is defined over the integers follows directly from the corresponding bound on L_0 from Theorem 7.20 and our bounds on $\overline{\log}(\mathbf{z})$, $\overline{\log}(\sigma(\mathbf{z}, \mathcal{F})^{-1})$, and $\overline{\log}(\partial(\mathbf{z}, \mathcal{F})^{-1})$ from Lemma 7.19. □

7.5 Application: Computing the Zeros of a Bivariate System

In this section, we report on an application of our technique in the context of elimination methods for the bivariate case. More precisely, we incorporate the algorithm #PolySol as an inclusion predicate in the Bisolve algorithm [Ber+13; KS15b]. Comparing Sage implementations of the original Bisolve algorithm and its modified variant, we empirically show that the idea of truncating the original system with respect to the multiplicity of the solution yields a considerable performance improvement. Bisolve is a classical elimination method for computing the real [Ber+13] or complex [KS15b] zeros within a given polydisc $\mathbf{\Delta} = \Delta_1 \times \Delta_2 \subset \mathbb{C}^2$ of a bivariate system

$$\mathcal{F} : f_1(x_1, x_2) = f_2(x_1, x_2) = 0, \text{ with polynomials } f_1, f_2 \in \mathbb{Z}[x_1, x_2].$$

It achieves the best known complexity bound (i.e. $\tilde{O}(d_{\mathcal{F}}^6 + d_{\mathcal{F}}^5 \cdot \tau_{\mathcal{F}})$ bit operations for computing all complex solution) that is currently known for this problem, and its implementation shows superior performance when compared to other complete and certified methods. As we aim to modify the Bisolve algorithm at some crucial steps, we start with a brief description of the original version.

Bisolve in a Nutshell. In an initial *projection phase*, Bisolve computes a set C of *candidate regions* using resultant computation and univariate root finding. More specifically, we first compute the hidden-variable resultants $R_\ell(x) := \mathrm{Res}(\mathcal{F}, x_\ell)$ for $\ell = 1, 2$. Then, for each root

$z_{\ell,i}$ in Δ_ℓ, we compute an isolating disc $\Delta_{\ell,i}$ such that $\mathcal{T}_\star(\Delta_{\ell,i}, R_\ell) = (\text{True}, k_{\ell,i}, \text{LB}_{\ell,i})$. That is, the $\mathcal{T}_\star$-test succeeds and yields the multiplicity of $z_{\ell,i}$ as a root of R_ℓ as well as a lower bound for $|R_\ell|$ on the boundary of $\Delta_{\ell,i}$. By taking the pairwise product of any two discs $\Delta_{1,i}$ and $\Delta_{2,j}$, we obtain a set C of polydiscs $\mathbf{\Delta}_{i,j} := \Delta_{1,i} \times \Delta_{2,j}$ in $\mathbb{C}^2$. Notice that each solution $\mathbf{z}$ in $\mathbf{\Delta}$ of $\mathcal{F}$ must be one of the *candidate solutions* $\mathbf{z}_{i,j} := (z_{1,i}, z_{1,j})$ as each coordinate of $\mathbf{z}$ is a root of the corresponding polynomial R_ℓ. Hence, each solutions must be contained in one of the candidate regions, even though most candidate regions do not contain any solution. In addition, each candidate region $\mathbf{\Delta}_{i,j}$ contains at most one solution, which must be $\mathbf{z}_{i,j}$

In the *validation phase*, the algorithm checks for every candidate region $\mathbf{\Delta}_{i,j}$ whether it contains a solution or not. In other words, we check whether the corresponding candidate solution $\mathbf{z} = \mathbf{z}_{i,j}$ is actually a solution or not. The approach used in Bisolve shares many similarities to the algorithm #PolySol that we proposed here. That is, we write

$$R_\ell = g_{\ell,1} \cdot f_1 + g_{\ell,2} \cdot f_2 \text{ with } g_{\ell,1}, g_{2,\ell} \in \mathbb{Z}[x,y] \text{ and } \ell = 1,2$$

and compute an upper bound UB for $|g_{\ell,i}(\mathbf{x})|$ for $\ell = 1,2$, $i = 1,2$, and arbitrary $\mathbf{x} \in \mathbf{\Delta}_{i,j}$. Similar as in #PolySol, this is achieved without actually computing the polynomials $g_{\ell,1}$ and $g_{\ell,2}$, but by exploiting the fact that these polynomials can be written as determinants of “Sylvester-like” matrices[7]; see [KS15b] for details. Together with the lower bounds $\text{LB}_{1,i}$ and $\text{LB}_{2,j}$ as computed above this yields a lower bound $\text{LB}^* = \frac{\min(\text{LB}_{1,i}, \text{LB}_{2,j})}{2\,\text{UB}}$ for $\|\mathcal{F}\| = \max(|f_1|, |f_2|)$ on the boundary of $\mathbf{\Delta}_{i,j}$.

Now, in order to discard or certify $\mathbf{z}$ as a solution, Bisolve proceed in rounds, where a 2^m-bit approximation ζ of $\mathbf{z}$ is computed at the beginning of the m-th round. As an *exclusion predicate*, interval arithmetic is used in order to compute a superset $\square f_\ell(\mathbf{\Delta}_{2^{-m}}(\zeta))$ of $f_\ell(\mathbf{\Delta}_{2^{-m}}(\zeta))$ for $\ell = 1,2$. If we can show that either f_1 or f_2 does not vanish, the candidate is discarded. As an *inclusion predicate* the above lower bound LB^* on the boundary of $\mathbf{\Delta}_{i,j}$ is compared to the values that f_1 and f_2 take at the approximation ζ of the candidate $\mathbf{z}$. More specifically, if $\max(|f(\zeta)|, |g(\zeta)|) < \text{LB}^*$, then $\mathbf{\Delta}_{i,j}$ contains a solution; see Theorem 4 in [Ber+13]. If neither the exclusion nor the inclusion predicate applies, we proceed with the next round.

The BisolvePlus Routine. Notice that, even though Bisolve computes the set

$$\mathbf{Z} := \{\mathbf{z}_{i,j} \in \mathbf{\Delta} : f_1(\mathbf{z}_{i,j}) = f_2(\mathbf{z}_{i,j}) = 0\}$$

of all solutions of $\mathcal{F}$ within $\mathbf{\Delta}$, it does not reveal the multiplicity k of a specific solution $\mathbf{z} = \mathbf{z}_{i,j} = (z_{1,i}, z_{2,j}) \in \mathbf{Z}$. However, due to the properties of the resultant polynomials, it holds that $k = \mu(z_{1,i}, R_1)$ if the following two conditions are both fulfilled:

$$\deg_{x_2} f_1 = \deg f_1 \text{ and } \deg_{x_2} f_2 = \deg f_2 \tag{7.25}$$

$$\forall x_2 \in \mathbb{C} \setminus \{z_{2,j}\} : f_1(z_{1,i}, x_2) \neq 0 \text{ or } f_2(z_{1,i}, x_2) \neq 0 \tag{7.26}$$

The first condition guarantees that there is no solution of $\mathcal{F}$ at infinity above any $z \in \mathbb{C}$, whereas the second condition guarantees that there is no other finite (complex) solution of $\mathcal{F}$ that shares the first coordinate with $\mathbf{z}$. We remark that it is easy to check the first condition, however, checking the second condition is more difficult. This is due to the fact that $\mathbf{z}$ might be the only solution in $\mathbf{\Delta}$ of $\mathcal{F}$ with $x_1 = z_{1,i}$, but there is a another solution of $\mathcal{F}$ with $x_2 = z_{i,1}$

[7]Notice that this is one crucial point, where our novel approach differs from Bisolve. Namely, for #PolySol, we use the results from [DKS13] on the arithmetic Nullstellensatz to derive corresponding bounds on the cofactors $g_{\ell,j}$. This was necessary for generalizing the method to arbitrary dimension. Another crucial difference is that no truncation of the system is considered in Bisolve.

Algorithm 7.2: BisolvePlus

```
Input  : Zero-dimensional bivariate system $\mathcal{F} : f_1(x_1, x_2) = f_2(x_1, x_2) = 0$ with
         $f_1, f_2 \in \mathbb{Q}[x]$, polydisc $\mathbf{\Delta}_r(\mathbf{m})$
Output: $\mathbf{Z}^+$ such that $\mathbf{Z}^+ = \{(\mathbf{z}, k) : f_1(\mathbf{z}) = f_2(\mathbf{z}) = 0, k = \mu(\mathbf{z}, \mathcal{F})\}$.
for $\rho := 1, 2, 4, \ldots$. do
    Choose a matrix $S \in \mathcal{S}_{D_{\mathcal{F}}}$ and compute $\mathcal{F}^* = (f_1^*, f_2^*) = \mathcal{F} \circ S^{-1}$
    if $\deg_{x_2} f_1^* = \deg f_1^*$ and $\deg_{x_2} f_2^* = \deg f_2^*$ then
        Call Bisolve with input $\mathcal{F}^*$ and $\mathbf{\Delta}_{2r}(\mathbf{m})$ to compute (for $\ell = 1, 2$):
          • Discs $\Delta_{\ell,i}$, $i = 1, \ldots, i_\ell$, that isolate the roots $z_{\ell,i}$ of $R_\ell := \mathrm{Res}(\mathcal{F}^*, x_\ell)$.
          • The multiplicity $k_{\ell,i} = \mu(z_{\ell,i})$ of $z_{\ell,i}$ as a root of $R_\ell$.
          • The set $\mathbf{Z} := \{\mathbf{z}_{i,j} = (z_{1,i}, z_{2,j}) : f_1^*(\mathbf{z}_{i,j}) = f_2^*(\mathbf{z}_{i,j}) = 0\}$ of all solutions of $\mathcal{F}^*$.
        for each solution $\mathbf{z} = \mathbf{z}_{i,j} \in \mathbf{Z}$ do
            for $\ell = 1, 2$ do
                Compute disjoint discs $D_{\ell,1}, \ldots, D_{\ell,s_\ell}$ of radius less than $2^{-\rho}$ such that
                  • Each disc $D_{\ell,s}$ contains at least one root of $f_\ell^*(z_i, x_2) \in \mathbb{C}[x_2]$.
                  • $\bigcup_{s=1}^{s_\ell} D_{\ell,s}$ contains all roots of $f_\ell^*(z_i, x_2) \in \mathbb{C}[x_2]$.
            Determine the set
                $\mathbf{D}^* := \{D_{1,s} : \exists D_{2,s'} \text{ with } D_{1,s} \cap D_{2,s'} \neq \emptyset\}$
            of all discs $D_{1,s}$ that have non-empty intersection with one of the discs $D_{2,s'}$.
            if for all $D_{1,s} \in \mathbf{D}^*$ it holds that $D_{1,s} \subset \Delta_i$ then
                Set $b_{i,j}$ = True
        if $\bigwedge_{i,j} b_{i,j}$ then
            return $\{(S^{-1} \circ \mathbf{z}_{i,j}, k_i) : \mathbf{z}_{i,j} \in \mathbf{Z} \text{ and } \mathbf{z}_{i,j} \in \mathbf{\Delta}_r(\mathbf{m})\}$
```

that is not contained within $\mathbf{\Delta}$. We aim to address this problem by the following approach (see also Algorithm 7.2):

Let $L \in \mathbb{N}$ be fixed non-negative integer. In a first step, we check whether (7.25) is fulfilled. If this is not the case, we return False, otherwise, we proceed. Now, for each solution $\mathbf{z}_{i,j} \in \mathbf{Z}$ and each $\ell \in \{1, 2\}$, we use a complex root finder[8] to compute a set of pairwise disjoint discs $D_{\ell,j}$ of radius less than $2^{-\rho}$ such that each disc contains at least one root and the union of all discs $D_{\ell,j}$ contains all complex roots of $f_\ell(z_{1,i}, x_2) \in \mathbb{C}[x]$. Then, we determine all discs $D_{1,j_1}, \ldots, D_{1,j_s}$ that have a non-empty intersection with one of the discs $D_{2,j'}$. It follows that each common root of $f(z_{1,i}, x_2)$ and $f(z_{1,i}, x_2)$ must be contained in one of the discs $D_{1,j_{s'}}$. Hence, if each of these discs is contained in $\Delta_{2,i}$, then $x_2 = z_{2,j}$ is the unique solution of $f(z_{1,i}, x_2) = f(z_{2,i}, x_2) = 0$, and thus (7.26) is fulfilled. In this case, we may conclude that $\mu(z_{1,i}, R_1)$ equals the multiplicity of $\mathbf{z}$. If we succeed in computing the multiplicities for

[8]Each of the methods in [Bec+16; MSW15] and the method from the previous chapter apply to polynomials with arbitrary complex coefficients. Also, for computing only approximations of the roots, there are no restrictions on the multiplicities of the roots. In our implementation, we use a strongly simplified variant of the algorithm from the previous chapter.

all solutions in $\mathbf{Z}$, we return the solutions together with their corresponding multiplicities. Otherwise, we return False.

Obviously, the above approach cannot succeed if one of the above conditions is not fulfilled. However, even if both conditions are fulfilled, it may still fail due to the fact that ρ has not been chosen large enough.

Lemma 7.23. *Suppose that both conditions (7.25) and (7.26) are fulfilled. Then, there exists a $L_0 \in \mathbb{N}$ such that Algorithm 7.2 succeeds for all $L > L_0$.*

Proof. Let ε be a lower bound on the distance between any distinct roots of $f_1(z_{1,i}, x_2)$ and $f_2(z_{1,i}, x_2)$. Now, if $2^{-L} < \varepsilon/4$, then two discs $D_{1,j'}$ and $D_{2,j''}$ can only intersect if they contain a common root of $f_1(z_{i,1}, x_2)$ and $f_2(z_{i,1}, x_2)$. Since $z_{2,j}$ is the only common root, we thus conclude that each of the discs $D_{1,j_{s'}}$ must contain $z_{2,j}$. Hence, if L is large enough, then Δ_2 contains $D_{1,j_{s'}}$. □

The problem with this approach is that we do neither know in advance whether the condition (7.26) is fulfilled nor do we know whether ρ has been chosen sufficiently large. In order to overcome this issue, we consider a rotation of the system by means of a rotation matrix $S \in \mathcal{S}_{D_{\mathcal{F}}}$. Then, with probability at least 1/2, both conditions (7.25) and (7.26) are fulfilled for the rotated system $\mathcal{F}^* := \mathcal{F} \circ S^{-1}$. We now proceed in rounds (numbered by m), where, in each round, we choose a matrix $S \in \mathcal{S}_{D_{\mathcal{F}}}$ at random and run Algorithm 7.2 with input $\mathcal{F}^* = \mathcal{F} \circ S^{-1}$ and $\rho := 2^m$. Since there are only finitely many different choices for S and since the conditions (7.25) and (7.26) are fulfilled for at least the half of the systems $\mathcal{F}^*$, Lemma 7.23 guarantees that, for sufficiently large ρ, Algorithm 7.2 returns the solutions of $\mathcal{F}^*$ in $\boldsymbol{\Delta}$ together with the corresponding multiplicities with probability at least 1/2.

New Validation Phase. We are now ready to modify Bisolve by using an inclusion predicate based on the algorithm #PolySol. More specifically, let $C := \{\mathbf{z}_{i,j}\}_{i,j}$ be the set of candidate solutions and let $\boldsymbol{\Delta}_{i,j} = \Delta_{1,i} \times \Delta_{2,j}$ be the corresponding candidate regions as computed in the projection phase of Bisolve. The validation routine, see also Algorithm 7.3, that is called for each candidate solution $\mathbf{z} = \mathbf{z}_{i,j} = (z_{1,i}, z_{2,j})$ again works in rounds, where in round m, we compute a $L = 2^m$-bit approximation $\zeta = (\zeta_1, \zeta_2)$ of $\mathbf{z}$ such that $\|\zeta - \mathbf{z}\| < 2^{-L}$ and $\Delta_{64\max(d_1,d_2)^2 \cdot 2^{-L}}(\zeta) \subset \Delta_{2,j}$. The exclusion predicate is identical to the original Bisolve routine, i.e., we check whether we can guarantee that f_1 or f_2 does not vanish on $\boldsymbol{\Delta}_{2^{-L}}(\zeta)$ by evaluating interval extensions $\square f_\ell(\boldsymbol{\Delta}_{2^{-L}}(\zeta))$ for $\ell = 1, 2$ using interval arithmetic. The inclusion predicate now works as follows. There is still one tiny detail that prevents us from directly plugging in #PolySol as an inclusion predicate. Namely, even if the candidate solution would actually turn out to be a solution of the system, we do not know the multiplicity k of this solution. Thus, we have to search for the multiplicity k. As we have seen in the previous section that #PolySol$(\mathcal{F}, \boldsymbol{\Delta}_{2^{-L}}(\zeta), K)$ actually succeeds for any $K \geq k$, we can use exponential search for k by calling #PolySol$(\mathcal{F}, \boldsymbol{\Delta}_{2^{-L}}(\zeta), K)$ for $K = 1, 2, 4, \ldots, 2^{\lceil \log(\min(k_1, k_2)) \rceil}$, where k_ℓ for $\ell \in \{1, 2\}$ is the multiplicity of z_ℓ as a root of $R_\ell(x) := \operatorname{Res}(\mathcal{F}, x_\ell)$. In the calls to #PolySol, we use the above described BisolvePlus-routine in order to implement the computation of the solutions of the truncated system in Line 4 of Algorithm 7.1. We remark that our actual implementation of the new inclusion predicate differs slightly from the description of #PolySol in one more detail. For efficiency reasons, we consider a partial change of order of the three considered steps *Solving the truncated system*, *Projection step*, and *Bound Computation and Comparison*.

7.5.1 Setting

We performed experiments on a compute server with 48 Intel (R) Xeon (R) CPU E5-2680 v3 @ 2.50GHz cores and a total of 256 GB RAM running Debian GNU/Linux 8. All code was implemented in SageMath version 7.6, release date 2017-03-25.

Algorithm 7.3: VALIDATE

Input : Zero-dimensional system $\mathcal{F} = (f_1, f_2)$ with polynomials $f_1, f_2 \in \mathbb{Z}[\mathbf{x}]$ of degrees d_1 and d_2, respectively, polydisc $\mathbf{\Delta} = \Delta_1 \times \Delta_2 \subset \mathbb{C}^2$, candidate $\mathbf{z} = (z_1, z_2)$ and multiplicities k_ℓ s.t. the multiplicity of z_ℓ as a root of $R_\ell(x) := \operatorname{Res}(\mathcal{F}, x_\ell)$ is k_ℓ for $\ell \in \{1, 2\}$.

Output: $k \in \mathbb{N}_0$. If $k \geq 1$, then there is a unique solution $\mathbf{z}$ of $\mathcal{F} = 0$ of multiplicity k within the polydisc $\mathbf{\Delta}$. Otherwise, $\mathbf{\Delta}$ contains no solution.

```
for $L = 1, 2, 4, \ldots$ do
    Compute $L$-bit approximation $\zeta$ of $\mathbf{z}$
    if $\Delta_{2^{-L+2\lceil\log(\min(k_1,k_2))\rceil+6}}(\zeta) \subset \mathbf{\Delta}$ then
        if $0 \notin \Box f_1(\mathbf{\Delta}_{2^{-L}}(\zeta))$ or $0 \notin \Box f_2(\mathbf{\Delta}_{2^{-L}}(\zeta))$ then
            return 0
        for $K = 1, 2, 4, \ldots, 2^{\lceil\log(\min(k_1,k_2))\rceil}$ do
            if $\#\textsc{PolySol}(\mathcal{F}, \Delta_{2^{-L}}(\zeta), K) = k \geq 1$ then
                return $k$
```

7.5.2 Instance Generation

The instances on which we compared the implementations are generated as follows. Given a trivariate polynomial $P \in \mathbb{Z}[x, y, z]$. There are several different ways of obtaining two bivariate polynomials f, g from P that have solutions of higher multiplicity. The different ways are encoded by the strings `0xx`, `0xy`, `0yy`, `x0y`, `y0x` in the file names. The following table summarizes the meaning of these abbreviations. We denote $p_v = \partial_v p$ for any polynomial $p \in R[v]$ for some ring R.

`0xx`	$f = \operatorname{Res}(P, P_z, z)$	$g = f_x \cdot f_x$
`0xy`	$f = \operatorname{Res}(P, P_z, z)$	$g = f_x \cdot f_y$
`0yy`	$f = \operatorname{Res}(P, P_z, z)$	$g = f_y \cdot f_y$
`x0y`	$f = \operatorname{Res}(P, P_z, z) \cdot f_x$	$g = f_y$
`y0x`	$f = \operatorname{Res}(P, P_z, z) \cdot f_y$	$g = f_x$

From the resulting system f, g, we construct the sheared system $f, g \leftarrow f(ax + by, cx + dy), g(ax + by, cx + dy)$ with integers a, b, c, d drawn uniformly at random from $[-2, 2]$. This is done in order to make degenerate situations where multiple solutions share the same x or y-value less likely. We create an even larger set of instances by renaming the variables of P from x, y, z to x, z, y or y, z, x (or equivalently considering P_x and P_y instead of P_z). We abbreviate this choice with `xyz`, `xzy`, and `yzx`. Now, let z be a solution of such a system f, g of multiplicity k. We pick random polynomials p, q of increasing degrees and consider the systems $f \cdot p, g \cdot q$. This results in systems f_d, g_d of increasing degrees d that have the same solution z of multiplicity k. For each degree d, we create three such system f_d, g_d by multiplying f, g with different random polynomials.

There are two different classes of instances that we consider depending on how the initial trivariate polynomial P is chosen. In the first class, called `herwig_hauser`, we pick the polynomial P from the set of polynomials given as three dimensional surfaces in the Herwig Hauser Classics gallery [Hau]. In the second class, called `random`, we pick P randomly. In the first class called `herwig_hauser` we let $d = 10, 12, \ldots, 40$, whereas in the second class `random`, we let $d = 16, 32, \ldots, 4096$. We note that in the latter case we pick the random polynomial with which we multiply f, g in order to get f_d, g_d as sparse polynomials as

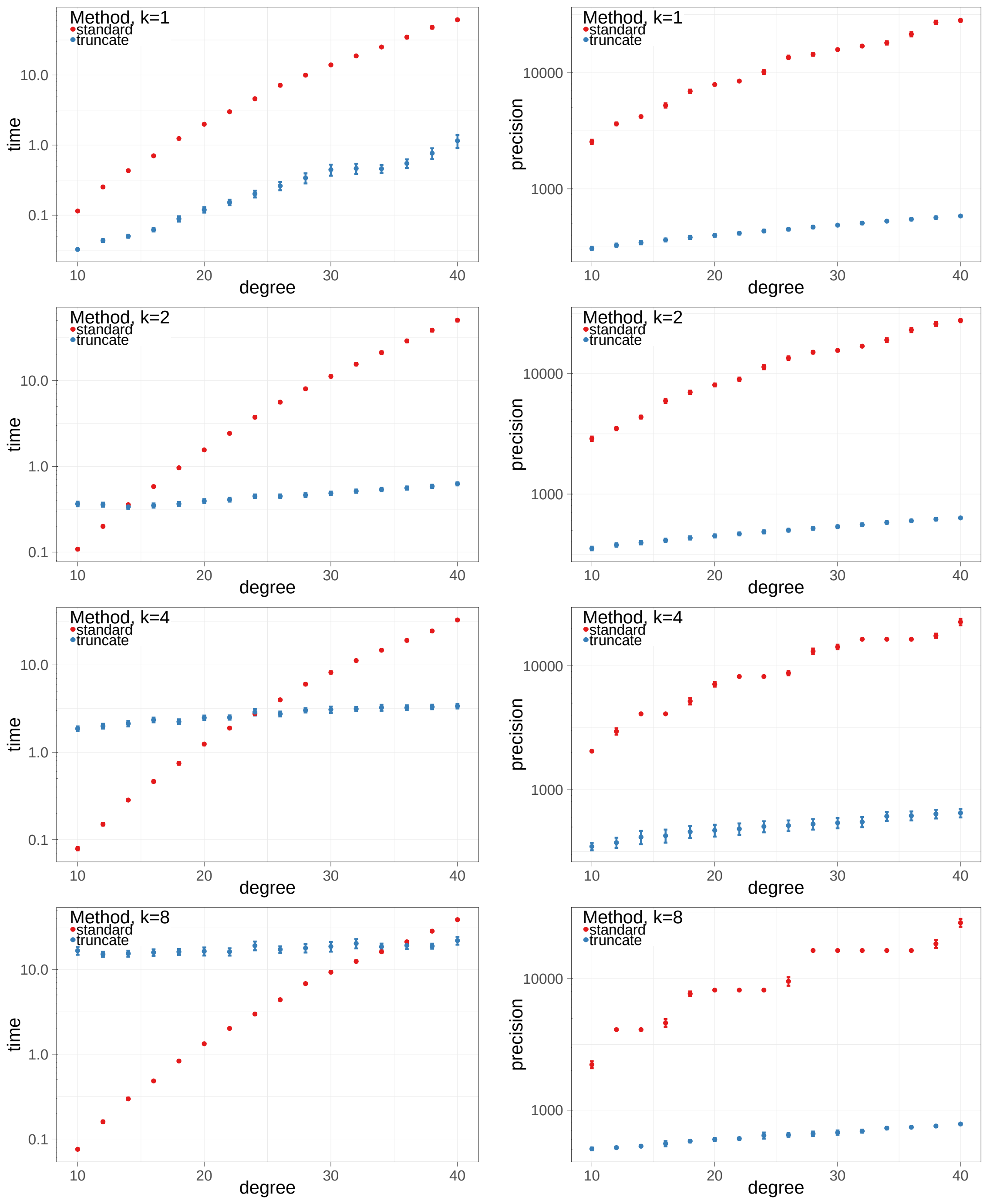

FIGURE 7.1: Evaluation for validation of k-fold roots, for $k = 1, 2, 4, 8$. In all plots the degree is on the horizontal axis. On the left the validation time is on the vertical axis and on the right the precision demand is on the vertical axis. The red dots correspond to the validation method of the original BISOLVE routine, called `standard`. The blue dots correspond to the validation method that uses the new inclusion predicate, called `truncate`.

otherwise evaluating f, g already becomes non-trivial.

The generated instances can be found on the project page.[9] A folder corresponding to

[9]`http://resources.mpi-inf.mpg.de/systemspellet/`

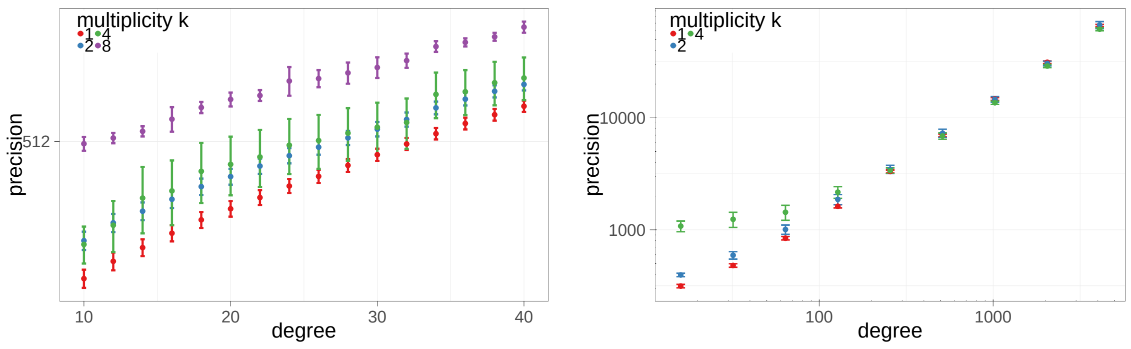

FIGURE 7.2: Comparison of the dependence of the precision demand on the degree for different values of k. For `herwig_hauser`-instances on the left, and for `random`-instances on the right.

a candidate contains one file called `orig.cnd`, which refers to the polynomials f, g. The remaining files correspond to the polynomials f_d, g_d as described above. Every file contains four lines, the first two contain the system, while the third and fourth contain the boundaries $x - r, x + r$ and $y - r, y + r$ such that the solution is contained within this range.

7.5.3 Experiments and Evaluation Results

In the first experiment, we compare the running time as well as the precision demand of the two respective validation methods called `standard` for the method included in the original BISOLVE routine and `truncate` for the method using the new inclusion predicate on the instance class `herwig_hauser`. In Figure 7.1, we can see the evaluation for validating k-fold roots for $k = 1, 2, 4, 8$. The measurements are repeated three times, for each method and system. This results in 9 measurements (3 different random polynomials, 3 different runs) per degree per method. On the left, the running times are on the vertical logarithmic axis, whereas the degree of the systems is on the linear horizontal axis. On the right, the precision demand is on the vertical logarithmic axis, whereas the degree of the systems is on the linear horizontal axis. The error bars indicate 95%-confidence intervals.

We can see a clear advantage for our new method `truncate`. On average over all instances of degree 40, we obtain an improvement of a factor of 43.6, 37.9, 29.8, 25.2 for $k = 1, 2, 4, 8$ in the precision demand. In Figure 7.2 on the left, we can see the precision demand for the `herwig_hauser` instances for different $k = 1, 2, 4, 8$ for the `truncate` method. We can see that the precision demand increases with k in a comparable amount as the theoretical worst-case bounds predict, namely, we can roughly see a quadratic dependence between the precision demand and the multiplicity k in Figure 7.2 on the left.

In Figure 7.2 on the right, we can see results for the same experiment for the `random` instances. In this experiment, we only include the `truncate` method as the `original` method does not scale well enough for solving instances of that degree. Here both axis are logarithmic and the degree goes up to 4096. Fitting a linear model to the data points leads an estimate for the exponent of 0.99 ± 0.05 0.94 ± 0.06, and 0.75 ± 0.13 for $k = 1, 2, 4$. The coefficients of determination lie above 0.94 in all three cases that is roughly 94% of the variance of the data can be explained by the fitted power model. Thus, we may conjecture that the precision demand depends at most linearly on d. We remark that the plot suggests that the impact of the degree d dominates over the impact of k for very large d as we cannot see a difference between the curves for different values of k for large d. Note that the impact of k for small d explains the smaller exponent in the fitted linear model for $k = 4$ compared to $k = 1, 2$.

The source code, the statistical data underlying the plots, the instances, and the script used for benchmarking are available for download on the project page.

7.6 Conclusion

We have presented a novel symbolic-numeric algorithm for counting the number of zeros of a polynomial system within a given region that can be seen as an extension of Pellet's test for counting roots of univariate polynomials. The efficiency of our method stems from the fact that we relate the solutions of the input polynomial system to the solutions of a lower degree truncated system. We have analyzed the precision requirement of our algorithm, i.e., we have given a bound on the size of the region for which the algorithm is guaranteed to succeed. We have shown that this bound is strongly dependent on the geometry of the solutions of the polynomial system similar to what we have seen for the univariate case in the previous chapter. We have seen that a Sage implementation of our approach for the bivariate case can lead to a significant improvement when integrated into a state of the art solver for bivariate systems.

References for Part II

[Abb14] John Abbott. "Quadratic Interval Refinement for Real Roots". In: *Communications in Computer Algebra* 28 (2014). Poster presented at the International Symposium on Symbolic and Algebraic Computation (ISSAC), 2006, pp. 3–12.

[AS05] Alkiviadis G. Akritas and A. Strzeboński. "A comparative study of two real root isolation methods". In: *Nonlinear Analysis:Modelling and Control* 10.4 (2005), pp. 297–304.

[Bat+13] D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. *Numerically Solving Polynomial Systems with Bertini:* Software, Environments, and Tools. Society for Industrial and Applied Mathematics, 2013.

[Bec12] Ruben Becker. "The Bolzano Method to Isolate the Real Roots of a Bitstream Polynomial". Bachelor Thesis. Saarland University, Saarbrücken, Germany, 2012.

[Bec+16] Ruben Becker, Michael Sagraloff, Vikram Sharma, Juan Xu, and Chee Yap. "Complexity Analysis of Root Clustering for a Complex Polynomial". In: *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2016, Waterloo, ON, Canada, July 19-22, 2016*. Ed. by Sergei A. Abramov, Eugene V. Zima, and Xiao-Shan Gao. ACM, 2016, pp. 71–78.

[Ber+13] Eric Berberich, Pavel Emeliyanenko, Alexander Kobel, and Michael Sagraloff. "Exact symbolic-numeric computation of planar algebraic curves". In: *Theor. Comput. Sci.* 491 (2013), pp. 1–32.

[Bes49] G.C. Best. "Notes on the Graeffe method of root squaring". In: *American Mathematical Monthly* 56.2 (1949), pp. 91–94.

[BR14] Dario A. Bini and Leonardo Robol. "Solving secular and polynomial equations: A multiprecision algorithm". In: *Journal of Computational and Applied Mathematics* 272 (2014), pp. 276–292.

[BF00] Dario Andrea Bini and Giuseppe Fiorentino. "Design, Analysis, and Implementation of a Multiprecision Polynomial Rootfinder". English. In: *Numerical Algorithms* 23 (2-3 2000), pp. 127–173.

[BS16] Cornelius Brand and Michael Sagraloff. "On the Complexity of Solving Zero-Dimensional Polynomial Systems via Projection". In: *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2016, Waterloo, ON, Canada, July 19-22, 2016*. Ed. by Sergei A. Abramov, Eugene V. Zima, and Xiao-Shan Gao. ACM, 2016, pp. 151–158.

[BL24] L.E.J. Brouwer and B. de Loer. "Intuitionischer Beweis des Fundamentalsatzes der Algebra". In: Amsterdam Konigl. Acad. Van Wetenschapen Proc., 27 (1924), pp. 186–188.

[Buc06] Bruno Buchberger. “Bruno Buchberger’s PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal”. In: *Journal of Symbolic Computation* 41.3 (2006). Logic, Mathematics and Computer Science: Interactions in honor of Bruno Buchberger (60th birthday), pp. 475–511.

[Bur+08] Michael A. Burr, Sung Woo Choi, Benjamin Galehouse, and Chee-Keng Yap. “Complete subdivision algorithms, II: isotopic meshing of singular algebraic curves”. In: *Symbolic and Algebraic Computation, International Symposium, ISSAC 2008, Linz/Hagenberg, Austria, July 20-23, 2008, Proceedings*. 2008, pp. 87–94.

[BK12] Michael Burr and Felix Krahmer. “SqFreeEVAL: An (almost) optimal real-root isolation algorithm”. In: *Journal of Symbolic Computation* 47.2 (2012), pp. 153–166.

[Col16] G. E. Collins. “Continued fraction real root isolation using the Hong bound”. In: *Journal of Symbolic Computation* 72 (2016), pp. 21–54.

[CA76] G. E. Collins and A. G. Akritas. “Polynomial Real Root Isolation Using Descartes’ Rule of Signs”. In: *Symposium on symbolic and algebraic computation (SYMSAC)*. 1976, pp. 272–275.

[CK92] George E. Collins and Werner Krandick. “An Efficient Algorithm for Infallible Polynomial Complex Root Isolation”. In: *Papers from the International Symposium on Symbolic and Algebraic Computation*. ISSAC ’92. Berkeley, California, USA: ACM, 1992, pp. 189–194.

[CLO05] D.A. Cox, J. Little, and D. O’Shea. *Using Algebraic Geometry*. Graduate Texts in Mathematics. Springer New York, 2005.

[DKS13] Carlos D’Andrea, Teresa Krick, and Martín Sombra. “Heights of varieties in multiprojective spaces and arithmetic Nullstellensätze”. eng. In: *Annales scientifiques de l’École Normale Supérieure* 46.4 (2013), pp. 549–627.

[Dav85] James H. Davenport. *Computer algebra for Cylindrical Algebraic Decomposition*. Tech. Rep. S-100 44, Stockholm, Sweden: The Royal Inst. of Technology, Dept. of Numerical Analysis and Computing Science, 1985.

[DSY07] Zilin Du, Vikram Sharma, and Chee K. Yap. “Amortized Bound for Root Isolation via Sturm Sequences”. In: *Symbolic-Numeric Computation*. Ed. by Dongming Wang and Lihong Zhi. Basel: Birkhäuser Basel, 2007, pp. 113–129.

[Eig08] Arno Eigenwillig. “Real Root Isolation for Exact and Approximate Polynomials Using Descartes’ Rule of Signs”. PhD thesis. Saarland University, 2008.

[Eig+05] Arno Eigenwillig, Lutz Kettner, Werner Krandick, Kurt Mehlhorn, Susanne Schmitt, and Nicola Wolpert. “A Descartes Algorithm for Polynomials with Bit-Stream Coefficients”. In: *Computer Algebra in Scientific Computing: 8th International Workshop, CASC 2005, Kalamata, Greece, September 12-16, 2005. Proceedings*. Ed. by Victor G. Ganzha, Ernst W. Mayr, and Evgenii V. Vorozhtsov. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 138–149.

[EMT10] Ioannis Z. Emiris, Bernard Mourrain, and Elias P. Tsigaridas. “The DMM bound: multivariate (aggregate) separation bounds”. In: *Symbolic and Algebraic Computation, International Symposium, ISSAC 2010, Munich, Germany, July 25-28, 2010, Proceedings*. Ed. by Wolfram Koepf. ACM, 2010, pp. 243–250.

[EP05] Ioannis Z. Emiris and Victor Y. Pan. “Improved algorithms for computing determinants and resultants”. In: *J. Complexity* 21.1 (2005), pp. 43–71.

[EPT14] Ioannis Z. Emiris, Victor Y. Pan, and Elias P. Tsigaridas. "Algebraic Algorithms". In: *Computing Handbook, Third Edition: Computer Science and Software Engineering*. 2014, 10: 1–30.

[Fau02] Jean Charles Faugère. "A New Efficient Algorithm for Computing Gröbner Bases Without Reduction to Zero (F5)". In: *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*. ISSAC '02. Lille, France: ACM, 2002, pp. 75–83.

[For02] Steven Fortune. "An Iterated Eigenvalue Algorithm for Approximating Roots of Univariate Polynomials". In: *Journal of Symbolic Computation* 33.5 (2002), pp. 627–646.

[Gal14] François Le Gall. "Powers of tensors and fast matrix multiplication". In: *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*. Ed. by Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó. ACM, 2014, pp. 296–303.

[GG03] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. 2nd edition. Cambridge, UK: Cambridge University Press, 2003.

[Giu+05] M. Giusti, G. Lecerf, B. Salvy, and J.-C. Yakoubsohn. "On Location and Approximation of Clusters of Zeros of Analytic Functions". In: *Foundations of Computational Mathematics* 5.3 (2005), pp. 257–311.

[HL17] Jonathan D. Hauenstein and Viktor Levandovskyy. "Certifying solutions to square systems of polynomial-exponential equations". In: *Journal of Symbolic Computation* 79.Part 3 (2017). SI: Numerical Algebraic Geometry, pp. 575–593.

[Hau] Herwig Hauser. *https://imaginary.org/users/herwig-hauser*.

[Hou59] Alston S. Householder. "Dandelin, Lobacevskii, or Graeffe". In: *The American Mathematical Monthly* 66.6 (1959), pp. 464–466.

[KS15a] Michael Kerber and Michael Sagraloff. "Root refinement for real polynomials using quadratic interval refinement". In: *Journal of Computational and Applied Mathematics* 280 (2015). Announced at ISSAC'11, pp. 377–395.

[Kir98] Peter Kirrinnis. "Partial Fraction Decomposition in C(z) and Simultaneous Newton Iteration for Factorization in C[z]". In: *Journal of Complexity* 14.3 (1998), pp. 378–444.

[KRS16] Alexander Kobel, Fabrice Rouillier, and Michael Sagraloff. "Computing Real Roots of Real Polynomials ... and now For Real!" In: *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2016, Waterloo, ON, Canada, July 19-22, 2016*. Ed. by Sergei A. Abramov, Eugene V. Zima, and Xiao-Shan Gao. ACM, 2016, pp. 303–310.

[KS13] Alexander Kobel and Michael Sagraloff. "Fast Approximate Polynomial Multipoint Evaluation and Applications". In: *CORR* abs/1304.8069 (2013).

[KS15b] Alexander Kobel and Michael Sagraloff. "On the complexity of computing with planar algebraic curves". In: *J. Complexity* 31.2 (2015), pp. 206–236.

[Laz09] Daniel Lazard. "Thirty years of Polynomial System Solving, and now?" In: *Journal of Symbolic Computation* 44.3 (2009). Polynomial System Solving in honor of Daniel Lazard, pp. 222–231.

[Llo75] N. G. Lloyd. "On Analytic Differential Equations". In: *Proceedings of the London Mathematical Society* s3-30.4 (1975), pp. 430–444.

[McN02] John Michael McNamee. “A 2002 Update of the Supplementary Bibliography on Roots of Polynomials”. In: *Journal of Computational and Applied Mathematics* 142.2 (2002), pp. 433–434.

[McN07] John Michael McNamee. *Numerical Methods for Roots of Polynomials*. Studies in Computational Mathematics 1. Elsevier Science, 2007.

[MP12] John Michael McNamee and Victor Y. Pan. “Efficient polynomial root-refiners: A survey and new record efficiency estimates”. In: *Computers & Mathematics with Applications* 63.1 (2012), pp. 239–254.

[MP13] John Michael McNamee and Victor Y. Pan. *Numerical Methods for Roots of Polynomials*. Studies in Computational Mathematics 2. Elsevier Science, 2013.

[MOS11] Kurt Mehlhorn, Ralf Osbild, and Michael Sagraloff. “A general approach to the analysis of controlled perturbation algorithms”. In: *Comput. Geom.* 44.9 (2011), pp. 507–528.

[MSW15] Kurt Mehlhorn, Michael Sagraloff, and Pengming Wang. “From approximate factorization to root isolation with application to cylindrical algebraic decomposition”. In: *Journal of Symbolic Computation* 66 (2015). Announced at ISSAC’13., pp. 34–69.

[MVY02] B. Mourrain, M.N. Vrahatis, and J.C. Yakoubsohn. “On the Complexity of Isolating Real Roots and Computing with Certainty the Topological Degree”. In: *Journal of Complexity* 18.2 (2002), pp. 612–640.

[MP09] Bernard Mourrain and Jean Pascal Pavone. “Subdivision methods for solving polynomial equations”. In: *J. Symb. Comput.* 44.3 (2009), pp. 292–306.

[MT09] Bernard Mourrain and Elias P. Tsigaridas. “On the complexity of complex root isolation”. In: *Proc. 10th Int. Symp. on Effective Methods in Algebraic Geometry (MEGA)*. Barcelona, Spain, 2009.

[NR96] C. Andrew Neff and John H. Reif. “An Efficient Algorithm for the Complex Roots Problem”. In: *J. Complexity* 12.2 (1996), pp. 81–115.

[Pan02] V. Pan. “Univariate Polynomials: Nearly Optimal Algorithms for Numerical Factorization and Root Finding”. In: *Journal of Symbolic Computation* 33.5 (2002), pp. 701–733.

[Pan97] Victor Y. Pan. “Solving a Polynomial Equation: Some History and Recent Progress”. In: *SIAM Review* 39.2 (1997), pp. 187–220.

[Pan00] Victor Y. Pan. “Approximating Complex Polynomial Zeros: Modified Weyl’s Quadtree Construction and Improved Newton’s Iteration”. In: *Journal of Complexity* 16.1 (2000), pp. 213–264.

[PT13] Victor Y. Pan and Elias P. Tsigaridas. “On the Boolean Complexity of Real Root Refinement”. In: *Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation*. ISSAC ’13. Boston, Maine, USA: ACM, 2013, pp. 299–306.

[PT14] Victor Y. Pan and Elias P. Tsigaridas. “Accelerated Approximation of the Complex Roots of a Univariate Polynomial”. In: *Proceedings of the 2014 Symposium on Symbolic-Numeric Computation*. SNC ’14. Shanghai, China: ACM, 2014, pp. 132–134.

[Pin76] James R. Pinkert. “An Exact Method for Finding the Roots of a Complex Polynomial”. In: *ACM Trans. Math. Softw.* 2.4 (Dec. 1976), pp. 351–363.

[RS02] Q.I. Rahman and G. Schmeisser. *Analytic Theory of Polynomials*. London Mathematical Society monographs. Clarendon Press, 2002.

[Ren87] James Renegar. "On the worst-case arithmetic complexity of approximating zeros of polynomials". In: *Journal of Complexity* 3.2 (1987), pp. 90–113.

[Rou99] Fabrice Rouillier. "Solving Zero-Dimensional Systems Through the Rational Univariate Representation". In: *Applicable Algebra in Engineering, Communication and Computing* 9.5 (May 1999), pp. 433–461.

[RZ04] Fabrice Rouillier and Paul Zimmermann. "Efficient isolation of polynomial's real roots". In: *Journal of Computational and Applied Mathematics* 162.1 (2004). Proceedings of the International Conference on Linear Algebra and Arithmetic 2001, pp. 33–50.

[Rum10] Siegfried M. Rump. "Verification methods: rigorous results using floating-point arithmetic". In: *Symbolic and Algebraic Computation, International Symposium, ISSAC 2010, Munich, Germany, July 25-28, 2010, Proceedings*. Ed. by Wolfram Koepf. ACM, 2010, pp. 3–4.

[Sag12] Michael Sagraloff. "When Newton meets Descartes: A Simple and Fast Algorithm to Isolate the Real Roots of a Polynomial". In: *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*. ISSAC '12. Grenoble, France: ACM, 2012, pp. 297–304.

[Sag14a] Michael Sagraloff. "A Near-optimal Algorithm for Computing Real Roots of Sparse Polynomials". In: *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. ISSAC '14. Kobe, Japan: ACM, 2014, pp. 359–366.

[Sag14b] Michael Sagraloff. "On the complexity of the Descartes method when using approximate arithmetic". In: *Journal of Symbolic Computation* 65.0 (2014), pp. 79–110.

[SM16] Michael Sagraloff and Kurt Mehlhorn. "Computing real roots of real polynomials". In: *Journal of Symbolic Computation* 73 (2016), pp. 46–86.

[SY11] Michael Sagraloff and Chee K. Yap. "A Simple but Exact and Efficient Algorithm for Complex Root Isolation". In: *Proceedings of the 36th International Symposium on Symbolic and Algebraic Computation*. ISSAC '11. San Jose, California, USA: ACM, 2011, pp. 353–360.

[Sch82a] Arnold Schönhage. "Asymptotically Fast Algorithms for the Numerical Multiplication and Division of Polynomials with Complex Coeficients". In: *Computer Algebra, EUROCAM '82, European Computer Algebra Conference, Marseille, France, 5-7 April, 1982, Proceedings*. Ed. by Jacques Calmet. Vol. 144. Lecture Notes in Computer Science. Springer, 1982, pp. 3–15.

[Sch82b] Arnold Schönhage. *The Fundamental Theorem of Algebra in Terms of Computational Complexity*. Tech. rep. Math. Inst. Univ. Tübingen, 1982.

[Sch70] E. Schröder. "Über unendliche viele Algorithmen zur Auflösung der Gleichungen". ger. In: *Mathematische Annalen* 2 (1870), pp. 317–365.

[Sha08] V. Sharma. "Complexity of real root isolation using continued fractions". In: *Theoretical Computer Science* 409 (2008), pp. 292–310.

[SB15] Vikram Sharma and Prashant Batra. "Near Optimal Subdivision Algorithms for Real Root Isolation". In: *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*. ISSAC '15. Bath, United Kingdom: ACM, 2015, pp. 331–338.

[Sma81] Steve Smale. "The fundamental theorem of algebra and complexity theory". In: *Bull. Amer. Math. Soc. (N.S.)* 4.1 (Jan. 1981), pp. 1–36.

[Sto05] Arne Storjohann. “The shifted number system for fast linear algebra on integer matrices”. In: *J. Complexity* 21.4 (2005), pp. 609–650.

[Str12] Adam Strzeboński. “Real root isolation for exp–log–arctan functions”. In: *Journal of Symbolic Computation* 47.3 (2012), pp. 282–314.

[Stu02] B. Sturmfels. *Solving Systems of Polynomial Equations*. Conference Board of the Mathematical Sciences Regional Conference Series in Mathematics no. 97. American Mathematical Soc., 2002.

[Tsi13] Elias P. Tsigaridas. “Improved bounds for the CF algorithm”. In: *Theoretical Computer Science* 479 (2013), pp. 120–126.

[TE08] Elias P. Tsigaridas and Ioannis Z. Emiris. “On the complexity of real root isolation using continued fractions”. In: *Theor. Comput. Sci.* 392.1-3 (2008), pp. 158–173.

[Ver99] Jan Verschelde. “Algorithm 795: PHCpack: a general-purpose solver for polynomial systems by homotopy continuation”. In: *ACM Trans. Math. Softw.* 25.2 (1999), pp. 251–276.

[VH94] Jan Verschelde and Ann Haegemans. “Homotopies for Solving Polynomial Systems Within a Bounded Domain”. In: *Theor. Comput. Sci.* 133.1 (1994), pp. 165–185.

[Wey24] H. Weyl. “Randbemerkungen zu Hauptproblemen der Mathematik. II. Fundamentalsatz der Algebra and Grundlagen der Mathematik”. In: *Mathematische Zeitschrift* 20 (1924), pp. 131–151.

[Wil78] Herbert S. Wilf. “A Global Bisection Algorithm for Computing the Zeros of Polynomials in the Complex Plane”. In: *J. ACM* 25.3 (July 1978), pp. 415–420.

[Yak05] J.-C. Yakoubsohn. “Numerical analysis of a bisection-exclusion method to find zeros of univariate analytic functions”. In: *Journal of Complexity* 21.5 (2005), pp. 652–690.

[Yak00] Jean-Claude Yakoubsohn. “Finding a Cluster of Zeros of Univariate Polynomials”. In: *Journal of Complexity* 16.3 (2000), pp. 603–638.

[YSS13] Chee Yap, Michael Sagraloff, and Vikram Sharma. “Analytic Root Clustering: A Complete Algorithm Using Soft Zero Tests”. In: *The Nature of Computation. Logic, Algorithms, Applications - 9th Conference on Computability in Europe, CiE 2013, Milan, Italy, July 1-5, 2013. Proceedings*. Ed. by Paola Bonizzoni, Vasco Brattka, and Benedikt Löwe. Vol. 7921. Lecture Notes in Computer Science. Springer, 2013, pp. 434–444.

[Yap00] C.K. Yap. *Fundamental Problems of Algorithmic Algebra*. Oxford University Press, 2000.