Convex optimization with combinatorial characteristics: new algorithms for linear programming, min-cost flow, and other structured problems

Sally Dong

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2024

Reading Committee:

Yin Tat Lee, Chair

James R. Lee

Thomas Rothvoss

Program Authorized to Offer Degree:

Paul G. Allen School of Computer Science and Engineering

University of Washington

Abstract

Convex optimization with combinatorial characteristics:
new algorithms for linear programming, min-cost flow, and other structured problems

Sally Dong

Chair of the Supervisory Committee:

Yin Tat Lee

Paul G. Allen School of Computer Science and Engineering

This thesis focuses on algorithmic questions arising from discrete mathematics, with a particular emphasis on optimization on planar graphs. Historically, research in this area followed in one of two approaches: 1). Design problem-specific algorithms that are combinatorial in nature and make use of structures in the underlying discrete object [148], and 2). Cast the problem as a general optimization problem, typically a linear program (LP), and apply a general-purpose LP solver [52, 98, 145, 161]. My work unifies the two approaches in the design and analysis of fast algorithms, guided by the question: *How can we customize general-purpose convex optimization techniques to apply to problems with significant underlying structural properties?* By combining a wide-ranging set of tools under this paradigm, including convex analysis, sketching algorithms, data structures, numerical linear algebra, and structural combinatorics, we are able to design new algorithms for cornerstone problems in theoretical computer science, with runtimes that are significant improvements over the existing state-of-the-art. This thesis contains the following results:

1. The *first* high-accuracy LP solver for *α-separable* LPs with n constraints and m variables. The algorithm runs in $\widetilde{O}(m^{1/2+2\alpha})$ time [57], compared to the previous best $\widetilde{O}(m^{\omega})$ time with no consideration for separability [44], where $\omega \approx 2.37$ is the matrix-multiplication constant. A special case here is *planar* LPs in $\widetilde{O}(n^{1.5})$ time;

2. The *fastest* algorithm to solve min-cost flow on planar graphs with n vertices in $\widetilde{O}(n)$ time [56], which is *nearly-optimal*, and predates the current best *almost-optimal*

algorithm for general graphs [39];

3. The *first* high-accuracy LP solver for LPs with n constraints, m variables, and *treewidth* τ. The algorithm runs in $\widetilde{O}(m\tau^{(\omega+1)/2})$ time [58];

4. The *first* algorithm to solve min-cost flow on graphs with treewidth τ, running in $\widetilde{O}(m\tau^{1/2} + n\tau)$ time [60];

5. The *fastest* algorithm to solve k-commodity flow on planar graphs on n vertices in $\widetilde{O}(k^{2.5}n^{1.5})$ time [57], compared to the previous best $\widetilde{O}(k^{\omega}n^{\omega})$ time with no consideration for planarity [44];

6. The *fastest* algorithm to compute circle-packing representations of planar graphs [59], with an improvement of a cubic factor over the existing algorithm [129].

Acknowledgments

First and most importantly, I give my deepest gratitude to my advisor Yin Tat Lee for his guidance and support throughout my PhD. Yin Tat is the most brilliant theory researcher I know, and I am forever grateful to have completed this collaboration with him, though we both know he easily could have done in six months what took me six years! I thank him for also having given me a tremendous amount of freedom to explore my interests or simply be unproductive; these experiences were ultimately invaluable for me to grow into an independent, self-driven researcher. As an advisor, Yin Tat was uniquely approachable as if just another grad student, and I will fondly remember our Slack channels filled with crying-laughing emojis.

I thank Thomas Rothvoss for being my second adoptive advisor. Thomas is the platonic ideal of a theory researcher – he is selective about projects to pursue, develops thorough background understanding, produces big, creative results, and values the clear communication of ideas through his writing. As the advisor on a project, he showed up to every meeting with ideas, questions, and the patience to step through proofs in detail with me. I was incredibly fortunate to have worked with him, and can only aspire to be more like him in the future.

I thank my collaborators, Kent Quanrud, Guanghao Ye, Sushant Sachdeva, Yu Gao, Gramoz Goranci, Richard Peng, Swati Padmananbhan, Haotian Jiang, and Lawrence Li, for shaping my research experience. In particular, I thank Guanghao, who did much of the hard stuff in our shared papers; the most fun periods of my PhD were the weeks before each conference deadline when we would work frantically around the clock together and submit within two minutes of the cutoff time.

I thank the rest of the theory group at UW. Each of the faculty, Paul Beame, Anna Karlin, James Lee, Rachel Lin, Shayan Oveis Gharan, Anup Rao, and Stefano Tessaro, is an inspiring researcher and a friendly presence around the department. Paul and Anna became trusted sources for career advice. My year had a large theory cohort, including Siddarth Iyer, Haotian Jiang, Nathan Klein, Kuikui Liu, Victor Reis, Ruoqi Shen, Oscar Sprumont, and Ewin Tang; alongside them my PhD experience was all the merrier. I thank especially my academic sister

Swati, whose sunny smile and warm attitude always brightened my day.

I thank all the researchers and PhD students with whom I crossed paths at various conferences, workshops, and during the research semester at the University of Bonn. Attending these events always invigorated my spirit as a researcher and my love for theory. The community is truly a big group of like-minded friends.

I thank Elise DeGoede Dorough for being the best administrator in the whole wide world who makes the UW/CSE bureaucracy so simple to navigate for grad students. I thank the National Science and Engineering Research Council of Canada for partially funding this PhD.

I thank my colleagues at the Voleon Group in Berkeley where I spent a summer, and at Amazon in Luxembourg where I spent six months. They welcomed me into their ranks and showed me research in the real world, which included unthinkable things like writing code to parse input data. In particular, I thank Yash and Nikos for their mentorship during this time and their lasting friendship.

I thank the UW political science department, where I spent a large chunk of my final year. It was the most delightful distraction from this dissertation. I admire their clarity of thought, eloquence, and openness to debate.

I thank my friends who have been a part of my journey. They have been a source of entertainment and support, and have influenced me immensely in my personal growth. In particular, I thank Armin for accepting my personality all these years; Lemuel for always hearing me; the Nani kids for their silly nonsense; Matt for the adventures; Victor for his generous spirit; Ofir for his extroversion, humor, and Israeli bluntness; Ian for his wit and his interesting number of thoughts; Aleks for being an aspiration; Lily for showing me strength in vulnerability; Adam for indulging my whims; and Stephanie for her grace.

I thank my family for always warmly welcoming me back to my true home.

Lastly, I thank my teachers and professors, among others Mr. Chomiuk, Eric Blais, Bill Cook, and Jim Geelen, for having given me the courage to set off on this path.

to all my mentors along the way

Table of Contents

Page

Preface

This thesis is organized as a monograph, rather than simply a compilation of my published papers. With this decision, I hope it can serve as a self-contained exposition on the topic of structured linear program solvers for future researchers.

The technical content is primarily based on the sequence of papers [58, 56, 57, 60], written with my co-authors Yu Gao, Gramoz Goranci, Yin Tat Lee, Lawrence Li, Richard Peng, Sushant Sachdeva, and Guanghao Ye. The robust interior point method in Chapter 2 was first introduced by Cohen, Lee, and Song [43], and has more general applications than what is discussed here. Due to the significant iterative changes across these papers, as well as their length and density of notation, the core ideas have been somewhat obfuscated. This thesis is an effort to remedy the situation, by presenting the material in a complete and logical manner.

I would like to thank the anonymous reviewers who gave helpful comments and feedback throughout this line of work. In particular, one reviewer for our submission to the Journal of the ACM painstaking went through the 100-page manuscript and found many errors and oversights in our proofs. I am grateful for their time and their rigorous standard.

Notation

Computational model.

In our algorithms, we assume arithmetic operations are performed exactly using real numbers in $O(1)$ time.

General.

- $\mathbb{R}$ denotes the set of real numbers, and $\mathbb{R}_{\geq 0}$ denotes the set of non-negative real numbers. By default, variables that are unspecified are assumed to be real numbers.
- $\mathbb{Z}$ denotes the set of integers, and $\boldsymbol{z}_{\geq 0}$ denotes the set of non-negative integers.
- Given a set S and an element v, we use $S - v$ as a shorthand to denote $S \setminus \{v\}$.
- Vectors are denoted by bold lowercase letters (Latin and Greek), such as $\boldsymbol{v}$.
- Matrices are denoted by bold uppercase letters, such as $\mathbf{M}$.
- Scalars are denoted by regular, unbolded letters, such as x.
- We use $\mathbf{0}$ for the all-zeros vector and matrix; $\mathbf{I}$ for the identity matrix; and $\mathbf{1}$ for the all-ones vector and matrix. The dimensions should always be determined by context.
- For two scalars x and y, we define $x \leq_t y$ to mean $x \leq e^t y$, and analogously $\geq_t$.
- $\omega \approx 2.37$ is the fast matrix-multiplication constant.
- We use standard big-O notation. Additionally, $\widetilde{O}(f(n))$ is used to hide logarithmic factors, i.e. $\widetilde{O}(f(n)) \stackrel{\text{def}}{=} O(f(n) \log^k f(n))$ for some universal constant k.

Calculus.

- For an n-dimensional, three-times differentiable function $f(\boldsymbol{x})$, we use $D^3 f(\boldsymbol{x})$ to denote the tensor of third-order partial derivatives.
- Given some set K in Euclidean space, we use $\operatorname{int}(K)$ to denote its interior, and $\partial(K)$ to denote its boundary.
- We use $B(\boldsymbol{x}, r)$ to denote the ball centered at point $\boldsymbol{x}$ with radius r.
- We make liberal use of Taylor expansions, usually without specifying.

Linear algebra.

- The transpose of a matrix or vector is denoted by a $^\top$ superscript.
- For any vector $\boldsymbol{v}$ and scalar x, we define $\boldsymbol{v} + x$ to be the vector obtained by adding x to each entry of $\boldsymbol{v}$, and similarly $\boldsymbol{v} - x$ the vector obtained by subtracting x from each entry of $\boldsymbol{v}$.
- For two vectors $\boldsymbol{x}, \boldsymbol{y}$, we use $\boldsymbol{x} \circ \boldsymbol{y}$ to denote their entry-wise product.
- For any vector $\boldsymbol{v}$, its Euclidean-norm is denoted by $\|\boldsymbol{v}\|_2$. Its local norm with respect to a symmetric matrix $\mathbf{M}$ is denoted by $\|\boldsymbol{v}\|_{\mathbf{M}} \stackrel{\text{def}}{=} \sqrt{\boldsymbol{v}^\top \mathbf{M} \boldsymbol{v}}$. Its zero norm, or equivalently, its number of non-zero entries, is denoted by $\mathrm{nnz}(\boldsymbol{v})$.
- For a symmetric $n \times n$ matrix $\mathbf{M}$, its real eigenvalues are ordered $\lambda_1(\mathbf{M}) \leq \lambda_2(\mathbf{M}) \leq \cdots \leq \lambda_n(\mathbf{M})$.
- The standard basis vectors are denoted by $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n$ in n-dimensional space.
- For a vector $\boldsymbol{x}$, we use $\boldsymbol{x}_i$ to denote its i-th entry, and we use $\boldsymbol{x}_{[i]}$ to denote its i-th block; that is, $\boldsymbol{x}_{[i]}$ is a vector, and $\boldsymbol{x}$ is the concatenation of $\boldsymbol{x}_{[1]}, \boldsymbol{x}_{[2]}, \ldots$, up to some $\boldsymbol{x}_{[m]}$.
- For any vector $\boldsymbol{x}$ and index set S, we use $\boldsymbol{x}|_S$ to denote the vector that is equal to $\boldsymbol{x}$ on S and zero outside S.
- For a matrix $\mathbf{M}$ and index sets S, T, we use $\mathbf{M}_{S,T}$ to denote the matrix equal to $\mathbf{M}$ on $S \times T$ and zero everywhere else. Note that we omit the vertical bar that is used for vectors.
- For two positive semidefinite (PSD) matrices $\mathbf{A}$ and $\mathbf{B}$, we write $\mathbf{A} \preccurlyeq \mathbf{B}$ to mean $\mathbf{B} - \mathbf{A}$ is positive semidefinite. We write $\mathbf{A} \approx_t \mathbf{B}$ to mean $e^{-t}\mathbf{A} \preccurlyeq \mathbf{B} \preceq e^t\mathbf{A}$.
- When multiplying two matrices of different dimensions, say an $m \times n$ matrix with an $n \times k$ matrix, we first partition both matrices into blocks of size $\min\{m, n, k\}$, and then use fast matrix-multiplication block-wise. The most common example, multiplying a $m \times n$ matrix with an $n \times n$ matrix, where $m \geq n$, takes $(m/n)(n^\omega)$ time.
- For any matrix $\mathbf{M}$, we use $\mathbf{M}^{-1}$ to denote the inverse of $\mathbf{M}$ if it is an invertible matrix, and the unique Moore-Penrose pseudo-inverse otherwise.

Graphs.

- In general, a graph is denoted by $G = (V, E)$, and has n vertices and m edges.
- In the discussion of flow problems, graphs are assumed to be directed unless stated otherwise.
- When we say a graph is edge-weighted, we mean an assignment of some positive weight to each edge in the graph.
- Given a planar graph $G = (V, E)$, we denote its dual graph by $G^* = (V^*, E^*)$.

- A hypergraph is a generalization of a graph. It has a set of vertices as a graph does, and a set of hyperedges. An edge can be viewed as a set of two vertices; a hyperedge generalizes this to be a set of any number of vertices.

Trees as data structures.

- While trees are graphs, we will often make use of them as data structures. When this is the case, we use calligraphic font, such as $\mathcal{T}$.
- We refer to vertices in a tree as nodes, so as to distinguish them from purely graph-theoretic objects.
- For a rooted tree $\mathcal{T}$, we write $H \in \mathcal{T}$ to mean H is a node in $\mathcal{T}$.
- $\mathcal{T}_H$ denotes the complete subtree of $\mathcal{T}$ rooted at node H.
- We say a node A is an ancestor of H and H is a descendant of A if H is in the subtree rooted at A, and $H \neq A$.
- Given a set of nodes $\mathcal{H}$ in a rooted tree $\mathcal{T}$, we use $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$ to denote the set of nodes in $\mathcal{T}$ that are either in $\mathcal{H}$ or are ancestors of some node in $\mathcal{H}$.
- The *level* of a node in a tree has the following properties: the root is at level 0; the maximum level is one less than the height of the tree; and the level of a node must be at least one greater than the level of its parent, but this difference does not have to be equal to one. We may assign levels to nodes arbitrarily as long as the above is satisfied.
- $\mathcal{T}(i)$ denotes the collection of all nodes at level i in $\mathcal{T}$.
- $\mathcal{P}_{\mathcal{T}}(\mathcal{H}, i)$ denotes the subset of $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$ at level i.

Iterative algorithms.

- Step 0 in an iterative algorithm, for example, the robust interior point method, refers to the initialization step.
- For $k > 0$, step k refers to the k-th iteration of the algorithm, which is the k-th time the solution is updated.
- If an algorithm updates a variable $\boldsymbol{x}$ iteratively, then $\boldsymbol{x}^{(k)}$ denotes the state of $\boldsymbol{x}$ at the end of the k-th iteration.
- Our iterative algorithms often contain subroutines that are called at every iteration with vectors as input. In these cases, the true input is not the full vector, but rather only the entries that have changed compared to the input in the previous iteration. Similarly, when the output is a vector, the true output is not the full vector but rather only the changed entries compared to the previous output. This distinction is key in bounding our runtimes, and can be implemented by representing vectors as lists.

Chapter 1

Introduction

Linear programming is one of the most fundamental problems in computer science and optimization. General techniques for solving linear programs, such as simplex methods [52], ellipsoid methods [105], and interior point methods [98], have been developed and continuously refined since the 1940s, and have later been found to be useful in a wide range of problems spanning optimization, combinatorics, and machine learning.

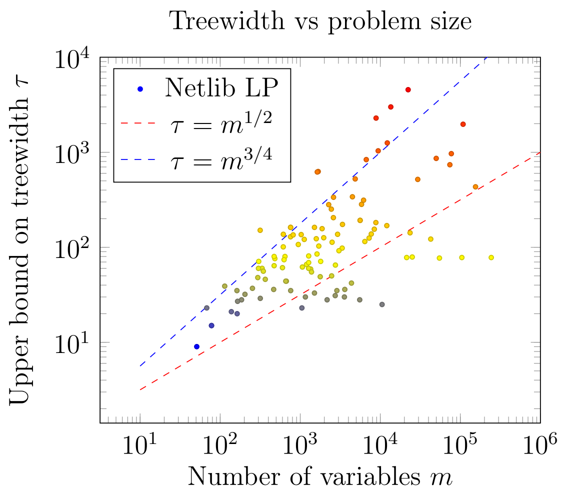
For a standard linear program

$$\begin{aligned} \min\ & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{s.t.}\ & \mathbf{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{l} \le \boldsymbol{x} \le \boldsymbol{u} \end{aligned} \tag{LP}$$

where $\mathbf{A} \in \mathbb{R}^{n\times m}$ is a full-rank matrix, the current fastest high-accuracy algorithms take either $\widetilde{O}(m^{2.373}\log(1/\varepsilon))$ time [43, 94] or $\widetilde{O}((\sqrt{nm\cdot \mathrm{nnz}(\mathbf{A})} + n^{2.5})\log(1/\varepsilon))$ time [165, 30] to solve (LP) to ε accuracy. The current fastest exact algorithms for linear program take either $\exp\left(O\left(\sqrt{n\log\frac{m-n}{n}}\right)\right)$ time [81], or the runtime depends on the magnitude of entries of $\mathbf{A}$. In this thesis, we are interested in high-accuracy algorithms for (LP).

When $\mathbf{A}$ is a dense matrix, in other words, the number of non-zero entries in $\mathbf{A}$ is on the order of mn, these runtimes are close to optimal, as they nearly match the runtime $\widetilde{O}((\mathrm{nnz}(\mathbf{A}) + n^{\omega})\log(1/\varepsilon))$ to solve the subproblem $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$, where $\omega \approx 2.373$ is the matrix multiplication exponent. When $\mathbf{A}$ is sparse, as is the case in many problems arising from both theory and applications, we ask if much faster runtimes are possible.

When n and m are the same order, this problem is highly non-trivial, even for linear systems. It is only recently known how to solve a $n \times m$ sparse linear system in slightly faster than m^{ω} time [142], and sub-quadratic time is insurmountable under the current techniques. It turns out in practice however, sparse linear systems often exhibit additional structure; in particular, separability is a condition much stronger than mere sparsity. For example, many of the linear programs in the Netlib repository have sublinear *treewidth*, a parameter tightly connected to separability (Fig. 1.1). For these structured linear systems, a small polynomial dependence on their structural parameter still implies a much faster than quadratic runtime, hence making them a particularly suitable target of study for this thesis.



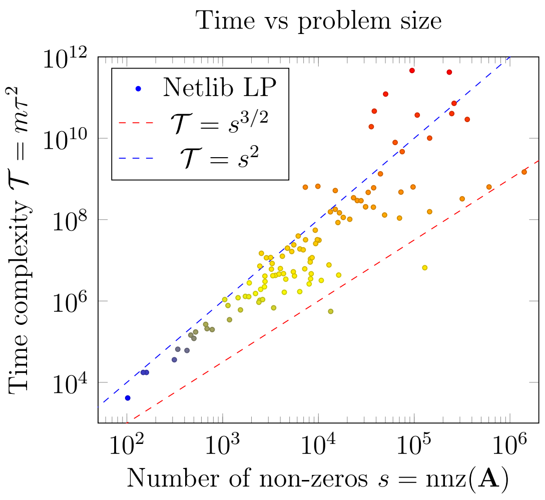


Figure 1.1: The left plot shows some upper bound τ of treewidth vs problem dimension m for all 109 feasible linear program instances in Netlib repository. We compute a upper bound of treewidth using [99]. This shows that treewidth is between $m^{1/2}$ and $m^{3/4}$ for many linear programs in this data set. The right plot shows that the runtime $m\tau^2$ is sub-quadratic in the input size $\mathrm{nnz}(\mathbf{A})$ for many linear programs in this data set.

The inspiration for focusing on separability stems from nested dissection [122] and methods in numerical linear algebra [53]. The majority of recent developments in the design of efficient linear program solvers have not leveraged these ideas underlying these faster linear system solvers nor exploited any structure of the constraint matrix. As such, this work fills an important gap in the existing literature.

Separability of a linear program can be captured succinctly by associating the constraint matrix with a graph, for which there are very natural structural definitions. A special case of *separable* is *planar*, and another special case is *bounded-treewidth*. We will present algorithms for all three settings. In addition to solving the standard linear programming formulation, we also present new results for flow problems, because the structure in the input graph readily translates to structure in the constraint matrix.

In our most general result, we consider (LP) where the associated graph is known to be n^α-separable. We present an $\widetilde{O}((m+m^{1/2+2\alpha})\log(1/\epsilon))$-time algorithm for solving these problems to ϵ relative accuracy. A notable case is when $\alpha = 1/2$, which includes the family of planar and bounded-genus graphs [123]. It has also been empirically observed that road networks on n vertices have separators of size $n^{1/3}$[54, 147]. Our result should be compared against the natural $\widetilde{O}(m^{1/2}(m + m^{\alpha\omega}))$ runtime, which directly follows from the fact that IPM-based methods require $\widetilde{O}(\sqrt{m})$ iterations, each of which can be implemented in $O(m + m^{\alpha\omega})$ time using the nested dissection [122, 8] algorithm. For linear programs whose dual graph are $O(n^{\alpha})$-separable with $\alpha \leq 1/4$, our algorithm achieves $\widetilde{O}(m \log(1/\varepsilon))$ time, which is optimal up to poly-logarithmic factors. Another implication of this result is an $\widetilde{O}(k^{5/2}m^{3/2}\log(1/\epsilon))$-time algorithm to approximately solve the k-multicommodity flow problem on planar graphs. Obtaining an LP solver whose time complexity is $\widetilde{O}(m + m^{\alpha\omega})$, which would in turn nearly match the time complexity for solving linear systems with recursively separable structure, remains an outstanding open problem [78].

Using the same techniques, we further consider when the LP constraint matrix has treewidth bounded by τ. Assuming a width τ tree decomposition is given, we design an algorithm for solving (LP) in $\widetilde{O}(m \cdot \tau^{(\omega+1)/2}\log(1/\varepsilon))$ time. When no tree decomposition is given, we can combine our result with recent techniques in vertex-capacitated flow [20] for an algorithm with $\widetilde{O}(m^{1+o(1)} \cdot \mathrm{tw}^2 \log(1/\varepsilon))$ runtime, where tw is the treewidth of the problem. Beyond the practical consideration, whether there is a $\widetilde{O}(n \cdot \mathrm{tw}^{O(1)})$ LP algorithm is important in parameterized complexity. Most algorithms designed for low treewidth graphs rely on dynamic programming, which naturally give algorithms with runtime exponential in treewidth even for problems in P, such as reachability and shortest paths [5, 37, 38, 143]. There are only a few problems in P that we know how to solve in $\widetilde{O}(n \cdot \mathrm{tw}^{O(1)})$ time [73]. We refer to the related works section for a discussion of these problems.

For min-cost flow, we consider the planar and bounded treewidth setting. We design a nearly-linear time algorithm for min-cost flow in planar graphs with polynomially bounded integer costs and capacities. The previous fastest algorithm for this problem is based on IPMs and works for general sparse graphs in $O(n^{1.5} \cdot \mathrm{poly}(\log n))$ time [51]. We also obtain an algorithm for min-cost flow in graphs with n vertices and m edges, given a tree decomposition of width τ and size S, and polynomially bounded, integral edge capacities and costs, running in $\widetilde{O}(m\sqrt{\tau} + S)$ time, where we use the exact same techniques as the planar result. In general graphs where treewidth is trivially bounded by n, the algorithm runs in $\widetilde{O}(m\sqrt{n})$ time, which is the best-known result without using the Lee-Sidford barrier or ℓ_1 IPM, demonstrating the surprising power of robust interior point methods. Finally, as a corollary, we obtain a $\widetilde{O}(\mathrm{tw}^3 \cdot m)$ time algorithm to compute a tree decomposition of width $O(\mathrm{tw} \cdot \log(n))$, given a graph with m edges.

1.1 Challenges

In this section, we discuss a few alternate approaches and why they likely prove unfruitful, in order to place our result in context.

Dynamic programming for low-treewith linear programming. Dynamic programming is a natural first approach, as has been applied to other low-treewidth problems. To explain the difficulty of achieving fully-polynomial-time fixed-parameter tractability in the optimization setting, we consider the following simplified problem: Given a graph $G = (V, E)$ with a convex function $f_e : \mathbb{R}^2 \to \mathbb{R}$ for every edge $e \in E$, consider the objective function on $\mathbb{R}^V$ defined by

$$f_G(\boldsymbol{x}) = \sum_{ij \in E} f_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j). \tag{1.1}$$

To divide the problem into smaller one, we consider any small balanced vertex separator $S \subset V$; namely V is partition into three sets S, L and R such that there are no edges between L and R. We can write the objective function $f(\boldsymbol{x})$ by

$$f_G(\boldsymbol{x}) = f_L(\boldsymbol{x}) + f_R(\boldsymbol{x}) + f_{G-E(L)-E(R)}(\boldsymbol{x}),$$

where $f_T(x) = \sum_{ij \in E(T)} f_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and $E(T)$ is the set of edges with at least one end point in T. To minimize f_G, it suffices to fix $\boldsymbol{x}_S$ and recursively minimizing $\boldsymbol{x}$ on L and R, and minimize over all fixed $\boldsymbol{x}_S$. Namely,

$$\min_{\boldsymbol{x}} f_G(\boldsymbol{x}) = \min_{\boldsymbol{x}_S} f_{G-E(L)-E(R)}(\boldsymbol{x}_S) + \widetilde{f_L}(\boldsymbol{x}_S) + \widetilde{f_R}(\boldsymbol{x}_S).$$

where $\widetilde{f_L}(\boldsymbol{x}_S) = \min_{\boldsymbol{x}_L} f(\boldsymbol{x}_S, \boldsymbol{x}_L)$ and $\widetilde{f_R}(\boldsymbol{x}_S) = \min_{\boldsymbol{x}_R} f(\boldsymbol{x}_S, \boldsymbol{x}_R)$. Here, we crucially use the fact that $f_{G-E(L)-E(R)}(\boldsymbol{x})$ depends only on the variables in S, but not L and R; the term f_L depends only on the variables in L and S, but not R; similarly for f_R. In general, if f is convex, then both $\widetilde{f_L}$ and $\widetilde{f_R}$ are convex functions on $\mathbb{R}^S$. Hence, the formula shows that we can solve the optimization problem by first constructing the reduced problem on $G[L]$ and $G[R]$, then solve a size $|S|$ optimization problem.

If the f_{ij}'s are all quadratic functions, then both $\widetilde{f_L}$ and $\widetilde{f_R}$ are quadratic functions, and it turns out they can be stored as matrices known as Schur complements. Hence, we can solve the problem with the approach described above; in fact, algebraic manipulation gives the sparse Cholesky factorization algorithm with runtime $\widetilde{O}(m \cdot \tau^2)$.

However, for general convex function f_G, it is not known how to store the functions $\widetilde{f_L}$ and $\widetilde{f_R}$ efficiently, and this will likely require runtime exponential in treewidth. At a high level, the reason is that before we solve the outer problem f_G, we do not know at which fixed x_S we should recurse on for $\widetilde{f_L}$ and $\widetilde{f_R}$. It is known that without adaptivity, exponentially many oracle calls are needed to minimize a general convex function [132, 16, 34]. This suggests we should compute $\widetilde{f_L}$ and $\widetilde{f_R}$ recursively for each different x_S. However, it is likely that we need to access at least two different points x_S, and this already leads to runtime recursion $T(n) \geq 4T(n/2) + O(1)$ which is at least n^2. Therefore, dynamic programming appears to be inefficient for general convex optimization.

Scanning through variables. When the underlying structure of the variable dependencies is simple enough, a simple scan through the variables may suffice for the problem at hand; for example, [62] successfully applies this approach for function-fitting problems on a path. To illustrate, consider a problem of the form

$$\min_{\boldsymbol{x}} F(\boldsymbol{x}) \overset{\text{def}}{=} f_1(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k) + f_2(\boldsymbol{x}_2, \ldots, \boldsymbol{x}_{k+1}) + f_3(\boldsymbol{x}_3, \ldots, \boldsymbol{x}_{k+2}) + \ldots.$$

Suppose $\boldsymbol{x}^*$ is the unique minimizer of the function and $\boldsymbol{x}_1^*, \boldsymbol{x}_2^*, \cdots, \boldsymbol{x}_{k-1}^*$ are given. By looking at the gradient of the function above at the first coordinate, we know that

$$\frac{\partial}{\partial \boldsymbol{x}_1} F(\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{x}_1} f_1(\boldsymbol{x}_1^*, \boldsymbol{x}_2^*, \cdots, \boldsymbol{x}_k^*) = \mathbf{0}.$$

Since $\boldsymbol{x}_1^*, \boldsymbol{x}_2^*, \cdots, \boldsymbol{x}_{k-1}^*$ is given, this is a one variable non-linear equation on $\boldsymbol{x}_k^*$ and it has a unique solution under mild assumptions. Solving these equations, we obtain $\boldsymbol{x}_k^*$. Now, looking at $\frac{\partial}{\partial \boldsymbol{x}_2} F(\boldsymbol{x})$, we have that

$$\frac{\partial}{\partial \boldsymbol{x}_2} F(\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{x}_2} f_1(\boldsymbol{x}_1^*, \boldsymbol{x}_2^*, \cdots, \boldsymbol{x}_k^*) + \frac{\partial}{\partial \boldsymbol{x}_2} f_2(\boldsymbol{x}_2^*, \boldsymbol{x}_3^*, \cdots, \boldsymbol{x}_{k+1}^*) = \mathbf{0}.$$

Since we already know $\boldsymbol{x}_1^*, \cdots, \boldsymbol{x}_k^*$, this is again a one variable non-linear equation. Therefore, we can solve this problem one variable at a time.

This approach can be modelled by an underlying graph structure in the following sense: Each variable $\boldsymbol{x}_i$ is represented by vertex i of the graph, and $i \sim j$ if there is some a term f_k dependent on both i and j. We say a vertex i is solved if we know $\boldsymbol{x}_i^*$. In the example above, the graph is a thick path, and if the first $k-1$ vertices are solved at the beginning, then we can follow the path to solve the remaining vertices one by one.

Unfortunately, this type of scan-based algorithm cannot be generalized. Consider a convex function of the form (1.1) where the graph G is a complete binary tree with n leaves. Let i be a vertex such that the subtree rooted at i is of height two containing four leaves. Observe that we cannot solve for the children of i by case analysis, if both i and the leaves are unsolved.

Since there are $n/4$ many subtree of height two in G, at least $n/4$ many variables must be known at the beginning, before we can follow the graph structure to solve for the remaining variables. As such, this approach does not produce any meaningful simplification.

Tightening the iterations bounds for interior point methods. Another natural approach for attacking the conjecture is to prove that existing polynomial time methods for linear program run faster automatically for graphs with low treewidth. Currently, there are two family of polynomial time algorithms – the ellipsoid method (more generally cutting plane methods) and interior point methods. For cutting plane methods, m iterations are needed in general, since the method only obtains one hyperplane per iteration, and we need m hyperplane simply to represent the solution even for the case of constant treewidth. In general, these hyperplanes are represented by dense vectors and will probably take m^2 time in total.

For interior point methods, the iteration bound is less clear since there is no information obstruction. In general, it is known that $O(\sqrt{m}\log(1/\varepsilon))$ iterations are needed to solve a linear program, and each iteration involves solving a linear system. For the case $n = \Theta(m)$ in particular, this bound has not been improved since the '80s. In fact, it has been shown that the standard interior point method used in practice indeed takes $\Omega(\sqrt{m}\log(1/\varepsilon))$ iterations in the worst case [131, 7], and some of these constructions have $O(1)$ treewidth. Even for concrete problems such as maximum flow, difficult instances for iterative methods often have treewidth $O(1)$ [103]. These lower bounds suggest that obtaining an optimization method with iteration count that is only a function of treewidth requires a substantively new algorithm.

In the case of the min-cost flow results, the $\Omega(\sqrt{n})$ term comes from the fact that IPM uses the electrical flow problem (ℓ_2-type problem) to approximate the shortest path problem (ℓ_1-type problem). This $\Omega(\sqrt{n})$ term is analogous to the flow decomposition barrier: in the worst case, we need $\Omega(n)$ shortest paths (ℓ_1-type problem) to solve the max-flow problem (ℓ_∞-type problem). Since ℓ_2 and ℓ_∞ problems differ a lot when there are s-t paths with drastically different lengths, difficult instances for electrical flow-based max-flow methods are often serial-parallel (see Figure 3 in [41] for an example). Therefore, planarity does not help to improve the $\sqrt{n}$ term. Although more general $\ell_2 + \ell_p$ primitives have been developed [2, 111, 3, 1], exploiting their power in designing current algorithms for exact max-flow problem has been limited to perturbing the IPM trajectory, and such a perturbation only works when the residual flow value is large. In all previous works tweaking IPMs for breaking the 3/2-exponent barrier [125, 126, 45, 100, 14], an augmenting path algorithm is used to send the remaining flow at the end. Due to the residual flow restriction, all these results assume unit-capacities on edges, and it seems unlikely that planarity can be utilized to design an algorithm for polynomially-large capacities with fewer than $\sqrt{n}$ IPM iterations.

Faster iterations via inverse maintenance. Dual to the previous approach is the idea of speeding up each iteration of interior point methods. Each iteration of these methods require some computation or maintenance involving a term $(\mathbf{AWA}^\top)^{-1}$ for some non-negative diagonal $\mathbf{W}$; previous work on linear programming focused on inverse maintenance techniques to accomplish this either explicitly or implicitly. In [43, 165, 94], the inverse is explicitly maintained and this takes at least n^2 time in total. [164, 30] focused on IPM for the bipartite matching problem and the maximum flow problem, where a sparsified Laplacian system $\mathbf{AWA}^\top \boldsymbol{x} = \boldsymbol{b}$ is solved directly in each iteration and hence the whole algorithm takes at least n per step and $n^{1.5}$ time in total, where d is the number of vertices. It seems that either approach cannot lead to nearly linear time (when $m = \Theta(n)$).

In our low-treewidth setting, one natural approach is to maintain the Cholesky factorization $\mathbf{LL}^\top = \mathbf{AWA}^\top$. This can be done in nearly-linear time in total, by combining ideas from numerical methods [53] and previous algorithms mentioned above. Unfortunately, in general, almost any sparse update in $\mathbf{W}$ leads to $\Omega(n)$ changes in $\mathbf{L}^{-1}$. Hence, it seems difficult to get a runtime faster than $n^{1.5}$ by just combining inverse maintenance with current knowledge of sparse Cholesky factorization.

When used to design graph algorithms, the i-th iteration of a robust IPM involves computing an electrical flow on G. The edge support remains unchanged between iterations, though the edge weights change. Further, if K_i is the number of edges with weight changes between G_i and G_{i+1}, then robust IPMs guarantee that

$$\sum_i \sqrt{K_i} = \widetilde{O}(\sqrt{m} \log M).$$

Roughly, this says that, on average, each edge weight changes only poly-log many times throughout the algorithm. Unfortunately, any sparsity bound is not enough to achieve nearly-linear time. Unlike the shortest path problem, changing *any* edge in a connected graph will result in the electrical flow changing on essentially *every* edge. Therefore, it is very difficult to implement (robust) IPMs in sublinear time per iteration, even if the subproblem barely changes every iteration. On moderately dense graphs with $m = \Omega(n^{1.5})$, this issue can be avoided by first approximating the graph by sparse graphs and solving the electrical flow on the sparse graphs. This leads to $\widetilde{O}(n) \ll \widetilde{O}(m)$ time cost per step [166]. However, on sparse graphs, significant obstacles remain. Recently, there has been a major breakthrough in this direction by using random walks to approximate the electrical flow [75, 163]. Unfortunately, this still requires $m^{1-\frac{1}{58}}$ time per iteration.

Our planar min-cost flow result follows the approach in [58] and shows that this dense graph can be sparsified. This is, however, subtle. Each step of the IPM makes a global update via the implicit representation, hence checking whether the flow is feasible takes at least linear time. Therefore, we need to ensure each step is exactly feasible despite the approximation. If we are unable to do that, the algorithm will need to fix the flow by augmenting paths at the end like [100, 14], resulting in super-linear time and polynomial dependence on capacities, rather than logarithmic.

1.2 Related works

Linear systems. For the problem of solving the linear system $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$, George first developed the method of nested dissection in [77], which leveraged the underlying graph structure of $\mathbf{A}$ for the case where it is a grid. This was generalized by the seminal work of Lipton, Rose and Tarjan in [122], to solving systems where $\mathbf{A}$ is any symmetric positive-definite matrix whose underlying graph has good balanced vertex separators. Specifically, their algorithm solves the linear system $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ in $O(m + m^{\alpha\omega})$ time when the associated graph is $O(n^{\alpha})$-separable. When $\alpha < 1$, it outperforms the canonical $O(m^{\omega})$-time algorithm for general linear systems. This was further extended by [9], to apply to non-singular matrices over any field. The Cholesky factorization of $\mathbf{A}$ is a key part of all aforementioned results; it has a long line of study in numerical analysis [53], and is used as the default sparse linear system solver in various languages such as Julia, MATLAB and Python.

Linear programming. The quest for understanding the computational complexity of linear programming has a long and rich history in computer science and mathematics. Since the seminal works of Khachiyan [105] and later Karmarkar [98], who were the first to prove that LPs can be solved in polynomial time, the interior point method and its subsequent variants have become the central methods for efficiently solving linear programs with provable guarantees. This has led to a series of refined and more efficient IPM-based solvers [145, 162, 135, 115, 116, 44, 95], which culminated in the recent breakthrough work of Cohen, Lee, and Song [44] who showed that an LP solver whose running time essentially matches the matrix multiplication cost, up to small low-order terms. In a follow-up work, Brand [29] managed to derandomize their algorithm while retaining the same time complexity.

Algorithms parametrized by treewidth. Arising from structural graph theory, treewidth has become a focus of study in fixed-parameter tractable algorithms in various communities including combinatorics, integer-linear programming, and numerical analysis. The notion of treewidth is closely tied to vertex separators; specifically, low treewidth graphs have small vertex separators, and this structure is amenable to a dynamic programming approach for various problems. Many NP-hard problems are known to have algorithms with runtime depending linearly on the problem size and exponentially on treewidth [22] as the result of dynamic programming. These include Independent Set, Hamiltonian Circuit, Steiner Tree, and Travelling Salesman. Analogously, many problems in P should be solvable in $\widetilde{O}(n \cdot \mathrm{tw}^{O(1)})$ time; however, due to the lack of appropriate tools, only a few such results are currently known. They are extensively studied as part of the class of fixed-parameter tractible problems. In general, dynamic programming style approaches based on the tree decomposition unfortunately almost always lead to an exponential dependence on treewidth, even for polynomial-time solvable problems.

When the problem is linear algebraic, such as solving linear systems and computing rank or determinant, the dynamic programming approaches often leads to runtime polynomial in treewidth.

Recently, [73] showed several problems can be reduced to matrix factorizations efficiently, including computing determinant, computing rank, and finding maximum matching, and this leads to $O(\tau^{O(1)} \cdot n)$ time algorithms where τ is the width of the given tree decomposition of the graph. The only non-linear algebraic $O(\tau^{O(1)} \cdot n)$ time problem we are aware of is Unweighted Maximum Vertex-Flow [73], which makes use of the crucial fact that the vertex separator size is directedly connected to the flow size to achieve a $\widetilde{O}(\tau^2 \cdot n)$ runtime.

A long line of work in the integer-linear programming (ILP) community studies solving ILPs with respect to fixed treedepth, a parameter related but more restrictive than treewidth; indeed, ILPs can be weakly NP-hard even on instances with treewidth at most two. For an ILP with treedepth denoted $\mathrm{td}(\mathbf{A})$, Eisenbrand et al [65] gave a weakly-polynomial ILPs algorithm running in time $O(g(\min\{\mathrm{td}(\mathbf{A}), \mathrm{td}(\mathbf{A}^\top)\}) \cdot \mathrm{poly}(m))$, where g is at least some doubly-exponential function. This is followed-up by [50], which gave a strongly polynomial algorithm running in $2^{O(\mathrm{td} \cdot 2^{\mathrm{td}})} \Delta^{O(2^{\mathrm{td}})} m^{1+o(1)}$ time, where Δ is an upper-bound on the absolute value of an entry of A. [65] also discussed how an algorithm for ILP may be used to solve LP; [28] built on this to give an algorithm solving mixed integer-linear programs in time $f(a, \mathrm{td}(A))\mathrm{poly}(m)$, where a is the largest coefficient of the constraint matrix.

Max flow and min-cost flow on general graphs. Max flow and min-cost flow are well-studied in both structured graphs and general graphs. Tables 1.1 and 1.2 summarize the current best algorithms for different settings.

In what follows, we will focus on surveying only *exact* algorithms for max-flow and min-cost flow on general graphs. For earlier developments on these problems, including fast approximation

Min-cost flow	Time bound	Reference
Strongly polytime	$O(m^2 \log n + mn \log^2 n)$	[138]
Weakly polytime	$\widetilde{O}(m^{1+o(1)} \log^2 M)$	[39]
Unit-capacity	$m^{\frac{4}{3}+o(1)} \log M$	[14]
Planar graph	$\widetilde{O}(n \log^2 M)$	[56]
Unit-capacity planar graph	$O(n^{4/3} \log M)$	[97]
Graph with treewidth τ	$\widetilde{O}(n\tau^2 \log M)$	[58]

Table 1.1: Fastest known exact algorithms for the min-cost flow problem, ordered by the generality of the result. Here, n is the number of vertices, m is the number of edges, and M is the maximum of edge capacity and cost value.

Max-flow	Time bound	Reference
Strongly polytime	$O(mn)$	[139, 106]
Weakly polytime	$\widetilde{O}(m^{\frac{3}{2}-\frac{1}{328}} \log U)$	[75]
Pseudo polytime	$m^{\frac{4}{3}+o(1)} U^{1/3}$	[100]
g-genus graph	$\min(g^{O(g)} n^{3/2}, O(g^8 n \log^2 n \log^2 U))$	[36]
Planar graph plus k edges	$O(k^3 n \log n)$	[88]
Planar graph	$O(n \log n)$	[26]
Undirected planar graph	$O(n \log \log n)$	[93]
st-planar graph	$O(n)$	[87]

Table 1.2: Fastest known exact algorithms for the max flow flow problem, ordered by the generality of the result. Min-cost flow algorithms are omitted in the max flow table. Here, n is the number of vertices, m is the number of edges, and U is the maximum edge capacity.

algorithms, we refer the reader to the following works [106, 4, 41, 149, 103, 140, 151, 21], and the references therein.

An important view, unifying almost all recent max-flow or min-cost flow developments, is interpreting max-flow as the problem of finding one unit of s-t flow that minimizes the ℓ_∞ congestion of the flow vector. Motivated by the near-linear Laplacian solver of Spielman and Teng [153] (which in turn can be used to solve the problem of finding one unit of s-t flow that minimizes the ℓ_2 congestion), and the fact that the gap between ℓ_∞ and ℓ_2 is roughly $O(\sqrt{m})$, Daitch and Spielman [51] showed how to implement the IPM for solving min-cost flows in $\widetilde{O}(m^{3/2})$ time.

Follow-up works initially made progress on the case of unit capacitated graphs, with the work of Madry [125] achieving an $\widetilde{O}(m^{10/7})$ time algorithm for max flow and thus being the first to break the 3/2-exponent barrier in the runtime. The running time was later improved to $O(m^{4/3+o(1)})$ and it was generalized to the min-cost flow problem [15, 100].

For general, polynomially bounded capacities, Brand et al. [31] gave an improved algorithm for dense graphs that runs in $\widetilde{O}(m+n^{3/2})$. In the sparse graph regime, Gao, Liu and Peng [75] were the first to break the 3/2-exponent barrier by giving an $\widetilde{O}(m^{3/2-1/128})$ time algorithm, which was later improved to $\widetilde{O}(m^{3/2-1/58})$ [163]. Very recently, the breakthrough work of Chen et al. [39] shows that the min-cost flow problem can be solved in $\widetilde{O}(m^{1+o(1)})$, which is optimal up to the subpolynomial term.

Max flow and min-cost flow on planar graphs. The study of flows on planar graphs dates back to the celebrated work of Ford and Fulkerson [74] who showed that for the case of s,t-planar graphs[1], there is an $O(n^2)$ time algorithm for max flow. This was subsequently improved to $O(n \log n)$ by Itai and Shiloach [92] and finally to $O(n)$ by Henzinger et al [87], the latter building upon a prior work of Hassin [83].

For general planar graphs, there have been two lines of work focusing on the undirected and the directed version of the problem respectively. In the first setting, Reif [144] (and later Hassin and Johnson [85]) gave an $O(n \log^2 n)$ time algorithm. The state-of-the-art algorithm is due to Italiano et al. [93] and achieves $O(n \log\log n)$ runtime. Weihe [169] gave the first speed-up for directed planar max flow running in $O(n \log n)$ time. However, his algorithm required some assumptions on the connectivity of the input graph. Later on, Borradaile and Klein [25] gave an $O(n \log n)$ algorithm for general planar directed graphs. Generalization of planar graphs, e.g., graphs of bounded genus have also been studied in the context of the

[1]planar graphs where s and t lie on the same face

max flow problem. The work of Chambers et al. [35] showed that these graphs also admit near-linear time max flow algorithms.

Imai and Iwano [90] obtained an $O(n^{1.594} \log M)$ min-cost flow algorithm for graphs that are $O(\sqrt{n})$-recursively separable. For the min-cost flow problem on planar graphs with unit capacities, Karczmarz and Sankowski [97] gave an $O(n^{4/3})$ algorithm. Very recently, Dong et al. [56] showed that the min-cost flow on planar directed graphs with polynomially bounded capacities admits an $\widetilde{O}(n)$ time algorithm, which is optimal up to polylogarithmic factors.

Multicommodity flow on general graphs. It is known that 2-commodity flow is as hard as linear programming [91]. Recently, [55] showed a linear-time reduction from linear programs to *sparse* k-commodity flow instance, indicating that sparse k-commodity flow instances are hard to solve. This has led to renewed interest in solving k-commodity flow in restricted settings, with the authors of [167] making progress on dense graphs. It is known that we can solve multicommodity flow in the high-accuracy regime using linear programming. For a graph with n nodes, m edges, and k commodities, the underlying constraint matrix has km variables and $kn + m$ equality constraints. Thus, using the best-known algorithms for solving linear programs [44, 29], one can achieve a runtime complexity of $\widetilde{O}((km)^{\omega})$ for solving multi-commodity flow. In the special case of dense graphs, Brand and Zhang [167] recently showed an improved algorithm achieving $\widetilde{O}(k^{2.5}\sqrt{m}n^{\omega-1/2})$ runtime.

In the approximate regime, Leighton et al. [121] show that $(1 + \epsilon)$ multi-commodity flow on *undirected graphs* can be solved in $\widetilde{O}(kmn)$, albeit with a rather poor dependency on ϵ. This result led to several follow-up improvements in the low-accuracy regime [76, 72, 124]. Later on, breakthrough works in approximating single commodity max flow in nearly-linear time were also extended to the k-commodity flow problem on undirected graphs [103, 149, 140], culminating in the work of Sherman [150] who achieved an $\widetilde{O}(mk\epsilon^{-1})$ time algorithm for the problem.

Multi-commodity flow on planar graphs. The multi-commodity flow problem on planar graphs was studied in the 1980s, but there has not been much interest in it until most recently. Results in the past focused on finding conditions under which solutions existed [136, 84], or finding simple algorithms in even more restricted settings, with the authors of [127] demonstrating that the problem could be solved in $O(kn + n^2(\log n)^{1/2})$ time if the sources and sinks were all on the outer face of the graph. More recently, [101] studied the all-or-nothing version of planar multi-commodity flow, where flows have to be integral, and demonstrate that an $O(1)$-approximation could be achieved in polynomial time.

1.3 Thesis organization

The focus of this thesis is on solving linear programs of the form (LP).

In Chapter 2, we present the robust interior point method (IPM), which was first introduced by Cohen, Lee, and Song in [117], and subsequently refined by Dong, Lee, Ye in [58]. This is a general method for solving linear and convex programs to high accuracy, on top of which we build our more refined techniques. The robust IPM algorithm iteratively improves the solution, by performing updates of the form

$$\boldsymbol{x}^{(\text{new})} \leftarrow \boldsymbol{x} + \mathbf{W}^{1/2}\mathbf{A}^\top(\mathbf{A}\mathbf{W}\mathbf{A}^\top)^{-1}\mathbf{A}\mathbf{W}^{1/2}\boldsymbol{v}, \tag{1.2}$$

where $\boldsymbol{x}$ is the current solution, $\boldsymbol{v}$ is a Newton step, and $\boldsymbol{w}$ controls how close $\boldsymbol{x}$ can come to its boundaries $\boldsymbol{l}$ and $\boldsymbol{u}$. In classical IPM, $\boldsymbol{w}$ and $\boldsymbol{v}$ are functions of $\boldsymbol{x}$, whereas in the robust IPM, they are functions of some ℓ_∞-approximation $\overline{\boldsymbol{x}}$ of $\boldsymbol{x}$.

In Chapter 3, we lay the technical foundations for incorporating the constraint matrix structure into the robust IPM algorithm. It connects the matrix $\mathbf{A}$ to a graph, characterizes the relevant structures in the graph, and encodes this information using a *separator tree*. Then it introduces the concept of a *tree operator*, which is a linear operator that leverages the structures of a tree. For our purposes, the tree is essentially the separator tree, though we hope the concept finds more general application in the future.

In Chapters 4 and 5, we incorporate the structural ideas developed earlier into the IPM. Chapter 4 writes the expression $\mathbf{W}^{1/2}\mathbf{A}^\top(\mathbf{A}\mathbf{W}\mathbf{A}^\top)^{-1}\mathbf{A}\mathbf{W}^{1/2}$ as tree operators, using the decomposition from Chapter 3. Building on this, Chapter 5 gives the data structure that *implicitly represents* $\boldsymbol{x}$ throughout the IPM. Dynamic updates to $\boldsymbol{x}$ are processed correctly and efficiently, but notably, $\boldsymbol{x}$ is "hidden" in the data structure and only accessible through specific types of queries. We maintain the ℓ_∞-approximation $\overline{\boldsymbol{x}}$ to $\boldsymbol{x}$ throughout the IPM, where the difficulty lies in the limited access to $\boldsymbol{x}$. The solution requires clever data structure design making use of the separator tree.

In Chapter 6, we present our results for linear programs, including separable LPs, bounded-treewidth LPs, and k-commodity flow on planar graphs, which is a special form of separable LPs. In Chapter 7, we present the min-cost flow results on planar and bounded-treewidth graphs. When written as LPs, their constraint matrices are edge-vertex incidence matrices of graphs, which allows us to leverage additional tools in fast Laplacian solvers and approximate Schur complements for further improved runtimes. Here, significant care is required to ensure feasibility in the solution updates, which is no longer guaranteed due to the approximations. We then present the corollaries about general graphs and tree decomposition computations.

Chapter 8 is a serendipitous result on circle packing, which perhaps best embodies the theme of combining convex optimization with combinatorial insights.

Chapter 2

Robust interior point method for general convex sets

Consider the convex optimization problem

$$\begin{aligned} \min \quad & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{s.t.} \quad & \mathbf{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x}_{[i]} \in K_i \qquad \forall i \in [m] \end{aligned} \tag{P}$$

where $\mathbf{A}$ is a $n \times d$ matrix, $\boldsymbol{x}$ is the concatenation of the $\boldsymbol{x}_{[i]}$ blocks, and each $K_i \subseteq \mathbb{R}^{d_i}$ is a convex set with $\sum_{i=1}^m d_i = d$. Let $K \overset{\text{def}}{=} K_1 \times \cdots \times K_m$. Define the following *given* parameters for the problem:

- Inner radius r: There exists some $\boldsymbol{z}$ such that $\mathbf{A}\boldsymbol{z} = \boldsymbol{b}$ and $B(\boldsymbol{z}, r) \subset K$.
- Outer radius R: There exists some $\boldsymbol{z}'$ such that $K \subset B(\boldsymbol{z}', R)$.
- Lipschitz constant L: $\|\boldsymbol{c}\|_2 \le L$.
- Self-concordant barrier function ϕ: For each i, we have a ν_i-self-concordant barrier function ϕ_i on K_i, and weight $w_i \ge 1$. Let $\phi(\boldsymbol{x}) \overset{\text{def}}{=} \sum_{i=1}^m w_i \phi_i(\boldsymbol{x}_{[i]})$ and $\kappa \overset{\text{def}}{=} \sum_{i=1}^m w_i \nu_i$.

This is a very general problem formulation. Linear programs in standard form,

$$\min\ \boldsymbol{c}^\top \boldsymbol{x} \text{ s.t. } \mathbf{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{l} \le \boldsymbol{x} \le \boldsymbol{u},$$

can be written as (P), where each block $\boldsymbol{x}_{[i]}$ simply refers to coordinate i, and K_i is the interval $[\boldsymbol{l}_i, \boldsymbol{u}_i]$. Convex problems of the form

$$\min f(\boldsymbol{x}) \text{ s.t. } \boldsymbol{x} \in K$$

can be transformed via the epigraph trick into

$$\min t \text{ s.t. } (\boldsymbol{x}, t) \in \{K \times \mathbb{R} : f(\boldsymbol{x}) \le t\},$$

where the feasible set is also called the *epigraph* of f and known to be convex.

A class of algorithms for solving (P) is *interior point methods (IPM)*; these are iterative algorithms that follow a *central path* in the interior of K, starting at some known point and ending close to the minimizer of the problem. Fig. 2.1 presents a conceptual illustration.

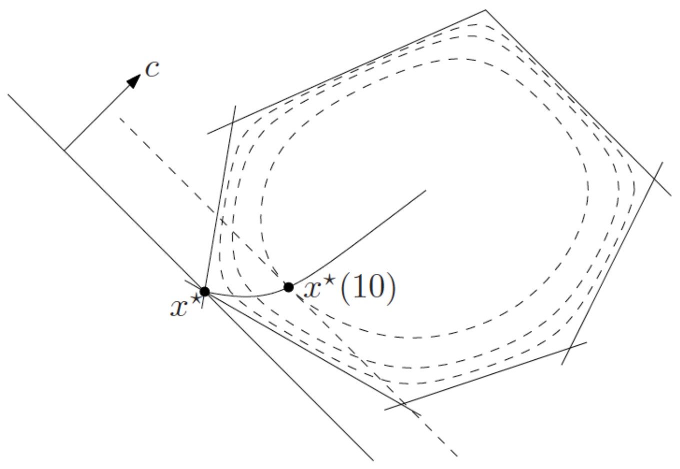


Figure 2.1: A simple illustration of the interior point method. The convex body represents the feasible set; $\boldsymbol{x}^\star$ represents the optimal solution; and the curve represents the parametrized central path. Source: [27, Figure 11.2].

In this chapter, we present a *robust* interior point method, first developed in [117], and later refined in [58]. Compared to [117], [58] introduced approximate t in the algorithm to simplify the main data structure; a new reduction for finding an initial point, allowing the algorithm to output $\boldsymbol{x}$ exactly satisfying $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$; and new potential function $\cosh(\|\cdots\|)$ instead of $\exp(\|\cdots\|)$ to simplify the proofs. We note that it is an interesting open question to extend this result to dynamic weighted barriers such as the Lee-Sidford barrier [115], beyond the case $d_i = 1$. For completeness, we begin with a brief review of standard interior point methods; more details can be found in [27] and [170].

2.1 Background on IPMs

Given the problem (P), we define the primal and dual feasible sets

$$\mathcal{P} \overset{\text{def}}{=} \{\boldsymbol{x} \in K \ : \ \mathbf{A}\boldsymbol{x} = \boldsymbol{b}\}$$
$$\mathcal{D} \overset{\text{def}}{=} \{\boldsymbol{s} \in \mathbb{R}^d_{\geq 0} \ : \ \mathbf{A}^\top \boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c} \text{ for some } \boldsymbol{y}\}.$$

We always assume $\mathcal{P}$ has non-empty interior.

The classical central path for the primal problem is given by

$$\boldsymbol{x}^\star(t) = \operatorname*{argmin}_{\boldsymbol{x} \in \mathcal{P}} \boldsymbol{c}^\top \boldsymbol{x} + t\phi(\boldsymbol{x}).$$

By definition of self-concordant barrier functions, $\phi(\boldsymbol{x}) \to \infty$ as $\boldsymbol{x} \to \partial K$, hence $\boldsymbol{x}(t)$ lies in the interior of $\mathcal{P}$ for any $t > 0$. When $t = 0$, we get back the original (P), so $\boldsymbol{x}^\star(0)$ is an optimal solution. When t is large, $\boldsymbol{x}^\star(t)$ approaches the center of K (with respect to the barrier function).

The barrier method is the most straighforwardly digestable approach and is described in Algorithm 1. We assume a feasible $\boldsymbol{x}^{(\text{init})} \in \text{int}(K)$ is given on input.

Algorithm 1 Barrier method for Eq. (P) [27, Algorithm 11.1].

```
1: procedure PrimalPathFollowing($\boldsymbol{x}^{(\text{init})}, t_{\text{start}}, t_{\text{end}}$)
2:     $(\boldsymbol{x}, t) \leftarrow (\boldsymbol{x}^{(\text{init})}, t_{\text{start}})$
3:     while $t \geq t_{\text{end}}$ do
4:         Compute $\boldsymbol{x}^\star(t)$ (using an iterative method) with $\boldsymbol{x}$ as the starting point
5:         $\boldsymbol{x} \leftarrow \boldsymbol{x}^\star(t)$
6:         $t \leftarrow (1-h)t$                    ▷ $h \in (0,1)$ is a fixed multiplicative factor
7:     end while
8:     return $\boldsymbol{x}$
9: end procedure
```

Primal-dual interior point methods further incorporate the dual variables into every iteration of the path-following algorithm. KKT conditions stipulate that $\boldsymbol{x}^\star(t)$ satisfies

$$\begin{aligned} \frac{\boldsymbol{s}^\star(t)}{t} + \nabla\phi(\boldsymbol{x}^\star(t)) &= \boldsymbol{0} \\ \mathbf{A}\boldsymbol{x}^\star(t) &= \boldsymbol{b}, \\ \mathbf{A}^\top \boldsymbol{y} + \boldsymbol{s}^\star(t) &= \boldsymbol{c}, \end{aligned} \tag{2.1}$$

for unique dual variable $\boldsymbol{s}^\star(t)$. Note that the first non-linear equation has an unique solution for any vector $\boldsymbol{\mu}$ on the right hand side; here we have $\boldsymbol{\mu} = \boldsymbol{0}$. In particular, the solution $\boldsymbol{x}$ is the solution of the optimization problem $\min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b}} \boldsymbol{c}^\top \boldsymbol{x} + t\sum_{i=1}^m w_i\phi_i(\boldsymbol{x}_{[i]}) - t\boldsymbol{\mu}^\top \boldsymbol{x}$. Hence, we can move $\boldsymbol{\mu}$ arbitrarily while maintaining (2.2) by moving $\boldsymbol{x}$ and $\boldsymbol{s}$.

We can define the central path via the modified KKT conditions

$$\begin{aligned} \frac{\boldsymbol{s}^\star(t)}{t} + \nabla\phi(\boldsymbol{x}^\star(t)) &= \boldsymbol{\mu} \\ \mathbf{A}\boldsymbol{x}^\star(t) &= \boldsymbol{b}, \\ \mathbf{A}^\top \boldsymbol{y} + \boldsymbol{s}^\star(t) &= \boldsymbol{c}, \end{aligned} \tag{2.2}$$

where $\boldsymbol{\mu}$ is close to $\mathbf{0}$ but does not need to be constant across iterations. Now, Algorithm 2 describes one attempt at a basic primal-dual interior point method. Similar to the primal-only barrier method, it assumes feasible $\boldsymbol{x}^{(\text{init})}, \boldsymbol{s}^{(\text{init})}$ on input.

Algorithm 2 Basic primal-dual interior point method for Eq. (P).

```
1: procedure PrimalDualPathFollowing($\boldsymbol{x}^{(\text{init})}, \boldsymbol{s}^{(\text{init})}, t_{\text{start}}, t_{\text{end}}$)
2:     $(\boldsymbol{x}, \boldsymbol{s}, t) \leftarrow (\boldsymbol{x}^{(\text{init})}, \boldsymbol{s}^{(\text{init})}, t_{\text{start}})$
3:     while $t \geq t_{\text{end}}$ do
4:         Solve Eq. (2.2) to get $\boldsymbol{x}^\star(t), \boldsymbol{s}^\star(t)$ with $\boldsymbol{x}, \boldsymbol{s}$ as the starting point
5:         $(\boldsymbol{x}, \boldsymbol{s}) \leftarrow (\boldsymbol{x}^\star(t), \boldsymbol{s}^\star(t))$
6:         $t \leftarrow (1-h)t$                    ▷ $h \in (0,1)$ is a fixed multiplicative factor
7:     end while
8:     return $\boldsymbol{x}$
9: end procedure
```

We observe that the central path's role is to iteratively guide the solution to the optimal solution. In particular, there is no real need for the current solution to land on the central path. Indeed, at one iteration, rather than solving for $\boldsymbol{x}^\star(t), \boldsymbol{s}^\star(t)$, it suffices to take a Newton step $\boldsymbol{\delta_x}, \boldsymbol{\delta_s}$ on (2.2). The first KKT condition is (we simply write $\boldsymbol{x}$ to denote $\boldsymbol{x}^\star(t)$ for readability):

$$\frac{\boldsymbol{s} + \boldsymbol{\delta_s}}{t} + \nabla\phi(\boldsymbol{x} + \boldsymbol{\delta_x}) = \boldsymbol{\mu} + \boldsymbol{\delta_\mu},$$

Approximating $\nabla\phi(\boldsymbol{x} + \boldsymbol{\delta_x}) \approx \nabla\phi(\boldsymbol{x}) + \nabla^2\phi(\boldsymbol{x})\boldsymbol{\delta_x}$, we conclude the step is defined by

$$\frac{1}{t}\boldsymbol{\delta_s} + \nabla^2\phi(\boldsymbol{x})\boldsymbol{\delta_x} = \boldsymbol{\delta_\mu}.$$

This gives us the next version of the primal-dual IPM in Algorithm 3. With it, we are ready to introduce the robust IPM.

2.2 Path following in the robust IPM

Similar to classical primal-dual interior point methods, our algorithm decreases the duality measure t multiplicatively by a factor of $(1-h)$ at every iteration for some fixed h, and then takes a Newton-like step on (2.2) to improve the centrality of the current solution $(\boldsymbol{x}, \boldsymbol{s})$, in other words, move it closer to the point on the central path corresponding to t. *Robust* refers to robustness of the choice in the Newton step, which we discuss in subsequent sections.

Algorithm 3 Primal-dual interior point method for Eq. (P), similar to [170, Page 8].

```
1: procedure PrimalDualPathFollowing2($\boldsymbol{x}^{(\text{init})}, \boldsymbol{s}^{(\text{init})}, t_{\text{start}}, t_{\text{end}}$)
2:     $(\boldsymbol{x}, \boldsymbol{s}, t) \leftarrow (\boldsymbol{x}^{(\text{init})}, \boldsymbol{s}^{(\text{init})}, t_{\text{start}})$
3:     while $t \geq t_{\text{end}}$ do
4:         Take a Newton-like step on Eq. (2.2). That is, find $\boldsymbol{\delta_x}, \boldsymbol{\delta_s}$ satisfying
               $$\begin{aligned} \frac{1}{t}\boldsymbol{\delta_s} + \nabla^2\phi(\boldsymbol{x})\boldsymbol{\delta_x} &= \boldsymbol{\delta_\mu} \\ \mathbf{A}\boldsymbol{\delta_x} &= \mathbf{0} \\ \mathbf{A}^\top\boldsymbol{\delta_y} + \boldsymbol{\delta_s} &= \mathbf{0}, \end{aligned}$$
5:         and update $(\boldsymbol{x}, \boldsymbol{s}) \leftarrow (\boldsymbol{x}, \boldsymbol{s}) + \alpha(\boldsymbol{\delta_x}, \boldsymbol{\delta_s})$
           with step size $\alpha$ chosen so that the new solution is strictly feasible.
6:         $t \leftarrow (1-h)t$                    ▷ $h \in (0,1)$ is a fixed multiplicative factor
7:     end while
8:     return $\boldsymbol{x}$
9: end procedure
```

Let us first introduce some notation.

Definition 2.1 (Induced norms)**.** Let $\boldsymbol{x} \in K$. For any i and any $\boldsymbol{v} \in \mathbb{R}^{n_i}$, we define

$$\|\boldsymbol{v}\|_{\boldsymbol{x}_{[i]}} \stackrel{\text{def}}{=} \|\boldsymbol{v}\|_{\nabla^2\phi_i(\boldsymbol{x}_{[i]})}, \quad \text{and} \quad \|\boldsymbol{v}\|_{\boldsymbol{x}_{[i]}}^* \stackrel{\text{def}}{=} \|\boldsymbol{v}\|_{(\nabla^2\phi_i(\boldsymbol{x}_{[i]}))^{-1}}.$$

For the whole domain $K \stackrel{\text{def}}{=} \prod_{i=1}^m K_i$ and any $\boldsymbol{v} \in \mathbb{R}^n$, we define

$$\begin{aligned} \|\boldsymbol{v}\|_{\boldsymbol{x}} &\stackrel{\text{def}}{=} \|\boldsymbol{v}\|_{\nabla^2\phi(\boldsymbol{x})} = \sqrt{\sum_i w_i \left(\left\|\boldsymbol{v}_{[i]}\right\|_{\boldsymbol{x}_{[i]}}\right)^2}, \text{ and} \\ \|\boldsymbol{v}\|_{\boldsymbol{x}}^* &\stackrel{\text{def}}{=} \|\boldsymbol{v}\|_{(\nabla^2\phi(\boldsymbol{x}))^{-1}} = \sqrt{\sum_i w_i^{-1} \left(\left\|\boldsymbol{v}_{[i]}\right\|_{\boldsymbol{x}_{[i]}}^*\right)^2}. \end{aligned}$$

To define centrality, we use the following potential function:

Definition 2.2 (Potential function)**.** For each $i \in [m]$, we define the error of $(\boldsymbol{x}, \boldsymbol{s})$ at time t on block i by

$$\boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}, \boldsymbol{s}) \stackrel{\text{def}}{=} \frac{\boldsymbol{s}_{[i]}}{t} + w_i \nabla\phi_i(\boldsymbol{x}_{[i]}).$$

Note that any point on the central path attains a value of $\mathbf{0}$. We define the centrality measure via a local norm:

$$\boldsymbol{\gamma}_i^t(\boldsymbol{x}, \boldsymbol{s}) \overset{\text{def}}{=} \left\|\boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}, \boldsymbol{s})\right\|_{\boldsymbol{x}_{[i]}}^* = \left\|(\nabla^2\phi_i(\boldsymbol{x}_{[i]}))^{-1/2}\boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}, \boldsymbol{s})\right\|_2,$$

The normalization term $(\nabla^2\phi_i(\boldsymbol{x}_{[i]}))^{1/2}$ makes the centrality measure scale-invariant in K_i.

For any fixed scaling factor $\lambda > 0$, we define the soft-max function $\Psi : \mathbb{R}^m \mapsto \mathbb{R}_{\geq 0}$ by

$$\Psi_\lambda(\boldsymbol{\gamma}) \overset{\text{def}}{=} \sum_{i=1}^m \cosh(\lambda \frac{\gamma_i}{w_i}).$$

Finally, the potential function is the soft-max of the centrality measure

$$\Phi^t(\boldsymbol{x}, \boldsymbol{s}) \overset{\text{def}}{=} \Psi_\lambda(\boldsymbol{\gamma}^t(\boldsymbol{x}, \boldsymbol{s})).$$

Observe that any point on the central path has potential m.

Since the primal-dual variables $\boldsymbol{x}, \boldsymbol{s}$ comes as a pair, we often omit the dual variables in the function arguments for readability.

The following lemma showing that when the centrality measure is sufficiently small, a feasible solution is indeed close to optimal.

Lemma 2.3 ([117, Lemma D.3])**.** *Given the convex program* (P)*, suppose we have solution* $\boldsymbol{x}, \boldsymbol{s}$ *and* t *satisfying*

- $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$*,*
- $\boldsymbol{x} \in \operatorname{int}(K)$*,*
- $\mathbf{A}^\top \boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c}$ *for some* $\boldsymbol{y}$*, and*
- $\boldsymbol{\gamma}_i^t(\boldsymbol{x}, \boldsymbol{s}) \overset{\text{def}}{=} \left\|\frac{\boldsymbol{s}_{[i]}}{t} + w_i \nabla\phi_i(\boldsymbol{x}_{[i]})\right\|_{\boldsymbol{x}_{[i]}}^* \leq w_i$ *for each* $i \in [m]$*.*

Then,

$$\boldsymbol{c}^\top \boldsymbol{x} \leq \min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b},\, \boldsymbol{x}\in K} \boldsymbol{c}^\top \boldsymbol{x} + 3t\kappa.$$

Proof. Let $\boldsymbol{x}^\star \overset{\text{def}}{=} \operatorname{argmin}_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b},\, \boldsymbol{x}\in K} \boldsymbol{c}^\top \boldsymbol{x}$, and let $\boldsymbol{x}^{(u)} \overset{\text{def}}{=} (1-u)\boldsymbol{x} + u\boldsymbol{x}^\star$ for some fixed $\boldsymbol{u}$ to be chosen. Recall $\phi \overset{\text{def}}{=} \sum w_i\phi_i$, so by Lemma 2.29, we know

$$\left\langle \nabla\phi(\boldsymbol{x}^{(u)}), \boldsymbol{x}^\star - \boldsymbol{x}^{(u)} \right\rangle \leq \sum_i w_i\nu_i \overset{\text{def}}{=} \kappa.$$

After rearranging, we get

$$\begin{aligned}
\frac{\kappa u}{1-u} &\geq \left\langle \nabla\phi(\boldsymbol{x}^{(u)}), \boldsymbol{x}^{(u)} - \boldsymbol{x} \right\rangle \\
&= \left\langle \nabla\phi(\boldsymbol{x}^{(u)}) - \nabla\phi(\boldsymbol{x}), \boldsymbol{x}^{(u)} - \boldsymbol{x} \right\rangle + \left\langle \nabla\phi(\boldsymbol{x}), \boldsymbol{x}^{(u)} - \boldsymbol{x} \right\rangle \\
&= \left\langle \nabla\phi(\boldsymbol{x}^{(u)}) - \nabla\phi(\boldsymbol{x}), \boldsymbol{x}^{(u)} - \boldsymbol{x} \right\rangle + \left\langle \boldsymbol{\mu}^t(\boldsymbol{x}, \boldsymbol{s}), \boldsymbol{x}^{(u)} - \boldsymbol{x} \right\rangle - \frac{1}{t}\left\langle \boldsymbol{s}, \boldsymbol{x}^{(u)} - \boldsymbol{x} \right\rangle \\
&\qquad \text{(By definition on the second term)}
\end{aligned}$$

For the third term, note that $\boldsymbol{s} = \mathbf{A}^\top \boldsymbol{y} - \boldsymbol{c}$ and $\mathbf{A}\boldsymbol{x}^{(u)} = \mathbf{A}\boldsymbol{x}$. For the first two terms, note that ϕ and $\boldsymbol{\mu}$ are both defined block-wise. Breaking it up into the m blocks, and applying Lemma 2.29 to the first term and Cauchy-Schwarz to the second term, we get

$$\geq \sum_{i=1}^{m} \left(\frac{w_i \left\| \boldsymbol{x}_{[i]}^{(u)} - \boldsymbol{x}_{[i]} \right\|_{\boldsymbol{x}_{[i]}}^2}{1 + \left\| \boldsymbol{x}_{[i]}^{(u)} - \boldsymbol{x}_{[i]} \right\|_{\boldsymbol{x}_{[i]}}} - \left\| \boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}, \boldsymbol{s}) \right\|_{\boldsymbol{x}_{[i]}}^* \cdot \left\| \boldsymbol{x}_{[i]}^{(u)} - \boldsymbol{x}_{[i]} \right\|_{\boldsymbol{x}_{[i]}} \right) - \frac{1}{t}\left\langle \boldsymbol{c}, \boldsymbol{x}^{(u)} - \boldsymbol{x} \right\rangle.$$

Next, we replace $\boldsymbol{x}^{(u)} - \boldsymbol{x}$ by $u(\boldsymbol{x}^\star - \boldsymbol{x})$, and apply the assumption $\boldsymbol{\gamma}_i^t(\boldsymbol{x}, \boldsymbol{s}) = \left\| \boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}, \boldsymbol{s}) \right\|_{\boldsymbol{x}_{[i]}}^* \leq w_i$, to get

$$\geq \sum_{i=1}^{m} \left(\frac{w_i u^2 \left\| \boldsymbol{x}_{[i]}^\star - \boldsymbol{x}_{[i]} \right\|_{\boldsymbol{x}_{[i]}}^2}{1 + u\left\| \boldsymbol{x}_{[i]}^\star - \boldsymbol{x}_{[i]} \right\|_{\boldsymbol{x}_{[i]}}} - w_i u \cdot \left\| \boldsymbol{x}_{[i]}^\star - \boldsymbol{x}_{[i]} \right\|_{\boldsymbol{x}_{[i]}} \right) - \frac{u}{t}\left\langle \boldsymbol{c}, \boldsymbol{x}^\star - \boldsymbol{x} \right\rangle.$$

Rearranging again, we get

$$\left\langle \boldsymbol{c}, \boldsymbol{x} - \boldsymbol{x}^\star \right\rangle \leq \frac{t\kappa}{1-u} + t\sum_{i=1}^{m} w_i \left(\frac{\left\| \boldsymbol{x}_{[i]}^\star - \boldsymbol{x}_{[i]} \right\|_{\boldsymbol{x}_{[i]}}}{1 + u\left\| \boldsymbol{x}_{[i]}^\star - \boldsymbol{x}_{[i]} \right\|_{\boldsymbol{x}_{[i]}}} \right).$$

Setting $u = \frac{1}{2}$, and recalling self-concordance is always greater than 1, we conclude

$$\begin{aligned}
&\leq 2t\kappa + t\sum_{i=1}^{m} w_i \nu_i \\
&\leq 3t\kappa.
\end{aligned}$$

□

Algorithm 4 Path following in the robust interior point method

Definition of parameters:

$$\lambda \stackrel{\text{def}}{=} 64\log(256m\sum_{i=1}^{m} w_i), \quad \overline{\varepsilon} \stackrel{\text{def}}{=} \frac{1}{1440\lambda}, \quad \alpha \stackrel{\text{def}}{=} \frac{\overline{\varepsilon}}{2},$$
$$h \stackrel{\text{def}}{=} \frac{\alpha}{64\sqrt{\sum_{i=1}^{m} w_i\nu_i}}, \quad \varepsilon_t \stackrel{\text{def}}{=} \frac{\overline{\varepsilon}}{4}\cdot\min_i\left(\frac{1}{w_i+\nu_i}\right)$$

```
1: procedure PathFollowingRobust($\mathbf{A}, \phi, \boldsymbol{w}, \boldsymbol{x}^{(\text{init})}, \boldsymbol{s}^{(\text{init})}, t_{\text{start}}, t_{\text{end}}$)
2:     $(\boldsymbol{x}, \boldsymbol{s}, t) \leftarrow (\boldsymbol{x}^{(\text{init})}, \boldsymbol{s}^{(\text{init})}, t_{\text{start}})$
3:     $(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}, \overline{t}) \leftarrow (\boldsymbol{x}, \boldsymbol{s}, t)$
4:     while $t \geq t_{\text{end}}$ do
5:         $\boldsymbol{\delta}_{\boldsymbol{\mu},[i]} \leftarrow -\alpha \cdot \boldsymbol{k}_i^{\overline{t}}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}) \cdot \boldsymbol{\mu}_{[i]}^{\overline{t}}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}})$ for all $i \in [m]$        ▷ defined in Section 2.2.2
6:         find $\boldsymbol{\delta}_{\boldsymbol{x}}$ and $\boldsymbol{\delta}_{\boldsymbol{s}}$ such that $\mathbf{A}\boldsymbol{\delta}_{\boldsymbol{x}} = \mathbf{0}$, $\boldsymbol{\delta}_{\boldsymbol{s}} \in \text{Range}(\mathbf{A}^\top)$ and
               $\left\|\boldsymbol{\delta}_{\boldsymbol{x}} - \mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2}(\mathbf{I} - \mathbf{P}_{\overline{\boldsymbol{x}}})\mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2}\boldsymbol{\delta}_{\boldsymbol{\mu}}\right\|_{\overline{\boldsymbol{x}}} \leq \overline{\varepsilon}\alpha,$
               $\left\|\boldsymbol{\delta}_{\boldsymbol{s}} - \overline{t}\mathbf{H}_{\overline{\boldsymbol{x}}}^{1/2}\mathbf{P}_{\overline{\boldsymbol{x}}}\mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2}\boldsymbol{\delta}_{\boldsymbol{\mu}}\right\|_{\overline{\boldsymbol{x}}}^{*} \leq \overline{\varepsilon}\alpha\overline{t}$
7:         $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{\delta}_{\boldsymbol{x}}$, $\boldsymbol{s} \leftarrow \boldsymbol{s} + \boldsymbol{\delta}_{\boldsymbol{s}}$
8:         update $\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}$ to satisfy $\overline{\boldsymbol{x}} \in \text{int}(K), \overline{\boldsymbol{s}} > \mathbf{0}$, and for each $i \in [m]$,
               $\|\overline{\boldsymbol{x}}_{[i]} - \boldsymbol{x}_{[i]}\|_{\overline{\boldsymbol{x}}_{[i]}} \leq \overline{\varepsilon},$
               $\|\overline{\boldsymbol{s}}_{[i]} - \boldsymbol{s}_{[i]}\|_{\overline{\boldsymbol{x}}_{[i]}}^{*} \leq \overline{t}\overline{\varepsilon}w_i$
9:         $t \leftarrow (1-h)\cdot t$
10:        update $\overline{t}$ to satisfy $|\overline{t} - t| \leq \varepsilon_t\overline{t}$
11:    end while
12:    return $(\boldsymbol{x}, \boldsymbol{s})$
13: end procedure
```

The next theorem summarizes the properties of our path-following algorithm:

Theorem 2.4. *Suppose we are given the convex program* (P)*, initial feasible primal-dual solutions* $\boldsymbol{x}^{(\text{init})}, \boldsymbol{s}^{(\text{init})}$ *satisfying* $\mathbf{A}\boldsymbol{x}^{(\text{init})} = \boldsymbol{b}$*,* $\boldsymbol{x}^{(\text{init})} \in K$*,* $\mathbf{A}^\top\boldsymbol{y} + \boldsymbol{s}^{(\text{init})} = \boldsymbol{c}$ *for some* $\boldsymbol{y}$*, and duality measure* t_{start}*. Then, at the end of every iteration of* PathFollowingRobust *(Algorithm 4),* $\boldsymbol{x}, \boldsymbol{s}, t$ *satisfy*

- $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$,
- $\boldsymbol{x} \in \mathrm{int}(K)$,
- $\mathbf{A}^\top \boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c}$ *for some* $\boldsymbol{y}$, *and*
- $\Phi^t(\boldsymbol{x}, \boldsymbol{s}) \leq \max\left\{\Phi^{t_{\text{start}}}(\boldsymbol{x}^{(\text{init})}, \boldsymbol{s}^{(\text{init})}), \cosh(\frac{\lambda}{64})\right\}$.

Proof. At every iteration, the algorithm guarantees $\mathbf{A}\boldsymbol{\delta}_{\boldsymbol{x}} = \mathbf{0}$ and $\boldsymbol{\delta}_{\boldsymbol{s}} \in \mathrm{Range}(\mathbf{A}^\top)$, and $\boldsymbol{x}, \boldsymbol{s}$ are updated to $\boldsymbol{x} + \boldsymbol{\delta}_{\boldsymbol{x}}, \boldsymbol{s} + \boldsymbol{\delta}_{\boldsymbol{s}}$ respectively. Hence, at the end of the iteration, the linear constraints are satisfied.

Theorem 2.14, proven shortly, shows that given the initial assumption, the potential is bounded after each step as required. The fact $\boldsymbol{x}$ remains in the interior of K is implicit through the choice of step size and in the proof of bounded potential. □

The next three sections are dedicated to the details of this proof. In Section 2.2.1, we discuss gradient descent on Φ that will motivate the definition of our Newton step. In Section 2.2.2, we present the Newton step taken in our algorithm. In Section 2.2.3, we formally prove that the potential is indeed bounded.

2.2.1 Gradient descent on Ψ_λ

Since our goal is to bound $\Phi(\boldsymbol{x}, \boldsymbol{s}) \stackrel{\text{def}}{=} \Psi_\lambda(\boldsymbol{\gamma}^t(\boldsymbol{x}, \boldsymbol{s}))$, we first discuss how to decrease $\Psi_\lambda(\boldsymbol{\gamma})$, if the input vector $\boldsymbol{\gamma}$ could be directly controlled. This section will motivate our choices for our robust path-following algorithm.

Suppose we can make step $\boldsymbol{\gamma} \leftarrow \boldsymbol{\gamma} + \boldsymbol{\delta}$ with the step size constrained by $\sqrt{\sum_i \boldsymbol{\delta}_i^2 / w_i} \leq \alpha$ for some α. Then, a natural choice is the steepest descent direction:

$$\boldsymbol{\delta}^\star \stackrel{\text{def}}{=} \operatorname*{argmin}_{\sqrt{\sum_i \boldsymbol{\delta}_i^2/w_i} \leq \alpha} \left\langle \nabla \Psi_\lambda(\boldsymbol{\gamma}), \boldsymbol{\delta} \right\rangle .$$

Since $\nabla_{\boldsymbol{\gamma}_i} \Psi_\lambda(\boldsymbol{\gamma}) = \frac{\lambda}{w_i} \sinh(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i)$, we have

$$\boldsymbol{\delta}_i^\star = \frac{-\alpha \cdot \sinh(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i)}{\sqrt{\sum_j w_j^{-1} \sinh^2(\frac{\lambda}{w_j}\boldsymbol{\gamma}_j)}}.$$

The following lemma shows that the direction $\boldsymbol{\delta}^\star$ indeed decreases Ψ_λ. Furthermore, this step is robust under ℓ_∞-perturbation of $\boldsymbol{\gamma}$ and ℓ_2-perturbation of $\boldsymbol{\delta}^\star$. To avoid the extra difficulties arising from 0 divided by 0, we replace the sinh by cosh in the denominator.

Lemma 2.5. *Fix any $\boldsymbol{\gamma} \in \mathbb{R}^m$ and step size $\alpha \in [0, \frac{1}{8\lambda}]$. Given any $\overline{\boldsymbol{\gamma}} \in \mathbb{R}^m$ with $\|\boldsymbol{\gamma} - \overline{\boldsymbol{\gamma}}\|_\infty \leq \frac{w_i}{8\lambda}$, and step $\boldsymbol{\delta}$ satisfying*

$$\boldsymbol{\delta}_i = \frac{-\alpha \cdot \sinh(\frac{\lambda}{w_i}\overline{\boldsymbol{\gamma}}_i)}{\sqrt{\sum_j w_j^{-1}\cosh^2(\frac{\lambda}{w_j}\overline{\boldsymbol{\gamma}}_j)}} + \boldsymbol{\varepsilon}_i \tag{2.3}$$

with $\sqrt{\sum_i \boldsymbol{\varepsilon}_i^2/w_i} \leq \frac{\alpha}{8}$, we have

$$\Psi_\lambda(\boldsymbol{\gamma} + \boldsymbol{\delta}) \leq \Psi_\lambda(\boldsymbol{\gamma}) - \frac{\alpha\lambda}{2}\sqrt{\sum_i w_i^{-1}\cosh^2(\lambda\frac{\boldsymbol{\gamma}_i}{w_i})} + \alpha\lambda\sqrt{\sum_i w_i^{-1}}.$$

Proof. By Taylor expansion, we have

$$\Psi_\lambda(\boldsymbol{\gamma} + \boldsymbol{\delta}) = \Psi_\lambda(\boldsymbol{\gamma}) + \langle \nabla\Psi_\lambda(\boldsymbol{\gamma}), \boldsymbol{\delta}\rangle + \frac{1}{2}\boldsymbol{\delta}^\top \nabla^2\Psi_\lambda(\widetilde{\boldsymbol{\gamma}})\boldsymbol{\delta} \tag{2.4}$$

where $\widetilde{\boldsymbol{\gamma}} = \boldsymbol{\gamma} + t\boldsymbol{\delta}$ for some $t \in [0, 1]$.

For the first order term in (2.4), note we have

$$\langle \nabla\Psi_\lambda(\boldsymbol{\gamma}), \boldsymbol{\delta} - \boldsymbol{\varepsilon}\rangle = -\alpha\lambda\frac{\sum_i w_i^{-1}\sinh(\frac{\lambda}{w_i}\overline{\boldsymbol{\gamma}}_i)\sinh(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i)}{\sqrt{\sum_j w_j^{-1}\cosh^2(\frac{\lambda}{w_j}\overline{\boldsymbol{\gamma}}_j)}}.$$

Using Lemma 2.37 and the assumption $|\boldsymbol{\gamma}_i - \overline{\boldsymbol{\gamma}}_i| < \frac{w_i}{8\lambda}$, we have

$$\sinh(\frac{\lambda}{w_i}\overline{\boldsymbol{\gamma}}_i)\sinh(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i) \geq \frac{6}{7}\sinh^2(\frac{\lambda}{w_i}\overline{\boldsymbol{\gamma}}_i) - \frac{1}{7}\left|\sinh(\frac{\lambda}{w_i}\overline{\boldsymbol{\gamma}}_i)\right|.$$

Combining the above, we get

$$\begin{aligned}
&\langle \nabla\Psi_\lambda(\boldsymbol{\gamma}), \boldsymbol{\delta} - \boldsymbol{\varepsilon}\rangle \\
\leq& -\frac{6}{7}\alpha\lambda\frac{\sum_i w_i^{-1}\sinh^2(\frac{\lambda}{w_i}\overline{\boldsymbol{\gamma}}_i)}{\sqrt{\sum_j w_j^{-1}\cosh^2(\frac{\lambda}{w_j}\overline{\boldsymbol{\gamma}}_j)}} + \frac{1}{7}\alpha\lambda\frac{\sum_i w_i^{-1}\left|\sinh(\frac{\lambda}{w_i}\overline{\boldsymbol{\gamma}}_i)\right|}{\sqrt{\sum_j w_j^{-1}\cosh^2(\frac{\lambda}{w_j}\overline{\boldsymbol{\gamma}}_j)}} \\
\leq& -\frac{6}{7}\alpha\lambda\frac{\sum_i w_i^{-1}\cosh^2(\frac{\lambda}{w_i}\overline{\boldsymbol{\gamma}}_i)}{\sqrt{\sum_j w_j^{-1}\cosh^2(\frac{\lambda}{w_j}\overline{\boldsymbol{\gamma}}_j)}} + \frac{6}{7}\alpha\lambda\frac{\sum_i w_i^{-1}}{\sqrt{\sum_j w_j^{-1}\cosh^2(\frac{\lambda}{w_j}\overline{\boldsymbol{\gamma}}_j)}} + \frac{1}{7}\alpha\lambda\frac{\sum_i w_i^{-1}\left|\sinh(\frac{\lambda}{w_i}\overline{\boldsymbol{\gamma}}_i)\right|}{\sqrt{\sum_j w_j^{-1}\sinh^2(\frac{\lambda}{w_j}\overline{\boldsymbol{\gamma}}_j)}} \\
\leq& -\frac{6}{7}\alpha\lambda\sqrt{\sum_i w_i^{-1}\cosh^2(\frac{\lambda}{w_i}\overline{\boldsymbol{\gamma}}_i)} + \alpha\lambda\sqrt{\sum_i w_i^{-1}}.
\end{aligned} \tag{2.5}$$

Using Lemma 2.37 and the assumption $|\boldsymbol{\gamma}_i - \overline{\boldsymbol{\gamma}}_i| < \frac{w_i}{8\lambda}$ again, we have

$$\sqrt{\sum_i w_i^{-1}\cosh^2(\frac{\lambda}{w_i}\overline{\boldsymbol{\gamma}}_i)} \geq \frac{6}{7}\sqrt{\sum_i w_i^{-1}\cosh^2(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i)}.$$

Finally, we substitute to get

$$(2.5) \leq -\frac{36}{49}\alpha\lambda\sqrt{\sum_i w_i^{-1}\cosh^2(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i)} + \alpha\lambda\sqrt{\sum_i w_i^{-1}}.$$

To bound the first order term in (2.4) containing error $\boldsymbol{\varepsilon}$, we have

$$\begin{aligned}\langle \nabla\Psi_\lambda(\boldsymbol{\gamma}), \boldsymbol{\varepsilon}\rangle &= \sum_i \frac{\lambda}{w_i}\sinh(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i)\boldsymbol{\varepsilon}_i \\ &\leq \lambda \cdot \sqrt{\sum_i w_i^{-1}\sinh^2(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i)}\sqrt{\sum_i w_i^{-1}\boldsymbol{\varepsilon}_r^2} \\ &\leq \frac{1}{8}\alpha\lambda\sqrt{\sum_i w_i^{-1}\cosh^2(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i)}.\end{aligned}$$

For the second order term in (2.4), first note that

$$\sqrt{\sum_i \boldsymbol{\delta}_i^2/w_i} \leq \sqrt{\sum_i \left(\frac{\alpha\cdot\sinh(\frac{\lambda}{w_i}\overline{\boldsymbol{\gamma}}_i)}{\sqrt{\sum_j w_j^{-1}\cosh^2(\frac{\lambda}{w_j}\overline{\boldsymbol{\gamma}}_j)}}\right)^2/w_i} + \sqrt{\sum_i \boldsymbol{\varepsilon}_i^2/w_i} \leq \alpha + \frac{\alpha}{8}.$$

In particular, this shows that $|\boldsymbol{\delta}_i| \leq \frac{9\alpha}{8}\sqrt{w_i} \leq \frac{9\alpha}{8}w_i$. Then, we have

$$\begin{aligned}\boldsymbol{\delta}^\top\nabla^2\Psi_\lambda(\widetilde{\boldsymbol{\gamma}})\boldsymbol{\delta} &= \lambda^2\sum_i \frac{\boldsymbol{\delta}_i^2}{w_i^2}\cosh(\lambda\frac{\widetilde{\boldsymbol{\gamma}}_i}{w_i}) \\ &\leq \frac{9\alpha}{8}\lambda^2\sum_i\frac{|\boldsymbol{\delta}_i|}{w_i}\cosh(\lambda\frac{\widetilde{\boldsymbol{\gamma}}_i}{w_i}) \\ &\leq \frac{9\alpha}{8}\lambda^2\sqrt{\sum_i \boldsymbol{\delta}_i^2/w_i}\cdot\sqrt{\sum_i w_i^{-1}\cosh^2(\lambda\frac{\widetilde{\boldsymbol{\gamma}}_i}{w_i})} \\ &\leq (\frac{9\alpha}{8})^2\lambda^2\left(\sqrt{\sum_i w_i^{-1}\cosh^2(\lambda\frac{\widetilde{\boldsymbol{\gamma}}_i}{w_i})}\right) \\ &\leq (\frac{9\alpha}{8})^2\lambda^2\left(\frac{8}{7}\sqrt{\sum_i w_i^{-1}\cosh^2(\lambda\frac{\boldsymbol{\gamma}_i}{w_i})}\right),\end{aligned}$$

where the last inequality follows from Lemma 2.37.

Substituting the bounds on each of the term in (2.4) gives

$$\begin{aligned}\Psi_\lambda(\boldsymbol{\gamma}+\boldsymbol{\delta}) &= \Psi_\lambda(\boldsymbol{\gamma}) + \langle \nabla\Psi_\lambda(\boldsymbol{\gamma}), \boldsymbol{\delta}\rangle + \frac{1}{2}\boldsymbol{\delta}^\top \nabla^2_\lambda(\widetilde{\gamma})\boldsymbol{\delta} \\ &\le \Psi_\lambda(\boldsymbol{\gamma}) - \frac{36}{49}\alpha\lambda\sqrt{\sum_i w_i^{-1}\cosh^2(\lambda\frac{\gamma_i}{w_i})} + \alpha\lambda\sqrt{\sum_i w_i^{-1}} \\ &\quad + (\frac{1}{8}\alpha\lambda + \frac{8}{7}(\frac{9\alpha}{8})^2\lambda^2)\sqrt{\sum_i w_i^{-1}\cosh^2(\lambda\frac{\gamma_i}{w_i})}.\end{aligned}$$

Using $\alpha \le \frac{1}{8\lambda}$, we simplify to get

$$\le \Psi_\lambda(\boldsymbol{\gamma}) - \frac{\alpha\lambda}{2}\sqrt{\sum_i w_i^{-1}\cosh^2(\lambda\frac{\gamma_i}{w_i})} + \alpha\lambda\sqrt{\sum_i w_i^{-1}},$$

as required. $\square$

2.2.2 Designing the robust IPM step

In the last section, we discussed how to decrease Ψ_λ by changing the input $\boldsymbol{\gamma}$ directly. But our real potential $\Phi^t(\boldsymbol{x},\boldsymbol{s}) = \Psi_\lambda(\boldsymbol{\gamma}^t(\boldsymbol{x},\boldsymbol{s}))$ is defined indirectly using $\boldsymbol{x}, \boldsymbol{s}$, and t. In this section, we discuss how to design the Newton-like step for $(\boldsymbol{x},\boldsymbol{s})$ to match a step directly in the input $\boldsymbol{\gamma}$. The definition of some relevant parameters are given in Algorithm 4.

Similar to Section 2.2.1, a natural choice for a gradient descent step on Φ as a function of $\boldsymbol{\mu}$ is in the steepest descent direction:

$$\boldsymbol{\delta}^\star_{\boldsymbol{\mu}} = \operatorname*{argmin}_{\|\boldsymbol{\delta}_{\boldsymbol{\mu}}\|^*_{\boldsymbol{x}}=\alpha} \langle \nabla_{\boldsymbol{\mu}}\Psi_\lambda(\|\boldsymbol{\mu}\|^*_{\boldsymbol{x}}), \boldsymbol{\mu}+\boldsymbol{\delta}_{\boldsymbol{\mu}}\rangle, \tag{2.6}$$

where α denotes the step size.

For any i, we have:

$$\nabla_{\left\|\boldsymbol{\mu}_{[i]}\right\|^*_{\boldsymbol{x}_{[i]}}}\Psi_\lambda(\left\|\boldsymbol{\mu}_{[i]}\right\|^*_{\boldsymbol{x}_{[i]}}) = \frac{\lambda}{w_i}\sinh\left(\frac{\lambda}{w_i}\left\|\boldsymbol{\mu}_{[i]}\right\|^*_{\boldsymbol{x}_{[i]}}\right)$$

and thus

$$\begin{aligned}\nabla_{\boldsymbol{\mu}_{[i]}}\Psi_\lambda(\|\boldsymbol{\mu}_i\|^*_{\boldsymbol{x}_{[i]}}) &= \frac{\lambda\sinh(\frac{\lambda}{w_i}\left\|\boldsymbol{\mu}_{[i]}\right\|^*_{\boldsymbol{x}_{[i]}})}{w_i\left\|\boldsymbol{\mu}_{[i]}\right\|^*_{\boldsymbol{x}_{[i]}}}\cdot\nabla\phi_i(\boldsymbol{x}_{[i]})^{-1}\boldsymbol{\mu}_{[i]} \\ &= \frac{\lambda\sinh(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i^t(\boldsymbol{x},\boldsymbol{s}))}{w_i\boldsymbol{\gamma}_i^t(\boldsymbol{x},\boldsymbol{s})}\cdot\nabla\phi_i(\boldsymbol{x}_{[i]})^{-1}\boldsymbol{\mu}_{[i]}.\end{aligned}$$

Solve (2.6)[1], and recalling that $\boldsymbol{\mu}_{[i]}$ is a function of $\boldsymbol{x}, \boldsymbol{s}, t$, we get

$$\boldsymbol{\delta}^{\star}_{\boldsymbol{\mu},[i]} = -\frac{\alpha \sinh(\frac{\lambda}{w_i}\boldsymbol{\gamma}^t_i(\boldsymbol{x},\boldsymbol{s}))}{\boldsymbol{\gamma}^t_i(\boldsymbol{x},\boldsymbol{s}) \cdot \sqrt{\sum_{j=1}^m w_j^{-1} \sinh^2(\frac{\lambda}{w_j}\boldsymbol{\gamma}^t_j(\boldsymbol{x},\boldsymbol{s}))}} \cdot \boldsymbol{\mu}^t_{[i]}(\boldsymbol{x},\boldsymbol{s}).$$

We will replace sinh by cosh in the denominator as in Lemma 2.5 to avoid issues with dividing by zero. For notational convenience, define

$$\boldsymbol{k}^t_i(\boldsymbol{x},\boldsymbol{s}) \stackrel{\text{def}}{=} \frac{\sinh(\frac{\lambda}{w_i}\boldsymbol{\gamma}^t_i(\boldsymbol{x},\boldsymbol{s}))}{\boldsymbol{\gamma}^t_i(\boldsymbol{x},\boldsymbol{s}) \cdot \sqrt{\sum_{j=1}^m w_j^{-1} \cosh^2(\frac{\lambda}{w_i}\boldsymbol{\gamma}^t_j(\boldsymbol{x},\boldsymbol{s}))}}, \tag{2.7}$$

So that the modified step can be written as

$$\boldsymbol{\delta}'_{\boldsymbol{\mu},[i]} = -\alpha \cdot \boldsymbol{k}^t_i(\boldsymbol{x},\boldsymbol{s}) \cdot \boldsymbol{\mu}^t_{[i]}(\boldsymbol{x},\boldsymbol{s}).$$

The central idea in the robust interior point method is to compute $\boldsymbol{\delta}_{\boldsymbol{\mu},[i]}$ at each step not as a function of $\boldsymbol{x}, \boldsymbol{s}$, and t, but instead as a function of their approximations $\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}, \overline{t}$. If we can guarantee that the approximations change in a limited manner across the path following algorithm, then special data structures can be used to implement more efficient steps.

Hence, the real step in $\boldsymbol{\mu}$ taken in the algorithm is given by

$$\boldsymbol{\delta}_{\boldsymbol{\mu},[i]} \stackrel{\text{def}}{=} -\alpha \cdot \boldsymbol{k}^{\overline{t}}_i(\overline{\boldsymbol{x}},\overline{\boldsymbol{s}}) \cdot \boldsymbol{\mu}^{\overline{t}}_{[i]}(\overline{\boldsymbol{x}},\overline{\boldsymbol{s}}), \tag{2.8}$$

where $\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}, \overline{t}$ are approximations to $\boldsymbol{x}, \boldsymbol{s}, t$ that we will discuss shortly. They are designed to match the conditions of Lemma 2.5, which showed that Φ is robust under perturbations in its input.

Next, we translate the step in $\boldsymbol{\mu}$ into the corresponding step in $(\boldsymbol{x},\boldsymbol{s})$. To approximately move $\boldsymbol{\mu}$ to $\boldsymbol{\mu} + \boldsymbol{\delta}_{\boldsymbol{\mu}}$, the corresponding Newton step in $(\boldsymbol{x},\boldsymbol{s})$ would be $(\boldsymbol{\delta}'_{\boldsymbol{x}}, \boldsymbol{\delta}'_{\boldsymbol{s}})$ that satisfies:

$$\begin{aligned} \frac{1}{t}\boldsymbol{\delta}'_{\boldsymbol{s}} + \nabla^2\phi(\boldsymbol{x})\boldsymbol{\delta}'_{\boldsymbol{x}} &= \boldsymbol{\delta}_{\boldsymbol{\mu}} \\ \mathbf{A}\boldsymbol{\delta}'_{\boldsymbol{x}} &= \mathbf{0} \\ \mathbf{A}^\top\boldsymbol{\delta}'_{\boldsymbol{y}} + \boldsymbol{\delta}'_{\boldsymbol{s}} &= \mathbf{0}. \end{aligned}$$

[1]The derivation of the formula is not used in the main proof as this is just a motivation for the choice of the step. Therefore, we skip the proof of this. An alternative choice is the gradient step on $\min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b},\mathbf{A}^\top\boldsymbol{y}+\boldsymbol{s}=\boldsymbol{c}} \Phi^t(\boldsymbol{x},\boldsymbol{s})$. This step will be very similar to the step we use in this chapter. But it contains few more terms and may make the proof longer.

Using $\mathbf{H}_{\boldsymbol{x}}$ to denote $\nabla^2\phi(\boldsymbol{x})$, we solve the system above to get

$$\begin{aligned}\boldsymbol{\delta}_{\boldsymbol{x}}' &= \mathbf{H}_{\boldsymbol{x}}^{-1}\boldsymbol{\delta}_{\boldsymbol{\mu}} - \mathbf{H}_{\boldsymbol{x}}^{-1}\mathbf{A}^\top(\mathbf{A}\mathbf{H}_{\boldsymbol{x}}^{-1}\mathbf{A}^\top)^{-1}\mathbf{A}\mathbf{H}_{\boldsymbol{x}}^{-1}\boldsymbol{\delta}_{\boldsymbol{\mu}},\\ \boldsymbol{\delta}_{\boldsymbol{s}}' &= t\mathbf{A}^\top(\mathbf{A}\mathbf{H}_{\boldsymbol{x}}^{-1}\mathbf{A}^\top)^{-1}\mathbf{A}\mathbf{H}_{\boldsymbol{x}}^{-1}\boldsymbol{\delta}_{\boldsymbol{\mu}}.\end{aligned}$$

Define $\mathbf{P}_{\boldsymbol{x}} \stackrel{\text{def}}{=} \mathbf{H}_{\boldsymbol{x}}^{-1/2}\mathbf{A}^\top(\mathbf{A}\mathbf{H}_{\boldsymbol{x}}^{-1}\mathbf{A}^\top)^{-1}\mathbf{A}\mathbf{H}_{\boldsymbol{x}}^{-1/2}$ to be the orthogonal projection matrix, then we can rewrite the step as

$$\begin{aligned}\boldsymbol{\delta}_{\boldsymbol{x}}' &\stackrel{\text{def}}{=} \mathbf{H}_{\boldsymbol{x}}^{-1/2}(\mathbf{I} - \mathbf{P}_{\boldsymbol{x}})\mathbf{H}_{\boldsymbol{x}}^{-1/2}\boldsymbol{\delta}_{\boldsymbol{\mu}},\\ \boldsymbol{\delta}_{\boldsymbol{s}}' &\stackrel{\text{def}}{=} t\mathbf{H}_{\boldsymbol{x}}^{1/2}\mathbf{P}_{\boldsymbol{x}}\mathbf{H}_{\boldsymbol{x}}^{-1/2}\boldsymbol{\delta}_{\boldsymbol{\mu}}.\end{aligned}$$

Once again, in the spirit of *robust* interior point methods, we do not want the step in $\boldsymbol{x}, \boldsymbol{s}$ to depend on $\boldsymbol{x}$ via the $\mathbf{H}_{\boldsymbol{x}}$ and $\mathbf{P}_{\boldsymbol{x}}$ terms and on t, but rather, we only want a dependency on $\overline{\boldsymbol{x}}$ and $\overline{t}$. Hence the real step should be more akin to

$$\begin{aligned}\boldsymbol{\delta}_{\boldsymbol{x}}'' &\stackrel{\text{def}}{=} \mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2}(\mathbf{I} - \mathbf{P}_{\overline{\boldsymbol{x}}})\mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2}\boldsymbol{\delta}_{\boldsymbol{\mu}},\\ \boldsymbol{\delta}_{\boldsymbol{s}}'' &\stackrel{\text{def}}{=} \overline{t}\mathbf{H}_{\overline{\boldsymbol{x}}}^{1/2}\mathbf{P}_{\overline{\boldsymbol{x}}}\mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2}\boldsymbol{\delta}_{\boldsymbol{\mu}}.\end{aligned}$$

Finally, we allow for extra error in the step, due to the robust properties of Φ. Instead of taking the step $(\boldsymbol{\delta}_{\boldsymbol{x}}'', \boldsymbol{\delta}_{\boldsymbol{s}}'')$ as defined above, our algorithm will work for any actual $(\boldsymbol{\delta}_{\boldsymbol{x}}, \boldsymbol{\delta}_{\boldsymbol{s}})$ satisfying

$$\begin{aligned}\|\boldsymbol{\delta}_{\boldsymbol{x}} - \boldsymbol{\delta}_{\boldsymbol{x}}''\|_{\overline{\boldsymbol{x}}} &\le \overline{\varepsilon}\alpha\\ \|\boldsymbol{\delta}_{\boldsymbol{s}} - \boldsymbol{\delta}_{\boldsymbol{s}}''\|_{\overline{\boldsymbol{x}}}^* &\le \overline{\varepsilon}\alpha\overline{t}.\end{aligned}$$

Expanding out expression, we get the equivalent ℓ_2-error conditions

$$\begin{aligned}\left\|\mathbf{H}_{\overline{\boldsymbol{x}}}^{1/2}\boldsymbol{\delta}_{\boldsymbol{x}} - (\mathbf{I} - \mathbf{P}_{\overline{\boldsymbol{x}}})\mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2}\boldsymbol{\delta}_{\boldsymbol{\mu}}\right\|_2 &\le \overline{\varepsilon}\alpha\\ \left\|\overline{t}^{-1}\mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2}\boldsymbol{\delta}_{\boldsymbol{s}} - \mathbf{P}_{\overline{\boldsymbol{x}}}\mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2}\boldsymbol{\delta}_{\boldsymbol{\mu}}\right\|_2 &\le \overline{\varepsilon}\alpha.\end{aligned} \tag{2.9}$$

We define $\boldsymbol{x}^{(\text{new})} \stackrel{\text{def}}{=} \boldsymbol{x} + \boldsymbol{\delta}_{\boldsymbol{x}}$ and $\boldsymbol{s}^{(\text{new})} \stackrel{\text{def}}{=} \boldsymbol{s} + \boldsymbol{\delta}_{\boldsymbol{s}}$ to be the new primal-dual solution pair after taking a step.

Next, we turn to the requirements on the approximations $\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}, \overline{t}$. As our proofs later will show, in order to take advantage of the robust guarantees in Φ, it suffices for each $i \in [m]$ to have

$$\begin{aligned}\left\|\boldsymbol{x}_{[i]} - \overline{\boldsymbol{x}}_{[i]}\right\|_{\overline{\boldsymbol{x}}_{[i]}} &\le \overline{\varepsilon}\\ \left\|\boldsymbol{s}_{[i]} - \overline{\boldsymbol{s}}_{[i]}\right\|_{\overline{\boldsymbol{x}}_{[i]}}^* &\le \overline{t}\overline{\varepsilon}w_i\\ \left|t - \overline{t}\right| &\le \varepsilon_t\overline{t},\end{aligned} \tag{2.10}$$

where $\varepsilon_t \stackrel{\text{def}}{=} \frac{\overline{\varepsilon}}{4} \cdot \min_i\left(\frac{1}{w_i+\nu_i}\right)$.

2.2.3 Bounding Φ after a step

We have already established Lemma 2.5 for bounding the potential Φ after a step directly on $\boldsymbol{\gamma}$. In this section, we show how to use Lemma 2.5 to bound potential Φ after the true step on $\boldsymbol{x}, \boldsymbol{s}$ that indirectly moves $\boldsymbol{\gamma}$.

Our notation convention is as follows: $\boldsymbol{x}, \boldsymbol{s}, t$ represent the primal and dual solutions and the duality measure at the start of an iteration; $\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}, \overline{t}$ represent their approximations; $\boldsymbol{x}^{(\text{new})}, \boldsymbol{s}^{(\text{new})}, t^{(\text{new})}$ represent these variables at the end of an iteration, with

$$\boldsymbol{x}^{(\text{new})} \stackrel{\text{def}}{=} \boldsymbol{x} + \boldsymbol{\delta}_{\boldsymbol{x}}, \ \boldsymbol{s}^{(\text{new})} \stackrel{\text{def}}{=} \boldsymbol{s} + \boldsymbol{\delta}_{\boldsymbol{s}}, \text{ and } t^{(\text{new})} \stackrel{\text{def}}{=} (1-h)t.$$

For readability, we omit the dual variable in all function parameters. Furthermore, we use the notation $\mathbf{H}_{\boldsymbol{x}} \stackrel{\text{def}}{=} \nabla^2 \phi(\boldsymbol{x})$ and $\mathbf{H}_{\boldsymbol{x},[i]} \stackrel{\text{def}}{=} \nabla^2 \phi_i(\boldsymbol{x}_{[i]})$.

Let

$$\begin{aligned}
\boldsymbol{\varepsilon}_{\boldsymbol{x}} &\stackrel{\text{def}}{=} \mathbf{H}_{\overline{\boldsymbol{x}}}^{1/2} \left(\boldsymbol{\delta}_{\boldsymbol{x}} - \mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2} (\mathbf{I} - \mathbf{P}_{\overline{\boldsymbol{x}}}) \mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2} \boldsymbol{\delta}_{\boldsymbol{\mu}} \right) \\
\boldsymbol{\varepsilon}_{\boldsymbol{s}} &\stackrel{\text{def}}{=} \overline{t}^{-1} \mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2} \left(\boldsymbol{\delta}_{\boldsymbol{s}} - t \mathbf{H}_{\overline{\boldsymbol{x}}}^{1/2} \mathbf{P}_{\overline{\boldsymbol{x}}} \mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2} \boldsymbol{\delta}_{\boldsymbol{\mu}} \right)
\end{aligned}$$

denote the error in the step of $\boldsymbol{\delta}_{\boldsymbol{x}}, \boldsymbol{\delta}_{\boldsymbol{s}}$ tolerated according to the condition (2.9).

For each $i \in [m]$, let $\boldsymbol{\zeta}_{[i]}$ be the error vector such that

$$\boldsymbol{\mu}_{[i]}^{t}(\boldsymbol{x}^{(\text{new})}) = \boldsymbol{\mu}_{[i]}^{t}(\boldsymbol{x}) + \boldsymbol{\delta}_{\boldsymbol{\mu},[i]} + \boldsymbol{\zeta}_{[i]},$$

which is once again tolerated according to the condition (2.9). Denote $\beta_i \stackrel{\text{def}}{=} \left\| \boldsymbol{\zeta}_{[i]} \right\|_{\boldsymbol{x}_{[i]}}^{*}$.

First, we bound the step size for each block i.

Lemma 2.6 (Step size of $\boldsymbol{\delta}_{\boldsymbol{x}}$ and $\boldsymbol{\delta}_{\boldsymbol{s}}$)**.** *Let $\alpha_i \stackrel{\text{def}}{=} \left\| \boldsymbol{\delta}_{\boldsymbol{x},[i]} \right\|_{\overline{\boldsymbol{x}}_{[i]}}$. Then we have $\sqrt{\sum_{i=1}^m w_i \alpha_i^2} \leq \frac{9}{8}\alpha$. In particular, we have $\alpha_i \leq \frac{9}{8}\alpha$.*

Similarly, let $\alpha_i' \stackrel{\text{def}}{=} \left\| \boldsymbol{\delta}_{\boldsymbol{s},[i]} \right\|_{\overline{\boldsymbol{x}}_{[i]}}^{*}$. Then we have $\sqrt{\sum_{i=1}^m {\alpha_i'}^2 / w_i} \leq \frac{9}{8}\alpha t$.

Proof. For $\boldsymbol{\delta}_{\boldsymbol{x}}$, we have

$$\begin{aligned}
\sqrt{\sum_{i=1}^m w_i \alpha_i^2} = \|\boldsymbol{\delta}_{\boldsymbol{x}}\|_{\overline{\boldsymbol{x}}} &\leq \left\| (\mathbf{I} - \mathbf{P}_{\overline{\boldsymbol{x}}}) \mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2} \boldsymbol{\delta}_{\boldsymbol{\mu}} \right\|_2 + \overline{\varepsilon}\alpha && \text{(by choice of } \boldsymbol{\delta}_{\boldsymbol{x}}\text{)} \\
&\leq \left\| \mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2} \boldsymbol{\delta}_{\boldsymbol{\mu}} \right\|_2 + \overline{\varepsilon}\alpha && \text{(since } (\mathbf{I} - \mathbf{P}) \text{ is an orthogonal projection)} \\
&\leq \alpha + \overline{\varepsilon}\alpha && \text{(by definition of step size)} \\
&\leq \frac{9}{8}\alpha. && \text{(by definition of } \overline{\varepsilon}\text{)}
\end{aligned}$$

Similarly, for $\boldsymbol{\delta_s}$, we have

$$\begin{aligned}\sqrt{\sum_{i=1}^{m} {\alpha_i'}^2/w_i} = \|\boldsymbol{\delta_s}\|_{\overline{\boldsymbol{x}}}^* &\le \bar{t}\left\|\mathbf{P}_{\overline{\boldsymbol{x}}}\mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2}\boldsymbol{\delta_\mu}\right\|_2 + \bar{\varepsilon}\alpha\bar{t} && \text{(by choice of } \boldsymbol{\delta_s}\text{)}\\ &\le \bar{t}\left\|\mathbf{H}_{\boldsymbol{x}}^{-1/2}\boldsymbol{\delta_\mu}\right\|_2 + \bar{\varepsilon}\alpha\bar{t} && \text{(since } \mathbf{P} \text{ is an orthogonal projection)}\\ &\le (1+\bar{\varepsilon})\alpha\bar{t} && \text{(by definition of step size)}\\ &\le \frac{9}{8}\alpha t. && \text{(by choice of } \bar{\varepsilon} \text{ and guarantees on } \bar{t}\text{)}\end{aligned}$$

□

Next, we show that $\boldsymbol{\mu}^{(\text{new})} \overset{\text{def}}{=} \boldsymbol{\mu}^t(\boldsymbol{x}^{(\text{new})})$ is close to $\boldsymbol{\mu} + \boldsymbol{\delta_\mu}$. That is, the additional error we allow on $\boldsymbol{\delta_x}, \boldsymbol{\delta_s}$ captured by the term $\boldsymbol{\zeta}$ does not affect the goodness of the step in $\boldsymbol{\mu}$.

Lemma 2.7 (Small error on $\boldsymbol{\mu}$). *We have* $\|\boldsymbol{\zeta}\|_{\boldsymbol{x}}^* \overset{\text{def}}{=} \sqrt{\sum_{i=1}^m \beta_i^2/w_i} \le 15\bar{\varepsilon}\alpha$.

Proof. By the definition established at the start of this section, we have

$$\boldsymbol{\delta}_{\boldsymbol{\mu},[i]} = \frac{1}{\bar{t}}\boldsymbol{\delta}_{\boldsymbol{s},[i]} + w_i\mathbf{H}_{\overline{\boldsymbol{x}},[i]}\boldsymbol{\delta}_{\boldsymbol{x},[i]} - \left(w_i\mathbf{H}_{\overline{\boldsymbol{x}},[i]}\right)^{1/2}(\boldsymbol{\varepsilon_x} + \boldsymbol{\varepsilon_s}). \tag{2.11}$$

By definition of $\boldsymbol{\mu}$, we have

$$\begin{aligned}&\boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}^{(\text{new})})\\ &= \frac{\boldsymbol{s}_{[i]}^{(\text{new})}}{t} + w_i\nabla\phi_i(\boldsymbol{x}^{(\text{new})})\\ &= \boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}) + \frac{1}{t}\boldsymbol{\delta}_{\boldsymbol{s},[i]} + w_i\left(\nabla\phi_i(\boldsymbol{x}^{(\text{new})}) - \nabla\phi_i(\boldsymbol{x}_{[i]})\right)\\ &= \boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}) + \boldsymbol{\delta}_{\boldsymbol{\mu},[i]}\\ &\quad + \underbrace{w_i(\nabla\phi_i(\boldsymbol{x}^{(\text{new})}) - \nabla\phi_i(\boldsymbol{x}_{[i]}) - \mathbf{H}_{\overline{\boldsymbol{x}},[i]}\boldsymbol{\delta_x})}_{\boldsymbol{\zeta}_{[i]}^{(1)}} + \underbrace{\left(\mathbf{H}_{\overline{\boldsymbol{x}}}^{1/2}(\boldsymbol{\varepsilon_x} + \boldsymbol{\varepsilon_s})\right)_i}_{\boldsymbol{\zeta}_{[i]}^{(2)}} + \underbrace{(\frac{1}{t} - \frac{1}{\bar{t}})\boldsymbol{\delta_s}}_{\boldsymbol{\zeta}_{[i]}^{(3)}},\end{aligned} \tag{2.12}$$

where the last line follows from (2.11). Note that we have $\boldsymbol{\zeta}_{[i]} = \boldsymbol{\zeta}_{[i]}^{(1)} + \boldsymbol{\zeta}_{[i]}^{(2)} + \boldsymbol{\zeta}_{[i]}^{(3)}$, and our goal is to bound

$$\sqrt{\sum_{i=1}^m \beta_i^2/w_i} = \|\boldsymbol{\zeta}\|_{\boldsymbol{x}}^* \le \left\|\boldsymbol{\zeta}^{(1)}\right\|_{\boldsymbol{x}}^* + \left\|\boldsymbol{\zeta}^{(2)}\right\|_{\boldsymbol{x}}^* + \left\|\boldsymbol{\zeta}^{(3)}\right\|_{\boldsymbol{x}}^*$$

We proceed by bounding each term separately.

To bound $\boldsymbol{\zeta}^{(1)}_{[i]}$, we first define $\boldsymbol{x}^{(u)} \overset{\text{def}}{=} u\boldsymbol{x}^{(\text{new})}+(1-u)\boldsymbol{x}$, which we use for a trick in simplification below. Then, we have

$$\begin{aligned}\boldsymbol{\zeta}^{(1)}_{[i]}/w_i &= \nabla\phi_i(\boldsymbol{x}^{(\text{new})}_{[i]}) - \nabla\phi_i(\boldsymbol{x}_{[i]}) - \mathbf{H}_{\overline{\boldsymbol{x}},[i]}\boldsymbol{\delta}_{\boldsymbol{x},[i]}\\ &= \int_0^1 \left(\mathbf{H}_{\boldsymbol{x}^{(u)},[i]} - \mathbf{H}_{\overline{\boldsymbol{x}},[i]}\right)\boldsymbol{\delta}_{\boldsymbol{x},[i]}\, du.\end{aligned}$$

Applying Lemma 2.30 with $\overline{\boldsymbol{x}}_{[i]}$ and $\boldsymbol{x}^{(u)}$, we get

$$\left(1-\left\|\boldsymbol{x}^{(u)}_{[i]} - \overline{\boldsymbol{x}}_{[i]}\right\|_{\overline{\boldsymbol{x}}_{[i]}}\right)^2 \mathbf{H}_{\overline{\boldsymbol{x}},[i]} \preccurlyeq \mathbf{H}_{\boldsymbol{x}^{(u)},[i]} \preccurlyeq \frac{1}{\left(1-\left\|\boldsymbol{x}^{(u)}_{[i]} - \overline{\boldsymbol{x}}_{[i]}\right\|_{\overline{\boldsymbol{x}}_{[i]}}\right)^2}\mathbf{H}_{\overline{\boldsymbol{x}},[i]}. \tag{2.13}$$

To use this inequality, first note that

$$\left\|\boldsymbol{x}^{(u)}_{[i]} - \overline{\boldsymbol{x}}_{[i]}\right\|_{\overline{\boldsymbol{x}}_{[i]}} \leq \left\|\boldsymbol{x}^{(u)}_{[i]} - \boldsymbol{x}_{[i]}\right\|_{\overline{\boldsymbol{x}}_{[i]}} + \left\|\boldsymbol{x}_{[i]} - \overline{\boldsymbol{x}}_{[i]}\right\|_{\overline{\boldsymbol{x}}_{[i]}}$$

The first term is bounded by $u \cdot \left\|\boldsymbol{\delta}_{\boldsymbol{x},[i]}\right\|_{\overline{\boldsymbol{x}}_{[i]}} \leq \alpha_i$ by definition, while the second term is bounded by $\overline{\varepsilon}$ by the approximation guarantee. Hence,

$$\left\|\boldsymbol{x}^{(u)}_{[i]} - \overline{\boldsymbol{x}}_{[i]}\right\|_{\overline{\boldsymbol{x}}_{[i]}} \leq \alpha_i + \overline{\varepsilon} \leq \frac{9}{8}\alpha + \overline{\varepsilon} \leq 2\overline{\varepsilon},$$

where we used $\alpha_i \leq \frac{9}{8}\alpha$ from Lemma 2.6, and $2\alpha \leq \overline{\varepsilon}$.Combined with (2.13), we get

$$(1-2\overline{\varepsilon})^2\,\mathbf{H}_{\overline{\boldsymbol{x}},[i]} \preccurlyeq \mathbf{H}_{\boldsymbol{x}^{(u)},[i]} \preccurlyeq (1+5\overline{\varepsilon})\,\mathbf{H}_{\overline{\boldsymbol{x}},[i]}. \tag{2.14}$$

We also get the following relationships between local norms for any $\boldsymbol{v}$:

$$\begin{aligned}(1-2\overline{\varepsilon})\left\|\boldsymbol{v}\right\|_{\overline{\boldsymbol{x}}_{[i]}} \leq \left\|\boldsymbol{v}\right\|_{\boldsymbol{x}_{[i]}} \leq (1+2\overline{\varepsilon})\left\|\boldsymbol{v}\right\|_{\overline{\boldsymbol{x}}_{[i]}},\\ (1-2\overline{\varepsilon})\left\|\boldsymbol{v}\right\|^*_{\overline{\boldsymbol{x}}_{[i]}} \leq \left\|\boldsymbol{v}\right\|^*_{\boldsymbol{x}_{[i]}} \leq (1+2\overline{\varepsilon})\left\|\boldsymbol{v}\right\|^*_{\overline{\boldsymbol{x}}_{[i]}}.\end{aligned} \tag{2.15}$$

Applying (2.14) and noting $\boldsymbol{x} = \boldsymbol{x}^{(u)}$ with $u = 0$, we get

$$\begin{aligned}&\left\|\left(\mathbf{H}_{\boldsymbol{x}^{(u)},[i]} - \mathbf{H}_{\boldsymbol{x},[i]}\right)\boldsymbol{\delta}_{\boldsymbol{x},[i]}\right\|^*_{\boldsymbol{x}_{[i]}}\\ &= \sqrt{\boldsymbol{\delta}^\top_{\boldsymbol{x},[i]}\left(\mathbf{H}_{\boldsymbol{x}^{(u)},[i]} - \mathbf{H}_{\overline{\boldsymbol{x}},[i]}\right)^\top\left(\mathbf{H}_{\boldsymbol{x},[i]}\right)^{-1}\left(\mathbf{H}_{\boldsymbol{x}^{(u)},[i]} - \mathbf{H}_{\overline{\boldsymbol{x}},[i]}\right)\boldsymbol{\delta}_{\boldsymbol{x},[i]}}\\ &\leq (1+5\overline{\varepsilon})\cdot 5\overline{\varepsilon}\sqrt{\boldsymbol{\delta}^\top_{\boldsymbol{x},[i]}\mathbf{H}_{\overline{\boldsymbol{x}},[i]}\boldsymbol{\delta}_{\boldsymbol{x},[i]}}\\ &\leq 6\overline{\varepsilon}\left\|\boldsymbol{\delta}_{\boldsymbol{x},[i]}\right\|_{\overline{\boldsymbol{x}}_{[i]}}\\ &= 6\overline{\varepsilon}\alpha_i.\end{aligned}$$

Hence,

$$\left\|\boldsymbol{\zeta}^{(1)}_{[i]}\right\|^*_{\boldsymbol{x}_{[i]}} \le w_i \int_0^1 \left\|\left(\mathbf{H}_{\boldsymbol{x}^{(u)},[i]} - \mathbf{H}_{\overline{\boldsymbol{x}},[i]}\right) \boldsymbol{\delta}_{\boldsymbol{x},[i]}\right\|^*_{\boldsymbol{x}_{[i]}} du \le 6\overline{\varepsilon} w_i \alpha_i.$$

Summing over all i, we get

$$\sqrt{\sum_i \left(\left\|\boldsymbol{\zeta}^{(1)}_{[i]}\right\|^*_{\boldsymbol{x}_{[i]}}\right)^2 / w_i} = \left\|\boldsymbol{\zeta}^{(1)}\right\|^*_{\boldsymbol{x}} \le 6\overline{\varepsilon}\sqrt{\sum_i w_i \alpha_i^2} \le 9\overline{\varepsilon}\alpha,$$

where the last step follows from Lemma 2.6.

For the $\boldsymbol{\zeta}^{(2)}$ term in (2.12), we have

$$\begin{aligned}
\sqrt{\sum_i \left(\left\|\boldsymbol{\zeta}^{(2)}_{[i]}\right\|^*_{\boldsymbol{x}_{[i]}}\right)^2 / w_i} = \left\|\boldsymbol{\zeta}^{(2)}\right\|^*_{\boldsymbol{x}} &= \left\|\mathbf{H}^{1/2}_{\overline{\boldsymbol{x}}}(\boldsymbol{\varepsilon}_{\boldsymbol{x}} + \boldsymbol{\varepsilon}_{\boldsymbol{s}})\right\|^*_{\boldsymbol{x}} \\
&\le (1+5\overline{\varepsilon}) \left\|\mathbf{H}^{1/2}_{\overline{\boldsymbol{x}}}(\boldsymbol{\varepsilon}_{\boldsymbol{x}} + \boldsymbol{\varepsilon}_{\boldsymbol{s}})\right\|^*_{\overline{\boldsymbol{x}}} \\
&\le 2\left\|\boldsymbol{\varepsilon}_{\boldsymbol{x}} + \boldsymbol{\varepsilon}_{\boldsymbol{s}}\right\|_2 \\
&\le 4\overline{\varepsilon}\alpha,
\end{aligned}$$

where we used $\|\boldsymbol{\varepsilon}_{\boldsymbol{x}}\|_2 \le \overline{\varepsilon}\alpha$ and $\|\boldsymbol{\varepsilon}_{\boldsymbol{s}}\|_2 \le \overline{\varepsilon}\alpha$ as guaranteed by the algorithm.

Finally, for the $\boldsymbol{\zeta}^{(3)}$ term in (2.12), we note that

$$\begin{aligned}
\sqrt{\sum_{i=1}^m \left(\left\|\boldsymbol{\zeta}^{(3)}_{[i]}\right\|^*_{\boldsymbol{x}_{[i]}}\right)^2 / w_i} = \left\|\boldsymbol{\zeta}^{(3)}\right\|^*_{\boldsymbol{x}} &= \sqrt{\sum_i \left(\left\|\left(\frac{1}{t} - \frac{1}{\overline{t}}\right)\boldsymbol{\delta}_{\boldsymbol{s},[i]}\right\|^*_{\boldsymbol{x}_{[i]}}\right)^2 / w_i} \\
&= \frac{1}{t}\left|\frac{\overline{t}-t}{\overline{t}}\right| \sqrt{\sum_i \left(\left\|\boldsymbol{\delta}_{\boldsymbol{s},[i]}\right\|^*_{\boldsymbol{x}_{[i]}}\right)^2 / w_i} \\
&\le \frac{1}{t}\left|\frac{\overline{t}-t}{\overline{t}}\right| \cdot \frac{9}{8}\alpha t \qquad \text{(by Lemma 2.6)} \\
&\le 2\overline{\varepsilon}\alpha,
\end{aligned}$$

where the final line follows from the guarantee that $|t - \overline{t}| \le \varepsilon_t \overline{t}$ and $\varepsilon_t \le \overline{\varepsilon}$.

Combining the bounds on the three terms, we have $9\overline{\varepsilon}\alpha + 4\overline{\varepsilon}\alpha + 2\overline{\varepsilon}\alpha \le 15\overline{\varepsilon}\alpha$, as required. □

Next, we can show that the first condition $|\boldsymbol{\gamma}_i - \overline{\boldsymbol{\gamma}}_i| \le \frac{w_i}{8\lambda}$ in Lemma 2.5 holds. In our setting, $\boldsymbol{\gamma}_i$ corresponds to the actual centrality measure $\boldsymbol{\gamma}^t_i(\boldsymbol{x})$ on block i, while $\overline{\boldsymbol{\gamma}}_i$ corresponds to the perturbation $\boldsymbol{\gamma}^{\overline{t}}_i(\overline{\boldsymbol{x}})$ where the centrality is computed for $\overline{\boldsymbol{x}}, \overline{t}$.

Lemma 2.8. *Assume $\boldsymbol{\gamma}_i^t(\boldsymbol{x}) \le w_i$ for all $i \in [m]$. Then, we have*

$$\left\|\boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}) - \boldsymbol{\mu}_{[i]}^{\overline{t}}(\overline{\boldsymbol{x}})\right\|_{\boldsymbol{x}_{[i]}}^* \le 4\overline{\varepsilon} w_i, \text{ and } \left|\boldsymbol{\gamma}_i^t(\boldsymbol{x}) - \boldsymbol{\gamma}_i^{\overline{t}}(\overline{\boldsymbol{x}})\right| \le 6\overline{\varepsilon} w_i.$$

Proof. We continue with notation introduced in the proof of Lemma 2.7. For the first result, by triangle inequality, we have

$$\left\|\boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}) - \boldsymbol{\mu}_{[i]}^{\overline{t}}(\overline{\boldsymbol{x}})\right\|_{\overline{\boldsymbol{x}}_{[i]}}^* \le \left\|\frac{\boldsymbol{s}_{[i]}}{t} - \frac{\overline{\boldsymbol{s}}_i}{\overline{t}}\right\|_{\overline{\boldsymbol{x}}_{[i]}}^* + w_i \left\|\nabla\phi_i(\boldsymbol{x}_{[i]}) - \nabla\phi_i(\overline{\boldsymbol{x}}_{[i]})\right\|_{\overline{\boldsymbol{x}}_{[i]}}^*. \tag{2.16}$$

Note the inconsistency between t and $\overline{t}$ in the denominator of the first term. To bound it, we have

$$\begin{aligned}
\left\|\frac{\boldsymbol{s}_{[i]}}{t} - \frac{\overline{\boldsymbol{s}}_{[i]}}{\overline{t}}\right\|_{\overline{\boldsymbol{x}}_{[i]}}^* &\le \left\|\frac{\boldsymbol{s}_{[i]}}{\overline{t}} - \frac{\overline{\boldsymbol{s}}_{[i]}}{\overline{t}}\right\|_{\overline{\boldsymbol{x}}_{[i]}}^* + \left\|\frac{\boldsymbol{s}_{[i]}}{t} - \frac{\boldsymbol{s}_{[i]}}{\overline{t}}\right\|_{\overline{\boldsymbol{x}}_{[i]}}^* \\
&\le \overline{\varepsilon} w_i + \left(1 - \frac{t}{\overline{t}}\right)\left\|\frac{\boldsymbol{s}_{[i]}}{t}\right\|_{\overline{\boldsymbol{x}}_{[i]}}^* && \text{(first term by guarantee of } \overline{\boldsymbol{s}}\text{)} \\
&\le \overline{\varepsilon} w_i + \left(1 - \frac{t}{\overline{t}}\right)\left(\boldsymbol{\gamma}_i^t(\boldsymbol{x}) + w_i \left\|\nabla\phi_i(\boldsymbol{x}_{[i]})\right\|_{\overline{\boldsymbol{x}}_{[i]}}^*\right) && \text{(by definition of } \boldsymbol{\gamma}\text{)} \\
&\le \overline{\varepsilon} w_i + \left(\frac{\overline{t} - t}{\overline{t}}\right)\left(w_i + 2w_i \left\|\nabla\phi_i(\boldsymbol{x}_{[i]})\right\|_{\boldsymbol{x}_{[i]}}^*\right) \\
& && \text{(by assumption of lemma, and (2.15))} \\
&\le \overline{\varepsilon} w_i + \left(\frac{\overline{t} - t}{\overline{t}}\right)(w_i + 2w_i\sqrt{\nu_i}) && \text{(by definition of self-concordance)} \\
&\le \overline{\varepsilon} w_i + \varepsilon_t w_i (1 + 2\sqrt{\nu_i}) \\
&\le 2\overline{\varepsilon} w_i. && \text{(by definition of } \varepsilon_t\text{)}
\end{aligned}$$

For the second term, we use the same integration trick as in the proof of Lemma 2.7. By (2.14), we have $\mathbf{H}_{\boldsymbol{x}^{(u)},[i]}(\mathbf{H}_{\overline{\boldsymbol{x}},[i]})^{-1}\mathbf{H}_{\boldsymbol{x}^{(u)},[i]} \preccurlyeq (1+5\overline{\varepsilon})^2 \mathbf{H}_{\overline{\boldsymbol{x}},[i]}$. Then,

$$\begin{aligned}
w_i \left\|\nabla\phi_i(\boldsymbol{x}_{[i]}) - \nabla\phi_i(\overline{\boldsymbol{x}}_{[i]})\right\|_{\overline{\boldsymbol{x}}_{[i]}}^* &= w_i \left\|\int_0^1 \mathbf{H}_{\boldsymbol{x}^{(u)},[i]}(\boldsymbol{x}_{[i]} - \overline{\boldsymbol{x}}_{[i]})\, du\right\|_{\overline{\boldsymbol{x}}_{[i]}}^* \\
&\le w_i \int_0^1 \left\|\mathbf{H}_{\boldsymbol{x}^{(u)},[i]}\left(\boldsymbol{x}_{[i]} - \overline{\boldsymbol{x}}_{[i]}\right)\right\|_{\overline{\boldsymbol{x}}_{[i]}}^* du \\
&\le (1+5\overline{\varepsilon}) w_i \left\|\boldsymbol{x}_{[i]} - \overline{\boldsymbol{x}}_{[i]}\right\|_{\overline{\boldsymbol{x}}_{[i]}} \\
&\le 2\overline{\varepsilon} w_i. && \text{(by guarantee of } \overline{\boldsymbol{x}}\text{)}
\end{aligned}$$

For the second result, we have

$$\begin{aligned}
&\left|\boldsymbol{\gamma}_i^{\bar{t}}(\overline{\boldsymbol{x}}) - \boldsymbol{\gamma}_i^{t}(\boldsymbol{x})\right| \\
&= \left|\left\|\boldsymbol{\mu}_{[i]}^{\bar{t}}(\overline{\boldsymbol{x}})\right\|_{\overline{\boldsymbol{x}}_{[i]}}^{*} - \left\|\boldsymbol{\mu}_{[i]}^{t}(\boldsymbol{x})\right\|_{\boldsymbol{x}_{[i]}}^{*}\right| && \text{(by definition)}\\
&\le \left|\left\|\boldsymbol{\mu}_{i}^{\bar{t}}(\overline{\boldsymbol{x}}) - \boldsymbol{\mu}_i^t(\boldsymbol{x})\right\|_{\overline{\boldsymbol{x}}_{[i]}}^{*} + \left\|\boldsymbol{\mu}_{i}^{t}(\boldsymbol{x})\right\|_{\overline{\boldsymbol{x}}_{[i]}}^{*} - \left\|\boldsymbol{\mu}_{i}^{t}(\boldsymbol{x})\right\|_{\boldsymbol{x}_{[i]}}^{*}\right| && \text{(by triangle inequality)}\\
&\le 4\overline{\varepsilon} w_i + \left|\left\|\boldsymbol{\mu}_{i}^{t}(\boldsymbol{x})\right\|_{\overline{\boldsymbol{x}}_{[i]}}^{*} - \left\|\boldsymbol{\mu}_{i}^{t}(\boldsymbol{x})\right\|_{\boldsymbol{x}_{[i]}}^{*}\right| && \text{(by the first result)}\\
&\le 4\overline{\varepsilon} w_i + 2\left\|\boldsymbol{x}_{[i]} - \overline{\boldsymbol{x}}_{[i]}\right\|_{\overline{\boldsymbol{x}}_{[i]}} \cdot \left\|\boldsymbol{\mu}_{[i]}^{t}(\boldsymbol{x})\right\|_{\boldsymbol{x}_{[i]}}^{*} && \text{(by Lemma 2.30)}\\
&= 4\overline{\varepsilon} w_i + 2\overline{\varepsilon}\cdot \boldsymbol{\gamma}_i^t(\boldsymbol{x}) && \text{(by guarantee on } \overline{\boldsymbol{x}})\\
&\le 6\overline{\varepsilon} w_i,
\end{aligned}$$

as required. □

Next, we show that the change in $\boldsymbol{\gamma}^t(\boldsymbol{x})$ after a Newton step on $\boldsymbol{x}$ is comparable to a step $\boldsymbol{\delta}$ in $\boldsymbol{\gamma}$ established in Lemma 2.5. We need the following helper lemma to bound some of the terms.

Lemma 2.9. *Suppose that* $\Phi^t(\boldsymbol{x}) \le \cosh(\lambda)$, *then we have*

- $\boldsymbol{\gamma}_i^t(\boldsymbol{x}) \le w_i$,
- $\boldsymbol{\gamma}_i^{\bar{t}}(\overline{\boldsymbol{x}}) \le 2w_i$,
- $0 \le \boldsymbol{k}_i^{\bar{t}}(\overline{\boldsymbol{x}}) \le \lambda$.

Proof. Recall $\Phi^t(\boldsymbol{x}) \stackrel{\text{def}}{=} \sum_{i=1}^m \cosh\left(\lambda \frac{\boldsymbol{\gamma}_i^t(\boldsymbol{x})}{w_i}\right)$, so by assumption of the lemma, we have $\boldsymbol{\gamma}_i^t(\boldsymbol{x}) \le w_i$ for all i.

Lemma 2.8 shows that
$$\left|\boldsymbol{\gamma}_i^t(\boldsymbol{x}) - \boldsymbol{\gamma}_i^{\bar{t}}(\overline{\boldsymbol{x}})\right| \le 6\overline{\varepsilon} w_i.$$
Combined with the first inequality, we have $\boldsymbol{\gamma}_i^{\bar{t}}(\overline{\boldsymbol{x}}) \le 2w_i$.

For the third inequality, we note that $\boldsymbol{k}_i^{\bar{t}} \ge 0$ by definition. If $\boldsymbol{\gamma}_i^{\bar{t}}(\overline{\boldsymbol{x}}) \ge \frac{w_i}{\lambda}$, then using the fact $0 \le \tanh(x) \le 1$ for all x, we have

$$\boldsymbol{k}_i^{\bar{t}}(\overline{\boldsymbol{x}}) \le \frac{w_i \sinh(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i^{\bar{t}}(\overline{\boldsymbol{x}}))}{\boldsymbol{\gamma}_i^{\bar{t}}(\overline{\boldsymbol{x}}) \cdot \cosh(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i^{\bar{t}}(\overline{\boldsymbol{x}}))} \le \lambda.$$

Alternatively, if $\boldsymbol{\gamma}_i^{\bar{t}}(\overline{\boldsymbol{x}}) \le \frac{w_i}{\lambda}$, then using the fact $|\sinh(x)| \le 2|x|$ for all $|x| \le 1$, we get

$$\boldsymbol{k}_i^{\bar{t}}(\overline{\boldsymbol{x}}) \le \frac{2\frac{\lambda}{w_i}\boldsymbol{\gamma}_i^{\bar{t}}(\overline{\boldsymbol{x}})}{\boldsymbol{\gamma}_i^{\bar{t}}(\overline{\boldsymbol{x}}) \cdot \sqrt{\sum_{j=1}^m w_j^{-1}\cosh^2(\frac{\lambda}{w_j}\boldsymbol{\gamma}_j^{\bar{t}}(\overline{\boldsymbol{x}}))}} \le \frac{2\lambda}{w_i\sqrt{4\sum_{j=1}^m w_j^{-1}}} \le \lambda.$$

□

Finally, we can bound the distance between the new centrality measure after a Newton step, $\boldsymbol{\gamma}^t(\boldsymbol{x}^{(\text{new})})$, and the result of a direct gradient step, $\boldsymbol{\gamma} - \alpha \cdot \boldsymbol{k}^{\bar{t}}(\overline{\boldsymbol{x}}) \circ \boldsymbol{\gamma}^{\bar{t}}(\overline{\boldsymbol{x}})$. Here we crucially use the fact that $\sinh(x)/x$ is bounded at $x = 0$.

Lemma 2.10 (Change in $\boldsymbol{\gamma}$)**.** *Assume* $\Phi^t(\boldsymbol{x}) \leq \cosh(\lambda)$*. For all* $i \in [m]$*, let*

$$\boldsymbol{\xi}_i \overset{\text{def}}{=} \boldsymbol{\gamma}_i^t(\boldsymbol{x}^{(\text{new})}) - \left(\boldsymbol{\gamma}_i^t(\boldsymbol{x}) - \alpha \cdot \boldsymbol{k}_i^{\bar{t}}(\overline{\boldsymbol{x}}) \cdot \boldsymbol{\gamma}_i^{\bar{t}}(\overline{\boldsymbol{x}})\right).$$

Then, we have

$$\sqrt{\sum_{i=1}^m \boldsymbol{\xi}_i^2 / w_i} \leq 90\overline{\varepsilon}\lambda\alpha + 4\alpha \max_i \left(\frac{\boldsymbol{\gamma}_i^t(\boldsymbol{x})}{w_i}\right).$$

Proof. For notation simplicity, we write $\overline{\boldsymbol{k}}_i \overset{\text{def}}{=} \boldsymbol{k}_i^{\bar{t}}(\overline{\boldsymbol{x}})$. To begin, we have

$$\begin{aligned}
|\boldsymbol{\xi}_i| &= \left| \left\| \boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}^{(\text{new})}) \right\|_{\boldsymbol{x}_{[i]}^{(\text{new})}}^* - \left(\boldsymbol{\gamma}_i^t(\boldsymbol{x}) - \alpha \cdot \overline{\boldsymbol{k}}_i \cdot \boldsymbol{\gamma}_i^{\bar{t}}(\overline{\boldsymbol{x}})\right) \right| \\
&\leq \left| \left\| \boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}^{(\text{new})}) \right\|_{\boldsymbol{x}_{[i]}}^* - \boldsymbol{\gamma}_i^t(\boldsymbol{x}) + \alpha \overline{\boldsymbol{k}}_i \boldsymbol{\gamma}_i^{\bar{t}}(\overline{\boldsymbol{x}}) \right| + \left| \left\| \boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}^{(\text{new})}) \right\|_{\boldsymbol{x}_{[i]}^{(\text{new})}}^* - \left\| \boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}^{(\text{new})}) \right\|_{\boldsymbol{x}_{[i]}}^* \right|
\end{aligned} \tag{2.17}$$

For the first term, we have

$$\begin{aligned}
&\left\| \boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}^{(\text{new})}) \right\|_{\boldsymbol{x}_{[i]}}^* \\
&= \left\| \boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}) + \boldsymbol{\delta}_{\boldsymbol{\mu},[i]} + \boldsymbol{\zeta}_{[i]} \right\|_{\boldsymbol{x}_{[i]}}^* \\
&\leq \left\| \boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}) - \alpha \overline{\boldsymbol{k}}_i \boldsymbol{\mu}_{[i]}^{\bar{t}}(\overline{\boldsymbol{x}}) \right\|_{\boldsymbol{x}_{[i]}}^* + \left\| \boldsymbol{\zeta}_{[i]} \right\|_{\boldsymbol{x}_{[i]}}^* \\
&\leq \| \boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}) - \alpha \overline{\boldsymbol{k}}_i \boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}) \|_{\boldsymbol{x}_{[i]}}^* + \alpha \overline{\boldsymbol{k}}_i \cdot \left\| \boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}) - \boldsymbol{\mu}_{[i]}^{\bar{t}}(\overline{\boldsymbol{x}}) \right\|_{\boldsymbol{x}_{[i]}}^* + \left\| \boldsymbol{\zeta}_{[i]} \right\|_{\boldsymbol{x}_{[i]}}^* \\
&= (1 - \alpha \overline{\boldsymbol{k}}_i) \boldsymbol{\gamma}_i^t(\boldsymbol{x}) + \alpha \overline{\boldsymbol{k}}_i \cdot \left\| \boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}) - \boldsymbol{\mu}_{[i]}^{\bar{t}}(\overline{\boldsymbol{x}}) \right\|_{\boldsymbol{x}_{[i]}}^* + \left\| \boldsymbol{\zeta}_{[i]} \right\|_{\boldsymbol{x}_{[i]}}^* \\
&\leq \boldsymbol{\gamma}_i^t(\boldsymbol{x}) + \alpha \overline{\boldsymbol{k}}_i \cdot \left\| \boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}) - \boldsymbol{\mu}_{[i]}^{\bar{t}}(\overline{\boldsymbol{x}}) \right\|_{\boldsymbol{x}_{[i]}}^* + \left\| \boldsymbol{\zeta}_{[i]} \right\|_{\boldsymbol{x}_{[i]}}^* \quad (\text{since } 0 \leq \alpha \overline{\boldsymbol{k}}_i \leq \alpha\lambda \leq 1 \text{ by Lemma 2.9}) \\
&\leq \boldsymbol{\gamma}_i^t(\boldsymbol{x}) + 4\alpha \overline{\boldsymbol{k}}_i \overline{\varepsilon} w_i + \beta_i,
\end{aligned} \tag{2.18}$$

where the last line follows from Lemma 2.8 and definition of β_i. Continuing, we have

$$\begin{aligned}
&\left| \left\| \boldsymbol{\mu}_{[i]}^t(\boldsymbol{x}^{(\text{new})}) \right\|_{\boldsymbol{x}_{[i]}}^* - \boldsymbol{\gamma}_i^t(\boldsymbol{x}) + \alpha \overline{\boldsymbol{k}}_i \boldsymbol{\gamma}_i^{\bar{t}}(\overline{\boldsymbol{x}}) \right| \\
&\leq \left| \boldsymbol{\gamma}_i^t(\boldsymbol{x}) + 4\alpha \overline{\boldsymbol{k}}_i \overline{\varepsilon} w_i + \beta_i - \boldsymbol{\gamma}_i^t(\boldsymbol{x}) + \alpha \overline{\boldsymbol{k}}_i \boldsymbol{\gamma}_i^{\bar{t}}(\overline{\boldsymbol{x}}) \right| \\
&\leq 4\alpha \overline{\boldsymbol{k}}_i \overline{\varepsilon} w_i + \beta_i + \alpha \overline{\boldsymbol{k}}_i \boldsymbol{\gamma}_i^{\bar{t}}(\overline{\boldsymbol{x}}) \\
&\leq 6\alpha \overline{\boldsymbol{k}}_i \overline{\varepsilon} w_i + \beta_i. \qquad (\text{by Lemma 2.9})
\end{aligned}$$

For the second term, we have

$$\begin{aligned}
&\left|\left\|\boldsymbol{\mu}^t_{[i]}(\boldsymbol{x}^{(\mathrm{new})})\right\|^*_{\boldsymbol{x}^{(\mathrm{new})}_{[i]}} - \left\|\boldsymbol{\mu}^t_{[i]}(\boldsymbol{x}^{(\mathrm{new})})\right\|^*_{\boldsymbol{x}_{[i]}}\right| \\
&\le 2\left\|{\boldsymbol{x}_{[i]}}^{(\mathrm{new})} - \boldsymbol{x}_{[i]}\right\|_{\boldsymbol{x}_{[i]}} \cdot \left\|\boldsymbol{\mu}^t_{[i]}(\boldsymbol{x}^{(\mathrm{new})})\right\|^*_{\boldsymbol{x}_{[i]}} && \text{(by Lemma 2.30)} \\
&\le 3\left\|\boldsymbol{\delta}_{\boldsymbol{x},[i]}\right\|_{\overline{\boldsymbol{x}}_{[i]}} \cdot \left(\boldsymbol{\gamma}^t_i(\boldsymbol{x}) + 4\alpha\overline{\boldsymbol{k}}_i\overline{\varepsilon}w_i + \beta_i\right) && \text{(by Eq. (2.18) and Eq. (2.15))} \\
&\le 3\alpha_i\boldsymbol{\gamma}^t_i(\boldsymbol{x}) + 12\alpha\overline{\boldsymbol{k}}_i\overline{\varepsilon}w_i + 3\beta_i. && \text{(by definition of } \alpha_i\text{)}
\end{aligned}$$

Returning to the overall bound in (2.17) by combining the two terms, we have

$$\begin{aligned}
|\boldsymbol{\xi}_i| &\le 6\alpha\overline{\boldsymbol{k}}_i\overline{\varepsilon}w_i + \beta_i + 3\alpha_i\boldsymbol{\gamma}^t_i(\boldsymbol{x}) + 12\alpha\overline{\boldsymbol{k}}_i\overline{\varepsilon}w_i + 3\beta_i \\
&\le 18\alpha\overline{\boldsymbol{k}}_i\overline{\varepsilon}w_i + 3\alpha_i\boldsymbol{\gamma}^t_i(\boldsymbol{x}) + 4\beta_i.
\end{aligned}$$

Next, as a helper, note that

$$\begin{aligned}
\sum_{i=1}^m w_i\overline{\boldsymbol{k}}_i^2 &= \frac{\sum_{i=1}^m w_i \frac{\sinh^2(\frac{\lambda}{w_i}\boldsymbol{\gamma}^{\overline{t}}_i(\overline{\boldsymbol{x}}))}{\boldsymbol{\gamma}^{\overline{t}}_i(\overline{\boldsymbol{x}})^2}}{\sum_{j=1}^m w_j^{-1}\cosh^2(\frac{\lambda}{w_j}\boldsymbol{\gamma}^{\overline{t}}_j(\overline{\boldsymbol{x}}))} \\
&= \lambda^2 \frac{\sum_{i=1}^m w_i^{-1}\frac{w_i^2}{\lambda^2\boldsymbol{\gamma}^{\overline{t}}_i(\overline{\boldsymbol{x}})^2}\sinh^2(\frac{\lambda}{w_i}\boldsymbol{\gamma}^{\overline{t}}_i(\overline{\boldsymbol{x}}))}{\sum_{j=1}^m w_j^{-1}\cosh^2(\frac{\lambda}{w_j}\boldsymbol{\gamma}^{\overline{t}}_j(\overline{\boldsymbol{x}}))} \\
&\le \lambda^2 \frac{\sum_{i=1}^m w_i^{-1}\cosh^2(\frac{\lambda}{w_i}\boldsymbol{\gamma}^{\overline{t}}_i(\overline{\boldsymbol{x}}))}{\sum_{j=1}^m w_j^{-1}\cosh^2(\frac{\lambda}{w_j}\boldsymbol{\gamma}^{\overline{t}}_j(\overline{\boldsymbol{x}}))} \\
&= \lambda^2.
\end{aligned} \tag{2.19}$$

where we used that $\frac{\sinh^2(x)}{x^2} \le \cosh^2(x)$ for all x for the second last inequality.

Finally, we bound the expression in the lemma statement:

$$\begin{aligned}
\sqrt{\sum_{i=1}^m \boldsymbol{\xi}_i^2/w_i} &\le \sqrt{\sum_{i=1}^m \left(18\alpha\overline{\boldsymbol{k}}_i\overline{\varepsilon}w_i\right)^2/w_i} + \sqrt{\sum_{i=1}^m \left(3\alpha_i\boldsymbol{\gamma}^t_i(\boldsymbol{x})\right)^2/w_i} + \sqrt{\sum_{i=1}^m \left(4\beta_i\right)^2/w_i} \\
&= 18\alpha\overline{\varepsilon}\sqrt{\sum_{i=1}^m w_i\overline{\boldsymbol{k}}_i^2} + 3\max\left(\frac{\boldsymbol{\gamma}^t_i(\boldsymbol{x})}{w_i}\right)\cdot\sqrt{\sum_{i=1}^m w_i\alpha_i^2} + 4\sqrt{\sum_{i=1}^m \beta_i^2/w_i} \\
&\le 18\alpha\lambda\overline{\varepsilon} + 4\alpha\max_i\left(\frac{\boldsymbol{\gamma}^t_i(\boldsymbol{x})}{w_i}\right) + 60\alpha\overline{\varepsilon} \\
&\le 90\alpha\lambda\overline{\varepsilon} + 4\alpha\max_i\left(\frac{\boldsymbol{\gamma}^t_i(\boldsymbol{x})}{w_i}\right),
\end{aligned}$$

where we used the helper (2.19), and Lemmas 2.6 and 2.7. □

We are now ready to bound the change of Φ after a Newton step using Lemma 2.5.

Lemma 2.11 (Change of Φ after a Newton step)**.** *Assume $\Phi^t(\boldsymbol{x}) \leq \cosh(\lambda/64)$. Then we have*

$$\Phi^t(\boldsymbol{x}^{(\text{new})}) \leq \Phi^t(\boldsymbol{x}) - \frac{\alpha\lambda}{2}\sqrt{\sum_{i=1}^m w_i^{-1}\cosh^2\left(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i^t(\boldsymbol{x})\right)} + \alpha\lambda\sqrt{\sum_i w_i^{-1}}.$$

Proof. Let $\boldsymbol{\gamma}, \overline{\boldsymbol{\gamma}}, \boldsymbol{\delta}$ from Lemma 2.5 be defined as $\boldsymbol{\gamma}^t(\boldsymbol{x})$, $\boldsymbol{\gamma}^{\bar{t}}(\overline{\boldsymbol{x}})$, and $\boldsymbol{\gamma}_i^t(\boldsymbol{x}^{(\text{new})}) - \boldsymbol{\gamma}_i^t(\boldsymbol{x})$ respectively. We verify the conditions are indeed satisfied for these values. First, Lemma 2.8 shows that $|\boldsymbol{\gamma}_i - \overline{\boldsymbol{\gamma}}_i| \leq 6\overline{\varepsilon}w_i \leq \frac{w_i}{8\lambda}$.

Lemma 2.10 shows that $\boldsymbol{\delta}_i \stackrel{\text{def}}{=} \boldsymbol{\gamma}_i^t(\boldsymbol{x}^{(\text{new})}) - \boldsymbol{\gamma}_i^t(\boldsymbol{x}) = -\alpha \cdot \boldsymbol{k}_i^{\bar{t}}(\overline{\boldsymbol{x}}) \cdot \boldsymbol{\gamma}_i^{\bar{t}}(\overline{\boldsymbol{x}}) + \boldsymbol{\xi}_i$ with

$$\begin{aligned}\sqrt{\sum_{i=1}^m \boldsymbol{\xi}_i^2/w_i} &\leq 90\alpha\lambda\overline{\varepsilon} + 4\alpha\max_i\left(\frac{\boldsymbol{\gamma}_i^t(\boldsymbol{x})}{w_i}\right)\\ &\leq \frac{90\alpha\lambda}{1440\lambda} + \frac{4}{64}\alpha \qquad \text{(by lemma assumption, } |\boldsymbol{\gamma}_i^t(\boldsymbol{x})| \leq \tfrac{w_i}{64}\text{)}\\ &\leq \frac{\alpha}{8}.\end{aligned}$$

Using the definition of $\boldsymbol{k}_i^{\bar{t}}(\overline{\boldsymbol{x}})$, we have

$$\boldsymbol{\delta}_i = \frac{-\alpha\sinh(\frac{\lambda}{w_i}\overline{\boldsymbol{\gamma}}_i)}{\sqrt{\sum_{j=1}^m w_j^{-1}\cosh^2(\frac{\lambda}{w_j}\overline{\boldsymbol{\gamma}}_j)}} + \boldsymbol{\xi}_i,$$

so $\boldsymbol{\xi}$ takes the role of $\boldsymbol{\varepsilon}$ in Lemma 2.5. Applying Lemma 2.5 gives the claimed conclusion. □

Now, we bound the change of Φ after reducing the duality measure t. We first prove a helper lemma:

Lemma 2.12. *We can relate $\boldsymbol{\gamma}_i^{t^{(\text{new})}}(\boldsymbol{x})$ to $\boldsymbol{\gamma}_i^t(\boldsymbol{x})$ by*

$$\boldsymbol{\gamma}_i^t(\boldsymbol{x}) - hw_i\sqrt{\nu_i} \leq \boldsymbol{\gamma}_i^{t^{(\text{new})}}(\boldsymbol{x}) \leq (1+2h)\boldsymbol{\gamma}_i^t(\boldsymbol{x}) + 2hw_i\sqrt{\nu_i}. \tag{2.20}$$

Moreover, we have

$$\cosh(\lambda\boldsymbol{\gamma}_i^t(\boldsymbol{x})/w_i) \leq 2\cosh(\lambda\boldsymbol{\gamma}_i^{t^{(\text{new})}}(\boldsymbol{x})/w_i). \tag{2.21}$$

Proof. Applying the definition, we have

$$\begin{aligned}
\boldsymbol{\gamma}_i^{t^{(\text{new})}}(\boldsymbol{x}) &\stackrel{\text{def}}{=} \left\| \frac{\boldsymbol{s}_{[i]}}{t^{(\text{new})}} + w_i \nabla \phi_i(\boldsymbol{x}_{[i]}) \right\|_{\boldsymbol{x}_{[i]}}^* \\
&= \frac{1}{1-h} \left\| \frac{\boldsymbol{s}_{[i]}}{t} + (1-h) w_i \nabla \phi_i(\boldsymbol{x}_{[i]}) \right\|_{\boldsymbol{x}_{[i]}}^* \\
&\leq \frac{1}{1-h} \boldsymbol{\gamma}_i^t(\boldsymbol{x}) + \frac{h}{1-h} w_i \left\| \nabla \phi_i(\boldsymbol{x}_{[i]}) \right\|_{\boldsymbol{x}_{[i]}}^* \\
&\leq (1+2h) \boldsymbol{\gamma}_i^t(\boldsymbol{x}) + 2h w_i \sqrt{\nu_i},
\end{aligned}$$

where the last line follows by definition of self-concordance. Similar to the argument for the upper bound, the lower bound is given by

$$\boldsymbol{\gamma}_i^{t^{(\text{new})}}(\boldsymbol{x}) \geq \frac{1}{1-h} \boldsymbol{\gamma}_i^t(\boldsymbol{x}) - \frac{h}{1-h} w_i \left\| \nabla \phi_i(\boldsymbol{x}_{[i]}) \right\|_{\boldsymbol{x}_{[i]}}^* .$$

Rearranging, we get

$$\boldsymbol{\gamma}_i^t(\boldsymbol{x}) \leq (1-h) \boldsymbol{\gamma}_i^{t^{(\text{new})}}(\boldsymbol{x}) + h w_i \sqrt{\nu_i},$$

which concludes the first part of the lemma.

We obtain the second part of the lemma by applying Lemma 2.38.

□

Lemma 2.13 (Change of Φ after reducing t)**.** *Assume that* $\Phi^t(\boldsymbol{x}) \leq \cosh(\lambda)$. *We have*

$$\Phi^{t^{(\text{new})}}(\boldsymbol{x}) \leq \Phi^t(\boldsymbol{x}) + \frac{\alpha \lambda}{8} \sqrt{\sum_{i=1}^m w_i^{-1} \cosh^2(\frac{\lambda}{w_i} \boldsymbol{\gamma}_i^t(\boldsymbol{x}))}.$$

Proof. We have,

$$\begin{aligned}
\Phi^{t^{(\text{new})}}(\boldsymbol{x}) &\stackrel{\text{def}}{=} \sum_{i=1}^m \cosh\left(\frac{\lambda}{w_i} \boldsymbol{\gamma}_i^{t^{(\text{new})}}(\boldsymbol{x}) \right) \\
&\leq \sum_{i=1}^m \cosh\left(\frac{\lambda}{w_i} \boldsymbol{\gamma}_i^t(\boldsymbol{x}) + 2h\lambda \left(\boldsymbol{\gamma}_i^t(\boldsymbol{x})/w_i + \sqrt{\nu_i} \right) \right). \qquad \text{(by (2.20))}
\end{aligned}$$

Since $\Phi^t(\boldsymbol{x}) \leq \cosh(\lambda)$, we know $\boldsymbol{\gamma}_i^t(\boldsymbol{x})/w_i \leq 1$; moreover, self-concordance ν_i is always at least 1, so

$$\begin{aligned}
&\leq \sum_{i=1}^m \cosh\left(\frac{\lambda}{w_i} \boldsymbol{\gamma}_i^t(\boldsymbol{x}) + 4h\lambda \sqrt{\nu_i} \right) \\
&\leq \Phi^t(\boldsymbol{x}) + 8h\lambda \sum_{i=1}^m \sqrt{\nu_i} \cosh\left(\frac{\lambda}{w_i} \boldsymbol{\gamma}_i^t(\boldsymbol{x}) \right). \qquad \text{(by Lemma 2.38)}
\end{aligned}$$

Recall $h \overset{\text{def}}{=} \frac{\alpha}{64\sqrt{\sum_{i=1}^m w_i \nu_i}}$. So we have

$$\begin{aligned}
&\le \Phi^t(\boldsymbol{x}) + \frac{\alpha\lambda}{8}\sum_{i=1}^m \frac{\sqrt{\nu_i}}{\sqrt{\sum_{i=1}^m w_i\nu_i}}\cosh\left(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i^t(\boldsymbol{x})\right)\\
&\le \Phi^t(\boldsymbol{x}) + \frac{\alpha\lambda}{8}\sqrt{\sum_{i=1}^m w_i^{-1}\cosh^2(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i^t(\boldsymbol{x}))}, \qquad \text{(by Cauchy-Schwarz)}
\end{aligned}$$

which is the desired conclusion. □

Combining the bound of Φ under a $\boldsymbol{x}$ step (Lemma 2.11) and the bound of Φ under a t step (Lemma 2.13), we get the bound on Φ after 1 iteration of PathFollowing.

Theorem 2.14. *If* $\Phi^t(\boldsymbol{x}) \ge \cosh(\frac{\lambda}{128})$, *then*

$$\Phi^{t^{(\text{new})}}(\boldsymbol{x}^{(\text{new})}) \le \left(1 - \frac{\alpha\lambda}{8\sqrt{\sum_i w_i}}\right)\Phi^t(\boldsymbol{x}) + \alpha\lambda\sqrt{\sum_i w_i^{-1}} \le \Phi^t(\boldsymbol{x}).$$

Otherwise, $\Phi^{t^{(\text{new})}}(\boldsymbol{x}^{(\text{new})}) \le \cosh\left(\frac{\lambda}{64}\right)$.

Proof. We combine the previous lemmas to get

$$\begin{aligned}
&\Phi^{t^{(\text{new})}}(\boldsymbol{x}^{(\text{new})})\\
&\le \Phi^{t^{(\text{new})}}(\boldsymbol{x}) - \frac{\alpha\lambda}{2}\sqrt{\sum_{i=1}^m w_i^{-1}\cosh^2(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i^{t^{(\text{new})}}(\boldsymbol{x}))} + \alpha\lambda\sqrt{\sum_i w_i^{-1}} \qquad \text{(by Lemma 2.11)}\\
&\le \Phi^t(\boldsymbol{x}) + \frac{\alpha\lambda}{8}\sqrt{\sum_{i=1}^m w_i^{-1}\cosh^2(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i^t(\boldsymbol{x}))} - \frac{\alpha\lambda}{4}\sqrt{\sum_{i=1}^m w_i^{-1}\cosh^2(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i^t(\boldsymbol{x}))} + \alpha\lambda\sqrt{\sum_i w_i^{-1}}\\
&\qquad\qquad \text{(by Lemma 2.13 on the first term and (2.21) on the second term)}\\
&\le \Phi^t(\boldsymbol{x}) - \frac{\alpha\lambda}{8}\sqrt{\sum_{i=1}^m w_i^{-1}\cosh^2(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i^t(\boldsymbol{x}))} + \alpha\lambda\sqrt{\sum_i w_i^{-1}}\\
&\le \Phi^t(\boldsymbol{x}) - \frac{\alpha\lambda}{8}\frac{\Phi^t(\boldsymbol{x})}{\sqrt{\sum_i w_i}} + \alpha\lambda\sqrt{\sum_i w_i^{-1}}. \qquad \text{(by Cauchy-Schwarz)}\\
&\le \Phi^t(\boldsymbol{x}) + \alpha\lambda\left(\sqrt{m} - \frac{\Phi^t(x)}{8\sqrt{m}}\right).
\end{aligned}$$

If $\Phi^t(\boldsymbol{x}) \ge \cosh(\frac{\lambda}{128})$, then

$$\frac{\Phi^t(\boldsymbol{x})}{8\sqrt{m}} \ge \frac{\cosh(\frac{\lambda}{128})}{8\sqrt{m}} \ge \frac{\exp(\frac{\lambda}{128})}{16\sqrt{m}} = \frac{\exp(\frac{1}{2}\log(256m\sum_{i=1}^m w_i))}{16\sqrt{m}} \ge \sqrt{m},$$

Hence, $\Phi^{t^{(\text{new})}}(\boldsymbol{x}^{(\text{new})}) \le \Phi^t(\boldsymbol{x})$. On the other hand, if $\Phi^t(\boldsymbol{x}) \le \cosh(\frac{\lambda}{128})$, then recall $\Phi^t(\boldsymbol{x}) \ge m$ for all $\boldsymbol{x}$, so

$$\alpha\lambda\left(\sqrt{m} - \frac{\Phi^t(x)}{8\sqrt{m}}\right) \le \alpha\lambda\left(\sqrt{\Phi^t(\boldsymbol{x})} - \frac{\sqrt{\Phi^t(x)}}{8}\right) \le \frac{\alpha\lambda}{8}\sqrt{\Phi^t(\boldsymbol{x})} \le \Phi^t(\boldsymbol{x}).$$

It follows that $\Phi^{t^{(\text{new})}}(\boldsymbol{x}^{(\text{new})}) \le 2\Phi^t(\boldsymbol{x}) \le 2\cosh(\frac{\lambda}{128}) \le \cosh(\frac{\lambda}{64})$ by choice of λ.

□

2.3 Staying far from the boundary

In this section, we show that the solution throughout PathFollowingRobust (Algorithm 4) is bounded away from the feasible set boundary. While this is not necessary for the proof of Theorem 2.18, it is a useful helper for the initial point reduction covered in the next section; is needed to bound the runtime of flow problem data structures covered in future chapters; and is simply nice to have, for a fuller understanding of the algorithm.

Lemma 2.15. *Let $\boldsymbol{x}, \boldsymbol{s}$ be the solution at the end of an iteration of* PathFollowingRobust*, and let t denote the duality measure used during the iteration. Without loss of generality, suppose $\Phi^t(\boldsymbol{x}, \boldsymbol{s}) \le \cosh(\frac{\lambda}{64})$. For each $i \in [m]$, let η_i be the minimum distance between $\boldsymbol{x}_{[i]}$ and the boundary of K_i, and let $\eta \overset{\text{def}}{=} \min_i \eta_i$. Then*

$$\eta \ge \frac{rt}{128LR + 256\kappa^2 t}.$$

Proof. We will set up an inequality based on some optimality condition of $\boldsymbol{x}$, and from it derive the conclusions of the lemma.

Recall $\mathcal{P} \overset{\text{def}}{=} \{\boldsymbol{x} : \mathbf{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \in K\}$ is the set of feasible solutions. We observe that $\boldsymbol{x}$ is the minimizer of the function $g(\boldsymbol{v}) \overset{\text{def}}{=} \langle \widetilde{\boldsymbol{c}}, \boldsymbol{v}\rangle + t\cdot\phi(\boldsymbol{v})$ over the domain $\mathcal{P}$, where $\widetilde{\boldsymbol{c}} \overset{\text{def}}{=} \boldsymbol{c} - \boldsymbol{s} - t\nabla\phi(\boldsymbol{x})$. Indeed, by definition, we have

$$\nabla g(\boldsymbol{x}) = \widetilde{\boldsymbol{c}} + t\nabla\phi(\boldsymbol{x}) = \boldsymbol{c} - \boldsymbol{s} = \mathbf{A}^\top \boldsymbol{y}$$

for some $\boldsymbol{y}$. Since $\langle \boldsymbol{v}, \nabla g(\boldsymbol{x})\rangle = \langle \boldsymbol{b}, \boldsymbol{y}\rangle$ is constant for all $\boldsymbol{v} \in \mathcal{P}$, we conclude $\nabla g(\boldsymbol{x})$ is orthogonal $\mathcal{P}$ and therefore $\boldsymbol{x}$ is optimal.

Define the path between $\boldsymbol{x}$ and $\boldsymbol{z}$:

$$p(\beta) \overset{\text{def}}{=} (1-\beta)\cdot\boldsymbol{x} + \beta\cdot\boldsymbol{z}.$$

Consider the directional derivative at $\boldsymbol{x}$ in the direction $\boldsymbol{z}-\boldsymbol{x}$. Since $p(0)$ minimizes g, we know $\frac{d}{d\beta}g(p(\beta))|_{\beta=0} \geq 0$. In particular,

$$0 \leq \frac{d}{d\beta}g(p(\beta))|_{\beta=0} = (\widetilde{\boldsymbol{c}} + t\nabla\phi(\boldsymbol{x}))^\top(\boldsymbol{z}-\boldsymbol{x}).$$

Note that $\widetilde{\boldsymbol{c}} = \boldsymbol{c} - t\boldsymbol{\mu}^t(\boldsymbol{x})$, and the assumption of the lemma implies that for all i, we have $\boldsymbol{\gamma}_i^t(\boldsymbol{x}) \overset{\text{def}}{=} \|\boldsymbol{\mu}^t(\boldsymbol{x})\|_{\boldsymbol{x}_{[i]}}^* \leq \frac{w_i}{64}$. Hence,

$$\begin{aligned}
0 &\leq (\widetilde{\boldsymbol{c}} + t\nabla\phi(\boldsymbol{x}))^\top (\boldsymbol{z}-\boldsymbol{x}) \\
&= \boldsymbol{c}^\top(\boldsymbol{z}-\boldsymbol{x}) - t\sum_{i=1}^m \boldsymbol{\mu}_{[i]}^\top(\boldsymbol{z}_{[i]} - \boldsymbol{x}_{[i]}) + t\sum_{i=1}^m w_i \nabla\phi_i(\boldsymbol{x})^\top(\boldsymbol{z}_{[i]} - \boldsymbol{x}_{[i]}) \\
&\leq 2LR + t\sum_{i=1}^m \frac{w_i}{64}\left\|\boldsymbol{z}_{[i]} - \boldsymbol{x}_{[i]}\right\|_{\boldsymbol{x}_{[i]}} + t\sum_{i=1}^m w_i \nabla\phi_i(\boldsymbol{x})^\top(\boldsymbol{z}_{[i]} - \boldsymbol{x}_{[i]}), \qquad (2.22)
\end{aligned}$$

where we used $\|\boldsymbol{c}\|_2 \leq L$ and $\|\boldsymbol{z}-\boldsymbol{x}\|_2 \leq 2R$.

To bound the last two terms, let us first fix $i \in [m]$, and define $\widetilde{\phi}$ to be the ϕ_i restricted on the line p containing $\boldsymbol{z}_{[i]}$ and $\boldsymbol{x}_{[i]}$. Then $\widetilde{\phi}$ is a ν_i-self-concordant barrier function on some interval $[\boldsymbol{\alpha}, \boldsymbol{\beta}]$, where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are two points of ∂K_i uniquely defined by p.

Without loss of generality, suppose the points are ordered $\boldsymbol{\alpha}, \boldsymbol{x}_{[i]}, \boldsymbol{z}_{[i]}, \boldsymbol{\beta}$ on the line p, and we view them in 1 dimension, denoted respectively by α, x, z, β. Then, using Lemma 2.32, we have

$$\begin{aligned}
u_i &\overset{\text{def}}{=} \frac{1}{64}\left\|\boldsymbol{z}_{[i]} - \boldsymbol{x}_{[i]}\right\|_{\boldsymbol{x}_{[i]}} + \nabla\phi_i(\boldsymbol{x}_{[i]})^\top(\boldsymbol{z}_{[i]} - \boldsymbol{x}_{[i]}) \\
&= \frac{1}{64}\sqrt{\widetilde{\phi}''(x)}\,|z-x| + \widetilde{\phi}'(x)(z-x) \\
&\leq 4\nu_i^2 - \frac{1}{64}\left(\frac{z-\alpha}{x-\alpha}\right).
\end{aligned}$$

For most $i \in [m]$, we simply use the bound $u_i \leq 4\nu_i^2$. For the i that attains $\eta_i = \eta$, we use a tighter bound that includes the second term in the above expression. In the next part, we refer to said i. Let $\boldsymbol{q} \overset{\text{def}}{=} \operatorname{argmin}_{\boldsymbol{v}\in\partial K_i}\left\|\boldsymbol{v} - \boldsymbol{x}_{[i]}\right\|_2$. Let ℓ be the line through $\boldsymbol{q}$ and $\boldsymbol{\alpha}$. Let $\mathbf{P}$ be the projection function of $\boldsymbol{x}_{[i]}$ onto ℓ so that $\mathbf{P}\boldsymbol{x}_{[i]} = \boldsymbol{q}$, then let $\boldsymbol{z}' \overset{\text{def}}{=} \mathbf{P}\boldsymbol{z}_{[i]}$ be the projection of $\boldsymbol{z}_{[i]}$ onto ℓ. Since K_i is convex, $\boldsymbol{z}' \notin K_i$. Finally, an argument about similar triangles show

$$\frac{\left\|\boldsymbol{q} - \boldsymbol{x}_{[i]}\right\|_2}{\left\|\boldsymbol{x} - \boldsymbol{\alpha}\right\|_2} = \frac{\left\|\boldsymbol{z}' - \boldsymbol{z}_{[i]}\right\|_2}{\left\|\boldsymbol{z} - \boldsymbol{\alpha}\right\|_2}.$$

Hence,

$$\frac{\eta}{x-\alpha} \geq \frac{r}{z-\alpha}.$$

This gives us the tighter bound $u_i \le 4\nu_i^2 - \frac{r}{64\eta}$ for the i with $\eta_i = \eta$. Substituting back into (2.22), we conclude

$$0 \le 2LR + t\sum_{i=1}^{m} w_i u_i \le 2LR + 4t\sum_{i=1}^{m} w_i \nu_i^2 - \frac{rt}{64\eta} \le 2LR + 4t\kappa^2 - \frac{rt}{64\eta}.$$

Rearranging gives the claimed result. □

Since $\overline{\boldsymbol{x}}$ approximates $\boldsymbol{x}$ in the local norm, we can also bound the distance from $\overline{\boldsymbol{x}}$ to the boundary of K. We omit its proof.

Corollary 2.16. *Let $\overline{\boldsymbol{x}}$ be the approximation of $\boldsymbol{x}$ guaranteed. Then the distance from $\overline{\boldsymbol{x}}$ to the boundary of K is also bounded from below with polynomial dependence on t.*

2.4 Initial point reduction

Path-following requires an initial point close to the the central path, and in this section, we discuss how to find such a point.

At a high level, we accomplish this by modifying the given convex program (P) to obtain a different convex program (P'_T), for which we know an explicit point on its central path. This point, however, may not directly translate to a feasible initial solution for (P). Rather, we will show that if (P'_T) is sufficiently optimized starting from the known point on its central path, then that solution does yield a feasible initial point for the original problem.

The intuition for defining (P'_T) is as follows: To satisfy the constraint $\boldsymbol{x} \in K$, we could consider solving $\min_{\boldsymbol{x}\in K} \boldsymbol{c}^\top \boldsymbol{x} + t\phi(\boldsymbol{x})$ for some parameter t. Since the solution $\boldsymbol{x}$ here may not satisfy the constraint $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$, we instead use variables $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}$, so that $\boldsymbol{x}^{(1)} \in K$ acts as the original variable and $\boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)} \in \mathbb{R}^n_{\ge 0}$ are the extra variables, and enforce $\mathbf{A}\boldsymbol{x}^{(\text{new})} = \boldsymbol{b}$ for $\boldsymbol{x}^{(\text{new})} = \boldsymbol{x}^{(1)} + \boldsymbol{x}^{(2)} - \boldsymbol{x}^{(3)}$. We put a large cost vector on $\boldsymbol{x}^{(2)}$ and $\boldsymbol{x}^{(3)}$ to ensure they are small in magnitude, so that $\boldsymbol{x}^{(1)}$ could be feasible alone.

Formally, consider a convex program of the form (P). For any $T > 0$, we define the modified convex program by

$$\min_{(\boldsymbol{x}^{(1)},\boldsymbol{x}^{(2)},\boldsymbol{x}^{(3)})\in\mathcal{P}} \left\langle \boldsymbol{c}^{(1)}, \boldsymbol{x}^{(1)} \right\rangle + \left\langle \boldsymbol{c}^{(2)}, \boldsymbol{x}^{(2)} \right\rangle + \left\langle \boldsymbol{c}^{(3)}, \boldsymbol{x}^{(3)} \right\rangle \qquad (P'_T)$$

where

$$\begin{aligned}
\mathcal{P} &\overset{\text{def}}{=} \{\boldsymbol{x}^{(1)} \in K, (\boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}) \in \mathbb{R}^{2d}_{\geq 0} : \mathbf{A}(\boldsymbol{x}^{(1)} + \boldsymbol{x}^{(2)} - \boldsymbol{x}^{(3)}) = \boldsymbol{b}\},\\
\boldsymbol{c}^{(1)} &\overset{\text{def}}{=} \boldsymbol{c},\\
\boldsymbol{c}^{(2)} &\overset{\text{def}}{=} \frac{T}{3R + \boldsymbol{x}_\circ - \boldsymbol{x}_c}, \text{ with } \boldsymbol{x}_c \overset{\text{def}}{=} \operatorname*{argmin}_{\boldsymbol{x}\in K} \boldsymbol{c}^\top \boldsymbol{x} + T\phi(\boldsymbol{x}) \text{ and } \boldsymbol{x}_\circ \overset{\text{def}}{=} \operatorname*{argmin}_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b}} \|\boldsymbol{x} - \boldsymbol{x}_c\|_2, \text{ and}\\
\boldsymbol{c}^{(3)} &\overset{\text{def}}{=} \frac{T}{3R} \cdot \mathbf{1}.
\end{aligned}$$

The dual feasible set, given by the dual slack variables, is

$$\mathcal{D} \overset{\text{def}}{=} \left\{\boldsymbol{s}^{(1)} \in K^*, (\boldsymbol{s}^{(2)}, \boldsymbol{s}^{(3)}) \in \mathbb{R}^{2d}_{\geq 0} : \begin{bmatrix} \mathbf{A}^\top \\ \mathbf{A}^\top \\ -\mathbf{A}^\top \end{bmatrix} \boldsymbol{y} + \begin{bmatrix} \boldsymbol{s}^{(1)} \\ \boldsymbol{s}^{(2)} \\ \boldsymbol{s}^{(3)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{c}^{(1)} \\ \boldsymbol{c}^{(2)} \\ \boldsymbol{c}^{(3)} \end{bmatrix} \text{ for some } \boldsymbol{y} \in \mathbb{R}^n\right\}.$$

To solve (P'_T), we could run the interior point method. In this case, the corresponding central path problem is

$$\min_{(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}) \in \mathcal{P}} f_t(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}) \tag{2.23}$$

where the objective function at time t is

$$\begin{aligned}
f_t(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}) \overset{\text{def}}{=} &\left\langle \boldsymbol{c}^{(1)}, \boldsymbol{x}^{(1)} \right\rangle + \left\langle \boldsymbol{c}^{(2)}, \boldsymbol{x}^{(2)} \right\rangle + \left\langle \boldsymbol{c}^{(3)}, \boldsymbol{x}^{(3)} \right\rangle \\
&+ t\phi(\boldsymbol{x}^{(1)}) - t \sum_{i=1}^{n} \left(\log \boldsymbol{x}_i^{(2)} + \log \boldsymbol{x}_i^{(3)} \right).
\end{aligned}$$

Lemma 2.17. *Fix $T > 0$ for (P'_T). The point $\boldsymbol{x} \overset{\text{def}}{=} (\boldsymbol{x}_c, 3R + \boldsymbol{x}_\circ - \boldsymbol{x}_c, 3R)$ is the minimizer of (2.23) when $t = T$, with corresponding dual variables $\boldsymbol{s} \overset{\text{def}}{=} (-T\nabla\phi(\boldsymbol{x}_c), \frac{T}{3R+\boldsymbol{x}_\circ-\boldsymbol{x}_c}, \frac{T}{3R})$.*

Proof. We will show that $\boldsymbol{x} \in \mathcal{P}$ and that it minimizes f_T over $\mathbb{R}^{3d}$, not just $\mathcal{P}$.

For the set constraint, we note that $\boldsymbol{x}^{(1)} \in K$ and $\boldsymbol{x}^{(3)} \geq \mathbf{0}$ by definition. For $\boldsymbol{x}^{(2)}$, we note that the inner radius guarantees the existence of some $\boldsymbol{z} \in K$ with $\mathbf{A}\boldsymbol{z} = \boldsymbol{b}$, and hence $\|\boldsymbol{x}_\circ - \boldsymbol{x}_c\|_2 \leq \|\boldsymbol{z} - \boldsymbol{x}_c\|_2 \leq 2R$. It follows that each coordinate of $\boldsymbol{x}^{(2)}$ is at least R, so $\boldsymbol{x}^{(2)} \geq \mathbf{0}$. Hence, $(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}) \in \mathcal{P}$.

For optimality, since $\boldsymbol{x}^{(1)} \overset{\text{def}}{=} \boldsymbol{x}_c = \operatorname{argmin}_{\boldsymbol{x}\in K} \boldsymbol{c}^\top \boldsymbol{x} + T\phi(\boldsymbol{x})$, we have

$$\begin{aligned}
\nabla_{\boldsymbol{x}^{(1)}} f_T(\boldsymbol{x}) &= \boldsymbol{c}^{(1)} + T\nabla\phi(\boldsymbol{x}^{(1)})\\
&= \boldsymbol{c} + T\nabla\phi(\boldsymbol{x}_c)\\
&= \mathbf{0}.
\end{aligned}$$

By definition of $\boldsymbol{c}^{(2)}$ and $\boldsymbol{c}^{(3)}$, we also have $\nabla_{\boldsymbol{x}^{(2)}} f_T(\boldsymbol{x}) = \boldsymbol{c}^{(2)} - T/\boldsymbol{x}^{(2)} = \boldsymbol{0}$, and similarly $\nabla_{\boldsymbol{x}^{(3)}} f_T(\boldsymbol{x}) = \boldsymbol{0}$. Hence $\boldsymbol{x}$ is the minimizer of f_T. The dual variables are determined by the negative of the barrier function gradient. $\square$

So we have established an initial point on the central path for the problem (P'_T). Using this initial point, we can run the path-following procedure of Algorithm 4, and reduce the central path parameter t from $t_{\text{start}} \overset{\text{def}}{=} T$ to a smaller value t_{end}. This will in turn give $(\boldsymbol{x}^\star, \boldsymbol{s}^\star)$ close to the minimizer of (2.23) at $t = t_{\text{end}}$. Our main theorem shows that this point suffices as an initial point for (P).

Theorem 2.18. *For any $0 \le \delta \le \frac{1}{2}$, let $T \ge 2^{16}(d+\kappa)^5 \cdot \frac{LR}{\delta} \cdot \frac{R}{r}$. Suppose we run Algorithm 4 on the convex program (P'_T) from $t_{\text{start}} \overset{\text{def}}{=} T$ to $t_{\text{end}} \overset{\text{def}}{=} LR$, with initial point given by Lemma 2.17, and receive primal and dual solutions $\boldsymbol{x}^\star \overset{\text{def}}{=} (\boldsymbol{x}^{(1)\star}, \boldsymbol{x}^{(2)\star}, \boldsymbol{x}^{(3)\star}) \in \mathcal{P}$ and $\boldsymbol{s}^\star \overset{\text{def}}{=} (\boldsymbol{s}^{(1)\star}, \boldsymbol{s}^{(2)\star}, \boldsymbol{s}^{(3)\star}) \in \mathcal{D}$ that satisfy*

$$\Phi^{t_{\text{end}}}(\boldsymbol{x}^\star, \boldsymbol{s}^\star) \le \cosh(\frac{\lambda}{64}).$$

Let $\boldsymbol{x}^{(\text{new})} = \boldsymbol{x}^{(1)\star} + \boldsymbol{x}^{(2)\star} - \boldsymbol{x}^{(3)\star}$ and $\boldsymbol{s}^{(\text{new})} = \boldsymbol{s}^{(1)\star}$. Then they satisfy $\mathbf{A}\boldsymbol{x}^{(\text{new})} = \boldsymbol{b}$, $\boldsymbol{x}^{(\text{new})} \in K$, $\mathbf{A}^\top \boldsymbol{y} + \boldsymbol{s}^{(\text{new})} = \boldsymbol{c}$ for some $\boldsymbol{y}$, and for each $i \in [m]$,

$$\gamma_i^{t_{\text{end}}}\left(\boldsymbol{x}^{(\text{new})}, \boldsymbol{s}^{(\text{new})}\right) \le \gamma_i^{t_{\text{end}}}\left(\boldsymbol{x}^{(1)\star}, \boldsymbol{s}^{(1)\star}\right) + \delta w_i.$$

Proof. First, by construction, we have $\mathbf{A}\boldsymbol{x}^{(\text{new})} = \boldsymbol{b}$. Since there exists some $\boldsymbol{y}$ so that $\mathbf{A}^\top \boldsymbol{y} + \boldsymbol{s}^{(1)\star} = \boldsymbol{c}$, we in turn have $\mathbf{A}^\top \boldsymbol{y} + \boldsymbol{s}^{(\text{new})} = \boldsymbol{c}$.

The first part of Lemma 2.19 shows that $\boldsymbol{x}^{(1)\star}$ is $\eta \ge \frac{r}{96(n+\kappa^2)}$ far from ∂K. The second part shows $\boldsymbol{x}^{(2)\star}$ and $\boldsymbol{x}^{(3)\star}$ are small, so that

$$\left\|\boldsymbol{x}^{(\text{new})} - \boldsymbol{x}^{(1)\star}\right\|_2 = \left\|\boldsymbol{x}^{(2)\star} - \boldsymbol{x}^{(3)\star}\right\|_2 \le \left\|\boldsymbol{x}^{(2)\star}\right\|_1 + \left\|\boldsymbol{x}^{(3)\star}\right\|_1 \le 30(d+\kappa^2) \cdot \frac{LR}{t_{\text{start}}} \cdot R,$$

where we also used $\boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)} \ge \boldsymbol{0}$. Hence, by choice of t_{start}, we have

$$\frac{\left\|\boldsymbol{x}^{(\text{new})} - \boldsymbol{x}^{(1)\star}\right\|_2}{\eta} \le 2^{12}(d+\kappa)^4 \cdot \frac{R}{r} \cdot \frac{LR}{t_{\text{start}}} < 1.$$

In particular, we conclude that $\boldsymbol{x}^{(\text{new})} \in K$.

For the second half of the proof, we bound $\boldsymbol{s}^{(\text{new})}/t_{\text{end}} + \boldsymbol{w}\nabla\phi(\boldsymbol{x}^{(\text{new})})$. The assumption on the potential Φ translates to

$$\begin{aligned} &\gamma_i^{t_{\text{end}}}\left(\boldsymbol{x}^{(1)}, \boldsymbol{s}^{(1)}\right) \le \frac{w_i}{64} \text{ for each } i \in [m], \text{ and} \\ &\boldsymbol{x}_i^{(j)\star} \cdot \boldsymbol{s}_i^{(j)\star} \in \left[1 \pm \frac{1}{64}\right] t_{\text{end}} \text{ for each entry } i \in [n] \text{ and } j \in \{2,3\}. \end{aligned} \tag{2.24}$$

Lemma 2.32 shows that $\nabla^2\phi_i(\boldsymbol{x}_{[i]}^{(1)\star}) \preccurlyeq 9\nu_i^2/\eta^2$. Then,

$$\begin{aligned}
\left\|\boldsymbol{x}_{[i]}^{(\text{new})} - \boldsymbol{x}_{[i]}^{(1)\star}\right\|_{\boldsymbol{x}_{[i]}^{(1)\star}} &\le \frac{3\nu_i}{\eta} \cdot \left\|\boldsymbol{x}_{[i]}^{(\text{new})} - \boldsymbol{x}_{[i]}^{(1)\star}\right\|_2 \\
&\le 3\nu_i \cdot \left(\frac{r}{384(d+\kappa^2)}\right)^{-1} \cdot 30(n+\kappa^2) \cdot \frac{LR}{t_{\text{start}}} \cdot R \\
&\le 2^{14}(d+\kappa)^5 \cdot \frac{LR}{t_{\text{start}}} \cdot \frac{R}{r} \qquad \text{(using the fact } w_i \ge 1) \\
&\le \frac{\delta}{4}.
\end{aligned}$$

Combined with Lemma 2.30, we have $\|\boldsymbol{v}\|_{\boldsymbol{x}_{[i]}^{(\text{new})}} \le (1+\frac{\delta}{2}) \|\boldsymbol{v}\|_{\boldsymbol{x}_{[i]}^{(1)\star}}$ for any $\boldsymbol{v}$, and moreover, $\left\|\nabla\phi_i(\boldsymbol{x}_{[i]}^{(\text{new})}) - \nabla\phi_i(\boldsymbol{x}_{[i]}^{(1)\star})\right\|_{\boldsymbol{x}_{[i]}^{(1)\star}}^* \le \frac{\delta}{2}$. Hence,

$$\begin{aligned}
&\boldsymbol{\gamma}_i^{t_{\text{end}}}\left(\boldsymbol{x}^{(\text{new})}, \boldsymbol{s}^{(\text{new})}\right) \\
&\overset{\text{def}}{=} \left\|\frac{\boldsymbol{s}_{[i]}^{(\text{new})}}{t_{\text{end}}} + w_i\nabla\phi_i(\boldsymbol{x}_{[i]}^{(\text{new})})\right\|_{\boldsymbol{x}_{[i]}^{(\text{new})}}^* \\
&\le \left(1+\frac{\delta}{2}\right)\left\|\frac{\boldsymbol{s}_{[i]}^{(\text{new})}}{t_{\text{end}}} + w_i\nabla\phi_i(\boldsymbol{x}_{[i]}^{(\text{new})})\right\|_{\boldsymbol{x}_{[i]}^{(1)\star}}^* \\
&= \left(1+\frac{\delta}{2}\right)\left\|\frac{\boldsymbol{s}_{[i]}^{(1)\star}}{t_{\text{end}}} + w_i\nabla\phi_i(\boldsymbol{x}_{[i]}^{(1)\star}) + w_i\left(\nabla\phi_i(\boldsymbol{x}_{[i]}^{(\text{new})}) - \nabla\phi_i(\boldsymbol{x}_{[i]}^{(1)\star})\right)\right\|_{\boldsymbol{x}_{[i]}^{(1)\star}}^* \\
&\le \left(1+\frac{\delta}{2}\right)\left\|\frac{\boldsymbol{s}_{[i]}^{(1)\star}}{t_{\text{end}}} + w_i\nabla\phi_i(\boldsymbol{x}_{[i]}^{(1)\star})\right\|_{\boldsymbol{x}_{[i]}^{(1)\star}}^* + \left(1+\frac{\delta}{2}\right)w_i\left\|\nabla\phi_i(\boldsymbol{x}_{[i]}^{(\text{new})}) - \nabla\phi_i(\boldsymbol{x}_{[i]}^{(1)\star})\right\|_{\boldsymbol{x}_{[i]}^{(1)\star}}^* \\
&\le \left\|\frac{\boldsymbol{s}_{[i]}^{(1)\star}}{t_{\text{end}}} + w_i\nabla\phi_i(\boldsymbol{x}_{[i]}^{(1)\star})\right\|_{\boldsymbol{x}_{[i]}^{(1)\star}}^* + \frac{\delta}{2}\cdot\frac{w_i}{64} + \left(1+\frac{\delta}{2}\right)w_i\cdot\frac{\delta}{2} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by the assumption of the theorem)} \\
&\le \left\|\frac{\boldsymbol{s}_{[i]}^{(1)\star}}{t_{\text{end}}} + w_i\nabla\phi_i(\boldsymbol{x}_{[i]}^{(1)\star})\right\|_{\boldsymbol{x}_{[i]}^{(1)\star}}^* + \delta w_i,
\end{aligned}$$

as required. □

Lemma 2.19. *Let η be the the minimum distance between $\boldsymbol{x}^{(1)\star}$ to the boundary of K, i.e. $\eta \overset{\text{def}}{=} \min_{i\in[m]}\min_{\boldsymbol{q}\in\partial K_i}\left\|\boldsymbol{q} - \boldsymbol{x}_{[i]}^{(1)\star}\right\|_2$. We have*

$$\eta \ge \frac{r}{384(d+\kappa^2)} \quad \text{and} \quad \sum_{i=1}^{d}(\boldsymbol{x}_i^{(2)\star} + \boldsymbol{x}_i^{(3)\star}) \le 30(d+\kappa^2)\cdot\frac{LR}{t_{\text{start}}}\cdot R.$$

Proof. This proof is very similar to the proof of Lemma 2.15. Let $\mathcal{P}$ be the set of feasible primal solutions.

Claim 2.20. *$\boldsymbol{x}^\star \stackrel{\text{def}}{=} (\boldsymbol{x}^{(1)\star}, \boldsymbol{x}^{(2)\star}, \boldsymbol{x}^{(3)\star})$ is the minimizer of the function*

$$g(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}) \stackrel{\text{def}}{=} \left\langle \widetilde{\boldsymbol{c}}, \boldsymbol{x}^{(1)} \right\rangle + \left\langle \boldsymbol{c}^{(2)}, \boldsymbol{x}^{(2)} \right\rangle + \left\langle \boldsymbol{c}^{(3)}, \boldsymbol{x}^{(3)} \right\rangle \\ + t_{\text{end}} \cdot \phi(\boldsymbol{x}^{(1)}) - \sum_{i=1}^{n} \boldsymbol{\mu}_i^{(2)} \log \boldsymbol{x}_i^{(2)} - \sum_{i=1}^{n} \boldsymbol{\mu}_i^{(3)} \log \boldsymbol{x}_i^{(3)}$$

over the domain $\mathcal{P}$, where $\widetilde{\boldsymbol{c}} \stackrel{\text{def}}{=} \boldsymbol{c}^{(1)} - \boldsymbol{s}^{(1)\star} - t_{\text{end}} \nabla \phi(\boldsymbol{x}^{(1)\star})$, $\boldsymbol{\mu}^{(2)} \stackrel{\text{def}}{=} \boldsymbol{x}^{(2)\star} \circ \boldsymbol{s}^{(2)\star}$, and $\boldsymbol{\mu}^{(3)} \stackrel{\text{def}}{=} \boldsymbol{x}^{(3)\star} \circ \boldsymbol{s}^{(3)\star}$.

Proof. By definition, we have

$$\begin{aligned} \nabla_{\boldsymbol{x}^{(1)}} g(\boldsymbol{x}^\star) &= \widetilde{\boldsymbol{c}} + t_{\text{end}} \nabla \phi(\boldsymbol{x}^{(1)\star}) = \boldsymbol{c}^{(1)} - \boldsymbol{s}^{(1)\star} = \mathbf{A}^\top \boldsymbol{y}, \\ \nabla_{\boldsymbol{x}^{(2)}} g(\boldsymbol{x}^\star) &= \boldsymbol{c}^{(2)} - \frac{\boldsymbol{\mu}^{(2)}}{\boldsymbol{x}^{(2)\star}} = \boldsymbol{c}^{(2)} - \boldsymbol{s}^{(2)\star} = \mathbf{A}^\top \boldsymbol{y}, \\ \nabla_{\boldsymbol{x}^{(3)}} g(\boldsymbol{x}^\star) &= \boldsymbol{c}^{(3)} - \frac{\boldsymbol{\mu}^{(3)}}{\boldsymbol{x}^{(3)\star}} = \boldsymbol{c}^{(3)} - \boldsymbol{s}^{(3)\star} = -\mathbf{A}^\top \boldsymbol{y}, \end{aligned}$$

for some $\boldsymbol{y}$. Since $\langle \boldsymbol{x}, \nabla g(\boldsymbol{x}^\star) \rangle = \langle \boldsymbol{b}, \boldsymbol{y} \rangle$ is constant for all $\boldsymbol{x}$ in the affine subspace $\mathcal{P}$, we conclude $\nabla g(\boldsymbol{x}^\star)$ is orthogonal $\mathcal{P}$ and therefore $\boldsymbol{x}^\star$ is optimal. $\square$

Next, we consider the directional derivative at $\boldsymbol{x}^\star$ in the direction $\boldsymbol{z} - \boldsymbol{x}^\star$, where $\boldsymbol{z} \in K$ is the point promised by the definition of inner radius. Since our domain is $\mathcal{P} \subset \mathbb{R}^{3n}$, we need to lift $\boldsymbol{z}$ to higher dimension. Define

$$\boldsymbol{z}^{(1)} \stackrel{\text{def}}{=} \boldsymbol{z}, \quad \boldsymbol{z}^{(2)} = \boldsymbol{z}^{(3)} \stackrel{\text{def}}{=} \frac{t_{\text{end}}}{t_{\text{start}}} R.$$

By construction, $\boldsymbol{z} \in \mathcal{P}$. Now, we define the path between $\boldsymbol{x}^\star$ and $\boldsymbol{z}$:

$$p(\beta) \stackrel{\text{def}}{=} (1 - \beta) \cdot (\boldsymbol{x}^{(1)\star}, \boldsymbol{x}^{(2)\star}, \boldsymbol{x}^{(3)\star}) + \beta \cdot (\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \boldsymbol{z}^{(3)}).$$

Since $p(0)$ minimizes g, we know $\frac{d}{d\beta} g(p(\beta))|_{\beta=0} \geq 0$. In particular, we have

$$0 \leq \frac{d}{d\beta} g(p(\beta))|_{\beta=0} = (\widetilde{\boldsymbol{c}} + t_{\text{end}} \nabla \phi(\boldsymbol{x}^{(1)\star}))^\top (\boldsymbol{z}^{(1)} - \boldsymbol{x}^{(1)\star}) \\ + \sum_{i=1}^{n} (\boldsymbol{c}_i^{(2)} - \frac{\boldsymbol{\mu}_i^{(2)}}{\boldsymbol{x}_i^{(2)\star}})(\boldsymbol{z}_i^{(2)} - \boldsymbol{x}_i^{(2)\star}) + \sum_{i=1}^{n} (\boldsymbol{c}_i^{(3)} - \frac{\boldsymbol{\mu}_i^{(3)}}{\boldsymbol{x}_i^{(3)\star}})(\boldsymbol{z}_i^{(3)} - \boldsymbol{x}_i^{(3)\star}). \tag{2.25}$$

Now, we bound the terms one by one. Together they will give the conclusion of the lemma.

Claim 2.21. *We have* $\left(\widetilde{\boldsymbol{c}}+t_{\mathrm{end}}\nabla\phi(\boldsymbol{x}^{(1)\star})\right)^{\top}(\boldsymbol{z}^{(1)}-\boldsymbol{x}^{(1)\star})\leq(6\kappa^{2}-\frac{r}{64\eta})LR.$

Proof. Note that $\widetilde{\boldsymbol{c}}=\boldsymbol{c}^{(1)}-t_{\mathrm{end}}\boldsymbol{\mu}$, where $\boldsymbol{\mu}_{[i]}\overset{\mathrm{def}}{=}\boldsymbol{s}_{[i]}^{(1)\star}/t_{\mathrm{end}}+\nabla\phi(\boldsymbol{x}_{[i]}^{(1)\star})$. By the assumption on $(\boldsymbol{x}^{\star},\boldsymbol{s}^{\star})$, we have

$$\boldsymbol{\gamma}^{t_{\mathrm{end}}}(\boldsymbol{x}^{(1)\star})\overset{\mathrm{def}}{=}\|\boldsymbol{\mu}\|_{\boldsymbol{x}_{[i]}^{(1)\star}}^{*}\leq\frac{w_{i}}{64}\text{ for all }i\in[m].$$

Hence, we have

$$\begin{aligned}&\left(\widetilde{\boldsymbol{c}}+t_{\mathrm{end}}\nabla\phi(\boldsymbol{x}^{(1)\star})\right)^{\top}(\boldsymbol{z}^{(1)}-\boldsymbol{x}^{(1)\star})\\&=\boldsymbol{c}^{(1)\top}(\boldsymbol{z}^{(1)}-\boldsymbol{x}^{(1)\star})-t_{\mathrm{end}}\sum_{i=1}^{m}\boldsymbol{\mu}_{[i]}^{\top}(\boldsymbol{z}_{[i]}^{(1)}-\boldsymbol{x}_{[i]}^{(1)\star})+t_{\mathrm{end}}\sum_{i=1}^{m}w_{i}\nabla\phi_{i}(\boldsymbol{x}^{(1)\star})^{\top}(\boldsymbol{z}_{[i]}^{(1)}-\boldsymbol{x}_{[i]}^{(1)\star})\\&\leq2LR+t_{\mathrm{end}}\sum_{i=1}^{m}\frac{w_{i}}{64}\left\|\boldsymbol{z}_{[i]}^{(1)}-\boldsymbol{x}_{[i]}^{(1)\star}\right\|_{\boldsymbol{x}_{[i]}^{(1)\star}}+t_{\mathrm{end}}\sum_{i=1}^{m}w_{i}\nabla\phi_{i}(\boldsymbol{x}^{(1)\star})^{\top}(\boldsymbol{z}_{[i]}^{(1)}-\boldsymbol{x}_{[i]}^{(1)\star}),\end{aligned}\tag{2.26}$$

where we used $\|\boldsymbol{c}^{(1)}\|_{2}\leq L$ and $\|\boldsymbol{z}^{(1)}-\boldsymbol{x}^{(1)\star}\|_{2}\leq2R$.

The bound on the last two terms are identical to the proof in Lemma 2.15. So we have

$$\begin{aligned}(2.26)&\leq2LR+t_{\mathrm{end}}\sum_{i=1}^{m}w_{i}u_{i}\\&\leq2LR+4t_{\mathrm{end}}\sum_{i=1}^{m}w_{i}\nu_{i}^{2}-\frac{rt_{\mathrm{end}}}{64\eta}\\&\leq2LR+4t_{\mathrm{end}}\kappa^{2}-\frac{rt_{\mathrm{end}}}{64\eta}.\end{aligned}$$

Using $t_{\mathrm{end}}=LR$ and $\kappa\geq1$, we have the claim. □

Next, we bound the second term and the third term in (2.25).

Claim 2.22. *For* $j=2$ *and* 3, *we have*

$$\sum_{i=1}^{d}\left(\boldsymbol{c}_{i}^{(j)}-\frac{\boldsymbol{\mu}_{i}^{(j)}}{\boldsymbol{x}_{i}^{(j)}}\right)(\boldsymbol{z}_{i}^{(j)}-\boldsymbol{x}_{i}^{(j)})\leq3LRd-\frac{t_{\mathrm{start}}}{5R}\sum_{i=1}^{d}\boldsymbol{x}_{i}^{(j)}.$$

Proof. We only prove the case $j=2$; the proof for $j=3$ is similar. As proved in Lemma 2.17, $\|\boldsymbol{x}_{\circ}-\boldsymbol{x}_{c}\|_{2}\leq2R$. Hence $\boldsymbol{c}_{i}^{(2)}=\frac{T}{3R+\boldsymbol{x}_{\circ i}-\boldsymbol{x}_{ci}}\in[\frac{t_{\mathrm{start}}}{5R},\frac{t_{\mathrm{start}}}{R}]$. Hence, we have

$$\begin{aligned}\left(\boldsymbol{c}_{i}^{(2)}-\frac{\boldsymbol{\mu}_{i}^{(2)}}{\boldsymbol{x}_{i}^{(2)}}\right)(\boldsymbol{z}_{i}^{(2)}-\boldsymbol{x}_{i}^{(2)})&=\boldsymbol{c}_{i}^{(2)}\boldsymbol{z}_{i}^{(2)}-\frac{\boldsymbol{\mu}_{i}^{(2)}}{\boldsymbol{x}_{i}^{(2)}}\boldsymbol{z}_{i}^{(2)}-\boldsymbol{c}_{i}^{(2)}\boldsymbol{x}_{i}^{(2)}+\boldsymbol{\mu}_{i}^{(2)}\\&\leq\frac{t_{\mathrm{start}}}{R}\cdot\frac{t_{\mathrm{end}}}{t_{\mathrm{start}}}R-\frac{t_{\mathrm{start}}}{5R}\cdot\boldsymbol{x}_{i}^{(2)}+2t_{\mathrm{end}}\\&\leq3t_{\mathrm{end}}-\frac{t_{\mathrm{start}}}{5R}\cdot\boldsymbol{x}_{i}^{(2)}.\end{aligned}$$

Summing over all i and using $t_{\text{end}} = LR$ gives the result. $\square$

Combining (2.25), Claim 2.21 and Claim 2.22, we have

$$0 \leq (6\kappa^2 - \frac{r}{64\eta})LR + 6LRd - \frac{t_{\text{start}}}{5R} \sum_{i=1}^{d} (\boldsymbol{x}_i^{(2)\star} + \boldsymbol{x}_i^{(3)\star}).$$

Rearranging, we conclude

$$\frac{r}{64\eta} + \frac{t_{\text{start}}}{5LR^2} \sum_{i=1}^{n} (\boldsymbol{x}_i^{(2)\star} + \boldsymbol{x}_i^{(3)\star}) \leq 6(d + \kappa^2),$$

as required.

$\square$

2.5 Main result statement

We are ready to combine all previous theorems for the overall IPM result.

Theorem 2.23. *Suppose we are given the convex program* (P). *For any* $0 < \varepsilon \leq 1/2$*, there is an algorithm that outputs an approximate solution* $\boldsymbol{x}$ *using* $T \stackrel{\text{def}}{=} O(\sqrt{\kappa} \log(m) \log(\frac{d\kappa R}{\varepsilon r}))$ *iterations of* PathFollowingRobust *(Algorithm 4), such that* $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$*,* $\boldsymbol{x} \in K$ *and*

$$\boldsymbol{c}^\top \boldsymbol{x} \leq \min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b},\, \boldsymbol{x} \in K} \boldsymbol{c}^\top \boldsymbol{x} + \varepsilon LR.$$

Proof. The algorithm consists of running PathFollowingRobust twice. The first run is on the problem (P_T') from $t_{\text{start}} = 2^{16}(d+\kappa)^5 \cdot \frac{LR}{\delta} \cdot \frac{R}{r}$ to $t_{\text{end}} = LR$ with $\delta = \frac{1}{128}$, with initial point given by Lemma 2.17. Since the initial point is known to be on the central path, its potential is indeed bounded, hence the potential is at most $\cosh(\frac{\lambda}{64})$ throughout the first run by Theorem 2.4. Let $\boldsymbol{x}^\star, \boldsymbol{s}^\star$ denote the output. Theorem 2.18 shows that we have $\boldsymbol{x}^{(\text{init})}, \boldsymbol{s}^{(\text{init})}$ at $t = LR$ feasible for (P), satisfying

$$\boldsymbol{\gamma}_i^t(\boldsymbol{x}^{(\text{init})}, \boldsymbol{s}^{(\text{init})}) \leq \boldsymbol{\gamma}_i^t(\boldsymbol{x}^{(1)\star}, \boldsymbol{s}^{(1)\star}) + \frac{w_i}{128}$$

for each $i \in [m]$. It follows that

$$\begin{aligned}
\Phi^t(\boldsymbol{x}^{(\text{init})}, \boldsymbol{s}^{(\text{init})}) &\overset{\text{def}}{=} \sum_i \cosh\left(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i^t(\boldsymbol{x}^{(\text{init})}, \boldsymbol{s}^{(\text{init})})\right) \\
&\le \sum_i \cosh\left(\frac{\lambda}{w_i}\boldsymbol{\gamma}_i^t(\boldsymbol{x}^{(1)\star}, \boldsymbol{s}^{(1)\star}) + \frac{\lambda}{128}\right) \\
&\le (1+\frac{\lambda}{64})\Phi(\boldsymbol{x}^\star, \boldsymbol{s}^\star) && \text{(by Lemma 2.38)} \\
&\le (1+\frac{\lambda}{64})\cosh(\frac{\lambda}{64}) \\
&\le \cosh(\frac{\lambda}{32})
\end{aligned}$$

by choice of λ. The second run of PathFollowingRobust is on (P) from $t_{\text{start}} = LR$ to $t_{\text{end}} = \frac{\varepsilon LR}{3\kappa}$. Let $\boldsymbol{x}^{(\text{final})}, \boldsymbol{s}^{(\text{final})}$ denote the output of the second run. Theorem 2.14 shows that the solution is feasible and the potential Φ is bounded by $\cosh(\frac{\lambda}{32})$ throughout. Hence, for each $i \in [m]$, we have $\boldsymbol{\gamma}_i^{t_{\text{end}}}(\boldsymbol{x}^{(\text{final})}, \boldsymbol{s}^{(\text{final})}) \le \frac{w_i}{32}$. By Lemma 2.3, we conclude

$$\boldsymbol{c}^\top \boldsymbol{x}^{(\text{final})} \le \min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b},\, \boldsymbol{x}\in K} \boldsymbol{c}^\top \boldsymbol{x} + 3t\kappa LR \le \min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b},\, \boldsymbol{x}\in K} \boldsymbol{c}^\top \boldsymbol{x} + \varepsilon LR.$$

Now we compute the total number of steps T to reduce t from $2^{16}(d+\kappa)^5 \cdot \frac{LR}{\delta} \cdot \frac{R}{r}$ to $\frac{\varepsilon LR}{3\kappa}$ over the two runs. Recall each iteration of the algorithm reduces t by a factor of $(1-h)$ where $h \overset{\text{def}}{=} \frac{\alpha}{64\sqrt{\kappa}}$. Hence,

$$T = O\left(\frac{1}{h}\log\left(\frac{t_{\text{start}}}{t_{\text{end}}}\right)\right) = O\left(\sqrt{\kappa}\log m \log\left(\frac{d\kappa R}{\varepsilon r}\right)\right),$$

as required. □

2.6 Path following for linear programs

For completeness, we simplify PathFollowingRobust (Algorithm 4) to apply to linear programs of the form

$$\begin{aligned}
\min \quad & \boldsymbol{c}^\top \boldsymbol{x} \\
\text{s.t.} \quad & \mathbf{A}\boldsymbol{x} = \boldsymbol{b} \qquad\qquad \text{(LP)} \\
& \boldsymbol{l} \le \boldsymbol{x} \le \boldsymbol{u}
\end{aligned}$$

where $\mathbf{A}$ is a $n \times m$ matrix.

Theorem 2.24. *Consider* (LP). *We are given a scalar $r > 0$ such that* there exists *some interior point $\boldsymbol{x}_\circ$ satisfying $\mathbf{A}\boldsymbol{x}_\circ = \boldsymbol{b}$ and $\boldsymbol{l} + r \le \boldsymbol{x}_\circ \le \boldsymbol{u} - r$. Let $L = \|\boldsymbol{c}\|_2$ and $R = \|\boldsymbol{u} - \boldsymbol{l}\|_2$.*

For any $0 < \varepsilon \leq 1/2$, the algorithm PathFollowingLP *(Algorithm 5) finds $\boldsymbol{x}$ such that $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$, $\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$ and*

$$\boldsymbol{c}^\top \boldsymbol{x} \leq \min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b},\, \boldsymbol{l}\leq\boldsymbol{x}\leq\boldsymbol{u}} \boldsymbol{c}^\top \boldsymbol{x} + \varepsilon LR.$$

Furthermore, the algorithm has the following properties:

- *Each call of* PathFollowingLP *involves $O(\sqrt{m}\log m \log(\frac{mR}{\epsilon r}))$-many steps, and $\overline{t}$ is only updated $O(\log m \log(\frac{mR}{\epsilon r}))$-many times.*
- *In each step of* PathFollowingLP*, the coordinate i in $\mathbf{W}$ and $\boldsymbol{v}$ changes only if $\overline{\boldsymbol{x}}_i$ or $\overline{\boldsymbol{s}}_i$ changes.*
- *In each step of* PathFollowingLP*, $\|\boldsymbol{v}\|_2 = O(\frac{1}{\log m})$.*

Proof. We set $w_i = \nu_i = 1$ for all i in Algorithm 4. Each block $[i]$ now references the single coordinate i. From Eq. (2.8) and Definition 2.2, we have

$$\begin{aligned}
\boldsymbol{\delta}_{\boldsymbol{\mu},i} &\stackrel{\text{def}}{=} -\alpha \cdot \boldsymbol{k}_i^{\overline{t}}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}) \cdot \boldsymbol{\mu}_i^{\overline{t}}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}) \\
&= -\alpha \cdot \frac{\sinh(\lambda \boldsymbol{\gamma}_i^{\overline{t}}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}))}{\sqrt{\sum_{j=1}^m \cosh^2(\lambda \boldsymbol{\gamma}_j^{\overline{t}}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}))}} \cdot \frac{\boldsymbol{\mu}_i^{\overline{t}}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}})}{\boldsymbol{\gamma}_i^{\overline{t}}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}})} \\
&= -\alpha \cdot \frac{\sinh(\lambda \boldsymbol{\gamma}_i^{\overline{t}}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}))}{\|\cosh(\lambda \boldsymbol{\gamma}^{\overline{t}}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}))\|_2} \cdot \mathbf{H}_{\overline{\boldsymbol{x}},i}^{1/2}.
\end{aligned}$$

Next, recall the requirements of $\boldsymbol{\delta}_{\boldsymbol{x}}$ and $\boldsymbol{\delta}_{\boldsymbol{s}}$ in Algorithm 4, which are $\mathbf{A}\boldsymbol{\delta}_{\boldsymbol{x}} = \mathbf{0}$, $\boldsymbol{\delta}_{\boldsymbol{s}} \in \text{Range}(\mathbf{A}^\top)$, and

$$\begin{aligned}
\left\|\mathbf{H}_{\overline{\boldsymbol{x}}}^{1/2}\left(\boldsymbol{\delta}_{\boldsymbol{x}} - \mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2}(\mathbf{I} - \mathbf{P}_{\overline{\boldsymbol{x}}})\mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2}\boldsymbol{\delta}_{\boldsymbol{\mu}}\right)\right\|_2 &\leq \overline{\varepsilon}\alpha, \\
\left\|\mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2}\left(\boldsymbol{\delta}_{\boldsymbol{s}} - \overline{t}\mathbf{H}_{\overline{\boldsymbol{x}}}^{1/2}\mathbf{P}_{\overline{\boldsymbol{x}}}\mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2}\boldsymbol{\delta}_{\boldsymbol{\mu}}\right)\right\|_2 &\leq \overline{\varepsilon}\alpha\overline{t}.
\end{aligned}$$

Let $\boldsymbol{v} \stackrel{\text{def}}{=} \mathbf{H}_{\overline{\boldsymbol{x}}}^{-1/2}\boldsymbol{\delta}_{\boldsymbol{\mu}}$. For reasons that will become clear in subsequent chapters, let $\mathbf{W} \stackrel{\text{def}}{=} \mathbf{H}_{\overline{\boldsymbol{x}}}^{-1}$. Then, it is equivalent to find $\boldsymbol{v}^\perp$ and $\boldsymbol{v}^\parallel$ satisfying $\mathbf{A}\mathbf{W}^{1/2}\boldsymbol{v}^\perp = \mathbf{0}$, $\mathbf{W}^{-1/2}\boldsymbol{v}^\parallel \in \text{Range}(\mathbf{A}^\top)$, and

$$\begin{aligned}
\left\|\boldsymbol{v}^\perp - (\mathbf{I} - \mathbf{P}_{\overline{\boldsymbol{x}}})\boldsymbol{v}\right\|_2 &\leq \overline{\varepsilon}\alpha, \\
\left\|\boldsymbol{v}^\parallel - \mathbf{P}_{\overline{\boldsymbol{x}}}\boldsymbol{v}\right\|_2 &\leq \overline{\varepsilon}\alpha;
\end{aligned} \tag{2.27}$$

then we can set $\boldsymbol{\delta}_{\boldsymbol{x}} = \mathbf{W}^{1/2}\boldsymbol{v}^\perp$ and $\boldsymbol{\delta}_{\boldsymbol{s}} = \overline{t}\mathbf{W}^{-1/2}\boldsymbol{v}^\parallel$ for the correct updates.

The bound on $\|\boldsymbol{v}\|_2$ follows from

$$\|\boldsymbol{v}\|_2 \leq \alpha \frac{\|\sinh(\lambda \boldsymbol{\gamma}^{\overline{t}}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}))\|_2}{\|\cosh(\lambda \boldsymbol{\gamma}^{\overline{t}}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}))\|_2} \leq \alpha = O\left(\frac{1}{\log m}\right).$$

As a result, we can relax the right-hand side of (2.27) to $\overline{\varepsilon}\left\|\boldsymbol{v}\right\|_2$. $\square$

Algorithm 5 PathFollowingRobust applied to linear programs

Definition of parameters:

$$\begin{aligned} \lambda &\overset{\text{def}}{=} 64\log(256m^2), \quad \overline{\varepsilon} \overset{\text{def}}{=} \frac{1}{1440\lambda}, \quad \alpha \overset{\text{def}}{=} \frac{\overline{\varepsilon}}{2}, \quad \varepsilon_t \overset{\text{def}}{=} \frac{\overline{\varepsilon}}{8}, \\ h &\overset{\text{def}}{=} \frac{\alpha}{64\sqrt{m}}, \\ \mathbf{W} &\overset{\text{def}}{=} \mathbf{H}_{\overline{\boldsymbol{x}}}^{-1}, \\ \boldsymbol{v}_i &\overset{\text{def}}{=} -\alpha \cdot \frac{\sinh(\lambda \boldsymbol{\gamma}_i^{\overline{t}}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}))}{\|\cosh(\lambda \boldsymbol{\gamma}_i^{\overline{t}}(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}))\|_2} \text{ for each } i \in [m] \end{aligned}$$

```
1: procedure PathFollowingLP($\mathbf{A}, \phi, \boldsymbol{w}, \boldsymbol{x}^{(\text{init})}, \boldsymbol{s}^{(\text{init})}, t_{\text{start}}, t_{\text{end}}$)
2:     $(\boldsymbol{x}, \boldsymbol{s}, t) \leftarrow (\boldsymbol{x}^{(\text{init})}, \boldsymbol{s}^{(\text{init})}, t_{\text{start}})$
3:     $(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}, \overline{t}) \leftarrow (\boldsymbol{x}, \boldsymbol{s}, t)$
4:     while $t \geq t_{\text{end}}$ do
5:         Update diagonal matrix $\mathbf{W}$ computed at $\overline{\boldsymbol{x}}$
6:         Update the direction $\boldsymbol{v}$
7:         Pick $\boldsymbol{v}^{\perp}$ and $\boldsymbol{v}^{\|}$ such that $\mathbf{A}\boldsymbol{v}^{\perp} = \mathbf{0}$, $\boldsymbol{v}^{\|} \in \text{Range}(\mathbf{A}^{\top})$, and
               $\|\boldsymbol{v}^{\perp} - (\mathbf{I} - \mathbf{P}_{\overline{\boldsymbol{x}}})\boldsymbol{v}\|_2 \leq \overline{\varepsilon}\|\boldsymbol{v}\|_2,$
               $\|\boldsymbol{v}^{\|} - \mathbf{P}_{\overline{\boldsymbol{x}}}\boldsymbol{v}\|_2 \leq \overline{\varepsilon}\|\boldsymbol{v}\|_2.$
                                        ▷ $\mathbf{P}_{\overline{\boldsymbol{x}}} \overset{\text{def}}{=} \mathbf{W}^{1/2}\mathbf{A}^{\top}(\mathbf{A}\mathbf{W}\mathbf{A}^{\top})^{-1}\mathbf{A}\mathbf{W}^{1/2}$
8:         $\boldsymbol{f} \leftarrow \boldsymbol{f} + \mathbf{W}^{1/2}\boldsymbol{v}^{\perp}$, $\boldsymbol{s} \leftarrow \boldsymbol{s} + \overline{t}\mathbf{W}^{-1/2}\boldsymbol{v}^{\|}$
9:         update $\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}$ to satisfy $\overline{\boldsymbol{x}} \in (\boldsymbol{l}, \boldsymbol{u}), \overline{\boldsymbol{s}} > \mathbf{0}$, and
               $\|\mathbf{W}^{-1/2}(\overline{\boldsymbol{x}} - \boldsymbol{x})\|_{\infty} \leq \overline{\varepsilon},$
               $\|\mathbf{W}^{1/2}(\overline{\boldsymbol{s}} - \boldsymbol{s})\|_{\infty} \leq \overline{t}\overline{\varepsilon}$
10:        $t \leftarrow (1 - h) \cdot t$
11:        update $\overline{t}$ to satisfy $|\overline{t} - t| \leq \varepsilon_t \cdot \overline{t}$
12:    end while
13:    return $(\boldsymbol{x}, \boldsymbol{s})$
14: end procedure
```

2.7 One scheme for solution approximations

As discussed in Eq. (2.10), every iteration of our path following algorithm uses approximate solutions $\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}$ to compute the subsequent Newton step. In this section, we present one scheme for computing these approximations. At a high level, the scheme begins with $\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}$ being equal to the exact solution $\boldsymbol{x}, \boldsymbol{s}$, and as $\boldsymbol{x}, \boldsymbol{s}$ is updated, $\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}$ is lazily updated on individual blocks to the latest exact values when the approximation guarantee is violated. Lazy updating is crucial for the implementation of our efficient data structures in later chapters.

Recall the approximation guarantees as specified by PathFollowingRobust(Algorithm 4): At every iteration, $\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}$ satisfy, for each $i \in [m]$,

$$\begin{aligned} \left\| \mathbf{H}_{\overline{\boldsymbol{x}}_{[i]}} (\boldsymbol{x}_{[i]} - \overline{\boldsymbol{x}}_{[i]}) \right\|_2 &\leq \overline{\varepsilon} \\ \left\| \mathbf{H}_{\overline{\boldsymbol{x}}_{[i]}}^{-1} (\boldsymbol{s}_{[i]} - \overline{\boldsymbol{s}}_{[i]}) \right\|_2 &\leq \overline{t} \overline{\varepsilon} w_i. \end{aligned}$$

For simplicity, we assume all blocks are one-dimensional in this section; these results are generalizable back to blocks in a straightforward way. Our task is then the following: Let $\boldsymbol{x}$ be the solution throughout PathFollowingRobust. We want to determine $\overline{\boldsymbol{x}}$ at every iteration, so that

$$\|\mathbf{D}(\overline{\boldsymbol{x}} - \boldsymbol{x})\|_\infty \leq \delta$$

is always satisfied at the end of an iteration, where δ is a constant error tolerance, and $\mathbf{D}$ is a diagonal scaling matrix that is a fixed entrywise function of $\overline{\boldsymbol{x}}$. We omit the problem for dual variables, which is analogous.

We refer to the k-th iteration of one run of PathFollowingRobust(Algorithm 4) as the k-th step. The values of $\boldsymbol{x}, \overline{\boldsymbol{x}}, \mathbf{D}$ at the end of the step are denoted by $\boldsymbol{x}^{(k)}, \overline{\boldsymbol{x}}^{(k)}, \mathbf{D}^{(k)}$ respectively.

In our scheme, we make clever use of dyadic intervals. At step k, for each ℓ such that $k \equiv 0 \bmod 2^\ell$, we find the index set $I_\ell^{(k)}$ that contains all coordinates i of $\boldsymbol{x}$ such that $\boldsymbol{x}_i^{(k)}$ changed significantly compared to $\boldsymbol{x}_i^{(k-2^\ell)}$, that is, compared to 2^ℓ steps ago. Formally:

Definition 2.25. At step k of the IPM, for each ℓ such that $k \equiv 0 \bmod 2^\ell$, we define

$$\begin{aligned} I_\ell^{(k)} \overset{\text{def}}{=} \{ i \in [m] : \mathbf{D}_{ii}^{(k-1)} \cdot |\boldsymbol{x}_i^{(k)} - \boldsymbol{x}_i^{(k-2^\ell)}| \geq \frac{\delta}{2 \lceil \log m \rceil}, \\ \text{and } \overline{\boldsymbol{x}}_i \text{ has not been updated after the } (k-2^\ell)\text{-th step} \}. \end{aligned}$$

The main theorem for this section shows that our lazy-updating scheme is correct, and that the problem is reduced to finding heavy hitters in $\boldsymbol{x}$.

Theorem 2.26 (Approximation scheme)**.** *Suppose* FindLargeCoordinates(ℓ) *is a procedure that correctly computes the set $I_\ell^{(k)}$ at the k-th step. Then* AbstractSolutionApproximation *(Algorithm 6) maintains an approximation $\overline{\boldsymbol{x}}$ of $\boldsymbol{x}$ throughout* PathFollowingLP *where $\mathbf{D}$ is a diagonal scaling matrix that is a fixed entrywise function of $\overline{\boldsymbol{x}}$.*

- Step($\boldsymbol{x}^{(\text{new})}$)*: Increment the step counter and update $\overline{\boldsymbol{x}}, \mathbf{D}$ such that $\|\mathbf{D}(\boldsymbol{x} - \overline{\boldsymbol{x}})\|_\infty \leq \delta$ using all the latest values.*

Suppose $\|\mathbf{D}^{(k-1)}(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)})\|_2 \leq \beta$ for all steps k. *Then, $O(2^{2\ell}(\beta/\delta)^2 \log^2 m)$ coordinates of $\overline{\boldsymbol{x}}$ are updated every 2^ℓ steps for each $\ell \geq 0$.*

Algorithm 6 AbstractSolutionApproximation

```
1:                                   ▷ Assume $\mathbf{D}$ is a fixed function of $\overline{\boldsymbol{x}}$ entrywise
2:     $\delta > 0$: additive approximation error
3:     $k$: current step
4:     $\overline{\boldsymbol{x}} \in \mathbb{R}^m$: current valid approximate vector
5:     $\{\boldsymbol{x}^{(j)} \in \mathbb{R}^m\}_{j=0}^k$: list of previous inputs
6:     $\{\mathbf{D}^{(j)} \in \mathbb{R}^{m\times m}\}_{j=0}^k$: list of previous diagonal scaling matrices

7:  procedure Step($\boldsymbol{x}^{(\text{new})}$)
8:      Increment step counter $k$
9:      $I \leftarrow \emptyset$
10:     for all $0 \leq \ell < \lceil \log m \rceil$ such that $k \equiv 0 \bmod 2^\ell$ do
11:         $I_\ell^{(k)} \leftarrow$ FindLargeCoordinates($\ell$)
12:         $I \leftarrow I \cup I_\ell^{(k)}$
13:     end for
14:     if $k \equiv 0 \bmod 2^{\lceil \log m \rceil}$ then
15:         $I \leftarrow [m]$                  ▷ Update $\overline{\boldsymbol{x}}$ in full every $2^{\lceil \log m \rceil}$ steps
16:     end if
17:     $\overline{\boldsymbol{x}}_i \leftarrow \boldsymbol{x}_i^{(k)}$ for all $i \in I$
18:     $\boldsymbol{x}^{(k)} \leftarrow \boldsymbol{x}^{(\text{new})}$
19:     $\mathbf{D}^{(k)} \leftarrow \mathbf{D}^{(k-1)}$, then update $\mathbf{D}_{ii}^{(k)}$ as a function of $\overline{\boldsymbol{x}}_i$ for all $i \in I$
20: end procedure
```

Proof. First we prove the correctness of Step. Fix some step k. Suppose for coordinate i, the latest update to $\overline{\boldsymbol{x}}_i$ is $\overline{\boldsymbol{x}}_i \leftarrow \boldsymbol{x}_i^{(k')}$ at step k'. So $\mathbf{D}_{ii}^{(k'')}$ is the same for all $k'' \in (k', k]$, and i is not in the set $I_\ell^{(k'')}$ returned by FindLargeCoordinates for all $k'' \in (k', k]$. Since we set $\overline{\boldsymbol{x}} \leftarrow \boldsymbol{x}$ every $2^{\lceil \log m \rceil}$ steps, we know $k - 2^{\lceil \log m \rceil} \leq k' < k$. Using dyadic intervals, we can define a sequence $k_0, k_1, \ldots, k_s$ with $s \leq 2\lceil \log m \rceil$, where $k' = k_0 < k_1 < k_2 < \cdots < k_s = k$, each $k_{j+1} - k_j$ is a power of 2, and $(k_{j+1} - k_j)$ is increasing in j. Hence, we have

$$\boldsymbol{x}_i^{(k)} - \overline{\boldsymbol{x}}_i^{(k)} = \boldsymbol{x}_i^{(k_s)} - \overline{\boldsymbol{x}}_i^{(k_0)} = \boldsymbol{x}_i^{(k_s)} - \boldsymbol{x}_i^{(k_0)} = \sum_{j=0}^{s-1} \left(\boldsymbol{x}_i^{(k_{j+1})} - \boldsymbol{x}_i^{(k_j)} \right).$$

Since i was not in $I_\ell^{(k)}$ returned by FindLargeCoordinates for all k_j with $0 \leq j < s$, we have

$$\frac{\delta}{2\lceil \log m \rceil} \geq \mathbf{D}_{ii}^{(k_{j+1}-1)} \cdot |\boldsymbol{x}_i^{(k_{j+1})} - \boldsymbol{x}_i^{(k_j)}|.$$

Since D_{ii} has not changed since iteration k', summing over all $j = 0, 1, \ldots, s-1$ and simplifying gives

$$\mathbf{D}_{ii}^{(k)} \cdot |\boldsymbol{x}_i^{(k)} - \overline{\boldsymbol{x}}_i^{(k)}| \leq \delta.$$

Hence, we have $\|\mathbf{D}(\boldsymbol{x} - \overline{\boldsymbol{x}})\|_\infty \leq \delta$.

To prove the number of coordinate changes, again fix step k, and fix some ℓ with $k \equiv 0 \bmod 2^\ell$. We bound the number of coordinates in $I_\ell^{(k)}$. For any $i \in I_\ell^{(k)}$, we know $\mathbf{D}_{ii}^{(j)} = \mathbf{D}_{ii}^{(k)}$ for all $j > k - 2^\ell$ because $\overline{\boldsymbol{x}}_i$ did not change in the meanwhile. By definition of $I_\ell^{(k)}$, we have

$$\mathbf{D}_{ii}^{(k)} \cdot \sum_{j=k-2^\ell}^{k-1} |\boldsymbol{x}_i^{(j+1)} - \boldsymbol{x}_i^{(j)}| \geq \mathbf{D}_{ii}^{(k)} \cdot |\boldsymbol{x}_i^{(k)} - \boldsymbol{x}_i^{(k-2^\ell)}| \geq \frac{\delta}{2\lceil \log m \rceil}.$$

Using $\mathbf{D}_{ii}^{(j)} = \mathbf{D}_{ii}^{(k)}$ for all $j > k - 2^\ell$ again, the above inequality yields

$$\begin{aligned}\frac{\delta}{2\lceil \log m \rceil} &\leq \sum_{j=k-2^\ell}^{k-1} \mathbf{D}_{ii}^{(j)} |\boldsymbol{x}_i^{(j+1)} - \boldsymbol{x}_i^{(j)}| \\ &\leq \sqrt{2^\ell \sum_{j=k-2^\ell}^{k-1} {\mathbf{D}_{ii}^{(j)}}^2 |\boldsymbol{x}_i^{(j+1)} - \boldsymbol{x}_i^{(j)}|^2}. \qquad \text{(by Cauchy-Schwarz)}\end{aligned}$$

Squaring and summing over all $i \in I_\ell^{(k)}$ gives

$$\begin{aligned}\Omega\left(\frac{2^{-\ell}\delta^2}{\log^2 m}\right) |I_\ell^{(k)}| &\leq \sum_{i \in I_\ell^{(k)}} \sum_{j=k-2^\ell}^{k-1} {\mathbf{D}_{ii}^{(j)}}^2 |\boldsymbol{x}_i^{(j+1)} - \boldsymbol{x}_i^{(j)}|^2 \\ &\leq \sum_{i=1}^{m} \sum_{j=k-2^\ell}^{k-1} {\mathbf{D}_{ii}^{(j)}}^2 |\boldsymbol{x}_i^{(j+1)} - \boldsymbol{x}_i^{(j)}|^2 \\ &\leq 2^\ell \beta^2,\end{aligned}$$

where we use $\|\mathbf{D}^{(j)}(\boldsymbol{x}^{(j+1)} - \boldsymbol{x}^{(j)})\|_2 \leq \beta$ at the end. Hence, we have

$$|I_\ell^{(k)}| = O(2^{2\ell}(\beta/\delta)^2 \log^2 m).$$

In other words, for each $\ell \geq 0$, we update $|I_\ell^{(k)}|$-many coordinates of $\overline{\boldsymbol{x}}$ at step k when $k \equiv 0 \mod 2^\ell$. So we conclude that for each $\ell \geq 0$, we update $O(2^{2\ell}(\beta/\delta)^2 \log^2 m)$-many coordinates of $\overline{\boldsymbol{x}}$ every 2^ℓ steps. $\square$

Appendix

2.A Self-concordant barrier functions

Here, we provide some relevant definitions and properties of self-concordant barrier functions used in our IPM proofs.

Definition 2.27 ([133]). A function ϕ is a *ν-self-concordant barrier* for a non-empty open convex set K if dom $\phi = K$, $\phi(x) \to +\infty$ as $x \to \partial K$, and for any $\boldsymbol{x} \in K$ and $\boldsymbol{u} \in \mathbb{R}^n$,

$$D^3\phi(x)[\boldsymbol{u}, \boldsymbol{u}, \boldsymbol{u}] \leq 2 \|\boldsymbol{u}\|_{\nabla^2\phi(\boldsymbol{x})} \text{ and } \|\nabla\phi(\boldsymbol{x})\|_{(\nabla^2\phi(\boldsymbol{x}))^{-1}} \leq \sqrt{\nu}.$$

A function ϕ is a self-concordant barrier if the first inequality holds.

For many convex sets, we have an explicit barrier with $\nu = O(n)$. For the case of linear programs, the convex set K_i is simply the feasible interval $[\ell_i, u_i]$ for each variable $\boldsymbol{x}_i$, and one can use the log barrier $\phi(x) = -\log(u_i - x) - \log(x - \ell_i)$ with self-concordance 1.

We sometimes use the fact that $\nu \geq 1$ to simplify formulas.

Lemma 2.28 ([133, Corollary 4.3.1]). *The self-concordance ν is at least 1 for any barrier function.*

The following lemma about self-concordance implies when the input $\boldsymbol{x}$ is not changing rapidly, then the Hessian is also well-approximated.

Lemma 2.29 ([133, Theorems 5.3.7]). *Let ϕ be a ν-self-concordant barrier. For any $\boldsymbol{x}, \boldsymbol{y} \in$ dom ϕ, we have*

$$\langle \nabla\phi(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} \rangle \leq \nu, \text{ and}$$
$$\langle \nabla\phi(\boldsymbol{y}) - \nabla\phi(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} \rangle \geq \frac{\|\boldsymbol{y} - \boldsymbol{x}\|_{\boldsymbol{x}}^2}{1 + \|\boldsymbol{y} - \boldsymbol{x}\|_{\boldsymbol{x}}}.$$

Lemma 2.30 ([133, Theorem 5.3.4]). *Let ϕ be a self-concordant barrier. For any $\boldsymbol{x} \in$ dom ϕ and any $\boldsymbol{y} \in$ dom ϕ such that $\|\boldsymbol{y} - \boldsymbol{x}\|_{\boldsymbol{x}} < 1$, we have*

$$(1 - \|\boldsymbol{y} - \boldsymbol{x}\|_{\boldsymbol{x}})^2 \nabla^2\phi(\boldsymbol{x}) \preccurlyeq \nabla^2\phi(\boldsymbol{y}) \preccurlyeq \frac{1}{(1 - \|\boldsymbol{y} - \boldsymbol{x}\|_{\boldsymbol{x}})^2} \nabla^2\phi(\boldsymbol{x}).$$

Lemma 2.31 ([133, Theorems 5.3.8]). *Let ϕ be a ν-self-concordant barrier. For any $\boldsymbol{x}, \boldsymbol{y} \in$ dom ϕ, with $\nabla\phi(\boldsymbol{x})^\top(\boldsymbol{y}-\boldsymbol{x}) \geq \boldsymbol{0}$, then $\|\boldsymbol{y}-\boldsymbol{x}\|_{\boldsymbol{x}} \leq \nu + 2\sqrt{\nu}$. In particular, for $\boldsymbol{x}^* = \operatorname{argmin}_{\boldsymbol{x}} \phi(\boldsymbol{x})$, we have*

$$\{\boldsymbol{x} : \|\boldsymbol{x}-\boldsymbol{x}^*\|_{\boldsymbol{x}^*} \leq 1\} \subset \operatorname{dom}\phi \subset \{\boldsymbol{x} : \|\boldsymbol{x}-\boldsymbol{x}^*\|_{\boldsymbol{x}^*} \leq \nu + 2\sqrt{\nu}\}.$$

Lemma 2.32. *Suppose ϕ is a ν-self-concordant for the interval $[\alpha, \beta]$. For any $x, z \in (\alpha, \beta)$, we have that*

$$\sqrt{\phi''(x)} \leq \frac{3\nu}{\min\{x-\alpha, \beta-x\}}$$

and

$$\phi'(x)(z-x) + \frac{1}{16}\sqrt{\phi''(x)}|z-x| \leq 4\nu^2 - \frac{1}{16}\max\left\{\frac{z-\alpha}{x-\alpha}, \frac{\beta-z}{\beta-x}\right\}.$$

Proof. For the first inequality, we bound ϕ'' in two cases: If $\phi'(x) \geq 0$, then $\phi'(x)(x-\alpha) \geq 0$ and Lemma 2.31 shows that $|\alpha - x|\sqrt{\phi''(x)} \leq \nu + 2\sqrt{\nu} \leq 3\nu$. Hence, we have $\sqrt{\phi''(x)} \leq \frac{3\nu}{x-\alpha}$. If $\phi'(x) \leq 0$, a similar argument shows that $\sqrt{\phi''(x)} \leq \frac{3\nu}{\beta-x}$.

For the second inequality, we split into four cases. First, we note that both sides of the inequality are invariant under affine transformation. Hence, we can assume $\alpha = 0$ and $\beta = 1$.

Case 1: $\phi'(x)(z-x) \geq 0$. Lemma 2.31 shows that

$$\sqrt{\phi''(x)}|z-x| \leq \nu + 2\sqrt{\nu} \leq 3\nu.$$

Together with the fact that $|\phi'(x)| \leq \sqrt{\nu\phi''(x)}$, we have

$$\phi'(x)(z-x) + \frac{1}{16}\sqrt{\phi''(x)}|z-x| \leq 2\nu^2.$$

Case 2: $x \in [\frac{1}{12\nu}, 1 - \frac{1}{12\nu}]$.

Since $\phi'(x)(z-x) \leq 0$ and $z, x \in [0,1]$, we have

$$\phi'(x)(z-x) + \frac{1}{16}\sqrt{\phi''(x)}|z-x| \leq \frac{1}{16}\sqrt{\phi''(x)} \leq \frac{1}{16}\cdot 36\nu^2 = 3\nu^2,$$

where we used the first result at the end.

Case 3: $x \leq \frac{1}{12\nu}$.

Let $x^* = \arg\min_{x\in[0,1]} \phi(x)$. Lemma 2.31 shows that there is an interval $I = [-\gamma, \gamma]$ such that

$$x^* + I \subset [0,1] \subset x^* + (\nu + 2\sqrt{\nu})I \subset x^* + 3\nu I.$$

In particular, this implies that $x^* \in [\frac{1}{6\nu}, 1-\frac{1}{6\nu}]$. Since $x \leq \frac{1}{12\nu}$, we have that $x \leq x^* - x$.

Now we use this to show $\phi'(x) \leq -\frac{1}{8}\sqrt{\phi''(x)}$. Note that

$$\phi'(x) = \phi'(x^*) - \int_x^{x^*} \phi''(t)dt = -\int_x^{x^*} \phi''(t)dt.$$

Lemma 2.30 shows that $\left[x - \frac{1}{\sqrt{\phi''(x)}}, x + \frac{1}{\sqrt{\phi''(x)}}\right] \subset \mathrm{dom}\,\phi$. In particular, this implies that

$$\frac{1}{\sqrt{\phi''(x)}} \leq x \leq x^* - x, \tag{2.28}$$

and hence $x^* \geq x + \frac{1}{\sqrt{\phi''(x)}}$. Hence, we have

$$\begin{aligned}\phi'(x) &\leq -\int_x^{x+(\phi''(x))^{-1/2}/2} \phi''(t)dt \\ &\leq -\frac{1}{4}\phi''(x) \cdot \frac{(\phi''(x))^{-1/2}}{2} \\ &= -\frac{1}{8}\sqrt{\phi''(x)},\end{aligned}$$

where we used $\phi''(t) \geq \frac{1}{4}\phi''(x)$ for all $|t-x| \leq \frac{1}{2\sqrt{\phi''(x)}}$ as given by Lemma 2.30.

Since $\phi'(x)(z-x) \leq 0$ and $\phi'(x) \leq 0$, we have $z \geq x$ and

$$\begin{aligned}\phi'(x)(z-x) + \frac{1}{16}\sqrt{\phi''(x)}(z-x) &\leq -\frac{1}{16}\sqrt{\phi''(x)}(z-x) \\ &\leq -\frac{z-x}{16x} = \frac{1}{16} - \frac{z}{16x}\end{aligned}$$

where we used $x \geq \frac{1}{\sqrt{\phi''(x)}}$ from (2.28) at the end.

Case 4: $x \geq 1 - \frac{1}{12\nu}$.

By the same argument as case 3, we have $\phi'(x)(z-x)+\frac{1}{16}\sqrt{\phi''(x)}(z-x) \leq \frac{1}{16} - \frac{1-z}{16(1-x)}$.

Combining all the cases, we have the result.

□

2.B Using the universal barrier

If the barrier function ϕ_i for K_i is not given, we can use $w_i = 1$ and the universal barrier for Algorithm 4 [134, 119]. In this case, the algorithm takes $O(\sqrt{n}\log n \log(\frac{n\kappa R}{\varepsilon r}))$ steps, and the cost of computing a good enough approximation of $\nabla\phi_i$ and $\nabla^2\phi_i$ both takes $n_i^{O(1)}\log(\frac{nR}{r})$ time for each i, assuming the following mild conditions:

1. We can check if any $\boldsymbol{x}_{[i]}$ is in K_i in $n_i^{O(1)}$ time.
2. We are given $\boldsymbol{z}_{[i]}$ such that $B(\boldsymbol{z}_{[i]}, r) \subset K_i$.

We briefly outline the technical details in this appendix section.

Definition 2.33. For any convex set $K \subset \mathbb{R}^n$ and point $x \in K$, the *polar set of K at $\boldsymbol{x}$* is given by

$$K^\circ(\boldsymbol{x}) = \{\boldsymbol{y} \in \mathbb{R}^n : \boldsymbol{y}^\top(\boldsymbol{z} - \boldsymbol{x}) \leq 1, \forall \boldsymbol{z} \in K\}.$$

We further use $\mathrm{vol}(K)$ to denote the *volume* of K, $\boldsymbol{\mu}(K)$ to denote its center of gravity, and $\mathrm{Cov}(K)$ to denote its covariance matrix.

Theorem 2.34 ([134, 119])**.** *For any convex set $K \subset \mathbb{R}^n$, the universal barrier function $\phi(\boldsymbol{x}) \stackrel{\text{def}}{=} \log\mathrm{vol}(K^\circ(\boldsymbol{x}))$ is a n-self-concordant barrier.*

The gradient and Hessian of the universal barrier function ϕ can be computed using the center of gravity and the covariance of $K^\circ(\boldsymbol{x})$.

Lemma 2.35 ([119, Lemma 1])**.** *For any convex set $K \subset \mathbb{R}^n$ and any $\boldsymbol{x} \in \mathrm{int}(K)$, we have*

$$\begin{aligned}
\nabla\phi(\boldsymbol{x}) &= -(n+1)\mu(K^\circ(\boldsymbol{x})),\\
\nabla^2\phi(\boldsymbol{x}) &= (n+1)(n+2)\mathrm{Cov}(K^\circ(\boldsymbol{x})) + (n+1)\mu(K^\circ(\boldsymbol{x}))\mu(K^\circ(\boldsymbol{x}))^\top.
\end{aligned}$$

Computing center of gravity and covariance takes polynomial time. See for example [118] for a survey.

Theorem 2.36 ([64, 155])**.** *Suppose we are given a membership oracle for a convex set $K \subset \mathbb{R}^n$ with cost $\mathcal{T}$. Assuming $B(\mathbf{0}, r) \subset K \subset B(\mathbf{0}, R)$, we can compute $\boldsymbol{x}$ and $\boldsymbol{\Sigma}$ such that*

$$\|\boldsymbol{x} - \boldsymbol{\mu}(K)\|_{\mathrm{Cov}(K)^{-1}} \leq \varepsilon \qquad \textit{and} \qquad (1-\varepsilon)\boldsymbol{\Sigma} \preccurlyeq \mathrm{Cov}(K) \preccurlyeq (1+\varepsilon)\boldsymbol{\Sigma}$$

in $O\left(n^{O(1)} \cdot \mathcal{T} \cdot \log(R/r)/\varepsilon^2\right)$ time.

Next, note that the membership oracle of $K^\circ(\boldsymbol{x})$ involves optimizing one linear function over the convex set K and can be implemented using a membership oracle of K and the ellipsoid method. Therefore, for any $\boldsymbol{x}$, we can compute an approximate gradient $\boldsymbol{g}$ and approximate Hessian $\mathbf{H}$ of the universal barrier function such that

$$\|\boldsymbol{g} - \nabla\phi(\boldsymbol{x})\|_{\nabla^2\phi(\boldsymbol{x})^{-1}} \leq \varepsilon \qquad \text{and} \qquad (1-\varepsilon)\mathbf{H} \preccurlyeq \nabla^2\phi(\boldsymbol{x}) \preccurlyeq (1+\varepsilon)\mathbf{H}$$

in $O(n^{O(1)} \cdot \mathcal{T} \cdot \log(R/r)/\varepsilon^2)$ time.

Finally, we note that as long as $\varepsilon \leq \frac{1}{\log^c m}$ for some large enough constant c, our robust interior point method works for these approximate gradient and Hessian with the same guarantee. Since the proof is analogous, we skip the analysis here. Most commonly used convex sets have known explicit barrier functions with good self-concordance, for which we do not need heavy machinery like the above.

2.C Hyperbolic function lemmas

Lemma 2.37. *For any $x, y \in \mathbb{R}$ with $|y| \leq \frac{1}{8}$, we have*

$$\begin{aligned} |\sinh(x+y) - \sinh(x)| &\leq \frac{1}{7}|\sinh(x)| + \frac{1}{7} \\ |\cosh(x+y) - \cosh(x)| &\leq \frac{1}{7}\cosh(x). \end{aligned}$$

Proof. Note that $\sinh(x+y) = \sinh(x)\cosh(y) + \cosh(x)\sinh(y)$. Using that $\big||\cosh(x)| - |\sinh(x)|\big| \leq 1$, we have

$$\begin{aligned} |\sinh(x+y) - \sinh(x)| &\leq |\sinh(x)| \cdot |\cosh(y) - 1| + \cosh(x)\sinh(y) \\ &\leq |\sinh(x)| \cdot \big(|\cosh(y) - 1| + |\sinh(y)|\big) + |\sinh(y)| \end{aligned}$$

The first result follows from this and $|\cosh(y) - 1| + |\sinh(y)| \leq \frac{1}{7}$ for $|y| \leq \frac{1}{8}$.

For the second result, note that $\cosh(x+y) = \cosh(x)\cosh(y) + \sinh(x)\sinh(y)$. Hence,

$$\begin{aligned} |\cosh(x+y) - \cosh(x)| &\leq (\cosh(y) - 1)\cosh(x) + \sinh(x)\sinh(y) \\ &\leq \big(\cosh(y) - 1 + |\sinh(y)|\big) \cdot \cosh(x) \\ &\leq \frac{1}{7}\cosh(x). \end{aligned}$$

□

Lemma 2.38. *For any $x \geq 0$ and $0 \leq y \leq 1$, we have $\cosh(x+y) \leq (1+2y)\cosh(x)$.*

Proof. Note that $\cosh(x+y) = \cosh(x)\cosh(y) + \sinh(x)\sinh(y)$ and $\exp(x) = \sinh(x) + \cosh(x)$. Then we have

$$\begin{aligned} \cosh(x+y) &= \cosh(x)\left[\exp(y) - \sinh(y)\right] + \sinh(x)\sinh(y) \\ &\leq \cosh(x)\left[\exp(y) - \sinh(y)\right] + \cosh(x)\sinh(y) \\ &= \cosh(x)\exp(y) \\ &\leq \cosh(x) + 2y\cosh(x), \end{aligned}$$

where we use $\exp(y) \leq 1 + 2y$ for $0 \leq y \leq 1$. □

Chapter 3

Matrices, graphs, and their decompositions

To design efficient data structures for PathFollowingRobust (Algorithm 4), we leverage problem-specific structure in the constraint of (P). In this chapter, we establish the necessary background for discussing structures in matrices and graphs.

3.1 Graph of a matrix

A widely-used method for identifying structures in a sparse matrix $\mathbf{A}$ involves associating a graph with its non-zero pattern, which captures the interactions between the equations in the system. We introduce one such graph:

Definition 3.1 (Dual graph)**.** The *dual graph* $G_{\mathbf{A}}$ of a constraint matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$ to be the *hypergraph* with vertex set $\{1, \ldots, n\}$ corresponding to the rows of $\mathbf{A}$ and hyperedges $\{e_1, \ldots, e_m\}$, such that vertex i is in hyperedge e_j if $\mathbf{A}_{i,j} \neq 0$. Equivalently, a hyperedge on the vertex set $S \subseteq V$ can be thought of as a clique on S in a regular graph.

Example 3.2. Consider the matrix

$$\mathbf{A} \overset{\text{def}}{=} \begin{bmatrix} 3 & 1 & 0 & 0 & 4 \\ 0 & 1 & 5 & 9 & 0 \\ 2 & 0 & 6 & 5 & 3 \\ 5 & 0 & 0 & 8 & 9 \end{bmatrix} \in \mathbb{R}^{4 \times 5}.$$

Its dual graph has 4 vertices and 5 hyperedges, with each hyperedge corresponding to one column of $\mathbf{A}$.

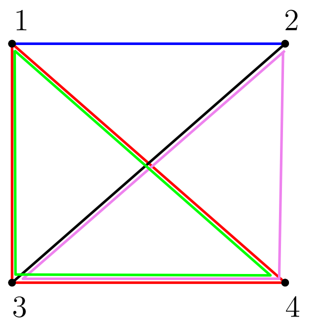


Figure 3.1: The dual graph $G_{\mathbf{A}}$. Hyperedges are identifiable by color, for example, column 4 is colored pink, and is indicated by the pink triangle on vertices 2,3,4.

The definition of the dual graph is motivated by flow problems. Recall the linear constraint for a problem is of the form $\mathbf{B}^\top \boldsymbol{f} = \boldsymbol{d}$, where $\mathbf{B} \in \mathbb{R}^{m\times n}$ is the edge-vertex incidence matrix of the input graph, $\boldsymbol{d}$ is the demand to be routed, and $\boldsymbol{f}$ is the flow we want to compute. Viewing $\mathbf{B}$ as a standalone constraint matrix, we can associate a dual graph with it. It turns out (by design) the dual graph of $\mathbf{B}$ is precisely the input graph.

In the following sections, we discuss structural properties that can be found in these dual graphs.

3.2 Treewidth and separability

Definition 3.3. A *tree-decomposition* of a graph G is a pair (X, T), where T is a tree, and $X : V(T) \mapsto 2^{V(G)}$ is a family of subsets of $V(G)$ called *bags* labelling the vertices of T, such that

1. $\bigcup_{t\in V(T)} X(t) = V(G)$,
2. for each $v \in V(G)$, the nodes $t \in V(T)$ with $v \in X(t)$ induces a connected subgraph of T, and
3. for each $e = uv \in V(G)$, there is a node $t \in V(T)$ such that $u, v \in X(t)$.

The *width* of a tree-decomposition (X, T) is $\max\{|X(t)| - 1 \ : \ t \in T\}$. The *treewidth* of G is the minimum width over all tree-decompositions of G. Intuitively, the treewidth of a graph captures how close the graph is to being a tree.

The following structural results about treewidth are elementary.

Lemma 3.4. *If G is a graph on n vertices and $tw(G) = \tau$, then $|E(G)| \leq n\tau$.* $\square$

Lemma 3.5. *If G' is a subgraph of G, then $tw(G') \leq tw(G)$.* $\square$

Lemma 3.6. *For the complete graph on k vertices, $tw(K_k) = k - 1$.* $\square$

There are some basic relations between the sparsity of a matrix $\mathbf{A}$ and the treewidth of its dual graph:

Lemma 3.7. *Any column of $\mathbf{A}$ with sparsity τ induces a clique of size τ in $G_{\mathbf{A}}$. Hence,* $\mathrm{tw}(\mathbf{A})$ *is lower bounded by the max number of nonzeros in a column of* $\mathbf{A}$. $\square$

Treewidth is a natural structural parameter of a graph, with close connections to graph algorithms of a recursive nature. At a high level, it is generalized by the notion of well-separable graphs. We are particularly interested in its connection to vertex separators.

Definition 3.8. Let $G = (V, E)$ be a graph. For any $W \subseteq V$ and $1/2 \leq \alpha < 1$, an *α-vertex separator of W* is a set $S \subseteq V$ of vertices such that every connected component of the graph $G[V - S]$ contains *at most* $\alpha \cdot |W|$ vertices of W. In the particular case when $W = V$, we call the separator an *α-vertex separator of G*. The *separator number* of G is the maximum over all subsets W of V of the size of the smallest $1/2$-vertex separator of W in G.

We sometimes denote an α-vertex separator S by (G_1, S, G_2), where $V(G_1) \cup S \cup V(G_2) = V(G)$, and G_1 and G_2 are disconnected in $G \setminus S$.

Similar to treewidth, separator numbers are monotone.

Lemma 3.9. *Let G' be a subgraph of G. For any constant $1/2 \leq \alpha < 1$, the size of the smallest α-vertex separator of G' is at most that of G.* $\square$

The following theorem relates the treewidth of a graph and the separator number.

Theorem 3.10 ([24], Lemma 6)**.** *If G is a graph with treewidth τ, then there exists a 1/2-balanced separator of G of size at most $\tau + 1$.* $\square$

Generalizing vertex separators, we have the concept of (recursively-)separable graphs.

Definition 3.11 (Separable graphs)**.** A subgraph-closed class $\mathcal{C}$ of (hyper-)graphs is said to be *α-separable* for some $\alpha \in [0, 1]$, if there exists universal constants $b \in (0, 1)$ and $c > 0$, such that for any $G = (V, \mathcal{E}) \in \mathcal{C}$, the vertices of G can be partitioned into S, A and B such that $|S| \leq c \cdot |V|^{\alpha}$, there are no edges between A and B, and $|A|, |B| \leq b|V|$. We call S the *(b-)balanced vertex separator* of H.

A notable case is $\alpha = 1/2$, which includes the family of planar and bounded-genus graphs [123]. It has also been empirically observed that road networks have separators of size $n^{1/3}$[54, 147].

Observe that separable graphs are recursively separable, with the balanced separator at each recursion level decreasing geometrically in size. Bounded-treewidth graphs are recursively separable as well, however, the balanced seaprators at each recursion level is only guaranteed to be bounded by the treewidth. The next section defines the notion of a separator tree to capture both cases.

3.3 Separator tree

Our technical contribution in this section is the definition of a fine-grained separator tree which we call the (a, b, λ)-*separator tree.* The parameters are defined based on the parameters of separable graphs, but they also capture important characteristics of other classes such as low-treewidth graphs. These trees guarantee that at any node, we are able to separate not only the associated graph region, but also the boundary of the region. We use them to maintain the tree operators from the implicit representations, and a careful analysis of node and boundary sizes allows us to conclude that the maintenance can be performed efficiently.

The notion of using a *separator tree* to represent the recursive decomposition of a separable graph is well-established in literature, c.f [67, 87]. In our work, we use the following definition:

Definition 3.12 (Separator tree)**.** Let G be a hypergraph with n vertices, m hyperedges, and max hyperedge size ρ. A *separator tree* $\mathcal{S}$ for G is a *constant-degree* tree whose nodes represent a recursive decomposition of G based on balanced separators.

Formally, each node of $\mathcal{S}$ is a *region* (edge-induced subgraph) H of G; we denote this by $H \in \mathcal{S}$. At a node H, we define subsets of vertices $\partial H, S(H), F_H$, where ∂H is the set of *boundary vertices* of H, i.e. vertices with neighbours outside H in G; $S(H)$ is a balanced vertex separator of H; and F_H is the set of *eliminated vertices* at H. Furthermore, let $E(H)$ denote the edges contained in H.

The nodes and associated vertex sets are defined in a top-down manner as follows:

1. The root of $\mathcal{S}$ is the node $H = G$, with $\partial H = \emptyset$ and $F_H = S(H)$.
2. A non-leaf node $H \in \mathcal{S}$ has a constant number of children whose union is H. The children form a edge-disjoint partition of H, and the intersection of their vertex sets is a balanced separator $S(H)$ of H. Define the set of eliminated vertices at H to be $F_H \overset{\text{def}}{=} S(H) \setminus \partial H$.
   The set $F_H \cup \partial H$ consists of all vertices in the boundary and separator, which can intuitively be interpreted as the *skeleton* of H. In later sections, we recursively construct

graphs (matrices) on $F_H \cup \partial H$ which capture compressed information about all of H. By definition of boundary vertices, for a child D of H, we have $\partial D \overset{\text{def}}{=} (\partial H \cup S(H)) \cap V(D)$.

3. At a leaf node H, we define $S(H) = \emptyset$ and $F_H = V(H) \setminus \partial H$. (This convention allows leaf nodes to exist at different levels in $\mathcal{S}$.) The leaf nodes of $\mathcal{S}$ partition the edges of G.

We use η to denote the height of $\mathcal{S}$.

For a separator tree to be meaningful, the leaf node regions should be sufficiently small, to indicate that we have a good overall decomposition of the graph. Additionally, for our work, we want a more careful bound on the sizes of the skeleton of regions. This motivates the following refined definition:

Definition 3.13 ((a, b, λ)-separator tree)**.** Let G be a graph with n vertices, m edges, and max hyperedge size ρ. Let $a \in [0, 1]$ and $b \in (0, 1)$ be constants, and $\lambda \geq 1$ be an expression in terms of m, n, ρ. An (a, b, λ)-separator tree $\mathcal{S}$ for G is a separator tree satisfying the following additional properties:

1. There are at most $O(b^{-i})$ nodes at level i in $\mathcal{S}$,
2. any node H at level i satisfies $|F_H \cup \partial H| \leq O(\lambda \cdot b^{ai})$,
3. a node H at level i is a leaf node if and only if $|V(H)| \leq O(\rho)$.

Intuitively, a and b come from the separability parameters of G, and λ is a scaling factor for node sizes in $\mathcal{S}$. Since there could be hyperedges of size ρ, regions of size ρ are not necessarily separable, so we set the region as a leaf.

We make extensive use of these properties in subsequent sections when computing runtimes.

For now, we show how to construct a separator tree for an n^α-separable graph by modifying the proof from [68], as well as for a treewidth-τ graph.

Lemma 3.14. *Suppose G is a graph on n vertices and m edges. If G is n^α-separable for $\alpha < 1$, then G admits an (α, b, cn^α)-separator tree, where $b \in (0, 1)$ and $c > 0$ are some constants. Furthermore, if a balanced vertex separator for G can be computed in $T(n)$ time, then the separator tree can be constructed in $\widetilde{O}(T(n))$ time.*

Proof. Let $b' \in (0, 1)$ and $c' = 1$ (without loss of generality) be the parameters for G being n^α-separable. In the separator tree construction process, assume inductively that we have constants $b \in (0, 1)$ and $c > 0$, both to be chosen later, such that for any node H at level i,

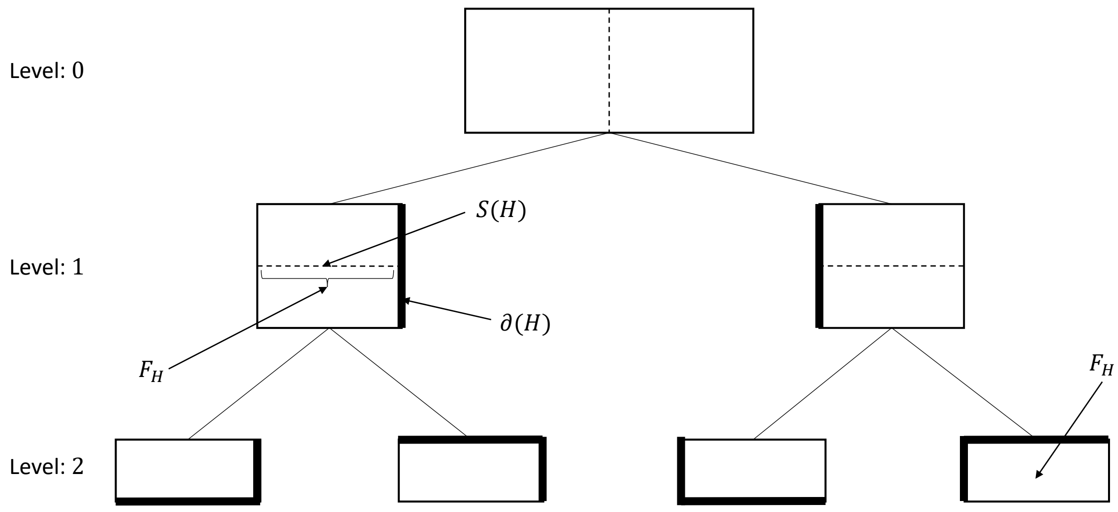


Figure 3.2: An example of a separator tree. The bold edges denote the boundary of each component, ∂H while the dotted lines denote the separators $S(H)$. Note that $F_H = S(H) \backslash \partial H$ is defined differently on the leaves.

we have $|V(H)| \leq b^i n$ and $|\partial H| \leq cn^\alpha \cdot b^{\alpha i}$. In the base case at the root node, we have $i = 0$, and $|V(G_{\mathbf{A}})| \leq n$ and $|\partial G_{\mathbf{A}}| = 0 \leq cn^\alpha$.

We show how to construct the nodes at level $i+1$. Let H be an already-constructed node at level i. There are three cases:

1. If H satisfies $|V(H)| \leq b^{i+1}n$ and $|\partial H| \leq cn^\alpha \cdot b^{\alpha(i+1)}$, put a copy of H as its only child at level $i+1$.

2. If $|V(H)| \geq b^{i+1}n$, then assign a weight of 1 to all vertices, find a balanced vertex separator $S(H)$, and partition H accordingly into H_1 and H_2. Let us consider H_1; the analogous holds for H_2.

   By definition of separability, we know $|V(H_1)| \leq b' \cdot |V(H)| + |V(H)|^\alpha \leq b \cdot |V(H)| \leq b^{i+1}n$ as long as $b \in (b', 1)$. If $|\partial H_1| \leq c \cdot |V(H_1)|^\alpha$, then we can upper bound this expression by $cn^\alpha \cdot b^{\alpha(i+1)}$, and we are done.

   On the other hand, if $|\partial H_1| > c \cdot |V(H_1)|^\alpha$, then by definition of boundary, we have $|\partial H_1| \leq |\partial H| + |S(H)| \leq (c+1)n^\alpha \cdot b^{\alpha i}$ using the guarantees at H. Next, we assign a weight of 1 to vertices in ∂H_1 and 0 to all other vertices, find a balanced separator $S(H_1)$ of H_1 with respect to these weights, and create two children D_1, D_2 of H_1 accordingly.

Then, for $j = 1, 2$, we have

$$\begin{aligned}|\partial D_j| &\leq b \cdot |\partial H_1| + |V(H_1)|^\alpha \\ &\leq b(c+1)n^\alpha \cdot b^{\alpha i} + n^\alpha \cdot b^{\alpha(i+1)} \\ &\leq \left(b^{1-\alpha} \cdot \frac{c+1}{c} + \frac{1}{c}\right) \cdot cn^\alpha \cdot b^{\alpha(i+1)},\end{aligned}$$

As long as c is large enough so the expression in the parentheses to be less than 1. In this case, observe that we can add $S(H_1)$ to the balanced separator $S(H)$, and set D_1 and D_2 directly as the children of H.

3. If $|V(H)| \leq b^{(i+1)n}$ and $|\partial H| \geq cn^\alpha \cdot b^{\alpha(i+1)}$, then we apply case 2 with H_1 being H.

So we have shown inductively that at the end of this construction, any node H at level i satisfies $|V(H)| \leq b^i n$ and $|\partial H| \leq cn^\alpha \cdot b^{\alpha i}$. It follows that $|F_H \cup \partial H| \leq |S(H)| + |\partial H| = O(cn^\alpha \cdot b^{\alpha i})$.

Next, we show that there are only $O(b^{-i})$ nodes at level i. Let $L_i(n)$ denote the total number of boundary vertices with multiplicities, when carrying out the construction starting on a graph of size n and ending when each leaf node H satisfies the level-i assumptions. We can recursively write

$$\begin{aligned}L_i(k) &= \sum_{j=1}^{4} L_i(b_j k + 3ck^\alpha), && \text{if } k > Cb^i n \\ B_i(k) &= 1 && \text{else.}\end{aligned}$$

where $\sum b_j = 1$, each $b_j \leq b'$, and C is a positive constant we choose. To see this, note that a node of size k has at most four children in the construction; the separator is of size $3ck^\alpha$ since we may need to compute up to three separators each of size ck^α and take their union; and child j has at most $b_j k$ vertices that are not from the separator. Solving the recursion yields $L_i(k) \leq k/(Cb^i n) - \gamma k^\alpha$ for some constant $\gamma > 0$. Therefore, there are at most $L_i(n) \leq O(b^{-i})$ nodes at level i.

Finally, it is straightforward to see that the separator tree can be computed in $\widetilde{O}(T(n))$ time, since the node sizes decrease by a geometric factor as we proceed down the tree during construction. □

Lemma 3.15. *Suppose G is a graph on n vertices and m edges. If G is n^α-separable for $\alpha < 1$, then G admits an (α, b, cn^α)-separator tree, where $b \in (0, 1)$ and $c > 0$ are some constants. Furthermore, if a balanced vertex separator for G can be computed in $T(n)$ time, then the separator tree can be constructed in $\widetilde{O}(T(n))$ time.*

First, we show how to construct a $(0, 1/2, O(\tau \log n))$-separator tree $\mathcal{S}$ for $G_{\mathbf{A}}$ when we have a tree decomposition of $G_{\mathbf{A}}$ of width τ. At the root of $\mathcal{S}$, we can use the tree decomposition to compute a balanced separator S of $G_{\mathbf{A}}$ of size $O(\tau)$ in $\widetilde{O}(n\tau)$ time (c.f. [58, Theorem 4.17]), so that the two parts A and B of $G_{\mathbf{A}} \setminus S$ each have size at most $\frac{2}{3}n$. We construct two children of the root node on the vertex sets $A \cup S$ and $B \cup S$ respectively, and apply this procedure recursively until the nodes are of size at most 9τ.

Claim 3.16. *There are $O(n/\tau)$-many leaves at the end of this construction.*

Proof. Let $L(k)$ denote the number of leaves when starting the construction with a size k subgraph. We know $L(k) = 1$ if $k \le 9\tau$, and $L(k) = L(k_1 + \tau) + L(k_2 + \tau)$ if $k > 9\tau$, where $k_1 + k_2 + \tau = k$ and $k_1, k_2 \le 2/3k$. By induction, we can show that $L(k) \le 2(k/\tau - 1)$ when $k > 2\tau$, where the balanced separator crucially ensures that the recursion does not reach the base case of $k \le 2\tau$. □

3.4 Laplacian, Cholesky decomposition and Schur complement

Now, we switch gears back to discussing matrices, but this time aided by our understanding of their structure as captured by their dual graph. Let us begin with a little linear algebra.

Definition 3.17. Let G be a graph on n vertices with positive weighted edges given by $\boldsymbol{w} \in \mathbb{R}_+^E$. Suppose its vertex-edge incidence matrix is given by $\mathbf{A} \in \mathbb{R}^{n \times m}$. Then its *(weighted) Laplacian* is the matrix $\mathbf{L} = \mathbf{A}\mathbf{W}\mathbf{A}^\top$, where $\mathbf{W} = \mathrm{diag}(\boldsymbol{w})$.

We treat graphs and their Laplacians interchangeably. We recall some useful facts about Laplacian matrices:

Fact 3.18. *For any $n \times n$ Laplacian $\mathbf{L}$, we have $\mathrm{rank}(\mathbf{L}) = n - 1$, and $\mathbf{1} \in \mathrm{Kernel}(\mathbf{L})$. The projection matrix onto $\mathbf{L}$'s image is $\mathbf{P}_{\mathbf{L}} = \mathbf{I} - \mathbf{1}\mathbf{1}^\top/n$.*

Definition 3.19. Let $\mathbf{M}$ be a real symmetric matrix. The *spectral decomposition* of $\mathbf{M}$ is given by

$$\mathbf{M} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^\top = \sum_{\lambda \in \Lambda} \lambda \mathbf{P}_\lambda,$$

where Λ is the set of eigenvalues of $\mathbf{M}$, $\boldsymbol{\Lambda}$ is the diagonal matrix of eigenvalues, and $\mathbf{P}_\lambda$ is the orthogonal projector onto the eigenspace with eigenvalue λ. We define the *Moore-Penrose pseudo-inverse* of $\mathbf{M}$, denote by $\mathbf{M}^\dagger$, by

$$\mathbf{M}^\dagger = \sum_{\lambda \in \Lambda, \lambda \neq 0} \lambda^{-1} \mathbf{P}_\lambda.$$

The Moore-Penrose pseudo-inverse is unique, and satisfies

$$\mathbf{M}\mathbf{M}^{\dagger}\mathbf{M} = \mathbf{M};$$
$$\mathbf{M}^{\dagger}\mathbf{M}\mathbf{M}^{\dagger} = \mathbf{M}^{\dagger}.$$

Fact 3.20. *Suppose* $\mathbf{M}$ *is a symmetric matrix and* $\mathbf{X}$ *is a non-singular matrix, and* $\mathbf{P}$ *is the projection matrix onto the image of* $\mathbf{X}^{\top}\mathbf{M}\mathbf{X}$. *Then,*

$$\left(\mathbf{X}^{\top}\mathbf{M}\mathbf{X}\right)^{\dagger} = \mathbf{P}\mathbf{X}^{-1}\mathbf{M}^{\dagger}\left(\mathbf{X}^{-1}\right)^{\top}\mathbf{P}.$$

Next, let G be an edge-weighted graph. Consider the partition of vertices in G into two subsets C and $F = V(G) \setminus C$ called *boundary* and *interior* vertices. This partitions $\mathbf{L}$ into four blocks:

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_{F,F} & \mathbf{L}_{F,C} \\ \mathbf{L}_{C,F} & \mathbf{L}_{C,C} \end{bmatrix}.$$

Definition 3.21 (Block Cholesky decomposition)**.** The *block Cholesky decomposition* of a symmetric matrix $\mathbf{L}$ with blocks indexed by F and C defined as above is:

$$\mathbf{L} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{L}_{C,F}(\mathbf{L}_{F,F})^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{L}_{F,F} & \mathbf{0} \\ \mathbf{0} & \mathbf{Sc}(\mathbf{L}, C) \end{bmatrix} \begin{bmatrix} \mathbf{I} & (\mathbf{L}_{F,F})^{-1}\mathbf{L}_{F,C} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}. \tag{3.1}$$

The middle matrix in the decomposition is a block-diagonal matrix with blocks indexed by F and C, with the lower-right block being:

Definition 3.22 (Schur complement)**.** The *Schur complement* $\mathbf{Sc}(\mathbf{L}, C)$ of $\mathbf{L}$ onto C is the Laplacian matrix resulting from a partial symmetric Gaussian elimination on $\mathbf{L}$. Formally,

$$\mathbf{Sc}(\mathbf{L}, C) = \mathbf{L}_{C,C} - \mathbf{L}_{C,F}\mathbf{L}_{F,F}^{-1}\mathbf{L}_{F,C}.$$

It is known that $\mathbf{Sc}(\mathbf{L}, C)$ is the Laplacian of another graph with vertex set C. We further use the convention that if H is a subgraph of G and $V(H) \subset C$, then $\mathbf{Sc}(H, C)$ simply means $\mathbf{Sc}(H, C \cap V(H))$. Graph theoretically, the Schur complement has the following interpretation:

Lemma 3.23. *Let* $V(G) = \{v_1, \ldots, v_n\}$. *Let* $C = V(G) - v_1$. *Let* $\boldsymbol{w}_{ij}$ *denote the weight of edge* $v_i v_j$. *Then*

$$\mathbf{Sc}(\mathbf{L}, C) = G[C] + H,$$

where $G[C]$ is the subgraph of G induced on the vertex set S, and H is the graph on S with edges v_iv_j where $i,j \in N(v_1)$, and $\boldsymbol{w}_{ij} = \boldsymbol{w}_{1i}\boldsymbol{w}_{1j}/\boldsymbol{w}_1$, where $\boldsymbol{w}_1$ is the total weight of edges incident to v_1 in G. Note that on the right hand side, we use a graph to mean its Laplacian. □

Taking Schur complement is an associative operation. Furthermore, it commutes with edge deletion, and more generally, edge weight deletion. Finally, for our purposes, it can be decomposed under certain special circumstances.

Lemma 3.24. *If $X \subseteq Y \subseteq V(G)$, then $\mathbf{Sc}(\mathbf{Sc}(\mathbf{L}, Y), X) = \mathbf{Sc}(\mathbf{L}, X)$.* □

Lemma 3.25. *Let $\boldsymbol{w}_e$ denote the weight of edge e in G. Suppose $C \subseteq V(G)$, and H is a subgraph of G on the vertex set C with edge weights $\boldsymbol{w}'_e \leq \boldsymbol{w}_e$ for all edges in $G[C]$. Let $\mathbf{L}'$ denote the Laplacian of H. Then, $\mathbf{Sc}(\mathbf{L} - \mathbf{L}', C) = \mathbf{Sc}(\mathbf{L}, C) - \mathbf{L}'$.* □

Lemma 3.26. *Let $\boldsymbol{w}_e$ denote the weight of edge e in G. Suppose $C \subseteq V(G)$, and e is an edge not incident to any vertex in C. Let $\mathbf{L}'$ be the Laplacian of $G \setminus e$. Then, $\mathbf{Sc}(\mathbf{L}', C) = \mathbf{Sc}(\mathbf{L}, C)$.* □

Lemma 3.27. *Let $\mathbf{L}$ be the Laplacian of graph G with the decomposition $\mathbf{L} = \mathbf{L}_1 + \mathbf{L}_2$, where $\mathbf{L}_1$ is a Laplacian supported on the vertex set V_1 and $\mathbf{L}_2$ on V_2. Furthermore, suppose $V_1 \cap V_2 \subseteq C$ for some vertex set $C \subseteq V(G)$. Then*

$$\mathbf{Sc}(\mathbf{L}, C) = \mathbf{Sc}(\mathbf{L}_1, C \cap V_1) + \mathbf{Sc}(\mathbf{L}_2, C \cap V_2).$$

Proof. We have

$$\begin{aligned}
\mathbf{Sc}(\mathbf{L}, C) &= \mathbf{Sc}(\mathbf{L}_1 + \mathbf{L}_2, C) \\
&= \mathbf{Sc}(\mathbf{Sc}(\mathbf{L}_1 + \mathbf{L}_2, C \cup V_2), C) \\
&= \mathbf{Sc}(\mathbf{Sc}(\mathbf{L}_1, C \cup V_2) + \mathbf{L}_2, C) && \text{(by Lemma 3.25)} \\
&= \mathbf{Sc}(\mathbf{Sc}(\mathbf{L}_1, C) + \mathbf{L}_2, C) && \text{(since } (C \cup V_2) \cap V_1 \subseteq C\text{)} \\
&= \mathbf{Sc}(\mathbf{L}_1, C) + \mathbf{Sc}(\mathbf{L}_2, C), && \text{(by Lemma 3.25)} \\
&= \mathbf{Sc}(\mathbf{L}_1, C \cap V_1) + \mathbf{Sc}(\mathbf{L}_2, C \cap V_2) && \text{(since } \mathbf{L}_i \text{ is supported on } V_i \text{ for } i = 1, 2\text{)}
\end{aligned}$$

as desired. □

3.5 Recursive Cholesky factorization

Let $\mathcal{S}$ be any separator tree of the dual graph $G_{\mathbf{A}}$ of matrix $\mathbf{A}$. In this section, we show how to use $\mathcal{S}$ to factor the matrix $\mathbf{L}^{\dagger} \overset{\text{def}}{=} (\mathbf{AWA}^{\top})^{\dagger}$ recursively.

Definition 3.28 (C_i, F_i)**.** Let $\mathcal{S}$ be the separator tree of $G_{\mathbf{A}}$ with height η. For all $0 \leq i \leq \eta$, define $F_i = \bigcup_{H \in \mathcal{T}(i)} F_H$ to be the vertices eliminated at level i, and define $C_i = \bigcup_{H \in \mathcal{T}(i)} \partial H$ to be the vertices remaining after eliminating vertices at level $\geq i$. We define C_η to be $V(G_{\mathbf{A}})$.

Note that F_i is the disjoint union of F_H over all nodes H at level i in the separator tree. $F_0, \dots, F_\eta$ partitions $V(G_{\mathbf{A}})$. By the definition of ∂H and F_H, we know $F_i = C_{i+1} \setminus C_i$ for all $0 \leq i \leq \eta$. It follows that $C_0 \overset{\text{def}}{=} F_R \subset \cdots \subset C_{\eta-1} \subset C_\eta = V(G_{\mathbf{A}})$ and $C_i = \cup_{j \leq i} F_j$ where R is the root node.

These sets generalize the sets C and F from the block Cholesky decomposition in (3.1).

Lemma 3.29. *By recursively applying Eq.* (3.1)*, we can decompose* $\mathbf{L}$ *as follows:*

$$\mathbf{L} = \mathbf{U}^{(\eta)\top} \cdots \mathbf{U}^{(1)\top} \begin{bmatrix} \mathbf{Sc}(\mathbf{L}, C_\eta)_{F_\eta, F_\eta} & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & \mathbf{Sc}(\mathbf{L}, C_0)_{F_0, F_0} \end{bmatrix} \mathbf{U}^{(1)} \cdots \mathbf{U}^{(\eta)}, \tag{3.2}$$

where the $\mathbf{U}^{(i)}$*'s are upper triangular matrices with*

$$\mathbf{U}^{(i)} = \mathbf{I} + \left(\mathbf{Sc}(\mathbf{L}, C_{i+1})_{F_i, F_i}\right)^{-1} \mathbf{Sc}(\mathbf{L}, C_{i+1})_{F_i, C_i}.$$

Furthermore, by Lemma 3.27, we can write

$$\mathbf{Sc}(\mathbf{L}, C_i) = \sum_{H \in \mathcal{S}(i)} \mathbf{Sc}(\mathbf{L}[H], F_H \cup \partial H).$$

□

Combining with Fact 3.20, we have the following decomposition for $\mathbf{L}^\dagger$:

Theorem 3.30 ($\mathbf{L}^\dagger$ factorization)**.** *Let* $\mathcal{S}$ *be the separator tree of* $G_{\mathbf{A}}$ *with height* η *and root node* R*. For each node* $H \in \mathcal{S}$ *with hyperedges* $E(H)$*, let* $\mathbf{A}_H \in \mathbb{R}^{n \times m}$ *denote the matrix* $\mathbf{A}$ *restricted to columns indexed by* $E(H)$*. Define*

$$\mathbf{L}[H] \overset{\text{def}}{=} \mathbf{A}_H \mathbf{W} \mathbf{A}_H{}^\top, \text{ and}$$
$$\mathbf{L}^{(H)} \overset{\text{def}}{=} \mathbf{Sc}(\mathbf{L}[H], F_H \cup \partial H).$$

Then, we have

$$\mathbf{L}^\dagger = \mathbf{P}_{\mathbf{L}} \mathbf{U}^{(\eta)^{-1}} \cdots \mathbf{U}^{(1)^{-1}} \mathbf{\Gamma} \mathbf{U}^{(1)^{-\top}} \cdots \mathbf{U}^{(\eta)^{-\top}} \mathbf{P}_{\mathbf{L}}, \tag{3.3}$$

where $\mathbf{P_L}$ *is the projection onto the image of* $\mathbf{L}$*, and*

$$\mathbf{\Gamma} \stackrel{\text{def}}{=} \begin{bmatrix} \sum_{H\in\mathcal{S}(\eta)} \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1} & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \dots & \sum_{H\in\mathcal{S}(1)} \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1} & \mathbf{0} \\ \mathbf{0} & \dots & \mathbf{0} & \left(\mathbf{L}^{(R)}_{F_R,F_R}\right)^{\dagger} \end{bmatrix}. \tag{3.4}$$

Proof. Note that the final block of $\mathbf{\Gamma}$ is $\left(\mathbf{L}^{(R)}_{F_R,F_R}\right)^{\dagger} = \left(\sum_{H\in\mathcal{S}(0)} \mathbf{L}^{(H)}_{F_H,F_H}\right)^{\dagger}$. In all other blocks of $\mathbf{\Gamma}$, each term $\mathbf{L}^{(H)}_{F_H,F_H}$ is indeed invertible. The identity follows immediately from Fact 3.20. $\square$

Remark 3.31. If $\mathbf{A}$ is the vertex-edge incidence matrix of graph $G_{\mathbf{A}}$ with Laplacian $\mathbf{L}$, then $\mathbf{P_L A} = \mathbf{A}$, so the projection matrix $\mathbf{P_L}$ can be safely omitted from the data structure.

3.6 Approximate recursive Cholesky factorization

Next, we show that the recursive factorization can also be approximated. We first formalize approximate Schur complements.

Definition 3.32 (Approximate Schur complement)**.** Let G be a weighted graph with Laplacian $\mathbf{L}$, and let C be a set of boundary vertices in G. We say that a Laplacian matrix $\widetilde{\mathbf{Sc}}(\mathbf{L}, C)$ is an *ε-approximate Schur complement* of $\mathbf{L}$ onto C if $\widetilde{\mathbf{Sc}}(\mathbf{L}, C) \approx_{\varepsilon} \mathbf{Sc}(\mathbf{L}, C)$, where we use $\approx_{\varepsilon}$ to mean an e^{ε}-spectral approximation.

Next, we have the approximate version of Lemma 3.29.

Lemma 3.33. *Suppose at every node $H \in \mathcal{S}$, we have an approximate Schur complement*

$$\mathbf{L}^{(H)} \approx_{\epsilon_{\mathbf{P}}} \mathbf{Sc}(\mathbf{L}[H], F_H \cup \partial H).$$

Then, we have

$$\mathbf{L} \approx_{\eta\epsilon_{\mathbf{P}}} \tilde{\mathbf{L}} \stackrel{\text{def}}{=} \tilde{\mathbf{U}}^{(\eta)\top} \cdots \tilde{\mathbf{U}}^{(1)\top} \tilde{\mathbf{T}} \tilde{\mathbf{U}}^{(1)} \cdots \tilde{\mathbf{U}}^{(\eta)}, \tag{3.5}$$

where

$$\tilde{\mathbf{T}} = \begin{bmatrix} \sum_{H\in\mathcal{S}(\eta)} \mathbf{L}^{(H)}_{F_H,F_H} & \dots & \mathbf{0} \\ \vdots & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \sum_{H\in\mathcal{S}(0)} \mathbf{L}^{(H)}_{F_H,F_H} \end{bmatrix}$$

and

$$\tilde{\mathbf{U}}^{(i)} = \mathbf{I} + \sum_{H\in\mathcal{S}(i)} \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1} \mathbf{L}^{(H)}_{F_H,\partial H}.$$

Furthermore, $\mathbf{1}$ *is in the null space of* $\tilde{\mathbf{L}}$*, and* $\operatorname{rank}(\tilde{\mathbf{L}}) = n-1$.

Proof. Let C_i, F_i be defined for each i according to Definition 3.28.

Let $\mathbf{L}^{(i)} \stackrel{\text{def}}{=} \sum_{H \in \mathcal{S}(i)} \mathbf{L}^{(H)}$. Note we have the decomposition

$$\begin{aligned}\mathbf{L}^{(i)}_{F_i,F_i} &\stackrel{\text{def}}{=} \sum_{H \in \mathcal{S}(i)} \mathbf{L}^{(H)}_{F_H,F_H}, \\ \mathbf{L}^{(i)}_{C_i,F_i} &= \sum_{H \in \mathcal{S}(i)} \mathbf{L}^{(H)}_{C_i,F_i} = \sum_{H \in \mathcal{T}(i)} \mathbf{L}^{(H)}_{\partial H,F_H}.\end{aligned}$$

Recall that the regions in $\mathcal{S}(i)$ partition the graph G. Furthermore, the intersection of $H, H' \in \mathcal{S}(i)$ is on their boundary, which is contained in C_{i-1}. Thus, we apply Lemma 3.27 to get

$$\begin{aligned}\mathbf{Sc}(\mathbf{L}, C_i) &= \sum_{H \in \mathcal{S}(i)} \mathbf{Sc}(\mathbf{L}[H], C_i \cap V(H)) \\ &= \sum_{H \in \mathcal{S}(i)} \mathbf{Sc}(\mathbf{L}[H], \partial H \cup F_H) \\ &\approx_{\epsilon_{\mathbf{P}}} \sum_{H \in \mathcal{S}(i)} \mathbf{L}^{(H)} = \mathbf{L}^{(i)}.\end{aligned} \tag{3.6}$$

Now, we prove inductively from $i=\eta$ to $i=1$ that

$$\mathbf{L} \approx_{(\eta-i+1)\epsilon_{\mathbf{P}}} \tilde{\mathbf{U}}^{(\eta)\top} \cdots \tilde{\mathbf{U}}^{(i+1)\top} \begin{bmatrix} \mathbf{L}^{(\eta)}_{F_\eta,F_\eta} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \ddots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{L}^{(i+1)}_{F_{i+1},F_{i+1}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{L}^{(i)} \end{bmatrix} \tilde{\mathbf{U}}^{(i+1)} \cdots \tilde{\mathbf{U}}^{(\eta)}. \tag{3.7}$$

We factor $\mathbf{L}^{(i)}$ in Eq. (3.7) using Cholesky decomposition. $\mathbf{L}^{(i)}$ is supported on C_i, and we can partition $C_i = F_i \cup C_{i-1}$. Then,

$$\mathbf{L}^{(i)} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{L}^{(i)}_{C_i,F_i}(\mathbf{L}^{(i)}_{F_i,F_i})^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{L}^{(i)}_{F_i,F_i} & \mathbf{0} \\ \mathbf{0} & \mathbf{Sc}(\mathbf{L}^{(i)}, C_{i-1}) \end{bmatrix} \begin{bmatrix} \mathbf{I} & (\mathbf{L}^{(i)}_{F_i,F_i})^{-1}\mathbf{L}^{(i)}_{F_i,C_i} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}. \tag{3.8}$$

For the Schur complement term in the factorization, we have

$$\begin{aligned}\mathbf{Sc}(\mathbf{L}^{(i)}, C_{i-1}) &\approx_{i\epsilon_{\mathbf{P}}} \mathbf{Sc}(\mathbf{Sc}(\mathbf{L}, C_i), C_{i-1}) && \text{(by Eq. (3.6))} \\ &= \mathbf{Sc}(\mathbf{L}, C_i) && \text{(by transitivity of Schur complements)} \\ &\approx_{\epsilon_{\mathbf{P}}} \mathbf{L}^{(i-1)}. && \text{(by Eq. (3.6))}\end{aligned}$$

So we can use $\mathbf{L}^{(i-1)}$ in place of the Schur complement term in Eq. (3.8), whose equality becomes an approximation with factor $\epsilon_{\mathbf{P}}$. At the root level, $\mathbf{L}^{(0)}_{F_0,F_0} = \mathbf{L}^{(0)}$ since $C_0 = \emptyset$, so we have the overall expression.

To show $\tilde{\mathbf{L}}\mathbf{1} = \mathbf{0}$, we start with the fact that $\mathbf{L}^{(0)}$ is a Laplacian, so its rows and columns sum to zero. Substituting $\mathbf{L}^{(0)}$ in (3.8) for $i = 1$, we see that $\mathbf{L}^{(1)}$'s rows and columns also sum to zero. Continuing this process, we conclude $\tilde{\mathbf{L}}$'s rows and columns sum to zero.

Finally, in the decomposition of $\tilde{\mathbf{L}}$, observe that $\tilde{\mathbf{U}}^{(1)} \cdots \tilde{\mathbf{U}}^{(\eta)}$ is full rank, and each block of $\tilde{\mathbf{T}}$ is full rank, except for the last block $\mathbf{L}^{(\eta)}$ whose rank is one less than full. Therefore, the rank of $\tilde{\mathbf{L}}$ is $n - 1$. □

Finally, we come to the approximation of $\mathbf{L}^{\dagger}$.

Theorem 3.34 ($\mathbf{L}^{\dagger}$ approximation)**.** *Suppose for each $H \in \mathcal{S}$, we have a Laplacian $\mathbf{L}^{(H)}$ satisfying*

$$\mathbf{L}^{(H)} \approx_{\epsilon_{\mathbf{P}}} \mathbf{Sc}(\mathbf{L}[H], \partial H \cup F_H).$$

Then, we have

$$\mathbf{L}^{\dagger} \approx_{\eta\epsilon_{\mathbf{P}}} \mathbf{\Pi}^{(\eta)\top} \cdots \mathbf{\Pi}^{(1)\top} \widetilde{\mathbf{\Gamma}} \mathbf{\Pi}^{(1)} \cdots \mathbf{\Pi}^{(\eta)}, \tag{3.9}$$

where $\mathbf{P}_{\mathbf{L}}$ is the projection onto the image of $\mathbf{L}$,

$$\mathbf{\Gamma} \overset{\text{def}}{=} \begin{bmatrix} \sum_{H \in \mathcal{S}(\eta)} \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \cdots & \sum_{H \in \mathcal{S}(1)} \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1} & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \left(\mathbf{L}^{(R)}_{F_R,F_R}\right)^{\dagger} \end{bmatrix}, \tag{3.10}$$

with R denoting the root, and

$$\mathbf{\Pi}^{(i)} = \mathbf{I} - \sum_{H \in \mathcal{S}(i)} \mathbf{L}^{(H)}_{\partial H, F_H} \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1}.$$

Proof of Theorem 3.34. Let $\tilde{\mathbf{L}}$ denote the $\eta\epsilon_{\mathbf{P}}$ approximation of $\mathbf{L}$ in Lemma 3.33. Applying Fact 3.20, we have

$$\tilde{\mathbf{L}}^{\dagger} = \mathbf{P}_{\tilde{\mathbf{L}}} \left(\tilde{\mathbf{U}}^{(1)} \cdots \tilde{\mathbf{U}}^{(\eta)}\right)^{-1} \tilde{\mathbf{T}}^{\dagger} \left(\left(\tilde{\mathbf{U}}^{(1)} \cdots \tilde{\mathbf{U}}^{(\eta)}\right)^{-1}\right)^{\top} \mathbf{P}_{\tilde{\mathbf{L}}},$$

where $\tilde{\mathbf{T}}$ is the block diagonal matrix in Lemma 3.33. We note that $(\tilde{\mathbf{U}}^{(i)})^{-\top} = \mathbf{\Pi}^{(i)}$, and $\tilde{\mathbf{T}}^{-1} = \widetilde{\mathbf{\Gamma}}$. Since $\mathbf{1}$ spans the null space of both $\mathbf{L}$ and $\tilde{\mathbf{L}}$, we conclude $\mathbf{P}_{\tilde{\mathbf{L}}} = \mathbf{P}_{\mathbf{L}}$ and $\tilde{\mathbf{L}}^{\dagger} \approx \mathbf{L}^{\dagger}$. □

3.7 Linear operator decomposition: tree operator

In this section, we fix $\mathcal{T}$ to be a *constant-degree* rooted tree with root node G, which we call the *operator tree.* We define two special classes of linear operators that build on the structure of $\mathcal{T}$. The advantage of these operators lie in their decomposability, which allows them to be efficiently updated.

Let each node $H \in \mathcal{T}$ be associated with a set F_H, where the F_H's are pairwise disjoint. Let each *leaf* node $L \in \mathcal{T}$ be further associated with a *non-empty* set $E(L)$, where the $E(L)$'s are pairwise disjoint. For a non-leaf node H, define $E(H) \stackrel{\text{def}}{=} \bigcup_{\text{leaf } L \in \mathcal{T}_H} E(L)$. Finally, define $E \stackrel{\text{def}}{=} E(G) = \bigcup_{\text{leaf } L \in \mathcal{T}} E(L)$ and $V \stackrel{\text{def}}{=} \bigcup_{H \in \mathcal{T}} F_H$.

Definition 3.35 (Inverse tree operator)**.** Let $\mathcal{T}$ be an operator tree with the associated sets as above. We say a linear operator $\nabla : \mathbb{R}^E \mapsto \mathbb{R}^V$ is an *inverse tree operator supported on* $\mathcal{T}$ if there exists a linear *edge operator* ∇_H for each non-root node H in $\mathcal{T}$, corresponding to the edge from H to its parent, such that ∇ can be decomposed as

$$\nabla = \sum_{\text{leaf } L, \text{ node } H \,:\, L \in \mathcal{T}_H} \mathbf{I}_{F_H} \nabla_{H \leftarrow L},$$

where $\nabla_{H \leftarrow L}$ is defined as follows: If $L = H$, then $\nabla_{H \leftarrow L} \stackrel{\text{def}}{=} \mathbf{I}$; otherwise, suppose the path in $\mathcal{T}$ from leaf L to node H is given by $(H_t \stackrel{\text{def}}{=} L, H_{t-1}, \ldots, H_1, H_0 \stackrel{\text{def}}{=} H)$, then

$$\nabla_{H \leftarrow L} \stackrel{\text{def}}{=} \nabla_{H_1} \cdots \nabla_{H_{t-1}} \nabla_{H_t}.$$

To maintain ∇, it will suffice to maintain ∇_H at each non-root node H in $\mathcal{T}$.

Intuitively, when applying an inverse tree operator to a vector $\boldsymbol{v} \in \mathbb{R}^E$, $\boldsymbol{v}$ is partitioned according to the leaves of $\mathcal{T}$, and then the edge operators are applied sequentially along the tree edges in a bottom-up fashion. It is natural to then also define the opposite process, where edge operators are applied along the tree edges in a top-down fashion.

Definition 3.36 (Tree operator)**.** Let $\mathcal{T}$ be an operator tree with the associated sets as above. We say a linear operator $\nabla : \mathbb{R}^V \mapsto \mathbb{R}^E$ is *tree operator supported on* $\mathcal{T}$ if there exists a linear edge operator ∇_H for each non-root node H in $\mathcal{T}$, corresponding to the edge from H to its parent, such that ∇ can be decomposed as

$$\boldsymbol{\Delta} \stackrel{\text{def}}{=} \sum_{\text{leaf } L, \text{ node } H \,:\, L \in \mathcal{T}_H} \boldsymbol{\Delta}_{L \leftarrow H} \mathbf{I}_{F_H}.$$

where $\boldsymbol{\Delta}_{H \leftarrow L}$ is defined as follows: If $L = H$, then $\boldsymbol{\Delta}_{L \leftarrow H} \stackrel{\text{def}}{=} \mathbf{I}$. Otherwise, suppose the path in $\mathcal{T}$ from node H to leaf L is given by $(H_t \stackrel{\text{def}}{=} L, H_{t-1}, \ldots, H_0 \stackrel{\text{def}}{=} H)$, then

$$\boldsymbol{\Delta}_{L \leftarrow H} \stackrel{\text{def}}{=} \boldsymbol{\Delta}_{H_t} \cdots \boldsymbol{\Delta}_{H_2} \boldsymbol{\Delta}_{H_1}.$$

We define the complexity of a tree (and inverse tree) operator to be parameterized by the number of edge operators applied.

Definition 3.37 (Query complexity)**.** Let $\mathbf{\Delta} \stackrel{\text{def}}{=} \{\mathbf{\Delta}_H : H \in \mathcal{T}\}$ be a tree (or inverse tree) operator on tree $\mathcal{T}$. Suppose for any set $\mathcal{H}$ of K distinct non-root nodes in $\mathcal{T}$, and any two families of K vectors indexed by $\mathcal{H}$, $\{\boldsymbol{u}_H : H \in \mathcal{H}\}$ and $\{\boldsymbol{v}_H : H \in \mathcal{H}\}$, the total time to compute $\{\boldsymbol{u}_H^\top \mathbf{\Delta}_H : H \in \mathcal{H}\}$ and $\{\mathbf{\Delta}_H \boldsymbol{v}_H : H \in \mathcal{H}\}$ is bounded by $f(K)$. Then we say $\mathbf{\Delta}$ has query complexity f for some function f.

Without loss of generality, we may assume $f(0) = 0$, $f(k) \geq k$, and f is concave.

By examining the definition of the inverse tree and tree operator, we see they are related.

Lemma 3.38. *If $\mathbf{\Delta}$ is a tree operator on $\mathcal{T}$, then $\mathbf{\Delta}^\top$ is an inverse tree operator on $\mathcal{T}$, where its edge operators are obtained from $\mathbf{\Delta}$'s edge operators by taking a transpose. Furthermore, $\mathbf{\Delta}$ and ∇ have the same query and update complexity.* □

First, we present some of the alternative decomposition properties of the tree operator.

Definition 3.39 (Subtree operator)**.** Let $\mathbf{\Delta}$ be a tree operator on $\mathcal{T}$. Recall $\mathcal{T}_H$ is the complete subtree of $\mathcal{T}$ rooted at H. We define the subtree operator $\mathbf{\Delta}^{(H)}$ at each node H to be

$$\mathbf{\Delta}^{(H)} \stackrel{\text{def}}{=} \sum_{\text{leaf } L \in \mathcal{T}_H} \mathbf{\Delta}_{L \leftarrow H}. \tag{3.11}$$

Corollary 3.40. *Based on the above definitions, we have*

$$\mathbf{\Delta} = \sum_{H \in \mathcal{T}} \mathbf{\Delta}^{(H)} \mathbf{I}_{F_H}. \tag{3.12}$$

Furthermore, if H has children H_1, H_2, then

$$\mathbf{\Delta}^{(H)} = \mathbf{\Delta}^{(H_1)} \mathbf{\Delta}_{H_1} + \mathbf{\Delta}^{(H_2)} \mathbf{\Delta}_{H_2}. \tag{3.13}$$

The output of $\mathbf{\Delta}$ when restricted to $E(H)$ for a node $H \in \mathcal{T}$ can be written in two parts, which is useful for our data structures. The first part involves summing over all nodes in $\mathcal{T}_H$, ie. descendants of H and H itself, and the second part involves a sum over all ancestors of H.

Lemma 3.41. *At any node $H \in \mathcal{T}$, we have*

$$\mathbf{I}_{E(H)} \mathbf{\Delta} = \sum_{D \in \mathcal{T}_H} \mathbf{\Delta}^{(D)} \mathbf{I}_{F_D} + \mathbf{\Delta}^{(H)} \sum_{\textit{ancestor } A \textit{ of } H} \mathbf{\Delta}_{H \leftarrow A} \mathbf{I}_{F_A}.$$

Proof. We consider the terms in the sum for $\boldsymbol{\Delta}$ that map into to $E(H)$, which is precisely the set of leaf nodes in the subtree rooted at H.

$$\mathbf{I}_{E(H)}\boldsymbol{\Delta} = \sum_{\text{leaf } L\in\mathcal{T}_H} \sum_{A: L\in\mathcal{T}_A} \boldsymbol{\Delta}_{L\leftarrow A}\mathbf{I}_{F_A}.$$

The right hand side involves a sum over the set $\{(L,A) \ : \ \text{leaf } L \in \mathcal{T}_H, L \in \mathcal{T}_A\}$. Observe that (L,A) is in this set if and only if A is a descendant of H, or $A = H$, or A is an ancestor of H. Hence, the summation can be written as

$$\sum_{\text{leaf } L\,\in\,\mathcal{T}_H} \sum_{\text{node } A\,\in\,\mathcal{T}_H} \boldsymbol{\Delta}_{L\leftarrow A}\mathbf{I}_{F_A} + \sum_{\text{leaf } L\,\in\,\mathcal{T}_H} \sum_{\text{ancestor } A \text{ of } H} \boldsymbol{\Delta}_{L\leftarrow A}\mathbf{I}_{F_A}.$$

The first term is precisely the first term in the lemma statement. For the second term, we can use the fact that A is an ancestor of H to expand $\boldsymbol{\Delta}_{L\leftarrow A} = \boldsymbol{\Delta}_{L\leftarrow H}\boldsymbol{\Delta}_{H\leftarrow A}$. Then, the second term is

$$\begin{aligned}
&\sum_{\text{leaf } L\,\in\,\mathcal{T}_H} \sum_{\text{ancestor } A \text{ of } H} \boldsymbol{\Delta}_{L\leftarrow H}\boldsymbol{\Delta}_{H\leftarrow A}\mathbf{I}_{F_A} \\
&= \sum_{\text{leaf } L\,\in\,\mathcal{T}_H} \boldsymbol{\Delta}_{L\leftarrow H} \left(\sum_{\text{ancestor } A \text{ of } H} \boldsymbol{\Delta}_{H\leftarrow A}\mathbf{I}_{F_A} \right) \\
&= \boldsymbol{\Delta}^{(H)} \left(\sum_{\text{ancestor } A \text{ of } H} \boldsymbol{\Delta}_{H\leftarrow A}\mathbf{I}_{F_A} \right),
\end{aligned}$$

by definition of $\boldsymbol{\Delta}^{(H)}$. □

Now, we consider the cost of applying the inverse tree operator and the tree operator.

Lemma 3.42. *Let $\nabla : \mathbb{R}^E \mapsto \mathbb{R}^V$ be an inverse tree operator on $\mathcal{T}$ with query complexity Q. Given $\boldsymbol{v} \in \mathbb{R}^E$, we can compute $\nabla\boldsymbol{v}$ as well as $\boldsymbol{y}_H \stackrel{\text{def}}{=} \sum_{\textit{leaf } L\in\mathcal{T}_H} \nabla_{H\leftarrow L}\boldsymbol{v}$ for all $H \in \mathcal{T}$ in $O(Q(\eta K))$ time, where $K = \operatorname{nnz}(\boldsymbol{v})$ and η is the height of $\mathcal{T}$.*

Proof. Recall the definition

$$\nabla\boldsymbol{v} \stackrel{\text{def}}{=} \sum_{\text{leaf } L} \left(\sum_{H:\, L\in\mathcal{T}_H} \mathbf{I}_{F_H}\nabla_{H\leftarrow L} \right) \boldsymbol{v}.$$

At a leaf node L, if we have $\boldsymbol{v}_e = 0$ for all $e \in E(L)$, then we can ignore the term for L in the outer sum. So we can reduce $\nabla\boldsymbol{v}$ to consist of at most K terms in the outer sum. We can further rearrange the order of applying the edge operators so that each edge operator is applied at most once, and this naturally gives the values for all non-zero $\boldsymbol{y}_H$'s. We bound the overall runtime loosely by $O(Q(\eta K))$. □

Unlike the inverse tree operator, the tree operator is applied downwards along a tree, and therefore we do not have non-trivial bounds on total number of edge operators applied. Instead, we have a more general bound:

Lemma 3.43. *Let $\mathbf{\Delta} : \mathbb{R}^V \mapsto \mathbb{R}^E$ be a tree operator on $\mathcal{T}$ with query complexity Q. Given $\boldsymbol{z} \in \mathbb{R}^V$, we can compute $\mathbf{\Delta}\boldsymbol{v}$ in $O(Q(|E|))$ time.*

Proof. We simply observe that we can compute $\mathbf{\Delta}\boldsymbol{v}$ by applying each edge operator at most once. Since the leaf nodes partition the set E, we know in $\mathcal{T}$, there are $O(|E|)$ edge operators in total, so the overall time is at most $O(Q(|E|))$. $\square$

With the appropriate partial computations taking advantage of the decomposition of ∇, we can maintain $\nabla\boldsymbol{v}$ efficiently for dynamic ∇ and $\boldsymbol{v}$. Specifically, we use the following property:

Lemma 3.44. *Given a vector $\boldsymbol{v} \in \mathbb{R}^E$, let $\boldsymbol{y}_H \stackrel{\text{def}}{=} \sum_{\text{leaf } L \in \mathcal{T}_H} \nabla_{H \leftarrow L}\boldsymbol{v}$ for each $H \in \mathcal{T}$. If H has children H_1, H_2, then*

$$\boldsymbol{y}_H = \nabla_{H_1}\boldsymbol{y}_{H_1} + \nabla_{H_2}\boldsymbol{y}_{H_2}. \tag{3.14}$$

Furthermore,

$$\sum_{H \in \mathcal{T}} \mathbf{I}_{F_H}\boldsymbol{y}_H = \nabla\boldsymbol{v}. \tag{3.15}$$

$\square$

Lemma 3.45. *Let $\nabla : \mathbb{R}^E \mapsto \mathbb{R}^V$ be an inverse tree operator with query complexity Q. Let $\nabla^{(\text{new})}$ be ∇ with K updated edge operators. Suppose we know $\nabla\boldsymbol{v}$, and we know $\boldsymbol{y}_H \stackrel{\text{def}}{=} \sum_{\text{leaf } L \in \mathcal{T}_H} \nabla_{H \leftarrow L}\boldsymbol{v}$ at all nodes H, then we can compute $(\nabla^{(\text{new})} - \nabla)\boldsymbol{v}$ and the $\boldsymbol{y}_H^{(\text{new})}$'s in $O(Q(\eta K))$ time.*

Proof. Observe that for a node $H \in \mathcal{T}$, if no edge operator in $\mathcal{T}_H$ was updated, then $\boldsymbol{y}_H$ remains the same. We use Eq. (3.14) to compute $\boldsymbol{y}_H^{(\text{new})}$ up the tree for the $O(\eta K)$-many nodes that admit changes, and then Eq. (3.15) to extract the change $(\nabla^{(\text{new})} - \nabla)\boldsymbol{v}$. $\square$

Chapter 4

IPM PROJECTION AS TREE OPERATOR

In this chapter, we explore the separable structure of the dual graph $G_{\mathbf{A}}$ of the LP constraint matrix $\mathbf{A}$, and use these properties to help define and maintain the tree operator and inverse tree operator as needed for the IPM framework.

Throughout this section, we fix $\mathbf{A} \in \mathbb{R}^{n \times m}$, so that the dual graph $G_{\mathbf{A}} = (V, E)$ has n vertices, m hyperedges. Additionally, let ρ denote the max hyperedge size in $G_{\mathbf{A}}$; equivalently, ρ is the column sparsity of $\mathbf{A}$.

4.1 Exact projection operator definition

Suppose $\mathcal{S}$ is a separator tree for $G_{\mathbf{A}}$. In this subsection, we define the operator tree $\mathcal{T}$ based on $\mathcal{S}$, followed by the tree operator $\mathbf{\Delta}$ and inverse tree operator ∇ which will be supported on $\mathcal{T}$. Finally, we will show that our definitions indeed satisfy

$$\mathbf{W}^{1/2}\mathbf{P}_{\boldsymbol{w}} = \mathbf{\Delta}\nabla.$$

Recall that $\mathcal{S}$ is a constant-degree tree. The leaf nodes of $\mathcal{S}$ partition the hyperedges of $G_{\mathbf{A}}$, however, we do not have a bound on the number of hyperedges in a leaf node. In constructing $\mathcal{T}$, we simply want to modify $\mathcal{S}$ so that each leaf contains exactly one hyperedge. Specifically, for each leaf node $H \in \mathcal{S}$ containing $|E(H)|$ hyperedges, we construct a complete binary tree $\mathcal{T}_H^+$ rooted at H with $|E(H)|$ leaves, assign one hyperedge from $E(H)$ to one new leaf, and attach $\mathcal{T}_H^+$ at the node H. This construction yields the desired operator tree $\mathcal{T}$ whose height is within a $\log |E|$ factor of $\mathcal{S}$.

We define the tree operator $\mathbf{\Delta}$ on $\mathcal{T}$ follows: For non-root node H in $\mathcal{T}$, let

$$\mathbf{\Delta}_H \overset{\text{def}}{=} \begin{cases} \mathbf{I}_{F_H \cup \partial H} - \mathbf{X}^{(H)\top} & \text{if } H \text{ exists in } \mathcal{S} \\ {\mathbf{W}_{E(H)}}^{1/2} {\mathbf{A}_H}^{\top} & \text{if } H \text{ is a leaf node in } \mathcal{T} \\ \mathbf{I} & \text{else.} \end{cases} \tag{4.1}$$

Note that the first two cases are indeed disjoint by construction. We pad zeros to all matrices in order to arrive at the correct overall dimensions.

Lemma 4.1. *Let $\mathbf{\Delta}$ be the tree operator as defined above. Then*

$$\mathbf{\Delta} = \mathbf{W}^{1/2}\mathbf{A}^\top \mathbf{\Pi}^{(\eta)\top} \cdots \mathbf{\Pi}^{(1)\top}, \tag{4.2}$$

where the $\mathbf{\Pi}^{(i)}$'s are defined in Theorem 3.30.

Note that this follows directly from the definitions in Theorem 3.30.

Next, we establish the query complexity of the tree operator:

Lemma 4.2. *Suppose L is the total number of leaf nodes in $\mathcal{S}$. The query complexity of $\mathbf{\Delta}$ is*

$$Q(K) = O\left(\rho K + \max_{\mathcal{H}\text{:set of } K \text{ leaves in } \mathcal{S}} \sum_{H \in \mathcal{P}_{\mathcal{S}}(\mathcal{H})} |F_H \cup \partial H|^2\right)$$

for $K \leq L$, where $\mathcal{P}_{\mathcal{S}}(\mathcal{H})$ is the set of all nodes in $\mathcal{S}$ that are ancestors of some node in $\mathcal{H}$ unioned with $\mathcal{H}$. When $K > L$, then we define $Q(K) = Q(L)$.

Proof. First, we consider the query time $Q(1)$ for a single edge. Let $\boldsymbol{u}$ be any vector, and let H be a non-root node in $\mathcal{T}$. If H is a leaf node, then computing $\mathbf{\Delta}_H \boldsymbol{u}$ and $\boldsymbol{u}^\top \mathbf{\Delta}_H$ both take $O(\rho)$ time. If H exists in $\mathcal{S}$, then computing $\mathbf{\Delta}_H \boldsymbol{u}$ takes $O\left(|F_H|^2 + |\partial H||F_H|\right) \leq O\left(|F_H \cup \partial H|^2\right)$ time, since the bottleneck is naively computing $\mathbf{L}^{(H)}_{\partial H, F_H}\left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1}\boldsymbol{u}$. Therefore, $Q(1) = O\left(\rho + \max_{H\in\mathcal{S}} |F_H \cup \partial H|^2\right)$.

For $K > 1$, we can simply bound the query time for K distinct edges by

$$Q(K) = O\left(\rho K + \max_{\mathcal{H}\text{:set of } K \text{ nodes in } \mathcal{S}} \sum_{H\in\mathcal{H}} |F_H \cup \partial H|^2\right).$$

Finally, note that we can take the summation over $H \in \mathcal{P}_{\mathcal{S}}(\mathcal{H})$ instead of $H \in \mathcal{H}$ for an upper bound. In this case, it suffices to take the max over sets of leaf nodes. □

By taking the transpose of $\mathbf{\Delta}$, we get an inverse tree operator, and together, they give the projection matrix using Eq. (3.3).

Corollary 4.3. *Let $\nabla \overset{\text{def}}{=} \mathbf{\Delta}^\top$ be the inverse tree operator obtained from $\mathbf{\Delta}$ by transposing the edge and leaf operators. Then*

$$\mathbf{W}^{1/2}\mathbf{P}_{\boldsymbol{w}} \overset{\text{def}}{=} \mathbf{W}^{1/2}\mathbf{W}^{1/2}\mathbf{A}^\top\mathbf{L}^{-1}\mathbf{A}\mathbf{W}^{1/2} = (\mathbf{W}^{1/2}\mathbf{\Delta})\mathbf{\Gamma}\nabla. \tag{4.3}$$

□

Remark 4.4. Without loss of generality, we have chosen to simplify our presentation and consider $\mathbf{\Delta}\nabla$ in place of $\mathbf{W}^{1/2}\mathbf{\Delta}\mathbf{\Gamma}\nabla$.

This is possible for two reasons: One, $\mathbf{W}^{1/2}\mathbf{\Delta}$ is a tree operator, which we can in fact maintain in the same time complexity as $\mathbf{\Delta}$. Two, $\mathbf{\Gamma}$ is a block-diagonal matrix, with a block for each $H \in \mathcal{T}$ that is indexed by F_H. It is straightforward to show we can maintain and apply $\mathbf{\Gamma}\nabla$ in the same time complexity as ∇.

4.2 Complexities

In this subsection, we summarize the runtime complexities for the tree operator, in the special case when $\mathcal{S}$ is a (a, b, λ)-separator tree for $G_{\mathbf{A}}$. Parametrizing the separator tree this way allows us to write the runtime expressions using geometric series. For non-negative x, we use the standard bound $\sum_{i=\ell}^{u} x^i \leq O(x^\ell + x^u)$. When it is clear $x < 1$, we bound $\sum_{i=\ell}^{u} x^i \leq O(x^\ell)$.

Lemma 4.5. *Suppose $\mathcal{S}$ is an (a, b, λ)-separator tree for $G_{\mathbf{A}}$ on n vertices, m edges, with max hyperedge size ρ, where $a \in [0, 1]$ and $b \in (0, 1)$. Let η denote the height of $\mathcal{S}$, and let L denote the number of leaf nodes. Let $\mathbf{\Delta}$ be the tree operator on $\mathcal{T}$ as defined in Section 4.1. Then there is a data structure to maintain $\mathbf{\Delta}$ as a function of the weights $\boldsymbol{w}$ throughout* Solve*, so that:*

- *The data structure can be initialize in time*

$$O\left(\rho^{\omega-1}m + \lambda^\omega \cdot \left(1 + (b^{a\omega-1})^\eta\right)\right). \tag{4.4}$$

- *The query complexity of $\mathbf{\Delta}$ is*

$$Q(K) = O\left(\rho K + \lambda^2\left(1 + (\min\{K, L\})^{1-2a}\right)\right) \tag{4.5}$$

- *When $a < 1$, the update complexity of $\mathbf{\Delta}$ is $U(K) =$*

$$\rho^{\omega-1}K + \lambda^2 \min\{K, \lambda\}^{\omega-2} + \begin{cases} \lambda^2 K^{1-2a} & \text{if } K \leq \lambda \\ \lambda^2 K^{1-2a} + \lambda^{\frac{\omega-1}{1-\alpha}} K^{\frac{1-\alpha\omega}{1-\alpha}} & \text{if } \lambda < K \leq \lambda \cdot b^{(a-1)\eta} \\ \lambda^\omega \cdot b^{(a\omega-1)\eta} & \text{if } K > \lambda \cdot b^{(a-1)\eta}. \end{cases} \tag{4.6}$$

  When $a = 1$, the update complexity is $U(K) = \rho^{\omega-1}K + \lambda^2 \min\{K, \lambda\}^{\omega-2}$.

Proof. The data structure we use to maintain $\mathbf{\Delta}$ is precisely the data structure DynamicSC with respect to $\mathcal{S}$.

Initialization time. We use the runtime expression for INITIALIZE in DYNAMICSC (Lemma 4.7) combined with the parameters of the (a, b, λ)-separator tree. For any H, we have $|F_H \cup \partial H| \leq \rho$, so $\sum_{\text{leaf } H \in \mathcal{S}} |E(H)| \cdot |F_H \cup \partial H|^{\omega-1} \leq \rho^{\omega-1} m$. Moreover,

$$\sum_{H \in \mathcal{S}} |F_H \cup \partial H|^{\omega} \leq \sum_{i=0}^{\eta} b^{-i} \left(\lambda \cdot b^{ai}\right)^{\omega} \leq O(\lambda^{\omega}) \cdot \left[1 + (b^{a\omega-1})^{\eta}\right].$$

Query complexity. We substitute the (a, b, λ)-separator tree bounds in Lemma 4.2, to conclude that the query complexity of $\mathbf{\Delta}$ is

$$Q(K) = O\left(\rho K + \sum_{H \in \mathcal{P}_{\mathcal{S}}(\mathcal{H})} (\lambda \cdot b^{ai})^2\right),$$

where $\mathcal{H}$ is any set of K leaf nodes in $\mathcal{S}$. We group terms according to their node level, and note that there are $\min\{K, b^{-i}\}$-many terms at any level i, so

$$= O\left(\rho K + \sum_{i=0}^{\eta} \min\{K, b^{-i}\} \cdot (\lambda \cdot b^{ai})^2\right)$$
$$= O(\rho K) + O(\lambda^2) \cdot \left(\sum_{i=0}^{-\log_b K} b^{-i} \cdot b^{2ai} + K \cdot \sum_{i=-\log_b K}^{\eta} b^{2ai}\right)$$

Note that K can be at most L in the summation, so we have

$$= O\left(\rho K + \lambda^2 (1 + (\min\{K, L\})^{1-2a})\right).$$

Update complexity. When $\boldsymbol{w}$ changes, we update $\mathbf{\Delta}$ by invoking REWEIGHT$(\delta_{\boldsymbol{w}})$ in DYNAMICSC, where $\delta_{\boldsymbol{w}}$ denotes the change in $\boldsymbol{w}$. By Lemma 4.7, the runtime for the fixed $\delta_{\boldsymbol{w}}$ is

$$\sum_{\text{leaf } H \colon \delta_{\boldsymbol{w}}|_{E(H)} \neq \mathbf{0}} \operatorname{nnz}(\delta_{\boldsymbol{w}}|_{E(H)}) \cdot |F_H \cup \partial H|^{\omega-1} + \sum_{\text{node } H \colon \delta_{\boldsymbol{w}}|_{E(H)} \neq \mathbf{0}} |F_H \cup \partial H|^2 \cdot {K_H}^{\omega-2}. \tag{4.7}$$

Hence, the update complexity of $\mathbf{\Delta}$ is the max of the above expression taken over all choices of $\delta_{\boldsymbol{w}}$. For any leaf node H, we upper bound $|F_H \cup \partial H|^{\omega-1} \leq \rho^{\omega-1}$, and therefore the first summation is at most $\rho^{\omega-1} K$.

For the second summation, we substitute in the (a, b, λ)-separator tree bounds, and group terms according to their node level. Let $\mathcal{S}(i)$ denotes all nodes at level i in $\mathcal{S}$. Then for any $\delta_{\boldsymbol{w}}$, we have

$$\sum_{\text{node } H \colon \delta_{\boldsymbol{w}}|_{E(H)} \neq \mathbf{0}} |F_H \cup \partial H|^2 \cdot {K_H}^{\omega-2} \leq \sum_{i=0}^{\eta} \left((\lambda \cdot b^{ai})^2 \cdot \sum_{H \in \mathcal{S}(i)} {K_H}^{\omega-2}\right), \tag{4.8}$$

where the K_H's are non-negative integers satisfying $\sum_{H\in\mathcal{S}(i)} K_H \le K$ and $K_H \le |F_H \cup \partial H| \le \lambda \cdot b^{ai}$ for $H \in \mathcal{S}(i)$. We are interested in upper bounding Eq. (4.8). At any level i, there are b^{-i} nodes, and the sum is maximized when all the K_H's are equal. Depending on the relationship between K and the level i, we have the following three cases:

- If $K \le b^{-i}$, that is, the total update rank is less than the number of nodes at the level, then the sum is maximized if $K_H = 1$ for K-many nodes, and $K_H = 0$ for the rest.
- If $b^{-i} < K \le b^{-i} \cdot (\lambda \cdot b^{ai})$, the sum is upper bounded by setting $K_H = K/b^{-i}$.
- If $K > O(b^{-i}) \cdot (\lambda \cdot b^{ai})$, the sum is upper bounded by setting $K_H = \lambda \cdot b^{ai}$.

Then, we can bound the summation term in Eq. (4.8) by

$$\begin{aligned}&\sum_{\substack{0\le i\le\eta\\ K\le b^{-i}}} K(\lambda\cdot b^{ai})^2 + \sum_{\substack{0\le i\le \eta:\\ b^{-i}<K\le\lambda\cdot b^{(a-1)i}}} (\lambda\cdot b^{ai})^2\cdot b^{-i}\cdot (Kb^i)^{\omega-2} + \sum_{\substack{0\le i\le\eta:\\ K>\lambda\cdot b^{(a-1)i}}} b^{-i}\cdot(\lambda\cdot b^{ai})^{\omega}\\ \le\ & \lambda^2 K \sum_{i=-\log_b K}^{\eta} b^{2ai} + \lambda^2 K^{\omega-2} \sum_{i=\frac{\log_b(K/\lambda)}{a-1}}^{-\log_b K} b^{(2a+\omega-3)i} + \lambda^{\omega} \sum_{i=0}^{\frac{\log_b(K/\lambda)}{a-1}} b^{(a\omega-1)i}.\end{aligned}$$

We need to further consider different cases for the possible values of K, which affects the summation indices. If $\log_b(K/\lambda) < 0$, i.e. $K < \lambda$, the expression simplifies to

$$\lambda^2 K^{1-2a} + \lambda^2 K^{\omega-2}.$$

If $0 \le \log_b(K/\lambda)/(a-1) \le \eta$, i.e. $\lambda \le K \le \lambda\cdot b^{(a-1)\eta}$, the expression simplifies to

$$\lambda^2 K^{1-2a} + \lambda^{\frac{\omega-1}{1-\alpha}} K^{\frac{1-\alpha\omega}{1-\alpha}} + \lambda^{\omega}.$$

And lastly, if $\log_b(K/\lambda)/(a-1) > \eta$, i.e $K > \lambda\cdot b^{(a-1)\eta}$, the expression simplifies to

$$\lambda^{\omega} + \lambda^{\omega}\cdot b^{(a\omega-1)\eta}.$$

We combine the cases to arrive at the overall update complexity, having implicitly assumed that $\alpha < 1$. When $\alpha = 1$, the summation in Eq. (4.8) is maximized when $K_H = \min\{K,\lambda\}\cdot b^i$ for H at level i. Then we can upper bound the summation term by

$$\sum_{i=0}^{\eta} (\lambda\cdot b^i)^2\cdot b^{-i}\cdot(\min\{K,\lambda\}\cdot b^i)^{\omega-2} \le O(\lambda^2 \min\{K,\lambda\}^{\omega-2}).$$

□

4.3 Dynamic nested dissection

So far, we have defined the separator tree $\mathcal{S}$ for the graph $G_{\mathbf{A}}$, which we then used to define the operator tree $\mathcal{T}$ that supports the tree operator $\mathbf{\Delta}$ needed for the IPM framework. Since the tree operator is dynamic across the IPM, we discuss how to update it efficiently in this section.

4.3.1 Exact version

Let us consider how to maintain $\mathbf{L}^{(H)}, \mathbf{Sc}(\mathbf{L}^{(H)}, \partial H)$, and $(\mathbf{L}^{(H)}_{F_H,F_H})^{-1}$ at each node $H \in \mathcal{S}$ using the data structure DynamicSC (Algorithm 7), as the weight vector $\boldsymbol{w}$ undergoes changes throughout the IPM. This will in turn allow us to maintain the tree operator $\mathbf{\Delta}$.

We begin with a lemma showing that given a symmetric matrix and a low-rank update, we can compute its new inverse and Schur complement quickly.

Lemma 4.6. *Let $\mathbf{L}' = \mathbf{L} + \mathbf{UV} \in \mathbb{R}^{n\times n}$ be a symmetric matrix plus a rank-K update, where $\mathbf{U}$ and $\mathbf{V}^\top$ both have dimensions $n \times K$. Given $\mathbf{L}', \mathbf{U}, \mathbf{V}$, we can compute $\mathbf{L}'^{-1}$ in $O(n^2K^{\omega-2})$ time.*

Additionally, suppose we are also given $\mathbf{L}^{-1}$ and $\mathbf{Sc}(\mathbf{L}, S)$ for an index set S. Then we can compute $\mathbf{Sc}(\mathbf{L}', S)$, $\mathbf{U}', \mathbf{V}'$ in $O(n^2K^{\omega-2})$ time, so that $\mathbf{Sc}(\mathbf{L}, S) + \mathbf{U}'\mathbf{V}' = \mathbf{Sc}(\mathbf{L}', S)$, and $\mathbf{U}', \mathbf{V}'^\top$ both have K columns.

Proof. The Sherman-Morrison formula states

$$\mathbf{L}'^{-1} = \mathbf{L}^{-1} - \mathbf{L}^{-1}\mathbf{U}(\mathbf{I}_K + \mathbf{VL}^{-1}\mathbf{U})^{-1}\mathbf{VL}^{-1}.$$

The time to compute this update is dominated by the time required to multiply an $n \times n$ matrix with an $n \times K$ matrix, which is $O(n^2K^{\omega-2})$.

For the second part of the lemma, recall that the Schur complement is defined to be:

$$\mathbf{Sc}(\mathbf{L}, C) \stackrel{\text{def}}{=} \mathbf{L}_{C,C} - \mathbf{L}_{C,F}\mathbf{L}^{-1}_{F,F}\mathbf{L}_{F,C}. \tag{4.9}$$

If we were to naively use this definition of the Schur complement to perform the updates and construct $\mathbf{U}'$ and $\mathbf{V}'^\top$, we will run into an issue where the rank of the new update blows up by a factor of 8, leading to an exponential blowup in the rank as we go up the levels recursively. Instead, we make use of the fact that the inverse of the Schur complement, $\mathbf{Sc}(\mathbf{L}, S)^{-1}$ is exactly the S, S submatrix of $\mathbf{L}^{-1}$ to control the rank of the updates.

We first apply the definition of Schur complement and then use the Sherman-Morrison formula to get

$$\begin{aligned}\mathbf{Sc}(\mathbf{L}', S)^{-1} &= {\mathbf{L}'^{-1}}_{S,S} \\ &= {\mathbf{L}^{-1}}_{S,S} - \left(\mathbf{L}^{-1}\mathbf{U}(\mathbf{I}_K + \mathbf{V}\mathbf{L}^{-1}\mathbf{U})^{-1}\mathbf{V}\mathbf{L}^{-1}\right)_{S,S} \\ &= \mathbf{Sc}(\mathbf{L}, S)^{-1} - \mathbf{I}_S\mathbf{L}^{-1}\mathbf{U}(\mathbf{I}_K + \mathbf{V}\mathbf{L}^{-1}\mathbf{U})^{-1}\mathbf{V}\mathbf{L}^{-1}\mathbf{I}_S.\end{aligned}$$

This gives us the new rank-K update $\mathbf{Sc}(\mathbf{L}', S)^{-1} = \mathbf{Sc}(\mathbf{L}, S)^{-1} + \mathbf{U}^*\mathbf{V}^*$ with

$$\begin{aligned}\mathbf{U}^* &= -\mathbf{I}_S\mathbf{L}^{-1}\mathbf{U} \\ \mathbf{V}^* &= (\mathbf{I}_K + \mathbf{V}\mathbf{L}^{-1}\mathbf{U})^{-1}\mathbf{V}\mathbf{L}^{-1}\mathbf{I}_S.\end{aligned}$$

We can now determine the Schur complement update by applying Sherman-Morrison again:

$$\mathbf{Sc}(\mathbf{L}', S) = \mathbf{Sc}(\mathbf{L}, S) - \mathbf{Sc}(\mathbf{L}, S)\mathbf{U}^*(\mathbf{I}_K + \mathbf{V}^*\mathbf{Sc}(\mathbf{L}, S)\mathbf{U}^*)^{-1}\mathbf{V}^*\mathbf{Sc}(\mathbf{L}, S).$$

This is a rank-K update $\mathbf{Sc}(\mathbf{L}', S) = \mathbf{Sc}(\mathbf{L}, S) + \mathbf{U}'\mathbf{V}'$ with

$$\begin{aligned}\mathbf{U}' &= -\mathbf{Sc}(\mathbf{L}, S)\mathbf{U}^* \\ \mathbf{V}' &= (\mathbf{I}_K + \mathbf{V}^*\mathbf{Sc}(\mathbf{L}, S)\mathbf{U}^*)^{-1}\mathbf{V}^*\mathbf{Sc}(\mathbf{L}, S).\end{aligned}$$

The time to compute $\mathbf{U}^*, \mathbf{V}^*, \mathbf{U}', \mathbf{V}'$ are all dominated by the time to multiply an $n \times n$ matrix with an $n \times K$ matrix, which is $O(n^2 K^{\omega-2})$. $\square$

Now, we are ready to present the data structure for maintaining the Schur complement matrices along a separator tree.

Lemma 4.7. *Let $\boldsymbol{w}$ be the weights changing at every step of the IPM. Let $\mathcal{S}$ be any separator tree for $G_{\mathbf{A}}$. Recall $G_{\mathbf{A}}$ has n vertices, m hyperedges, and max hyperedge size ρ. Then the data structure* DynamicSC *(Algorithm 7) correctly maintains the matrices $\mathbf{L}^{(H)}, (\mathbf{L}^{(H)}_{F_H,F_H})^{-1}, \mathbf{Sc}(\mathbf{L}^{(H)}, \partial H)$ at every node $H \in \mathcal{S}$ dependent on $\boldsymbol{w}$ throughout the IPM. The data structure supports the following procedures and runtimes:*

- Initialize$(\mathcal{S}, \boldsymbol{w}^{(\mathrm{init})} \in \mathbb{R}^m)$*: Set $\boldsymbol{w} \leftarrow \boldsymbol{w}^{(\mathrm{init})}$, and compute all matrices with respect to $\boldsymbol{w}$, in time*
$$O\left(\sum_{\textit{leaf } H \in \mathcal{S}} |E(H)| \cdot |F_H \cup \partial H|^{\omega-1} + \sum_{H \in \mathcal{S}} |F_H \cup \partial H|^{\omega}\right).$$
- Reweight$(\boldsymbol{\delta_w} \in \mathbb{R}^m)$*: Update the weight vector to $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{\delta_w}$, and update all the maintained matrices with respect to the new weights, in time*
$$O\left(\sum_{\textit{leaf } H \in \mathcal{H}} \mathrm{nnz}(\boldsymbol{\delta_w}|_{E(H)}) \cdot |F_H \cup \partial H|^{\omega-1} + \sum_{H \in \mathcal{H}} |F_H \cup \partial H|^2 \cdot K_H^{\omega-2}\right).$$

Algorithm 7 Data structure to maintain dynamic Schur complements

```
1: data structure DynamicSC
2: private: member
3:     Hypergraph $G_{\mathbf{A}}$ with incidence matrix $\mathbf{A}$
4:     $\boldsymbol{w} \in \mathbb{R}^m$: Dynamic weight vector
5:     $\mathcal{S}$: Separator tree of height $\eta$. Every node $H$ of $\mathcal{S}$ stores:
6:         $F_H$, $\partial H$: Sets of eliminated vertices and boundary vertices of region $H$
7:         $E(H)$: Set of hyperedges of region $H$
8:         $\mathbf{L}^{(H)}, (\mathbf{L}^{(H)}_{F_H,F_H})^{-1}, \mathbf{Sc}(\mathbf{L}^{(H)}, \partial H)$,: Matrices to maintain as a function of $\boldsymbol{w}$
9:         ${\mathbf{L}^{(H)}}^{-1}$: Additional inverse matrix to maintain as a function of $\boldsymbol{w}$
10:        $\mathbf{U}_H, \mathbf{V}_H$: Low-rank update at $H$, used in Reweight

11: procedure Initialize($\mathcal{S}$, $\boldsymbol{w}^{(\text{init})} \in \mathbb{R}^m$)
12:     $\mathcal{S} \leftarrow \mathcal{S}, \boldsymbol{w} \leftarrow \boldsymbol{w}^{(\text{init})}$
13:     for level $i = \eta$ to 0 do
14:         for each node $H$ at level $i$ do
15:             $\mathbf{L}^{(H)}, (\mathbf{L}^{(H)}_{F_H,F_H})^{-1}, \mathbf{Sc}(\mathbf{L}^{(H)}, \partial H) \leftarrow \mathbf{0}, \mathbf{0}, \mathbf{0}$
16:             SchurNode($H, \boldsymbol{w}$)
17:         end for
18:     end for
19: end procedure

20: procedure Reweight($\boldsymbol{\delta_w} \in \mathbb{R}^m$)
21:     $\mathcal{H} \leftarrow$ set of nodes $H$ in $\mathcal{S}$ where $\boldsymbol{\delta_w}|_{E(H)} \neq \mathbf{0}$
22:     for level $i = \eta$ to 0 do
23:         for each node $H \in \mathcal{H}$ at level $i$ do
24:             SchurNode($H, \boldsymbol{\delta_w}$)
25:         end for
26:     end for
27:     $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{\delta_w}$
28: end procedure
```

Algorithm 7 Data structure to maintain dynamic Schur complements

```
29: data structure DynamicSC
30: procedure SchurNode($H \in \mathcal{S}$, $\boldsymbol{\delta_w} \in \mathbb{R}^m$)
31:     if $H$ is a leaf node then          ▷ rank of update $\leq \min\{\mathrm{nnz}(\boldsymbol{\delta_w}|_{E(H)}), |F_H \cup \partial H|\}$
32:         $\mathbf{L}^{(H)} \leftarrow \mathbf{L}^{(H)} + \mathbf{A}_H \mathrm{diag}(\boldsymbol{\delta_w}|_{E(H)}) \mathbf{A}_H^\top$
33:     else if $\mathrm{nnz}(\boldsymbol{\delta_w}|_{E(H)}) \leq |F_H \cup \partial H|$ then
                                        ▷ rank of update $\leq \sum_{\text{child } D} K_D \leq \mathrm{nnz}(\boldsymbol{\delta_w}|_{E(H)})$
34:         $\mathbf{L}^{(H)} \leftarrow \mathbf{L}^{(H)} + \sum_{\text{child } D \text{ of } H} \mathbf{U}_D \mathbf{V}_D$
35:     else                                ▷ rank of update $\leq |F_H \cup \partial H|$
36:         $\mathbf{L}^{(H)} \leftarrow \sum_{\text{child } D \text{ of } H} \mathbf{Sc}(\mathbf{L}^{(D)}, \partial D)$
37:     end if
38:     Let $K_H \stackrel{\text{def}}{=} \min\{\mathrm{nnz}(\boldsymbol{\delta_w}|_{E(H)}), |F_H \cup \partial H|\}$
                                        ▷ $K_H$ is upper bound on the rank of update to $\mathbf{L}^{(H)}$
39:     Compute $(\mathbf{L}^{(H)}_{F_H,F_H})^{-1}$ and $\mathbf{L}^{(H)^{-1}}$ by Lemma 4.6
40:     Compute $\mathbf{Sc}(\mathbf{L}^{(H)}, \partial H)$ and its rank-$K_H$ update factorization $\mathbf{U}_H, \mathbf{V}_H$ by Lemma 4.6
41: end procedure
```

where $\mathcal{H}$ is the set of nodes H with $\boldsymbol{\delta_w}|_{E(H)} \neq \mathbf{0}$, and $K_H \stackrel{\text{def}}{=} \min\{\mathrm{nnz}(\boldsymbol{\delta_w}|_{E(H)}), |F_H \cup \partial H|\}$.

Proof. INITIALIZE is a special case of REWEIGHT, where the change in the weight vector is from $\mathbf{0}$ to $\boldsymbol{w}^{(\text{init})}$, so we focus on a single call of REWEIGHT.

It suffices for REWEIGHT visits only nodes in $\mathcal{H}$, since if none of the edges in a region admits a weight update, then the matrices stored at the node remain the same by definition. Also note that $H \in \mathcal{H}$ implies all ancestors of H are also in $\mathcal{H}$.

Correctness. We use the superscript $^{(\text{new})}$ on $\mathbf{L}^{(H)}$ to indicate that it is computed with respect to the new weights, and $^{(\text{old})}$ otherwise. Recall that $\mathbf{L}^{(H)}$ is supported on $F_H \cup \partial H$.

We maintain some additional matrices at each node, in order to efficiently compute low-rank updates. Specifically, we use helper matrices $\mathbf{U}_H, \mathbf{V}_H$ at H, and guarantee that during a single REWEIGHT($\boldsymbol{\delta_w}$) call, after SCHURNODE($H, \boldsymbol{\delta_w}$) is run, they satisfy $\mathbf{Sc}(\mathbf{L}^{(H)^{(\text{old})}}, \partial H) = \mathbf{Sc}(\mathbf{L}^{(H)^{(\text{new})}}, \partial H) + \mathbf{U}_H \mathbf{V}_H$, and $\mathbf{U}_H, {\mathbf{V}_H}^\top$ both have at most K_H-many columns.

Now, we show inductively that after SCHURNODE($H, \boldsymbol{\delta_w}$) is run, all matrices at H, as well as all matrices at all descendants of H, are updated correctly: When H is leaf node, recall $\mathbf{L}^{(H)}$ is defined to be $\mathbf{L}[H] \stackrel{\text{def}}{=} \mathbf{A}_H \mathbf{W}_{E(H)} {\mathbf{A}_H}^\top$, so clearly SCHURNODE updates $\mathbf{L}^{(H)}$ correctly, and the rank of the update is at most K_H. The remaining matrices at H are computed correctly by Lemma 4.6.

Inductively, when H is a non-leaf node, the recursive property of Schur complements (c.f. [56, Lemma 18]) allows us to write $\mathbf{L}^{(H)^{(\text{new})}} = \sum_{\text{child } D \text{ of } H} \mathbf{Sc}(\mathbf{L}^{(D)^{(\text{new})}}, \partial D)$ at every node $H \in \mathcal{S}$. This formula trivially shows that the update $\mathbf{L}^{(H)^{(\text{new})}} - \mathbf{L}^{(H)^{(\text{old})}}$ has rank $|F_H \cup \partial H|$ (ie. full-rank). Alternatively, if $\text{nnz}(\boldsymbol{\delta_w}|_{E(H)}) \leq |F_H \cup \partial H|$, then by the guarantees on the helper matrices, we have

$$\begin{aligned}
\mathbf{L}^{(H)^{(\text{new})}} &= \sum_{\text{child } D \text{ of } H} \mathbf{Sc}(\mathbf{L}^{(D)^{(\text{new})}}, \partial D) \\
&= \sum_{\text{child } D \text{ of } H} \mathbf{Sc}(\mathbf{L}^{(D)^{(\text{old})}}, \partial D) + \mathbf{U}_D \mathbf{V}_D \\
&= \mathbf{L}^{(H)^{(\text{old})}} + \sum_{\text{child } D \text{ of } H} \mathbf{U}_D \mathbf{V}_D.
\end{aligned}$$

This gives a low-rank factorization of the update $\mathbf{L}^{(H)^{(\text{new})}} - \mathbf{L}^{(H)^{(\text{old})}}$ with rank at most $\sum_{\text{child } D} K_D$, which we can show by induction is at most $\text{nnz}(\boldsymbol{\delta_w}|_{E(H)})$. Since we have the correct low-rank update to $\mathbf{L}^{(H)}$, the remaining matrices at H again are computed correctly by Lemma 4.6.

This completes the correctness proof.

Runtime. Consider the runtime of the procedure SCHURNODE$(H, \boldsymbol{\delta_w})$ at a node H: If H is a leaf node, then computing the update to $\mathbf{L}^{(H)}$ involves multiplying a $|F_H \cup \partial H| \times \text{nnz}(\boldsymbol{\delta_w}|_{E(H)})$-sized matrix with its transpose (Line 31). Note that if $|F_H \cup \partial H| > \text{nnz}(\boldsymbol{\delta_w}|_{E(H)})$, then this runtime can be absorbed into the runtime expression for the remaining steps of the procedure, since $K_H = \text{nnz}(\boldsymbol{\delta_w}|_{E(H)})$. Otherwise, we use fast matrix multiplication which takes $O(\text{nnz}(\boldsymbol{\delta_w}|_{E(H)}) \cdot |F_H \cup \partial H|^{\omega-1})$ time. If H is a non-leaf node, there are two cases for the update to $\mathbf{L}^{(H)}$ in the algorithm. The first case (Line 33) takes $O\left(|F_H \cup \partial H| \cdot (\sum K_D)\right) \leq O(|F_H \cup \partial H| \cdot K_H)$ time, and the second case (Line 35) takes $O(|F_H \cup \partial H|^2)$ time. Computing the other matrices at any node H takes $O\left(|F_H \cup \partial H|^2 \cdot {K_H}^{\omega-2}\right)$ time by Lemma 4.6.

The runtime of REWEIGHT$(\boldsymbol{\delta_w})$ is therefore given by

$$\begin{aligned}
&\sum_{H \in \mathcal{H}} \text{SCHURNODE}(H, \boldsymbol{\delta_w}) \text{ time} \\
&= O\left(\sum_{\text{leaf } H \in \mathcal{H}} \text{nnz}(\boldsymbol{\delta_w}|_{E(H)}) \cdot |F_H \cup \partial H|^{\omega-1} + \sum_{H \in \mathcal{H}} |F_H \cup \partial H|^2 \cdot K_H^{\omega-2}\right).
\end{aligned}$$

For INITIALIZE, we further simplify the expression using $\text{nnz}(\boldsymbol{\delta_w}|_{E(H)}) = |E(H)|$ and $K_H \leq |F_H \cup \partial H|$. □

4.3.2 Approximate version

In this section, we prove the analogous results as the previous section, except we maintain *approximate* Schur complements at each node H in $\mathcal{T}$. We use the following result as a black-box for computing sparse approximate Schur complements:

Lemma 4.8 (ApproxSchur procedure [63]). *Let $\mathbf{L}$ be the weighted Laplacian of a graph with n vertices and m edges, and let C be a subset of boundary vertices of the graph. Let $\gamma = 1/n^3$ be the error tolerance. Given approximation parameter $\varepsilon \in (0, 1/2)$, there is an algorithm* ApproxSchur$(\mathbf{L}, C, \varepsilon)$ *that computes and outputs a ε-approximate Schur complement $\widetilde{\mathbf{Sc}}(\mathbf{L}, C)$ that satisfies the following properties with probability at least $1 - \gamma$:*

1. *The graph corresponding to $\widetilde{\mathbf{Sc}}(\mathbf{L}, C)$ has $O(\varepsilon^{-2}|C| \log(n/\gamma))$ edges.*
2. *The total running time is $O(m \log^3(n/\gamma) + \varepsilon^{-2} n \log^4(n/\gamma))$.*

First, we prove the correctness and runtime of ApproxSchurNode(H) in Algorithm 7. We say ApproxSchurNode(H) ran correctly on a node H at level i in $\mathcal{T}$, if at the end of the procedure, the following properties are satisfied:

- $\mathbf{L}^{(H)}$ is the Laplacian of a graph on vertices $\partial H \cup F_H$ with $\widetilde{O}(\delta^{-2}|\partial H \cup F_H|)$ edges,
- $\mathbf{L}^{(H)} \approx_{i\delta} \mathbf{Sc}(\mathbf{L}[H], \partial H \cup F_H)$,
- $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H) \approx_{(i+1)\delta} \mathbf{Sc}(\mathbf{L}[H], \partial H)$, and the graph is on ∂H with $\widetilde{O}(\delta^{-2}|\partial H|)$ edges.

Lemma 4.9. *Suppose* ApproxSchurNode(D) *has run correctly for all descendants D of H. then* ApproxSchurNode(H) *runs correctly.*

Proof. When H is a leaf, the proof is trivial. $\mathbf{L}^{(H)}$ is set to the exact Laplacian matrix of the induced subgraph H of constant size. $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ δ-approximates $\mathbf{Sc}(\mathbf{L}^{(H)}, \partial H) = \mathbf{Sc}(\mathbf{L}[H], \partial H)$ by Lemma 4.8.

Otherwise, suppose H is at level i with children D_1 and D_2. By construction of the separator tree,we have $\partial D_1 \cup \partial D_2 = \partial H \cup F_H$. For each $j = 1, 2$, we know inductively $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_j)}, \partial D_j)$ has $\widetilde{O}(\delta^{-2}|\partial D_j|)$ edges. Since we define $\mathbf{L}^{(H)}$ to be the sum, it has $\widetilde{O}(\delta^{-2}(|\partial D_1| + |\partial D_2|)) = \widetilde{O}(\delta^{-2}|\partial H \cup F_H|)$ edges, and is supported on vertices $\partial H \cup F_H$, so we have the first correctness property.

Inductively, we know $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_j)}, \partial D_j) \approx_{(i-1)\delta} \mathbf{Sc}(\mathbf{L}[D_j], \partial D_j)$ for both $j = 1, 2$. (The height of

Algorithm 8 Data structure to maintain dynamic Schur complement approximations, part 1

```
 1: data structure ApproxDynamicSC
 2: private: member
 3:     $\boldsymbol{w} \in \mathbb{R}^m$: weight vector
 4:     $\epsilon_{\mathbf{P}} > 0$: Overall approximation factor
 5:     $\delta > 0$: Fast Schur complement approximation factor
 6:     $\mathcal{T}$: Separator tree of height $\eta$. Every node $H$ of $\mathcal{T}$ stores:
 7:         $F_H$, $\partial H$: Interior and boundary vertices of region $H$
 8:         $\mathbf{L}^{(H)} \in \mathbb{R}^{m \times m}$: Laplacian supported on $F_H \cup \partial H$
 9:         $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H) \in \mathbb{R}^{m \times m}$: $\delta$-approximate Schur complement
10:
11: procedure Initialize($\mathcal{T}$, $\boldsymbol{w} \in \mathbb{R}^m$, $\epsilon_{\mathbf{P}} > 0$)
12:     $\mathcal{T} \leftarrow \mathcal{T}$
13:     $\delta \leftarrow \epsilon_{\mathbf{P}}/(\eta + 1)$
14:     $\boldsymbol{w} \leftarrow \boldsymbol{w}$
15:     for $i = 0, \ldots, \eta$ do
16:         for each node $H$ at level $i$ in $\mathcal{T}$ do
17:             ApproxSchurNode($H$)
18:         end for
19:     end for
20: end procedure
21:
22: procedure Reweight($\boldsymbol{\delta_w} \in \mathbb{R}^m$)
23:     $\mathcal{H} \leftarrow$ set of leaf nodes $H$ in $\mathcal{T}$ such that $\boldsymbol{\delta}_{E(H)} \neq \mathbf{0}$
24:     $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{\delta_w}$
25:     for $i = 0, \ldots, \eta$ do          ▷ $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$ is the set of nodes in $\mathcal{H}$ and their ancestors
26:         for each node $H$ at level $i$ in $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$ do
27:             ApproxSchurNode($H$)
28:         end for
29:     end for
30: end procedure
31:
```

Algorithm 8 Data structure to maintain dynamic Schur complement approximations, part 2

```
32: data structure ApproxDynamicSC
33: procedure ApproxSchurNode($H \in \mathcal{T}$)
34:     if $H$ is a leaf node then
35:         $\mathbf{L}^{(H)} \leftarrow (\mathbf{B}[H])^\top \mathbf{W}_{E(H)} \mathbf{B}[H]$
36:         $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H) \leftarrow$ ApproxSchur($\mathbf{L}^{(H)}, \partial H, \delta$)        ▷ Lemma 4.8
37:     else
38:         Let $D_1, D_2$ be the children of $H$
39:         $\mathbf{L}^{(H)} \leftarrow \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)}, \partial D_1) + \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)}, \partial D_2)$
40:         $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H) \leftarrow$ ApproxSchur($\mathbf{L}^{(H)}, \partial H, \delta$)
41:     end if
42: end procedure
```

D_j may or may not equal to $i-1$ but it is guaranteed to be no more than $i-1$.) Then

$$\begin{aligned}
\mathbf{L}^{(H)} &= \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)}, \partial D_1) + \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)}, \partial D_2) \\
&\approx_{(i-1)\delta} \mathbf{Sc}(\mathbf{L}[D_1], \partial D_1) + \mathbf{Sc}(\mathbf{L}[D_2], \partial D_2) \\
&= \mathbf{Sc}(\mathbf{L}[D_1], (\partial H \cup F_H) \cap V(D_1)) + \mathbf{Sc}(\mathbf{L}[D_2], (\partial H \cup F_H) \cap V(D_2)) \\
&\quad \text{(by construction of the separator tree, } \partial D_j = (\partial H \cup F_H) \cap V(D_j) \text{ for } j = 1, 2) \\
&= \mathbf{Sc}(\mathbf{L}[H], \partial H \cup F_H), \qquad \text{(by Lemma 3.27)}
\end{aligned}$$

so we have the second correctness property.

Line 40 returns $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ with $\widetilde{O}(\delta^{-2}|\partial H|)$ edges by Lemma 4.8. Also,

$$\begin{aligned}
\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H) &\approx_{\delta} \mathbf{Sc}(\mathbf{L}^{(H)}, \partial H) \\
&\approx_{(i-1)\delta} \mathbf{Sc}(\mathbf{Sc}(\mathbf{L}[H], \partial H \cup F_H), \partial H) \\
&= \mathbf{Sc}(\mathbf{L}[H], \partial H), \qquad \text{(by Lemma 3.24)}
\end{aligned}$$

giving us the third correctness property. □

Lemma 4.10. *The runtime of* ApproxSchurNode(H) *is* $\widetilde{O}(\delta^{-2}|\partial H \cup F_H|)$.

Proof. When H is a leaf node, computing $\mathbf{L}^{(H)} = \mathbf{L}[H]$ takes time proportional to $|H| = \partial H \cup F_H$. Computing $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ takes $\widetilde{O}(\delta^{-2}|H|)$ time by Lemma 4.8.

Otherwise, when H has children D_1, D_2, computing $\mathbf{L}^{(H)}$ requires accessing $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_j)}, \partial D_j)$ for $j = 1, 2$ and summing them together, in time $\widetilde{O}(|\partial D_1| + |\partial D_2|) = \widetilde{O}(|\partial H \cup F_H|)$. Then, computing $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ take $\widetilde{O}(\delta^{-2}|\partial H \cup F_H|)$ by Lemma 4.8. □

Next, we prove the overall data structure correctness and runtime:

Theorem 4.11 (Schur complements maintenance)**.** *Given a separator tree $\mathcal{T}$ of height $\eta = O(\log m)$ for the IPM input graph G, the deterministic data structure* DynamicSC *(Algorithm 7) correctly maintains two Laplacians $\mathbf{L}^{(H)}$ and $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H \cup F_H)$ at every node $H \in \mathcal{T}$, which are dependent on the dynamic weights $\boldsymbol{w}$ from the IPM. The data structure supports the following procedures:*

- Initialize$(\mathcal{T}, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \epsilon_{\mathbf{P}} > 0)$*: Given the separator tree $\mathcal{T}$, initial weights $\boldsymbol{w}$, target step accuracy $\epsilon_{\mathbf{P}}$, preprocess in $\widetilde{O}(\delta^{-2} \sum_{H \in \mathcal{T}} |F_H \cup \partial H|)$ time.*
- Reweight$(\boldsymbol{\delta_w} \in \mathbb{R}^m)$*: Update the weight vector to $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{\delta_w}$, and update all the maintained matrices with respect to the new weights, in time*
$$O\left(\sum_{H \in \mathcal{H}} \delta^{-2} \cdot |F_H \cup \partial H|\right),$$
where $\mathcal{H}$ is the set of nodes H with $\boldsymbol{\delta_w}|_{E(H)} \neq \mathbf{0}$.
- *Access to Laplacian $\mathbf{L}^{(H)}$ at any node $H \in \mathcal{T}$ in time $\widetilde{O}\left({\epsilon_{\mathbf{P}}}^{-2} |F_H \cup \partial H|\right)$.*
- *Access to Laplacian $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ at any node $H \in \mathcal{T}$ in time $\widetilde{O}\left({\epsilon_{\mathbf{P}}}^{-2} |\partial H|\right)$.*

Furthermore, at all points during the IPM,
$$\mathbf{L}^{(H)} \approx_{\epsilon_{\mathbf{P}}} \mathbf{Sc}(\mathbf{L}[H], F_H \cup \partial H) \quad \text{and} \quad \widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H) \approx_{\epsilon_{\mathbf{P}}} \mathbf{Sc}(\mathbf{L}[H], \partial H) \tag{4.10}$$
for all $H \in \mathcal{T}$ with high probability.

Proof. Because we set $\delta \leftarrow \epsilon_{\mathbf{P}}/(\eta + 1)$ in Initialize, combined with Lemma 4.9, we conclude that for each $H \in \mathcal{T}$,
$$\mathbf{L}^{(H)} \approx_{\epsilon_{\mathbf{P}}} \mathbf{Sc}(\mathbf{L}[H], \partial H \cup F_H)$$
and
$$\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H) \approx_{\epsilon_{\mathbf{P}}} \mathbf{Sc}(\mathbf{L}[H], \partial H).$$

We next prove the correctness and runtime of Initialize. Because ApproxSchurNode(H) is called in increasing order of level of H, each ApproxSchurNode(H) runs correctly and stores the initial value of $\mathbf{L}^{(H)}$ by Lemma 4.9. The runtime of Initialize is bounded by running ApproxSchurNode on each node, i.e:
$$\widetilde{O}(\delta^{-2} \sum_{H \in \mathcal{T}} |\partial H \cup F_H|)$$

The proof for Reweight is similar to Initialize. Let K be the number of coordinates changed in $\boldsymbol{w}$. Then $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$ contains all the regions with an edge with weight update. For each node H not in $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$, no edge in H has a modified weight, and in this case, we do not need to update $\mathbf{L}^{(H)}$. For the nodes that do require updates, since ApproxSchurNode(H) is called in increasing order of level of H, we can prove inductively that all ApproxSchurNode(H) for $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$ run correctly. The time spent is bounded by $\widetilde{O}(\delta^{-2} \sum_{H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})} |\partial H \cup F_H|)$.

For accessing $\mathbf{L}^{(H)}$ and $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$, we simply return the stored values. The time required is proportional to the size of $\mathbf{L}^{(H)}$ and $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ respectively, by the correctness properties of these Laplacians, we get the correct size and therefore the runtime. $\square$

Chapter 5

IPM DATA STRUCTURES

In this chapter, we give the remaining data structures for our robust IPM. For simplicity, we reference PATHFOLLOWINGLP instead of the more general PATHFOLLOWINGROBUST.

5.1 Dynamic implicit representation

Assuming we have dynamic inverse tree and tree operators ∇ and $\boldsymbol{\Delta}$ on tree $\mathcal{T}$ dependent on $\boldsymbol{w}$ such that $\mathbf{W}^{1/2}\mathbf{P}_{\boldsymbol{w}} = \boldsymbol{\Delta}\nabla$, we can now state how to abstractly maintain the implicit representation of the solutions throughout the PATHFOLLOWINGLP procedure in the IPM. Specifically, we want to maintain the solution $\boldsymbol{x}$, and at every step k, carry out an update of the form

$$\boldsymbol{x} \leftarrow \boldsymbol{x} + \mathbf{W}^{1/2}\mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}^{(k)}. \tag{5.1}$$

We design a data structure MAINTAINREP to accomplish this, by:

- At the start of PATHFOLLOWINGLP, initializing the data structure using the procedure INITIALIZE with $\boldsymbol{x} = \boldsymbol{x}^{(\mathrm{init})}$,
- At each step k, updating the weights $\boldsymbol{w}$ in the data structure using the procedure REWEIGHT, followed by updating $\boldsymbol{x}$ according to Eq. (5.1) using the procedure MOVE,
- At the end of PATHFOLLOWINGLP, outputing the final $\boldsymbol{x}$ using the procedure EXACT.

The key to designing an efficient data structure is to make use of the structure of the operators. Due to their decomposition along $\mathcal{T}$, we can update the operators and apply them to vectors without exploring all of $\mathcal{T}$ every time.

Theorem 5.1 (Implicit representation maintenance)**.** *Let $\boldsymbol{w}$ be the weights changing at every step of* PATHFOLLOWINGLP*. Suppose there exists dynamic inverse tree and tree operators ∇ and $\boldsymbol{\Delta}$ on tree $\mathcal{T}$ both dependent on $\boldsymbol{w}$ such that $\mathbf{W}^{1/2}\mathbf{P}_{\boldsymbol{w}} = \boldsymbol{\Delta}\nabla$ throughout the IPM. Let Q be the max of the query complexity of the tree and inverse tree operator, and let U be the max of the update complexity of the two operators. Suppose $\mathcal{T}$ has constant degree and height η.*

Then there is a data structure MAINTAINREP *that satisfies the following invariants at the end of step k:*

- *It explicitly maintains the dynamic weights $\boldsymbol{w}$ and step direction $\boldsymbol{v}$ from the current step.*
- *It explicitly maintains scalar c and vectors $\boldsymbol{z}^{(\text{step})}, \boldsymbol{z}^{(\text{sum})}$, which together represent the implicitly-maintained vector $\boldsymbol{z} \stackrel{\text{def}}{=} c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})}$. At the end of step k, $\boldsymbol{z}^{(\text{step})} = \nabla^{(k)}\boldsymbol{v}^{(k)}$, and*
$$\boldsymbol{z} = \sum_{i=1}^{k} \nabla^{(i)}\boldsymbol{v}^{(i)}.$$
- *It implicitly maintains $\boldsymbol{x}$ so that at the end of step k,*
$$\boldsymbol{x} = \boldsymbol{x}^{(\text{init})} + \sum_{i=1}^{k} \boldsymbol{\Delta}^{(i)}\nabla^{(i)}\boldsymbol{v}^{(i)},$$
where $\boldsymbol{x}^{(\text{init})}$ is some initial value set at the start of PATHFOLLOWINGLP*.*

The data structure supports the following procedures and runtimes:

- INITIALIZE$(\boldsymbol{\Delta}, \nabla, \boldsymbol{v}^{(\text{init})} \in \mathbb{R}^m, \boldsymbol{w}^{(\text{init})} \in \mathbb{R}^m_{>0}, \boldsymbol{x}^{(\text{init})} \in \mathbb{R}^m)$*: Preprocess and set $\boldsymbol{x} \leftarrow \boldsymbol{x}^{(\text{init})}$.*

  The procedure runs in $O(U(m) + Q(m))$ time.
- REWEIGHT$(\boldsymbol{\delta_w} \in \mathbb{R}^m_{>0})$*: Update the weights to $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{\delta_w}$.*

  The procedure runs in $O(U(\eta K) + Q(\eta K))$ total time, where $K = \operatorname{nnz}(\boldsymbol{\delta_w})$.
- MOVE$(h \in \mathbb{R}, \boldsymbol{\delta_v} \in \mathbb{R}^m)$*: Update the current step direction to $\boldsymbol{v} \leftarrow \boldsymbol{v} + \boldsymbol{\delta_v}$. Update the implicit representation of $\boldsymbol{x}$ to reflect the following change in value:*
$$\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{\Delta}\nabla\boldsymbol{v}.$$
  The procedure runs in $O(Q(\eta K))$ time, where $K = \operatorname{nnz}(\boldsymbol{\delta_v})$.
- EXACT*: Output the current exact value of $\boldsymbol{x}$ in $O(Q(m))$ time.*

In this section, we give the general data structure MAINTAINREP, which implicitly maintains a vector $\boldsymbol{x}$ throughout a call of PATHFOLLOWINGLP. We break up the representation into two parts, the first using the inverse tree operator, and the second using the tree operator.

Now we are ready for the complete data structure involving the inverse tree operator.

Algorithm 9 Dynamic data structure to maintain cumulative $\nabla \boldsymbol{v}$

```
1: dynamic data structure InverseTreeOp
2: member:
3:     $\mathcal{T}$: tree supporting $\nabla$ with edge operators on the edges
4:     $\boldsymbol{w} \in \mathbb{R}^m$: dynamic weight vector
5:     $\boldsymbol{v} \in \mathbb{R}^n$: dynamic vector
6:     $c, \boldsymbol{z}^{(\text{step})}, \boldsymbol{z}^{(\text{sum})} \in \mathbb{R}^n$: coefficient, result vectors
7:     $\boldsymbol{y}_H \in \mathbb{R}^n$ for each $H \in \mathcal{T}$: sparse partial computations
8:
9: procedure Initialize($\mathcal{T}, \boldsymbol{w}^{(\text{init})}, \boldsymbol{v}^{(\text{init})}$)
10:     $\boldsymbol{w} \leftarrow \boldsymbol{w}^{(\text{init})}, \boldsymbol{v} \leftarrow \boldsymbol{v}^{(\text{init})}, c \leftarrow 0, \boldsymbol{z}^{(\text{sum})} \leftarrow \boldsymbol{0}$
11:     Initialize $\nabla$ on $\mathcal{T}$ based on $\boldsymbol{w}$
12:     Compute $\nabla \boldsymbol{v}$ and $\boldsymbol{y}_H$'s, set $\boldsymbol{z}^{(\text{step})} \leftarrow \nabla \boldsymbol{v}$
13: end procedure
14:
15: procedure Reweight($\boldsymbol{\delta}_{\boldsymbol{w}}$)
16:     $\boldsymbol{w}^{(\text{new})} \leftarrow \boldsymbol{w} + \boldsymbol{\delta}_{\boldsymbol{w}}$
17:     Let $\nabla^{(\text{new})}$ be the new tree operator using $\boldsymbol{w}^{(\text{new})}$
18:     $\boldsymbol{z}' \leftarrow (\nabla^{(\text{new})} - \nabla)\boldsymbol{v}$, and update $\boldsymbol{y}_H$'s        ▷ Lemma 3.45
19:     $\boldsymbol{z}^{(\text{step})} \leftarrow \boldsymbol{z}^{(\text{step})} + \boldsymbol{z}'$
20:     $\boldsymbol{z}^{(\text{sum})} \leftarrow \boldsymbol{z}^{(\text{sum})} - c \cdot \boldsymbol{z}'$
21:     $\boldsymbol{w} \leftarrow \boldsymbol{w}^{(\text{new})}, \nabla \leftarrow \nabla^{(\text{new})}$
22: end procedure
23:
24: procedure Move($h$, $\boldsymbol{\delta}_{\boldsymbol{v}}$)
25:     Compute $\boldsymbol{z}' \overset{\text{def}}{=} \nabla \boldsymbol{\delta}_{\boldsymbol{v}}$ and the $\boldsymbol{y}'_H \overset{\text{def}}{=} \sum_{\text{leaf } L \in \mathcal{T}_H} \nabla_{H \leftarrow L} \boldsymbol{\delta}_{\boldsymbol{v}}$ for each node $H$
26:                                                                 ▷ Lemma 3.42
27:     $\boldsymbol{z}^{(\text{step})} \leftarrow \boldsymbol{z}^{(\text{step})} + \boldsymbol{z}'$, and $\boldsymbol{y}_H \leftarrow \boldsymbol{y}_H + \boldsymbol{y}'_H$ for each node $H$
28:     $\boldsymbol{z}^{(\text{sum})} \leftarrow \boldsymbol{z}^{(\text{sum})} - c\boldsymbol{z}'$
29:     $c \leftarrow c + 1$
30:     $\boldsymbol{v} \leftarrow \boldsymbol{v} + \boldsymbol{\delta}_{\boldsymbol{v}}$
31: end procedure
```

Theorem 5.2 (Inverse tree operator data structure)**.** *Let $\boldsymbol{w} \in \mathbb{R}^m$ be the weights changing at every step of* PathFollowingLP*, and let $\boldsymbol{v} \in \mathbb{R}^n$ be a dynamic vector. Suppose $\nabla : \mathbb{R}^m \mapsto \mathbb{R}^n$ is an inverse tree operator dependent on $\boldsymbol{w}$ supported on $\mathcal{T}$ with query complexity Q and update complexity U. Let η be the height of $\mathcal{T}$. Then the data structure* InverseTreeOp *(Algorithm 9) maintains $\boldsymbol{z}^{(k)} \stackrel{\text{def}}{=} \sum_{i=1}^{k} \nabla^{(i)} \boldsymbol{v}^{(i)}$ so that at the end of each step k, the variables in the algorithm satisfy*

- $\boldsymbol{z} = c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})}$,
- $\boldsymbol{z}^{(\text{step})} = \nabla \boldsymbol{v}$, *and*
- $\boldsymbol{y}_H = \sum_{\text{leaf } L \in \mathcal{T}_H} \nabla_{H \leftarrow L} \boldsymbol{v}$ *for all nodes H.*

The data structure is initialized via Initialize *in $O(U(m) + Q(m))$ time. At step k, there is one call* Reweight$(\boldsymbol{\delta_w})$ *taking $O(U(K) + Q(\eta K))$ time, where $K = \operatorname{nnz}(\boldsymbol{\delta_w})$, followed by one call of* Move$(h, \boldsymbol{\delta_v})$ *taking $O(Q(\eta \cdot \operatorname{nnz}(\boldsymbol{\delta_v})))$ time.*

Proof. In the data structure, we always maintain $\boldsymbol{z}^{(\text{step})}$ and the $\boldsymbol{y}_H$'s together. Specifically, at every step, we update the $\boldsymbol{y}_H$'s up the tree using the recursive property Eq. (3.14) only at the necessary nodes, and from the $\boldsymbol{y}_H$'s, we get $\boldsymbol{z}^{(\text{step})} = \sum_H \mathbf{I}_{F_H} \boldsymbol{y}_H$.

Consider Initialize. At the end of the function, the variables satisfy

$$\boldsymbol{z} \stackrel{\text{def}}{=} c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})} = 0 \cdot \nabla \boldsymbol{v} + \mathbf{0} = \mathbf{0},$$

and $\boldsymbol{z}^{(\text{step})} = \nabla \boldsymbol{v}$, as required.

Let us consider Reweight. Let the superscript $^{(\text{new})}$ denote the value of an algorithm variable at the end of the function, and let no superscript denote the value at the start.

$$\begin{aligned}
\boldsymbol{z}^{(\text{new})} &= c^{(\text{new})} \boldsymbol{z}^{(\text{step})^{(\text{new})}} + \boldsymbol{z}^{(\text{sum})^{(\text{new})}} \\
&= c(\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}') + \boldsymbol{z}^{(\text{sum})} - c\boldsymbol{z}' \\
&= c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})}, \\
\text{and } \boldsymbol{z}^{(\text{step})^{(\text{new})}} &= \boldsymbol{z}^{(\text{step})} + (\nabla^{(\text{new})} - \nabla)\boldsymbol{v} \\
&= \nabla^{(\text{new})} \boldsymbol{v},
\end{aligned}$$

as required. Similarly, let us consider MOVE:

$$\begin{aligned}
\boldsymbol{z}^{(\text{new})} &= c^{(\text{new})}\boldsymbol{z}^{(\text{step})^{(\text{new})}} + \boldsymbol{z}^{(\text{sum})^{(\text{new})}} \\
&= (c+1)(\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}') + \boldsymbol{z}^{(\text{sum})} - c\boldsymbol{z}' \\
&= c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})} + \boldsymbol{z}^{(\text{step})}, \\
\text{and } \boldsymbol{z}^{(\text{step})^{(\text{new})}} &= \boldsymbol{z}^{(\text{step})} + \nabla(\boldsymbol{v}^{(\text{new})} - \boldsymbol{v}) \\
&= \nabla\boldsymbol{v} + \nabla(\boldsymbol{v}^{(\text{new})} - \boldsymbol{v}) \\
&= \nabla\boldsymbol{v}^{(\text{new})},
\end{aligned}$$

which is exactly the update we want to make to $\boldsymbol{z}$, and the invariant we want to maintain.

The runtimes follow directly from Lemmas 3.42 and 3.45. □

Next, we present the tree operator data structure, which is significantly more involved compared to the inverse tree operator. Applying the tree operator involves going down the tree to the leaves, which is too costly to do at every step. To circumvent the issue, we use lazy computations.

Theorem 5.3 (Tree operator data structure)**.** *Let $\boldsymbol{w} \in \mathbb{R}^m$ be the weights changing at every step of* PATHFOLLOWINGLP*. Suppose $\boldsymbol{\Delta} : \mathbb{R}^n \mapsto \mathbb{R}^m$ is a tree operator dependent on $\boldsymbol{w}$ supported on $\mathcal{T}$ with query complexity Q and update complexity U. Let $\boldsymbol{z} \in \mathbb{R}^n$ be the vector maintained by Algorithm 9, so that at the end of step k, $\boldsymbol{z} = \sum_{i=1}^{k} \nabla^{(i)}\boldsymbol{v}^{(i)}$. Then the data structure* TREEOP *(Algorithm 10) implicitly maintains $\boldsymbol{x}$ so that at the end of step k,*

$$\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(\text{init})} + \sum_{i=1}^{k} \boldsymbol{\Delta}^{(i)}\nabla^{(i)}\boldsymbol{v}^{(i)}.$$

The data structure is initialized via INITIALIZE *in $O(U(m))$ time. At step k, there is one call to* REWEIGHT$(\boldsymbol{\delta_w})$ *taking $O(U(K) + Q(\eta K))$ time, where $K = \text{nnz}(\boldsymbol{\delta_w})$, followed by one call to* MOVE$(\boldsymbol{\delta_z})$ *taking $O(\text{nnz}(\boldsymbol{\delta_z}))$ time. At the end of* PATHFOLLOWINGLP*, $\boldsymbol{x}$ is returned via* EXACT *in $O(Q(m))$ time.*

Proof. We will show that the data structure maintains the implicit representation via the identity

$$\boldsymbol{x} = c\boldsymbol{\Delta}\boldsymbol{z} + \sum_{H \in \mathcal{T}} \boldsymbol{\Delta}^{(H)}\boldsymbol{u}_H, \tag{5.2}$$

where the RHS expression refers to the state of the variables at the end of step k during the algorithm.

Algorithm 10 Dynamic data structure to maintain cumulative $\boldsymbol{\Delta z}$

```
1: dynamic data structure TreeOp
2: member:
3:     $\mathcal{T}$: tree supporting $\boldsymbol{\Delta}$
4:     $\boldsymbol{w} \in \mathbb{R}^m$: dynamic weight vector
5:     $\boldsymbol{z} \in \mathbb{R}^n$: dynamic vector
6:     $\boldsymbol{u}_H$ for each $H \in \mathcal{T}$: lazy pushdown computation vectors

7: procedure Initialize($\mathcal{T}, \boldsymbol{w}^{(\text{init})}, \boldsymbol{z}^{(\text{init})}, \boldsymbol{x}^{(\text{init})}$)
8:     $\boldsymbol{w} \leftarrow \boldsymbol{w}^{(\text{init})}, \boldsymbol{z} \leftarrow \boldsymbol{z}^{(\text{init})}$
9:     Initialize $\boldsymbol{\Delta}$ on $\mathcal{T}$ based on $\boldsymbol{w}$
10:    $\boldsymbol{u}_H \leftarrow \boldsymbol{0}$ for each non-leaf $H \in \mathcal{T}$
11:    $\boldsymbol{u}_H \leftarrow \boldsymbol{x}^{(\text{init})}|_{E(H)}$ for each leaf $H \in \mathcal{T}$
12: end procedure

13: procedure Reweight($\boldsymbol{\delta}_{\boldsymbol{w}}$)
14:    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{\delta}_{\boldsymbol{w}}$
15:    Let $\boldsymbol{\Delta}^{(\text{new})}$ be the new tree operator wrt the new weights
16:    Let $\mathcal{H}$ be all nodes $H$ where $\boldsymbol{\Delta}_H$ changed
17:    for $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$ going down the tree level by level do
18:        Pushdown($H$)
19:    end for
20:    for $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$ going down the tree level by level do
21:        $\boldsymbol{u}_H \leftarrow c\boldsymbol{z}|_{F_H}$
22:        Pushdown($H$)
23:    end for
24:    $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta}^{(\text{new})}$
25:    for $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$ going down the tree level by level do
26:        $\boldsymbol{u}_H \leftarrow -c\boldsymbol{z}|_{F_H}$
27:        Pushdown($H$)
28:    end for
29: end procedure

30: procedure Move($\boldsymbol{\delta}_{\boldsymbol{z}}$)
31:    $\boldsymbol{z} \leftarrow \boldsymbol{z} + \boldsymbol{\delta}_{\boldsymbol{z}}$
32: end procedure
```

Algorithm 10 Dynamic data structure to maintain cumulative $\boldsymbol{\Delta z}$, part 2

```
33: dynamic data structure TreeOp
34: procedure Exact
35:     for $H \in \mathcal{T}$ going down the tree level by level do
36:         $\boldsymbol{u}_H \leftarrow \boldsymbol{u}_H + \boldsymbol{z}|_{F_H}$
37:         Pushdown($H$)
38:     end for
39:     return $\boldsymbol{x}$ defined by $\boldsymbol{x}|_{E(H)} \overset{\text{def}}{=} \boldsymbol{u}_H$ at each leaf $H \in \mathcal{T}$
40: end procedure

41: procedure Pushdown($H \in \mathcal{T}$)
42:     for each child $D$ of $H$ do
43:         $\boldsymbol{u}_D \leftarrow \boldsymbol{u}_D + \boldsymbol{\Delta}_D \boldsymbol{u}_H$
44:     end for
45:     $\boldsymbol{u}_H \leftarrow \boldsymbol{0}$
46: end procedure
```

At a high level, the variables $\boldsymbol{\Delta}$ and $\boldsymbol{z}$ in the data structure at step k represent the latest $\boldsymbol{\Delta}^{(k)}$ and $\boldsymbol{z}^{(k)}$. We need to introduce additional vectors $\boldsymbol{u}_H$ at every node H which intuitively stores lazy computations at node H, in order to take advantage of the tree structure of $\boldsymbol{\Delta}$. The function Pushdown performs the accumulated computation at H, and moves the result to its children nodes to be computed lazily at a later point. The next claim describes this process formally.

Claim 5.4. *Let $H \in \mathcal{T}$ be a non-leaf node.* Pushdown(H) *does not change the value of the implicit representation in Eq.* (5.2)*. Also, at the end of the procedure, $\boldsymbol{u}_H = \boldsymbol{0}$.*

Proof. For any variable in the algorithm, we add the superscript $^{(\text{new})}$ to mean its state at the end of Pushdown; if there is no superscript, then it refers to the state at the start.

We show the claim for when H has two children H_1, H_2. Note that $\boldsymbol{\Delta}$ and $\boldsymbol{z}$ are not touched by PUSHDOWN, so we may ignore the term $c\boldsymbol{\Delta z}$ in Eq. (5.2). Then,

$$\begin{aligned}
&\sum_{H'\in\mathcal{T}} \boldsymbol{\Delta}^{(H')}\boldsymbol{u}_{H'}^{(\text{new})} \\
&= \boldsymbol{\Delta}^{(H)}\boldsymbol{u}_{H}^{(\text{new})} + \sum_{i=1,2} \boldsymbol{\Delta}^{(H_i)}\boldsymbol{u}_{H_i}^{(\text{new})} + \sum_{H'\in\mathcal{T}, H'\neq H, H_1, H_2} \boldsymbol{\Delta}^{(H')}\boldsymbol{u}_{H'} && \text{(expand terms)} \\
&= \sum_{i=1,2} \boldsymbol{\Delta}^{(H_i)}(\boldsymbol{u}_{H_i} + \boldsymbol{\Delta}_{H_i}\boldsymbol{u}_H) + \sum_{H'\in\mathcal{T}, H'\neq H, H_1, H_2} \boldsymbol{\Delta}^{(H')}\boldsymbol{u}_{H'} && \text{(substitute values)} \\
&= \sum_{i=1,2} \boldsymbol{\Delta}^{(H_i)}\boldsymbol{\Delta}_{H_i}\boldsymbol{u}_H + \sum_{H\in\mathcal{T}, H'\neq H} \boldsymbol{\Delta}^{(H')}\boldsymbol{u}_{H'} \\
&= \boldsymbol{\Delta}^{(H)}\boldsymbol{u}_H + \sum_{H\in\mathcal{T}, H'\neq H} \boldsymbol{\Delta}^{(H')}\boldsymbol{u}_{H'} && \text{(by Eq. (3.13))} \\
&= \sum_{H'\in\mathcal{T}} \boldsymbol{\Delta}^{(H')}\boldsymbol{u}_{H'},
\end{aligned}$$

so the implicit representation of $\boldsymbol{x}$ has not changed in value. □

This claim can be generalized from $H \in \mathcal{T}$ to $\mathcal{H} \subseteq \mathcal{T}$; we omit the full details. Next, we show that the implicit representation of $\boldsymbol{x}$ by Eq. (5.2) is correctly maintained after reweight.

Claim 5.5. *After the k-th call* REWEIGHT*, the value of $\boldsymbol{x}$ is unchanged, while the value of $\boldsymbol{\Delta}$ is updated to $\boldsymbol{\Delta}^{(k)}$ which is a function of $\boldsymbol{w}^{(k)}$.*

Proof. We begin by observing that if $H \notin \mathcal{P}_{\mathcal{T}}(\mathcal{H})$, then $\boldsymbol{\Delta}^{(H)^{(\text{new})}} = \boldsymbol{\Delta}^{(H)}$ by definition, as there are no edges in $\mathcal{T}_H$ with updated operators.

At a high level, we traverse the subtree $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$ three rounds and perform PUSHDOWN at every node. During the first round, we simply push down the current $\boldsymbol{u}_H$ values at each node H. By Claim 5.4, we know this does not change the value of the implicit representation.

During the second round, we first initialize $\boldsymbol{u}_H \leftarrow c\boldsymbol{z}|_{F_H}$ at each node $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$, and then perform PUSHDOWN. Since PUSHDOWN does not affect the value of the implicit representation, we can use the initial change in $\boldsymbol{u}_H$ to determine the overall change in the implicit representation. Crucially, note that we perform PUSHDOWN using the old tree operator. So, the change in value of the implicit representation is given by

$$+c \sum_{H\in\mathcal{P}_{\mathcal{T}}(\mathcal{H})} \boldsymbol{\Delta}^{(H)}\boldsymbol{z}|_{F_H}.$$

After the second round of PUSHDOWN, we update the tree operator $\boldsymbol{\Delta}$ to $\boldsymbol{\Delta}^{(\text{new})}$. Note that $\boldsymbol{\Delta}^{(H)}$ changes if and only if $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$, and in this case, $\boldsymbol{u}_H = \boldsymbol{0}$. So, updating the tree operator at this point induces a change in the value of the implicit representation of

$$c\boldsymbol{\Delta}^{(\text{new})}\boldsymbol{z} - c\boldsymbol{\Delta}\boldsymbol{z} = c\sum_{H\in\mathcal{T}}\left(\boldsymbol{\Delta}^{(H)^{(\text{new})}} - \boldsymbol{\Delta}^{(H)}\right)\boldsymbol{z}|_{F_H} = c\sum_{H\in\mathcal{P}_{\mathcal{T}}(\mathcal{H})}\left(\boldsymbol{\Delta}^{(H)^{(\text{new})}} - \boldsymbol{\Delta}^{(H)}\right)\boldsymbol{z}|_{F_H}.$$

During the third round, we initialize $\boldsymbol{u}_H \leftarrow -c\boldsymbol{z}|_{F_H}$ at each node $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$ and perform PUSHDOWN. Similar to the first round, the change to the value of the implicit representation induced by this round is given by

$$-c\sum_{H\in\mathcal{P}_{\mathcal{T}}(\mathcal{H})}\boldsymbol{\Delta}^{(H)^{(\text{new})}}\boldsymbol{z}|_{F_H}.$$

The sum of the changes from each of the three rounds is exactly $\mathbf{0}$, so we conclude the value of the implicit representation did not change. □

Finally, we consider the other functions.

For INITIALIZE, we see that by substituting the values assigned during INITIALIZE and applying the definition from Eq. (3.12), we have

$$c\boldsymbol{\Delta}\boldsymbol{z} + \sum_{H\in\mathcal{T}}\boldsymbol{\Delta}^{(H)}\boldsymbol{u}_H = \boldsymbol{x}^{(\text{init})} + \boldsymbol{\Delta}\boldsymbol{z},$$

where $\boldsymbol{\Delta}$ is the initial $\boldsymbol{\Delta}^{(\text{init})}$ and $\boldsymbol{z}$ is the initial $\boldsymbol{z}^{(\text{init})}$, which is exactly how we want to initialize $\boldsymbol{x}$.

For MOVE, we see the value of $\boldsymbol{x}$ is incremented by $\boldsymbol{\Delta}^{(k)}(\boldsymbol{z}^{(k)} - \boldsymbol{z}^{(k-1)})$ after the step k. By definition of $\boldsymbol{z}$, we know $\boldsymbol{z}^{(k)} - \boldsymbol{z}^{(k-1)} = \nabla^{(k)}\boldsymbol{v}^{(k)}$, so we conclude MOVE correctly makes the update $\boldsymbol{\Delta}^{(k)}\nabla^{(k)}\boldsymbol{v}^{(k)}$.

For EXACT, we perform the computation $\sum_{H\in\mathcal{T}}\boldsymbol{\Delta}^{(H)}(\boldsymbol{u}_H + \boldsymbol{z}|_{F_H})$ using a sequence of PUSHDOWN's down the tree, in order to calculate the value of $\boldsymbol{x}$ explicitly. The final answer $\boldsymbol{x}$ is stored in parts in the $\boldsymbol{u}_H$'s along the leaf nodes.

Note that by definition of the query complexity of $\boldsymbol{\Delta}$, PUSHDOWN uses $O(Q(1))$ time. The remaining runtimes are straightforward. □

Finally, we combine Algorithm 9 and Algorithm 10 to get the overall data structure MAINTAINREP for maintaining $\boldsymbol{x}$ throughout PATHFOLLOWINGLP as given by Eq. (5.1). We omit the pseudocode implementation.

Proof of Theorem 5.1. We use one copy of InverseTreeOp, which maintains $\boldsymbol{z} \stackrel{\text{def}}{=} c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})}$. We want to use TreeOp to maintain $\boldsymbol{z}$ which is given in two terms by Inverse-TreeOp. To resolve this, we can simply use two copies of the data structure and track the two terms in $\boldsymbol{z}$ separately; then we correctly maintain $\boldsymbol{x}$. During PathFollowingLP, at step k, we first call Reweight and Move in InverseTreeOp, followed by Reweight and Move in each copy of TreeOp. The runtimes follow in a straightforward manner. $\square$

5.2 Dynamic vector approximation

In Section 2.7, we gave AbstractSolutionApproximation (Algorithm 6) for maintaining an approximate vector $\overline{\boldsymbol{x}}$ throughout PathFollowingLP, so that at every step,

$$\left\|\mathbf{D}^{1/2}\left(\overline{\boldsymbol{x}} - \boldsymbol{x}\right)\right\|_{\infty} \leq \delta,$$

where $\mathbf{D}$ is a dynamic diagonal scaling matrix that is a fixed entry-wise function of $\overline{\boldsymbol{x}}$.

In our data structure setting established in the previous section, we do not have full access to the exact vector $\boldsymbol{x}$ at a step, therefore, we cannot implement FindLargeCoordinates in Algorithm 6 naively. Instead, we make use of the tree operator $\mathcal{T}$ and limit access to $\boldsymbol{x}$ to two types: accessing the JL-sketches of subvectors corresponding to nodes of $\mathcal{T}$, and accessing exact coordinates and constant-sized subvectors corresponding to leaf nodes.

In Section 5.2.1, we detect coordinates of $\boldsymbol{x}$ with large changes using a sampling technique on a constant-degree tree, where Johnson-Lindenstrauss sketches of subvectors of $\boldsymbol{x}$ are maintained at each node the tree. In Section 5.2.2, we show how to compute and maintain the necessary collection of JL-sketches on the operator tree $\mathcal{T}$; in particular, we do this efficiently with only an implicit representation of $\boldsymbol{x}$. Finally, we put the two parts together in Section 5.2.3.

Recall we use the superscript $^{(k)}$ to denote the variable at the end of the k-th step of the IPM; that is, $\mathbf{D}^{(k)}$ and $\boldsymbol{x}^{(k)}$ are $\mathbf{D}$ and $\boldsymbol{x}$ at the end of the k-th step. Step 0 is the state of the data structure immediately after initialization.

5.2.1 From change detection to sketch maintenance

To implement FindLargeCoordinates(ℓ) required for Algorithm 6 to find the set $I_{\ell}^{(k)}$, we repeatedly sampling a coordinate i with probability proportional to $\mathbf{D}_{ii}^{(k-1)} \cdot |\boldsymbol{x}_i^{(k)} - \boldsymbol{x}_i^{(k-2^{\ell})}|^2$, among all coordinates i where $\overline{\boldsymbol{x}}_i$ has not been updated since 2^{ℓ} steps ago. With high probability, we can find all $i \in I_{\ell}^{(k)}$ in this way efficiently. To implement the sampling procedure, we make use of a data structure based on segment trees [49] along with sketching based on the Johnson-Lindenstrauss lemma.

Formally, at a step k, we define the vector $\boldsymbol{q} \in \mathbb{R}^m$ where $\boldsymbol{q}_i \overset{\text{def}}{=} \mathbf{D}_{ii}^{(k-1)}(\boldsymbol{x}_i^{(k)} - \boldsymbol{x}_i^{(k-2^\ell)})$ if $\overline{\boldsymbol{x}}_i$ has not been updated after the $k-2^\ell$-th step, and $\boldsymbol{q}_i = 0$ otherwise. Our goal is precisely to find all large coordinates of $\boldsymbol{q}$.

Let $\mathcal{T}$ be a constant-degree rooted tree with m leaves, where leaf i represents coordinate $\boldsymbol{q}_i$. For each node $u \in \mathcal{T}$, we define $E(u) \subseteq [m]$ to be set of indices of leaves in the subtree rooted at u. We make a random descent down $\mathcal{T}$, in order to sample a coordinate i with probability proportional to $\boldsymbol{q}_i^2$. At a node u, for each child u' of u, the total probability of the leaves under u' is given precisely by $\left\|\boldsymbol{q}|_{E(u')}\right\|_2^2$. We can estimate this by the Johnson-Lindenstrauss lemma using a sketching matrix $\boldsymbol{\Phi}$. Then we randomly move from u down to child u' with probability proportional to the estimated value. To tolerate the estimation error, when reaching some leaf node representing coordinate i, we accept with probability proportional to the ratio between the exact probability of i and the estimated probability of i. If i is rejected, we repeat the process from the root again independently.

Lemma 5.6. *Assume that $\|\mathbf{D}^{(k)}(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)})\|_2 \leq \beta$ for all IPM steps k. Let $\rho < 1$ be any given failure probability, let ℓ be a fixed granularity, and let $N \overset{\text{def}}{=} \Theta(2^{2\ell}(\beta/\delta)^2 \log^2 m \log(m/\rho))$ be the number of samples* FindLargeCoordinates(ℓ) *takes. Then with probability $\geq 1-\rho$, during the k step,* FindLargeCoordinates(ℓ) *(Section 5.2.1) finds the set $I_\ell^{(k)}$ correctly. Furthermore, the while-loop in Line 8 happens only $O(1)$ times in expectation per sample.*

Proof. The proof is similar to Lemma 6.17 in [58]. We include it for completeness. For a set S of indices, let $\mathbf{I}_S$ be the $m \times m$ diagonal matrix that is one on S and zero otherwise.

We first prove that Line 15 breaks with probability at least $\frac{1}{4}$. By the choice of w, Johnson–Lindenstrauss lemma shows that $\|\boldsymbol{\Phi}_{E(u)}\boldsymbol{q}\|_2^2 = (1 \pm \frac{1}{9\eta})\|\mathbf{I}_{E(u)}\boldsymbol{q}\|_2^2$ for all $u \in \mathcal{T}$ with probability at least $1-\rho$. Therefore, the probability we move from a node u to its child node u' is given by

$$\mathbf{P}(u \to u') = \left(1 \pm \frac{1}{3\eta}\right) \frac{\|\mathbf{I}_{E(u')}\boldsymbol{q}\|_2^2}{\sum_{u'' \text{ is a child of } u} \|\mathbf{I}_{E(u'')}\boldsymbol{q}\|_2^2} = \left(1 \pm \frac{1}{3\eta}\right) \frac{\|\mathbf{I}_{E(u')}\boldsymbol{q}\|_2^2}{\|\mathbf{I}_{E(u)}\boldsymbol{q}\|_2^2}.$$

Hence, the probability the walk ends at a leaf $u \in \mathcal{T}$ is given by

$$p_u = \left(1 \pm \frac{1}{3\eta}\right)^\eta \frac{\|\mathbf{I}_u \boldsymbol{q}\|_2^2}{\|\boldsymbol{q}\|_2^2} = (1 \pm \frac{1}{3\eta})^\eta \frac{\left\|\boldsymbol{q}|_{E(u)}\right\|^2}{\|\boldsymbol{q}\|_2^2}.$$

Now, p_{accept} on Line 15 is at least

$$p_{\text{accept}} = \frac{\left\|\boldsymbol{q}|_{E(u)}\right\|^2}{2 \cdot p_u \cdot \|\boldsymbol{\Phi}\boldsymbol{q}\|_2^2} \geq \frac{\left\|\boldsymbol{q}|_{E(u)}\right\|^2}{2 \cdot (1 + \frac{1}{3\eta})^\eta \frac{\left\|\boldsymbol{q}|_{E(u)}\right\|^2}{\|\boldsymbol{q}\|_2^2} \cdot \|\boldsymbol{\Phi}\boldsymbol{q}\|_2^2} \geq \frac{\|\boldsymbol{q}\|_2^2}{2 \cdot (1 + \frac{1}{3\eta})^\eta \|\boldsymbol{\Phi}\boldsymbol{q}\|_2^2} \geq \frac{1}{4}.$$

Algorithm 11 Implementation of FindLargeCoordinates at step k using a tree

```
1: procedure FindLargeCoordinates($\ell$)
2:     $\overline{\mathbf{D}}$: diagonal matrix such that
           $$\overline{\mathbf{D}}_{ii} = \begin{cases} \mathbf{D}_{ii}^{(k)} & \text{if } \overline{\boldsymbol{x}}_i \text{ has not been updated after the } (k-2^\ell)\text{-th step} \\ 0 & \text{otherwise.} \end{cases}$$
3:     $\boldsymbol{q} \overset{\text{def}}{=} \overline{\mathbf{D}}(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-2^\ell)})$        ▷ vector to sample coordinates from
4:
5:     $I \leftarrow \emptyset$        ▷ set of candidate coordinates
6:     for $N \overset{\text{def}}{=} \Theta(2^{2\ell}(\beta/\delta)^2 \log^2 m \log(m/\rho))$ iterations do
7:         ▷ Sample coordinate $i$ of $\boldsymbol{q}$ w.p. proportional to $\boldsymbol{q}_i^2$ by random descent down $\mathcal{T}$ to a leaf
8:         while true do
9:             $u \leftarrow \text{root}(\mathcal{T})$, $p_u \leftarrow 1$
10:            while $u$ is not a leaf node do
11:                Sample a child $u'$ of $u$ with probability
                       $$\mathbf{P}(u \to u') \overset{\text{def}}{=} \frac{\|\boldsymbol{\Phi}_{E(u')}\boldsymbol{q}\|_2^2}{\sum_{\text{child } u'' \text{ of } u} \|\boldsymbol{\Phi}_{E(u'')}\boldsymbol{q}\|_2^2}$$
                                   ▷ let $\boldsymbol{\Phi}_{E(u)} \overset{\text{def}}{=} \boldsymbol{\Phi}\mathbf{I}_{E(u)}$ for each node $u$
12:                $p_u \leftarrow p_u \cdot \mathbf{P}(u \to u')$
13:                $u \leftarrow u'$
14:            end while
15:            break with probability $p_{\text{accept}} \overset{\text{def}}{=} \left\|\boldsymbol{q}|_{E(u)}\right\|^2 / (2 \cdot p_u \cdot \|\boldsymbol{\Phi}\boldsymbol{q}\|_2^2)$
16:        end while
17:        $I \leftarrow I \cup E(u)$
18:    end for
19:    return $\{i \in I \ : \ \boldsymbol{q}_i \geq \frac{\delta}{2\lceil \log m \rceil}\}$.
20: end procedure
```

On the other hand, we have that $p_{\text{accept}} \leq \frac{\|\boldsymbol{q}\|_2^2}{2(1-\frac{1}{3\eta})^{\eta}\|\boldsymbol{\Phi q}\|_2^2} < 1$ and hence this is a valid probability.

Next, we note that u is accepted on Line 15 with probability

$$p_{\text{accept}} p_u = \frac{\left\|\boldsymbol{q}|_{E(u)}\right\|^2}{2 \cdot \|\boldsymbol{\Phi q}\|_2^2}.$$

Since $\|\boldsymbol{\Phi q}\|_2^2$ remains the same in all iterations, this probability is proportional to $\left\|\boldsymbol{q}|_{E(u)}\right\|^2$. Since the algorithm repeats when u is rejected, on Line 17, u is chosen with probability exactly $\left\|\boldsymbol{q}|_{E(u)}\right\|^2 / \|\boldsymbol{q}\|^2$.

Now, we want to show the output set is exactly $\{i \in [n] : |\boldsymbol{q}_i| \geq \frac{\delta}{2\lceil \log m \rceil}\}$. Let S denote the set of indices where $\overline{\boldsymbol{x}}$ did not update between the $(k - 2^{\ell})$-th step and the current k-th step. Then

$$\begin{aligned}
\|\boldsymbol{q}\|_2 &= \|\mathbf{I}_S(\mathbf{D}^{(k)})^{1/2}(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-2^{\ell})})\|_2 \\
&\leq \sum_{i=k-2^{\ell}}^{k-1} \|\mathbf{I}_S(\mathbf{D}^{(k)})^{1/2}(\boldsymbol{x}^{(i+1)} - \boldsymbol{x}^{(i)})\|_2 \\
&= \sum_{i=k-2^{\ell}}^{k-1} \|\mathbf{I}_S(\mathbf{D}^{(i+1)})^{1/2}(\boldsymbol{x}^{(i+1)} - \boldsymbol{x}^{(i)})\|_2 \\
&\leq \sum_{i=k-2^{\ell}}^{k-1} \|(\mathbf{D}^{(i+1)})^{1/2}(\boldsymbol{x}^{(i+1)} - \boldsymbol{x}^{(i)})\|_2 \\
&\leq 2^{\ell}\beta,
\end{aligned}$$

where we used $\mathbf{I}_S\mathbf{D}^{(i+1)} = \mathbf{I}_S\mathbf{D}^{(k)}$, because $\overline{\boldsymbol{x}}_i$ changes whenever $\mathbf{D}_{ii}$ changes at a step. Hence, each leaf u is sampled with probability at least $\left\|\boldsymbol{q}|_{E(u)}\right\|^2 / (2^{\ell}\beta)^2$. If $|\boldsymbol{q}_i| \geq \frac{\delta}{2\lceil \log m \rceil}$, and $i \in E(u)$ for a leaf node u, then the coordinate i is not in I with probability at most

$$\left(1 - \frac{\left\|\boldsymbol{q}|_{E(u)}\right\|^2}{(2^{\ell}\beta)^2}\right)^N \leq \left(1 - \frac{1}{2^{2\ell+2}(\beta/\delta)^2 \lceil \log m \rceil^2}\right)^N \leq \frac{\rho}{m},$$

by our choice of N. Hence, all i with $|\boldsymbol{q}_i| \geq \frac{\delta}{2\lceil \log m \rceil}$ lies in I with probability at least $1 - \rho$. This proves that the output set is exactly $I_{\ell}^{(k)}$ with probability at least $1 - \rho$. □

Remark 5.7. In Section 5.2.1, we only need to compute $\|\boldsymbol{\Phi}_{E(u)}\boldsymbol{q}\|_2^2$ for $O(N)$ many nodes $u \in \mathcal{T}$. Furthermore, the randomness of the sketch is not leaked and we can use the same random sketch $\boldsymbol{\Phi}$ throughout the algorithm. This allows us to efficiently maintain $\boldsymbol{\Phi}_{E(u)}\boldsymbol{q}$ for each $u \in \mathcal{T}$ throughout the IPM.

5.2.2 Sketch maintenance

In FindLargeCoordinates in the previous subsection, we assumed the existence of a constant degree sampling tree $\mathcal{T}$, and for the dynamic vector $\boldsymbol{q}$, the ability to access $\boldsymbol{\Phi}_{E(u)}\boldsymbol{q}$ at each node $u \in \mathcal{T}$ and $\boldsymbol{q}|_{E(u)}$ at each leaf node u.

In this section, we consider when the required sampling tree is the operator tree $\mathcal{T}$ supporting a tree operator $\boldsymbol{\Delta}$, and the vector $\boldsymbol{q}$ is $\boldsymbol{x} \stackrel{\text{def}}{=} \boldsymbol{\Delta z} + \sum_{H\in\mathcal{T}} \boldsymbol{\Delta}^{(H)}\boldsymbol{u}_H$, where each of $\boldsymbol{\Delta}, \boldsymbol{z}$ and the $\boldsymbol{u}_H$'s undergo changes at every IPM step. We present a data structure that implements two features efficiently on $\mathcal{T}$:

- access $\boldsymbol{x}|_{E(H)}$ at every leaf node H,
- access $\boldsymbol{\Phi}_{E(H)}\boldsymbol{x}$ at every node H, where $\boldsymbol{\Phi}_{E(H)}$ is $\boldsymbol{\Phi}$ restricted to columns given by $E(H)$.

Lemma 5.8. *Let $\mathcal{T}$ be a constant degree rooted tree with height η supporting tree operator $\boldsymbol{\Delta}$ with query complexity Q. Let $w = \Theta(\eta^2 \log(\frac{m}{\rho}))$, and let $\boldsymbol{\Phi} \in \mathbb{R}^{w\times m}$ be a JL-sketch matrix. Then* MaintainSketch *(Algorithm 12) is a data structure that maintains $\boldsymbol{\Phi x}$, where $\boldsymbol{x}$ is implicitly represented by*

$$\boldsymbol{x} \stackrel{\text{def}}{=} \boldsymbol{\Delta z} + \sum_{H\in\mathcal{T}} \boldsymbol{\Delta}^{(H)}\boldsymbol{u}_H.$$

The data structure supports the following procedures:

- Initialize*(operator tree $\mathcal{T}$, implicit $\boldsymbol{x}$): Initialize the data structure and compute the initial sketches in $O(Q(wm))$ time.*
- Update*($\mathcal{H} \subseteq \mathcal{T}$): Update all the necessary sketches in $O(w \cdot Q(\eta|\mathcal{H}|))$ time, where $\mathcal{H}$ is the set of all nodes H where $\boldsymbol{u}_H$ or $\boldsymbol{\Delta}_H$ changed.*
- Estimate$(H \in \mathcal{T})$*: Return $\boldsymbol{\Phi}_{E(H)}\boldsymbol{x}$.*
- Query$(H \in \mathcal{T})$*: Return $\boldsymbol{x}|_{E(H)}$.*

If we call Query *on N nodes, the total runtime is $O(Q(w\eta N))$.*

If we call Estimate *along a sampling path (by which we mean starting at the root, calling estimate at both children of a node, and then recursively descending to one child until reaching a leaf), and then we call* Query *on the resulting leaf, and we repeat this N times with no updates during the process, then the total runtime of these calls is $O(Q(w\eta N))$.*

We note that $\boldsymbol{\Delta z} = \sum_{H\in\mathcal{T}} \boldsymbol{\Delta}^{(H)}\boldsymbol{z}|_{F_H}$. For simplicity, it suffices to give the algorithm for sketching the simpler $\boldsymbol{x} \stackrel{\text{def}}{=} \sum_{H\in\mathcal{T}} \boldsymbol{\Delta}^{(H)}\boldsymbol{u}_H$.

Algorithm 12 Data structure for maintaining $\mathbf{\Phi}\boldsymbol{x}$, Part 1

```
1: data structure MaintainSketch
2: private : member
3:     $\mathcal{T}$ : rooted constant degree tree, where at every node $H$, there is
4:         $\mathbf{S}^{(H)} \in \mathbb{R}^{w \times |F_H \cup \partial H|}$ : sketched subtree operator $\mathbf{\Phi}\mathbf{\Delta}^{(H)}$
5:         $\boldsymbol{t}^{(H)} \in \mathbb{R}^{w}$ : sketched vector $\mathbf{\Phi} \sum_{H' \in \mathcal{T}_H} \mathbf{\Delta}^{(H')} \boldsymbol{u}_{H'}$
6:     $\mathbf{\Phi} \in \mathbb{R}^{w \times m}$ : JL-sketch matrix
7:     $\mathbf{\Delta} \in \mathbb{R}^{m \times n}$ : dynamic tree operator on $\mathcal{T}$
8:     $\boldsymbol{u}_H$ at every $H \in \mathcal{T}$ : dynamic vectors
9:
10: procedure Initialize(tree $\mathcal{T}$, $\mathbf{\Phi} \in \mathbb{R}^{w \times m}$, tree operator $\mathbf{\Delta}$, $\boldsymbol{u}_H$ for each $H \in \mathcal{T}$)
11:     $\mathbf{\Phi} \leftarrow \mathbf{\Phi}, \mathcal{T} \leftarrow \mathcal{T}, \mathbf{\Delta} \leftarrow \mathbf{\Delta}, \boldsymbol{u}_H \leftarrow \boldsymbol{u}_H$ for each $H \in \mathcal{T}$
12:     $\mathbf{S}^{(H)} \leftarrow \mathbf{0}$, $\boldsymbol{t}^{(H)} \leftarrow \mathbf{0}$ for each $H \in \mathcal{T}$
13:     Update($V(\mathcal{T})$)
14: end procedure
15:
16: procedure Update($\mathcal{H} \stackrel{\text{def}}{=}$ set of nodes admitting implicit representation changes)
17:     for $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$ going up the tree level by level do
18:         $\mathbf{S}^{(H)} \leftarrow \sum_{\text{child } D \text{ of } H} \mathbf{S}^{(D)} \mathbf{\Delta}_D$
19:         $\boldsymbol{t}^{(H)} \leftarrow \mathbf{S}^{(H)} \boldsymbol{u}_H + \sum_{\text{child } D \text{ of } H} \boldsymbol{t}^{(D)}$
20:     end for
21: end procedure
22:
23: procedure SumAncestors($H \in \mathcal{T}$)
24:     if Update has not been called since the last call to SumAncestors($H$) then
25:         return the result of the last SumAncestors($H$)
26:     end if
27:     if $H$ is the root then return $\mathbf{0}$
28:     end if
29:     return $\mathbf{\Delta}_H(\boldsymbol{u}_P + \text{SumAncestors}(P))$          $\triangleright$ $P$ is the parent of $H$
30: end procedure
```

Algorithm 13 Data structure for maintaining $\mathbf{\Phi}\boldsymbol{x}$, Part 2

```
1: data structure MaintainSketch
2: procedure Estimate($H \in \mathcal{T}$)
3:     Let $\boldsymbol{y}$ be the result of SumAncestors($H$)
4:     return $\mathbf{S}^{(H)}\boldsymbol{y} + \boldsymbol{t}^{(H)}$
5: end procedure
6:
7: procedure Query(leaf $H \in \mathcal{T}$)
8:     return $\boldsymbol{u}_H$ + SumAncestors($H$)
9: end procedure
```

Proof. Let us consider the correctness of the data structure, starting with the helper function SumAncestors. We implement it using recursion and memoization as it is crucial for bounding subsequent runtimes.

Claim 5.9. SumAncestors($H \in \mathcal{T}$) *returns* $\sum_{\text{ancestor } A \text{ of } H} \mathbf{\Delta}_{H \leftarrow A} \boldsymbol{u}_A$.

Proof. At the root, there are no ancestors, hence we return the zero matrix. When H is not the root, suppose P is the parent of H. Then we can recursively write

$$\sum_{\text{ancestor } A \text{ of } H} \mathbf{\Delta}_{H \leftarrow A} \boldsymbol{u}_A = \mathbf{\Delta}_H \left(\boldsymbol{u}_P + \sum_{\text{ancestor } A \text{ of } P} \mathbf{\Delta}_{P \leftarrow A} \boldsymbol{u}_A \right).$$

The procedure implements the right hand side, and is therefore correct. □

Assuming we correctly maintain $\mathbf{S}^{(H)} \stackrel{\text{def}}{=} \mathbf{\Phi}\mathbf{\Delta}^{(H)}$ and $\boldsymbol{t}^{(H)} \stackrel{\text{def}}{=} \mathbf{\Phi} \sum_{H' \in \mathcal{T}_H} \mathbf{\Delta}^{(H')} \boldsymbol{u}_{H'}$ at every node H, Estimate and Query return the correct answers by the tree operator decomposition given in Lemma 3.41.

For Update, note that if a node H is not in $\mathcal{H}$ and it has no descendants in $\mathcal{H}$, then by definition, the sketches at H are not changed. Hence, it suffices to update the sketches only at all nodes in $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$. We update the nodes from the bottom of $\mathcal{T}$ upwards, so that when we are at a node H, all the sketches at its descendant nodes are correct. Therefore, by definition, the sketches at H is also correct.

Now we consider the runtimes:

Initialize: It sets the sketches to $\mathbf{0}$ in $O(wm)$ time, and then calls Update to update the sketches everywhere on $\mathcal{T}$. By the correctness runtime of Update, this step is correct and runs in $\widetilde{O}(Q(wm))$ time.

Update(set of nodes $\mathcal{H}$ admitting implicit representation changes): First note that $|\mathcal{P}_{\mathcal{T}}(\mathcal{H})| \leq \eta|\mathcal{H}|$. For each node $H \in \mathcal{H}$ with children D_1, D_2, Line 18 multiplies each row of $\mathbf{S}^{(D_1)}$ with $\mathbf{\Delta}_{(D_1,H)}$, each row of $\mathbf{S}^{(D_2)}$ with $\mathbf{\Delta}_{D_2}$, and sums the results. Summing over w-many rows and over all nodes in $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$, we see the total runtime of Line 18 is $O(Q(w\eta|\mathcal{H}|))$.

Line 19 multiply each row of $\mathbf{S}^{(H)}$ with a vector and then performs a constant number of additions of w-length vectors. Since $\mathbf{S}^{(H)}$ is computed for all $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$ in $O(Q(w\eta|\mathcal{H}|))$ total time, this must also be a bound on their number of total non-zero entries. Since each $\mathbf{S}^{(H)}$ is used once in Line 19 for a matrix-vector multiplication, the total runtime of Line 19 is $O(Q(w\eta|\mathcal{H}|))$.

All other lines are not bottlenecks.

Overall Estimate and Query time along N sampling paths: We show that if we call Estimate along N sampling paths each from the root to a leaf, and we call Query on the leaves, the total cost is $O(Q(w\eta N))$:

Suppose the set of nodes visited is given by $\mathcal{H}$, then $|\mathcal{H}| \leq \eta N$. Since there is no update, and Estimate is called for a node only after it is called for its parent, we know that SumAncestors(H) is called exactly once for each $H \in \mathcal{H}$. Each SumAncestor(H) multiplies a unique edge operator $\mathbf{\Delta}_{(H,P)}$ with a vector. Hence, the total runtime of SumAncestors is $Q(|\mathcal{H}|)$.

Finally, each Query applies a leaf operator to the output of a unique SumAncestors call, so the overall runtime is certainly bounded by $O(Q(|\mathcal{H}|))$. Similarly, each Estimate multiplies $\mathbf{S}^{(H)}$ with the output of a unique SumAncestors call. This can be computed as w-many vectors each multiplied with the SumAncestors output. Then two vectors of length w are added. Summing over all nodes in $\mathcal{H}$, the overall runtime is $O(Q(w|\mathcal{H}|)) = O(Q(w\eta N))$.

Query time on N leaves: Since this is a subset of the work described above, the runtime must also be bounded by $O(Q(w\eta N))$.

□

5.2.3 Overall approximation guarantees

We combine AbstractSolutionApproximation and Theorem 2.26 with implementations from the previous two sections for the overall data structure:

Theorem 5.10 (Approximation scheme with tree operator)**.** *Let $0 < \rho < 1$ be a failure probability, and let δ be an error tolerance. Suppose $\boldsymbol{\Delta} \in \mathbb{R}^{m \times n}$ is a tree operator with query complexity Q, supported on a constant-degree tree $\mathcal{T}$ with height η. There is a data structure which maintains an approximation $\overline{\boldsymbol{x}}$ of $\boldsymbol{x}$ throughout* PathFollowingLP *where $\mathbf{D}$ is a diagonal scaling matrix that is a fixed entrywise function of $\overline{\boldsymbol{x}}$. Across all steps, $\overline{\boldsymbol{x}}, \mathbf{D}$ in the data structure satisfy $\|\mathbf{D}(\boldsymbol{x} - \overline{\boldsymbol{x}})\|_\infty \leq \delta$ with probability $1 - \rho$.*

Furthermore, suppose $\|\mathbf{D}^{(k-1)}(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)})\|_2 \leq \beta$ for all steps k. Then, $O(2^{2\ell}(\beta/\delta)^2 \log^2 m)$ coordinates of $\overline{\boldsymbol{x}}$ are updated every 2^ℓ steps for each $\ell \geq 0$. Over N total steps, the total cost of the data structure is

$$\widetilde{O}(\eta^3(\beta/\delta)^2 \log^3(mN/\rho)) \left(Q(m) + \sum_{k=1}^{N} Q(S^{(k)}) + \sum_{\ell=0}^{\log N} \frac{N}{2^\ell} \cdot Q(2^{2\ell}) \right), \tag{5.3}$$

where $S^{(k)}$ is the number of nodes H at which the implicit representation of $\boldsymbol{x}$ changed at step k.

We omit the pseudocode and give only the algorithm descriptions.

Proof. $\boldsymbol{x}$ is maintained by MaintainRep. We create $O(\log m)$ copies of MaintainSketch as given in Lemma 5.8, so that for each $0 \leq \ell \leq O(\log m)$, we have one copy $\texttt{sketch}_{\ell,x}$ which maintains sketches of $\boldsymbol{\Phi}\overline{\mathbf{D}}\boldsymbol{x}^{(k)}$ at step k, and one copy $\texttt{sketch}_\ell$ which maintains sketches of $\boldsymbol{\Phi}\overline{\mathbf{D}}\boldsymbol{x}^{(k-2^\ell)}$ at step $k \geq 2^\ell$, where $\overline{\mathbf{D}}$ is defined so $\overline{\mathbf{D}}_{i,i} = \mathbf{D}_{i,i}$ if $\overline{\boldsymbol{x}}_i$ has not been updated after the $k - 2^\ell$-th step, and $\overline{\mathbf{D}}_{i,i} = 0$ otherwise (as needed in Section 5.2.1). Note that $\overline{\mathbf{D}}$ can be absorbed into the tree operator in the implicit representation of $\boldsymbol{x}$, so Lemma 5.8 does indeed apply.

To access skeches of the vector $\boldsymbol{q} \stackrel{\text{def}}{=} \overline{\mathbf{D}}(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-2^\ell)})$ as needed in FindLargeCoordinates in Section 5.2.1, we can simply access the corresponding sketch in $\texttt{sketch}_{\ell,x}$ and $\texttt{sketch}_\ell$, and then take the difference.

We now describe each procedure, and then prove their correctness and runtime.

Initialize: We initialize the $O(\log m)$ copies of MaintainSketch in $O(Q(wm)\log m)$ time by Lemma 5.8.

UPDATE($\boldsymbol{x}^{(\text{new})}, \mathbf{D}^{(\text{new})}$): To implement UPDATE, it suffices to update all the sketching data structures. Let us fix ℓ, and consider the update time for $\texttt{sketch}_{\ell,x}$ and $\texttt{sketch}_{\ell}$.

Theorem 2.26 shows there are $O(2^{2\ell}(\beta/\delta)^2 \log^2 m)$-many coordinate updates to $\overline{\boldsymbol{x}}$ every 2^ℓ steps. Since $\mathbf{D}$ is a function of $\overline{\boldsymbol{x}}$ coordinate-wise, $\overline{\boldsymbol{x}}_i = \boldsymbol{x}_i^{(k-1)}$ for all i where $\mathbf{D}_{ii}^{(k)} \neq \mathbf{D}_{ii}^{(k-1)}$. The diagonal matrix $\overline{\mathbf{D}}$ is the same as $\mathbf{D}$, except $\overline{\mathbf{D}}_{ii}$ is temporarily zeroed out for 2^ℓ steps after $\overline{\boldsymbol{x}}_i$ changes at a step. So, the overall number of coordinate changes to $\overline{\mathbf{D}}$ is $O(2^{2\ell})$-many every 2^ℓ steps.

Let $S^{(k)}$ denote the number of nodes H where $\boldsymbol{\Delta}_H$ or $\boldsymbol{u}_H$ in the implicit representation of $\boldsymbol{x}$ changed at step k. Additionally, since the sketching data structures maintain some variant of $\overline{\mathbf{D}}\boldsymbol{x}$ (where $\overline{\mathbf{D}}$ is viewed as absorbed in the tree operator), every coordinate change in $\overline{\mathbf{D}}$ implies an edge operator update. Now we apply Lemma 5.8 to conclude that the total time for all UPDATE calls for $\texttt{sketch}_{\ell,x}$ and $\texttt{sketch}_{\ell}$ over N steps is:

$$O(1) \cdot \left(\sum_{k=1}^{N} Q\left(w\eta S^{(k)}\right) + \frac{N}{2^\ell} \cdot Q(w\eta \cdot 2^{2\ell}) \right) \leq O(w\eta) \cdot \left(\sum_{k=1}^{N} Q(S^{(k)}) + \frac{N}{2^\ell} \cdot Q(2^{2\ell}) \right).$$

We then sum this over all ℓ to get the total update time for the sketching data structures.

APPROXIMATE: There are two operations to be implemented in the subroutine FINDLARGECOORDINATES(ℓ): Accessing $\boldsymbol{\Phi}_{E(u)}\boldsymbol{q}$ at a node u, and accessing $\boldsymbol{q}|_{E(u)}$ at a leaf node u. For the first, we call $\texttt{sketch}_{\ell,x}$.ESTIMATE(u) $-$ $\texttt{sketch}_{\ell}$.ESTIMATE(u). For the second, we call $\texttt{sketch}_{\ell,x}$.QUERY(u) $-$ $\texttt{sketch}_{\ell}$.QUERY(u).

To set $\overline{\boldsymbol{x}}_i$ as $\boldsymbol{x}_i^{(k)}$ for a single coordinate at step k, we find the leaf node H containing the edge e, and call $\texttt{sketch}_{0,x}$.QUERY(H). This returns the sub-vector $\boldsymbol{x}^{(k)}|_{E(H)}$, from which we can extract $\boldsymbol{x}_i^{(k)}$ and set $\overline{\boldsymbol{x}}_i$ to be the value. This line is not a bottleneck in the runtime.

We compute the total runtime over N APPROXIMATE calls. For every $\ell \geq 0$, we call FINDLARGECOORDINATES(ℓ) once every 2^ℓ steps, for a total of $N/2^\ell$ calls. In a single call, $M_\ell \overset{\text{def}}{=} \Theta(2^{2\ell}(\beta/\delta)^2 \log^2 m \log(mN/\rho))$ sampling paths are explored in the $\texttt{sketch}_{\ell}$ and $\texttt{sketch}_{\ell,x}$ data structures by Lemma 5.6, where a sampling path correspond to one iteration of the while-loop. This takes a total of $O(Q(w\eta M_\ell))$ time by Lemma 5.8. Therefore, for every fixed ℓ, the total time for all FINDLARGECOORDINATES(ℓ) calls is

$$\frac{N}{2^\ell} \cdot O\left(Q(w\eta M_\ell)\right).$$

The total time for all LARGECOORDINATES calls is obtained by summing over all values of $\ell = 0, \ldots, \log N$. To achieve overall failure probability at most ρ, it suffices to set the failure probability of each call to be $O(\rho/N)$. $\square$

We sum up the initialization time, update and approximate time for all values of $\ell = 0, \dots, \log N$ and over N total IPM steps, to get the overall runtime of the data structure:

$$\begin{aligned}
&\widetilde{O}(Q(wm)) + O(w\eta)\sum_{k=1}^{N} Q(S^{(k)}) + O(w\eta)\sum_{\ell=0}^{\log N} \frac{N}{2^\ell}\left(Q(2^{2\ell}) + Q(M_\ell)\right) \\
&= \widetilde{O}(\eta^3(\beta/\delta)^2 \log^3(mN/\rho))\left(Q(m) + \sum_{k=1}^{N} Q(S^{(k)}) + \sum_{\ell=0}^{\log N} \frac{N}{2^\ell} \cdot Q(2^{2\ell})\right).
\end{aligned}$$

5.3 Main IPM data structure theorem

We are now ready to state and prove the main result in this framework.

Theorem 5.11 (RIPM framework)**.** *Consider an LP of the form*

$$\min_{\boldsymbol{x} \in \mathcal{P}} \boldsymbol{c}^\top \boldsymbol{x} \quad \textit{where} \quad \mathcal{P} = \{\mathbf{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{l} \le \boldsymbol{x} \le \boldsymbol{u}\} \tag{5.4}$$

where $\mathbf{A} \in \mathbb{R}^{n\times m}$. For any vector $\boldsymbol{w}$, let $\mathbf{P}_{\boldsymbol{w}} \stackrel{\text{def}}{=} \mathbf{W}^{1/2}\mathbf{A}^\top(\mathbf{A}\mathbf{W}\mathbf{A}^\top)^{-1}\mathbf{A}\mathbf{W}^{1/2}$, and suppose there exists dynamic tree and inverse tree operators $\boldsymbol{\Delta}$ and ∇ dependent on $\boldsymbol{w}$, such that $\mathbf{W}^{1/2}\mathbf{P}_{\boldsymbol{w}} = \boldsymbol{\Delta}\nabla$. Let U be the update complexity of $\boldsymbol{\Delta}$ and ∇, and let Q be their query complexity. Let r and $R = \|\boldsymbol{u} - \boldsymbol{l}\|_2$ be the inner and outer radius of $\mathcal{P}$, and let $L = \|\boldsymbol{c}\|_2$. Then, there is a data structure to solve Eq. (5.4) *to εLR accuracy with probability $1 - 2^{-m}$ in time*

$$\widetilde{O}\left(\eta^4\sqrt{m}\log(\frac{R}{\varepsilon r}) \cdot \sum_{\ell=0}^{\frac{1}{2}\log m} \frac{U(2^{2\ell}) + Q(2^{2\ell})}{2^\ell}\right).$$

Proof. We implement PathFollowingLP using the data structures from the previous chapters, and bound the cost of each operations of the data structures. For simplicity, we only discuss the primal variables in this proof, but the slack variables are analogous. We use one copy of MaintainRep to maintain $\boldsymbol{x}$, and one copy of MaintainApprox to maintain $\overline{\boldsymbol{x}}$. At each step, we perform the implicit update of $\boldsymbol{x}$ using Move and update $\boldsymbol{w}$ using Reweight in MaintainRep. We construct the explicit approximations $\overline{\boldsymbol{x}}$ using Approximate in MaintainApprox.

Theorem 5.10 shows that throughout the IPM, for each $\ell \ge 0$, there are $2^{2\ell}$ coordinate changes to $\overline{\boldsymbol{x}}$ every 2^ℓ steps. Since $\boldsymbol{w}$ is a function of $\overline{\boldsymbol{x}}$ coordinate-wise, there are also $2^{2\ell}$ coordinate changes in $\boldsymbol{w}$ every 2^ℓ steps. Similarly, we observe that $\boldsymbol{v}$ is defined as a function of $\overline{\boldsymbol{x}}$ and $\overline{\boldsymbol{s}}$ coordinate-wise, so there are $O(2^{2\ell})$ coordinate changes to $\boldsymbol{v}$ every 2^ℓ steps. Then

Theorem 5.1 shows that the total runtime over N steps for the MAINTAINREP data structure is

$$\widetilde{O}(U(m)+Q(m))+\widetilde{O}\left(\sum_{\ell=0}^{\log N}\frac{N}{2^{\ell}}\cdot\left(U(\eta\cdot 2^{2\ell})+Q(\eta\cdot 2^{2\ell})\right)\right). \tag{5.5}$$

Theorem 5.10 shows that the total runtime over N steps for MAINTAINAPPROX is

$$\widetilde{O}(\eta^3(\beta/\delta)^2\log^3(mN/\rho))\left(Q(m)+\sum_{k=1}^{N}Q(S^{(k)})+\sum_{\ell=0}^{\log N}\frac{N}{2^{\ell}}\cdot Q(2^{2\ell})\right), \tag{5.6}$$

where the variables are defined as in the theorem statement. By examining Theorem 5.1, we see that when a coordinate of $\boldsymbol{w}$ or $\boldsymbol{v}$ changes, the implicit representation of $\boldsymbol{x}$ admits updates at $O(\eta)$-many nodes. Combined with the concavity of Q, we can bound

$$\sum_{k=1}^{N}Q(S^{(k)})\leq O(\eta)\cdot\sum_{\ell=0}^{\log N}\frac{N}{2^{\ell}}\cdot Q(2^{2\ell}).$$

Theorem 2.24 guarantees that there are $N=\sqrt{m}\log m\log(\frac{mR}{\varepsilon r})$ total IPM steps, and at each step k, we have $\left\|\mathbf{W}^{(k-1)^{-1/2}}(\boldsymbol{x}^{(k)}-\boldsymbol{x}^{(k-1)})\right\|_2=\left\|\boldsymbol{v}^{(k)}-\mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}^{(k)}\right\|_2\leq O(\frac{1}{\log m})$, so we can set $\beta=O(\frac{1}{\log m})$. By examining the algorithm, we see it suffices to set $\delta=O(\frac{1}{\log m})$. We choose the failure probability ρ to be appropriately small, e.g. 2^{-m}. Finally, we conclude that the overall runtime of the IPM framework is

$$\widetilde{O}\left(\eta^4\sqrt{m}\log(\frac{R}{\varepsilon r})\cdot\sum_{\ell=0}^{\frac{1}{2}\log m}\frac{U(2^{2\ell})+Q(2^{2\ell})}{2^{\ell}}\right),$$

where the terms for initialization times have been absorbed.

□

Chapter 6

Structured linear programming

In this chapter, we finally present the main results for structured linear programming. We apply Lemma 4.5 to the separator tree for each setting get the complexity of the tree operator, which we combine with Theorem 5.11 to conclude the overall IPM runtimes.

6.1 Separable linear programs

Theorem 6.1. *Given a linear program* $\min\{\boldsymbol{c}^\top \boldsymbol{x} \,:\, \mathbf{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{l} \le \boldsymbol{x} \le \boldsymbol{u}\}$*, where* $\mathbf{A} \in \mathbb{R}^{n\times m}$ *is a full-rank matrix with* $n \le m$*, suppose the dual graph* $G_{\mathbf{A}}$ *is* $O(n^\alpha)$*-separable with* $\alpha \in [0,1]$*, and a balanced separator is computable in* $T(n)$ *time.*

Suppose that r *is the inner radius of the polytope, namely, there is* $\boldsymbol{x}$ *such that* $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ *and* $\boldsymbol{l} + r \le \boldsymbol{x} \le \boldsymbol{u} - r$*. Let* $L = \|\boldsymbol{c}\|_2$ *and* $R = \|\boldsymbol{u} - \boldsymbol{l}\|_2$*. Then, for any* $0 < \varepsilon \le 1/2$*, we can find a feasible* $\boldsymbol{x}$ *with high probability such that*

$$\boldsymbol{c}^\top \boldsymbol{x} \le \min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b},\, \boldsymbol{l}\le\boldsymbol{x}\le\boldsymbol{u}} \boldsymbol{c}^\top \boldsymbol{x} + \varepsilon \cdot LR,$$

in time

$$\widetilde{O}\left((m + m^{1/2+2\alpha}) \cdot \log(R/(r\varepsilon)) + T(n)\right).$$

Proof. We consider the cases when $\alpha = 1$ and $\alpha < 1$ separately.

All hypergraphs are trivially n-separable with max hyperedge size $\rho = n$. In this case, let $\mathcal{S}$ be the separator tree consisting of simply one node representing $G_{\mathbf{A}}$, which is a $(1, 1/2, n)$-separator tree. By Lemma 4.5, the tree operator data structure can be initialized in $O(m^\omega)$ time; the query complexity is $Q(K) = O(nK + n^2)$, and the update complexity is $U(K) = O(n^{\omega-1}K + n^2K^{\omega-2})$.

We apply Theorem 5.11 to get the overall runtime:

$$\widetilde{O}\left(\sqrt{m}\log(\frac{R}{\varepsilon r}) \cdot \sum_{\ell=0}^{\frac{1}{2}\log m} \frac{n2^{2\ell} + n^2 + n^2 2^{2\ell(\omega-2)}}{2^\ell}\right) = \widetilde{O}\left(\sqrt{m}n^2\log(\frac{R}{\varepsilon r})\right).$$

If $G_{\mathbf{A}}$ is n^{α}-separable for $\alpha < 1$, then by Lemma 3.14, $G_{\mathbf{A}}$ admits a (α, b, cn^{α})-separator tree computable in $\widetilde{O}(n)$ time. In this case, $\rho = O(1)$, and $\eta = O(\log_{1/b} n)$. Plugging the parameters into Lemma 4.5, we get the following tree operator runtimes:

- The data structure can be initialize in $O\left(m + n^{\alpha\omega}(1 + n^{1-\alpha\omega})\right) \le O(m + m^{\alpha\omega})$ time.
- The query complexity is $Q(K) \le O(K + n^{2\alpha}(1 + K^{1-2\alpha}))$.
- The update complexity is

$$U(K) \le O\left(K + n^{2\alpha}\min\{K, n^{\alpha}\}^{\omega-2} + n^{2\alpha}K^{1-2\alpha} + n^{\frac{\alpha(\omega-1)}{1-\alpha}}K^{\frac{1-\alpha\omega}{1-\alpha}} \cdot \mathbb{1}_{K \ge n^{\alpha}}\right).$$

We apply Theorem 5.11 to get the overall runtime:

$$\begin{aligned}
&\widetilde{O}\left(\sqrt{m}\log(\frac{R}{\varepsilon r})\right) \cdot \sum_{\ell=0}^{\frac{1}{2}\log m} \frac{2^{2\ell} + n^{2\alpha} + n^{\alpha\omega} \cdot \mathbb{1}_{2^{2\ell} > n^{\alpha}} + n^{2\alpha}2^{(1-2\alpha)2\ell} + n^{\frac{\alpha(\omega-1)}{1-\alpha}} 2^{\frac{1-\alpha\omega}{1-\alpha}2\ell} \cdot \mathbb{1}_{2^{2\ell} > n^{\alpha}}}{2^{\ell}} \\
&= \widetilde{O}\left(\sqrt{m}\log(\frac{R}{\varepsilon r})\right) \cdot \left(\sqrt{m} + n^{2\alpha} + n^{\alpha\omega - \frac{\alpha}{2}} + n^{2\alpha}m^{1-2\alpha-\frac{1}{2}} + n^{\frac{\alpha(\omega-1)}{1-\alpha}}\left(n^{\frac{\alpha(1-\alpha\omega)}{1-\alpha} - \frac{\alpha}{2}} + m^{\frac{1-\alpha\omega}{1-\alpha} - \frac{1}{2}}\right)\right) \\
&= \widetilde{O}\left(\left(m + m^{\frac{1}{2}+2\alpha}\right) \cdot \log(\frac{R}{\varepsilon r})\right),
\end{aligned}$$

where in the last step, we used the fact $\alpha\omega - \frac{\alpha}{2} \le 2\alpha$. □

6.2 k-commodity flow

An immediate application of Theorem 6.1 is a faster algorithm for solving the fractional k-commodity flow problem on planar graphs to high accuracy. For general sparse graphs, an $\widetilde{O}((km)^{\omega})$ time algorithm for this problem follows by the recent linear program solvers that run in matrix multiplication time [44, 29]. It is known that solving the k-commodity flow problem is as hard as linear programming [91, 55], suggesting that additional structural assumptions on the input graph are necessary to obtain faster algorithms. As shown in the theorem below, our result achieves a polynomial speed-up when the input graph is planar.

Theorem 6.2. *Given a planar graph $G = (V, E)$ on n vertices and m edges, with edge-vertex incidence matrix $\mathbf{B}$, integer edge capacities $\boldsymbol{u} \in \mathbb{R}^m_{\ge 0}$, integer edge costs $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k \in \mathbb{R}^m$ and integer demands $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_k \in \mathbb{R}^m$ for each commodity, we can solve the minimum-cost*

k-multicommodity flow problem on G, equivalent to the following linear program,

$$\begin{aligned} \min \quad & \sum_{i=1}^{k} \boldsymbol{c}_i^\top \boldsymbol{f}_i \\ s.t \quad & \mathbf{B}^\top \boldsymbol{f}_i = \boldsymbol{d}_i \qquad \forall i \in [k] \\ & \sum_{i=1}^{k} \boldsymbol{f}_i \leq \boldsymbol{u} \\ & \boldsymbol{f}_i \geq \mathbf{0} \qquad \forall i \in [k] \end{aligned} \tag{6.1}$$

to ϵ accuracy in $\widetilde{O}(k^{2.5} m^{1.5} \log(M/\varepsilon))$ time, where M is an upper on the absolute values of $\boldsymbol{u}, \boldsymbol{c}, \boldsymbol{d}$.

Let $G = (V, E)$ denote the planar graph for the original problem, with $V = \{v_1, \dots, v_n\}$ and $E = \{e_1, \dots, e_m\}$. First, we write the LP in Eq. (6.1) in standard form by adding slack variables $\boldsymbol{s} \in \mathbb{R}^E$:

$$\begin{aligned} \min \quad & \sum_{i=1}^{k} \boldsymbol{c}_i^\top \boldsymbol{f}_i \\ \text{s.t} \quad & \mathbf{B}^\top \boldsymbol{f}_i = \boldsymbol{d}_i \qquad \forall i \in [k] \\ & \sum_{i=1}^{k} \boldsymbol{f}_i + \boldsymbol{s} = \boldsymbol{u} \\ & \boldsymbol{f}_i \geq \mathbf{0} \qquad \forall i \in [k] \\ & \boldsymbol{s} \geq \mathbf{0} \end{aligned} \tag{P'}$$

Let $\mathbf{A}$ denote the full constraint matrix of P'. Then

$$\mathbf{A} = \left[\begin{array}{cccc|c} \mathbf{B}^\top & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}^\top & & \vdots & \vdots \\ & & \ddots & & \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{B}^\top & \mathbf{0} \\ \hline \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} & \mathbf{I} \end{array}\right] \in \mathbb{R}^{(kn+m)\times(k+1)m} \tag{6.2}$$

where the top left part of $\mathbf{A}$ contains k copies of $\mathbf{B}^\top$ in block-diagonal fashion, and all the identity matrices are of dimension $m \times m$. The dual graph of $\mathbf{B}^\top$ is precisely G. Let $G_{\mathbf{A}}$ be the dual graph of $\mathbf{A}$.

First, we describe $G_{\mathbf{A}}$: It contains k independent copies of the vertices V, which we label with $V^i = (v_1^i, \dots, v_n^i)$, so that v_j^i is a copy of $v_j \in V$. Additionally, $G_{\mathbf{A}}$ contains m vertices $u_1, \dots, u_m$, where the vertex u_i is identified with edge $e_i \in E$. For each edge $e_i \in E$ with endpoints v_{i_1}, v_{i_2}, there are k hyper-edges in $G_{\mathbf{A}}$ of the form $\{v_{i_1}^{\ell}, v_{i_2}^{\ell}, u_i\}$ for $\ell = 1, \dots, k$. Additionally, there are m hyper-edges $f_1, \dots, f_m$ where f_i contains only the vertex u_i.

Next, we show how to construct an appropriate separator tree efficiently.

Claim 6.3. *$G_{\mathbf{A}}$ admits a $(\frac{1}{2}, b, kn^{1/2})$-separator tree that can be computed in $O(kn \log n)$ time.*

Proof. Let G be the original planar graph which is $\sqrt{n}$-separable, and let $\tilde{\mathcal{S}}$ be the $(\frac{1}{2}, b, n^{1/2})$-separator tree for G constructed using Lemma 3.14 in $O(n \log n)$ time by [122]. We show how to construct a $(\frac{1}{2}, b, kn^{1/2})$-separator tree $\mathcal{S}$ for $G_{\mathbf{A}}$ based on $\tilde{\mathcal{S}}$. Without loss of generality, we ignore the hyper-edges $f_1, \dots f_m$ in this construction.

Intuitively, $\mathcal{S}$ will have the same tree structure as $\tilde{\mathcal{S}}$, but each node will be larger by a factor of $O(k)$ due to the k copies of G in $G_{\mathbf{A}}$. For each $\tilde{H} \in \tilde{\mathcal{S}}$, we construct a corresponding $H \in \mathcal{S}$ as follows: if $v_j \in \tilde{H}$, then $v_j^i \in H$ for all $i \in [k]$; if $e_j \in E(\tilde{H})$, i.e. both endpoints of e_j are in $\tilde{H}$, add u_j to H. Since the k copies $v_j^1, \dots, v_j^k$ are always grouped together, we will refer to them together as v_j in $G_{\mathbf{A}}$ as well.

Let us show that this is indeed a $(\frac{1}{2}, b, kn^{1/2})$-separator tree. Suppose H is a node with children D_1 and D_2 in $\mathcal{S}$, corresponding to nodes $\tilde{H}, \tilde{D}_1, \tilde{D}_2$ in $\tilde{\mathcal{S}}$. Let $S(H) \stackrel{\text{def}}{=} V(D_1) \cap V(D_2)$, then $v_j \in S(H)$ iff $v_j \in S(\tilde{H})$, and $u_j \in S(H)$ iff $e_j \in E(S(\tilde{H}))$ for all values of j. It is straightforward to see that $S(H)$ is indeed a separator of H. When it comes to the set of boundary vertices, we see $v_j \in \partial H$ iff $v_j \in \partial \tilde{H}$, and if $u_j \in \partial H$ with v_{j_1}, v_{j_2} being the two endpoints of e_j, then v_{j_1}, v_{j_2} are both in ∂H. Since G is a planar graph, the number of edges in $\tilde{H}$ is on the same order as the number of vertices, so we conclude that $|V(H)| \leq O(k) \cdot |V(\tilde{H})|$, and similarly, $|F_H \cup \partial H| \leq O(k) \cdot |F_{\tilde{H}} \cup \partial \tilde{H}|$. Since node sizes in $\mathcal{S}$ have increased by a factor of $O(k)$ compared to $\tilde{\mathcal{S}}$, we conclude $\mathcal{S}$ is a $(\frac{1}{2}, b, kn^{1/2})$-separator tree.

Finally, we can compute $\tilde{\mathcal{S}}$ for G in $O(n \log n)$ time, so we can compute $\mathcal{S}$ in $O(kn \log n)$ time. □

We reduce our problem to minimum cost multi-commodity circulation problem in order to establish the existence of an interior point in the polytope, before invoking the RIPM in Theorem 2.24. For each commodity $i \in [k]$, we add extra vertices s_i and t_i. Let $\boldsymbol{d}_i$ be the demand vector of the i-th commodity. For every vertex v with $\boldsymbol{d}_{i,v} < 0$, we add a directed edge from s_i to v with capacity $-\boldsymbol{d}_{i,v}$ and cost 0. For every vertex v with $\boldsymbol{d}_{i,v} > 0$, we add a directed edge from v to t_i with capacity $\boldsymbol{d}_{i,v}$ and cost 0. Then, we add a directed edge from t_i to s_i with capacity $4kmM$ and cost $-4kmM$. The modified graph G' has only $2k$ extra vertices of the form s_i and t_i compared to $G_{\mathbf{A}}$, so we can construct a $(\frac{1}{2}, b, kn^{1/2}+2k)$-separator tree for G' based on the $(\frac{1}{2}, kn^{1/2})$-separator tree for $G_{\mathbf{A}}$, where we include the extra vertices at every node of the tree.

To show the existence of the interior point, we remove all the directed edges that no single commodity flow from s_i to t_i can pass for any $i \in [k]$. This can be done by running BFS k times, which takes $O(km)$ time. For the interior point $\boldsymbol{f}$, we construct this finding a circulation $\boldsymbol{f}^{(e)}$ that passing through e and s_i, t_i for some i with flow value $1/(10km)$ for all the remaining edge e. Then, since the capacities are integers, we find a feasible $\boldsymbol{f}, \boldsymbol{s}$ with value at least $1/(10km)$. This shows the inner radius r of the polytope is at least $1/(10km)$. For the L and R, we note we can bound it by $O(kmM)$.

Let $\mathbf{A}'$ be the constraint matrix of the reduced problem with dual graph G'. The RIPM in Theorem 2.24 invokes the subroutine PathFollowing twice. In the first run, we make a new constraint matrix by concatenating $\mathbf{A}'$ three times. One can check that the dual graph is G' with each edge duplicated three times, so the corresponding separator tree is straightforward to construct.

Now, we bounding the running time. The tree operator complexities are similar to the analysis in the previous section with an additional factor of k in the expression for λ. The initialization time is $O(km + (kn^{1/2})^{\omega})$. The query complexity is $Q(K) = O(K + k^2 n)$. After simplifying, the update complexity is

$$U(K) = K + \begin{cases} k^2 n K^{\omega-2} & \text{if } K \leq kn^{1/2} \\ (kn^{1/2})^{\omega} & \text{else.} \end{cases}$$

Note that the number of variables is km. Plugging our choice of L, R, and r, by Theorem 5.11, the total runtime simplifies to

$$\widetilde{O}\left(k^{2.5} m^{1.5} \log(M/\varepsilon)\right).$$

6.3 Low-treewidth linear programs

Theorem 6.4. *Suppose we have a linear program with the same setup as Theorem 6.1, and we are given a tree-decomposition of the dual graph $G_{\mathbf{A}}$[1] of width τ. Then we can solve the linear program in time*

$$\widetilde{O}(m\tau^2 \log(R/(\varepsilon r))) \text{ or } \widetilde{O}(m\tau^{(\omega+1)/2} \log(R/(\varepsilon r))).$$

[1]We can view the hypergraph $G_{\mathbf{A}}$ as a graph, where we interpret each hyper-edge as a clique, and consider its treewidth as usual.

Lemma 3.15 shows how to construct a separator tree for $G_{\mathbf{A}}$. Since the resulting tree is binary, so there are at most 2^i nodes at level i. Since there are $L = O(n/\tau)$-many leaves, the height η is at most $\eta \leq \log_2(n/\tau)$. The boundary of a node H is contained in the union of balanced separators over its ancestors, so $|F_H \cup \partial H| \leq \tau\eta \leq O(\tau \log n)$. The max hyperedge size of $G_{\mathbf{A}}$ is $\rho = \tau$.

Using these values, we simplify the complexities in Lemma 4.5: The initialization time for the tree operator data structure is $\widetilde{O}\left(\tau^{\omega-1}m + \tau^{\omega}\left(1 + n/\tau\right)\right) = \widetilde{O}(\tau^{\omega-1}m)$. The query complexity of $\mathbf{\Delta}$ is $Q(K) = \widetilde{O}\left(\tau K + \tau^2 \min\{K, L\}\right)$. The update complexity of $\mathbf{\Delta}$ is

$$U(K) \leq \tau^{\omega-1}K + \begin{cases} \tau^2 K & \text{if } K \leq n \\ \tau^{\omega} & \text{if } K > n \end{cases}$$

Finally, we apply Theorem 5.11 to get the overall runtime, which is clearly bounded by

$$\widetilde{O}\left(\sqrt{m}\log(\frac{R}{\varepsilon r})\right) \cdot \sum_{\ell=0}^{\frac{1}{2}\log m} \frac{\tau^2 2^{2\ell}}{2^{\ell}} = \widetilde{O}\left(m\tau^2 \log(R/(\varepsilon r))\right).$$

To obtain the faster runtime given in [80], we use the data structure restarting trick: Recall MaintainApprox guarantees there are $2^{2\ell}$-many coordinate updates to $\overline{\boldsymbol{x}}$ and $\overline{\boldsymbol{s}}$ every 2^{ℓ} steps, i.e. the number of coordinate updates grows superlinearly with respect to the total number of steps taken. By reinitializing MaintainApprox with the exact solution once in a while, we limit the total number of coordinate updates. In the proof of Theorem 5.11, we showed that running M steps of the RIPM takes

$$\widetilde{O}\left(U(m) + Q(m) + \eta^4 M \log(\frac{R}{\varepsilon r}) \cdot \sum_{\ell=0}^{\log M} \frac{U(2^{2\ell}) + Q(2^{2\ell})}{2^{\ell}}\right)$$

time, where $U(m) + Q(m)$ is the time to initialize the data structures and obtain the final exact solutions. There are $N = \sqrt{m}\log m \log(\frac{mR}{\varepsilon r})$-many total IPM steps, and we reinitialize the data structures every M steps. Then the total running time is (ignoring the big-O notation and log factors)

$$\begin{aligned} &\frac{N}{M}\left(U(m) + Q(m) + M\sum_{\ell=0}^{\log M} \frac{U(2^{2\ell}) + Q(2^{2\ell})}{2^{\ell}}\right) \\ &= \frac{\sqrt{m}}{M}\left(\tau^{\omega-1}m + \tau^2 M^2\right). \end{aligned}$$

The expression is minimized by taking $M = \sqrt{m}\tau^{\frac{\omega-3}{2}}$, which gives an overall runtime of

$$\widetilde{O}\left(m\tau^{(\omega+1)/2}\log(R/(\varepsilon r))\right).$$

□

6.4 Convex generalization

Since our robust IPM works for general convex sets, the results in this chapter naturally generalize to the convex optimization setting. For the sake of this discussion, consider it for the bounded-treewidth case; the corresponding theorem would be:

Theorem 6.5. *Given a convex program*

$$\min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b},\boldsymbol{x}_{[i]}\in K_i \text{ for } i\in[m]} \boldsymbol{c}^\top \boldsymbol{x} \tag{6.3}$$

where $A \in \mathbb{R}^{n\times d}$ is a full rank matrix with $n \le d$ and $K_i \subset \mathbb{R}^{d_i}$ are convex sets, with $\sum_{i=1}^{m} d_i = d$. We identify the columns of $\mathbf{A}$ in blocks, such that block i contains the d_i columns corresponding to $\boldsymbol{x}_i$. We define the generalized dual graph *$G_{\mathbf{A}}$ to be the graph with vertices set $\{1,\cdots d\}$, such that $ij \in E(G_{\mathbf{A}})$ if there is a block r such that $\mathbf{A}_{i,r} \neq \mathbf{0}$ and $\mathbf{A}_{j,r} \neq \mathbf{0}$. We define the product convex set $K = \Pi_{i=1}^{m} K_i$. Suppose that*

- *we are given a tree decomposition of $G_{\mathbf{A}}$ with width τ,*
- *R is the diameter of the set K,*
- *There exists $\boldsymbol{z}$ such that $\mathbf{A}\boldsymbol{z} = \boldsymbol{b}$ and $B(\boldsymbol{z}, r) \subset K$,*
- *$d_i = O(1)$ for all $i \in [m]$,*
- *we are given initial points $\boldsymbol{x}_i \in \mathbb{R}^{d_i}$ such that $B(\boldsymbol{x}_i, r) \subset K_i$ for each i,*
- *we can check if $\boldsymbol{y} \in K_i$ in $O(1)$ time for all $i \in [m]$.*

Then, for any $0 < \varepsilon \le 1/2$, we can find $\boldsymbol{x} \in K$ with $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ such that

$$\boldsymbol{c}^\top \boldsymbol{x} \le \min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b},\boldsymbol{x}\in K} \boldsymbol{c}^\top \boldsymbol{x} + \varepsilon \cdot \|\boldsymbol{c}\|_2 \cdot R$$

in expected time

$$\widetilde{O}(d \cdot \tau^2 \log(R/r) \log(R/(r\varepsilon))).$$

Remark 6.6. The proofs for the convex program and the linear program are almost identical. Any operation involving the entry $\mathbf{A}[i,j]$ in the linear program setting is generalized to operations on the $1 \times d_j$ submatrix of A from row i and block j. Since each block has size $O(1)$, the overall runtime relating to all matrix operations is maintained. We analyze our interior point method directly using this generalized formulation in this paper; the linear programming formulation follows as a special case.

This natural convex generalization in fact captures a large number of problem formulations. We illustrate with one example from signal processing, the fused lasso model for denoising [159]: Given a 1-D input signal $u_1, u_2, \cdots, u_n$, find an output $\boldsymbol{x}$ that minimizes the potential

$$V(\boldsymbol{x}) = \sum_{i=1}^{n} (\boldsymbol{x}_i - u_i)^2 + \lambda \sum_{i=1}^{n-1} |\boldsymbol{x}_{i+1} - \boldsymbol{x}_i|,$$

where the first term restricts the output signal to be close to the input, and the second term controls the amount of irregularity, and λ is the regularization parameter. To relate it back to our problem Eq. (6.3), we consider a generalized formulation: Given a family of convex functions $\phi_1, \ldots, \phi_N$ of $\boldsymbol{x} \stackrel{\text{def}}{=} (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d)$, where for each i, the function $\phi_i(\boldsymbol{x}) = \phi_i(\boldsymbol{x}_{S_i})$ only depends on the variables $\{\boldsymbol{x}_j \ : \ j \in S_i\}$ for some subset $S_i \subseteq [d]$, we want to solve the problem

$$\min_{x \in \mathbb{R}^d} \sum_{i=1}^{N} \phi_i(\boldsymbol{x}_{S_i}). \tag{6.4}$$

By creating extra variables $\boldsymbol{y}_{i,j}$ for all $i \in [d]$ and $j \in S_i$, we can write the problem as $\min \sum_i t_i$, subjected to $\boldsymbol{y}_{i,j} = \boldsymbol{x}_j$ and $t_i \geq \phi_i(\boldsymbol{y}_{i,j})$ for all i and all $j \in S_i$. The inequality constraints is equivalent to requiring that $(t, \boldsymbol{y})$ lie in the convex set $\{(t, \boldsymbol{y}) : t_i \geq \phi_i(\boldsymbol{y}_{i,j})\}$. This is exactly in the form of Eq. (6.3). The dual graph $G_{\mathbf{A}}$ of this problem is closely related to the intersection graph $G_{\mathcal{I}}$ of the set family $\{S_i\}_{i \in N}$: Specifically, each set of constraints $\boldsymbol{y}_{i,j} = \boldsymbol{x}_j$ corresponds to $|S_i|$ many vertices in $G_{\mathbf{A}}$, and contracting each such set into one vertex produces $G_{\mathcal{I}}$. Hence, we have that the treewidth $\operatorname{tw}(G_{\mathbf{A}})$ of this convex program is at most the treewidth of $G_{\mathcal{I}}$. For the denoising problem above, the intersection graph is in fact close to a path and has constant treewidth. Therefore, our result shows that this problem can be solved in nearly-linear time, without relying on the specific formula or structure.

Chapter 7

Min-cost flow

Given a graph $G = (V, E)$ on n vertices and m edges, with edge-vertex incidence matrix $\mathbf{B}$, vertex demands $\boldsymbol{d}$, edge costs $\boldsymbol{c}$, and edge capacity bounds $[\boldsymbol{l}, \boldsymbol{u}]$, the min-cost flow problem on G can be formulated as

$$\begin{aligned} \min\ & \boldsymbol{c}^\top \boldsymbol{f} \\ \text{s.t.}\quad & \mathbf{B}^\top \boldsymbol{f} = \boldsymbol{d} \\ & \boldsymbol{l} \le \boldsymbol{f} \le \boldsymbol{u}. \end{aligned} \tag{7.1}$$

Because the constraint matrix is the incidence matrix of a graph, we can apply additional techniques to speed up our data structures for LP solvers. In particular, we use the same framework as previous chapters consisting of the tree operator and Cholesky factorization, but in the recursive Schur complements, we make additional use of sparsification.

Before we proceed, we introduce the fast Laplacian solver as a black box.

Theorem 7.1 ([153])**.** *There is a randomized algorithm which is an ε-approximate SDD-system solver. Given a symmetrical diagonally-dominant (SDD) matrix $\mathbf{L} \in \mathbb{R}^{n\times n}$ with $O(m)$ non-zeros, $\boldsymbol{d} \in \mathbb{R}^n$, and $\varepsilon \in (0, 1)$, it finds $\boldsymbol{x}$ such that*

$$\left\|\boldsymbol{x} - \mathbf{L}^{-1}\boldsymbol{d}\right\|_{\mathbf{L}} \le \varepsilon \left\|\mathbf{L}^{-1}\boldsymbol{d}\right\|_{\mathbf{L}}$$

in $O(m \cdot \operatorname{poly}(\log\log n) \log(1/(\varepsilon\lambda_2(\mathbf{L})))$ time, where $\lambda_2(\mathbf{L})$ is the second smallest eigenvalue of $\mathbf{L}$. Moreover, the solver guarantees that $\boldsymbol{x} = \mathbf{Z}\boldsymbol{d}$, where $\mathbf{Z}$ is an $n \times n$-matrix depending only on $\mathbf{L}$ and ε, is a symmetric linear operator satisfying $\mathbf{Z} \approx_\varepsilon \mathbf{L}^{-1}$, and has the same image as $\mathbf{L}^{-1}$ [168, Section 3.4 and Theorem 9.2].

7.1 Problem reduction

In this section, we show how to write down the planar min-cost flow problem as a linear program of the form

$$(\text{Primal}) = \min_{\mathbf{B}^\top \boldsymbol{f} = \mathbf{0},\, \boldsymbol{l} \le \boldsymbol{f} \le \boldsymbol{u}} \boldsymbol{c}^\top \boldsymbol{f} \quad \text{and} \quad (\text{Dual}) = \min_{\mathbf{B}\boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c}} \sum_i \min(\boldsymbol{l}_i \boldsymbol{s}_i, \boldsymbol{u}_i \boldsymbol{s}_i),$$

where $\mathbf{B} \in \mathbb{R}^{m \times n}$ is an edge-vertex incidence matrix of the graph, $\boldsymbol{f}$ is the flow and $\boldsymbol{s}$ is the slack (or adjusted cost vector). We will run PathFollowingLP on the above formulation. From the returned solution, we postprocess to produce an optimal flow.

First, we add extra vertices s and t to the input graph. For every vertex v with $\boldsymbol{d}_v < 0$, we add a directed edge from s to v with capacity $-\boldsymbol{d}_v$ and cost 0. For every vertex v with $\boldsymbol{d}_v > 0$, we add a directed edge from v to t with capacity $\boldsymbol{d}_v$ and cost 0. Then, we add a directed edge from t to s with capacity $4nM$ and cost $-4nM$. The cost and capacity on the $t \to s$ edge are chosen such that the min-cost flow problem on the original graph is equivalent to the min-cost circulation on this new graph. Namely, if the min-cost circulation in this new graph satisfies all the demand $\boldsymbol{d}_v$, then by ignoring the flow on the new edges we obtain the min-cost flow in the original graph.

Since the IPM requires an interior point in the polytope, we first remove all directed edges e through which no flow from s to t can pass. To do this, we simply check, for every directed edge $e = (v_1, v_2)$, if s can reach v_1 and if v_2 can reach t. This can be done in $O(m)$ time by a BFS from s and a reverse BFS from t. With this preprocessing, we write the minimum cost circulation problem as the following linear program

$$\min_{\mathbf{B}^\top \boldsymbol{f} = \mathbf{0},\, \boldsymbol{l}^{\text{new}} \le \boldsymbol{f} \le \boldsymbol{u}^{\text{new}}} (\boldsymbol{c}^{\text{new}})^\top \boldsymbol{f}$$

where $\mathbf{B}$ is the signed incidence matrix of the new graph, $\boldsymbol{c}^{\text{new}}$ is the new cost vector, and $\boldsymbol{l}^{\text{new}}, \boldsymbol{u}^{\text{new}}$ are the new capacity constraints. This LP is the input to the interior point method; we call the new graph the IPM input graph.

Now, we bound the parameters L, R, r in Theorem 2.24. Clearly, $L = \|\boldsymbol{c}^{\text{new}}\|_2 = O(Mm)$ and $R = \|\boldsymbol{u}^{\text{new}} - \boldsymbol{l}^{\text{new}}\|_2 = O(Mm)$. To bound r, we prove that there exists an interior point $\boldsymbol{f}$ in the polytope $\mathcal{F}$. We construct this $\boldsymbol{f}$ by $\boldsymbol{f} = \sum_{e \in E} \boldsymbol{f}^{(e)}$, where $\boldsymbol{f}^{(e)}$ is a circulation passing through edges e and (t, s) with flow value $1/(4m)$. All such circulations exist because of the removal preprocessing. This $\boldsymbol{f}$ satisfies the capacity constraints because all capacities are at least 1. This shows $r \ge \frac{1}{4m}$.

Let OPT denote the optimal objective value of the original min-cost flow with flow value F. Let OPT$'$ denote the optimal objective value of the min-cost circulation which also has flow value F. We know $\text{OPT}' = \text{OPT} - 4FnM$. Theorem 2.24 shows that we can find a fractional circulation $\boldsymbol{f}'$ with flow value F' such that $(\boldsymbol{c}^{\text{new}})^\top \boldsymbol{f}' \le \text{OPT}' + \frac{1}{2}$, by setting $\epsilon = \frac{1}{CM^2m^2}$ for some large constant C. We know F' is smaller than$F$, but $F - F' \le \frac{1}{2nM}$, because otherwise sending extra k units of fractional flow from s to t would give extra negative cost $\le -knM$, leading to an objective value smaller than OPT$'$. Let $\boldsymbol{f}$ denote $\boldsymbol{f}'$ restricted to the original planar graph (still with flow value F'), then we can round $\boldsymbol{f}$ to an integral flow $\boldsymbol{f}^{\text{int}}$ with same or better flow value and no worst cost using $\widetilde{O}(m)$ time [96]. Since $\boldsymbol{f}^{\text{int}}$ is integral with flow value $\geq F - \frac{1}{2nM}$, we conclude it routes the original demand completely. Moreover,

$$\boldsymbol{c}^\top \boldsymbol{f}^{\text{int}} \leq \boldsymbol{c}^\top \boldsymbol{f} \leq \text{OPT}' + \frac{1}{2} + 4F'nM = \text{OPT} + \frac{1}{2} + 4nM(F' - F) \leq \text{OPT} + \frac{1}{2},$$

so $\boldsymbol{f}^{\text{int}}$ must have the minimum cost.

7.2 Tree operator for slack

To use our existing IPM data structure framework, we must define the tree operators for both the slack and flow projections. Because we use approximate projection matrices, the proof of feasibility becomes significantly more involved compared to the exact computations used in general LPs.

The full slack update at IPM step k with step direction $\boldsymbol{v}^{(k)}$ and step size $\bar{t}h$ is

$$\boldsymbol{s} \leftarrow \boldsymbol{s} + \mathbf{W}^{-1/2}\widetilde{\mathbf{P}}_{\boldsymbol{w}}(\bar{t}h\boldsymbol{v}^{(k)}),$$

where we require $\widetilde{\mathbf{P}}_{\boldsymbol{w}} \approx \mathbf{P}_{\boldsymbol{w}}$ and $\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}^{(k)} \in \text{Range}(\mathbf{W}^{1/2}\mathbf{B})$.

Let $\widetilde{\mathbf{L}}^{-1}$ denote the approximation of $\mathbf{L}^{-1}$ from Eq. (3.9), maintained and computable with a DynamicSC data structure. If we define

$$\widetilde{\mathbf{P}}_{\boldsymbol{w}} = \mathbf{W}^{1/2}\mathbf{B}\widetilde{\mathbf{L}}^{-1}\mathbf{B}^\top\mathbf{W}^{1/2} = \mathbf{W}^{1/2}\mathbf{B}\mathbf{\Pi}^{(0)\top}\cdots\mathbf{\Pi}^{(\eta-1)\top}\widetilde{\mathbf{\Gamma}}\mathbf{\Pi}^{(\eta-1)}\cdots\mathbf{\Pi}^{(0)}\mathbf{B}^\top\mathbf{W}^{1/2}.$$

then $\widetilde{\mathbf{P}}_{\boldsymbol{w}} \approx_{\eta\epsilon_{\mathbf{P}}} \mathbf{P}_{\boldsymbol{w}}$, and $\text{Range}(\widetilde{\mathbf{P}}_{\boldsymbol{w}}) = \text{Range}(\mathbf{P}_{\boldsymbol{w}})$ by definition, where η and $\epsilon_{\mathbf{P}}$ are parameters in DynamicSC. Hence, this suffices as our approximate slack projection matrix. In order to use MaintainRep to maintain $\boldsymbol{s}$ throughout the IPM, it remains to define a slack tree operator $\mathbf{M}^{(\text{slack})}$ so that

$$\mathbf{W}^{-1/2}\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}^{(k)} = \mathbf{M}^{(\text{slack})}\boldsymbol{z}^{(k)},$$

where $\boldsymbol{z}^{(k)} \stackrel{\text{def}}{=} \widetilde{\mathbf{\Gamma}}\mathbf{\Pi}^{(\eta-1)}\cdots\mathbf{\Pi}^{(0)}\mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}^{(k)}$ at IPM step k. We proceed by defining a tree operator $\mathbf{M}$ satisfying $\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}^{(k)} = \mathbf{M}\boldsymbol{z}^{(k)}$. Namely, we show that $\mathbf{M} \stackrel{\text{def}}{=} \mathbf{W}^{1/2}\mathbf{B}\mathbf{\Pi}^{(0)\top}\cdots\mathbf{\Pi}^{(\eta-1)\top}$ is indeed a tree operator. Then we set $\mathbf{M}^{(\text{slack})} \stackrel{\text{def}}{=} \mathbf{W}^{-1/2}\mathbf{M}$.

For the remainder of the section, we abuse notation and use $\boldsymbol{z}$ to mean $\boldsymbol{z}^{(k)}$ for one IPM step k.

Definition 7.2 (Slack projection tree operator)**.** Let $\mathcal{T}$ be the separator tree from data structure DynamicSC, with Laplacians $\mathbf{L}^{(H)}$ and $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ at each node $H \in \mathcal{T}$. We use $\mathbf{B}[H]$ to denote the adjacency matrix of G restricted to the region.

For a node $H \in \mathcal{T}$, define $V(H)$ and F_H required by the tree operator as $\partial H \cup F_H$ and F_H from the separator tree construction respectively. Note the slightly confusing fact that $V(H)$ *is not* the set of vertices in region H of the input graph G, *unless* H is a leaf node. Suppose node H has parent P, then define the tree edge operator $\mathbf{M}_{(H,P)} : \mathbb{R}^{V(P)} \mapsto \mathbb{R}^{V(H)}$ as:

$$\mathbf{M}_{(H,P)} \stackrel{\text{def}}{=} \mathbf{I}_{\partial H \cup F_H} - \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1} \mathbf{L}^{(H)}_{F_H,\partial H} \stackrel{\text{def}}{=} \mathbf{I}_{\partial H \cup F_H} - \mathbf{X}^{(H)\top}. \tag{7.2}$$

At each leaf node H of $\mathcal{T}$, define the leaf operator $\mathbf{J}_H = \mathbf{W}^{1/2}\mathbf{B}[H]$.

The remainder of this section proves the correctness of the tree operator.

Lemma 7.3. *Let* $\mathbf{M}$ *be the tree operator as defined in Definition 7.2. We have*

$$\mathbf{M}\boldsymbol{z} = \mathbf{W}^{1/2}\mathbf{B}\mathbf{\Pi}^{(0)\top} \cdots \mathbf{\Pi}^{(\eta-1)\top}\boldsymbol{z}.$$

We begin with a few observations about the $\mathbf{\Pi}^{(i)}$'s:

Observation 7.4. *For any* $0 \leq i < \eta$, *and for any vector* $\boldsymbol{x}$, *we have* $\mathbf{\Pi}^{(i)\top}\boldsymbol{x} = \boldsymbol{x} + \boldsymbol{y}_i$, *where* $\boldsymbol{y}_i$ *is a vector supported on* $F_i = \cup_{H \in \mathcal{T}(i)} F_H$. *Extending this observation, for* $0 \leq i < j < \eta$,

$$\mathbf{\Pi}^{(i)\top} \cdots \mathbf{\Pi}^{(j-1)\top}\boldsymbol{x} = \boldsymbol{x} + \boldsymbol{y},$$

where $\boldsymbol{y}$ *is a vector supported on* $F_i \cup \cdots \cup F_{j-1} = \cup_{H: i \leq \eta(H) < j} F_H$. *Furthermore, if* $\boldsymbol{x}$ *is supported on* F_A *for* $\eta(A) = j$, *then* $\boldsymbol{y}$ *is supported on* $\cup_{H \in \mathcal{T}_A} F_H$.

The following helper lemma describes a sequence of edge operators from a node to a leaf.

Lemma 7.5. *For any leaf node* $H \in \mathcal{T}$, *and a node* A *with* $H \in \mathcal{T}_A$ *(*A *is an ancestor of* H *or* H *itself), we have*

$$\mathbf{M}_{H \leftarrow A}\boldsymbol{z}|_{F_A} = \mathbf{I}_{\partial H \cup F_H}\mathbf{\Pi}^{(0)\top} \cdots \mathbf{\Pi}^{(\eta-1)\top}\boldsymbol{z}|_{F_A}. \tag{7.3}$$

Proof. For simplicity of notation, let $V(H) \stackrel{\text{def}}{=} \partial H \cup F_H$ for a node H.

To start, observe that for a node A at level $\eta(A)$, we have $\mathbf{\Pi}^{(i)}\boldsymbol{z}|_{F_A} = \boldsymbol{z}|_{F_A}$ for all $i \geq \eta(A)$. So it suffices to prove

$$\mathbf{M}_{H \leftarrow A}\boldsymbol{z}|_{F_A} = \mathbf{I}_{V(H)}\mathbf{\Pi}^{(0)\top} \cdots \mathbf{\Pi}^{(\eta(A)-1)\top}\boldsymbol{z}|_{F_A}.$$

Let the path from leaf H up to node A in $\mathcal{T}$ be denoted $(H_0 \stackrel{\text{def}}{=} H, H_1, \ldots, H_t \stackrel{\text{def}}{=} A)$, for some $t \leq \eta(A)$. We will prove by induction for k decreasing from t to 0:

$$\mathbf{M}_{H_k \leftarrow A}\boldsymbol{z}|_{F_A} = \mathbf{I}_{V(H_k)}\mathbf{\Pi}^{(\eta(H_k))\top}\mathbf{\Pi}^{(\eta(H_k)+1)\top} \cdots \mathbf{\Pi}^{(\eta(A)-1)\top}\boldsymbol{z}|_{F_A}. \tag{7.4}$$

For the base case of $H_t = A$, we have $\mathbf{M}_{H_t \leftarrow A}\boldsymbol{z}|_{F_A} = \boldsymbol{z}|_{F_A} = \mathbf{I}_{V(H_t)}\boldsymbol{z}|_{F_A}$.

For the inductive step at H_k, we first apply induction hypothesis for H_{k+1} to get

$$\mathbf{M}_{H_{k+1} \leftarrow A}\boldsymbol{z}|_{F_A} = \mathbf{I}_{V(H_{k+1})}\mathbf{\Pi}^{(\eta(H_{k+1}))\top} \cdots \mathbf{\Pi}^{(\eta(A)-1)\top}\boldsymbol{z}|_{F_A}. \tag{7.5}$$

Multiplying by the edge operator $\mathbf{M}_{(H_k,H_{k+1})}$ on both sides gives

$$\mathbf{M}_{H_k \leftarrow A}\boldsymbol{z}|_{F_A} = \mathbf{M}_{(H_k,H_{k+1})}\mathbf{I}_{V(H_{k+1})}\mathbf{\Pi}^{(\eta(H_{k+1}))\top} \cdots \mathbf{\Pi}^{(\eta(A)-1)\top}\boldsymbol{z}|_{F_A}. \tag{7.6}$$

Recall the edge operator $\mathbf{M}_{(H_k,H_{k+1})}$ maps vectors supported on $V(H_{k+1})$ to vectors supported on $V(H_k)$ and zeros otherwise. So we can drop the $\mathbf{I}_{V(H_{k+1})}$ term in the right hand side. Let $\boldsymbol{x} \stackrel{\text{def}}{=} \mathbf{\Pi}^{(\eta(H_{k+1}))\top} \cdots \mathbf{\Pi}^{(\eta(A)-1)\top}\boldsymbol{z}|_{F_A}$. Now, by the definition of the edge operator, the above equation becomes

$$\mathbf{M}_{H_k \leftarrow A}\boldsymbol{z}|_{F_A} = (\mathbf{I}_{V(H_k)} - \mathbf{X}^{(H_k)\top})\boldsymbol{x}. \tag{7.7}$$

On the other hand, we have

$$\begin{aligned}\mathbf{I}_{V(H_k)}\mathbf{\Pi}^{(\eta(H_k))\top} \cdots \mathbf{\Pi}^{(\eta(H_{k+1})-1)\top}\boldsymbol{x} &= \mathbf{I}_{V(H_k)}\mathbf{\Pi}^{(\eta(H_k))\top}\left(\mathbf{\Pi}^{(\eta(H_k)+1)\top} \cdots \mathbf{\Pi}^{(\eta(H_{k+1})-1)\top}\boldsymbol{x}\right)\\ &= \mathbf{I}_{V(H_k)}\mathbf{\Pi}^{(\eta(H_k))\top}(\boldsymbol{x}+\boldsymbol{y}),\end{aligned}$$

where $\boldsymbol{y}$ is a vector supported on $\cup F_R$ for nodes R at levels $\eta(H_k)+1, \cdots, \eta(H_{k+1})-1$ by Observation 7.4. In particular, $\boldsymbol{y}$ is zero on F_{H_k}. Also, $\boldsymbol{y}$ is zero on ∂H_k, since $\partial H_k \subseteq \cup_{\text{ancestor } A' \text{ of } H_k} F_{A'}$, and ancestors of H_k are at level $\eta(H_{k+1})$ or higher. Then $\boldsymbol{y}$ is zero on $V(H_k) = \partial H_k \cup F_{H_k}$, and the right hand side is

$$= (\mathbf{I}_{V(H_k)} - \mathbf{X}^{(H_k)\top})\boldsymbol{x},$$

where we apply the definition of $\mathbf{\Pi}^{(\eta(H_k))\top}$ and expand the left-multiplication by $\mathbf{I}_{V(H_k)}$.

Combining with Eq. (7.7) and substituting back the definition of $\boldsymbol{x}$, we get

$$\mathbf{M}_{H_k \leftarrow A}\boldsymbol{z}|_{F_A} = \mathbf{I}_{V(H_k)}\mathbf{\Pi}^{(\eta(H_k))\top} \cdots \mathbf{\Pi}^{(\eta(A)-1)\top}\boldsymbol{z}|_{F_A}.$$

which completes the induction.

□

To prove Lemma 7.3, we apply the leaf operators to the result of the previous lemma and sum over all nodes and leaf nodes.

Proof of Lemma 7.3. Let H be a leaf node. We sum Eq. (7.3) over all A with $H \in \mathcal{T}_A$ to get

$$\begin{aligned}\sum_{A:H\in\mathcal{T}_A} \mathbf{M}_{H\leftarrow A}\boldsymbol{z}|_{F_A} &= \mathbf{I}_{\partial H\cup F_H} \sum_{A:H\in\mathcal{T}_A} \mathbf{\Pi}^{(0)\top}\cdots\mathbf{\Pi}^{(\eta-1)\top}\boldsymbol{z}|_{F_A}\\ &= \mathbf{I}_{\partial H\cup F_H}\mathbf{\Pi}^{(0)\top}\cdots\mathbf{\Pi}^{(\eta-1)\top}\boldsymbol{z},\end{aligned}$$

where we relax the sum in the right hand side to be over all nodes in $\mathcal{T}$, since by Observation 7.4, for any A with $H \notin \mathcal{T}_A$, we simply have $\mathbf{I}_{\partial H\cup F_H}\mathbf{\Pi}^{(0)\top}\cdots\mathbf{\Pi}^{(\eta-1)\top}\boldsymbol{z}|_{F_A} = \mathbf{0}$. Next, we apply the leaf operator $\mathbf{J}_H = \mathbf{W}^{1/2}\mathbf{B}[H]$ to both sides to get

$$\sum_{A:H\in\mathcal{T}_A} \mathbf{J}_H\mathbf{M}_{H\leftarrow A}\boldsymbol{z}|_{F_A} = \mathbf{W}^{1/2}\mathbf{B}[H]\mathbf{I}_{\partial H\cup F_H}\mathbf{\Pi}^{(0)\top}\cdots\mathbf{\Pi}^{(\eta-1)\top}\boldsymbol{z}.$$

Since $\mathbf{B}[H]$ is zero on columns supported on $V(G)\setminus(\partial H\cup F_H)$, we can simply drop the $\mathbf{I}_{\partial H\cup F_H}$ in the right hand side.

Finally, we sum up the equation above over all leaf nodes. The left hand side is precisely the definition of $\mathbf{M}\boldsymbol{z}$. Recall the regions of the leaf nodes partition the original graph G, so we have

$$\begin{aligned}\sum_{H\in\mathcal{T}(0)}\sum_{A:H\in\mathcal{T}_A} \mathbf{J}_H\mathbf{M}_{H\leftarrow A}\boldsymbol{z}|_{F_A} &= \mathbf{W}^{1/2}\left(\sum_{H\in\mathcal{T}(0)}\mathbf{B}[H]\right)\mathbf{\Pi}^{(0)\top}\cdots\mathbf{\Pi}^{(\eta-1)\top}\boldsymbol{z}\\ \mathbf{M}\boldsymbol{z} &= \mathbf{W}^{1/2}\mathbf{B}\mathbf{\Pi}^{(0)\top}\cdots\mathbf{\Pi}^{(\eta-1)\top}\boldsymbol{z}.\end{aligned}$$

□

The tree operator $\mathbf{M}$ defined in Definition 7.2 satisfies $\mathbf{M}\boldsymbol{z}^{(k)} = \widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}^{(k)}$ at step k, by the definition of $\boldsymbol{z}^{(k)}$. To support the proper update $\boldsymbol{s} \leftarrow \boldsymbol{s} + \bar{t}h\mathbf{W}^{-1/2}\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}^{(k)}$, we define $\mathbf{M}^{(\text{slack})} \stackrel{\text{def}}{=} \mathbf{W}^{-1/2}\mathbf{M}$ and note it is clearly also a tree operator with the same complexity.

We now examine the slack tree operator complexity.

Lemma 7.6. *The query and update complexities of the slack tree operator as defined in Definition 7.7 are* $Q(K) = O\left({\epsilon_{\mathbf{P}}}^2 \max_{\mathcal{H}:\text{set of } K \text{ nodes in } \mathcal{S}} \sum_{H\in\mathcal{H}} |F_H\cup\partial H|\right)$, *where* $\epsilon_{\mathbf{P}}$ *is the Schur complement approximation factor from data structure* DynamicSC.

Proof. Let $\mathbf{M}_{(D,P)}$ be a tree edge operator. Applying $\mathbf{M}_{(D,P)} = \mathbf{I}_{\partial D} - \left(\mathbf{L}^{(D)}_{F_D,F_D}\right)^{-1}\mathbf{L}^{(D)}_{F_D,\partial D}$ to the left or right consists of three steps which are applying $\mathbf{I}_{\partial D}$, applying $\mathbf{L}^{(D)}_{F_D,\partial D}$ and solving for $\mathbf{L}^{(D)}_{F_D,F_D}\boldsymbol{v} = \boldsymbol{b}$ for some vectors $\boldsymbol{v}$ and $\boldsymbol{b}$. Each of the three steps costs time $O({\epsilon_{\mathbf{P}}}^{-2}|F_D\cup\partial D|)$ by Lemma 4.8 and Theorem 7.1.

For any leaf node H, H has a constant number of edges, and it takes constant time to compute $\mathbf{J}_H \boldsymbol{u}$ for any vector $\boldsymbol{u}$. The number of vertices may be larger but the nonzeros of $\mathbf{J}_H = \mathbf{W}^{1/2}\mathbf{B}[H]$ only depends on the number of edges. $\square$

7.3 Tree operator for flow

We hope to use MAINTAINREP for $\boldsymbol{f}^{\perp}$ throughout the IPM. In order to do so, it remains to define a flow tree operator $\mathbf{M}^{(\text{flow})}$ so that at step k,

$$\mathbf{M}^{(\text{flow})}\boldsymbol{z}^{(k)} = \mathbf{W}^{1/2}\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v}^{(k)},$$

where $\boldsymbol{z}^{(k)} \stackrel{\text{def}}{=} \widetilde{\mathbf{\Gamma}}\mathbf{\Pi}^{(\eta-1)}\cdots\mathbf{\Pi}^{(0)}\mathbf{B}^{\top}\mathbf{W}^{1/2}\boldsymbol{v}^{(k)}$. We will define a tree operator $\mathbf{M}$ so that $\tilde{\boldsymbol{f}} \stackrel{\text{def}}{=} \mathbf{M}\boldsymbol{v}^{(k)}$ satisfies $\left\|\tilde{\boldsymbol{f}} - \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}^{(k)}\right\|_2 \leq O(\eta\epsilon_{\mathbf{P}})\left\|\boldsymbol{v}^{(k)}\right\|_2$ and $\mathbf{B}^{\top}\mathbf{W}^{1/2}\tilde{\boldsymbol{f}} = \mathbf{B}^{\top}\mathbf{W}^{1/2}\boldsymbol{v}^{(k)}$. This means it is feasible to set $\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v}^{(k)} = \tilde{\boldsymbol{f}}$. Then, we define $\mathbf{M}^{(\text{flow})} \stackrel{\text{def}}{=} \mathbf{W}^{-1/2}\mathbf{M}$.

For the remainder of the section, we assume the IPM step is fixed and omit all superscripts in our notation.

Definition 7.7 (Flow projection tree operator)**.** Let $\mathcal{T}$ be the separator tree from data structure DYNAMICSC, with Laplacians $\mathbf{L}^{(H)}$ and $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ at each node $H \in \mathcal{T}$. We use $\mathbf{B}[H]$ to denote the adjacency matrix of G restricted to the region.

To define the flow projection tree operator $\mathbf{M}$, we proceed as follows: The tree operator is supported on the tree $\mathcal{T}$. For a node $H \in \mathcal{T}$ with parent P, define the tree edge operator $\mathbf{M}_{(H,P)}$ as:

$$\mathbf{M}_{(H,P)} \stackrel{\text{def}}{=} (\mathbf{L}^{(H)})^{-1}\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H). \tag{7.8}$$

At each node H, we let F_H in the tree operator be the set F_H of eliminated vertices defined in the separator tree. At each leaf node H of $\mathcal{T}$, we have the leaf operator $\mathbf{J}_H = \mathbf{W}^{1/2}\mathbf{B}[H]$.

The complexity of this operator is clearly the same as the slack.

The remainder of the section is dedicated to proving the following theorem of correctness:

Theorem 7.8. *Let $\boldsymbol{v} \in \mathbb{R}^m$, and $\boldsymbol{z} \stackrel{\text{def}}{=} \widetilde{\mathbf{\Gamma}}\mathbf{\Pi}^{(\eta-1)}\cdots\mathbf{\Pi}^{(0)}\mathbf{B}^{\top}\mathbf{W}^{1/2}\boldsymbol{v}$. Let $\mathbf{M}$ be the flow projection tree operator from Definition 7.7, and let $\epsilon_{\mathbf{P}}$ be the overall target step accuracy from* DYNAMICSC*. Then $\tilde{\boldsymbol{f}} \stackrel{\text{def}}{=} \mathbf{M}\boldsymbol{z}$ satisfies $\mathbf{B}^{\top}\mathbf{W}^{1/2}\tilde{\boldsymbol{f}} = \mathbf{B}^{\top}\mathbf{W}^{1/2}\boldsymbol{v}$ and $\left\|\tilde{\boldsymbol{f}} - \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}\right\|_2 \leq O(\eta\epsilon_{\mathbf{P}})\left\|\boldsymbol{v}\right\|_2$.*

We first recall some terminology: A vector $\boldsymbol{d}$ is a *demand vector* if $\sum_i \boldsymbol{d}_i = 0$. If $\mathbf{B}$ is the edge-vertex incidence matrix of a graph, then $\mathbf{B}^{\top}\boldsymbol{x}$ is a demand for any $\boldsymbol{x}$. Similarly, $\mathbf{B}^{\top}\mathbf{W}\mathbf{B}\boldsymbol{x} = \mathbf{L}\boldsymbol{x}$ is a demand vector. We say a flow $\boldsymbol{f}$ routes a demand $\boldsymbol{d}$ if $(\mathbf{W}^{1/2}\mathbf{B})^{\top}\boldsymbol{f} = \boldsymbol{d}$.

Electrical flow lemmas

Here we introduce some definitions and properties of electrical flow, which we use in later to prove Theorem 7.8. In this setting, the graph is viewed as an electrical circuit with each edge e being a wire with resistance $\mathbf{W}_e^{-1/2}$.

Definition 7.9. Let $\mathbf{W}^{1/2}\mathbf{B} \in \mathbb{R}^{m\times n}$ be the edge-vertex weighted incidence matrix of some graph G, and let $\mathbf{L} \stackrel{\text{def}}{=} \mathbf{B}^\top\mathbf{W}\mathbf{B}$ be the Laplacian. Let $\boldsymbol{d} \stackrel{\text{def}}{=} \mathbf{L}\boldsymbol{z}$ be a demand vector and $\boldsymbol{f}$ be any flow that routes $\boldsymbol{d}$; that is, $(\mathbf{W}^{1/2}\mathbf{B})^\top\boldsymbol{f} = \boldsymbol{d}$. Then we say $\|\boldsymbol{f}\|_2^2$ is the *energy* of the flow $\boldsymbol{f}$.

There is a unique energy-minimizing flow $\boldsymbol{f}^\star$ routing the demand $\boldsymbol{d}$ on G. From the study of electrical flows, we know $\boldsymbol{f}^\star = \mathbf{W}^{1/2}\mathbf{B}\mathbf{L}^{-1}\boldsymbol{d}$. Hence, we can refer to its energy as the energy of the demand $\boldsymbol{d}$ on the graph of $\mathbf{L}$, given by

$$\mathcal{E}_{\mathbf{L}}(\boldsymbol{d}) \stackrel{\text{def}}{=} \min_{(\mathbf{W}^{1/2}\mathbf{B})^\top\boldsymbol{f}=\boldsymbol{d}} \|\boldsymbol{f}\|_2^2 = \boldsymbol{d}^\top(\mathbf{B}^\top\mathbf{W}\mathbf{B})^{-1}\boldsymbol{d} = \boldsymbol{d}^\top\mathbf{L}^{-1}\boldsymbol{d} = \boldsymbol{z}^\top\mathbf{L}\boldsymbol{z}. \tag{7.9}$$

If another flow $\tilde{\boldsymbol{f}}$ routing $\boldsymbol{d}$ is approximately energy-minimizing, then $\tilde{\boldsymbol{f}}$ must be close to $\boldsymbol{f}^\star$:

Lemma 7.10. *We continue using the notation from Definition 7.9. For any flow $\tilde{\boldsymbol{f}}$ routing $\boldsymbol{d}$ on G, if $\|\tilde{\boldsymbol{f}}\|_2^2 \le_\varepsilon \mathcal{E}_{\mathbf{L}}(\boldsymbol{d})$, then $\|\tilde{\boldsymbol{f}} - \boldsymbol{f}^\star\|_2^2 \le O(\varepsilon)\,\|\boldsymbol{f}^\star\|_2^2$.*

Proof. If a flow $\tilde{\boldsymbol{f}}$ routes $\boldsymbol{d}$ on G, then $(\mathbf{W}^{1/2}\mathbf{B})^\top\tilde{\boldsymbol{f}} = \boldsymbol{d}$. So we have

$$\boldsymbol{f}^{\star\top}(\tilde{\boldsymbol{f}} - \boldsymbol{f}^\star) = \boldsymbol{d}^\top\mathbf{L}^{-1}\mathbf{B}^\top\mathbf{W}^{1/2}(\tilde{\boldsymbol{f}} - \boldsymbol{f}^\star) = \boldsymbol{d}^\top\mathbf{L}^{-1}(\boldsymbol{d} - \boldsymbol{d}) = \mathbf{0}.$$

Hence, we have an orthogonal decomposition $\|\tilde{\boldsymbol{f}}\|_2^2 = \|\tilde{\boldsymbol{f}}^\star\|_2^2 + \|\tilde{\boldsymbol{f}} - \boldsymbol{f}^\star\|_2^2$. It follows that

$$\|\tilde{\boldsymbol{f}} - \boldsymbol{f}^\star\|^2 \le (e^\varepsilon - 1)\cdot\|\boldsymbol{f}^\star\|_2^2 = O(\varepsilon)\cdot\|\boldsymbol{f}^\star\|_2^2\,.$$

□

We want to understanding how the energy changes when, instead of routing $\boldsymbol{d}$ using the edges of G, we use edges of some other graphs related to G. In particular, we are interested in the operations of graph decompositions and taking Schur complements. It turns out the energy behaves nicely:

Lemma 7.11. *Suppose G is a weighted graph that can be decomposed into weighted subgraphs G_1, G_2. That is, if $\mathbf{L}$ is the Laplacian of G, and $\mathbf{L}^{(i)}$ is the Laplacian of G_i, then $\mathbf{L} = \mathbf{L}^{(1)} + \mathbf{L}^{(2)}$. Suppose $\boldsymbol{d} \stackrel{\text{def}}{=} \mathbf{L}\boldsymbol{z}$ is a demand on the vertices of G. Then if we decompose $\boldsymbol{d} = \boldsymbol{d}^{(1)} + \boldsymbol{d}^{(2)}$, where $\boldsymbol{d}^{(i)} = \mathbf{L}^{(i)}\boldsymbol{z}$, then the energies are related by:*

$$\mathcal{E}_{\mathbf{L}}(\boldsymbol{d}) = \mathcal{E}_{\mathbf{L}^{(1)}}\left(\boldsymbol{d}^{(1)}\right) + \mathcal{E}_{\mathbf{L}^{(2)}}\left(\boldsymbol{d}^{(2)}\right).$$

Proof. We have, by definition,

$$\begin{aligned}\mathcal{E}_{\mathbf{L}^{(1)}}\left(\boldsymbol{d}^{(1)}\right)+\mathcal{E}_{\mathbf{L}^{(2)}}\left(\boldsymbol{d}^{(2)}\right) &= \boldsymbol{z}^{\top}\mathbf{L}^{(1)}\boldsymbol{z}+\boldsymbol{z}^{\top}\mathbf{L}^{(2)}\boldsymbol{z}\\ &= \boldsymbol{z}^{\top}\mathbf{L}\boldsymbol{z}\\ &= \mathcal{E}_{\mathbf{L}}(\boldsymbol{d}).\end{aligned}$$

□

The following lemma shows if G' is a graph derived from G by taking Schur complement on a subset of the vertices C, and $\boldsymbol{d}$ is a demand supported on C, then the flow routing $\boldsymbol{d}$ on G will have lower energy than the flow routing $\boldsymbol{d}$ on G'.

Lemma 7.12. *Suppose G is a weighted graph with Laplacian $\mathbf{L}$. Let C be a subset of vertices of G. Let $\mathbf{L}' \overset{\text{def}}{=} \widetilde{\mathbf{Sc}}(\mathbf{L}, C) \approx_{\varepsilon} \mathbf{Sc}(\mathbf{L}, C)$ be an ε-approximate Schur complement. Then for any demand $\boldsymbol{d}$ supported on C,*

$$\mathcal{E}_{\mathbf{L}}\left(\boldsymbol{d}\right) \approx_{\varepsilon} \mathcal{E}_{\mathbf{L}'}\left(\boldsymbol{d}\right).$$

Proof. We have, by definition,

$$\mathcal{E}_{\mathbf{L}}\left(\boldsymbol{d}\right)=\boldsymbol{d}^{\top}\mathbf{L}^{-1}\boldsymbol{d}=\boldsymbol{d}^{\top}\mathbf{Sc}(\mathbf{L},C)^{-1}\boldsymbol{d}\approx_{\varepsilon}\boldsymbol{d}^{\top}\mathbf{L}'^{-1}\boldsymbol{d}=\mathcal{E}_{\mathbf{L}'}(\boldsymbol{d}),$$

where the second equality follows from the fact that $\boldsymbol{d}$ is supported on C combined with the formula for $\mathbf{L}^{-1}$. □

7.3.1 Proof of Theorem 7.8

Let G denote the input graph with weights $\mathbf{W}$ and Laplacian $\mathbf{L}$. Let $\boldsymbol{d} \overset{\text{def}}{=} \mathbf{B}^{\top}\mathbf{W}^{1/2}\boldsymbol{v}$ be the demand vector. Let $\boldsymbol{f}^{\star} \overset{\text{def}}{=} \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v} = \mathbf{W}^{1/2}\mathbf{B}\mathbf{L}^{-1}\boldsymbol{d}$, that is, $\boldsymbol{f}^{\star}$ is the electrical flow routing $\boldsymbol{d}$. In the first part of the proof, we show that $\widetilde{\boldsymbol{f}}$ routes the demand $\boldsymbol{d}$ (Lemma 7.16). In the second part of the proof, we show that $\widetilde{\boldsymbol{f}}$ is close to $\boldsymbol{f}^{\star}$.

Lemma 7.13. *Let $\boldsymbol{z} = \widetilde{\mathbf{\Gamma}}\mathbf{\Pi}^{(\eta-1)}\cdots\mathbf{\Pi}^{(0)}\mathbf{B}^{\top}\mathbf{W}^{1/2}\boldsymbol{v}$ be as given in Theorem 7.8. For each node $H \in \mathcal{T}$, let $\boldsymbol{z}|_{F_H}$ be the sub-vector of $\boldsymbol{z}$ supported on the vertices F_H, and define the demand $\boldsymbol{d}^{(H)} \overset{\text{def}}{=} \mathbf{L}^{(H)}\boldsymbol{z}|_{F_H}$. Then $\boldsymbol{d} = \sum_{H\in\mathcal{T}}\boldsymbol{d}^{(H)}$.*

Proof. In the proof, note that all $\mathbf{I}$ are $n \times n$ matrices, and we implicitly pad all vectors with the necessary zeros to match the dimensions. For example, $\boldsymbol{z}|_{F_H}$ should be viewed as an n-dimensional vector supported on F_H. Define

$$\mathbf{X}^{(i)} \overset{\text{def}}{=} \sum_{H\in\mathcal{T}(i)}\mathbf{X}^{(H)}.$$

We have

$$\mathbf{\Pi}^{(i)} = \mathbf{I} - \mathbf{X}^{(i)} = \mathbf{I} - \sum_{H \in \mathcal{T}(i)} \mathbf{L}^{(H)}_{\partial H, F_H} \left(\mathbf{L}^{(H)}_{F_H, F_H}\right)^{-1}.$$

Suppose H is at level i of $\mathcal{T}$. We have

$$\begin{aligned} \boldsymbol{z}|_{F_H} &= (\mathbf{L}^{(H)}_{F_H,F_H})^{-1}\mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d} \\ &= (\mathbf{L}^{(H)}_{F_H,F_H})^{-1}\mathbf{\Pi}^{(i-1)} \cdots \mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d}, \end{aligned} \tag{7.10}$$

where we use the fact $\mathrm{Im}(\mathbf{X}^{(H')}) \cap F_H = \emptyset$ if $\eta(H') \geq i$. From this expression for $\boldsymbol{z}|_{F_H}$, we have

$$\begin{aligned} \boldsymbol{d}^{(H)} &\stackrel{\text{def}}{=} \mathbf{L}^{(H)}\boldsymbol{z}|_{F_H} \\ &= \mathbf{L}^{(H)}_{\partial H, F_H}\boldsymbol{z}|_{F_H} + \mathbf{L}^{(H)}_{F_H,F_H}\boldsymbol{z}|_{F_H} \\ &= \mathbf{X}^{(H)}(\mathbf{\Pi}^{(i-1)} \cdots \mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d})_{F_H} + (\mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d})|_{F_H}, \end{aligned}$$

where the last line follows from Eq. (7.10). By padding zeros to $\mathbf{X}^{(H)}$, we can write the equation above as

$$\boldsymbol{d}^{(H)} = \mathbf{X}^{(H)}\mathbf{\Pi}^{(i-1)} \cdots \mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d} + (\mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d})|_{F_H}.$$

Now, computing the sum, we have

$$\begin{aligned} \sum_{H \in \mathcal{T}} \boldsymbol{d}^{(H)} &= \sum_{i=0}^{\eta} \sum_{H \in \mathcal{T}(i)} \mathbf{X}^{(H)}\mathbf{\Pi}^{(i-1)} \cdots \mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d} + \sum_{i=0}^{\eta} \sum_{H \in \mathcal{T}(i)} (\mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d})|_{F_H} \\ &= \left(\sum_{i=0}^{\eta} \mathbf{X}^{(i)}\mathbf{\Pi}^{(i-1)} \cdots \mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d}\right) + \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d} \qquad (F_H \text{ partition } V(G)) \\ &= \left(\sum_{i=0}^{\eta-1} (\mathbf{I} - \mathbf{\Pi}^{(i)})\mathbf{\Pi}^{(i-1)} \cdots \mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d}\right) + \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d} \\ &= \boldsymbol{d}, \qquad \text{(telescoping sum)} \end{aligned}$$

completing our proof. □

Next, we examine the feasibility of $\tilde{\boldsymbol{f}}$. To begin, we introduce a decomposition of $\tilde{\boldsymbol{f}}$ based on the decomposition of $\boldsymbol{d}$, and prove its feasibility.

Definition 7.14. Let $\mathbf{M}^{(H)}$ be the flow tree operator supported on the tree $\mathcal{T}_H$ (Definition 3.39). We define the flow $\tilde{\boldsymbol{f}}^{(H)} \stackrel{\text{def}}{=} \mathbf{M}^{(H)}\boldsymbol{z}|_{F_H}$.

Lemma 7.15. *We have that $(\mathbf{W}^{1/2}\mathbf{B})^\top \tilde{\boldsymbol{f}}^{(H)} = \boldsymbol{d}^{(H)}$. In other words, the flow $\tilde{\boldsymbol{f}}^{(H)}$ routes the demand $\boldsymbol{d}^{(H)}$ using the edges of the original graph G (in fact, the edges are all from the region H).*

Proof. We will first show inductively that for each $H \in \mathcal{T}$, we have $\mathbf{B}^\top \mathbf{W}^{1/2}\mathbf{M}^{(H)} = \mathbf{L}^{(H)}$. In the base case, if H is a leaf node of $\mathcal{T}$, then $\mathcal{T}_H$ is a tree with root H and a single leaf node under it. Then $\mathbf{M}^{(H)} = \mathbf{W}^{1/2}\mathbf{B}[H]$. It follows that

$$\mathbf{B}^\top \mathbf{W}^{1/2}\mathbf{M}^{(H)} = \mathbf{B}^\top \mathbf{W}^{1/2}\mathbf{W}^{1/2}\mathbf{B}[H] = \mathbf{L}^{(H)},$$

by definition of $\mathbf{L}^{(H)}$ for a leaf H of $\mathcal{T}$. In the inductive case, H is not a leaf node of $\mathcal{T}$. Let D_1, D_2 be the two children of H. Then

$$\begin{aligned}
\mathbf{B}^\top \mathbf{W}^{1/2}\mathbf{M}^{(H)} &= \mathbf{B}^\top \mathbf{W}^{1/2}\left(\mathbf{M}^{(D_1)}\mathbf{M}_{(D_1,H)} + \mathbf{M}^{(D_2)}\mathbf{M}_{(D_2,H)}\right) \\
&= \mathbf{L}^{(D_1)}\mathbf{M}_{(D_1,H)} + \mathbf{L}^{(D_2)}\mathbf{M}_{(D_2,H)} && \text{(by induction)} \\
&= \mathbf{L}^{(D_1)}(\mathbf{L}^{(D_1)})^{-1}\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)}, \partial D_1) + \mathbf{L}^{(D_2)}(\mathbf{L}^{(D_2)})^{-1}\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)}, \partial D_2) \\
&= \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)}, \partial D_1) + \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)}, \partial D_2) \\
&= \mathbf{L}^{(H)}.
\end{aligned}$$

Finally, we conclude that $(\mathbf{W}^{1/2}\mathbf{B})^\top \tilde{\boldsymbol{f}}^{(H)} = \mathbf{B}^\top \mathbf{W}^{1/2}\mathbf{M}^{(H)}\boldsymbol{z}|_{F_H} = \mathbf{L}^{(H)}\boldsymbol{z}|_{F_H} = \boldsymbol{d}^{(H)}$, where the last inequality follows by definition of $\boldsymbol{d}^{(H)}$. □

Lemma 7.16. *$\tilde{\boldsymbol{f}}$ is a feasible flow routing $\boldsymbol{d}$ on G.*

Proof. We first decompose $\boldsymbol{d} = \sum_{H \in \mathcal{T}} \boldsymbol{d}^{(H)}$ according to Lemma 7.13. By definition of the flow tree operator,

$$\tilde{\boldsymbol{f}} \stackrel{\text{def}}{=} \mathbf{M}\boldsymbol{z} \stackrel{\text{def}}{=} \sum_{H \in \mathcal{T}} \mathbf{M}^{(H)}\boldsymbol{z}|_{F_H} = \sum_{H \in \mathcal{T}} \tilde{\boldsymbol{f}}^{(H)},$$

where $\tilde{\boldsymbol{f}}^{(H)}$ routes demand $\boldsymbol{d}^{(H)}$ by Lemma 7.15. Hence,

$$(\mathbf{W}^{1/2}\mathbf{B})^\top \tilde{\boldsymbol{f}} = \sum_{H \in \mathcal{T}} (\mathbf{W}^{1/2}\mathbf{B})^\top \tilde{\boldsymbol{f}}^{(H)} = \sum_{H \in \mathcal{T}} \boldsymbol{d}^{(H)} = \boldsymbol{d}.$$

□

Next, we show $\tilde{\boldsymbol{f}}$ is close to $\boldsymbol{f}^\star$ in terms of energy. To start, we know $\tilde{\boldsymbol{f}}^{(H)}$ routes $\boldsymbol{d}^{(H)}$ in the region H, and we want to relate its energy to the minimum energy flow routing $\boldsymbol{d}^{(H)}$ in H:

Lemma 7.17. *Let H be a node at level i in $\mathcal{T}$. Given* any *$\boldsymbol{z}$, let $\boldsymbol{d} \stackrel{\text{def}}{=} \mathbf{L}^{(H)}\boldsymbol{z}$ be a demand. Then the flow $\boldsymbol{f} \stackrel{\text{def}}{=} \mathbf{M}^{(H)}\boldsymbol{z}$ satisfies $\|\boldsymbol{f}\|_2^2 \leq_{i\epsilon_{\mathbf{P}}} \mathcal{E}_{\mathbf{L}[H]}(\boldsymbol{d})$. Consequently,*

$$\left\|\tilde{\boldsymbol{f}}^{(H)}\right\|_2^2 \leq_{i\epsilon_{\mathbf{P}}} \mathcal{E}_{\mathbf{L}[H]}\left(\boldsymbol{d}^{(H)}\right).$$

Proof. We proceed by induction for the first part of the lemma. In the base case, H is a leaf node, and we have

$$\left\|\mathbf{M}^{(H)}\boldsymbol{z}\right\|_2^2 = \boldsymbol{z}^\top(\mathbf{B}[H])^\top\mathbf{W}\mathbf{B}[H]\boldsymbol{z} = \boldsymbol{z}^\top\mathbf{L}[H]\boldsymbol{z} = \mathcal{E}_{\mathbf{L}[H]}\left(\boldsymbol{d}\right).$$

Suppose H is at level $i > 0$ in $\mathcal{T}$, with children D_1 and D_2 at level at most $i-1$. Then

$$\begin{aligned}&\left\|\mathbf{M}^{(H)}\boldsymbol{z}\right\|_2^2\\ =\ &\left\|\left(\mathbf{M}^{(D_1)}\mathbf{M}_{(D_1,H)} + \mathbf{M}^{(D_2)}\mathbf{M}_{(D_2,H)}\right)\boldsymbol{z}\right\|_2^2\end{aligned}$$

Since $\mathrm{Range}(\mathbf{M}^{(D_1)})$ and $\mathrm{Range}(\mathbf{M}^{(D_2)})$ are orthogonal, we have

$$\begin{aligned}&= \left\|\mathbf{M}^{(D_1)}\mathbf{M}_{(D_1,H)}\boldsymbol{z}\right\|_2^2 + \left\|\mathbf{M}^{(D_2)}\mathbf{M}_{(D_2,H)}\boldsymbol{z}\right\|_2^2\\ &\leq_{(i-1)\epsilon_{\mathbf{P}}} \mathcal{E}_{\mathbf{L}[D_1]}\left(\mathbf{L}^{(D_1)}\mathbf{M}_{(D_1,H)}\boldsymbol{z}\right) + \mathcal{E}_{\mathbf{L}[D_2]}\left(\mathbf{L}^{(D_2)}\mathbf{M}_{(D_2,H)}\boldsymbol{z}\right)\\ &\qquad\qquad\text{(by inductive hypothesis with } \boldsymbol{z} = \mathbf{M}_{(D_i,H)}\boldsymbol{z})\\ &= \mathcal{E}_{\mathbf{L}[D_1]}\left(\mathbf{L}^{(D_1)}(\mathbf{L}^{(D_1)})^{-1}\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)}, \partial D_1)\boldsymbol{z}\right) + \mathcal{E}_{\mathbf{L}[D_2]}\left(\mathbf{L}^{(D_2)}(\mathbf{L}^{(D_2)})^{-1}\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)}, \partial D_2)\boldsymbol{z}\right)\\ &\qquad\qquad\text{(substituting the definition of } \mathbf{M}_{(D_i,H)})\\ &= \mathcal{E}_{\mathbf{L}[D_1]}\left(\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)}, \partial D_1)\boldsymbol{z}\right) + \mathcal{E}_{\mathbf{L}[D_2]}\left(\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)}, \partial D_2)\boldsymbol{z}\right).\end{aligned}$$

Since the demand vectors are supported on ∂D_1 and ∂D_2 respectively, we may take *exact* Schur complements and apply Lemma 7.12 with $\varepsilon = 0$ to get

$$= \mathcal{E}_{\mathbf{Sc}(\mathbf{L}[D_1],\partial D_1)}\left(\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)}, \partial D_1)\boldsymbol{z}\right) + \mathcal{E}_{\mathbf{Sc}(\mathbf{L}[D_2],\partial D_2)}\left(\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)}, \partial D_2)\boldsymbol{z}\right).$$

Theorem 4.11 guarantees $\mathbf{Sc}(\mathbf{L}[D_i], \partial D_i) \approx_{\epsilon_{\mathbf{P}}} \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_i)}, \partial D_i)$, so again by Lemma 7.12,

$$\begin{aligned}&\leq_{\epsilon_{\mathbf{P}}} \mathcal{E}_{\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)},\partial D_1)}\left(\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)}, \partial D_1)\boldsymbol{z}\right) + \mathcal{E}_{\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)},\partial D_2)}\left(\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)}, \partial D_2)\boldsymbol{z}\right)\\ &= \mathcal{E}_{\mathbf{L}^{(H)}}(\mathbf{L}^{(H)}\boldsymbol{z}). \qquad\qquad \text{(by Lemma 7.11)}\end{aligned}$$

Applying the lemma to $\boldsymbol{d}^{(H)} = \mathbf{L}^{(H)}\boldsymbol{z}|_{F_H}$ gives the bound on $\left\|\tilde{\boldsymbol{f}}^{(H)}\right\|_2^2$ as required. □

Next, we show that the sum of energies for routing the demand terms on different regions is approximately equal to the energy for routing the entire demand on G.

Lemma 7.18. *We have the following approximation of the energy of routing* $\boldsymbol{d}$*:*

$$\sum_{H \in \mathcal{T}} \mathcal{E}_{\mathbf{L}[H]}\left(\boldsymbol{d}^{(H)}\right) \approx_{(\eta+1)\epsilon_{\mathbf{P}}} \mathcal{E}_{\mathbf{L}}\left(\boldsymbol{d}\right).$$

Proof. We need the following matrix multiplication property: For any matrices $\mathbf{A}, \mathbf{B}, \mathbf{D}$,

$$\begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^{\top} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{7.11}$$

Recall in our setting, all matrices are padded with zeros so that their dimension is $n \times n$, and vectors padded with zeros so their dimension is n.

Define $\boldsymbol{\beta} \stackrel{\text{def}}{=} \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(1)} \mathbf{\Pi}^{(0)} \boldsymbol{d}$ for simplicity of notation, so that $\boldsymbol{z}|_{F_H} = \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1} \boldsymbol{\beta}$. Then,

$$\begin{aligned}
\mathcal{E}_{\mathbf{L}[H]}\left(\boldsymbol{d}^{(H)}\right) &\approx_{\epsilon_{\mathbf{P}}} \mathcal{E}_{\mathbf{L}^{(H)}}\left(\boldsymbol{d}^{(H)}\right) && \text{(by Lemma 7.12)} \\
&= \boldsymbol{z}^{\top}|_{F_H} \mathbf{L}^{(H)} \boldsymbol{z}|_{F_H} \\
&= \boldsymbol{\beta}^{\top} \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1} \mathbf{L}^{(H)} \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1} \boldsymbol{\beta} \\
&= \boldsymbol{\beta}^{\top} \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1} \boldsymbol{\beta}. && \text{(by Eq. (7.11))}
\end{aligned}$$

Summing over all $H \in \mathcal{T}$, we get

$$\begin{aligned}
\sum_{H \in \mathcal{T}} \mathcal{E}_{\mathbf{L}[H]}\left(\boldsymbol{d}^{(H)}\right) &\approx_{\epsilon_{\mathbf{P}}} \boldsymbol{\beta}^{\top} \sum_{H \in \mathcal{T}} (\mathbf{L}^{(H)}_{F_H,F_H})^{-1} \boldsymbol{\beta} \\
&= \boldsymbol{d}^{\top} \mathbf{\Pi}^{(0)\top} \cdots \mathbf{\Pi}^{(\eta-1)\top} \left[\sum_{H \in \mathcal{T}} (\mathbf{L}^{(H)}_{F_H,F_H})^{-1}\right] \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)} \boldsymbol{d} \\
&\approx_{\eta\epsilon_{\mathbf{P}}} \boldsymbol{d}^{\top} \mathbf{L}^{-1} \boldsymbol{d} \\
&= \mathcal{E}_{\mathbf{L}}(\boldsymbol{d}).
\end{aligned}$$

□

Finally, we conclude the proof of Theorem 7.8 by showing $\tilde{\boldsymbol{f}}$ is close to $\boldsymbol{f}^{\star} \stackrel{\text{def}}{=} \mathbf{P}_{\boldsymbol{w}} \boldsymbol{v}$.

Lemma 7.19. *We have* $\left\|\tilde{\boldsymbol{f}} - \boldsymbol{f}^{\star}\right\|_2 \leq O(\eta\epsilon_{\mathbf{P}}) \left\|\boldsymbol{v}\right\|_2$.

Proof. We know $\tilde{\boldsymbol{f}}$ routes $\boldsymbol{d}$ on G. Its energy is bounded by

$$\begin{aligned}
\left\|\tilde{\boldsymbol{f}}\right\|_2^2 &= \left\|\sum_{H\in\mathcal{T}} \tilde{\boldsymbol{f}}^{(H)}\right\|_2^2 \\
&\leq \left(\sum_{i=0}^{\eta}\left\|\sum_{H\in\mathcal{T}(i)} \tilde{\boldsymbol{f}}^{(H)}\right\|_2\right)^2 && \text{(by triangle inequality)} \\
&\leq (\eta+1)\cdot\sum_{i=0}^{\eta}\left\|\sum_{H\in\mathcal{T}(i)} \tilde{\boldsymbol{f}}^{(H)}\right\|_2^2 && \text{(by Cauchy-Schwarz)} \\
&\leq (\eta+1)\cdot\sum_{i=0}^{\eta}\sum_{H\in\mathcal{T}(i)}\left\|\tilde{\boldsymbol{f}}^{(H)}\right\|_2^2 && \text{(by orthogonality)} \\
&\leq (\eta+1)\cdot\sum_{i=0}^{\eta}\sum_{H\in\mathcal{T}(i)} e^{i\epsilon_{\mathbf{P}}}\cdot\mathcal{E}_{\mathbf{L}[H]}\left(\boldsymbol{d}^{(H)}\right) && \text{(by Lemma 7.17)} \\
&\approx_{O(\eta\epsilon_{\mathbf{P}})} \mathcal{E}_{\mathbf{L}}\left(\boldsymbol{d}\right). && \text{(by Lemma 7.18)}
\end{aligned}$$

Now we apply Lemma 7.10 to get that

$$\left\|\tilde{\boldsymbol{f}}-\boldsymbol{f}^\star\right\|_2^2 \leq O(\eta\epsilon_{\mathbf{P}})\cdot\|\boldsymbol{f}^\star\|_2^2 \leq O(\eta\epsilon_{\mathbf{P}})\cdot\|\boldsymbol{v}\|_2^2,$$

where the last inequality follows from the fact that $\mathbf{P}_{\boldsymbol{w}}$ is an orthogonal projection matrix.

□

7.4 Inexact Laplacian solver

In the previous subsections, and in the computation of $\boldsymbol{z}$, we have implicitly assumed an exact Laplacian solver, when in reality, we are using the SDD-solver from Theorem 7.1. This difference affects the feasibility of the solution updates, which we address here.

For the computation of $\boldsymbol{z}$ and the slack solution, we apply the second part of Theorem 7.1, which shows that for any SDD-matrix $\mathbf{M}$ and vector $\boldsymbol{d}$, the solver is in fact computing $\mathbf{Z}\boldsymbol{d}$ for some $\mathbf{Z}\approx\mathbf{M}^{-1}$. The spectral approximation is carried through the entire tree operator, therefore, as long as the solver solves to $\epsilon_{\mathbf{P}}$ accuracy, we still have $\widetilde{\mathbf{P}}_{\boldsymbol{w}} \approx_{O(\eta\epsilon_{\mathbf{P}})} \mathbf{P}_{\boldsymbol{w}}$ for the slack.

For the flow update, we have an additional issue of feasibility at each step. While $\|\tilde{\boldsymbol{f}} - \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}\|_2 \leq \widetilde{O}(\eta\epsilon_{\mathbf{P}})\|\boldsymbol{v}\|_2$ is satisfied even with the inexact Laplacian solver, we now do not immediately have $\mathbf{B}^\top\mathbf{W}^{1/2}\tilde{\boldsymbol{f}} = \mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}$; in other words, the update at each IPM step is not a circulation. To resolve this, recall the leaf region decomposition $\tilde{\boldsymbol{f}} = \sum_{\text{leaf } H} \tilde{\boldsymbol{f}}|_{E(H)}$, and $\boldsymbol{d} \stackrel{\text{def}}{=} \mathbf{B}^\top \mathbf{W}^{1/2} \boldsymbol{v} = \sum_{\text{leaf } H} \mathbf{B}[H]^\top \mathbf{W}^{1/2} \boldsymbol{v}$. We define the excess demand at each leaf node H by $\bar{\boldsymbol{d}}^{(H)} \stackrel{\text{def}}{=} \mathbf{B}[H]^\top \mathbf{W}^{1/2}(\boldsymbol{v} - \tilde{\boldsymbol{f}})$. The vector $\bar{\boldsymbol{d}}^{(H)}$ is indeed a demand on region H, which we can route exactly using a maximum-weighted spanning tree in time linear in the size of H. Let the resulting flow be denoted $\boldsymbol{r}|_{E(H)}$, and let $\boldsymbol{r} = \sum_{\text{leaf } H} \boldsymbol{r}|_{E(H)}$, which we also include in the flow update. Then $\mathbf{B}^\top \mathbf{W}^{1/2}(\tilde{\boldsymbol{f}} + \boldsymbol{r}) = \mathbf{B}^\top \mathbf{W}^{1/2} \boldsymbol{v}$, meaning we once again guarantee that the flow update is a circulation. Next, it remains to show $\|\boldsymbol{r}\|_2 \leq \widetilde{O}(\eta \epsilon_{\mathbf{P}}) \|\boldsymbol{v}\|_2$.

Let us first consider $\boldsymbol{r}|_{E(H)}$ for any leaf region H. We have

$$
\begin{aligned}
\|\boldsymbol{r}|_{E(H)}\|_2^2 &\leq O(1) \cdot \left(\|\boldsymbol{r}|_{E(H)}\|_\infty\right)^2 && \text{(regions are constant size)} \\
&\leq O(1) \cdot \left(\frac{\|\bar{\boldsymbol{d}}^{(H)}\|_1}{\boldsymbol{w}_{\min}^{1/2}}\right)^2 && (\boldsymbol{r} \text{ routes } \boldsymbol{d} \text{ in a weighted manner}) \\
&\leq O\left(\boldsymbol{w}_{\min}^{-1}\right) \cdot \left(\|\bar{\boldsymbol{d}}^{(H)}\|_1\right)^2 \\
&\leq O\left(\boldsymbol{w}_{\min}^{-1}\right) \cdot \left(\|\bar{\boldsymbol{d}}^{(H)}\|_2\right)^2. && \text{(again regions are constant size)}
\end{aligned}
$$

Next, since $\|\tilde{\boldsymbol{f}} - \mathbf{P}_{\boldsymbol{w}} \boldsymbol{v}\|_2 \leq \widetilde{O}(\eta \epsilon_{\mathbf{P}}) \|\boldsymbol{v}\|_2$, and $\mathbf{B}$ is an adjacency matrix, we also know that

$$
\left\| \sum_{\text{leaf } H} \bar{\boldsymbol{d}}^{(H)} \right\|_2^2 = \left\| \sum_{\text{leaf } H} \mathbf{B}[H]^\top \mathbf{W}^{1/2} (\boldsymbol{v} - \tilde{\boldsymbol{f}}) \right\|_2^2 \leq \widetilde{O}\left(\eta \epsilon_{\mathbf{P}} \boldsymbol{w}_{\max}\right) \cdot \|\boldsymbol{v}\|_2^2.
$$

We are happy to incur poly(m) error; hence, combining the two inequalities above with the bound on $\boldsymbol{w}_{\max}/\boldsymbol{w}_{\min}$ from earlier, we get

$$
\|\boldsymbol{r}\|_2^2 \leq \widetilde{O}(\eta \epsilon_{\mathbf{P}} \cdot \text{poly}(m)) \|\boldsymbol{v}\|_2^2,
$$

as required.

Finally, we bound the runtime of the solver; more specifically, we bound the term $\log(1/\lambda_2(\mathbf{L}))$. Whenever we apply the SDD-solver, the matrix is either $\mathbf{L}^{(H)}$ or $\mathbf{L}^{(H)}_{F_H, F_H}$ for a node H. Without loss of generality, we may assume the graphs associated with these matrices are connected, otherwise we independently solve for each connected component. By the Cauchy Interlacing Theorem, we know $\lambda_2(\mathbf{L}^{(H)}_{F_H, F_H}) \geq \lambda_2(\mathbf{L}^{(H)})$. Furthermore, recall $\mathbf{L}^{(H)} \approx_{\epsilon_{\mathbf{P}}} \mathbf{Sc}(\mathbf{L}^{(H)}, F_H \cup \partial H)$, and Schur complements are better conditioned than the original matrix, therefore $\lambda_2(\mathbf{L}^{(H)}) \geq \lambda_2(\mathbf{L})$ up to polynomial factors. Next, recall $\mathbf{L}$ is a weighted Laplacian with weights $(\nabla^2 \phi(\overline{\boldsymbol{f}}))^{-1}$ from the IPM, so to lowerbound $\lambda_2(\mathbf{L})$ up to polynomial factors, it suffices to lowerbound these weights. We have

$$
\mathbf{W}_{ii} = \left((\boldsymbol{u}_i - \overline{\boldsymbol{f}}_i)^{-2} + (\overline{\boldsymbol{f}}_i - \boldsymbol{l}_i)^{-2}\right)^{-1} \geq \eta_i^2/2,
$$

where η_i is the distance from $\overline{\boldsymbol{f}}_i$ to the boundary, i.e. $\boldsymbol{l}_i$ and $\boldsymbol{u}_i$. Corollary 2.16 shows that η_i is polynomially bounded in terms of the polytope parameters L, R, and the central path t, meaning that all $k \times k$ SDD-solves in our algorithm can be performed in time $\widetilde{O}(k \log M)$ time, where M is an upper bound on the values from the original problem inputs.

7.5 Main flow results

Theorem 7.20. *Let $G = (V, E)$ be a directed planar graph with n vertices and m edges. Assume that the demands $\boldsymbol{d}$, edge capacities $\boldsymbol{u}$ and costs $\boldsymbol{c}$ are all integers and bounded by M in absolute value. Then there is an algorithm that computes a minimum-cost flow satisfying demand $\boldsymbol{d}$ in $\widetilde{O}(n \log^2 M)$ expected time.*

Theorem 7.21. *Let $G = (V, E)$ be a directed graph with n vertices and m edges. Assume that the demands $\boldsymbol{d}$, edge capacities $\boldsymbol{u}$ and costs $\boldsymbol{c}$ are all integers and bounded by M in absolute value. Given a tree decomposition of G with width τ and size S, there is an algorithm that computes a minimum-cost flow in $\widetilde{O}(m\sqrt{\tau} \log M + S)$ expected time.*

The proofs for the two theorems are analogous. In either case, we combine the data structure framework established in Theorem 5.11, the definition of projection operators and their complexities in the previous sections of this chapter, and the parameters for separator trees for 1/2-separable (planar) and for treewidth τ graphs. The results follow.

As a direct corollary of Theorem 7.21, we can solve min-cost flow on any graph with n vertices, m edges and integral polynomially-bounded costs and constraints in $\widetilde{O}(m\sqrt{n})$ expected time, as the treewidth of any n-vertex graph is at most $\tau = n$, and the tree decomposition is trivially the graph itself. This result matches that of [114] obtained using the Lee-Sidford barrier for the IPM, which requires $\widetilde{O}(\sqrt{n})$ iterations. In contrast, we use the standard log barrier which requires $\widetilde{O}(\sqrt{m})$ iterations, and leverage the robustness of the IPM and custom data structures to reduce the amortized cost per iteration. We find it noteworthy that all other max flow results matching or beating our time require either the Lee-Sidford barrier ([31]) or significantly divergent IPM techniques ([39]).

7.6 Approximating treewidth

Using our faster max-flow algorithm as a subroutine, we can efficiently compute a tree decomposition of any given graph, where the width is within a $O(\log n)$-factor of the optimal:

Corollary 7.22. *Let $G = (V, E)$ be a graph with n vertices, m edges, and treewidth $\mathrm{tw}(G)$. There is an algorithm to find a tree decomposition of G with width at most $O(\mathrm{tw}(G) \cdot \log n)$*

in $\widetilde{O}(\mathrm{tw}(G)^3 \cdot m)$ expected time.

It is well known that computing the treewidth of a graph is NP-hard [11], and there is conditional hardness result for even constant-factor approximation algorithm [13]. For polynomial time algorithms, the best known result is a $O(\sqrt{\log \mathrm{tw}})$-approximation algorithm by [69], which involves solving a semidefinite program with n^2 variables.

There is a series of works focused on computing approximate treewidth for small treewidth graph in nearly-linear time; we refer the readers to [23] for a more detailed survey. Notably, [73] showed for any graph G, there is an algorithm to compute a tree decomposition of width $O(\mathrm{tw}(G)^2)$ in $\widetilde{O}(\mathrm{tw}(G)^7 \cdot n)$ time. [32] improved the running time to $\widetilde{O}(\mathrm{tw}(G)^3 \cdot m)$ with slightly compromised approximation ratio $O(\mathrm{tw}(G)^2 \cdot \log^{1+o(1)} n)$. More recently, [21] showed how to compute a tree decomposition of width $O(\mathrm{tw}(G) \cdot \log^3 n)$ in $O(m^{1+o(1)})$ time.

Our algorithm for Corollary 7.22 requires some tree decomposition of the graph as input. We use the following lemma to construct the initial tree decomposition.

Lemma 7.23 ([32])**.** *For any $\frac{2}{3} < \alpha < 1$ and $0 < \epsilon < 1 - \alpha$, given a graph G with n vertices and m edges, if the graph G contains an α-balanced vertex separator of size K, then there is a randomized algorithm that finds a balanced vertex separator of size $\widetilde{O}(K^2/\epsilon)$ in $\widetilde{O}(mK^3/\epsilon)$ expected time. The algorithm does not require knowledge of K.*

Next, the lemma below establishes the relationship between max flow and balanced edge separators. We first give the relevant definitions. For a given constant $c \leq 1/2$, a directed edge-cut $(S, \overline{S})$ is called a *c-balanced edge separator* if both $|S| \geq cn$ and $|\overline{S}| \geq cn$. The capacity of the cut $(S, \overline{S})$ is the total capacity of all edges crossing the cut. The *minimum c-balanced edge separator* is the c-balanced edge separator with minimum capacity. A λ *pseudo-approximation* to the minimum c-balanced edge separator is a c'-balanced cut $(S, \overline{S})$ for some other constant c', whose capacity is within a factor of λ of that of the minimum c-balanced edge separator.

Lemma 7.24 ([12])**.** *An $O(\log n)$ pseudo-approximation to the minimum c-balanced edge separator in directed graphs can be computed using* polylog$n$ *single-commodity flow computations on the same graph.*

Proof of Corollary 7.22. It is well known that given a $O(\log n)$ approximation algorithm for finding a balanced vertex separator, one can construct a tree decomposition of width $O(\mathrm{tw}(G) \log n)$. Specifically, the algorithm of [24] finds such a tree decomposition by recursively using a balanced vertex separator algorithm and requires only an additional log factor in the runtime.

Now, it suffices to show we can find a $\log(n)$ pseudo-approximation balanced vertex separator in $\widetilde{O}(m \cdot \mathrm{tw}(G)^3)$ expected time. Using the reduction from [120], we reduce the balanced vertex separator to directed edge separator on graph $G^* = (V^*, E^*)$, where

$$V^* = \left\{v \mid v \in V\right\} \cup \left\{v' \mid v \in V\right\},$$

and

$$E^* = \left\{\, (v, v') \mid v \in V\right\} \cup \left\{\, (u', v) \mid (u, v) \in E\right\} \cup \left\{\, (v', u) \mid (u, v) \in E\right\}.$$

We note that $\mathrm{tw}(G^*) = O(\mathrm{tw}(G))$. This shows G^* has a 2/3-balanced vertex separator of size $O(\mathrm{tw}(G))$. We first use Lemma 7.23 to construct a $\widetilde{O}(\mathrm{tw}(G)^2)$-separator tree for G^*. Then, we use the algorithm in [12] combined with our flow algorithm to find a balanced edge separator in $\widetilde{O}(m \cdot \mathrm{tw}(G))$ expected time. Hence, we can find a balanced vertex separator in $\widetilde{O}(m \cdot \mathrm{tw}(G)^3)$ expected time. $\square$

Chapter 8

Circle packing representation of planar graphs

This final chapter is a standalone result in computational geometry. The result is quite unique in the way it combines structural combinatorics with a black-box convex optimzation algorithm: To compute a circle packing representation of a given planar graph, it suffices to optimize a convex function that captures the requirements for a feasible circle packing. To efficiently solve the optimization problem via the black-box algorithm, we require the objective function to be strongly convex around the minimizer; we show this is indeed the case by proving new combinatorial properties of circle packing representations. The spirit of this thesis is arguably best embodied by this result, so without further ado:z

8.1 Introduction

Given a planar graph $G = (V, E)$, a *circle packing representation* of G consists of a vector of radii $\boldsymbol{r}$ indexed by V, and a straight line embedding of G in the plane given by $\boldsymbol{p} : V \mapsto \mathbb{R}^2 \times \mathbb{R}^2$, such that

1. For each vertex v at location $\boldsymbol{p}_v$, a circle C_v of radius $\boldsymbol{r}_v$ can be drawn centered at v,
2. all circles' interiors are disjoint, and
3. two circles C_u, C_v are tangent if and only if $uv \in E(G)$.

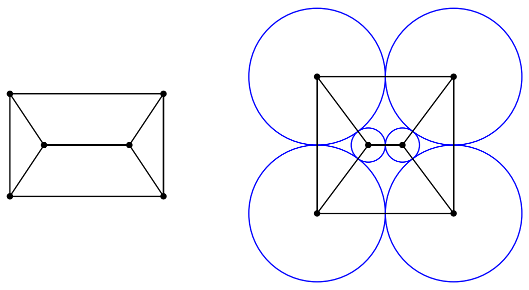


Figure 8.1: Example of a planar graph G and its circle packing representation.

It is easy to see that any graph with a circle packing representation is planar. Amazingly, the following deep and fundamental theorem asserts the converse is also true.

Theorem 8.1 (Koebe-Andreev-Thurston Circle Packing Theorem [107, 10, 158].)**.** *Every planar graph G admits a circle packing representation. Furthermore, if G is a triangulation, then the representation is unique up to Möbius transformations.*

Recall that every embedded planar graph has an associated planar dual graph, where each face becomes a vertex and each vertex a face. In our context, we will primarily focus on primal-dual circle packing, which intuitively consists of two circle packings, one for the original (primal) graph and another for the dual, that interact in a specific way. Formally:

Definition 8.2 (Simultaneous primal-dual circle packing.)**.** Let $G = (V, E)$ be a 3-connected planar graph, and $G^* = (V^*, E^*)$ its planar dual. Let f_∞ denote the unbounded face in a fixed embedding of G; it also naturally identifies a vertex of G^*.

The *(simultaneous) primal-dual circle packing representation of G with unbounded face f_∞* consists of vectors $\boldsymbol{r} \in \mathbb{R}^V, \tilde{\boldsymbol{r}} \in \mathbb{R}^{V^* \setminus f_\infty}$ and straight-line embeddings of G and $G^* - f_\infty$ in the plane such that:

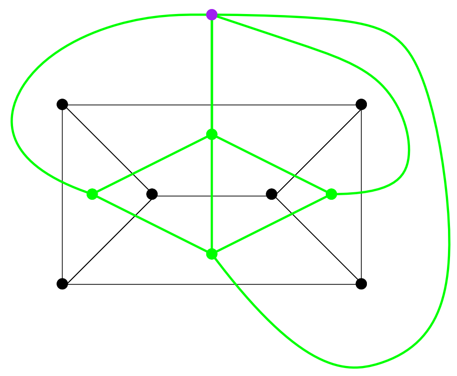
1. The radii $\boldsymbol{r}$, positions $\boldsymbol{p}_{\boldsymbol{r}}$, and circles $\{C_v \,:\, v \in V\}$ give a circle-packing of G,
2. The radii $\tilde{\boldsymbol{r}}$, positions $\boldsymbol{p}_{\tilde{\boldsymbol{r}}}$, and circles $\{C_f \,:\, f \in V^* - f_\infty\}$ is a circle packing of $G^* - f_\infty$,
3. C_{f_∞}, the circle corresponding to f_∞, has radius $\boldsymbol{r}_{f_\infty}$ and contains C_f for all $f \in V(G^*)$ in the plane. Furthermore, C_{f_∞} is tangent to C_g if and only if $f_\infty g \in E(G^*)$.
4. In the embedding, dual edges cross at a right angle, and no other edges cross. Furthermore, if $uv \in E(G)$ and $fg \in E(G^*)$ are a pair of dual edges, then C_u is tangent to C_v at the same point where C_f is tangent to C_g.

Section 8.2.1 provides more geometric intuitions regarding the definition.

The Circle Packing Theorem is generalized in this framework by Pulleyblank and Rote (unpublished), Brightwell and Scheinerman [33], and Mohar [129]:

Theorem 8.3. *Every 3-connected planar graph admits a simultaneous primal-dual circle packing representation. Furthermore, the representation is unique up to Möbius transformations.*

A primal-dual circle packing for a graph G naturally produces a circle packing of G by ignoring the dual circles. It also has a simple and elegant characterization based on angles in the planar embeddings, which we discuss in detail later. Moreover, the problem instance does not blow up in size compared to the original circle packing, since the number of faces is



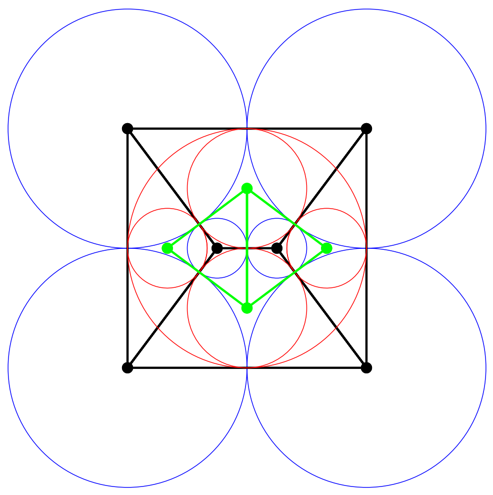


Figure 8.2: The planar graph G from the previous example is in black on top. Its planar dual, G^*, is overlaid in green. The vertex corresponding to the unbounded face f_∞ is marked in purple. On the bottom is the simultaneous primal-dual circle packing representation of G. The largest red circle is C_{f_∞}.

on the same order as the number of vertices in planar graphs.

We remark here that either the radii vectors or the embeddings suffice in defining the primal-dual circle packing representation: Given the radii, the locations of the vertices are uniquely determined up to isometries of the plane; the procedure for computing them is discussed in Section 8.2.2. Given the embedding, the radii are determined by the tangency requirements in Condition (4) of Definition 8.2.

8.1.1 Related works and applications

Circle packing representations have many connections to theoretical computer science and mathematics. The Circle Packing Theorem is used in the study of vertex separators: It gives a geometric proof of the Planar Separator Theorem of Lipton and Tarjan [128, 82]; an analysis of circle packing properties further gives an improved constant bound for the separator size [152]; it is also used crucially to design a simple spectral algorithm for computing optimal separators in graphs of bounded genus and degree [102]. In graph drawings, these representations give rise to straight-line planar embeddings; the existence of simultaneous straight-line planar embeddings of the graph and its dual, in which dual edges are orthogonal, was first conjectured by Tutte in his seminal paper in the area [160]. They are also used to prove the existence of Lombardi drawings and strongly monotone drawings for certain classes of graphs [66, 70]. Benjamini used the Circle Packing Theorem as a key component in his study of distributional limits of sequences of planar graphs [19]. In polyhedral combinatorics, Steinitz's Theorem states that a graph is formed by the edges and vertices of a 3-dimensional convex polyhedron if and only if it is a 3-connected planar graph. The theorem and its generalization, the Cage Theorem, can be proved using the (Primal-Dual) Circle Packing Theorem [171]. For a more comprehensive overview of the other related works, see Felsner and Rote [71].

In Riemannian geometry, circle packing of triangulations is tightly connected to the Riemann Mapping Theorem, which states that there is a conformal (angle-preserving) mapping between two simply connected open sets in the plane. Thurston had conjectured that circle packings can be used to construct approximate conformal maps; this was later proved by Rodin and Sullivan [146], which formed the basis of extensive work in discrete conformal mappings [86] and analytic functions [18, 61, 156]. An excellent high-level exposition of this research direction is given by Stephenson [157]. One unique and important application is in neuroscience research: Conformal maps, and specifically their approximations using circle packings, can be used to generate brain mappings while preserving structural information [79, 89]. This suggests a real-world interest in efficient circle packing algorithms.

Computationally, Bannister et al. [17] showed that numerical approximations of circle packing representations are necessary. Specifically, they proved for all large n, there exists graphs on n vertices whose exact circle packing representations involve roots of polynomials of degree $\Omega(n^{0.677})$; solving these exactly, even under extended arithmetic models, is impossible. Mohar [129, 130] gave a polynomial-time iterative algorithm to compute ε-approximations of primal-dual circle packings in two phases: the radii are approximated first, followed by the position of the vertices. The presentation was very recently simplified by Felsner and Rote [71]. However, because run-time was not the focus beyond demonstrating that it is polynomial, a rudimentary analysis of the algorithm puts the complexity at $\widetilde{\Omega}(n^5)$. For general circle packing, Alam et al. [6] gave algorithms with a more combinatorial flavour for special classes of graphs, including trees and outerpaths in linear time, and fan-free graphs in quadratic time. Chow [40] showed an algorithm based on Ricci flows that converges exponentially fast to the circle packing of the triangulation of a closed surface.

In practice, for general circle packing, there is a numerical algorithm `CirclePack` by Stephenson which takes a similar approach as Mohar and works well for small instances [48]. The current state-of-the-art is by Orick, Collins and Stephenson [137]; here the approach is to alternate between adjusting the radii and the position of the vertices at every step. The algorithm is implemented in the `GOPack` package in MATLAB; numerical experiments using randomly generated graphs of up to a million vertices show that it performs in approximately linear time. However, there is no known proof of convergence.

8.1.2 Our contribution

We follow the recent trend of attacking major combinatorial problems using tools from convex optimization. Although the combinatorial constraints on the radii had been formulated as a minimization problem in the past (e.g. by Colin de Vedière [47] and Ziegler [171]), the objective function is ill-conditioned, and therefore standard optimization techniques would only give a large polynomial time. Our key observation is that the primal-dual circle packing problem looks very similar to the minimum s-t cut problem when written as a function of the logarithm of the radii (see Equation (8.3)). Due to this formulation, we can combine recent techniques in interior point methods and Laplacian system solvers [108, 104, 113, 154, 109, 42, 141, 110, 112] to get a run-time of $\widetilde{O}(n^{1.5} \log R)$, where R is the ratio between the maximum and minimum radius of the circles. In the worst case, this ratio can be exponential in n; however the approach still gives a significantly improved run-time of $\widetilde{O}(n^{2.5})$.

For further improvements, our starting point is the recent breakthrough on matrix scaling problems [46], which showed that certain class of convex problems can be solved efficiently using vertex sparsifier chains [110]. When applied to the primal-dual circle packing problem, it gives a run-time of $\widetilde{O}(n \log^2 R)$, which is worse than interior point in the worst case. One of the $\log R$ term comes from the accuracy requirement for circle packing; this term seems to be unavoidable for almost all existing iterative techniques. The second term comes from the problem diameter.

To obtain a better bound, we present new properties of the primal-dual circle packing representation for triangulations using graph theoretic arguments. In particular, we show there is a spanning tree in a related graph such that the radii of neighbouring vertices are polynomially close to each other (Section 8.2.3). This allows us to show that the objective function is locally strongly convex (Lemma 8.26). Combining this with techniques in matrix scaling [46], we achieve a run-time of $\widetilde{O}(n \log R)$ for primal-dual circle packing for triangulations and general circle packing. Given the problem requires minimizing a convex function with accuracy $1/R$, we attain the natural run-time barrier of existing convex optimization techniques.

8.1.3 Our result

For primal-dual circle packing, we focus on triangulations, which are maximal planar graphs, and present a worst-case nearly quadratic time algorithm.

Theorem 8.4. *Let $G = (V, E)$ be a triangulation with dual $G^* = (V^*, E^*)$, where $|V| + |V^*| = n$. Let $f_\infty \in V^*$ denote its unbounded face. There is an explicit algorithm that finds radii $\boldsymbol{r} \in \mathbb{R}^n$ with $\boldsymbol{r}_{f_\infty} = 1$, and locations $\boldsymbol{p} \in \mathbb{R}^{2(n-1)}$ of $V \cup V^* - f_\infty$ in the plane, such that*

1. *there exists a target primal-dual circle packing representation of G with radii vector $\boldsymbol{r}^\star \in \mathbb{R}^n$ and vertex locations $\boldsymbol{p}^\star \in \mathbb{R}^{2(n-1)}$; furthermore, $\boldsymbol{r}^\star_{f_\infty} = 1$ and $\|\boldsymbol{r}^\star\|_\infty = O(1)$,*
2. *$1 - \varepsilon \le \boldsymbol{r}_u / \boldsymbol{r}^\star_u \le 1 + \varepsilon$ for each $u \in V \cup V^*$, and*
3. *$\|\boldsymbol{p}_u - \boldsymbol{p}^\star_u\|_\infty \le \varepsilon / R$ for each $u \in V \cup V^* - f_\infty$,*

where $R = r^\star_{\max} / r^\star_{\min}$ is the ratio between the maximum and minimum radius in the target representation. The algorithm is randomized and runs in expected time

$$\widetilde{O}\left(n \log \frac{R}{\varepsilon}\right).$$

Remark 8.5. We use the more natural ε/R for the location error instead of ε, in order to reflect the necessary accuracy at the smallest circle, which has radius $\Theta(1/R)$. We use $\widetilde{O}$ in the runtime to hide a $\mathrm{poly}(\log(n), \log\log(R/\varepsilon))$ factor.

From Theorem 8.4, an algorithm for general circle packing is easily obtained.

Theorem 8.6. *Let G be* any *planar graph where $|V(G)| = n$. There is an explicit algorithm that finds radii $\boldsymbol{r} \in \mathbb{R}^n$ and locations $\boldsymbol{p} \in \mathbb{R}^{2n}$, such that*

1. *there exists a target circle packing of G with radii vector $\boldsymbol{r}^\star \in \mathbb{R}^n$ and vertex locations $\boldsymbol{p}^\star \in \mathbb{R}^{2n}$, and $\|\boldsymbol{r}\|_\infty = O(1)$,*
2. *$1 - \varepsilon \leq \boldsymbol{r}_u/\boldsymbol{r}^\star_u \leq 1 + \varepsilon$ for each $u \in V$, and*
3. *$\|\boldsymbol{p}_u - \boldsymbol{p}^\star_u\|_\infty \leq \varepsilon/R$ for each $u \in V(G)$,*

where $R = r^\star_{\max}/r^\star_{\min}$ is the ratio between the maximum and minimum radius in the target representation. The algorithm is randomized and runs in expected time

$$\widetilde{O}\left(n \log \frac{R}{\varepsilon}\right).$$

Remark 8.7. R is a natural parameter of the circle packing problem and is poly(n) for several classes of graphs as described in [6]; it is bounded by $(2n)^n$ in the worst case (Corollary 8.17). When R is poly(n), our algorithm achieve nearly linear-time complexity.

8.2 Solution characterization

In this section, we present some structural properties of primal-dual circle packing representations which will be crucial to the algorithm. We begin with a review of basic graph theory concepts.

A *plane graph* is a planar graph G with an associated planar embedding. The embedding encodes additional information beyond the vertex and edge sets of G; in particular, it defines the faces of G and therefore a *cyclic ordering* of edges around each vertex. It is folklore that any 3-connected planar graph has a well-defined set of faces.

The *dual graph* of a plane graph G is denoted by G^*. Its vertex set is the set of faces of G, and two vertices are adjacent in G^* whenever the corresponding faces in G share a common edge on their boundary. Note that there is a natural bijection between the edges of G^* and the edges of G. We denote the unbounded face of a plane graph G by f_∞.

8.2.1 Representations on the extended plane

In the definition of primal-dual circle packing representation, we specified that the embeddings are in the Euclidean plane. For a more intuitive view, consider the embeddings in the extended plane with the appropriate geometry: Here, Conditions (2) and (3) in Definition 8.2 collapse

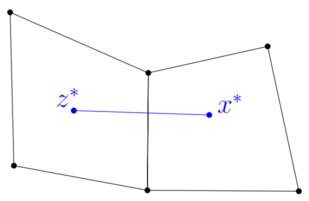


Figure 8.3: Two bounded faces z, x of a plane graph P is shown in black; they correspond to dual vertices z^* and X^*, in blue. The dual of the edge $z^*x^* \in E(P^*)$ is the unique edge that z^*x^* crosses in the embedding; note that it is on the boundaries of both faces z and x in P.

into one, which asks for a valid circle packing of G^* in the extended plane, such that C_{f_∞} is a circle centered at infinity (its radius becomes irrelevant). All other tangency requirements hold as before, and the interaction between the primal and dual embeddings are not changed.

This view ties into Möbius transforms, mentioned in Theorems 8.1 and 8.3. A *Möbius transform* is an angle-preserving map of the extended plane to itself; moreover, it maps circles to lines or circles. It can be shown that for any two faces f, g of G, a primal-dual circle packing representation of G with unbounded face f can be obtained from one with unbounded face g via an appropriately defined Möbius transform. Furthermore, the roles of G and G^* become interchangeable.

For our algorithms, we compute a primal-dual circle packing representation of G after fixing an unbounded face, and do not concern ourselves with these transforms. We continue with the original definition of embedding in the Euclidean plane.

8.2.2 Angle graph

Given a 3-connected plane graph $G = (V, E)$, the *angle graph* of G is the bipartite plane graph $\hat{H}_G = (V \cup V^*, E(\hat{H}))$ constructed as follows: For each vertex $v \in V$, fix its position in the plane based on G; place a vertex f in each face of G (including the unbounded face f_∞); connect $v, f \in V(\hat{H}_G)$ with a straight line segment if and only if v is a vertex on the boundary of f in G. When the original graph G is clear, we simply write $\hat{H}$. It is convenient to also define the *reduced angle graph* H, obtained from $\hat{H}$ by removing the vertex corresponding to f_∞. H is again a bipartite plane graph; all its bounded faces are of size four.

The (reduced) angle graph is so named because of the properties that become apparent when its embedding derives from a primal-dual circle packing representation of G: Specifically, suppose $\boldsymbol{r}, \boldsymbol{p}$ are the radii and location vectors of a valid representation, and that the locations of vertices of H are given by $\boldsymbol{p}$. Note that G's outer cycle $C_o = (s_1, \ldots, s_k)$ must be embedded

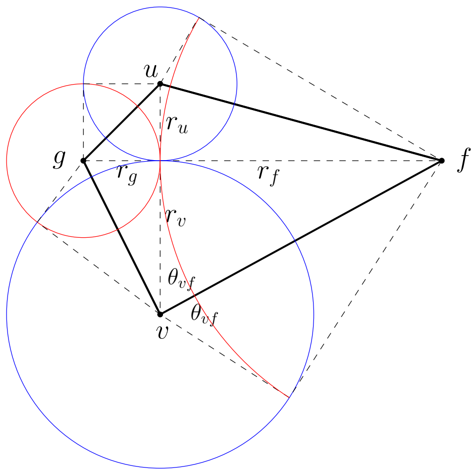


Figure 8.4: An illustration of the structure of H locally, with edges of H shown in black. Primal circles are in blue and dual circles in red. Vertices u, f, v, g define the boundary of a face; $u, v \in V$ and $f, g \in V^*$. The edges $uv \in E$ and $fg \in E^*$ are dual to each other and cross at a right angle, as required by the circle packing representation. Observe, for example, C_v is partially covered by K_{vg} and K_{vf}.

as a convex polygon, in order for conditions on C_{f_∞} to be satisfied; suppose the polygon has interior angle α_i at vertex s_i. Then for any $u \in V(H)$,

$$\sum_{w\,:\,uw \in E(H)} \arctan \frac{\boldsymbol{r}_w}{\boldsymbol{r}_u} = \begin{cases} \pi & u \notin C_o \\ \alpha_i/2 & u \in C_o. \end{cases} \tag{8.1}$$

To see this, first observe that an edge uw in this embedding has a natural *kite* K_{uw} in the plane associated with it, formed by the vertices u, w and the two intersection points of C_u and C_w. (See, for example, edge vf in Figure 8.4.) Furthermore, distinct kites do not intersect in the interior. Suppose $u \notin C_o$, and let $w_1, \ldots, w_l$ denote its neighbours in cyclic order. Then C_v is covered by the kites $K_{uw_1}, \ldots, K_{uw_l}$, which all meet at the vertex u and are consecutively tangent. Each neighbour w_i contributes an angle of $2\arctan(\boldsymbol{r}_{w_i}/\boldsymbol{r}_u)$ at u for a total of 2π. For the vertices on C_o, it can be shown that if $u = s_i$, the kites will cover an angle equal to α_i.

Conversely, any $\boldsymbol{r} \in \mathbb{R}^{|V(H)|}$ with the above property *almost* suffices as the radii of a primal-dual circle packing representation. Indeed, we can embed H (and therefore G and $G^* - f_\infty$) based on $\boldsymbol{r}$ as follows: Fix any vertex u to start; embed the vertices in $N(u)$ in cyclic order around u, by forming the consecutively tangent kites using $\boldsymbol{r}$. The process continues in a breadth-first fashion until all the vertices are placed. By construction, this embedding with radii $\boldsymbol{r}$ satisfies conditions (1),(2),(4) in Definition 8.2. Moreover, the outer cycle of G forms a convex k-gon with interior angles $\alpha_1, \dots, \alpha_k$.

The following theorem states that vectors r satisfying Equation (8.1) must exist.

Theorem 8.8 ([130]). *Let G be a 3-connected plane graph with outer cycle $C_o = (s_1, \dots, s_k)$ and unbounded face f_∞. Let H be its reduced angle graph, and $\alpha_1, \dots, \alpha_k \in (0, \pi)$ such that $\sum_i \alpha_i = (k-2)\pi$. Then, up to scaling, there exists a* unique $\boldsymbol{r} \in \mathbb{R}^{|V(H)|}$ *satisfying Equation* (8.1)*.*

For our purposes, G is a triangulation with outer cycle $C_o = (s_1, s_2, s_3)$ and unbounded face f_∞. By Theorem 8.8, there exists $\boldsymbol{r} \in \mathbb{R}^{|V(H)|}$ such that Equation (8.1) is satisfied with $\alpha_i = \pi/3$ for $i = 1, 2, 3$. This gives rise to a primal-dual circle packing representation without C_{f_∞}, where the outer cycle C_o is embedded as a triangle with interior angles all equal to $\pi/3$, i.e an equilateral triangle. It follows that all the $\boldsymbol{r}_{s_i}$'s must be equal, and therefore we can take C_{f_∞} to be the unique circle inscribed in the outer triangle, leading to an overall valid representation.

This construction motivates the next definition.

Definition 8.9. For a triangulation G with outer cycle $C_o = (s_1, s_2, s_3)$ and unbounded face f_∞, the *C_o-regular primal-dual circle packing representation* of G is the unique representation where C_o is embedded as an equilateral triangle, and C_{f_∞} is the circle of radius 1 inscribed in the triangle.

Our algorithm will therefore focus on finding the C_o-regular representation, using the characterization of the radii from Theorem 8.8.

8.2.3 Existence of a good spanning tree

Throughout this section, G denotes a triangulation with outer cycle $C_o = (s_1, s_2, s_3)$ and unbounded face f_∞; $\boldsymbol{r}$ denotes the radii vector of the unique C_o-regular primal-dual circle packing of G; $\hat{H}$ denotes the angle graph of G; and H denotes the reduced angle graph.

The C_o-regular circle packing representation of G naturally gives rise to a simultaneous planar embedding of G, G^*, and H. All subsequent arguments will be in the context of this embedding.

Definition 8.10. A *good edge in $\hat{H}$ with respect to $\boldsymbol{r}$* is an edge $uw \in E(\hat{H})$ so that $1/(2n) \leq \boldsymbol{r}_u / \boldsymbol{r}_w \leq 2n$. A set of edges is good if each edge in the set is good. Predictably, what is not good is *bad*.

Since we examine the radius of a vertex in relation to those of its neighbours, the next definition is natural:

Definition 8.11. Let $u \in V(\hat{H})$. For any good edge uw, we say w is a *good* neighbour of u. For a bad edge uw, we say w is a *bad* neighbour of u; we further specify that w is a *large* neighbour if $\boldsymbol{r}_w/\boldsymbol{r}_u > 2n$ or a *small* neighbour if $\boldsymbol{r}_u/\boldsymbol{r}_w > 2n$.

Recall that H is a bipartite graph, with vertex partitions V and $V^* - f_\infty$. For the last piece of notation, we will call a vertex u of H a *V-vertex* if it is in the first partition, and call u an *F-vertex* if it is in the second partition.

Our main theorem in this section is the following:

Theorem 8.12. *There exists a good spanning tree in $\hat{H}$ with respect to $\boldsymbol{r}$.*

Proof. First, we consider $f_\infty \in V(\hat{H})$: It has radius 1, and is inscribed in the equilateral triangle with vertices $C_o = \{s_1, s_2, s_3\}$. Hence, $\boldsymbol{r}_{s_i} = \tan\frac{\pi}{3}$ for each $s_i \in C_o$. It follows that all of f_∞'s incident edges in $\hat{H}$ are good, so any good spanning tree in H extends to one in $\hat{H}$.

It remains to find a good spanning tree in H. To continue, we require the following lemma regarding a special circle packing structure.

Lemma 8.13. *Let C_1 and C_2 be two circles with centers X and Y and radii R_1, R_2 respectively, and tangent at a point P. Suppose without loss of generality $R_2 \leq R_1$. Let Q be a point of distance R_2/n from P, so that PQ and XY are perpendicular. Let L_1 be a line segment parallel to XY through Q with endpoints on C_1 and C_2. Let L_2 be the line parallel to XY, further away from XY than L_1, and tangent to C_2.*

Suppose we place a family $\mathcal{C} = \{D_1, \ldots, D_m\}$ of m internally-disjoint circles (of any radius) in the plane, where $m < n$, such that:

1. *no circles from $\mathcal{C}$ intersect C_1 or C_2 in the interior,*
2. *at least one circle from $\mathcal{C}$ intersects L_1, and*
3. *the tangency graph of $\mathcal{C}$ is connected,*

then all circles in $\mathcal{C}$ are contained in the region bounded by C_1, C_2, L_2.

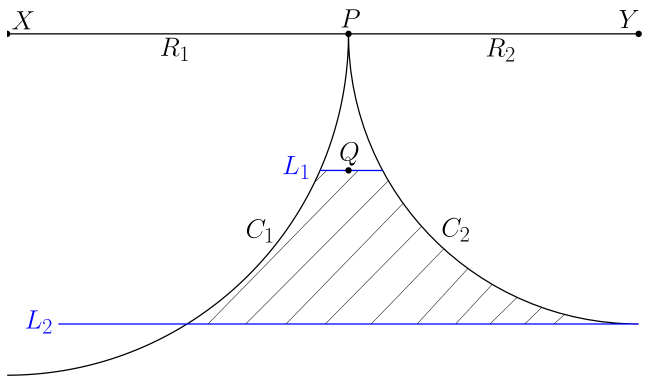


Figure 8.5: Illustration of Lemma 8.13. Any family $\mathcal{C}$ of circles satisfying the conditions of the lemma must be contained in the shaded region. The diagram is slightly deceptive: in reality Q is much closer to P than depicted.

Proof. Let $h(\mathcal{C})$ denote the maximum Euclidean distance between a point x on a circle in $\mathcal{C}$ and the line XY; in other words, $h(\mathcal{C}) = \max\{d(x, XY) \ : \ x \in \bigcup_{D\in\mathcal{C}} C\}$. We want to show $h(\mathcal{C}) < R_2$.

Suppose we place the circles one at a time while maintaining tangency, starting with D_1 intersecting L_1. By elementary geometry, it is clear that each additional circle D_i should be below D_{i-1}, tangent to only D_{i-1}, and be of maximum size possible, i.e. tangent to both C_1 and C_2. Given this observation, it suffices to consider when $R_1 = R_2 = R$.

In this case, $h(\mathcal{C})$ is maximized when all the circles are arranged as described above, and have their centers on the line through P and Q. Let a_i be the radius of D_i for each $i \in [m]$, and let $a_0 = d(P, Q)$. By the Pythagorean Theorem, we know that if $h_0 = d(P, Q)$ and $h_i = h(\{D_1, \dots, D_i\})$, then

$$(a_{i+1} + h_i)^2 + R^2 = (a_{i+1} + R)^2.$$

Hence we have the following recurrence relationship:

$$\begin{aligned} h_0 &= R/n \\ h_{i+1} &= h_i + 2a_{i+1} = h_i \left(1 + \frac{h_i}{R - h_i}\right) \end{aligned}$$

If we let $b_i = 2R/h_i$, then

$$\begin{aligned} b_0 &= 2n \\ \frac{2R}{b_{i+1}} &= \frac{2R}{b_i}\left(1 + \frac{2}{b_i - 2}\right) \\ b_{i+1} &= \frac{b_i}{1 + \frac{2}{b_i - 2}} \\ &= b_i - 2. \end{aligned}$$

So $h_m = \frac{R}{b_m} = \frac{R}{2n-2m} < R$, as desired. □

Recall that $\boldsymbol{r}$ satisfy the following angle constraints for each $u \in V(H) - C_o$:

$$\sum_{w\,:\,uw \in E(H)} \arctan \frac{\boldsymbol{r}_w}{\boldsymbol{r}_u} = \pi \tag{8.2}$$

Claim 8.14. *Every F-vertex in H can have at most one large neighbour. Furthermore, they have no small neighbours. Consequently, there are no F-vertices with only bad neighbours.*

Proof. W require G to be a triangulation, so that all F-vertices have degree three.

Suppose f is an F-vertex with 2 large neighbours v_1, v_2, and without loss of generality, $\boldsymbol{r}_{v_1} \le \boldsymbol{r}_{v_2}$. By the angle constraints in Equation (8.2), f's third neighbour u must be small.

Consider the primal-dual circle packing locally around f: The circles C_{v_1}, C_{v_2} are tangent at a point P, which is on the line L connecting the centers of the two circles. Furthermore, C_f is tangent to L at the point P. By the definition of large neighbours, we know $\boldsymbol{r}_f < \boldsymbol{r}_1/2n$. Moreover, C_u must intersect C_f, so C_u is at a distance of at most $2r_f$ away from P. Now, let us restrict our attention to primal circles (which include C_{v_1}, C_{v_2}, C_u) and apply Lemma 8.13.

Let $N_G(v_1) = \{v_2, u = w_1, \ldots, w_l\}$ denote the neighbours of v_1 in G in cyclic order. Since G is a triangulation, we know that $w_i w_{i+1} \in E(G)$ for each $i \in [l]$, and $w_l v_2 \in E(G)$. This means in the primal circle packing, C_{w_i} is tangent to $C_{w_{i+1}}$ for each i, and C_{w_l} is tangent to C_{v_2}. There are two cases to consider:

1. $v_1 \notin C_o$: In this case, v_1, v_2, w_l are the vertices of a bounded face of G. Hence, the circles $C_{v_2}, C_u, C_{w_1}, \ldots, C_{w_l}$ are consecutively tangent and surround C_{v_1}. This contradicts the conclusion of Lemma 8.13.

2. $v_1 \in C_o$: Note that the primal circles with the largest radii correspond to the vertices in C_o. Since $\boldsymbol{r}_{v_2} \ge \boldsymbol{r}_{v_1}$, we must have $\boldsymbol{r}_{v_2} = \boldsymbol{r}_{v_1}$ and $v_2 \in C_o$. Then $w_l \in C_o$ must be the third vertex on the boundary of f_∞, with v_1, v_2, w_l forming an equilateral triangle.

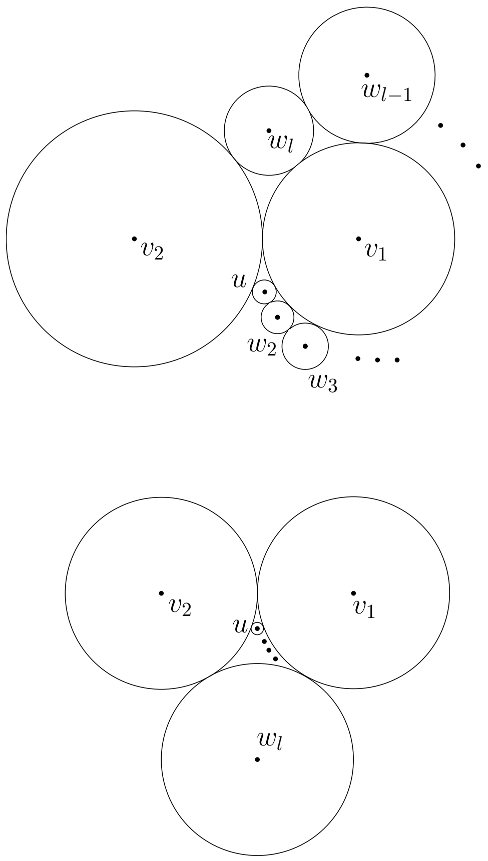


Figure 8.6: Illustration of the primal circles around v_1. There are two possible cases: in the former, circles corresponding to the neighbours of v_1 surround C_{v_1}; in the latter, they do not. Ellipses indicate additional C_{w_i}'s that are tangent to C_{v_1}.

Staring at the position of C_{w_l}, we see that this also contradicts the conclusion of Lemma 8.13.

So we have shown that f has at most one large neighbour.

Suppose f has a small neighbour u and two other neighbours v, w, at most one of which is large. Then by the angle constraints in (8.2),

$$\begin{aligned}\pi &= \arctan\frac{\boldsymbol{r}_u}{\boldsymbol{r}_f} + \arctan\frac{\boldsymbol{r}_v}{\boldsymbol{r}_f} + \arctan\frac{\boldsymbol{r}_w}{\boldsymbol{r}_f} \\ &< \arctan n^{-2} + \arctan n^2 + \pi/2 \\ &= \pi,\end{aligned}$$

a contradiction. □

Claim 8.15. *There are no V-vertices in H with only bad neighbours.*

Proof. Suppose v is a V-vertex with only bad neighbours. Again, by Equation (8.2), two of its neighbours are large and the remaining are small. Let f denote one of its large neighbours. But then v is a small neighbour of f, contradicting the previous claim. □

We have shown there are no vertices incident to only bad edges. Before proceeding, we observe the following:

Claim 8.16. *Recall $C_o = (s_1, s_2, s_3)$ is the outer cycle of G. Let t_1, t_2, t_3 be F-vertices corresponding to faces in G that are adjacent to f_∞. Let $B = (s_1, t_1, s_2, t_2, s_3, t_3)$ be the outer cycle of H. Then all the edges of $H[B]$ are good.*

Proof. Suppose without loss of generality that $s_i t_i$ is a bad edge. Since s_1, s_2, s_3 have equal radii, $t_i s_{i+1}$ must also be a bad edge. This contradicts Claim 8.14 which specifies that t_i has no small neighbours and at most one large neighbour. □

It remains to show there are no bad cuts in H. Suppose for a contradiction $T \subset E(H)$ is a minimal bad cut. Since H is a planar graph, T^* is a cycle in the dual graph H^*.

For a face z_i in H, we denote its dual vertex in H^* by z_i^*. Recall the dual of an edge $z^*x^* \in E(H^*)$ is a well-defined edge that is contained in both boundaries of $z, x \in F(H)$. Suppose the edges of T^*, in order, are $z_1^*z_2^*, z_2^*z_3^*, \ldots, z_k^*z_1^*$. Then $(z_1, \ldots, z_k)$ is a sequence of distinct faces of H such that $T = \{e_1, e_2, \ldots, e_k\}$, where e_i is a well-defined edge on the boundaries of both z_i and z_{i+1}, and e_k is on the boundaries of both z_k and z_1.

Consider $H[T]$, the subgraph induced by the edges of T: Since F-vertices in $H[T]$ have degree one, the components of $H[T]$ must be star graphs. If there is only one component, say with center u and leaves $N(u)$, then T disconnects u from the rest of the graph, contradicting the fact that u must have a good neighbour. Hence there must be at least two components in $H[T]$. (For example, in Figure 8.7, $H[T]$ is in red and consists of 4 components.)

Suppose e_i and e_{i+1} are in distinct components of $H[T]$. Both edges are on the boundary of face z_i. By Claim 8.16, we know z_i is not the unbounded face of H. Recall each bounded face of H has size four, hence we may denote the four vertices on the boundary of z_i by u, f, v, g, where u, v are V-vertices and f, g are F-vertices. Suppose without loss of generality $e_i = uf$. Then since e_i and e_{i+1} are not connected, we must have $e_{i+1} = vg$.

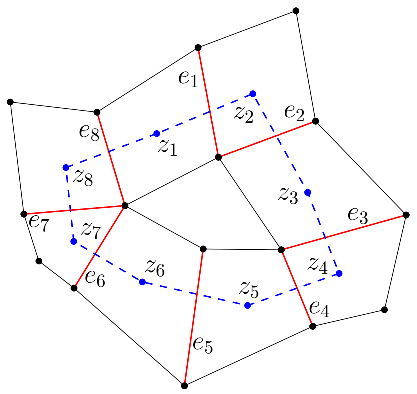


Figure 8.7: An illustration of what T looks like in H with respect to the dual. A subgraph of H is shown in black, with edges of T highlighted in red. Each z_i denotes a face in H, and correspond to a vertex z_i^* in the dual. T^* is shown in dashed blue. (Note the vertices are in the correct relative locations but do not necessarily reflect a proper circle packing representation.)

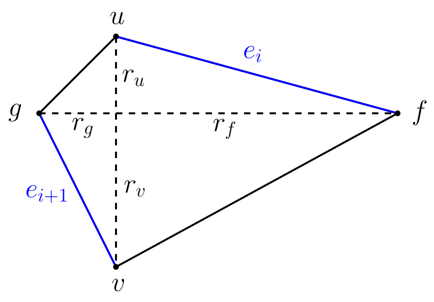


Figure 8.8: The face z_i in H. Note the radii are not necessarily accurate.

Recall $\boldsymbol{r}_u > 2n \cdot \boldsymbol{r}_f$ and $\boldsymbol{r}_v > 2n \cdot \boldsymbol{r}_g$ by definition of bad edges and the fact that F-vertices only have big neighbours. Consider the edge fv:

1. If $\boldsymbol{r}_v > 2n \cdot \boldsymbol{r}_f$, then both u and v are large neighbours of f;

2. If $n^2 \cdot \boldsymbol{r}_f \geq \boldsymbol{r}_v$, then $\boldsymbol{r}_u > 2n \cdot \boldsymbol{r}_f \geq \boldsymbol{r}_v > 2n \cdot \boldsymbol{r}_g$, so both u and v are large neighbours of g.

In both cases, we get a contradiction to Claim 8.14. It follows that there are no bad cuts in H, which concludes the overall proof. $\square$

As a corollary of Theorem 8.12, we have the following:

Corollary 8.17 ([130]). *Let G be a triangulation with $|V(G)| + |V(G^*)| = n$. Let $\boldsymbol{r}$ be the radii vector of a valid primal-dual circle packing for G. Then $\boldsymbol{r}_{\max}/\boldsymbol{r}_{\min} \leq (2n)^n$.*

We remark here that if the maximum degree of G is Δ, then $2n$ in the definition of good edge can be replaced by 2Δ, and the good tree proof would still hold true. Furthermore, for any edge $vf \in E(H)$, we can show there is a good path from v to f of length $O(\Delta)$ by a careful case analysis around vertex v similar to above. It follows that $\boldsymbol{r}_{\max}/\boldsymbol{r}_{\min} \leq \Delta^{O(\Delta D)}$ where D is the diameter of G. The proof is omitted.

Finally, although we assume in this section that the original graph G is a triangulation, we conjecture the analogous result holds for general graphs:

Conjecture 8.18. *Let G be a 3-connected planar graph, and let $\hat{H}_G$ be its angle graph. Suppose r is the radii vector of a valid primal-dual circle packing representation for G. Then there exists a good tree in $\hat{H}_G$ with respect to r.*

8.3 Computing the primal-dual circle packing

Throughout this section, G denotes the triangulation with outer cycle $C_o = (s_1, s_2, s_3)$ and unbounded face f_∞ given as input to the algorithm; $\hat{H}$ denotes the angle graph of G and H the reduced angle graph. Let $n = |V(G)| + |F(G)| - 1 = |V(H)|$. We index vectors by vertices rather than integers. Our goal is to compute the radii for the C_o-regular representation of G. Recall $\boldsymbol{r}_{f_\infty}, \boldsymbol{r}_{s_1}, \boldsymbol{r}_{s_2}, \boldsymbol{r}_{s_3}$ are fixed by definition of C_o-regularity.

8.3.1 Convex formulation

We transform the combinatorial question of finding the radii into a minimization problem of a continuous function. A variant of this formulation was first given in [47].

Definition 8.19. Consider the following convex function Φ over $\mathbb{R}^{V(H) \setminus C_o}$:

$$\Phi(\boldsymbol{x}) \stackrel{\text{def}}{=} 2\pi \sum_{u \in V(H)} \boldsymbol{x}_u + \sum_{uw \in E(H)} \Big(F(\boldsymbol{x}_u - \boldsymbol{x}_w) + F(\boldsymbol{x}_w - \boldsymbol{x}_u) - \frac{\pi}{2}(\boldsymbol{x}_u + \boldsymbol{x}_w) \Big) \qquad (8.3)$$

where $F(x) = \int_{-\infty}^{x} \arctan(e^t)dt$, and instances of $\boldsymbol{x}_{s_i}$ in the expression take constant value of $\log\tan(\frac{\pi}{3})$ for all $s_i \in C_o$.

The construction of this function Φ is motivated by the optimality condition at its minimum

$\boldsymbol{x}^\star$: for all $u \in V(H) \setminus C_o$,

$$\begin{aligned}
0 &= \frac{\partial \Phi}{\partial \boldsymbol{x}_u}(\boldsymbol{x}^\star) && (8.4)\\
&= \sum_{w\,:\,uw \in E(H)} \left(F'(\boldsymbol{x}_u^\star - \boldsymbol{x}_w^\star) - F'(\boldsymbol{x}_w^\star - \boldsymbol{x}_u^\star) - \frac{\pi}{2} \right) + 2\pi \\
&= \sum_{w\,:\,uw \in E(H)} \left(\arctan(e^{\boldsymbol{x}_u^\star - \boldsymbol{x}_w^\star}) - \arctan(e^{\boldsymbol{x}_w^\star - \boldsymbol{x}_u^\star}) - \frac{\pi}{2} \right) \\
&\qquad + 2\pi \\
&= -2 \sum_{w\,:\,uw \in E(H)} \arctan(e^{\boldsymbol{x}_w^\star - \boldsymbol{x}_u^\star}) + 2\pi
\end{aligned}$$

where we used that $\arctan(u) + \arctan(\frac{1}{u}) = \pi/2$ for all $u > 0$ at the end. Hence, $\exp(\boldsymbol{x}^\star)$ satisfies the angle constraints from Equation (8.1) for all $u \in V(H) \setminus C_o$.

8.3.2 Correctness

To show the minimizer of Φ gives a primal-dual circle-packing, we first show Φ is strictly convex, which implies the solution is unique.

Observe that the definitions are reminiscent of Laplacian matrices from earlier chapters.

Lemma 8.20. *For any x,*

$$\nabla^2 \Phi(\boldsymbol{x}) = \sum_{uw \in E(H)} 2F''(\boldsymbol{x}_u - \boldsymbol{x}_w) \boldsymbol{b}_{uw} \boldsymbol{b}_{uw}^\top$$

where $\boldsymbol{b}_{uw} \in \mathbb{R}^{V(H) \setminus C_o}$ is the vector with 1 in the u entry, -1 in the w entry, and zeros everywhere else. If u or v or both belongs to C_o, then $\boldsymbol{b}_{uv}$ has only one or no non-zero entries. Furthermore,

$$\nabla^2 \Phi(\boldsymbol{x}) \succcurlyeq \frac{2}{n^2} \cdot \min_{uw \in E(T)} F''(\boldsymbol{x}_u - \boldsymbol{x}_w) \cdot \mathbf{I} \succ 0$$

for any spanning tree $T \subset H$.

Proof. The formula of $\nabla^2 \Phi(\boldsymbol{x})$ follows from direct calculation. To prove $\nabla^2 \Phi(\boldsymbol{x})$ is positive-definite, we pick any spanning tree T in H. Note that

$$\begin{aligned}
\nabla^2 \Phi(\boldsymbol{x}) &\succcurlyeq \sum_{uw \in E(T)} 2F''(\boldsymbol{x}_u - \boldsymbol{x}_w) \boldsymbol{b}_{uw} \boldsymbol{b}_{uw}^\top \\
&\succcurlyeq \left(\min_{uw \in E(T)} 2F''(\boldsymbol{x}_u - \boldsymbol{x}_w) \right) \sum_{uw \in E(T)} \boldsymbol{b}_{uw} \boldsymbol{b}_{uw}^\top. && (8.5)
\end{aligned}$$

Fix any $\boldsymbol{h} \in \mathbb{R}^{V(H)\setminus C_o}$ with $\|\boldsymbol{h}\|_2^2 = 1$. Then

$$\sum_{uw\in E(T)} (\boldsymbol{b}_{uw}^\top \boldsymbol{h})^2 = \sum_{uw\in E(T)} (\boldsymbol{h}_u - \boldsymbol{h}_w)^2,$$

where we define $\boldsymbol{h}_s = 0$ for all $s \in C_o$. Since $\|\boldsymbol{h}\|_2^2 = 1$, there exists a vertex v such that $\boldsymbol{h}_v \geq \frac{1}{\sqrt{n}}$. Now, consider the path P from v to some $s \in C_o$. We have

$$\begin{aligned}\sum_{uw\in E(T)} (\boldsymbol{b}_{uw}^\top \boldsymbol{h})^2 &\geq \sum_{uw\in P} (\boldsymbol{h}_u - \boldsymbol{h}_w)^2 \\ &\geq \sum_{uw} (\frac{1}{\sqrt{n}|P|})^2 \\ &= \frac{1}{n|P|} \\ &\geq \frac{1}{n^2},\end{aligned}$$

where we used the fact that the minimum of $\sum_{uw\in P}(\boldsymbol{h}_u - \boldsymbol{h}_w)^2$ is attained by the vector $\boldsymbol{h}$ whose entries decrease from $\boldsymbol{h}_v = \frac{1}{\sqrt{n}}$ to $\boldsymbol{h}_s = 0$ uniformly on the path P. Using this in (8.5), we have that for any $\boldsymbol{h}$ with $\|\boldsymbol{h}\|_2^2 = 1$,

$$\boldsymbol{h}^\top \nabla^2 \Phi(\boldsymbol{x}) \boldsymbol{h} \geq \frac{2}{n^2} \cdot \min_{uw\in E(T)} F''(\boldsymbol{x}_u - \boldsymbol{x}_w)$$

Since $F''(x) = \frac{\exp(\boldsymbol{x})}{\exp(2x)+1} > 0$ for all x, we have

$$\nabla^2 \Phi(\boldsymbol{x}) \succcurlyeq \frac{2}{n^2} \cdot \min_{uw\in E(T)} F''(\boldsymbol{x}_u - \boldsymbol{x}_w) \cdot \mathbf{I} \succ 0.$$

This proves that Φ is strictly convex. □

Now, we prove that the minimizer of Φ is indeed a primal-dual circle packing.

Theorem 8.21. *Let $\boldsymbol{x}^\star$ be the minimizer of Φ. Then, $\boldsymbol{r}^\star = \exp(\boldsymbol{x}^\star)$, where the exponentiation is applied coordinate-wise, is the radii vector of the unique C_o-regular primal-dual circle packing representation of G.*

Proof. As discussed in Section 8.2.2, there exists a unique C_o-regular circle packing representation. Theorem 8.8 shows that the associated radii vector $\boldsymbol{r}$ satisfies

$$\sum_{w\,:\,uw\in E(H)} \arctan(\boldsymbol{r}_w/\boldsymbol{r}_u) = \pi,$$

for all $u \in V(H) \setminus C_o$. By the formula of $\nabla\Phi$ in Equation (8.4), we know $\nabla\Phi(\log \boldsymbol{r}) = 0$; therefore, $\boldsymbol{r}$ is a minimizer of Φ. Since Φ is strictly convex by Lemma 8.20, the minimizer is unique. Hence, $\boldsymbol{r}^\star = \boldsymbol{r}$. □

8.3.3 Algorithm for second-order robust functions

To solve for the minimizer of Φ, a convex programming result is used as a black-box. We define the relevant terminology below, and then present the theorem.

Definition 8.22. A function f is *second-order robust* with respect to ℓ_∞ if for any $\boldsymbol{x}, \boldsymbol{y} \in \operatorname{dom} f$ with $\|\boldsymbol{x}-\boldsymbol{y}\|_\infty \leq 1$,

$$\frac{1}{c}\nabla^2 f(\boldsymbol{x}) \preccurlyeq \nabla^2 f(\boldsymbol{y}) \preccurlyeq c\nabla^2 f(\boldsymbol{x})$$

for some universal constant $c > 0$.

Intuitively, the Hessian of a second-order robust function does not change too much within a unit ball.

Theorem 8.23 ([46, Thm 3.2]). *Let $g : \mathbb{R}^n \to \mathbb{R}$ be a second-order robust function with respect to ℓ_∞, such that its Hessian is symmetric diagonally dominant (SDD) with non-positive off-diagonals, and has m non-zero entries. Given a starting point $\boldsymbol{x}^{(0)} \in \mathbb{R}^n$, we can compute a point $\boldsymbol{x}$ such that $g(\boldsymbol{x}) - g(\boldsymbol{x}^\star) \leq \varepsilon$ in expected time*

$$\widetilde{O}\left((m+T)(1+D_\infty)\log\left(\frac{g(\boldsymbol{x}^{(0)})-g(\boldsymbol{x}^\star)}{\varepsilon}\right)\right)$$

where $\boldsymbol{x}^\star$ is a minimizer of g, $D_\infty = \sup_{\boldsymbol{x}\,:\,g(\boldsymbol{x})\leq g(\boldsymbol{x}^{(0)})} \left\|\boldsymbol{x}-\boldsymbol{x}^{(0)}\right\|_\infty$ is the ℓ_∞-diameter of the corresponding level-set of g, and T is the time required to compute the gradient and Hessian of g.

The algorithm behind the above result essentially uses Newton's method iteratively, each time optimizing within a unit ℓ_∞-ball. The key component involves approximately minimizing a SDD matrix with non-positive off-diagonals in nearly linear time, by recursively approximating Schur complements.

For our function Φ, there are two difficulties in using this theorem. First, the level-set diameter D_∞ could be very large because Φ is only slightly strongly-convex. So it would be better if D_∞ were replaced with the distance between $\boldsymbol{x}^{(0)}$ and $\boldsymbol{x}^\star$. Second, we are multiplying D_∞ and $\log(1/\varepsilon)$ in the runtime expression when both terms could be very large; we would like to add the two instead. It turns out both can be achieved at the same time by modifying the objective.

Theorem 8.24. *Let $g : \mathbb{R}^n \to \mathbb{R}$ be a second-order robust function with respect to ℓ_∞, such that its Hessian is symmetric diagonally dominant with non-positive off-diagonals, and has m non-zero entries. Let $\boldsymbol{x}^\star$ be the minimizer of g, and suppose that $\nabla^2 g(\boldsymbol{x}^\star) \succcurlyeq \alpha\mathbf{I}$ for some*

$\alpha < 1$. *Given a starting point $x^{(0)} \in \mathbb{R}^n$ and any $\varepsilon \leq \alpha/2$, we can compute a point $\boldsymbol{x}$ such that $g(\boldsymbol{x}) - g(\boldsymbol{x}^\star) \leq \varepsilon$ in expected time*

$$\widetilde{O}\left((m+T)\left(R_\infty \log^2\left(\frac{g(\boldsymbol{x}^{(0)}) - g(\boldsymbol{x}^\star)}{\alpha}\right) + \log\left(\frac{\alpha}{\varepsilon}\right)\right)\right)$$

where $R_\infty = \left\|\boldsymbol{x}^\star - \boldsymbol{x}^{(0)}\right\|_\infty$, and T is the time required to compute the gradient and Hessian of g. Furthermore, we have that $\|\boldsymbol{x} - \boldsymbol{x}^\star\|_2^2 \leq \varepsilon/\alpha$.

Proof. The algorithm builds on Theorem 8.23, and the high level idea can be broken into two steps: The first step transforms the dependence on the diameter of the level set in Theorem 8.23 to the ℓ_∞ distance $\left\|\boldsymbol{x}^{(0)} - \boldsymbol{x}^\star\right\|_\infty$ from the initial point; the second step leverages the strong-convexity at the minimum to obtain an improved running time.

For the first step, given the function $g(\boldsymbol{x})$ and an initial point $\boldsymbol{x}^{(0)}$, we construct an auxiliary function $\tilde{g}(\boldsymbol{x})$ that adds a small convex penalty reflecting the distance between x and the initial point $\boldsymbol{x}^{(0)}$. Analytically, this allows us to replace the dependency on the diameter of the level-set in Theorem 8.23 with the initial ℓ_∞-distance $\left\|\boldsymbol{x}^{(0)} - \boldsymbol{x}^\star\right\|$.

The second step leverages the fact that $\nabla g(\boldsymbol{x}^\star) \succcurlyeq \alpha\mathbf{I}$ at the minimum. Since the Hessian of g is also robust, it is $\succcurlyeq \Omega(\alpha)\mathbf{I}$ near the minimum. Strong convexity implies that the additive error at a point is proportional to the distance from the point to $\boldsymbol{x}^\star$. Hence, running a robust Newton's method to roughly α additive accuracy guarantees that the output point $\boldsymbol{x}^{(1)}$ is within an ℓ_∞ distance of roughly 1 from the minimum. The runtime to this point is less than running to ε-accuracy when $\alpha > \varepsilon$. We then run the algorithm a second time to ε-accuracy starting from $\boldsymbol{x}^{(1)}$; this instance has a much reduced R_∞ distance. The runtime for the two phases together is lower compared to running the algorithm just once starting from $\boldsymbol{x}^{(0)}$.

The above overview is informal; in particular, the two steps cannot be as cleanly separated as described. Indeed, when constructing the auxiliary function $\tilde{g}$, we require prior knowledge of the initial distance $R_\infty = \left\|\boldsymbol{x}^{(0)} - \boldsymbol{x}^\star\right\|_\infty$ within a constant factor. To overcome this, we use a standard doubling trick: Starting from a safe lower bound, we presuppose an estimate for R_∞ and run the two steps as above. If the estimate for R_∞ was too small (which we can detect), then we double our guess and try again. Since the overall runtime is proportional to R_∞, we do not add to it asymptotically.

We now describe the algorithm and prove the theorem in full detail. To begin, suppose $R_\infty := \left\|\boldsymbol{x}^\star - \boldsymbol{x}^{(0)}\right\|_\infty$ is given. To minimize g, we construct a new function

$$\tilde{g}(\boldsymbol{x}) = g(\boldsymbol{x}) + \frac{\varepsilon}{4n}\sum_i \cosh\left(\frac{\boldsymbol{x}_i - \boldsymbol{x}_i^{(0)}}{R_\infty}\right).$$

Note that if $\boldsymbol{x}^\dagger$ is the minimizer of $\tilde{g}$, then

$$\begin{aligned}
\tilde{g}(\boldsymbol{x}^\dagger) &= \min_{\boldsymbol{x}} \tilde{g}(\boldsymbol{x}) \\
&\le g(\boldsymbol{x}^\star) + \frac{\varepsilon}{4n}\sum_i \cosh\left(\frac{\boldsymbol{x}_i^\star - \boldsymbol{x}_i^{(0)}}{R_\infty}\right) \\
&\le g(\boldsymbol{x}^\star) + \frac{\varepsilon}{4n}\sum_i \cosh(1) \\
&\le g(\boldsymbol{x}^\star) + \frac{\varepsilon}{2},
\end{aligned}$$

and $\tilde{g}(\boldsymbol{x}) \ge g(\boldsymbol{x})$ for all x. Therefore, to minimize g with ε accuracy, it suffices to minimize $\tilde{g}$ with $\varepsilon/2$ accuracy.

We check the condition of Theorem 8.23 for $\tilde{g}$. The Hessian of $\tilde{g}$ is simply the Hessian of g plus a diagonal matrix, so $\nabla^2\tilde{g}$ is still SDD with non-positive off-diagonals. A simple calculation shows that $\tilde{g}$ is second-order robust. To bound $D_\infty := \sup_{\boldsymbol{x} \,:\, \tilde{g}(\boldsymbol{x}) \le \tilde{g}(\boldsymbol{x}^{(0)})} \left\|\boldsymbol{x} - \boldsymbol{x}^{(0)}\right\|_\infty$, note that for any $\boldsymbol{x}$ with $\tilde{g}(\boldsymbol{x}) \le \tilde{g}(\boldsymbol{x}^{(0)})$, we have

$$\begin{aligned}
\tilde{g}(\boldsymbol{x}^0) &= g(\boldsymbol{x}^{(0)}) + \varepsilon/4 \\
&\ge g(\boldsymbol{x}) + \frac{\varepsilon}{4n}\sum_i \cosh\left(\frac{\boldsymbol{x}_i - \boldsymbol{x}_i^{(0)}}{R_\infty}\right) \\
&\ge g(\boldsymbol{x}^\star) + \frac{\varepsilon}{8n}\exp\left(\frac{\|\boldsymbol{x} - \boldsymbol{x}^{(0)}\|_\infty}{R_\infty}\right).
\end{aligned}$$

Hence,

$$\begin{aligned}
D_\infty &= \sup_{\boldsymbol{x} \,:\, \tilde{g}(\boldsymbol{x}) \le \tilde{g}(\boldsymbol{x}^{(0)})} \|\boldsymbol{x} - \boldsymbol{x}^{(0)}\|_\infty \\
&\le R_\infty \log\left(\frac{8n}{\varepsilon}(g(\boldsymbol{x}^{(0)}) - g(\boldsymbol{x}^\star) + \varepsilon/4)\right).
\end{aligned}$$

We apply Theorem 8.23 to $\tilde{g}$ to get a point $\boldsymbol{x}$ such that $\tilde{g}(\boldsymbol{x}) - \tilde{g}(\boldsymbol{x}^\dagger) < \varepsilon/2$, using time

$$\begin{aligned}
&\widetilde{O}\left((m+T)(1+D_\infty)\log\left(\frac{\tilde{g}(\boldsymbol{x}^{(0)}) - \tilde{g}(\boldsymbol{x}^\dagger)}{\varepsilon/2}\right)\right) \\
= &\widetilde{O}\left((m+T)\left(1 + \log\left(\frac{g(\boldsymbol{x}^{(0)}) - g(\boldsymbol{x}^\star)}{\varepsilon}\right)R_\infty\right)\right. \\
&\left.\log\left(\frac{g(\boldsymbol{x}^{(0)}) - g(\boldsymbol{x}^\star)}{\varepsilon}\right)\right).
\end{aligned}$$

This $\boldsymbol{x}$ minimizes g to ε accuracy. Henceforth we view the above reduction from g to $\tilde{g}$ as a black-box. Now, we make some further observations regarding g.

Lemma 8.25. *For any constant $C \leq 1$ and $\boldsymbol{x}$ such that $\left\|\boldsymbol{x}-\boldsymbol{x}^{\star}\right\|_{\infty}=C$, we have $g(\boldsymbol{x}) \geq g(\boldsymbol{x}^{\star})+\Omega(\alpha)\cdot C^{2}$. Furthermore, if $\boldsymbol{x}'$ satisfies $g(\boldsymbol{x}')-g(\boldsymbol{x}^{\star}) \leq o(\alpha)\cdot C^{2}$, then $\left\|\boldsymbol{x}'-\boldsymbol{x}^{\star}\right\|_{\infty} \leq C$.*

Proof. Since $\nabla^{2} g(\boldsymbol{x}^{\star}) \succcurlyeq \alpha \cdot \mathbf{I}$ and g is second-order robust, $\nabla^{2} g(\boldsymbol{x}) \succcurlyeq \Omega(\alpha) \cdot \mathbf{I}$ for all $\boldsymbol{x}$ with $\left\|\boldsymbol{x}-\boldsymbol{x}^{\star}\right\|_{\infty} \leq 1$. Applying the Mean Value Theorem for $\boldsymbol{x}$ with $\left\|\boldsymbol{x}-\boldsymbol{x}^{\star}\right\|_{\infty}=C$, we get

$$g(\boldsymbol{x}) \geq g(\boldsymbol{x}^{\star})+\Omega(\alpha) \cdot\left\|\boldsymbol{x}-\boldsymbol{x}^{\star}\right\|_{2}^{2} \geq g(\boldsymbol{x}^{\star})+\Omega(\alpha) \cdot C^{2}. \tag{8.6}$$

Moreover, by convexity of g, we have $g(\boldsymbol{x}) \geq g(\boldsymbol{x}^{\star})+\Omega(\alpha)\cdot C^{2}$ for all $\boldsymbol{x}$ where $\|\boldsymbol{x}-\boldsymbol{x}^{\star}\|_{\infty} \geq C$. The second part of the lemma is the contrapositive. □

To achieve the runtime stated in the theorem, we minimize g in two phases. In the first phase, we use $\varepsilon_1 = \alpha/\log^2(\alpha/\varepsilon)$ and initial point $\boldsymbol{x}^{(0)}$ as given, to get a point $\boldsymbol{x}^{(1)}$ such that $g(\boldsymbol{x}^{(1)})-g(\boldsymbol{x}^{\star}) \leq \varepsilon_1$. By Lemma 8.25, we have $\|\boldsymbol{x}^{(1)}-\boldsymbol{x}^{\star}\|_{\infty} \leq 1/\log(\alpha/\varepsilon)$. In the second phase, we minimize to ε error. However, since $\boldsymbol{x}^{(1)}$ can be used as the initial point, we know $R_{\infty}=1/\log(\alpha/\varepsilon)$. The algorithm returns $\boldsymbol{x}^{(2)}$ such that $g(\boldsymbol{x}^{(2)})-g(\boldsymbol{x}^{\star}) \leq \varepsilon$. Summing the runtime of the two phases carefully, we get the desired total time,

$$\widetilde{O}\left((m+T)\left(R_{\infty}\log^{2}\left(\frac{g(\boldsymbol{x}^{(0)})-g(\boldsymbol{x}^{\star})}{\alpha}\right)+\log\left(\frac{\alpha}{\varepsilon}\right)\right)\right),$$

where factors of $\log\log(\alpha/\varepsilon)$ are hidden. The claim $\left\|\boldsymbol{x}^{(2)}-\boldsymbol{x}^{\star}\right\|_{2}^{2} \leq \varepsilon/\alpha$ follows from Equation (8.6) in Lemma 8.25.

Finally, we resolve the initial assumption of R_{∞} being given. Note that we only use R_{∞} during the first phase of the algorithm, where we use the target accuracy ε_1. To run the first phase without knowing R_{∞}, we apply Lemma 8.25 again in a doubling trick.

Let $\boldsymbol{x}^{(r)}=\arg\min_{\|\boldsymbol{x}\|_{\infty} \leq r} g(\boldsymbol{x})$. Consider $\hat{\boldsymbol{x}}=\frac{\boldsymbol{x}^{(r)}-\boldsymbol{x}^{\star}}{\|\boldsymbol{x}^{(r)}-\boldsymbol{x}^{\star}\|_{\infty}}$, which satisfies $\left\|\boldsymbol{x}^{\star}-(\boldsymbol{x}^{\star}+\hat{\boldsymbol{x}})\right\|_{\infty}= 1$. Hence, by Lemma 8.25,

$$g(\boldsymbol{x}^{\star}+\hat{\boldsymbol{x}}) \geq g(\boldsymbol{x}^{\star})+\Omega(\alpha).$$

Furthermore, $\boldsymbol{x}^{\star}+\hat{\boldsymbol{x}}$ is on the straight line connecting $\boldsymbol{x}^{\star}$ and $\boldsymbol{x}^{(r)}$. Since the slope of g is increasing from $\boldsymbol{x}^{\star}$ to $\boldsymbol{x}^{(r)}$, if $\|\boldsymbol{x}^{\star}\|_{\infty}>2r$, we also have

$$g(\boldsymbol{x}^{(r)}) \geq g(\boldsymbol{x}^{(r)}-\hat{\boldsymbol{x}})+\Omega(\alpha).$$

Note that $\left\|\boldsymbol{x}^{(r)}-\hat{\boldsymbol{x}}\right\|_{\infty} \leq 2r$. This shows that $\|\boldsymbol{x}^{\star}\|_{\infty}>2r$ implies

$$\min_{\|\boldsymbol{x}\|_{\infty} \leq r} g(\boldsymbol{x}) \geq \min_{\|\boldsymbol{x}\|_{\infty} \leq 2r} g(\boldsymbol{x})+\Omega(\alpha).$$

Hence, if $R_{\infty}>2r$, then we can detect it by comparing $\boldsymbol{x}^{(r)}$ and $\boldsymbol{x}^{(2r)}$. To estimate R_{∞}, first we run the algorithm while pretending $R_{\infty}=1$ and compare the result against $R_{\infty}=2$. If it fails this test, then we compare the result for $R_\infty = 2$ against $R_\infty = 4$, and so on. We stop when the test passes, at which point the guess for R_∞ is correct to a constant factor, and the true $\boldsymbol{x}^\star$ has been found. This does not affect the runtime asymptotically, since the total time simply involves a term $1 + 2 + \cdots + R_\infty$ instead of R_∞. □

8.3.4 Strong convexity at the minimum

To apply Theorem 8.24, we need to show Φ is strongly-convex at $\boldsymbol{x}^\star$.

Lemma 8.26. *Let $\boldsymbol{x}^\star$ be the minimizer of Φ. Then,*

$$\nabla^2 \Phi(\boldsymbol{x}^\star) \succcurlyeq \frac{1}{n^3}\mathbf{I}.$$

Proof. Lemma 8.20 shows that

$$\nabla^2 \Phi(\boldsymbol{x}) \succcurlyeq \frac{2}{n^2} \cdot \min_{uw \in E(T)} F''(\boldsymbol{x}_u - \boldsymbol{x}_w) \cdot \mathbf{I} \tag{8.7}$$

for any spanning tree $T \subset H$. Theorem 8.12 shows that there is a spanning tree T such that $|\boldsymbol{x}_u - \boldsymbol{x}_w| \leq \log 2n$ for any $uw \in E(T)$. Hence, we have

$$F''(\boldsymbol{x}_u - \boldsymbol{x}_w) \geq \exp(-|\boldsymbol{x}_u - \boldsymbol{x}_w|) \geq (2n)^{-1}$$

for any $uw \in E(T)$. Putting it into Equation (8.7) gives the desired bound. □

8.3.5 Result

We combine the previous sections for the overall result.

Theorem 8.27. *Let $\boldsymbol{r} \in \mathbb{R}^{V(H)\setminus C_o}$ be the radii of the C_o-regular primal-dual circle packing representation of the triangulation G. Let $\boldsymbol{x}^\star = \log \boldsymbol{r}$. For any $0 < \varepsilon < \frac{1}{2}$, we can compute a point $\boldsymbol{x}$ such that $\|\boldsymbol{x} - \boldsymbol{x}^\star\|_\infty \leq \varepsilon$ in expected time*

$$\widetilde{O}\left(n \log \frac{R}{\varepsilon}\right),$$

where $R = \boldsymbol{r}_{\max}/\boldsymbol{r}_{\min}$ is the ratio of the maximum to minimum radius of the circles.

Proof. We check the conditions of Theorem 8.24. Lemma 8.20 shows that

$$\nabla^2 \Phi(\boldsymbol{x}) = \sum_{uw \in E(H)} 2F''(\boldsymbol{x}_u - \boldsymbol{x}_w)\boldsymbol{b}_{uw}\boldsymbol{b}_{uw}^\top.$$

Since $F'' > 0$, we know $\nabla^2\Phi(\boldsymbol{x})$ is a positive combination of $\boldsymbol{b}_{uw}\boldsymbol{b}_{uw}^\top$'s, which are each SDD with non-positive off-diagonals. Hence, $\nabla^2\Phi(\boldsymbol{x})$ is an SDD matrix with non-positive off-diagonals.

To show second-order robustness, note that if $\boldsymbol{x}$ changes by at most 1 in the ℓ_∞-norm, then $\boldsymbol{x}_u - \boldsymbol{x}_w$ changes by at most 2. Recall

$$F''(\boldsymbol{x}_u - \boldsymbol{x}_w) = \frac{\exp(\boldsymbol{x}_u - \boldsymbol{x}_w)}{\exp(2(\boldsymbol{x}_u - \boldsymbol{x}_w)) + 1},$$

so F'' changes by at most a factor of e^2. It follows that $\nabla^2\Phi$ changes by at most a constant multiplicative factor.

Lemma 8.26 shows that $\nabla^2\Phi(\boldsymbol{x}^\star) \succcurlyeq \alpha\mathbf{I}$ with $\alpha = n^{-3}$.

Now, we can apply Theorem 8.24 as a black-box. Since the Hessian has $O(n)$ entries, each of which consists of a constant number of hyperbolic computations, both m and T are $O(n)$. A simple initial point $\boldsymbol{x}^{(0)}$ is the all zeros vector; it follows that $\Phi(\boldsymbol{x}^{(0)}) = O(n)$.

R_∞ in Theorem 8.24 is precisely $\|\boldsymbol{x}^\star\|_\infty$. Since $\boldsymbol{x}_u^\star = \log \boldsymbol{r}_u^\star \leq 0$ for all $u \in V(H) \setminus C_o$, we know $\boldsymbol{x}^\star$ satisfies $\|\boldsymbol{x}^\star\|_\infty = -\log r^\star_{\min} < \log(r^\star_{\max}/r^\star_{\min})$, where $r^\star_{\min}$ is the radius of the smallest circle in the true circle packing representation, and $r^\star_{\max} = \tan(\frac{\pi}{3})$ is the radius of the largest circle, attained by vertices in C_o. Therefore $R_\infty = \|\boldsymbol{x}^\star\|_\infty \leq \log R$.

Lastly we estimate $\Phi(\boldsymbol{x}^\star)$. Note that $\frac{\pi}{2}|z| \leq F(z) + F(-z) \leq \frac{\pi}{2}|z| + 2$ for all $z \in \mathbb{R}$. By Corollary 8.17, $\|\boldsymbol{x}^\star\|_\infty = \widetilde{O}(n)$ in the worst case. Hence,

$$\begin{aligned}
&-\Phi(\boldsymbol{x}^\star)\\
&\leq \sum_{uw\in E(H)} \left(\frac{\pi}{2}(\boldsymbol{x}_u^\star + \boldsymbol{x}_w^\star) - \frac{\pi}{2}|\boldsymbol{x}_u^\star - \boldsymbol{x}_w^\star|\right) + 2\pi \sum_{u\in V(H)} \boldsymbol{x}_u^\star\\
&\leq \widetilde{O}(n^2).
\end{aligned}$$

It follows that $\Phi(\boldsymbol{x}^{(0)}) - \Phi(\boldsymbol{x}^\star) \leq \widetilde{O}(n^2)$.

Now, Theorem 8.24 shows how to find $\boldsymbol{x}$ with $\|\boldsymbol{x} - \boldsymbol{x}^\star\|_\infty \leq \varepsilon$ in time

$$\widetilde{O}\left(n\left(\log R \log n + \log\frac{n}{\varepsilon}\right)\right) = \widetilde{O}\left(n \log \frac{R}{\varepsilon}\right).$$

Exponentiating the resulting $\boldsymbol{x}$, we get $\boldsymbol{r}$ such that $1 - 2\varepsilon \leq \boldsymbol{r}_u/\boldsymbol{r}_u^\star \leq 1 + 2\varepsilon$ for all u. $\square$

8.3.6 Computing the locations of the vertices

After approximating the radii, we embed the primal and dual vertices using the reduced angle graph H. We emphasize at this point that H is already a plane graph, so the cyclic ordering of neighbours around each vertex is known.

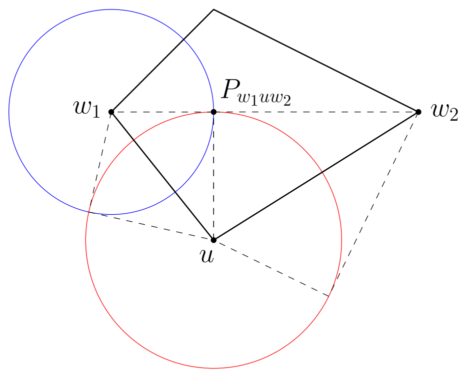


Figure 8.9: Embedding H locally around u. The two kites K_{uw_1} and K_{uw_2} are shown in dashed lines. After u and w_1 are embedded, the kite K_{uw_2} is placed with one vertex at u, and one side tangent to K_{uw_1} along the line $uP_{w_1uw_2}$. Its side lengths are r_u and r_{w_2}.

Suppose $\boldsymbol{r}$ is the ε-approximation of the radii we obtained, and $\boldsymbol{r}^\star$ is the true radii vector of the C_o-regular primal-dual representation. We define an edge uw in H to be *approximately-good* (with respect to $\boldsymbol{r}$) if it satisfies $(1-\varepsilon)/(1+\varepsilon) \cdot (2n)^{-1} \leq \boldsymbol{r}_u/\boldsymbol{r}_w \leq (1+\varepsilon)/(1-\varepsilon) \cdot 2n$, and *approximately-bad* if it does not. Note that a good edge (with respect to $\boldsymbol{r}^\star$) is an approximately-good edge.

Recall the outer cycle of G is $C_o = (s_1, s_2, s_3)$, and let the outer cycle of H be denoted by $B = (s_1, t_1, s_2, t_2, s_3, t_3)$. We may assume for both the true embedding and our appropximate embedding, that s_1 is positioned at the origin, and s_1s_2 lie on the x-axis.

The high-level idea is to embed the vertices one-by-one following a breadth-first style traversal through H using only approximately-good edges. Since the true positions of $s_1, s_2, s_3 \in C_o$ are known, and the outer cycle of H consists of good edges, t_1, t_2, t_3 are embedded first. We proceed in a breadth-first fashion with one additional traversal restriction: Suppose we visited the vertex u, and let the neighbours of u be $w_1, \ldots, w_m$ in cyclic order. We can visit a neighbour w_i only if either w_{i-1} or w_{i+1} has been visited already. This is so that when we move from u to an unvisited neighbour w_i (suppose w_{i-1} was visited), we can place the kite K_{uw_i} (see Section 8.2.2) with one point at u and one side tangent to the previous kite $K_{uw_{i-1}}$. The kite in turn determines the position of w_i in the approximate circle packing representation. First, we show all vertices in H can be reached this way.

Suppose the vertex w has neighbours $v_1, \ldots, v_m$ in cyclic order, and we visited v_i from w but cannot reach v_{i+1}, due to the fact that wv_{i+1} is an approximately-bad edge. Observe

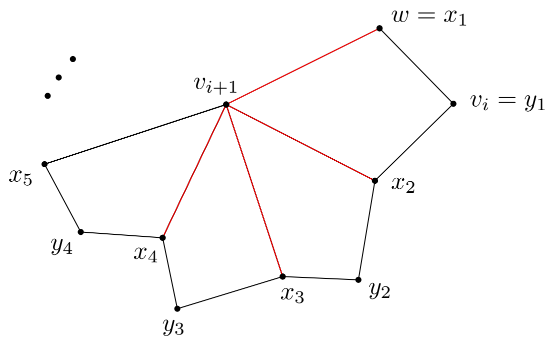


Figure 8.10: H locally around a vertex v_{i+1}. The bad edges are shown in red; they may not be all approximately-bad. However, an approximately-good path from w to v_{i+1} exists and can be followed given the traversal restrictions. (The angles in this diagram do not reflect a correct circle packing representation.)

that w, v_i, v_{i+1} are in a face together with another vertex x_2 (recall any bad edge is on the boundaries of two faces of degree four). By the arguments in Section 8.2.3 and a simple case analysis, we see that either we can reach v_{i+1} from w by going through v_i and then x_2 (implying wv_i, v_ix_2, xv_{i+1} are all approximately-good edges), or v_{i+1} is an V-vertex, $v_{i+1}x_2, v_{i+1}w$ are bad edges, and v_ix_2 is a good edge. In the latter case, let the neighbours of v_{i+1} be $w = x_1, x_2, \ldots, x_l$ in cyclic order, and suppose $j \geq 3$ is the smallest index at which $v_{i+1}x_j$ is a good edge. Note that for each $k < j$, the vertices v_{i+1}, x_k, x_{k+1} are in a face together with another vertex y_k. As v_{i+1} is a V-vertex, all the x's are F-vertices of degree three; furthermore, since $v_{i+1}x_k$ is a bad edge, we know x_ky_k and x_ky_{k+1} must be good and hence approximately-good. (See Figure 8.10 for an example with $j = 5$.) It follows that $(w = x_1, y_1, x_2, y_2, \ldots, y_{j-1}, x_j, v_{i+1})$ is an approximately-good path from w to v_{i+1}, and going along this path does not violate our traversal restrictions. So we have shown that all vertices in H can be reached.

For any edge uw, we can define the angle θ_{uw} as half the angle contributed by the kite K_{uw} at u (see Section 8.2.2). Compared to the true angle $\theta^\star_{uw}$, the error is

$$|\theta_{uw} - \theta^\star_{uw}| = \left| \arctan \frac{\boldsymbol{r}_w}{\boldsymbol{r}_u} - \arctan \frac{\boldsymbol{r}^\star_w}{\boldsymbol{r}^\star_u} \right| \leq 4\varepsilon \frac{\boldsymbol{r}^\star_w}{\boldsymbol{r}^\star_u},$$

where we used a coarse first order approximation, and the fact that $\boldsymbol{r}_u$ is an ε-approximation of $\boldsymbol{r}^\star_u$ for all vertices u. Along a good edge, this error is bounded by $8\varepsilon n$.

For an edge uw whose embedding has been approximated, we can further define α_{uw} as the angle formed by the edge uw (viewed as directed from u to w) and the x-axis, and $\alpha^\star_{uw}$ as the true angle. Observe that if $\alpha_{uw} - \alpha^\star_{uw} = \delta$, then $\alpha_{wu} - \alpha^\star_{wu} = \delta$. Furthermore, suppose uw_1, uw_2 are approximately-good edges, and that we embed w_2 right after u and w_1. Then

$$\begin{aligned}&|\alpha_{uw_2} - \alpha^\star_{uw_2}|\\ \leq &|\alpha_{uw_1} - \alpha^\star_{uw_1}| + |\theta_{uw_1} - \theta^\star_{uw_1}| + |\theta_{uw_2} - \theta^\star_{uw_2}|.\end{aligned}$$

In other words, the angle errors accumulate linearly as we traverse through H. Since only approximately-good edges are used, we conclude that for all edge uw used in the traversal, $|\alpha_{uw} - \alpha^\star_{uw}| \leq O(\varepsilon n^2)$.

Finally, we compare the approximate position $\boldsymbol{p}$ of the vertices with the true positions $\boldsymbol{p}^\star$. Suppose u is embedded in its true position, and v, w are consecutive neighbours of u such that uv, uw are approximately-good edges, and v has been embedded. In embedding w, error is introduced by α_{uw} as well as by $\boldsymbol{r}_u$ and $\boldsymbol{r}_w$. Specifically, $\boldsymbol{p}^\star_w$ is at a distance of $\sqrt{\boldsymbol{r}^{\star 2}_u + \boldsymbol{r}^{\star 2}_w}$ away from $\boldsymbol{p}^\star_u$ in the direction given by $\alpha^\star_{uw}$, while the approximate position $\boldsymbol{p}_w$ will be at a distance of $\sqrt{\boldsymbol{r}^2_u + \boldsymbol{r}^2_w}$ away in the direction α_{uw}. Basic geometry shows

$$\begin{aligned}&\|\boldsymbol{p}^\star_w - \boldsymbol{p}_w\|_2\\ &\leq |\sqrt{\boldsymbol{r}^{\star 2}_u + \boldsymbol{r}^{\star 2}_w} - \sqrt{\boldsymbol{r}^2_u + \boldsymbol{r}^2_w}| + |\alpha_{uw} - \alpha^\star_{uw}|\sqrt{\boldsymbol{r}^2_u + \boldsymbol{r}^2_w}\\ &\leq (O(\varepsilon) + |\alpha_{uw} - \alpha^\star_{uw}|)\sqrt{\boldsymbol{r}^2_u + \boldsymbol{r}^2_w}\\ &\leq O(\varepsilon n^2),\end{aligned}$$

where we use the fact that $\boldsymbol{r}_u \leq 1$ for all $u \in V(H) \setminus C_o$. The error accumulates linearly as we embed each vertex, hence $\|\boldsymbol{p}^\star_u - \boldsymbol{p}_u\|_2 \leq O(\varepsilon n^3)$ for all vertices u. It follows that if $\boldsymbol{r}$ is an $\varepsilon/(n^3 R)$-approximation of the true radii, then we can recover positions $\boldsymbol{p}$ such that $\|\boldsymbol{p}_u - \boldsymbol{p}^\star_u\|_2 \leq \varepsilon/R$ for each vertex u. Changing ε by a polynomial factor of R and n does not affect the runtime in the O-notation; this completes the proof of Theorem 8.4.

8.3.7 A remark about numerical precision

In the proof of Theorem 8.27, we only apply the algorithm from [46] on convex functions $\tilde{g}$ that are well-conditioned; specifically, $n^{-O(1)} \cdot \mathbf{I} \preccurlyeq \nabla^2 \tilde{g}(\boldsymbol{x}) \preccurlyeq n^{O(1)} \cdot \mathbf{I}$ for all x the algorithm queries. In this case, it suffices to perform all calculations in finite-precision with $O(\log(\frac{nR}{\varepsilon}))$ bits (See Section 4.3 in [46] for the discussion). Therefore, with $O(\log(\frac{nR}{\varepsilon}))$ bits calculations, we can compute the radius with $(1 \pm \varepsilon)$ multiplicative error and the location with ε/R additive error.

8.4 Computing the primal circle packing

Finally, we prove Theorem 8.6 as a corollary of Theorem 8.4.

We may assume G is 2-edge-connected, otherwise, we can simply split the graph at the cut-edge, compute the circle packing representations for the two components separately, and combine them at the edge after rescaling appropriately.

First, a planar embedding of G can be found in linear time such that all face boundaries are cycles. Next, G can be triangulated by adding a vertex in each face of degree greater than three and connecting it to all the vertices on the face boundary. Let V^+ denote the set of additional vertices, and let T denote the resulting triangulation with outer cycle C_o. Note that $|V(T)| = O(n)$, since G is planar.

We then run the primal-dual circle packing algorithm on T, which returns radius $\boldsymbol{r}_u$ and position $\boldsymbol{p}_u$ for each $u \in V(T)$, corresponding to an approximate C_o-regular representation (See Section 8.2.2). By discarding the dual graph and additional vertices V^+, we obtain an ε-approximation of the primal circle packing of G.

The total runtime is $\widetilde{O}(|V(T)| \log R_{PD}/\varepsilon)$, where $R_{PD} = \boldsymbol{r}^\star_{PD,\max}/\boldsymbol{r}^\star_{PD,\min}$ is the ratio of the maximum to minimum radius in the target primal-dual circle packing of T. Let $R = \boldsymbol{r}^\star_{\max}/\boldsymbol{r}^\star_{\min}$ be the ratio of the maximum to minimum radius in the target primal circle packing. If $R_{PD} \leq \operatorname{poly}(n)R$, then $\widetilde{O}(|V(T)| \log R_{PD}/\varepsilon) \leq \widetilde{O}(n \log R/\varepsilon)$, and we have the claimed runtime.

Note that $\boldsymbol{r}^\star_{PD,\max} = \tan(\pi/3)$ is attained by one of the vertices on the outer cycle C_o of T. By construction of T, at least two of the vertices on C_o belong to V, hence $\boldsymbol{r}^\star_{\max} = \boldsymbol{r}^\star_{PD,\max}$. It remains to show that $\boldsymbol{r}^\star_{\min}$ is polynomially close to $\boldsymbol{r}^\star_{PD,\min}$.

We will make use of the terminology and lemmas introduced in Section 8.2.3. There are two cases to consider:

1. $\boldsymbol{r}^\star_{PD,\min}$ is attained by some primal vertex $v \in V(T)$. If $v \in V$, then we are done. So we may assume $v \in V^+$. In the reduced graph H_T, since v is an V-vertex, it at least one good neighbour f by Claim 8.15; moreover, f has at most one bad neighbour by Claim 8.14, so it has another good neighbour $w \neq v$. Note that w is at distance two from v in H_T, implying that w is a neighbour of v in T, and therefore $w \in V$. Using properties of good edges, we get $\boldsymbol{r}^\star_w \leq (2n)^2 \boldsymbol{r}^\star_v$.

2. $\boldsymbol{r}^\star_{PD,\min}$ is attained by some dual vertex $f \in V(T^\star)$. We know f is an F-vertex in H_T and has at least two good neighbours by Claim 8.14; moreover, it has at least two neighbours in V, since every face in T contains at least two vertices from V on its

boundary. It follows that f has a good neighbour w where $w \in V$, so $\boldsymbol{r}^\star_w \leq 2n\boldsymbol{r}^\star_f$.

In both cases, there exists $w \in V$ such that $\boldsymbol{r}^\star_w \leq (2n)^2\boldsymbol{r}^\star_{PD,\min}$. Hence, $\boldsymbol{r}^\star_{\min} \leq (2n)^2\boldsymbol{r}^\star_{PD,\min}$, as desired.

Bibliography

[1] Deeksha Adil, Brian Bullins, Rasmus Kyng, and Sushant Sachdeva. Almost-Linear-Time Weighted ℓ_p-norm Solvers in Slightly Dense Graphs via Sparsification. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198, pages 9:1–9:15, 2021. 6

[2] Deeksha Adil, Rasmus Kyng, Richard Peng, and Sushant Sachdeva. Iterative refinement for p-norm regression. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1405–1424. SIAM, 2019. 6

[3] Deeksha Adil and Sushant Sachdeva. Faster p-norm minimizing flows, via smoothed q-norm problems. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 892–910. SIAM, 2020. 6

[4] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network Flows*. Prentice Hall, 1988. 11

[5] Takuya Akiba, Christian Sommer, and Ken-ichi Kawarabayashi. Shortest-path queries for complex networks: exploiting low tree-width outside the core. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 144–155, 2012. 3

[6] Md Jawaherul Alam, David Eppstein, Michael T Goodrich, Stephen G Kobourov, and Sergey Pupyrev. Balanced circle packings for planar graphs. In *International Symposium on Graph Drawing*, pages 125–136. Springer, 2014. 144, 146

[7] Xavier Allamigeon, Pascal Benchimol, Stéphane Gaubert, and Michael Joswig. Log-barrier interior point methods are not strongly polynomial. *SIAM Journal on Applied Algebra and Geometry*, 2(1):140–178, 2018. 6

[8] Noga Alon and Raphael Yuster. Solving linear systems through nested dissection. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 225–234. IEEE, 2010. 3

[9] Noga Alon and Raphael Yuster. Matrix sparsification and nested dissection over arbitrary fields. *Journal of the ACM (JACM)*, 60(4):1–18, 2013. 8

[10] E M Andreev. On convex polyhedra in lobačevskiĭ spaces. *Mathematics of the USSR-Sbornik*, 10(3):413–440, apr 1970. 141

[11] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987. 138

[12] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. *Journal of the ACM (JACM)*, 63(2):1–35, 2016. 138, 139

[13] Per Austrin, Toniann Pitassi, and Yu Wu. Inapproximability of treewidth, one-shot pebbling, and related layout problems. In *International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 13–24. Springer, 2012. 138

[14] Kyriakos Axiotis, Aleksander Mądry, and Adrian Vladu. Circulation control for faster minimum cost flow in unit-capacity graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 93–104. IEEE Computer Society, 2020. 6, 8, 10

[15] Kyriakos Axiotis, Aleksander Mądry, and Adrian Vladu. Faster sparse minimum cost flow by electrical flow localization. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 528–539, 2022. 11

[16] Eric Balkanski and Yaron Singer. Parallelization does not accelerate convex optimization: Adaptivity lower bounds for non-smooth convex minimization. *arXiv preprint arXiv:1808.03880*, 2018. 5

[17] Michael J Bannister, William E Devanny, David Eppstein, and Michael T Goodrich. The galois complexity of graph drawing: Why numerical solutions are ubiquitous for force-directed, spectral, and circle packing drawings. In *International Symposium on Graph Drawing*, pages 149–161. Springer, 2014. 143

[18] Itai Benjamini and Oded Schramm. Harmonic functions on planar and almost planar graphs and manifolds, via circle packings. *Inventiones mathematicae*, 126(3):565–587, 1996. 143

[19] Itai Benjamini and Oded Schramm. Recurrence of distributional limits of finite planar graphs. *Electron. J. Probab.*, 6:13 pp., 2001. 143

[20] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental SSSP and approximate min-cost flow in almost-linear time. In

62st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021. IEEE, 2021. 3

[21] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental sssp and approximate min-cost flow in almost-linear time. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1000–1008. IEEE, 2022. 11, 138

[22] Hans L Bodlaender. A tourist guide through treewidth. 1994. 9

[23] Hans L Bodlaender, Pål Grønås Drange, Markus S Dregi, Fedor V Fomin, Daniel Lokshtanov, and Michał Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM Journal on Computing*, 45(2):317–378, 2016. 138

[24] Hans L Bodlaender, John R Gilbert, Hjálmtyr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995. 62, 138

[25] Glencora Borradaile. *Exploiting Planarity for Network Flow and Connectivity Problems*. Brown University, 2008. 11

[26] Glencora Borradaile and Philip N. Klein. An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. *J. ACM*, 56(2):9:1–9:30, 2009. 10

[27] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004. 16, 17

[28] Cornelius Brand, Martin Koutecký, and Sebastian Ordyniak. Parameterized algorithms for milps with small treedepth. *arXiv preprint arXiv:1912.03501*, 2019. 9

[29] Jan van den Brand. A deterministic linear program solver in current matrix multiplication time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 259–278. SIAM, 2020. 8, 12, 115

[30] Jan van den Brand, Yin-Tat Lee, Yang P. Liu, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Maximum flow in nearly-linear time on moderately dense graphs. *Personal communication*, 2020. 1, 7

[31] Jan van den Brand, Yin Tat Lee, Yang P Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, MDPs, and $\ell 1$-regression in nearly linear time for dense instances. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 859–869, 2021. 11, 137

[32] Sebastian Brandt and Roger Wattenhofer. Approximating small balanced vertex separators in almost linear time. *Algorithmica*, 81:4070–4097, 2019. 138

[33] G. Brightwell and E. Scheinerman. Representations of planar graphs. *SIAM Journal on Discrete Mathematics*, 6(2):214–229, 1993. 141

[34] Sébastien Bubeck, Qijia Jiang, Yin-Tat Lee, Yuanzhi Li, and Aaron Sidford. Complexity of highly parallel non-smooth convex optimization. In *Advances in Neural Information Processing Systems*, pages 13900–13909, 2019. 5

[35] Erin W. Chambers, Jeff Erickson, Kyle Fox, and Amir Nayyeri. Minimum cuts in surface graphs. *SIAM J. Comput.*, 52(1):156–195, 2023. 12

[36] Erin W. Chambers, Jeff Erickson, and Amir Nayyeri. Homology flows, cohomology cuts. *SIAM J. Comput.*, 41(6):1605–1634, 2012. 10

[37] Krishnendu Chatterjee and Jakub Łącki. Faster algorithms for markov decision processes with low treewidth. In *International Conference on Computer Aided Verification*, pages 543–558. Springer, 2013. 3

[38] Shiva Chaudhuri and Christos D Zaroliagis. Shortest paths in digraphs of small treewidth. part i: Sequential algorithms. *Algorithmica*, 27(3-4):212–226, 2000. 3

[39] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 612–623. IEEE, 2022. i, 10, 11, 137

[40] Bennett Chow, Feng Luo, et al. Combinatorial ricci flows on surfaces. *Journal of Differential Geometry*, 63(1):97–129, 2003. 144

[41] Paul Christiano, Jonathan A Kelner, Aleksander Madry, Daniel A Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the forty-third Annual ACM Symposium on Theory of Computing*, pages 273–282, 2011. 6, 11

[42] Michael B Cohen, Rasmus Kyng, Gary L Miller, Jakub W Pachocki, Richard Peng, Anup B Rao, and Shen Chen Xu. Solving sdd linear systems in nearly m log 1/2 n time. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 343–352. ACM, 2014. 144

[43] Michael B. Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. In *Proceedings of the 51st Annual ACM SIGACT*

Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019., pages 938–942, 2019. iv, 1, 7

[44] Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *Journal of the ACM (JACM)*, 68(1):1–39, 2021. , i, 8, 12, 115

[45] Michael B Cohen, Aleksander Mądry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in $\widetilde{O}(m^{10/7} \log w)$ time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 752–771. SIAM, 2017. 6

[46] Michael B Cohen, Aleksander Madry, Dimitris Tsipras, and Adrian Vladu. Matrix scaling and balancing via box constrained newton's method and interior point methods. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 902–913. IEEE, 2017. 144, 145, 159, 167

[47] Yves Colin de Verdière. Un principe variationnel pour les empilements de cercles. *Inventiones mathematicae*, 104(1):655–669, 1991. 144, 156

[48] Charles R Collins and Kenneth Stephenson. A circle packing algorithm. *Computational Geometry*, 25(3):233–256, 2003. 144

[49] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms.* MIT press, 2009. 102

[50] Jana Cslovjecsek, Friedrich Eisenbrand, Christoph Hunkenschröder, Lars Rohwedder, and Robert Weismantel. Block-structured integer and linear programming in strongly polynomial and near linear time, 2020. 9

[51] 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*, pages 451–460, 2008. 3, 11

[52] George B Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity analysis of production and allocation*, 13:339–347, 1951. , 1

[53] Timothy A. Davis. *Direct Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics, jan 2006. 2, 7, 8

[54] Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. In Joachim Gudmundsson and Jyrki Katajainen, editors, *Experimental Algorithms*, pages 271–282, Cham, 2014. Springer International Publishing. 3, 62

[55] Ming Ding, Rasmus Kyng, and Peng Zhang. Two-commodity flow is equivalent to linear programming under nearly-linear time reductions. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 54:1–54:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. 12, 115

[56] Sally Dong, Yu Gao, Gramoz Goranci, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Guanghao Ye. Nested dissection meets ipms: Planar min-cost flow in nearly-linear time. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 124–153. SIAM, 2022. , iv, 10, 12, 87

[57] Sally Dong, Gramoz Goranci, Lawrence Li, Sushant Sachdeva, and Guanghao Ye. Fast algorithms for separable linear programs. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3558–3604. SIAM, 2024. , i, iv

[58] Sally Dong, Yin Tat Lee, and Guanghao Ye. A nearly-linear time algorithm for linear programs with small treewidth: A multiscale representation of robust central path. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 1784–1797. ACM, 2021. i, iv, 8, 10, 13, 16, 67, 103

[59] Sally Dong, Yin Tat Lee, and Kent Quanrud. Computing circle packing representations of planar graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2860–2875. SIAM, 2020. i

[60] Sally Dong and Guanghao Ye. Faster min-cost flow on bounded treewidth graphs. *arXiv preprint arXiv:2308.14727*, 2023. i, iv

[61] Tomasz Dubejko and Kenneth Stephenson. Circle packing: Experiments in discrete analytic function theory. *Experimental Mathematics*, 4(4):307–348, 1995. 143

[62] David Durfee, Yu Gao, Anup B Rao, and Sebastian Wild. Efficient second-order shape-constrained function fitting. In *Workshop on Algorithms and Data Structures*, pages 395–408. Springer, 2019. 5

[63] David Durfee, Rasmus Kyng, John Peebles, Anup B. Rao, and Sushant Sachdeva. Sampling random spanning trees faster than matrix multiplication. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 730–742, 2017. 88

[64] Martin Dyer, Alan Frieze, and Ravi Kannan. A random polynomial-time algorithm for

approximating the volume of convex bodies. *Journal of the ACM (JACM)*, 38(1):1–17, 1991. 58

[65] Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. An algorithmic theory of integer programming. *arXiv preprint arXiv:1904.01361*, 2019. 9

[66] David Eppstein. A möbius-invariant power diagram and its applications to soap bubbles and planar lombardi drawing. *Discrete & Computational Geometry*, 52(3):515–550, 2014. 143

[67] David Eppstein, Zvi Galil, Giuseppe F Italiano, and Thomas H Spencer. Separator based sparsification: I. planarity testing and minimum spanning trees. *journal of computer and system sciences*, 52(1):3–27, 1996. 63

[68] Greg N. Federickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16(6):1004–1022, 1987. 64

[69] Uriel Feige, MohammadTaghi Hajiaghayi, and James R Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38(2):629, 2008. 138

[70] Stefan Felsner, Alexander Igamberdiev, Philipp Kindermann, Boris Klemz, Tamara Mchedlidze, and Manfred Scheucher. Strongly monotone drawings of planar graphs. *arXiv preprint arXiv:1601.01598*, 2016. 143

[71] Stefan Felsner and Günter Rote. On primal-dual circle representations. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*, pages 8:1–8:18, 2019. 143, 144

[72] Lisa K Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics*, 13(4):505–520, 2000. 12

[73] Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, Michał Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Transactions on Algorithms (TALG)*, 14(3):1–45, 2018. 3, 9, 138

[74] Lester R Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956. 11

[75] Yu Gao, Yang P. Liu, and Richard Peng. Fully dynamic electrical flows: Sparse

maxflow faster than Goldberg-Rao. In *62st IEEE Annual Symposium on Foundations of Computer Science, FOCS2021*. IEEE, 2021. 7, 10, 11

[76] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007. 12

[77] Alan George. Nested dissection of a regular finite element mesh. *SIAM journal on numerical analysis*, 10(2):345–363, 1973. 8

[78] I. Gohberg, T. Kailath, and I. Koltracht. Efficient solution of linear systems of equations with recursive structure. *Linear Algebra and its Applications*, 80:81–113, 1986. 3

[79] Xianfeng David Gu and Shing-Tung Yau. *Computational conformal geometry*. International Press Somerville, MA, 2008. 143

[80] Yuzhou Gu and Zhao Song. A faster small treewidth SDP solver. *CoRR*, abs/2211.06033, 2022. 119

[81] Thomas Dueholm Hansen and Uri Zwick. An improved version of the random-facet pivoting rule for the simplex algorithm. In *Proceedings of the forty-seventh Annual ACM Symposium on Theory of Computing*, pages 209–218, 2015. 1

[82] Sariel Har-Peled. A simple proof of the existence of a planar separator. *CoRR*, abs/1105.0103, 2011. 143

[83] Refael Hassin. Maximum flow in (s, t) planar networks. *Information Processing Letters*, 13(3):107, 1981. 11

[84] Refael Hassin. On multicommodity flows in planar graphs. *Networks*, 14(2):225–235, 1984. 12

[85] Refael Hassin and Donald B Johnson. An $\mathrm{O}(n \log^2 n)$ algorithm for maximum flow in undirected planar networks. *SIAM Journal on Computing*, 14(3):612–624, 1985. 11

[86] Zheng-Xu He and Oded Schramm. On the convergence of circle packings to the riemann map. *Inventiones mathematicae*, 125(2):285–305, 1996. 143

[87] Monika R Henzinger, Philip Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997. 10, 11, 63

[88] Jan M Hochstein and Karsten Weihe. Maximum st-flow with k crossings in $O(k3n \log n)$

time. In *Proceedings of the eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 843–847, 2007. 10

[89] Monica K Hurdal and Ken Stephenson. Discrete conformal methods for cortical brain flattening. *Neuroimage*, 45(1):S86–S98, 2009. 143

[90] Hiroshi Imai and Kazuo Iwano. Efficient sequential and parallel algorithms for planar minimum cost flow. In *Algorithms, International Symposium SIGAL '90, Tokyo, Japan*, volume 450 of *Lecture Notes in Computer Science*, pages 21–30. Springer, 1990. 12

[91] Alon Itai. Two-commodity flow. *J. ACM*, 25(4):596–611, 1978. 12, 115

[92] Alon Itai and Yossi Shiloach. Maximum flow in planar networks. *SIAM Journal on Computing*, 8(2):135–150, 1979. 11

[93] Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011*, pages 313–322. ACM, 2011. 10, 11

[94] Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. Faster dynamic matrix inverse for faster lps. *arXiv preprint arXiv:2004.07470*, 2020. 1, 7

[95] Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. A faster algorithm for solving general lps. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 823–832, 2021. 8

[96] Donggu Kang and James Payor. Flow Rounding. *arXiv preprint arXiv:1507.08139*, 2015. 124

[97] Adam Karczmarz and Piotr Sankowski. Min-cost flow in unit-capacity planar graphs. In *27th Annual European Symposium on Algorithms, ESA 2019*, volume 144 of *LIPIcs*, pages 66:1–66:17. Schloss Dagstuhl, 2019. 10, 12

[98] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984. , 1, 8

[99] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998. 2

[100] Tarun Kathuria, Yang P. Liu, and Aaron Sidford. Unit capacity maxflow in almost $o(m^{4/3})$ time. In *61st IEEE Annual Symposium on Foundations of Computer Science,*

FOCS 2020, Durham, NC, USA, November 16-19, 2020, pages 119–130, 2020. 6, 8, 10, 11

[101] Ken-ichi Kawarabayashi and Yusuke Kobayashi. All-or-nothing multicommodity flow problem with bounded fractionality in planar graphs. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 187–196, 2013. 12

[102] Jonathan A Kelner. Spectral partitioning, eigenvalue bounds, and circle packings for graphs of bounded genus. *SIAM Journal on Computing*, 35(4):882–902, 2006. 143

[103] 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*, pages 217–226. SIAM, 2014. 6, 11, 12

[104] Jonathan A Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 911–920, 2013. 144

[105] Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980. 1, 8

[106] Valerie King, Satish Rao, and Rorbert Tarjan. A faster deterministic maximum flow algorithm. *Journal of Algorithms*, 17(3):447–474, 1994. 10, 11

[107] Paul Koebe. Kontaktprobleme der konformen abbildung. *Ber. Verh. Sächs. Akad. Leipzig, Math.-Phys. Klasse*, 88:141–164, 1936. 141

[108] Ioannis Koutis, Gary L Miller, and Richard Peng. A nearly-m log n time solver for sdd linear systems. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 590–598. IEEE, 2011. 144

[109] Ioannis Koutis, Gary L Miller, and Richard Peng. Approaching optimality for solving sdd linear systems. *SIAM Journal on Computing*, 43(1):337–354, 2014. 144

[110] 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 forty-eighth annual ACM symposium on Theory of Computing*, pages 842–850. ACM, 2016. 144

[111] Rasmus Kyng, Richard Peng, Sushant Sachdeva, and Di Wang. Flows in almost linear

time via adaptive preconditioning. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 902–913, 2019. 6

[112] Rasmus Kyng and Sushant Sachdeva. Approximate gaussian elimination for laplacians-fast, sparse, and simple. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 573–582. IEEE, 2016. 144

[113] Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 147–156. IEEE, 2013. 144

[114] 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 *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 424–433. IEEE, 2014. 137

[115] Yin Tat Lee and Aaron Sidford. Solving linear programs with sqrt(rank) linear system solves. *CoRR*, abs/1910.08033, 2019. 8, 16

[116] Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. In *Conference on Learning Theory*, pages 2140–2157. PMLR, 2019. 8

[117] Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. In *Conference on Learning Theory, COLT 2019, 25-28 June 2019, Phoenix, AZ, USA*, pages 2140–2157, 2019. 13, 16, 20

[118] Yin Tat Lee and Santosh S Vempala. The Kannan-Lovász-Simonovits conjecture. *arXiv preprint arXiv:1807.03465*, 2018. 58

[119] Yin Tat Lee and Man-Chung Yue. Universal barrier is n-self-concordant. *arXiv preprint arXiv:1809.03011*, 2018. 57, 58

[120] Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM (JACM)*, 46(6):787–832, 1999. 139

[121] Tom Leighton, Clifford Stein, Fillia Makedon, Éva Tardos, Serge Plotkin, and Spyros Tragoudas. Fast approximation algorithms for multicommodity flow problems. In *Proceedings of the twenty-third annual ACM symposium on Theory of Computing*, pages 101–111, 1991. 12

[122] Richard J Lipton, Donald J Rose, and Robert Endre Tarjan. Generalized nested dissection. *SIAM journal on numerical analysis*, 16(2):346–358, 1979. 2, 3, 8, 117

[123] Richard J Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979. 3, 62

[124] Aleksander Madry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proceedings of the forty-second ACM Symposium on Theory of Computing*, pages 121–130, 2010. 12

[125] Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 253–262. IEEE, 2013. 6, 11

[126] Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 593–602. IEEE, 2016. 6

[127] Kazuhiko Matsumoto, Takao Nishizeki, and Nobuji Saito. An efficient algorithm for finding multicommodity flows in planar networks. *SIAM Journal on Computing*, 14(2):289–302, 1985. 12

[128] Gary L Miller, Shang-Hua Teng, William Thurston, and Stephen A Vavasis. Separators for sphere-packings and nearest neighbor graphs. *Journal of the ACM (JACM)*, 44(1):1–29, 1997. 143

[129] Bojan Mohar. A polynomial time circle packing algorithm. *Discrete Mathematics*, 117(1-3):257–263, 1993. i, 141, 144

[130] Bojan Mohar. Circle packings of maps —the euclidean case. *Rendiconti del Seminario Matematico e Fisico di Milano*, 67(1):191–206, Dec 1997. 144, 149, 156

[131] Murat Mut and Tamás Terlaky. A tight iteration-complexity upper bound for the mty predictor-corrector algorithm via redundant klee-minty cubes. 2014. 6

[132] Arkadi Nemirovski. On parallel complexity of nonsmooth convex optimization. *Journal of Complexity*, 10(4):451–463, 1994. 5

[133] Yurii Nesterov. Introductory lectures on convex optimization - a basic course. In *Lecture Notes*, 1998. 55, 56

[134] Yurii Nesterov and Arkadii Nemirovskii. *Interior-point polynomial algorithms in convex programming.* SIAM, 1994. 57, 58

[135] Yurii Nesterov and Arkadi Nemirovsky. Acceleration and parallelization of the path-following interior point method for a linearly constrained convex quadratic problem. *SIAM Journal on Optimization*, 1(4):548–564, 1991. 8

[136] Haruko Okamura and P.D. Seymour. Multicommodity flows in planar graphs. *Journal of Combinatorial Theory, Series B*, 31(1):75–81, 1981. 12

[137] Gerald L Orick, Kenneth Stephenson, and Charles Collins. A linearized circle packing algorithm. *Computational Geometry*, 64:13–29, 2017. 144

[138] James Orlin. A faster strongly polynomial minimum cost flow algorithm. In *Proceedings of the Twentieth annual ACM symposium on Theory of Computing*, pages 377–387, 1988. 10

[139] James B Orlin. Max flows in $O(nm)$ time, or better. In *Proceedings of the forty-fifth annual ACM Symposium on Theory of Computing*, pages 765–774, 2013. 10

[140] Richard Peng. Approximate undirected maximum flows in $O(m \operatorname{poly}\log(n))$ time. In *Proceedings of the twenty-seventh annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1862–1867. SIAM, 2016. 11, 12

[141] Richard Peng and Daniel A Spielman. An efficient parallel solver for sdd linear systems. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 333–342. ACM, 2014. 144

[142] Richard Peng and Santosh Vempala. Solving sparse linear systems faster than matrix multiplication, 2020. 1

[143] Leon R Planken, Mathijs M de Weerdt, and Roman PJ van der Krogt. Computing all-pairs shortest paths by leveraging low treewidth. *Journal of artificial intelligence research*, 43:353–388, 2012. 3

[144] John H Reif. Minimum s-t cut of a planar undirected network in $O(n \log^2 n)$ time. *SIAM Journal on Computing*, 12(1):71–81, 1983. 11

[145] James Renegar. A polynomial-time algorithm, based on Newton's method, for linear programming. *Mathematical programming*, 40(1-3):59–93, 1988. , 8

[146] Burt Rodin and Dennis Sullivan. The convergence of circle packings to the riemann mapping. *Journal of Differential Geometry*, 26(2):349–360, 1987. 143

[147] Aaron Schild and Christian Sommer. On balanced separators in road networks. In Evripidis Bampis, editor, *Experimental Algorithms*, pages 286–297, Cham, 2015. Springer International Publishing. 3, 62

[148] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

[149] Jonah Sherman. Nearly maximum flows in nearly linear time. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 263–269, 2013. 11, 12

[150] Jonah Sherman. Area-convexity, linf regularization, and undirected multicommodity flow. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 452–460, 2017. 12

[151] Aaron Sidford and Kevin Tian. Coordinate methods for accelerating linf regression and faster approximate maximum flow. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 922–933. IEEE, 2018. 11

[152] Daniel A Spielman and Shang-Hua Teng. Disk packings and planar separators. In *Symposium on Computational Geometry*, pages 349–358, 1996. 143

[153] Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90, 2004. 11, 122

[154] Daniel A Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM Journal on Matrix Analysis and Applications*, 35(3):835–885, 2014. 144

[155] Nikhil Srivastava and Roman Vershynin. Covariance estimation for distributions with $2+\varepsilon$ moments. *The Annals of Probability*, 41(5):3081–3111, 2013. 58

[156] Kenneth Stephenson. Circle packing and discrete analytic function theory, 2002. 143

[157] Kenneth Stephenson. Circle packing: a mathematical tale. *Notices of the AMS*, 50(11):1376–1388, 2003. 143

[158] William P. Thurston. *Geometry and Topology of Three-Manifolds*, volume 1. Princeton University Press, 1980. 141

[159] Robert Tibshirani, Michael Saunders, Saharon Rosset, Ji Zhu, and Keith Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108, 2005. 120

[160] William Thomas Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, 3(1):743–767, 1963. 143

[161] Pravin M Vaidya. A new algorithm for minimizing convex functions over convex sets. In *30th Annual Symposium on Foundations of Computer Science*, pages 338–343. IEEE Computer Society, 1989.

[162] Pravin M Vaidya. A new algorithm for minimizing convex functions over convex sets. *Mathematical programming*, 73(3):291–341, 1996. 8

[163] Jan van den Brand, Yu Gao, Arun Jambulapati, Yin Tat Lee, Yang P. Liu, Richard Peng, and Aaron Sidford. Faster maxflow via improved dynamic spectral vertex sparsifiers. *CoRR*, abs/2112.00722, 2021. 7, 11

[164] Jan van den Brand, Yin-Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 919–930. IEEE, 2020. 7

[165] Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 775–788, 2020. 1, 7

[166] Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 775–788, 2020. 7

[167] Jan van Den Brand and Daniel J Zhang. Faster high accuracy multi-commodity flow from single-commodity techniques. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 493–502. IEEE, 2023. 12

[168] Nisheeth K Vishnoi. $Lx = b$ laplacian solvers and their algorithmic applications. *Foundations and Trends in Theoretical Computer Science*, 8(1–2):1–141, 2013. 122

[169] Karsten Weihe. Maximum (s,t)-flows in planar networks in $O(|v| \log |v|)$ time. *Journal of Computer and System Sciences*, 55(3):454–475, 1997. 11

[170] Stephen J Wright. *Primal-dual interior-point methods.* SIAM, 1997. 16, 19

[171] Günter M. Ziegler. Convex Polytopes: Extremal Constructions and f-Vector Shapes. *arXiv Mathematics e-prints*, 2004. 143, 144