Nested Dissection Meets IPMs: Planar Min-Cost Flow in Nearly-Linear Time

SALLY DONG, Paul G Allen School of Computer Science and Engineering, University of Washington, Seattle, United States
YU GAO, School of Computer Science, Georgia Institute of Technology, Atlanta, United States
GRAMOZ GORANCI, Faculty of Computer Science, University of Vienna, Vienna, Austria
YIN TAT LEE, Paul G Allen School of Computer Science and Engineering, University of Washington, Seattle, United States
SUSHANT SACHDEVA, Department of Computer Science, University of Toronto, Toronto, Canada
RICHARD PENG, Computer Science Department, Carnegie Mellon University, Pittsburgh, United States
GUANGHAO YE, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, United States

We present a nearly-linear time algorithm for finding a minimum-cost flow in planar graphs with polynomially-bounded integer costs and capacities. The previous fastest algorithm for this problem is based on interior point methods (IPMs) and works for general sparse graphs in $O(n^{1.5} \cdot \text{poly}(\log n))$ time [Daitch-Spielman, STOC'08].

Intuitively, $\Omega(n^{1.5})$ is a natural runtime barrier for IPM-based methods, since they require $\sqrt{n}$ iterations, each routing a possibly-dense electrical flow. To break this barrier, we develop a new implicit representation for flows based on generalized nested dissection [Lipton-Rose-Tarjan, SINUM'79] and approximate Schur complements [Kyng-Sachdeva, FOCS'16]. This implicit representation permits us to design a data structure

Sally Dong supported by a PGS-D postdoctoral fellowship from Natural Sciences and Engineering Research Council of Canada (NSERC).
Yu Gao supported by National Science Foundation (NSF) award CCF-1846218.
Part of this work was done while the Gramoz Goranci was a postdoc at University of Toronto.
Yin Tat Lee supported by NSF awards CCF-1749609, DMS-1839116, DMS-2023166, CCF-2105772, a Microsoft Research Faculty Fellowship, a Sloan Research Fellowship, and a Packard Fellowship.
Sushant Sachdeva supported by an NSERC Discovery Grant.
Richard Peng supported by NSF awards CCF-1846218 and CCF-2106444.
Guanghao Ye part of this work was done while the author was a student at University of Washington.
Authors' Contact Information: Sally Dong, Paul G Allen School of Computer Science and Engineering, University of Washington, Seattle, Washington, United States; e-mail: sallyqd@uw.edu; Yu Gao, School of Computer Science, Georgia Institute of Technology, Atlanta, Georgia, United States; e-mail: ygao380@gatech.edu; Gramoz Goranci, Faculty of Computer Science, University of Vienna, Vienna, Austria; e-mail: gramoz.goranci@gmail.com; Yin Tat Lee, Paul G Allen School of Computer Science and Engineering, University ofWashington, Seattle,Washington, United States; e-mail: yintat@uw.edu; Sushant Sachdeva, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada; e-mail: sachdeva@cs.toronto.edu; Richard Peng, Computer Science Department, Carnegie Mellon University, Pittsburgh, Pennsylvania, United States; e-mail: yangp@cs.cmu.edu; Guanghao Ye, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, United States; e-mail: ghye@mit.edu.

ACM 0004-5411/2025/07-ART27
https://doi.org/10.1145/3744639

to route an electrical flow with sparse demands in roughly $\sqrt{n}$ update time, resulting in a total runtime of $O(n \cdot \text{poly}(\log n))$.

Our results immediately extend to all families of separable graphs.

CCS Concepts: • **Theory of computation** → **Data structures design and analysis**; **Streaming, sublinear and near linear time algorithms**; **Discrete optimization**; **Network flows**; • **Mathematics of computing** → **Network flows**

Additional Key Words and Phrases: Network flow, planar graph

ACM Reference Format:
Sally Dong, Yu Gao, Gramoz Goranci, Yin Tat Lee, Sushant Sachdeva, Richard Peng, and Guanghao Ye. 2025. Nested Dissection Meets IPMs: Planar Min-Cost Flow in Nearly-Linear Time. *J. ACM* 72, 4, Article 27 (July 2025), 75 pages. https://doi.org/10.1145/3744639

1 Introduction

The minimum-cost flow problem on planar graphs is a foundational problem in combinatorial optimization studied since the 1950s. It has diverse applications including network design, VLSI layout, and computer vision. The seminal article of Ford and Fulkerson in the 1950s [24] presented an $O(n^2)$ time algorithm for the special case of max-flow on s, t-planar graphs, i.e., planar graphs with both the source and sink lying on the same face. Over the decades since, a number of nearly-linear time max-flow algorithms have been developed for special graph classes, including undirected planar graphs by Reif and Hassin-Johnson [31, 57], planar graphs by Borradaile-Klein [9], and bounded-genus graphs by Chambers-Erickson-Nayyeri [11]. However, for the more general min-cost flow problem, there is no known result specializing on planar graphs with better guarantees than on general graphs. In this article, we present the first nearly-linear time algorithm for min-cost flow on planar graphs:

Theorem 1 (Main Result). *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 $\widetilde{O}(n \log^2 M)$[1] expected time.*

Our algorithm is fairly general and uses the planarity assumption minimally. It builds on a combination of interior point methods (IPMs), approximate Schur complements, and nested dissection, with the latter being the only component that exploits planarity. Specifically, we require that for any subgraph of the input graph with k vertices, we can find an $O(\sqrt{k})$-sized balanced vertex-separator in nearly-linear time. As a result, the algorithm naturally generalizes to all graphs with small separators. We say a subset-closed family of graphs $\mathcal{G}$ is *α-separable* if there are universal constants $0 < b < 1$ and $c > 0$ such that for any $G \in \mathcal{G}$ with n vertices and m edges, G has a balanced vertex-separator S with at most cn^α vertices, and the components of $G \setminus S$ each have at most bm edges. Our algorithm generalizes as follows:

Corollary 2 (Separable Min-cost Flow). *Let $\mathcal{G}$ be an α-separable graph class, and suppose there is convex function s such that we can compute a balanced vertex-separator for any $G \in \mathcal{G}$ with m edges in $s(m)$ time. Then given connected $G \in \mathcal{G}$ with m edges, and integral demands $\boldsymbol{d}$, edge capacities $\boldsymbol{u}$, costs $\boldsymbol{c}$, all bounded by M in absolute value, there is an algorithm that computes a minimum-cost flow on G satisfying demand $\boldsymbol{d}$ in $\widetilde{O}((m + m^{1/2+\alpha}) \log^2 M + s(m))$ expected time.*

[1]Throughout the article, we use $\widetilde{O}(f(n))$ to denote $O(f(n) \log^{O(1)} f(n))$.

Table 1. Fastest Known Exact Algorithms for the Min-cost Flow Problem, Ordered by the Generality of the Result

Min-cost flow	Time bound	Reference
Strongly polytime	$O(m^2 \log n + mn \log^2 n)$	[56]
Weakly polytime	$\widetilde{O}(m^{1+o(1)} \log^2 M)$	[12]
Unit-capacity	$m^{\frac{4}{3}+o(1)} \log M$	[6]
Planar graph	$\widetilde{O}(n \log^2 M)$	**this article**
Unit-capacity planar graph	$O(n^{4/3} \log M)$	[39]
Graph with treewidth τ	$\widetilde{O}(n\tau^2 \log M)$	[20]
Outerplanar graph	$O(n \log^2 n)$	[38]
Unidirectional, bidirectional cycle	$O(n)$, $O(n \log n)$	[63]

Here, n is the number of vertices, m is the number of edges, and M is the maximum of edge capacity and cost value.

Beyond the study of structured graphs, we believe our article is of broader interest. The study of efficient optimization algorithms on geometrically structured graphs is a topic at the intersection of computational geometry, graph theory, combinatorial optimization, and scientific computing, that has had a profound impact on each of these areas. Connections between planarity testing and 3-vertex connectivity motivated the study of depth-first search algorithms [62], and using geometric structures to find faster solvers for structured linear systems provided foundations of Laplacian algorithms as well as combinatorial scientific computing [29, 51]. Several surprising insights from our nearly-linear time algorithm are:

(1) We are able to design a data structure for maintaining feasible primal-dual solutions that allows sublinear time updates—updating the flow value of K edges at once requires only $\widetilde{O}(\sqrt{nK})$ time. The IPM takes roughly $\sqrt{n}$ iterations and makes K-sparse updates roughly $\sqrt{n/K}$ times, elegantly combining to give a total runtime of $\widetilde{O}(n)$.
(2) We show that the subspace constraints on the feasible primal-dual solutions can be maintained implicitly under dynamic updates to the solutions. This circumvents the need to track the infeasibility of primal solutions (flows), which was required in previous works.

We hope our result provides both a host of new tools for designing algorithms for separable graphs, as well as insights on how to further improve such algorithms for general graphs.

1.1 Previous Work

The min-cost flow problem is well studied in both structured graphs and general graphs. Table 1 summarizes the best algorithms for different settings prior to this work.

Min-cost flow/max-flow on general graphs. Here, we focus on recent exact max-flow and min-cost flow algorithms. For an earlier history, we refer the reader to the monographs [4, 44]. For the approximate max-flow problem, we refer the reader to the recent articles [7, 13, 41, 58–60].

To understand recent progress, we view the max-flow problem as finding a unit s,t-flow with minimum ℓ_∞-norm, and the shortest path problem as finding a unit s,t-flow with minimum ℓ_1-norm. Prior to 2008, almost all max-flow algorithms reduced this ℓ_∞ problem to a sequence of ℓ_1 problems, since the latter can be solved efficiently. This changed with the celebrated work of Spielman and Teng, which showed how to find electrical flows (ℓ_2-minimizing unit s,t-flow) in nearly-linear time [61]. Since the ℓ_2-norm is closer to ℓ_∞ than ℓ_1, this gives a more powerful primitive for the max-flow problem. In 2008, Daitch and Spielman demonstrated that one could apply IPMs to reduce min-cost flow to roughly $\sqrt{m}$ electrical flow computations. This follows from the fact that IPMs take $\widetilde{O}(\sqrt{m})$ iterations and each iteration requires solving an electrical flow problem, which can now be solved in $\widetilde{O}(m)$ time due to the work of Spielman and Teng. Consequently, they obtained an algorithm with a $\widetilde{O}(m^{3/2} \log^2 M)$ runtime [17]. Since then, several algorithms have utilized electrical flows and other stronger primitives for solving max-flow and min-cost flow problems.

For graphs with unit capacities, Mądry gave a $\widetilde{O}(m^{10/7})$-time max-flow algorithm, the first that broke the 3/2-exponent barrier [52]. It was later improved and generalized to $O(m^{4/3+o(1)} \log M)$ [6] for the min-cost flow problem. Kathuria et al. [40] gave a similar runtime of $O(m^{4/3+o(1)} U^{1/3})$ where U is the max capacity. The runtime improvement comes from decreasing the number of iterations of IPM to $\widetilde{O}(m^{1/3})$ via a more powerful primitive of $\ell_2 + \ell_p$ minimizing flows [47].

For graphs with general capacities, the runtime has recently been improved to $\widetilde{O}((m + n^{3/2}) \log^2 M)$ for min-cost flow on dense graphs [67], and $\widetilde{O}(m^{\frac{3}{2}-\frac{1}{328}} \log M)$ for max-flow on sparse graphs [25]. These algorithms focus on decreasing the per-iteration cost of IPMs by dynamically maintaining electrical flows. After the preliminary version of this work was accepted to SODA 2022, [66] gave a runtime of $\widetilde{O}(m^{\frac{3}{2}-\frac{1}{58}} \log^2 M)$ for min-cost flow following the dynamic electrical flow framework. Most recently, [12] improved the runtime for min-cost flow to $\widetilde{O}(m^{1+o(1)} \log^2 M)$ by solving a sequence of approximate undirected minimum-ratio cycles, capping off the line of work.

Max-flow on planar graphs. The planar max-flow problem has an equally long history. We refer the reader to the thesis by Borradaile [8] for a detailed exposition. In the seminal work of Ford and Fulkerson that introduced the max-flow min-cut theorem [24], they also gave a max-flow algorithm for s,t-planar graphs (planar graphs where the source and sink lie on the same face). This algorithm iteratively sends flow along the top-most augmenting path. Itai and Shiloach showed how to implement each step in $O(\log n)$ time, thus giving an $O(n \log n)$ time algorithm for s,t-planar graphs [35]. In this setting, Hassin also showed that the max-flow can be computed using shortest-path distances in the planar dual in $O(n \log n)$ time [30]. Building on Hassin's work, the current best runtime is $O(n)$ by Henzinger, Klein, Rao, and Subramanian [32]. For undirected planar graphs, Reif first gave an $O(n \log^2 n)$ time algorithm for finding the max-flow value [57]. Hassin and Johnson then showed how to compute the flow in the same runtime [31]. The current best runtime is $O(n \log \log n)$ by Italiano, Nussbaum, Sankowski, and Wulff-Nilsen [36].

For general planar graphs, Weihe gave the first $O(n \log n)$ time algorithm, assuming the graph satisfies certain connectivity conditions [70]. Later, Borradaile and Klein gave an $O(n \log n)$ time algorithm for any planar graph [9].

The multiple-source multiple-sink version of max-flow is considered much harder on planar graphs. The first result of $O(n^{1.5})$ time was by Miller and Naor when sources and sinks are all on same face [54]. This was then improved to $O(n \log^3 n)$ in [10].

For generalizations of planar graphs, Chambers, Erickson and Nayyeri gave the first nearly-linear time algorithm for max-flow on graphs embedded on bounded-genus surfaces [11]. Miller and Peng gave an $\widetilde{O}(n^{6/5})$-time algorithm for approximating undirected max-flow for the class of $O(\sqrt{n})$-separable graphs [55], although this is superseded by the previously mentioned works for general graphs [41, 58].

Min-cost flow on planar graphs. Imai and Iwano gave a $O(n^{1.594} \log M)$ time algorithm for min-cost flow for the more general class of $O(\sqrt{n})$-separable graphs [34]. To the best of our knowledge, there is little else known about min-cost flow on general planar graphs. In the special case of unit capacities, [5, 49] gives an $O(n^{6/5} \log M)$ time algorithm for min-cost perfect matching in bipartite planar graphs, and Karczmarz and Sankowski gives a $O(n^{4/3} \log M)$ time algorithm for min-cost flow [39]. Currently, bounded treewidth graphs is the only graph family we know that admits min-cost flow algorithms that run in nearly-linear time [20].

1.2 Challenges

Here, we discuss some of the challenges in developing faster algorithms for the planar min-cost flow problem from a convex optimization perspective. For a discussion on challenges in designing combinatorial algorithms, we refer the reader to [43]. Prior to our result, the fastest min-cost flow algorithm for planar graphs is based on IPMs and takes $\widetilde{O}(n^{3/2}\log^2 M)$ time [17]. Intuitively, $\Omega(n^{3/2})$ is a natural runtime barrier for IPM-based methods, since they require $\Omega(\sqrt{n})$ iterations, each computing a possibly-dense electrical flow.

Challenges in improving the number of iterations. 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 [13] 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 [1–3, 47], 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 [6, 15, 40, 52, 53], 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.

Challenges in improving the cost per iteration. Recently, there has been much progress using data structures to design faster IPM algorithms for linear programs as well as flow problems on general graphs. For general linear programs, robust IPMs have been developed recently with runtimes that essentially match the matrix multiplication cost [14, 33, 64, 65, 68]. This version of IPM ensures that the ℓ_2 problem solved changes in a sparse manner from iteration to iteration. When used to design graph algorithms, the ith iteration of a robust IPM involves computing an electrical flow on some weighted graph G_i. The edge support remains unchanged between iterations, while the edge weights, representing conductance, change. Furthermore, if K_i is the number of edges with weight changes between G_i and G_{i+1}, then robust IPMs guarantee that

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

Roughly, this says on average, each edge undergoes only poly-log many weight updates 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 [68]. 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 [25, 66]. Unfortunately, this still requires $m^{1-\frac{1}{58}}$ time per iteration.

Finally, we note that [20] gives an $\widetilde{O}(n\tau^2 \log M)$-time algorithm for linear programs with τ treewidth. Their algorithm maintains the solution using an implicit representation. This implicit representation involves a $\tau \times \tau$ matrix that records the interaction between every variable within the vertex-separator set. Each step of the algorithm updates this matrix once and it is not the bottleneck for the $\widetilde{O}(n\tau^2 \log M)$-time budget. However, for planar graphs where $\tau = O(\sqrt{n})$, this $\tau \times \tau$ matrix is a dense graph on $\sqrt{n}$ vertices given by the Schur complement on the separator. Hence, updating this using their method requires $\Omega(n)$ time per step.

Our article follows the approach in [20] 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 [6, 40], resulting in super-linear time and polynomial dependence on capacities, rather than logarithmic.

1.3 Our Approach

In this section, we introduce our approach and explain how we overcome the difficulties mentioned previously. To begin, the min-cost flow problem can be reformulated into a linear program in the following primal-dual form:

$$\text{(Primal)} = \min_{\mathbf{B}^\top \boldsymbol{f}=0,\ \boldsymbol{l} \le \boldsymbol{f} \le \boldsymbol{u}} \boldsymbol{c}^\top \boldsymbol{f} \quad \text{and} \quad \text{(Dual)} = \min_{\mathbf{B}\boldsymbol{y}+\boldsymbol{s}=\boldsymbol{c}} \sum_i \min(\boldsymbol{l}_i \boldsymbol{s}_i, \boldsymbol{u}_i \boldsymbol{s}_i),$$

where $\mathbf{B} \in \mathbb{R}^{m \times n}$ is the edge-vertex incidence matrix, $\boldsymbol{f}$ is the flow and $\boldsymbol{s}$ is the slack (or adjusted cost vector). The primal is the min-cost circulation problem on a graph that is planar with two additional vertices, and the dual is a variant of the min-cut problem. Our algorithm for min-cost flow is composed of a novel application of IPM (Section 3.1) and new data structures (Section 3.4). The IPM method reduces solving a linear program to applying a sequence of $\widetilde{O}(\sqrt{m} \log M)$ projections and the data structures implement the primal and dual projection steps in roughly $\widetilde{O}(\sqrt{m} \log M)$ amortized time.

Robust IPM. We first explain the IPM briefly. To minimize $\boldsymbol{c}^\top \boldsymbol{f}$, each step of the IPM method moves the flow vector $\boldsymbol{f}$ in the direction of $-\boldsymbol{c}$. However, such $\boldsymbol{f}$ may exceed the maximum or minimum capacities. IPM incorporates these capacity constraints by routing flows slower when they are approaching their capacity bounds. This is achieved by controlling the edge weights $\boldsymbol{w}$ and direction $\boldsymbol{v}$ in each projection step. Both $\boldsymbol{w}$ and $\boldsymbol{v}$ are roughly chosen from some explicit entry-wise formula of $\boldsymbol{f}$ and $\boldsymbol{s}$, namely, $\boldsymbol{w}_i = \psi_1(\boldsymbol{f}_i, \boldsymbol{s}_i)$ and $\boldsymbol{v}_i = \psi_2(\boldsymbol{f}_i, \boldsymbol{s}_i)$. Hence, the main bottleneck is to implement the projection step. For the min-cost flow problem, this projection step corresponds to an electrical flow computation.

It has recently been observed by [14] that there is a lot of freedom in choosing the weights $\boldsymbol{w}$ and the direction $\boldsymbol{v}$. Specifically, instead of using $\boldsymbol{f}, \boldsymbol{s}$ to compute $\boldsymbol{w}$ and $\boldsymbol{v}$, it is possible to use entry-wise approximations $\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}$ By updating an entry of $\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}$ only when the corresponding entry of $\boldsymbol{f}, \boldsymbol{s}$ change significantly, we can ensure $\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}$ has mostly sparse updates, which in turn ensures that $\boldsymbol{w}, \boldsymbol{v}$ change sparsely and, therefore, permits low-rank updates to $\mathbf{P}_{\boldsymbol{w}}$.

We refer to IPMs that use approximations $\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}$ as *robust IPMs*, which were a crucial breakthrough of [14]. In this article, we apply the version given in [20] in a black-box manner. In Section 3.1, we state the IPM in full detail. The key challenge is implementing each step in roughly $\widetilde{O}(\sqrt{m})$ time.

Separators and nested dissection. Our data structures rely on the separability property of the input graph, which dates back to the nested dissection algorithms for solving planar linear systems

[27, 51]. By recursively partitioning the graph into edge-disjoint subgraphs using balanced vertex separators, we can construct a hierarchical decomposition of a planar graph G which is called a *separator tree* [23]. This is a binary search tree over the edges in G, where each node in the separator tree represents a *region* of G. In planar graphs, for a region H with $|V(H)|$ vertices, an $O(\sqrt{|V(H)|})$-sized vertex separator suffices to partition it into two balanced sub-regions which are represented by the two children of H in the separator tree. The two sub-regions partition the edges in H and share only vertices in the separator. The *boundary* of a region H is the set of vertices in H that are adjacent to vertices outside H in G; these vertices appear in the separators of the ancestor nodes of H. Any two regions can only share vertices on their boundaries, unless one of them is an ancestor of the other.

Nested dissection algorithms [27, 51] essentially replace each region by a graph supported only its boundary vertices in a bottom-up manner. For planar linear systems, solving the dense $\sqrt{n} \times \sqrt{n}$ submatrix corresponding to the top level vertex-separator leads to a runtime of $n^{\omega/2}$ where ω is the matrix multiplication exponent. For other problems such as shortest path, this primitive involving dense graphs can be further accelerated using additional properties of distance matrices [23].

Technique 1: Approximate nested dissection and lazy propagation. Our representation of the Laplacian inverse, and in turn the projection matrix, hinges upon a sparsified version of the nested dissection representation. That is, instead of a dense inverse involving all pairs of boundary vertices, we maintain a sparse approximation. This sparsified nested dissection has been used in the approximate undirected planar flow algorithm from [55]. However, that work predated (and in some sense motivated) subsequent works on nearly-linear time approximations of Schur complements on general graphs [45, 46, 48]. Re-incorporating these sparsified algorithms gives runtime dependencies that are nearly-linear, instead of quadratic, in separator sizes, with an overall error that is acceptable to the robust IPM framework.

By maintaining objects with size nearly equal to the separator size in each node of the separator tree, we can support updating a single edge or a batch of edges in the graph efficiently. Our data structures for maintaining the approximate Schur complements and the slack and flow projection matrices all utilize this idea For example, to maintain the Schur complement of a region H onto its boundary, we maintain (1) Schur complements of its children onto their boundaries recursively, and (2) Schur complement of the children's boundaries onto the boundary H. Thus, to update an edge, the path in the separator tree from the leaf node containing the edge to the root is visited. To update multiple edges in a batch, each node in the union of the tree paths is visited. The runtime is nearly linear in the total number of boundary vertices of all nodes in the union. For K edges being updated, the runtime is bounded by $\widetilde{O}(\sqrt{mK})$. Step i of our IPM algorithm takes $\widetilde{O}(\sqrt{mK_i})$ time, where K_i is the number of entries changed in $\mathbf{W}$ and $\boldsymbol{v}$ in the step. Such a recursive approximate Schur complement structure was used in [28], where the authors achieved a runtime of $\widetilde{O}(\sqrt{m}K_i)$.

Technique 2: Batching the changes. It is known that over t iterations of an IPM, the number of coordinate updates (by more than a constant factor) in $\boldsymbol{w}$ and $\boldsymbol{v}$ is bounded by $O(t^2)$ This directly gives $\sum_{i=1}^{\widetilde{O}(\sqrt{m})} K_i = m$ and thus, a total runtime of $\sqrt{m}(\sum_{i=1}^{\widetilde{O}(\sqrt{m})} \sqrt{K_i}) = \widetilde{O}(m^{1.25})$. In order to obtain a nearly-linear runtime, the robust IPM carefully batches the updates in different steps. In the i-th step, if the change in an edge variable has exceeded some fixed threshold compared to its value in the $(i-2^{\ell})$-th step for some $\ell \leq \ell_i$, we adjust its approximation. Here, ℓ_i is the number of trailing zeros in the binary representation of i, i.e., 2^{ℓ_i} is the largest power of 2 that divides i. This ensures that K_i, the number of coordinate changes at step i is bounded by $\widetilde{O}(2^{2\ell_i})$. Since each value of ℓ_i

arises once every 2^{ℓ_i} steps, we can show that the sum of square roots of the number of changes over all steps is bounded by $\widetilde{O}(m)$, i.e., $\sum_{i=1}^{\widetilde{O}(\sqrt{m})} \sqrt{K_i} = \widetilde{O}(\sqrt{m})$. Combined with the runtime of the data structures, we obtain an $\widetilde{O}(m)$ overall runtime.

Technique 3: Maintaining feasibility via two projections. A major difficulty in the IPM is maintaining a flow vector $\boldsymbol{f}$ that satisfies the demands exactly and a slack vector $\boldsymbol{s}$ that can be expressed as $\boldsymbol{s} = \boldsymbol{c} - \mathbf{B}\boldsymbol{y}$. If we simply project $\boldsymbol{v}$ approximately in each step, the flow we send is not exactly a circulation. Traditionally, this can be fixed by computing the excess demand each step and sending flow to fix this demand. Since our edge capacities can be polynomially large, this step can take $\Omega(m)$ time To overcome this feasibility problem, we note that *distinct* projection operators $\mathbf{P}_{\boldsymbol{w}}$ can be used in IPMs for $\boldsymbol{f}$ and $\boldsymbol{s}$ as long as each projection is close to the true projection and that the step satisfies $\mathbf{B}^\top \Delta \boldsymbol{f} = \mathbf{0}$ and $\mathbf{B}\Delta \boldsymbol{y} + \Delta \boldsymbol{s} = \mathbf{0}$ for some $\Delta \boldsymbol{y}$.

This two-operator scheme is essential to our improvement since one can prove that any projection that gives feasible steps for $\boldsymbol{f}$ and $\boldsymbol{s}$ simultaneously must be the exact electrical projection, which takes linear time to compute.

2 Preliminaries

We assume all matrices and vectors in an expression have matching dimensions. That is, we will trivially pad matrices and vectors with zeros when necessary. This slight abuse of notation is unavoidable as we need to operate heavily on submatrices and subvectors in the technical sections.

Computational model. Our algorithm uses the realRAM model, where arithmetic operations are performed with real numbers in $O(1)$ time Ghadiri, Peng, and Vempala [26] showed that the arithmetic operations in IPM are stable and do not require more than log bits of precision in the fixed precision model. Kelner et al. [42] showed that symmetrical diagonally dominant (SDD) solvers similarly do not require higher bit precision. Perhaps conveniently, our algorithm makes extensive use of approximations, and therefore, can tolerate additional error coming from fixed precision; as a result, we would only incur an additional $\log M$ factor in the runtime under the fixed precision model.

General notations. An event holds with high probability if it holds with probability at least $1 - n^c$ for arbitrarily large constant c. The choice of c affects the hidden constant factors in the runtime.

We use boldface lowercase variables to denote vectors, and boldface uppercase variables to denote matrices. We use $\|\boldsymbol{v}\|_2$ to denote the 2-norm of vector $\boldsymbol{v}$ and $\|\boldsymbol{v}\|_{\mathbf{M}}$ to denote $\boldsymbol{v}^\top \mathbf{M}\boldsymbol{v}$. A scalar function applied to a vector means the function is applied entry-wise, for example, $\boldsymbol{v} + 1$ or $\cosh \boldsymbol{v}$. We use $\mathrm{nnz}(\boldsymbol{v})$ to denote the number of nonzero entries in $\boldsymbol{v}$ We use $\mathbf{0}$ for all-zero vectors and matrices where dimensions are determined by context, and $\mathbf{I}$ for the identity matrix.

For any vector $\boldsymbol{x} \in \mathbb{R}^S$, we use $\boldsymbol{x}|_C$ to denote $\boldsymbol{x}$ restricted to $C \subseteq S$; *more specifically, $\boldsymbol{x}|_C$ has length $|S|$, where $\boldsymbol{x}_i = 0$ for all $i \notin C$.* For any matrix $\mathbf{M} \in \mathbb{R}^{A \times B}$, we use $\mathbf{M}_{C,D}$ to denote the sub-matrix of $\mathbf{M}$ supported on $C \times D$ where $C \subseteq A$ and $D \subseteq B$.

For two positive semidefinite matrices $\mathbf{L}_1$ and $\mathbf{L}_2$, we write $\mathbf{L}_1 \approx_t \mathbf{L}_2$ if $e^{-t}\mathbf{L}_1 \preccurlyeq \mathbf{L}_2 \preccurlyeq e^t \mathbf{L}_1$, where $\mathbf{A} \preccurlyeq \mathbf{B}$ means $\mathbf{B} - \mathbf{A}$ is positive semidefinite. We define $\approx_t$ analogously for scalars, that is, $x \approx_t y$ if $e^{-t}x \leq y \leq e^t x$.

Trees. For a rooted tree $\mathcal{T}$, we write $H \in \mathcal{T}$ to mean H is a node in $\mathcal{T}$. We use *node* instead of *vertex* to distinguish between the contexts of tree vs. graph. We write $\mathcal{T}_H$ to mean the complete subtree of $\mathcal{T}$ rooted at 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 $H \neq A$.

The *level* of a node in a tree is defined so that leaf nodes have level 0, and the root has level η, where η is the height of the tree. For interior nodes, the level is the length of the longest path from the node to a leaf. By this definition, note that the level of a node and its child can differ by more than 1.

IPM data structures. When we discuss the data structures in the context of the IPM, step 0 means the initialization step. For $k > 0$, step k means the kth iteration of the while-loop in Algorithm 1; that is, it is the kth time we update the current solutions. For any vector or matrix $\boldsymbol{x}$ used in the IPM, we use $\boldsymbol{x}^{(k)}$ to denote the value of $\boldsymbol{x}$ at the end of the kth step.

In all dynamic data structure functions, we assume inputs are given by the set of changed coordinates and their values *compared to the previous step* Similarly, we output a vector by the set of changed coordinates and their values compared to the previous output This can be implemented using lists instead of full-dimensional vectors.

Laplacian matrix. We use $\mathbf{L} = \mathbf{B}^\top \mathbf{W} \mathbf{B}$ to denote the Laplacian matrix of a graph G with edge-vertex incidence matrix $\mathbf{B}$ and with non-negative edge weights $\mathbf{W}$ as a diagonal matrix For a subgraph $H \subseteq G$, we use $\mathbf{L}[H]$ to denote the Laplacian on H, and $\mathbf{B}[H]$ to denote the incidence matrix of H. $\mathbf{M}^\dagger$ denotes the unique Moore–Penrose pseudo-inverse of $\mathbf{M}$.

When we write $\mathbf{L}^\dagger \boldsymbol{x}$ for some Laplacian $\mathbf{L}$ and vector $\boldsymbol{x}$, we imply the use of an SDD-solver rather than computing the pseudo-inverse:

Theorem 3 ([61]). *There is a randomized algorithm which is an ε-approximate SDD-system solver. Given an 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*

$$\left\| \boldsymbol{x} - \mathbf{L}^\dagger \boldsymbol{d} \right\|_{\mathbf{L}} \leq \varepsilon \left\| \mathbf{L}^\dagger \boldsymbol{d} \right\|_{\mathbf{L}}$$

in $O(m \cdot \mathrm{poly}(\log \log n) \log(1/(\varepsilon \lambda_2(\mathbf{L}))))$ 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 $\mathbf{Z} \approx_\varepsilon \mathbf{L}^\dagger$, and has the same image as $\mathbf{L}^\dagger$ [69, Section 3.4 and Theorem 9.2].

3 Overview

In this overview, we give formal theorem statements for the various components of our min-cost flow algorithm, along with the proof of the main result. We provide high-level explanations using a simplified setup. Due to the large amount of notation introduced at different points, this section should be read in its entirety sequentially. Since the completion of this manuscript, a simplified version of our data structures has appeared in [18, 19].

The main components of our algorithm are: the primal-dual IPM from [20] (Section 3.1); a standard reduction of planar min-cost flow into the form according to IPM (Section 3.2); a data structure to maintain a collection of Schur complements via nested dissection of the graph (Section 3.3); data structures to maintain the primal-dual solutions $\boldsymbol{f}, \boldsymbol{s}$ implicitly, notably using an abstract tree operator (Section 3.4); a sketching-based data structure to maintain the primal-dual approximations $\overline{\boldsymbol{f}}$ and $\overline{\boldsymbol{s}}$ needed in the IPM (Section 3.5); and finally, the definition of the tree operators for primal and dual solutions corresponding to the IPM projection matrices onto their respective feasible subspaces (Section 3.6) These are combined in the overall proof (Section 3.7).

3.1 Robust IPM

In this subsection, we explain the robust IPM developed in [20], which is a refinement of the methods in [14, 64]. Although there are many other robust IPMs, we simply refer to this method as IPM.

ALGORITHM 1: Robust Interior Point Method from [20]

```
1:  procedure SolveLP($\mathbf{B} \in \mathbb{R}^{m \times n}, \boldsymbol{b} \in \mathbb{R}^n, \boldsymbol{c} \in \mathbb{R}^m, \boldsymbol{l} \in \mathbb{R}^m, \boldsymbol{u} \in \mathbb{R}^m, \epsilon$)
2:      Let $L \stackrel{\text{def}}{=} \|\boldsymbol{c}\|_2, R \stackrel{\text{def}}{=} \|\boldsymbol{u} - \boldsymbol{l}\|_2$ and $\phi_i(x) \stackrel{\text{def}}{=} -\log(\boldsymbol{u}_i - x) - \log(x - \boldsymbol{l}_i)$

    ▹ Modify the linear program and obtain initial $(\boldsymbol{f}', \boldsymbol{s}')$
3:      $t \leftarrow 2^{21} m^5 \cdot \frac{LR}{128} \cdot \frac{R}{r}$
4:      Compute $\boldsymbol{f}_c \leftarrow \arg\min_{\boldsymbol{l} \le \boldsymbol{f} \le \boldsymbol{u}} \boldsymbol{c}^\top \boldsymbol{f} + t\phi(\boldsymbol{f})$ and $\boldsymbol{f}_\circ \leftarrow \arg\min_{\mathbf{B}^\top \boldsymbol{f} = \boldsymbol{b}} \|\boldsymbol{f} - \boldsymbol{f}_c\|_2$
5:      Define the new matrix $\mathbf{B}^{\text{new}} \stackrel{\text{def}}{=} [\mathbf{B}; \mathbf{B}; -\mathbf{B}]$, the new barrier
            $$\phi_i^{\text{new}}(x) \stackrel{\text{def}}{=} \begin{cases} \phi_i(x) & \text{if } i \in [m], \\ -\log x & \text{else.} \end{cases}$$
6:      $\boldsymbol{f}' \leftarrow (\boldsymbol{f}_c, 3R + \boldsymbol{f}_\circ - \boldsymbol{f}_c, 3R)$
7:      $\boldsymbol{s}' \leftarrow (-t\nabla\phi(\boldsymbol{f}_c), \frac{t}{3R + \boldsymbol{f}_\circ - \boldsymbol{f}_c}, \frac{t}{3R})$

    ▹ Find initial $(\boldsymbol{f}, \boldsymbol{s})$ for the original linear program
8:      $((\boldsymbol{f}^{(1)}, \boldsymbol{f}^{(2)}, \boldsymbol{f}^{(3)}), (\boldsymbol{s}^{(1)}, \boldsymbol{s}^{(2)}, \boldsymbol{s}^{(3)})) \leftarrow$ IPMRun($\mathbf{B}^{\text{new}}, \phi^{\text{new}}, \boldsymbol{f}', \boldsymbol{s}', t, LR$)
9:      $(\boldsymbol{f}, \boldsymbol{s}) \leftarrow (\boldsymbol{f}^{(1)} + \boldsymbol{f}^{(2)} - \boldsymbol{f}^{(3)}, \boldsymbol{s}^{(1)})$

    ▹ Optimize the original linear program
10:     $(\boldsymbol{f}, \boldsymbol{s}) \leftarrow$ IPMRun($\mathbf{B}, \phi, \boldsymbol{f}, \boldsymbol{s}, LR, \frac{\epsilon}{4m}$)
11:     return $\boldsymbol{f}$
12: end procedure

13: procedure IPMRun($\mathbf{B} \in \mathbb{R}^{m \times n}$, barrier function $\phi, \boldsymbol{f} \in \mathbb{R}^m, \boldsymbol{s} \in \mathbb{R}^m, t_{\text{start}}, t_{\text{end}}$)
14:     Let $\alpha \stackrel{\text{def}}{=} \frac{1}{2^{20}\lambda}$ and $\lambda \stackrel{\text{def}}{=} 64\log(256m^2)$
15:     $t \leftarrow t_{\text{start}}, \overline{\boldsymbol{f}} \leftarrow \boldsymbol{f}, \overline{\boldsymbol{s}} \leftarrow \boldsymbol{s}, \bar{t} \leftarrow t$
16:     while $t \ge t_{\text{end}}$ do
17:         $t \leftarrow (1 - \frac{\alpha}{\sqrt{m}})t$
18:         Update $h = -\alpha / \|\cosh(\lambda\gamma^{\bar{t}}(\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}))\|_2$ where $\gamma$ is defined in Equation (3.2)
19:         Update the diagonal weight matrix $\mathbf{W} = \text{diag}(\boldsymbol{w}) \stackrel{\text{def}}{=} (\nabla^2\phi(\overline{\boldsymbol{f}}))^{-1}$
20:         Update the direction $\boldsymbol{v}$ where $\boldsymbol{v}_i \stackrel{\text{def}}{=} \sinh(\lambda\gamma^{\bar{t}}(\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}})_i)$
21:         Pick $\boldsymbol{v}^{\parallel}$ and $\boldsymbol{v}^{\perp}$ such that $\mathbf{W}^{-1/2}\boldsymbol{v}^{\parallel} \in \text{Range}(\mathbf{B})$, $\mathbf{B}^\top \mathbf{W}^{1/2} \boldsymbol{v}^{\perp} = \mathbf{0}$ and
                $$\|\boldsymbol{v}^{\parallel} - \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}\|_2 \le \alpha\|\boldsymbol{v}\|_2,$$
                $$\|\boldsymbol{v}^{\perp} - (\mathbf{I} - \mathbf{P}_{\boldsymbol{w}})\boldsymbol{v}\|_2 \le \alpha\|\boldsymbol{v}\|_2$$
                                                    ▹ $\mathbf{P}_{\boldsymbol{w}} \stackrel{\text{def}}{=} \mathbf{W}^{1/2}\mathbf{B}(\mathbf{B}^\top\mathbf{W}\mathbf{B})^{\dagger}\mathbf{B}^\top\mathbf{W}^{1/2}$
22:         $\boldsymbol{f} \leftarrow \boldsymbol{f} + h\mathbf{W}^{1/2}\boldsymbol{v}^{\perp}, \boldsymbol{s} \leftarrow \boldsymbol{s} + \bar{t}h\mathbf{W}^{-1/2}\boldsymbol{v}^{\parallel}$
23:         Update $\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}$ so that $\|\mathbf{W}^{-1/2}(\overline{\boldsymbol{f}} - \boldsymbol{f})\|_\infty \le \alpha$ and $\|\mathbf{W}^{1/2}(\overline{\boldsymbol{s}} - \boldsymbol{s})\|_\infty \le \bar{t}\alpha$
24:         Update $\bar{t} \leftarrow t$ if $|\bar{t} - t| \ge \alpha\bar{t}$
25:     end while
26:     return $(\boldsymbol{f}, \boldsymbol{s})$
27: end procedure
```

Consider a linear program of the form[2]

$$\min_{f \in \mathcal{F}} \boldsymbol{c}^\top \boldsymbol{f} \quad \text{where} \quad \mathcal{F} \stackrel{\text{def}}{=} \{\mathbf{B}^\top \boldsymbol{f} = \boldsymbol{b},\ \boldsymbol{l} \le \boldsymbol{f} \le \boldsymbol{u}\} \tag{3.1}$$

for some matrix $\mathbf{B} \in \mathbb{R}^{m \times n}$. The dual space is $\mathcal{S} \stackrel{\text{def}}{=} \{\boldsymbol{s} : \mathbf{B}\boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c} \text{ for some } \boldsymbol{y}\}$. IPM follows the primal-dual central path from an interior point ($t \gg 0$) of $\mathcal{F} \times \mathcal{S}$ to the optimal solution ($t = 0$). The primal component of the path is given by

$$\boldsymbol{f}^\star(t) \stackrel{\text{def}}{=} \arg\min_{f \in \mathcal{F}} \boldsymbol{c}^\top \boldsymbol{f} - t\phi(\boldsymbol{f}) \quad \text{where } \phi(\boldsymbol{f}) \stackrel{\text{def}}{=} -\sum_i \log(\boldsymbol{f}_i - \boldsymbol{l}_i) - \sum_i \log(\boldsymbol{u}_i - \boldsymbol{f}_i),$$

where the barrier function ϕ controls how close the flow $\boldsymbol{f}$ can be to the capacity constraints $\boldsymbol{u}$ and $\boldsymbol{l}$. The optimality condition for $\boldsymbol{f}^\star(t)$ is given by

$$\begin{aligned} \mu^t(\boldsymbol{f}, \boldsymbol{s}) &\stackrel{\text{def}}{=} \boldsymbol{s}/t + \nabla\phi(\boldsymbol{f}) = \boldsymbol{0} \\ (\boldsymbol{f}, \boldsymbol{s}) &\in \mathcal{F} \times \mathcal{S}, \end{aligned} \tag{3.2}$$

where $\mu^t(\boldsymbol{f}, \boldsymbol{s})$ measures how close $\boldsymbol{f}$ is to the minimizer $\boldsymbol{f}^\star(t)$.

Following the central path exactly is expensive. Instead, IPM maintains feasible primal and dual solution $(\boldsymbol{f}, \boldsymbol{s}) \in \mathcal{F} \times \mathcal{S}$ over discretized t, and ensures that after each step (decrease in value of t), $\boldsymbol{f}$ is an approximate minimizer. Specifically, $\boldsymbol{f}, \boldsymbol{s}$ satisfy

$$\|\gamma^t(\boldsymbol{f}, \boldsymbol{s})\|_\infty \le \frac{1}{C \log m} \text{ where } \gamma^t(\boldsymbol{f}, \boldsymbol{s})_i = \frac{\mu^t(\boldsymbol{f}, \boldsymbol{s})_i}{(\nabla^2\phi(\boldsymbol{f}))_{ii}^{1/2}}, \tag{3.3}$$

for some universal constant C. The normalization term $(\nabla^2\phi)_{ii}^{1/2}$ makes the centrality measure $\|\gamma^t(\boldsymbol{f}, \boldsymbol{s})\|_\infty$ scale-invariant in $\boldsymbol{l}$ and $\boldsymbol{u}$.

The procedure IPMRun in Algorithm 1 takes as input a point close to the central path $(\boldsymbol{f}(t_{\text{start}}), \boldsymbol{s}(t_{\text{start}}))$, and outputs another point on the central path $(\boldsymbol{f}(t_{\text{end}}), \boldsymbol{s}(t_{\text{end}}))$. Each step of the procedure decreases t by a multiplicative factor of $(1 - \frac{1}{\sqrt{m}\log m})$ and moves $(\boldsymbol{f}, \boldsymbol{s})$ within $\mathcal{F} \times \mathcal{S}$ such that $\boldsymbol{s}/t + \nabla\phi(\boldsymbol{f})$ is smaller for the current t. [20] proved that even if each step is computed approximately, IPMRun still outputs a point close to $(\boldsymbol{f}(t_{\text{end}}), \boldsymbol{s}(t_{\text{end}}))$ using $\widetilde{O}(\sqrt{m}\log(t_{\text{end}}/t_{\text{start}}))$ steps.

SolveLP calls IPMRun twice. The first call to IPMRun finds a feasible point by following the central path of the modified linear program

$$\min_{\substack{\mathbf{B}^\top(\boldsymbol{f}^{(1)} + \boldsymbol{f}^{(2)} - \boldsymbol{f}^{(3)}) = \boldsymbol{b} \\ \boldsymbol{l} \le \boldsymbol{f}^{(1)} \le \boldsymbol{u},\ \boldsymbol{f}^{(2)} \ge 0,\ \boldsymbol{f}^{(3)} \ge 0}} \quad \boldsymbol{c}^{(1)\top}\boldsymbol{f}^{(1)} + \boldsymbol{c}^{(2)\top}\boldsymbol{f}^{(3)} + \boldsymbol{c}^{(2)\top}\boldsymbol{f}^{(3)},$$

where $\boldsymbol{c}^{(1)} \stackrel{\text{def}}{=} \boldsymbol{c}$, and $\boldsymbol{c}^{(2)}, \boldsymbol{c}^{(3)}$ are some positive large vectors. The above modified linear program is chosen so that we know an explicit point on its central path, and any approximate minimizer to this new linear program gives an approximate central path point for the original problem. In the first call to IPMRun, each edge e of the original input graph G becomes three copies of the edge with flow value $\boldsymbol{f}_e^{(1)}, \boldsymbol{f}_e^{(2)}, \boldsymbol{f}_e^{(3)}$, however, edge duplication does not affect planarity for our use case. The second call to IPMRun finds an approximate solution by following the central path of the original linear program.

We note crucially that IPMRun requires access to $(\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}})$ but not $(\boldsymbol{f}, \boldsymbol{s})$ during the main while-loop. Hence, $(\boldsymbol{f}, \boldsymbol{s})$ can be implicitly maintained via any data structure. The complete solutions $(\boldsymbol{f}, \boldsymbol{s})$ only need to be produced at the very end.

[2]Although the min-cost flow problem can be written as a one-sided linear program, it is more convenient for the linear program solver to have both sides. Everything in this section works for general linear programs and hence we will not use the fact $m = O(n)$ in this subsection.

Theorem 4. *Consider the linear program*

$$\min_{\mathbf{B}^\top \boldsymbol{f}=\boldsymbol{b},\ \boldsymbol{l}\le \boldsymbol{f}\le \boldsymbol{u}} \boldsymbol{c}^\top \boldsymbol{f}$$

with $\mathbf{B} \in \mathbb{R}^{m\times n}$. *We are given a scalar* $r > 0$ *such that* there exists *some interior point* $\boldsymbol{f}_\circ$ *satisfying* $\mathbf{B}^\top \boldsymbol{f}_\circ = \boldsymbol{b}$ *and* $\boldsymbol{l} + r \le \boldsymbol{f}_\circ \le \boldsymbol{u} - r$. *Let* $L = \|\boldsymbol{c}\|_2$ *and* $R = \|\boldsymbol{u} - \boldsymbol{l}\|_2$. *For any* $0 < \epsilon \le 1/2$, *SolveLP in Algorithm 1 finds* $\boldsymbol{f}$ *such that* $\mathbf{B}^\top \boldsymbol{f} = \boldsymbol{b}$, $\boldsymbol{l} \le \boldsymbol{f} \le \boldsymbol{u}$ *and*

$$\boldsymbol{c}^\top \boldsymbol{f} \le \min_{\mathbf{B}^\top \boldsymbol{f}=\boldsymbol{b},\ \boldsymbol{l}\le \boldsymbol{f}\le \boldsymbol{u}} \boldsymbol{c}^\top \boldsymbol{f} + \epsilon L R.$$

Furthermore, the procedure IPMRun has the following properties:

- *The while-loop runs for* $O(\sqrt{m}\log m \log(\frac{mR}{\epsilon r}))$ *many iterations (steps), and* $\bar{t}$ *is only updated* $O(\log m \log(\frac{mR}{\epsilon r}))$ *times.*
- $\boldsymbol{w}_i, \boldsymbol{v}_i$ *are updated at a step only if* $\overline{\boldsymbol{f}}_i$ *or* $\overline{\boldsymbol{s}}_i$ *changed in the previous step.*
- *In each step,* $h\|\boldsymbol{v}\|_2 = O(\frac{1}{\log m})$.
- *Line 18 to Line 20 takes* $O(K)$ *time in total, where* K *is the total number of entry changes in* $\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}$ *throughout IPMRun.*

Proof. The number of steps follows from Theorem A.1 in [21], with the parameter $w_i = \nu_i = 1$ for all i The number of coordinate changes in $\boldsymbol{w}, \boldsymbol{v}$ and the runtime of Line 18 to Line 20 follows directly from the entry-wise formula of $\mu^t(\boldsymbol{f}, \boldsymbol{s})$ and $\gamma^t(\boldsymbol{f}, \boldsymbol{s})$ The bound on $h\|\boldsymbol{v}\|_2$ follows from

$$h\|\boldsymbol{v}\|_2 \le \alpha \frac{\|\sinh(\lambda\gamma^{\bar{t}}(\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}))\|_2}{\|\cosh(\lambda\gamma^{\bar{t}}(\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}))\|_2} \le \alpha = O\left(\frac{1}{\log m}\right). \qquad \square$$

In working with $\mathbf{P}_{\boldsymbol{w}}$, we apply ideas from nested dissection and approximate Schur complements to the matrix $\mathbf{L}$.

3.2 Problem Reduction

In this subsection, we show how to write down the planar min-cost flow problem as a linear program of the form Equation (3.1) and produce an optimal flow using the solution returned by the IPM (Algorithm 1). Suppose the input is as given in Theorem 1.

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 $t \longrightarrow s$ 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 Theorem 4 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 SolveLP (Algorithm 1); we call the new graph the IPM input graph.

Now, we bound the parameters L, R, r in Theorem 4. 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 $r \geq \frac{1}{4m}$.

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 4 shows that we can find a fractional circulation $\boldsymbol{f}'$ with flow value F' such that $(\boldsymbol{c}^{\text{new}})^\top \boldsymbol{f}' \leq \text{OPT}' + \frac{1}{2}$ by setting $\epsilon = \frac{1}{CM^2m^2}$ for some large constant C in Algorithm 1 F' is smaller F, but $F - F' \leq \frac{1}{2nM}$ because otherwise sending extra k units of fractional flow from s to t would give extra negative cost $\leq -knM$, 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 [37]. 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.

3.3 Nested Dissection and Approximate Schur Complements

A key idea in our work involves the manipulation of projection matrices required for the IPM. As defined in Algorithm 1, the true projection matrix $\mathbf{P}_{\boldsymbol{w}}$ is

$$\mathbf{P}_{\boldsymbol{w}} \stackrel{\text{def}}{=} \mathbf{W}^{1/2}\mathbf{B}(\mathbf{B}^\top\mathbf{W}\mathbf{B})^\dagger\mathbf{B}^\top\mathbf{W}^{1/2}.$$

We let $\mathbf{L}$ denote the weighted Laplacian where $\mathbf{L} = \mathbf{B}^\top\mathbf{W}\mathbf{B}$, so that

$$\mathbf{P}_{\boldsymbol{w}} = \mathbf{W}^{1/2}\mathbf{B}\mathbf{L}^\dagger\mathbf{B}^\top\mathbf{W}^{1/2}.$$

In this subsection, we discuss nested dissection and the corresponding Schur complements, and explain how it relates to the inverse Laplacian $\mathbf{L}^{-1}$. We first illustrate the key ideas using a two-layer nested dissection scheme.

By the well-known planar separator theorem [50] a planar graph can be decomposed into two *edge-disjoint* subgraphs H_1 and H_2 called *regions* each with at most $2n/3$ vertices. Furthermore, let ∂H_i denote the *boundary* of region H_i that is, the set of vertices $v \in H_i$ such that v is adjacent to some $u \notin H_i$, then ∂H_i has size $O(\sqrt{n})$. As discussed in the previous section, the input graph to the IPM is the original planar graph plus two additional vertices and $O(n)$ additional edges. We may add two vertices to both H_1 and H_2, and partition the edges accordingly.

Let $F_{H_i} = V(H_i) \setminus \partial H_i$ denote the remaining interior vertices *eliminated* at region H_i. Let $C = \partial H_1 \cup \partial H_2$ denote the union of the boundaries which is also a *balanced vertex separator* of G, with size $O(\sqrt{n})$. Let $F = F_{H_1} \cup F_{H_2}$ be the disjoint union of the two interior sets. F and C give a natural partition of the vertices of G Using block Cholesky decomposition, we can show that the pseudo-inverse of $\mathbf{L}$ is

$$\mathbf{L}^\dagger = \mathbf{P}_{\mathbf{L}} \begin{bmatrix} \mathbf{I} & -\mathbf{L}_{F,F}{}^{-1}\mathbf{L}_{F,C} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{L}_{F,F}{}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{Sc}(\mathbf{L}, C)^\dagger \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{L}_{C,F}\mathbf{L}_{F,F}{}^{-1} & \mathbf{I} \end{bmatrix} \mathbf{P}_{\mathbf{L}}, \tag{3.4}$$

where $\mathbf{P}_{\mathbf{L}}$ is the projection matrix onto the image of $\mathbf{L}$ the term $\mathbf{Sc}(\mathbf{L}, C) \stackrel{\text{def}}{=} \mathbf{L}_{C,C} - \mathbf{L}_{C,F}\mathbf{L}_{F,F}{}^{-1}\mathbf{L}_{F,C}$ is the *Schur complement* of $\mathbf{L}$ onto vertex set C, and $\mathbf{L}_{F,C} \in \mathbb{R}^{F \times C}$ is the $F \times C$-indexed submatrix of $\mathbf{L}$.

IPMRun in Algorithm 1 updates $\mathbf{L}^{\dagger}$ at every step; written as the above decomposition, we must in turn update the Schur complement $\mathbf{Sc}(\mathbf{L}, C)$ at every step. Hence, the update cost must be sub-linear in n. Computing $\mathbf{Sc}(\mathbf{L}, C)$ exactly takes $\Omega(|C|^2) = \Omega(n)$ time, which is already too expensive. Our key idea here is to maintain an approximate Schur complement, which is of a smaller size based on the graph decomposition and can be maintained in amortized $\sqrt{n}$ time per step throughout the IPM.

Let $\mathbf{L}[H_i]$ denote the weighted Laplacian of the region H_i for $i = 1, 2$ Since these regions are edge-disjoint, we can write the Laplacian $\mathbf{L}$ as the sum

$$\mathbf{L} = \mathbf{L}[H_1] + \mathbf{L}[H_2].$$

Based on the graph decomposition, we have the Schur complement decomposition

$$\mathbf{Sc}(\mathbf{L}, C) = \mathbf{Sc}(\mathbf{L}[H_1], C) + \mathbf{Sc}(\mathbf{L}[H_2], C).$$

This decomposition allows us to localize edge weight updates. Namely, if the weight of edge e is updated, and e is contained in region H_i, we only need to recompute the single Schur complement term for H_i, rather than both terms in the sum.

For the appropriate projection matrices in the IPM it further suffices to maintain a sparse *approximate Schur complement* $\widetilde{\mathbf{Sc}}(\mathbf{L}[H_i], C) \approx \mathbf{Sc}(\mathbf{L}[H_i], C)$ for each region H_i rather than the exact. Then, the approximate Schur complement of $\mathbf{L}$ on C is given by

$$\widetilde{\mathbf{Sc}}(\mathbf{L}, C) \stackrel{\text{def}}{=} \widetilde{\mathbf{Sc}}(\mathbf{L}[H_1], C) + \widetilde{\mathbf{Sc}}(\mathbf{L}[H_2], C). \tag{3.5}$$

Each term $\widetilde{\mathbf{Sc}}(\mathbf{L}[H_i], C)$ can be computed in time nearly-linear in the size of H_i. Furthermore, $\widetilde{\mathbf{Sc}}(\mathbf{L}[H_i], C)$ is supported only on the vertex set ∂H_i, which is of size $O(\sqrt{n})$ Hence, any *sparse* approximate Schur complement has only $\widetilde{O}(\sqrt{n})$ edges. When we need to compute $\widetilde{\mathbf{Sc}}(\mathbf{L}, C)^{\dagger}\boldsymbol{x}$ for some vector $\boldsymbol{x}$, we use a fast SDD-solver which runs in $\widetilde{O}(|C|)$ time; this is crucial in bounding the overall runtime.

To extend the two-level scheme to more layers, we apply nested dissection recursively to each region H_i, until the regions are of constant size This recursive procedure naturally gives rise to a *separator tree* $\mathcal{T}$ of the input graph G, which we discuss in detail in Section 4.2 Each node of $\mathcal{T}$ correspond to a region of G, and can be obtained by taking the union of the regions corresponding to its two children Taking the union over all leaf regions gives the original graph G The separator tree $\mathcal{T}$ allows us to define a set F_H of *eliminated vertices* and a set ∂H of *boundary vertices* for each node H, analogous to what was shown in the two-layer dissection. Let η denote the height of $\mathcal{T}$, then $\mathbf{L}^{\dagger}$ admits an η-level recursive decomposition, and within each level, the expression can be further decomposed according to nodes at the level. Then, combined with approximate Schur complements, we can obtain an approximation to $\mathbf{L}^{\dagger}$ as follows:

Theorem 5 ($\mathbf{L}^{\dagger}$ Approximation). *Let $\mathbf{P}_{\mathbf{L}}$ be the projection onto the image of $\mathbf{L}$. Suppose for each $H \in \mathcal{T}$, 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

$$\mathbf{L}^{\dagger} \approx_{\eta\epsilon_{\mathbf{P}}} \mathbf{P}_{\mathbf{L}}\mathbf{\Pi}^{(0)\top} \cdots \mathbf{\Pi}^{(\eta-1)\top}\widetilde{\mathbf{\Gamma}}\mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)}\mathbf{P}_{\mathbf{L}}, \tag{3.6}$$

where[3]

[3]Note that $(\mathbf{L}^{(H)}_{F_H, F_H})^{\dagger}$ is a pseudo-inverse when H is the root node; for notational convenience we use $^{-1}$ even for the pseudo-inverse.

$$\widetilde{\boldsymbol{\Gamma}} = \begin{bmatrix} \sum_{H\in\mathcal{T}(0)} \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \sum_{H\in\mathcal{T}(\eta)} \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1} \end{bmatrix}$$

is a block-diagonal matrix, and

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

where $\mathcal{T}(i)$ denotes the set of nodes at level i of $\mathcal{T}$, and $\mathbf{I}$ is the $n\times n$ identity matrix.

To compute and maintain the necessary $\mathbf{L}^{(H)}$'s as the edge weights undergo updates throughout the IPM, we have the following data structure:

Theorem 6 (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 3) 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 $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}m)$ time.*
- *Reweight($\Delta\boldsymbol{w}\in\mathbb{R}^m_{>0}$): Update the weights to $\boldsymbol{w}+\Delta\boldsymbol{w}$, and recompute the relevant Schur complements in $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK})$ time, where $K = \mathrm{nnz}(\Delta\boldsymbol{w})$.*
  If $\mathcal{H}$ is the set of leaf nodes in $\mathcal{T}$ that contain an edge whose weight is updated, then $\mathbf{L}^{(H)}$ and $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ are updated only for nodes $H\in\mathcal{P}_{\mathcal{T}}(\mathcal{H})$.
- *Access to Laplacian $\mathbf{L}^{(H)}$ at any node $H\in\mathcal{T}$ in $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}|F_H\cup\partial H|)$ time.*
- *Access to Laplacian $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ at any node $H\in\mathcal{T}$ in $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}|\partial H|)$ time.*

Furthermore, at all points during the IPM,

$$\mathbf{L}^{(H)} \approx_{\epsilon_{\mathbf{P}}} \mathbf{Sc}(\mathbf{L}[H], F_H\cup\partial H) \quad \textit{and} \quad \widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H) \approx_{\epsilon_{\mathbf{P}}} \mathbf{Sc}(\mathbf{L}[H], \partial H) \tag{3.7}$$

for all $H\in\mathcal{T}$ with high probability.

3.4 Implicit Representations Using Tree Operator

In this subsection, we outline the data structures for maintaining the flow and slack solutions $\boldsymbol{f}, \boldsymbol{s}$ as needed in Algorithm 1, Line 22. Recall at IPM step k with step direction $\boldsymbol{v}^{(k)}$, we want to update

$$\boldsymbol{s} \leftarrow \boldsymbol{s} + \bar{t}h\mathbf{W}^{-1/2}\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}^{(k)},$$
$$\boldsymbol{f} \leftarrow \boldsymbol{f} + h\mathbf{W}^{1/2}\boldsymbol{v}^{(k)} - h\mathbf{W}^{1/2}\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v}^{(k)},$$

for some approximate matrices $\widetilde{\mathbf{P}}_{\boldsymbol{w}}$ and $\widetilde{\mathbf{P}}'_{\boldsymbol{w}}$ satisfying $\mathrm{Range}(\mathbf{W}^{-1/2}\widetilde{\mathbf{P}}_{\boldsymbol{w}}) \subseteq \mathrm{Range}(\mathbf{B})$ and $\mathbf{B}^\top\mathbf{W}^{1/2}\widetilde{\mathbf{P}}'_{\boldsymbol{w}} = \mathbf{B}^\top\mathbf{W}^{1/2}$. The first term for the flow update is straightforward to maintain. For this overview, we therefore, focus on maintaining the second term

$$\boldsymbol{f}^\perp \leftarrow \boldsymbol{f}^\perp + h\mathbf{W}^{1/2}\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v}^{(k)}.$$

Computing $\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}^{(k)}$ and $\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v}^{(k)}$, respectively, is too costly to do at every IPM step. Instead, we maintain vectors $\boldsymbol{s}_0, \boldsymbol{f}_0^\perp, \boldsymbol{z}$, and implicitly maintain two linear operators $\mathbf{M}^{(\mathrm{slack})}, \mathbf{M}^{(\mathrm{flow})}$ which depend on the weights $\boldsymbol{w}$, so at the end of every IPM step, the correct current solutions $\boldsymbol{s}, \boldsymbol{f}^\perp$ are recoverable via the identity

$$s = s_0 + \mathbf{M}^{(\text{slack})} z$$
$$f^{\perp} = f_0^{\perp} + \mathbf{M}^{(\text{flow})} z.$$

We will abstract away the difference between slack and flow to give a general data structure MaintainRep, which maintains $x = y + \mathbf{M}z$ for $\mathbf{M}$ with a special tree structure.

At a high level, MaintainRep implements the IPM step: To move in step k with direction $v^{(k)}$ and step size $\alpha^{(k)}$ the data structure first computes $z^{(k)}$ as a function of $v^{(k)}$, then updates $z \leftarrow z + \alpha^{(k)} z^{(k)}$, which translates to the desired overall update in x of $x \leftarrow x + \mathbf{M}(\alpha^{(k)} z^{(k)})$ To update the data structure with respect to new weights $w^{(\text{new})}$ (which does not change the value of x) the data structure first computes $\mathbf{M}^{(\text{new})}$ using $w^{(\text{new})}$ and $\Delta\mathbf{M} \overset{\text{def}}{=} \mathbf{M}^{(\text{new})} - \mathbf{M}$, then updates $\mathbf{M} \leftarrow \mathbf{M}^{(\text{new})}$. This causes an increase in value in the $\mathbf{M}z$ term by $\Delta\mathbf{M}z$ which is then offset in the y term with $y \leftarrow y - \Delta\mathbf{M}z$.

In later sections, we will define $\mathbf{M}^{(\text{slack})}$ and $\mathbf{M}^{(\text{flow})}$ so that $\mathbf{M}^{(\text{slack})} z^{(k)} = \mathbf{W}^{1/2} \widetilde{\mathbf{P}}_w v^{(k)}$ and $\mathbf{M}^{(\text{flow})} z^{(k)} = \mathbf{W}^{-1/2} \widetilde{\mathbf{P}}'_w v^{(k)}$ for the desired approximate projection matrices. With these operators appropriately defined, observed that MaintainRep correctly captures the updates to s and $f^{\perp}$ at every IPM step.

Let us now discuss the definition of z, which is common to both slack and flow: Recall the DynamicSC data structure from the previous section maintains some Laplacian $\mathbf{L}^{(H)}$ for every node H in the separator tree $\mathcal{T}$, so that at each IPM step, we can implicitly represent the matrices $\mathbf{\Pi}^{(0)}, \dots, \mathbf{\Pi}^{(\eta-1)}, \widetilde{\mathbf{\Gamma}}$ based on the current weights w, which together give an $\eta\epsilon_{\mathbf{P}}$-approximation of $\mathbf{L}^{\dagger}$ MaintainRep will contain a DynamicSC data structure, so we can use these Laplacians in the definition of z.

At step k, let

$$z^{(k)} \overset{\text{def}}{=} \widetilde{\mathbf{\Gamma}} \mathbf{\Pi}^{(\eta-1)} \dots \mathbf{\Pi}^{(0)} \mathbf{B}^{\top} \mathbf{W}^{1/2} v^{(k)},$$

where $\widetilde{\mathbf{\Gamma}}$, the $\mathbf{\Pi}^{(i)}$'s, and $\mathbf{W}$ are based on the state of the data structure at the end of step k. Next, z is defined to be the accumulation of $\alpha^{(i)} z^{(i)}$'s up to the current step; that is, at the end of step k,

$$z = \sum_{i=1}^{k} \alpha^{(i)} z^{(i)}.$$

Rather than naively maintaining z, we decompose z and explicitly maintaining c, $z^{(\text{step})}$, and $z^{(\text{sum})}$, such that

$$z \overset{\text{def}}{=} c \cdot z^{(\text{step})} + z^{(\text{sum})},$$

where we have the additional guarantee that at the end of IPM step k,

$$z^{(\text{step})} = \widetilde{\mathbf{\Gamma}} \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)} \mathbf{B}^{\top} \mathbf{W}^{1/2} v^{(k)}.$$

The other term, $z^{(\text{sum})}$, is some remaining accumulation so that the overall representation is correct.

The purpose of this decomposition of z is to facilitate sparse updates to v between IPM steps: Suppose $v^{(k)} = v^{(k-1)} + \Delta v$ where $\text{nnz}(\Delta v) = K$, then we can update $z^{(\text{step})}$ and $z^{(\text{sum})}$ with runtime as a function of K, while producing the correct overall update in z. We compute $\Delta z^{(\text{step})} = \widetilde{\mathbf{\Gamma}} \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)} \mathbf{B}^{\top} \mathbf{W}^{1/2} \Delta v$, and then set

$$z^{(\text{step})} \leftarrow z^{(\text{step})} + \Delta z^{(\text{step})}, \; z^{(\text{sum})} \leftarrow z^{(\text{sum})} - c \cdot \Delta z^{(\text{step})}, c \leftarrow c + \alpha,$$

which can be performed in $O(\text{nnz}(\Delta z^{(\text{step})}))$ time.

Let us briefly discuss how to compute $\widetilde{\Gamma}\Pi^{(\eta-1)}\cdots\Pi^{(0)}\boldsymbol{d}$ for some vector $\boldsymbol{d}$. We use the two-layer nested dissection setup from Section 3.3 for intuition, so

$$\widetilde{\Gamma}\Pi^{(0)}\boldsymbol{d} = \begin{bmatrix} \mathbf{L}_{F,F}^{-1} & \mathbf{0} \\ \mathbf{0} & \widetilde{\mathbf{Sc}}(\mathbf{L},C)^{\dagger} \end{bmatrix} \begin{bmatrix} \mathbf{I} & 0 \\ -\mathbf{L}_{C,F}\mathbf{L}_{F,F}^{-1} & \mathbf{I} \end{bmatrix} \boldsymbol{d}.$$

The only difficult part for the next left matrix multiplication is $-\mathbf{L}_{C,F}\mathbf{L}_{F,F}^{-1}$. However, we note that $\mathbf{L}_{F,F}$ is block-diagonal with two blocks, each corresponding to a region generated during nested dissection. Hence, we can solve the Laplacians on the two subgraphs separately. Next, we note that the two terms of $\mathbf{L}_{C,F}\mathbf{L}_{F,F}^{-1}\boldsymbol{d}$ are both fed into $\widetilde{\mathbf{Sc}}(\mathbf{L},C)^{\dagger}$, and we solve this Laplacian in time linear in the size of $\widetilde{\mathbf{Sc}}(\mathbf{L},C)$. The rest of the terms are not the bottleneck in the overall runtime. In the more general nested-dissection setting with $O(\log n)$ layers, we solve a sequence of Laplacians corresponding to the regions given by paths in the separator tree. We can bound the runtime of these Laplacian solves by the size of the corresponding regions for the desired overall runtime.

On the other hand, to work with $\mathbf{M}$ efficiently, we define the notion of a *tree operator* $\mathbf{M}$ supported on a tree. In our setting, we use the separator tree $\mathcal{T}$. Informally, our tree operator is a linear operator mapping $\mathbb{R}^n$ to $\mathbb{R}^m$. It is constructed from the concatenation of a collection of *edge operators* and *leaf operators* defined on the edges and leaves of $\mathcal{T}$. If H is a node in $\mathcal{T}$ with parent P, then the edge operator for edge (H,P) will map vectors supported on $F_P \cup \partial P$ to vectors supported on $F_H \cup \partial H$. If H is a leaf node, the leaf operator for H will map vectors on $F_H \cup \partial H$ to vectors on $E(H)$. In this way, we take advantage of the recursive partitioning of G via $\mathcal{T}$ to map a vector supported on $V(G)$ recursive to be supported on smaller vertex subsets and finally to the edges. Furthermore, we will show that when edge weights update, the change to $\mathbf{M}$ can be localized to a small collection of edge and leaf operators along some tree paths, thus allowing for an efficient implementation. We postpone the formal definition of the operator until Section 5.2.

THEOREM 7 (IMPLICIT REPRESENTATION MAINTENANCE). *Suppose, for simplicity of this theorem statement, that we can solve $m\times m$ SDD matrices in $\widetilde{O}(m)$ time to high accuracy. Given an appropriate separator tree $\mathcal{T}$ for the IPM input graph with height η, the deterministic data structure* MAINTAINREP *(Algorithm 6) maintains the following variables correctly at the end of every IPM step:*

— *the dynamic edge weights $\boldsymbol{w}$ and step direction $\boldsymbol{v}$ from the current IPM step,*
— *a* DYNAMICSC *data structure on $\mathcal{T}$ based on the current IPM edge weights $\boldsymbol{w}$,*
— *an implicitly represented tree operator $\mathbf{M}$ supported on $\mathcal{T}$ with complexity $T(K)$,* computable using information from DYNAMICSC,
— *scalar c and vectors $\boldsymbol{z}^{(\text{step})}, \boldsymbol{z}^{(\text{sum})}$, which together represent $\boldsymbol{z} = c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})}$, such that at the end of step k, the variables satisfy*

$$\boldsymbol{z} = \sum_{i=1}^{k} \alpha^{(i)}\boldsymbol{z}^{(i)},$$

where $\alpha^{(i)}$ is the step size α given in MOVE *for step i,*
— *$\boldsymbol{z}^{(\text{step})}$ satisfies $\boldsymbol{z}^{(\text{step})} = \widetilde{\Gamma}\Pi^{(\eta-1)}\cdots\Pi^{(0)}\mathbf{B}^{\top}\mathbf{W}^{1/2}\boldsymbol{v}$,*
— *an offset vector $\boldsymbol{y}$ which together with $\mathbf{M}, \boldsymbol{z}$ represent $\boldsymbol{x} = \boldsymbol{y} + \mathbf{M}\boldsymbol{z}$, such that after step k,*

$$\boldsymbol{x} = \boldsymbol{x}^{(\text{init})} + \sum_{i=1}^{k} \mathbf{M}^{(i)}(\alpha^{(i)}\boldsymbol{z}^{(i)}),$$

where $\boldsymbol{x}^{(\text{init})}$ is an initial value from INITIALIZE, *and $\mathbf{M}^{(i)}$ is the state of $\mathbf{M}$ after step i.*

The data structure supports the following procedures:

— *Initialize*$(\mathcal{T}, \mathbf{M} \in \mathbb{R}^{m\times n}, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \boldsymbol{x}^{(\text{init})} \in \mathbb{R}^m, \epsilon_{\mathbf{P}} > 0)$*: Given a separator tree* $\mathcal{T}$ *for the input graph, a tree operator* $\mathbf{M}$ *supported on* $\mathcal{T}$ *with complexity* T*, initial step direction* $\boldsymbol{v}$*, initial weights* $\boldsymbol{w}$*, initial vector* $\boldsymbol{x}^{(\text{init})}$*, and target step accuracy* $\epsilon_{\mathbf{P}}$*, preprocess in* $\widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}m+T(m))$ *time and set* $\boldsymbol{x} \leftarrow \boldsymbol{x}^{(\text{init})}$.

— *Reweight*$(\Delta\boldsymbol{w} \in \mathbb{R}^m)$*: Update the weights to* $\boldsymbol{w} + \Delta\boldsymbol{w}$*. Update the implicit representation of* $\boldsymbol{x}$ *without changing its value, so that all the variables in the data structure are based on the new weights.*
The procedure runs in $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK} + T(K))$ *total time, where* K *is an upper bound on* $\text{nnz}(\Delta\boldsymbol{w})$ *and the number of leaf or edge operators changed in* $\mathbf{M}$*. There are most* $\widetilde{O}(K)$ *nodes* $H \in \mathcal{T}$ *for which* $\boldsymbol{z}^{(\text{step})}|_{F_H}$ *and* $\boldsymbol{z}^{(\text{sum})}|_{F_H}$ *are updated.*

— *Move*$(\alpha \in \mathbb{R}, \Delta\boldsymbol{v} \in \mathbb{R}^n)$*: Update the current direction to* $\boldsymbol{v} + \Delta\boldsymbol{v}$*, and then update the implicit representation of* $\boldsymbol{x}$ *to reflect the following change in value:*

$$\boldsymbol{x} \leftarrow \boldsymbol{x} + \mathbf{M}(\alpha \boldsymbol{z}^{(\text{step})}).$$

The procedure runs in $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK})$ *time.*

— *Exact*$()$*: Output the current exact value of* $\boldsymbol{x} = \boldsymbol{y} + \mathbf{M}\boldsymbol{z}$ *in* $\widetilde{O}(T(m))$ *time.*

3.5 Solution Approximation

In the flow and slack maintenance data structures, one key operation is to maintain vectors $\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}$ that are close to $\boldsymbol{f}, \boldsymbol{s}$ throughout the IPM. Since we have implicit representations of the solutions of the form $\boldsymbol{x} = \boldsymbol{y} + \mathbf{M}\boldsymbol{z}$ we now show how to maintain $\overline{\boldsymbol{x}}$ close to $\boldsymbol{x}$. To accomplish this, we will give a meta data structure that solves this in a more general setting. The data structure involves three steps; the first two steps are similar to [20] and the key contribution is the last step:

(1) We maintain an approximate vector by detecting coordinates of the exact vector $\boldsymbol{x}$ with large changes In step k of the IPM, for every ℓ such that $2^\ell | k$, we consider all coordinates of the approximate vector $\overline{\boldsymbol{x}}$ that did not change in the last 2^ℓ steps If any of them is off by more than $\bar{\epsilon}/(2\lceil\log m\rceil)$ from $\boldsymbol{x}$, it is updated We can prove that each coordinate of $\overline{\boldsymbol{x}}$ has additive error at most $\bar{\epsilon}$ compared to $\boldsymbol{x}$ The number of updates to $\overline{\boldsymbol{x}}$ will be roughly $O(2^{2\ell_k})$, where 2^{ℓ_k} is the largest power of 2 that divides k. This guarantees that K-sparse updates only happen $\sqrt{m/K}$ times throughout the IPM algorithm.
(2) We detect coordinates with large changes in $\boldsymbol{x}$ via a random sketch and sampling using the separator tree We can sample a coordinate with probability exactly proportional to the magnitude of its change, when given access to the approximate sum of probabilities in each region of the separator tree and to the exact value of any single coordinate of $\boldsymbol{x}$.
(3) We show how to maintain random sketches for vectors of the form $\boldsymbol{x} = \boldsymbol{y} + \mathbf{M}\boldsymbol{z}$, where $\mathbf{M}$ is an implicit tree operator supported on a tree $\mathcal{T}$ Specifically, to maintain sketches of $\mathbf{M}\boldsymbol{z}$, we store intermediate sketches for every complete subtree of $\mathcal{T}$ at their roots. When an edge operator of $\mathbf{M}$ or a coordinate of $\boldsymbol{z}$ is modified, we only need to update the sketches along a path in $\mathcal{T}$ from a node to the root For our use case, the cost of updating the sketches at a node H will be proportional to its separator size, so that a K-sparse update takes $\widetilde{O}(\sqrt{mK})$ time.

While the data structure is randomized, it is guaranteed to work against an adaptive adversary that is allowed to see the entire internal state of the data structure, including the random bits.

Theorem 8 (Approximate Vector Maintenance with Tree Operator). *Given a constant degree tree* $\mathcal{T}$ *with height* η *that supports tree operator* $\mathbf{M}$ *with complexity* T *there is a randomized data structure* MaintainApprox *that takes as input the dynamic variables* $\mathbf{M}, c, \boldsymbol{z}^{(\text{step})}, \boldsymbol{z}^{(\text{sum})}, \boldsymbol{y}$ *at* *every IPM step, and maintains the approximation* $\overline{\boldsymbol{x}}$ *to* $\boldsymbol{x} \overset{\text{def}}{=} \boldsymbol{y} + \mathbf{M}\boldsymbol{z} = \boldsymbol{y} + \mathbf{M}(c \cdot \boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})})$ *satisfying* $\left\|\mathbf{D}^{1/2}(\boldsymbol{x} - \overline{\boldsymbol{x}})\right\|_\infty \leq \bar{\epsilon}$*, where* $\mathbf{D}$ *is coordinate-wise a fixed function of* $\overline{\boldsymbol{x}}$*. It supports the following procedures:*

— *Initialize*(*tree* $\mathcal{T}$, *tree operator* $\mathbf{M}, c \in \mathbb{R}, \boldsymbol{z}^{(\text{step})} \in \mathbb{R}^n, \boldsymbol{z}^{(\text{sum})} \in \mathbb{R}^n, \boldsymbol{y} \in \mathbb{R}^m, \rho > 0, \bar{\epsilon} > 0$)*: Initialize the data structure with initial vector* $\boldsymbol{x} = \boldsymbol{y} + \mathbf{M}(c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})})$ *target approximation accuracy* $\bar{\epsilon}$*, success probability* $1 - \rho$*, in* $O(m\eta^2 \log m \log(\frac{m}{\rho}))$ *time. Initialize* $\overline{\boldsymbol{x}} \leftarrow \boldsymbol{x}$*.*

— *Approximate*($\mathbf{M}, c, \boldsymbol{z}^{(\text{step})}, \boldsymbol{z}^{(\text{sum})}, \boldsymbol{y}$)*: Update the internal variables to their new iterations as given. Then output a vector* $\overline{\boldsymbol{x}}$ *such that* $\|\mathbf{D}^{1/2}(\boldsymbol{x} - \overline{\boldsymbol{x}})\|_\infty \leq \bar{\epsilon}$ *for the current vector* $\boldsymbol{x}$ *and diagonal scaling* $\mathbf{D}$ *corresponding to* $\overline{\boldsymbol{x}}$*.*

Suppose $\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\|_{\mathbf{D}^{(k)}} \leq \beta$ for all k *where* $\mathbf{D}^{(k)}$ *and* $\boldsymbol{x}^{(k)}$ *are the* $\mathbf{D}$ *and* $\boldsymbol{x}$ *at the* k*th call to* *Approximate. Then, for the* k*th call to* *Approximate, we have*

— *the data structure updates* $\overline{\boldsymbol{x}}_i \leftarrow \boldsymbol{x}_i^{(k)}$ *for* $O(N_k \overset{\text{def}}{=} 2^{2\ell_k}(\beta/\bar{\epsilon})^2 \log^2 m)$ *coordinates, where* ℓ_k *is the largest integer* ℓ *with* $k \equiv 0 \bmod 2^\ell$*.*

— *The amortized time cost of* *Approximate is*

$$\Theta\left(\eta^2 \log\left(\frac{m}{\rho}\right) \log m\right) \cdot T\left(\eta \cdot \left(N_{k-2^{\ell_k}} + |\mathcal{S}|\right)\right),$$

where $\mathcal{S}$ *is the set of nodes* H *where either* $\mathbf{M}_{(H,P)}, \mathbf{J}_H, \boldsymbol{z}^{(\text{step})}|_{F_H}$*, or* $\boldsymbol{z}^{(\text{sum})}|_{F_H}$ *changed, or where* $\boldsymbol{y}_e$ *changed for coordinate* e*, compared to the* $(k-1)$*-th step.*

3.6 Slack and Flow Projection

We want to use MaintainRep data structures to implicitly maintain the flow and slack solutions throughout the IPM, and use MaintainApprox data structures to explicitly maintain their approximations. It remains to define suitable tree operators $\mathbf{M}^{(\text{slack})}$ and $\mathbf{M}^{(\text{flow})}$, so that at IPM step k, the updates in MaintainRep are correct IPM updates:

$$\boldsymbol{s} \leftarrow \boldsymbol{s} + \bar{t}h\mathbf{W}^{-1/2}\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}^{(k)} = \boldsymbol{s} + \bar{t}h\mathbf{M}^{(\text{slack})}\boldsymbol{z}^{(\text{step})}$$
$$\boldsymbol{f}^{\perp} \leftarrow \boldsymbol{f}^{\perp} + h\mathbf{W}^{1/2}\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v}^{(k)} = \boldsymbol{f}^{\perp} + h\mathbf{M}^{(\text{flow})}\boldsymbol{z}^{(\text{step})},$$

where $\boldsymbol{z}^{(\text{step})} = \widetilde{\mathbf{\Gamma}}\mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)}\mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}^{(k)}$ at the end of IPM step k. Let $\widetilde{\mathbf{L}}^\dagger$ denote the approximation of $\mathbf{L}^\dagger$ from Equation (3.6), maintained with a DynamicSC data structure. We define

$$\widetilde{\mathbf{P}}_{\boldsymbol{w}} = \mathbf{W}^{1/2}\mathbf{B}\widetilde{\mathbf{L}}^\dagger\mathbf{B}^\top\mathbf{W}^{1/2} = \mathbf{W}^{1/2}\mathbf{B}\mathbf{\Pi}^{(0)} \cdots \mathbf{\Pi}^{(\eta-1)}\widetilde{\mathbf{\Gamma}}\mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)}\mathbf{B}^\top\mathbf{W}^{1/2};$$

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. Hence, $\widetilde{\mathbf{P}}_{\boldsymbol{w}}$ suffices as the approximate slack projection matrix.

Using Section 3.4, we can write

$$\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}^{(k)} = \mathbf{W}^{1/2}\mathbf{B}\mathbf{\Pi}^{(0)\top} \cdots \mathbf{\Pi}^{(\eta-1)\top}\boldsymbol{z}^{(\text{step})}. \tag{3.8}$$

The matrix multiplication applied to $\boldsymbol{z}^{(\text{step})}$ can indeed be represented by a tree operator $\mathbf{M}$ on the tree $\mathcal{T}$ Intuitively, observe that each $\mathbf{\Pi}^{(i)}$ operates on level i of $\mathcal{T}$ and can be decomposed to be written in terms of the nodes at level i. Furthermore, the $\mathbf{\Pi}^{(i)}$'s are applied in order of descending level in $\mathcal{T}$ Finally, at the leaf level, $\mathbf{W}^{1/2}\mathbf{B}$ maps vectors on vertices to vectors on edges. In Section 7, we present the exact tree operator and its correctness proof. With it, we have

$$\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}^{(k)} = \mathbf{W}^{1/2}\mathbf{M}\boldsymbol{z}^{(\text{step})}.$$

We set $\mathbf{M}^{(\text{slack})} \overset{\text{def}}{=} \mathbf{W}^{1/2}\mathbf{M}$, which is also a valid tree operator.

For the flow update, observe we need a flow $\tilde{\boldsymbol{f}} \approx \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}$ satisfying $\mathbf{B}^\top\mathbf{W}^{1/2}\tilde{\boldsymbol{f}} = \mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}$. Let us define demands on vertices by $\boldsymbol{d} \overset{\text{def}}{=} \mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}$. Unwrapping the definition of $\mathbf{P}_{\boldsymbol{w}} \overset{\text{def}}{=} \mathbf{W}^{1/2}\mathbf{B}\mathbf{L}^\dagger\boldsymbol{d}$, we see $\tilde{\boldsymbol{f}}$ must be a flow routing demand $\boldsymbol{d}$, i.e., $\mathbf{B}^\top\mathbf{W}^{1/2}\tilde{\boldsymbol{f}} = \mathbf{B}^\top\mathbf{W}\mathbf{B}\mathbf{L}^\dagger\boldsymbol{d} = \boldsymbol{d}$. We will in fact define $\tilde{\boldsymbol{f}} = \widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}$ where $\widetilde{\mathbf{P}}_{\boldsymbol{w}}$ is the same as the slack update. However, $\tilde{\boldsymbol{f}}$ might not satisfy the feasible flow condition; the insight then is to route the remaining demands in individual leaf-node regions in the separator tree, rather than on the whole graph.

In Appendix C, we formalize an interpretation of $\tilde{\boldsymbol{f}}$ in association with the tree operator. Here, we give an intuitive explanation. For simplicity of notation, let $\boldsymbol{z}$ denote $\boldsymbol{z}^{(\text{step})}$. The first step is recognizing a decomposition of $\boldsymbol{d}$ using the separator tree, such that we have a demand term $\boldsymbol{d}^{(H)}$ for each node $H \in \mathcal{T}$. Furthermore, $\boldsymbol{d}^{(H)} = \mathbf{L}^{(H)}\boldsymbol{z}|_{F_H}$ for the Laplacian $\mathbf{L}^{(H)}$ supported on the region H maintained by DYNAMICSC. This decomposition allows us to route each demand $\boldsymbol{d}^{(H)}$ by electrical flows using only the corresponding region H rather than the entire graph. To show that the resulting flow $\tilde{\boldsymbol{f}}$ indeed is close to the electrical flow one key insight is that the decomposed demands are orthogonal (Lemma 74). Hence, routing them separately by electrical flows gives a good approximation to the true electrical flow of the whole demand (Theorem 71).

To give an illustrative example, consider the two-layer graph decomposition scheme from Section 3.3, and suppose we have a demand $\boldsymbol{d}$ *that is non-zero only on vertices of* C Then, observe that

$$\boldsymbol{z} = \begin{bmatrix} \mathbf{L}_{F,F}^{-1} & \mathbf{0} \\ \mathbf{0} & \widetilde{\mathbf{Sc}}(\mathbf{L}, C)^\dagger \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{L}_{C,F}\mathbf{L}_{F,F}^{-1} & \mathbf{I} \end{bmatrix} \boldsymbol{d}.$$

Looking at the sub-vector indexed by C on both sides, we have that

$$\widetilde{\mathbf{Sc}}(\mathbf{L}, C)\boldsymbol{z} = \boldsymbol{d},$$

where we abuse the notation to extend $\widetilde{\mathbf{Sc}}(\mathbf{L}, C)$ from $C \times C$ to $[n] \times [n]$ by padding zeros. Using Equation (3.5), we have

$$\left(\widetilde{\mathbf{Sc}}(\mathbf{L}[H_1], C) + \widetilde{\mathbf{Sc}}(\mathbf{L}[H_2], C)\right)\boldsymbol{z} = \boldsymbol{d}.$$

This gives a decomposition of the demand $\boldsymbol{d}$ into demand terms $\widetilde{\mathbf{Sc}}(\mathbf{L}[H_i], C)\boldsymbol{z}$ for $i = 1, 2$. Crucially, each demand $\widetilde{\mathbf{Sc}}(\mathbf{L}[H_i], C)\boldsymbol{z}$ is supported on the vertices of the region H_i, and we can route the flow on the corresponding region only In a $O(\log n)$-level decomposition, we recursively decompose the demand further based on the sub-regions according to the separator tree $\mathcal{T}$. This guarantees that $\tilde{\boldsymbol{f}}_i$ is the electrical flow on the subgraph H_i that satisfies the demand $\widetilde{\mathbf{Sc}}(\mathbf{L}[H_i], C)\boldsymbol{z}$ Finally, we will let the output be $\tilde{\boldsymbol{f}} = \sum \tilde{\boldsymbol{f}}_i$ Note that by construction, if all the Schur complements were exact, then this $\tilde{\boldsymbol{f}}$ routes the demand $\boldsymbol{d}$ exactly.

We only state and prove the data structure theorem for the flow solution in our article. The slack solution is essentially identical and omitted.

THEOREM 9 (FLOW MAINTENANCE). *Suppose, for simplicity of this theorem statement, that we can solve $m \times m$ SDD matrices in $\widetilde{O}(m)$ time to high accuracy. Given the appropriate separator tree $\mathcal{T}$ with height η for the IPM input graph, the data structure* MAINTAINFLOW *(Algorithm 10) implicitly maintains the flow solution $\boldsymbol{f}$ undergoing IPM changes, and explicitly maintains its approximation $\overline{\boldsymbol{f}}$, and supports the following procedures with high probability against an adaptive adversary:*

- *INITIALIZE($\mathcal{T}, \boldsymbol{f}^{(\text{init})} \in \mathbb{R}^m, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \epsilon_{\mathbf{P}} > 0, \overline{\epsilon} > 0$): Given the separator tree $\mathcal{T}$, initial solution $\boldsymbol{f}^{(\text{init})}$, initial direction $\boldsymbol{v}$ initial weights $\boldsymbol{w}$, target step accuracy $\epsilon_{\mathbf{P}}$, and target approximation accuracy $\overline{\epsilon}$ preprocess in $\widetilde{O}(m\epsilon_{\mathbf{P}}^{-2})$ time and set the internal representation $\boldsymbol{f} \leftarrow \boldsymbol{f}^{(\text{init})}$ and $\overline{\boldsymbol{f}} \leftarrow \boldsymbol{f}$.*

— *Reweight*($\Delta \boldsymbol{w} \in \mathbb{R}^m$)*: Set the current weights to $\boldsymbol{w} + \Delta \boldsymbol{w}$ in $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK})$ time, where $K = \mathrm{nnz}(\Delta \boldsymbol{w})$.*

— *Move*($\alpha \in \mathbb{R}, \Delta \boldsymbol{v} \in \mathbb{R}^m$)*: Update $\boldsymbol{v}$ to $\boldsymbol{v} + \Delta \boldsymbol{v}$. Implicitly update $\boldsymbol{f} \leftarrow \boldsymbol{f} + \alpha \mathbf{W}^{1/2}\boldsymbol{v} - \alpha \mathbf{W}^{1/2}\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}$ for some $\widetilde{\mathbf{P}}_{\boldsymbol{w}}$ where $\|(\widetilde{\mathbf{P}}_{\boldsymbol{w}} - \mathbf{P}_{\boldsymbol{w}})\boldsymbol{v}\|_2 \le O(\eta\epsilon_{\mathbf{P}})\,\|\boldsymbol{v}\|_2$ and $\mathbf{B}^\top \mathbf{W}^{1/2}\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v} = \mathbf{B}^\top \mathbf{W}^{1/2}\boldsymbol{v}$. The runtime is $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK})$, where $K = \mathrm{nnz}(\Delta \boldsymbol{v})$.*

— *Approximate*() $\to \mathbb{R}^m$*: Output the vector $\overline{\boldsymbol{f}}$ such that $\|\mathbf{W}^{-1/2}(\overline{\boldsymbol{f}} - \boldsymbol{f})\|_\infty \le \overline{\epsilon}$ for the current weight $\boldsymbol{w}$ and the current vector $\boldsymbol{f}$.*

— *Exact*() $\to \mathbb{R}^m$*: Output the current vector $\boldsymbol{f}$ in $\widetilde{O}(m\epsilon_{\mathbf{P}}^{-2})$ time.*

Suppose $\alpha\|\boldsymbol{v}\|_2 \le \beta$ for some β for all calls to Move. Suppose in each step, Reweight, Move and Approximate are called in order. Let K denote the total number of coordinates changed in $\boldsymbol{v}$ and $\boldsymbol{w}$ between the $(k-1)$-th and kth Reweight and Move calls. Then at the kth Approximate call,

— *the data structure sets $\overline{\boldsymbol{f}}_e \leftarrow \boldsymbol{f}_e^{(k)}$ for $O(N_k \overset{\text{def}}{=} 2^{2\ell_k}(\frac{\beta}{\overline{\epsilon}})^2 \log^2 m)$ coordinates e where ℓ_k is the largest integer ℓ with $k = 0 \mod 2^\ell$ when $k \neq 0$ and $\ell_0 = 0$.*

— *The amortized time for the kth Approximate call is $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{m(K + N_{k-2^{\ell_k}})})$.*

3.7 Main Proof

We are now ready to prove our main result, restated here:

Theorem 1 (Main Result). *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 $\widetilde{O}(n \log^2 M)$[4] expected time.*

Proof. First, we use Section 3.2 to reduce the planar min-cost flow problem to a form suitable for the solver in Theorem 4 The solver uses Algorithm 1, which involves the procedure IPMRun twice. In the first run, the constraint matrix is the incidence matrix of a new underlying graph, constructed by making three copies of each edge in the original graph G Since copying edges does not affect planarity, and our data structures allow for duplicate edges, we simply focus on the second run.

In the pseudocode given in Algorithm 2, we use $\boxed{\boldsymbol{f}}$ and $\boxed{s}$ to denote the solution data structures for flow and slack accordingly (as described in Theorem 9). During initialization, the data structures are given the initial IPM step direction $\boldsymbol{v}$ for preprocessing. At each step of IPM, we perform implicit updates of $\boldsymbol{f}$ and $\boldsymbol{s}$ using the procedure Move in their respective data structures. Since $\mathbf{W}$ is changing between steps, the data structures need to update their internal representations at each step, which we do using Reweight Finally, we construct the explicit approximations $\overline{\boldsymbol{f}}$ and $\overline{\boldsymbol{s}}$ using Approximate. We return the true solutions $(\boldsymbol{f}, \boldsymbol{s})$ by Exact.

Since $\overline{\boldsymbol{s}}$ must approximate $\boldsymbol{s}$ to an accuracy of $\overline{t}\alpha$, we must reinitialize the slack data structure during the IPM when $\overline{t}$ changes, which is not required for the flow data structure. However, note that $\overline{t}$ changes every $\widetilde{O}(\sqrt{m})$ steps, so this is not a bottleneck.

By the guarantees of Theorem 9, we correctly maintain $\boldsymbol{s}$ and $\boldsymbol{f}$ at every step in Algorithm 2, and the requirements on $\overline{\boldsymbol{f}}$ and $\overline{\boldsymbol{s}}$ for the IPM are satisfied.

To prove the runtime, we first introduce a lemma bounding the number of entry-changes in $\boldsymbol{v}$ and $\boldsymbol{w}$ each step.

Lemma 10. *When updating $\boldsymbol{w}$ and $\boldsymbol{v}$ at the $(k+1)$-th step of the IPM implementation in Algorithm 2, $\boldsymbol{w}$ and $\boldsymbol{v}$ change in $O(2^{2\ell_k} \log^2 m)$ coordinates, where ℓ_k is the largest integer ℓ with $k \equiv 0 \mod 2^\ell$.*

[4]Throughout the article, we use $\widetilde{O}(f(n))$ to denote $O(f(n) \log^{O(1)} f(n))$.

ALGORITHM 2: Implementation of IPM of Algorithm 1

```
1: procedure IPMRun($\mathbf{B} \in \mathbb{R}^{m\times n}$, barrier function $\phi$, $f \in \mathbb{R}^m$, $s \in \mathbb{R}^m$, $t_{\text{start}}$, $t_{\text{end}}$)
2:     Let $\alpha \overset{\text{def}}{=} \frac{1}{2^{20}\lambda}$ and $\lambda \overset{\text{def}}{=} 64\log(256m^2)$
3:     Construct separator tree $\mathcal{T}$ for the input graph $G$ by Corollary 25
4:     $\overline{f} \leftarrow f, \overline{s} \leftarrow s, \overline{t} \leftarrow t_{\text{start}}$
5:     $\mathbf{W} \leftarrow \nabla^2\phi(\overline{f})^{-1}$                                   ▷ initial weights
6:     $\boldsymbol{v}_i \leftarrow \sinh(\lambda\gamma^{\overline{t}}(\overline{f},\overline{s})_i)$ for all $i \in [n]$          ▷ initial step direction
7:     $\boxed{f}$.Initialize($\mathcal{T}, f, \boldsymbol{v}, \mathbf{W}, \epsilon_{\mathbf{P}} \overset{\text{def}}{=} O(\alpha/\log m), \bar{\epsilon} \overset{\text{def}}{=} \alpha$)      ▷ implicit data structs
8:     $\boxed{s}$.Initialize($\mathcal{T}, \overline{t}^{-1}s, \boldsymbol{v}, \mathbf{W}, \epsilon_{\mathbf{P}} \overset{\text{def}}{=} O(\alpha/\log m), \bar{\epsilon} \overset{\text{def}}{=} \alpha$)
9:     while $t \geq t_{\text{end}}$ do
10:        $t \leftarrow \max\{(1-\frac{\alpha}{\sqrt{m}})t, t_{\text{end}}\}$
11:        Update step size $h = -\alpha/\|\cosh(\lambda\gamma^{\overline{t}}(\overline{f},\overline{s}))\|_2$
12:        Update diagonal weight matrix $\mathbf{W} = \nabla^2\phi(\overline{f})^{-1}$
13:        Update step direction $\boldsymbol{v}$ where $\boldsymbol{v}_i \leftarrow \sinh(\lambda\gamma^{\overline{t}}(\overline{f},\overline{s})_i)$
14:        $\boxed{f}$.Reweight($\mathbf{W}$)
15:        $\boxed{s}$.Reweight($\mathbf{W}$)
16:        $\boxed{f}$.Move($h, \boldsymbol{v}$)          ▷ Update $f \leftarrow f + h\mathbf{W}^{1/2}\boldsymbol{v} - h\mathbf{W}^{1/2}\tilde{f}$ with $\tilde{f} \approx \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}$
17:        $\boxed{s}$.Move($h, \boldsymbol{v}$)                 ▷ Update $s \leftarrow s + \overline{t}h\mathbf{W}^{-1/2}\tilde{s}$ with $\tilde{s} \approx \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}$
18:        $\overline{f} \leftarrow \boxed{f}$.Approximate()           ▷ Maintain $\overline{f}$ such that $\|\mathbf{W}^{-1/2}(\overline{f}-f)\|_\infty \leq \alpha$
19:        $\overline{s} \leftarrow \overline{t}\cdot(\boxed{s}$.Approximate())       ▷ Maintain $\overline{s}$ such that $\|\mathbf{W}^{1/2}(\overline{s}-s)\|_\infty \leq \overline{t}\alpha$
20:        if $|\overline{t}-t| \geq \alpha\overline{t}$ then
21:            $s \leftarrow \overline{t}\cdot(\boxed{s}$.Exact())
22:            $\overline{t} \leftarrow t$
23:            $\boxed{s}$.Initialize($\mathcal{T}, \overline{t}^{-1}s, \boldsymbol{v}, \mathbf{W}, \epsilon_{\mathbf{P}} \overset{\text{def}}{=} O(\alpha/\log m), \bar{\epsilon} \overset{\text{def}}{=} \alpha$)
24:        end if
25:    end while
26:    return $\boxed{f}$.Exact(), $\overline{t}\cdot(\boxed{s}$.Exact())
27: end procedure
```

Proof. Since both $\boldsymbol{w}$ and $\boldsymbol{v}$ are an entry-wise function of $\overline{f}, \overline{s}$ and $\overline{t}$, we need to examine these variables. First, $\overline{t}$ changes every $\widetilde{O}(\sqrt{m})$ steps, and when $\overline{t}$ changes, every coordinate of $\boldsymbol{w}$ and $\boldsymbol{v}$ changes. Over the entire algorithm, $\overline{t}$ changes $\widetilde{O}(1)$ number of times, so we may incur an additive $\widetilde{O}(m)$ term overall, and assume $\overline{t}$ does not change for the rest of the analysis.

By Theorem 4, we have $h\|\boldsymbol{v}\|_2 = O(\frac{1}{\log m})$ at all steps. So, we apply Theorem 9 both with parameters $\beta = O(\frac{1}{\log m})$ and $\overline{\epsilon} = \alpha = \Theta(\frac{1}{\log m})$. At step k, there are $O(2^{2\ell_k}\log^2 m)$-many coordinates of $\overline{f}$ which change to the exact solution $f^{(k)}$ and no other changes. Hence $\boldsymbol{w}^{(k+1)}$ differ from $\boldsymbol{w}^{(k)}$ in $O(2^{2\ell_k}\log^2 m)$ coordinates. We have an analogous bound for $\overline{s}$, and, therefore, for the change in $\boldsymbol{v}$. □

Finally, we bound the runtime of Algorithm 2. The total cost is dominated by the cost of the while-loop. Since we call Move, Reweight and Approximate in order in each step and the runtime for Move, Reweight are both dominated by the runtime for Approximate, it suffices to bound the runtime for Approximate only. Theorem 4 guarantees that there are $T = O(\sqrt{m}\log n\log(nM))$ total Approximate calls. Lemma 10 shows that at the kth call, the number of coordinates changed in $\boldsymbol{w}$ and $\boldsymbol{v}$ is bounded by $K \overset{\text{def}}{=} O(2^{2\ell_{k-1}}\log^2 m)$, where ℓ_k is the largest integer ℓ with $k \equiv 0$

mod 2^ℓ, or equivalently, the number of trailing zeros in the binary representation of k. Theorem 4 further guarantees we can apply Theorem 9 with parameter $\beta = O(1/\log m)$, which in turn shows the amortized time for the kth call is

$$\widetilde{O}\left(\epsilon_{\mathbf{P}}{}^{-2}\sqrt{m(K + N_{k-2^{\ell_k}})}\right).$$

where $N_k \overset{\text{def}}{=} 2^{2\ell_k}(\beta/\alpha)^2\log^2 m = O(2^{2\ell_k}\log^2 m)$, where $\alpha = O(1/\log m)$ and $\epsilon_{\mathbf{P}} = O(1/\log m)$. Observe that $K + N_{k-2^{\ell_k}} = O(N_{k-2^{\ell_k}})$. Now, summing over all T calls, the total time is

$$\begin{aligned} O\left(\sqrt{m}\log m\right)\sum_{k=1}^{T}\sqrt{N_{k-2^{\ell_k}}} &= O\left(\sqrt{m}\log^2 m\right)\sum_{k=1}^{T}2^{\ell_{(k-2^{\ell_k})}} \\ &= O\left(\sqrt{m}\log^2 m\right)\sum_{k'=1}^{T}2^{\ell_{k'}}\sum_{k=1}^{T}[k-2^{\ell_k}=k'], \end{aligned}$$

where we use $[\cdot]$ for the indicator function, i.e., $[k-2^{\ell_k}=k']=1$ if $k-2^{\ell_k}=k'$ is true and 0, otherwise. As there are only $\log T$ different powers of 2 in $[1,T]$, the count $\sum_{1\le k\le T}[k-2^{\ell_k}=k']$ is bounded by $O(\log T)$ for any $k' \in \{1,\ldots,T\}$. Then the above expression is

$$= O\left(\sqrt{m}\log^2 m\log T\right)\sum_{k'=1}^{T}2^{\ell_{k'}}.$$

Since ℓ_k is the number of trailing zeros on k it can be at most $\log T$ for $k \le T$. We again rearrange the summation by possible values of $\ell_{k'}$ and note that there are at most $T/2^{i+1}$ numbers between 1 and T with i trailing zeros, so

$$\sum_{k'=1}^{T}2^{\ell_{k'}} = \sum_{i=0}^{\log T}2^i\cdot T/2^{i+1} = O(T\log T).$$

So the overall runtime is $O(\sqrt{m}T\log m\log^2 T)$, where $T = O(\sqrt{m}\log n\log(nM))$ as guaranteed by Theorem 4. Our data structure theorems assumed for convenience that solving an $m\times m$ SDD-system takes $\widetilde{O}(m)$ time; as discussed in Section A, an additional $\widetilde{O}(\log M)$ factor is incurred. Altogether, we conclude the overall runtime is $\widetilde{O}(m\log^2 M)$. □

4 Nested Dissection and Approximate Schur Complements

This section lays the foundation for a recursive decomposition of the input graph. Our goal is to set up the machinery to approximate $\mathbf{P}_{\boldsymbol{w}} \overset{\text{def}}{=} \mathbf{W}^{1/2}\mathbf{B}(\mathbf{B}^\top\mathbf{W}\mathbf{B})^\dagger\mathbf{B}^\top\mathbf{W}^{1/2}$ as needed in the robust IPM. In particular, we are interested in the weighted Laplacian matrix $\mathbf{L} \overset{\text{def}}{=} \mathbf{B}^\top\mathbf{W}\mathbf{B}$.

We begin with a discussion of nested dissection and the associated Schur complements.

4.1 Cholesky Decomposition and Schur Complement

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

$$\mathbf{L} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{L}_{C,F}(\mathbf{L}_{F,F})^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{L}_{F,F} & \mathbf{0} \\ \mathbf{0} & \mathbf{Sc}(\mathbf{L}, C) \end{bmatrix} \begin{bmatrix} \mathbf{I} & (\mathbf{L}_{F,F})^{-1}\mathbf{L}_{F,C} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}. \tag{4.1}$$

Note that $\mathbf{L}_{F,F}$ is full-rank. 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 12 (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 13. *Let $V(G) = \{v_1, \dots, v_n\}$. Let $C = V(G) - v_1$. Let $\mathbf{w}_{ij}$ denote the weight of edge v_iv_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 $\mathbf{w}_{ij} = \mathbf{w}_{1i}\mathbf{w}_{1j}/\mathbf{w}_1$, where $\mathbf{w}_1$ 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 14. *If $X \subseteq Y \subseteq V(G)$, then $\mathbf{Sc}(\mathbf{Sc}(\mathbf{L}, Y), X) = \mathbf{Sc}(\mathbf{L}, X)$.* □

Lemma 15. *Let $\mathbf{w}_e$ 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 $\mathbf{w}'_e \leq \mathbf{w}_e$ 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 16. *Let $\mathbf{w}_e$ 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 17. *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

$$\begin{aligned} \mathbf{Sc}(\mathbf{L}, C) &= \mathbf{Sc}(\mathbf{L}_1 + \mathbf{L}_2, C) \\ &= \mathbf{Sc}(\mathbf{Sc}(\mathbf{L}_1 + \mathbf{L}_2, C \cup V_2), C) \\ &= \mathbf{Sc}(\mathbf{Sc}(\mathbf{L}_1, C \cup V_2) + \mathbf{L}_2, C) && \text{(by Lemma 15)} \\ &= \mathbf{Sc}(\mathbf{Sc}(\mathbf{L}_1, C) + \mathbf{L}_2, C) && \text{(since } (C \cup V_2) \cap V_1 \subseteq C) \\ &= \mathbf{Sc}(\mathbf{L}_1, C) + \mathbf{Sc}(\mathbf{L}_2, C), && \text{(by Lemma 15)} \\ &= \mathbf{Sc}(\mathbf{L}_1, C \cap V_1) + \mathbf{Sc}(\mathbf{L}_2, C \cap V_2) && \text{(since } \mathbf{L}_i \text{ is supported on } V_i \text{ for } i = 1, 2) \end{aligned}$$

as desired. □

4.2 Separator Tree

In the overview, we briefly gave the intuition for a 2-level partition of the input graph; here we extend it to a recursive partitioning scheme with $O(\log n)$-levels. We begin with the formal definitions.

Definition 18 (Separable Graph). A subgraph-closed family of graphs $\mathcal{G}$ is *α-separable* if there exists two constants $c > 0$ and $b \in (0, 1)$ such that every $G \in \mathcal{G}$ with n vertices and $m \geq 2$ edges can be partitioned into H_1 and H_2 such that

— $E(H_1) \cup E(H_2) = E(G)$, $E(H_1) \cap E(H_2) = \emptyset$,
— $|V(H_1) \cap V(H_2)| \leq c\lceil n^{\alpha} \rceil$,
— $|E(H_i)| \leq bm$, for $i = 1, 2$.

We call $S(G) \overset{\text{def}}{=} V(H_1) \cap V(H_2)$ the *balanced vertex separator* of G.

It is known that planar graphs are $1/2$-separable.

Remark 19. As we discussed in Section 3.2, the IPM input graph is planar with two additional vertices and $O(n)$ additional edges incident to them. This graph is $1/2$-separable with the constant c in Definition 18 increased by 2.

We apply nested dissection recursively to each region using balanced vertex separators, until the regions are of constant size The resulting hierarchical structure can be represented by a tree $\mathcal{T}$, which is known as the *separator tree* of G:

Definition 20 (Separator Tree). A separator tree $\mathcal{T}$ of a graph G is a binary tree whose nodes represent subgraphs of G such that the children of each node H form a balanced partition of H.

Formally, each node of $\mathcal{T}$ is a *region* (edge-induced subgraph) H of G; we denote this by $H \in \mathcal{T}$ At a node H, we store subsets of vertices $\partial H, S(H), F_H \subseteq V(H)$ where ∂H is the set of *boundary vertices* that are incident to vertices outside H in G; $S(H)$ is the balanced vertex separator of H; and F_H is the set of *eliminated vertices* at H Concretely, the nodes and associated vertex sets are defined recursively in a top-down way as follows:

(1) The root of $\mathcal{T}$ is the node $H = G$, with $\partial H = \emptyset$ and $F_H = S(H)$.
(2) A non-leaf node $H \in \mathcal{T}$ has exactly two children $D_1, D_2 \in \mathcal{T}$ that form an edge-disjoint partition of H in Definition 18, and their vertex sets intersect on the balanced separator $S(H)$ of H. D_1 and D_2 does not have any isolated vertex. Define $\partial D_1 = (\partial H \cup S(H)) \cap V(D_1)$, and similarly $\partial D_2 = (\partial H \cup S(H)) \cap V(D_2)$. Define $F_H = S(H) \setminus \partial H$.
(3) If a region H contains a constant number of edges, then we stop the recursion and H becomes a leaf node. Furthermore, we define $S(H) = \emptyset$ and $F_H = V(H) \setminus \partial H$. Note that by construction, each edge of G is contained in a unique leaf node.

Let $\eta(H)$ denote the height of node H which is defined as the maximum number of edges on a tree path from H to one of its descendants. $\eta(H) = 0$ if H is a leaf. Note that the height difference between a parent and child node could be greater than one. Let η denote the height of $\mathcal{T}$ which is defined as the maximum height of nodes in $\mathcal{T}$. We say H is at *level* i if $\eta(H) = i$.

Observation 21. *Using the above definition, $\{F_H : H \in \mathcal{T}\}$ partitions the vertex set $V(G)$.*

Observation 22. *Suppose H is a node in $\mathcal{T}$ with children D_1 and D_2. We have $\partial D_1 \cup \partial D_2 = \partial H \cup F_H$.*

Observation 23. *Suppose H is a node in $\mathcal{T}$. Then $\partial H \subseteq \cup_{\text{ancestor } A \text{ of } H} F_A$.*

Fakcharoenphol and Rao [23] gave an algorithm that computes the separator tree for any planar graph.

Theorem 24 (Separator Tree Construction [23]). *Given a planar graph G there is an algorithm that computes a separator tree of G with height $\eta = O(\log n)$ in $O(n \log n)$ time*

For computing the separator tree $\mathcal{T}$ of our IPM input graph we may apply their method to the original planar graph to get the separator $\mathcal{T}'$, and add the two new vertices s, t to F_G at the root node G, and to the boundary sets ∂H at every non-root node H The additional edges incident to s, t can be recursively partitioned from a node to its children, which increases the height of $\mathcal{T}$ by $O(\log n)$ Thus, we have

Corollary 25. *There is an algorithm that computes a separator tree $\mathcal{T}$ for the IPM input graph with height $\eta = O(\log n)$ in $O(n \log n)$ time.*

To discuss the structures in the separator tree, we define the following terms:

Definition 26. Let $\mathcal{T}(i)$ be the subset of nodes in $\mathcal{T}$ at level i. For a node H, let $\mathcal{T}_H$ be the subtree of $\mathcal{T}$ rooted at H. Let $\mathcal{P}_{\mathcal{T}}(H)$ be the set nodes on the path from H to the root of $\mathcal{T}$, including H. Given a set of nodes $\mathcal{H} = \{H : H \in \mathcal{T}\}$, define

$$\mathcal{P}_{\mathcal{T}}(\mathcal{H}) := \bigcup_{H \in \mathcal{H}} \mathcal{P}_{\mathcal{T}}(H).$$

Finally, we partition these nodes by their level in $\mathcal{T}$, and use $\mathcal{P}_{\mathcal{T}}(\mathcal{H}, i)$ to denote all the nodes in $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$ at level i in $\mathcal{T}$.

Fakcharoenphol and Rao [23, Section 3.5] showed that for a set $\mathcal{H}$ of K nodes in T, the total number of boundary vertices from the nodes in $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$ is $O(\sqrt{mK})$. However, their claim is not stated as a result we can cite here We provide a simple, self-contained proof in the appendix of a slightly weaker bound that in addition requires bounding the number of separator vertices.

Lemma 27. *Suppose $\mathcal{T}$ is the appropriate separator tree for the IPM input graph. Let $\mathcal{H}$ be a set of K nodes in $\mathcal{T}$. Then*

$$\sum_{H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})} |\partial H| + |F_H| \leq \widetilde{O}\left(\sqrt{mK}\right).$$

4.3 Approximating $\mathbf{L}^{\dagger}$ Using Separator Tree

For a height-η separator tree $\mathcal{T}$, we generalize the sets C and F from the block Cholesky decomposition (Equation (4.1)) to a sequence of sets $C_0, \ldots, C_\eta$, and $F_0, \ldots, F_\eta$ based on $\mathcal{T}$.

Definition 28 (C_i, F_i). Let $\mathcal{T}$ be the separator tree from Corollary 25. 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. For all $0 \leq i \leq \eta$, define $C_i = \bigcup_{H \in \mathcal{T}(i)} \partial H$ to be the vertices remaining after eliminating vertices in F_i. We define C_{-1} to be $V(G)$.

By Observation 21, F_i is the disjoint union of F_H over all nodes H at level i in the separator tree. $F_0, \ldots, F_\eta$ partitions $V(G)$. 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 $V(G) = C_{-1} \supset C_0 \supset \cdots \supset C_{\eta-1} \supset C_\eta = \emptyset$ and $C_i = \cup_{j>i} F_j$.

Lemma 29. *By recursively applying Equation (4.1), we can decompose $\mathbf{L}$ as follows:*

$$\mathbf{L} = \mathbf{U}^{(0)\top} \cdots \mathbf{U}^{(\eta-1)\top} \begin{bmatrix} \mathbf{Sc}(\mathbf{L}, C_{-1})_{F_0,F_0} & \cdots & \mathbf{0} \\ \vdots & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{Sc}(\mathbf{L}, C_{\eta-1})_{F_\eta,F_\eta} \end{bmatrix} \mathbf{U}^{(\eta-1)} \cdots \mathbf{U}^{(0)}, \tag{4.2}$$

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, the projection matrix onto $\mathbf{L}$*'s image is* $\mathbf{P}_{\mathbf{L}} = \mathbf{I} - \mathbf{1}\mathbf{1}^\top/n$. □

Next, we consider approximating $\mathbf{L}$:

Definition 30 (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.

We will approximate $\mathbf{L}$ by approximating the terms in its decomposition (4.2). First, a piece of notation:

Definition 31 ($\mathbf{L}^{(H)}$). Let $\epsilon_{\mathbf{P}} > 0$. For each $H \in \mathcal{T}$, let $\mathbf{L}^{(H)}$ denote a Laplacian on the vertex set $F_H \cup \partial H$ such that

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

We show how to compute and maintain $\mathbf{L}^{(H)}$ in the next subsection Using these matrices at the nodes of $\mathcal{T}$, we have the following approximation of $\mathbf{L}$:

Lemma 32. *We have*

$$\mathbf{L} \approx_{\eta\epsilon_{\mathbf{P}}} \tilde{\mathbf{L}} \overset{\text{def}}{=} \tilde{\mathbf{U}}^{(0)\top} \cdots \tilde{\mathbf{U}}^{(\eta-1)\top} \tilde{\mathbf{T}} \tilde{\mathbf{U}}^{(\eta-1)} \cdots \tilde{\mathbf{U}}^{(0)}, \tag{4.3}$$

where

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

and[5]

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

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 28.

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

$$\mathbf{L}^{(i)}_{F_i,F_i} \overset{\text{def}}{=} \sum_{H\in\mathcal{T}(i)} \mathbf{L}^{(H)}_{F_H,F_H},$$
$$\mathbf{L}^{(i)}_{C_i,F_i} = \sum_{H\in\mathcal{T}(i)} \mathbf{L}^{(H)}_{C_i,F_i} = \sum_{H\in\mathcal{T}(i)} \mathbf{L}^{(H)}_{\partial H,F_H}.$$

Recall that the regions in $\mathcal{T}(i)$ partition the graph G. Furthermore, the intersection of $H, H' \in \mathcal{T}(i)$ is on their boundary, which is contained in $C_i \subseteq C_{i-1}$. Thus, we apply Lemma 17 to get

$$\begin{aligned} \mathbf{Sc}(\mathbf{L}, C_{i-1}) &= \sum_{H\in\mathcal{T}(i)} \mathbf{Sc}(\mathbf{L}[H], C_{i-1} \cap V(H)) \\ &\approx_{\epsilon_{\mathbf{P}}} \sum_{H\in\mathcal{T}(i)} \widetilde{\mathbf{Sc}}(\mathbf{L}[H], \partial H \cup F_H) = \sum_{H\in\mathcal{T}(i)} \mathbf{L}^{(H)} = \mathbf{L}^{(i)}. \end{aligned} \tag{4.4}$$

[5]when H is the root node, $(\mathbf{L}^{(H)}_{F_H,F_H})$ is rank-deficient, but we still use $^{-1}$ to denote the pseudo-inverse instead of † for notational convenience.

Now, we prove inductively that

$$\mathbf{L} \approx_{i\epsilon_{\mathbf{P}}} \tilde{\mathbf{U}}^{(0)\top} \cdots \tilde{\mathbf{U}}^{(i-1)\top} \begin{bmatrix} \mathbf{L}^{(0)}_{F_0,F_0} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \ddots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{L}^{(i-1)}_{F_{i-1},F_{i-1}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{L}^{(i)} \end{bmatrix} \tilde{\mathbf{U}}^{(i-1)} \cdots \tilde{\mathbf{U}}^{(0)}. \tag{4.5}$$

When $i = 0$, we have the approximation trivially as $\mathbf{L}^{(0)} = \mathbf{L}$.

For general i, we factor $\mathbf{L}^{(i)}$ in Equation (4.5) recursively using Cholesky decomposition $\mathbf{L}^{(i)}$ is supported on C_{i-1}, and we can partition $C_{i-1} = F_i \cup C_i$. Then,

$$\mathbf{L}^{(i)} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{L}^{(i)}_{C_i,F_i}(\mathbf{L}^{(i)}_{F_i,F_i})^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{L}^{(i)}_{F_i,F_i} & \mathbf{0} \\ \mathbf{0} & \mathbf{Sc}(\mathbf{L}^{(i)}, C_i) \end{bmatrix} \begin{bmatrix} \mathbf{I} & (\mathbf{L}^{(i)}_{F_i,F_i})^{-1}\mathbf{L}^{(i)}_{F_i,C_i} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}. \tag{4.6}$$

For the Schur complement term in the factorization, we have

$$\begin{aligned} \mathbf{Sc}(\mathbf{L}^{(i)}, C_i) &\approx_{i\epsilon_{\mathbf{P}}} \mathbf{Sc}(\mathbf{Sc}(\mathbf{L}, C_{i-1}), C_i) && \text{(by Equation (4.4))} \\ &= \mathbf{Sc}(\mathbf{L}, C_i) && \text{(by transitivity of Schur complements)} \\ &\approx_{\epsilon_{\mathbf{P}}} \mathbf{L}^{(i+1)}. && \text{(by Equation (4.4))} \end{aligned}$$

So we can use $\mathbf{L}^{(i+1)}$ in place of the Schur complement term in Equation (4.6), whose equality becomes an approximation with factor $\epsilon_{\mathbf{P}}$. At the ηth level, $\mathbf{L}^{(\eta)}_{F_\eta,F_\eta} = \mathbf{L}^{(\eta)}$ since $C_\eta = \emptyset$, so we have the overall expression.

To show $\tilde{\mathbf{L}}\mathbf{1} = \mathbf{0}$, we start with the fact that $\mathbf{L}^{(\eta)}$ is a Laplacian, so its rows and columns sum to zero. Substituting $\mathbf{L}^{(\eta)}$ in (4.6) for $i = \eta - 1$, we see that $\mathbf{L}^{(\eta-1)}$'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 $\tilde{\mathbf{U}}^{(\eta-1)} \cdots \tilde{\mathbf{U}}^{(0)}$ 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 $n - 1$. □

Before we consider the pseudo-inverse of $\mathbf{L}$, we need the following standard fact about the pseudo-inverse of a product of matrices:

Fact 33. *Suppose* $\mathbf{A}$ *is a symmetric matrix and* $\mathbf{X}$ *is a non-singular matrix, and* $\mathbf{P}$ *is the projection matrix onto the image of* $\mathbf{X}^\top\mathbf{A}\mathbf{X}$. *Then,*

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

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

Theorem 5 ($\mathbf{L}^\dagger$ Approximation). *Let* $\mathbf{P}_\mathbf{L}$ *be the projection onto the image of* $\mathbf{L}$. *Suppose for each* $H \in \mathcal{T}$, *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

$$\mathbf{L}^\dagger \approx_{\eta\epsilon_{\mathbf{P}}} \mathbf{P}_\mathbf{L}\mathbf{\Pi}^{(0)\top} \cdots \mathbf{\Pi}^{(\eta-1)\top}\widetilde{\mathbf{\Gamma}}\mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)}\mathbf{P}_\mathbf{L}, \tag{3.6}$$

where[6]

[6]Note that $(\mathbf{L}^{(H)}_{F_H,F_H})^\dagger$ is a pseudo-inverse when H is the root node; for notational convenience we use $^{-1}$ even for the pseudo-inverse.

$$\widetilde{\Gamma} = \begin{bmatrix} \sum_{H\in\mathcal{T}(0)} \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \sum_{H\in\mathcal{T}(\eta)} \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1} \end{bmatrix}$$

is a block-diagonal matrix, and

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

where $\mathcal{T}(i)$ denotes the set of nodes at level i of $\mathcal{T}$, and $\mathbf{I}$ is the $n \times n$ identity matrix.

Proof of Theorem 5. Let $\tilde{\mathbf{L}}$ denote the $\eta\epsilon_{\mathbf{P}}$ approximation of $\mathbf{L}$ in Lemma 32. Applying Fact 33, we have

$$\tilde{\mathbf{L}}^{\dagger} = \mathbf{P}_{\tilde{\mathbf{L}}} \left(\tilde{\mathbf{U}}^{(\eta-1)} \cdots \tilde{\mathbf{U}}^{(0)}\right)^{-1} \tilde{\mathbf{T}}^{\dagger} \left(\left(\tilde{\mathbf{U}}^{(\eta-1)} \cdots \tilde{\mathbf{U}}^{(0)}\right)^{-1}\right)^{\top} \mathbf{P}_{\tilde{\mathbf{L}}},$$

where $\tilde{\mathbf{T}}$ is the block diagonal matrix in Lemma 32. We note that $(\tilde{\mathbf{U}}^{(i)})^{-\top} = \Pi^{(i)}$, and $\tilde{\mathbf{T}}^{-1} = \widetilde{\Gamma}$. 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 $\tilde{\mathbf{L}}^{\dagger} \approx \mathbf{L}^{\dagger}$. □

Remark 34. Our overall goal is to efficiently maintain an approximation of the matrix

$$\mathbf{P}_{\boldsymbol{w}} \stackrel{\text{def}}{=} \mathbf{W}^{1/2}\mathbf{B}\mathbf{L}^{\dagger}\mathbf{B}^{\top}\mathbf{W}^{1/2}.$$

Since one can easily verify that $\mathbf{P}_{\mathbf{L}}\mathbf{B}^{\top} = \mathbf{B}^{\top}$, the projection matrix $\mathbf{P}_{\mathbf{L}}$ can be safely omitted from the data structure without affecting the approximation.

4.4 Recursive Schur Complements on Separator Tree

In this section, we prove Theorem 6 which maintains 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 35 (ApproxSchur Procedure [22]). *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 3. 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 36. *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 35.

ALGORITHM 3: Data structure to maintain dynamic approximate Schur complements

```
1: data structure DynamicSC
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($\boldsymbol{w}^{(\text{new})} \in \mathbb{R}^m$)
23:     $\mathcal{H} \leftarrow$ set of leaf nodes in $\mathcal{T}$ that contain an edge $e$ whose weight is updated
24:     $\boldsymbol{w} \leftarrow \boldsymbol{w}^{(\text{new})}$
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:
32: procedure ApproxSchurNode($H \in \mathcal{T}$)
33:     if $H$ is a leaf node then
34:         $\mathbf{L}^{(H)} \leftarrow (\mathbf{B}[H])^\top \mathbf{W}_{E(H)} \mathbf{B}[H]$
35:         $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H) \leftarrow$ ApproxSchur($\mathbf{L}^{(H)}, \partial H, \delta$)          ▷ Lemma 35
36:     else
37:         Let $D_1, D_2$ be the children of $H$
38:         $\mathbf{L}^{(H)} \leftarrow \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)}, \partial D_1) + \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)}, \partial D_2)$
39:         $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H) \leftarrow$ ApproxSchur($\mathbf{L}^{(H)}, \partial H, \delta$)
40:     end if
41: end procedure
```

Otherwise, suppose H is at level i with children D_1 and D_2. By construction of the separator tree and Observation 22, 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 D_j may or may not equal to $i - 1$ but it is guaranteed to be no more than $i - 1$.) Then

$$\begin{aligned}
\mathbf{L}^{(H)} &= \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)}, \partial D_1) + \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)}, \partial D_2) \\
&\approx_{(i-1)\delta} \mathbf{Sc}(\mathbf{L}[D_1], \partial D_1) + \mathbf{Sc}(\mathbf{L}[D_2], \partial D_2) \\
&= \mathbf{Sc}(\mathbf{L}[D_1], (\partial H \cup F_H) \cap V(D_1)) + \mathbf{Sc}(\mathbf{L}[D_2], (\partial H \cup F_H) \cap V(D_2)) \\
&\quad \text{(by construction of the separator tree, } \partial D_j = (\partial H \cup F_H) \cap V(D_j) \text{ for } j = 1, 2) \\
&= \mathbf{Sc}(\mathbf{L}[H], \partial H \cup F_H), \qquad \text{(by Lemma 17)}
\end{aligned}$$

so we have the second correctness property.

Line 39 returns $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ with $\widetilde{O}(\delta^{-2}|\partial H|)$ edges by Lemma 35. Also,

$$\begin{aligned}
\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H) &\approx_{\delta} \mathbf{Sc}(\mathbf{L}^{(H)}, \partial H) \\
&\approx_{(i-1)\delta} \mathbf{Sc}(\mathbf{Sc}(\mathbf{L}[H], \partial H \cup F_H), \partial H) \\
&= \mathbf{Sc}(\mathbf{L}[H], \partial H), \qquad \text{(by Lemma 14)}
\end{aligned}$$

giving us the third correctness property. □

Lemma 37. *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 35.

Otherwise, when H has children D_1, D_2, computing $\mathbf{L}^{(H)}$ requires summing together $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_j)}, \partial D_j)$ for $j = 1, 2$, 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 35. □

Next, we prove the overall data structure correctness and runtime:

Theorem 6 (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 3) 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* $\mathbf{w}$ *from the IPM. The data structure supports the following procedures:*

- Initialize$(\mathcal{T}, \mathbf{w} \in \mathbb{R}^m_{>0}, \epsilon_{\mathbf{P}} > 0)$: *Given the separator tree* $\mathcal{T}$*, initial weights* $\mathbf{w}$*, target step accuracy* $\epsilon_{\mathbf{P}}$*, preprocess in* $\widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}m)$ *time.*
- Reweight$(\Delta\mathbf{w} \in \mathbb{R}^m_{>0})$: *Update the weights to* $\mathbf{w} + \Delta\mathbf{w}$*, and recompute the relevant Schur complements in* $\widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}\sqrt{mK})$ *time, where* $K = \mathrm{nnz}(\Delta\mathbf{w})$.
  If $\mathcal{H}$ *is the set of leaf nodes in* $\mathcal{T}$ *that contain an edge whose weight is updated, then* $\mathbf{L}^{(H)}$ *and* $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ *are updated only for nodes* $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$.
- *Access to Laplacian* $\mathbf{L}^{(H)}$ *at any node* $H \in \mathcal{T}$ *in* $\widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}|F_H \cup \partial H|)$ *time.*
- *Access to Laplacian* $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ *at any node* $H \in \mathcal{T}$ *in* $\widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}|\partial H|)$ *time.*

Furthermore, at all points during the IPM,

$$\mathbf{L}^{(H)} \approx_{\epsilon_{\mathbf{P}}} \mathbf{Sc}(\mathbf{L}[H], F_H \cup \partial H) \quad \text{and} \quad \widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H) \approx_{\epsilon_{\mathbf{P}}} \mathbf{Sc}(\mathbf{L}[H], \partial H) \tag{3.7}$$

for all $H \in \mathcal{T}$ *with high probability.*

Proof of Theorem 6. Because we set $\delta \leftarrow \epsilon_{\mathbf{P}}/(\eta + 1)$ in Initialize, combined with Lemma 36, 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. Computing the separator tree costs $O(n \log n)$ time by Corollary 25. 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 36. The runtime of Initialize is bounded by running ApproxSchurNode on each node, i.e:

$$\widetilde{O}\left(\delta^{-2} \sum_{H \in \mathcal{T}} |\partial H \cup F_H|\right) = \widetilde{O}\left(\delta^{-2} m\right) = \widetilde{O}\left({\epsilon_{\mathbf{P}}}^{-2} m\right).$$

Where we bound the sum using Lemma 27 with $K = O(m)$, since $\mathcal{T}$ has $O(m)$ nodes in total.

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 $\widetilde{O}(\delta^{-2} \sum_{H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})} |\partial H \cup F_H|)$. By Lemma 27, this is further bounded by $\widetilde{O}({\epsilon_{\mathbf{P}}}^{-2}\sqrt{mK})$.

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. □

5 Maintaining the Implicit Representation

In this section, we give a general data structure MaintainRep. Since the completion of this manuscript, a significantly simplified version of this section has appeared in [19].

At a high level, MaintainRep implicitly maintains a vector $\boldsymbol{x}$ throughout the IPM by explicitly maintaining vector $\boldsymbol{y}$, and implicitly maintaining a *tree operator* $\mathbf{M}$ and vector $\boldsymbol{z}$, with $\boldsymbol{x} \overset{\text{def}}{=} \boldsymbol{y} + \mathbf{M}\boldsymbol{z}$. MaintainRep supports the IPM operations Move and Reweight as follows: To move in step k with direction $\boldsymbol{v}^{(k)}$ and step size $\alpha^{(k)}$ the data structure computes some $\boldsymbol{z}^{(k)}$ from $\boldsymbol{v}^{(k)}$ and updates $\boldsymbol{x} \leftarrow \boldsymbol{x} + \mathbf{M}(\alpha^{(k)}\boldsymbol{z}^{(k)})$ To reweight with new weights $\boldsymbol{w}^{(\text{new})}$ (which does not change the value of $\boldsymbol{x}$) the data structure computes $\mathbf{M}^{(\text{new})}$ using $\boldsymbol{w}^{(\text{new})}$, updates $\mathbf{M} \leftarrow \mathbf{M}^{(\text{new})}$, and updates $\boldsymbol{y}$ to offset the change in $\mathbf{M}\boldsymbol{z}$. In Section 5.1, we define $\boldsymbol{z}^{(k)}$ and show how to maintain $\boldsymbol{z} = \sum_{i=1}^{k} \boldsymbol{z}^{(i)}$ efficiently. In Section 5.2, we define tree operators. Finally in Section 5.3, we implement MaintainRep for a general tree operator $\mathbf{M}$.

Our goal is for this data structure to maintain the updates to the slack and flow solutions at every IPM step Recall at step k, we want to update the slack solution by $\bar{t}h\mathbf{W}^{1/2}\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}^{(k)}$ and the partial flow solution by $h\mathbf{W}^{-1/2}\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v}^{(k)}$. In later sections, we define specific tree operators $\mathbf{M}^{(\text{slack})}$ and $\mathbf{M}^{(\text{flow})}$ so that the slack and flow updates can be written as $\mathbf{M}^{(\text{slack})}(\bar{t}h\boldsymbol{z}^{(k)})$ and $\mathbf{M}^{(\text{flow})}(h\boldsymbol{z}^{(k)})$, respectively. This then allows us to use two copies of MaintainRep to maintain the solutions throughout the IPM.

To start, recall the information stored in the DynamicSC data structure: at every node H we have weighted Laplacian $\mathbf{L}^{(H)}$. In the previous section, we defined matrices $\widetilde{\mathbf{\Gamma}}$ and $\mathbf{\Pi}^{(i)}$'s as functions of the $\mathbf{L}^{(H)}$'s, in order to approximate $\mathbf{L}^{\dagger}$. MaintainRep will contain a copy of the DynamicSC data structure; therefore, the remainder of this section will freely refer to $\widetilde{\mathbf{\Gamma}}$ and $\mathbf{\Pi}^{(0)}, \ldots, \mathbf{\Pi}^{(\eta-1)}$.

5.1 Maintaining the Intermediate Vector z

We define a partial computation at each step of the IPM, which will be shared by both the slack and flow solutions:

Definition 38 ($\boldsymbol{z}^{(k)}$). At the kth step of the IPM, let $\boldsymbol{v}^{(k)}$ be the step direction. Let $\boldsymbol{d} \overset{\text{def}}{=} \mathbf{B}^{\top}\mathbf{W}^{1/2}\boldsymbol{v}^{(k)}$. Define $\boldsymbol{z}^{(k)}$ to be the partial computation

$$\boldsymbol{z}^{(k)} \overset{\text{def}}{=} \widetilde{\mathbf{\Gamma}}\mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)}\boldsymbol{d}. \tag{5.1}$$

ALGORITHM 4: Data structure to maintain the intermediate vector z, Part 1

```
data structure MaintainZ
private: member
    $\mathcal{T}$: separator tree of input graph $G$ of height $\eta$
    $c \in \mathbb{R}, z^{(\text{step})}, z^{(\text{sum})} \in \mathbb{R}^n$: coefficient and vectors to be maintained
    $u \in \mathbb{R}^n$: vector to be maintained such that $u = \Pi^{(\eta-1)} \cdots \Pi^{(0)} \mathbf{B}^\top \mathbf{W} v$
    $v \in \mathbb{R}^m$: direction vector from the current step
    $w \in \mathbb{R}^m$: weight vector                              ▷ we also use $\mathbf{W} \overset{\text{def}}{=} \text{diag}(w)$
    DynamicSC: data struct which gives access to $\mathbf{L}^{(H)}$ for each $H \in \mathcal{T}$

procedure Initialize($\mathcal{T}, v \in \mathbb{R}^m, w \in \mathbb{R}^m, \epsilon_{\mathbf{P}}$)
    DynamicSC.Initialize($\mathcal{T}, w, \epsilon_{\mathbf{P}}$)
    $w \leftarrow w, v \leftarrow v$
    $u \leftarrow$ PartialProject($\mathbf{B}^\top \mathbf{W}^{1/2} v$)
    $z^{(\text{step})} \leftarrow \widetilde{\Gamma} u$
    $z^{(\text{sum})} \leftarrow \mathbf{0}$
    $c \leftarrow 0$
end procedure

procedure PartialProject($d \in \mathbb{R}^n, \mathcal{H} = \{H \in \mathcal{T} : d|_{F_H} \neq \mathbf{0}\}$)
                    ▷ if $\mathcal{H}$ is not given in the argument, then it takes the default value above
    $u \leftarrow d$
    for $i$ from 0 to $\eta - 1$ do
        $u \leftarrow u - \sum_{H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H}, i)} \mathbf{L}^{(H)}_{\partial H, F_H} (\mathbf{L}^{(H)}_{F_H, F_H})^{-1} \cdot u|_{F_H}$
    end for
    return $u$
end procedure

procedure InversePartialProject($u \in \mathbb{R}^n, \mathcal{H}$)
    for $i$ from $\eta - 1$ to 0 do
        $u \leftarrow u + \sum_{H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H}, i)} \mathbf{L}^{(H)}_{\partial H, F_H} (\mathbf{L}^{(H)}_{F_H, F_H})^{-1} \cdot u|_{F_H}$
    end for
    $d \leftarrow u$
    return $d$
end procedure
```

Observe that this is a partial projection: If we apply $\mathbf{W}^{1/2}\mathbf{B}\Pi^{(0)\top} \cdots \Pi^{(\eta-1)\top}$ to $z^{(k)}$, then the result is an approximation to $\mathbf{P}_w v^{(k)}$.

We first show how to multiply $\widetilde{\Gamma}\Pi^{(\eta-1)} \cdots \Pi^{(0)}$ to a vector efficiently. The main idea is to take advantage of the hierarchical structure of the separator tree $\mathcal{T}$ in a bottom-up fashion. If d is a sparse vector with only K non-zero entries, then we can apply the operator while avoiding exploring parts of $\mathcal{T}$ that are guaranteed to contain zero values.

For notational convenience, let us define $\mathbf{X}^{(H)} \overset{\text{def}}{=} \mathbf{L}^{(H)}_{\partial H, F_H} (\mathbf{L}^{(H)}_{F_H, F_H})^{-1}$.

Lemma 39. *Given a vector $d \in \mathbb{R}^n$, let $\mathcal{H} \supseteq \{H \in \mathcal{T} : d|_{F_H} \neq \mathbf{0}\}$ and suppose $|\mathcal{H}| = K$. Then the procedure PartialProject$(d, \mathcal{H})$ in the MaintainZ data structure (Algorithm 4) returns the vector*

$$u = \Pi^{(\eta-1)} \cdots \Pi^{(1)} \Pi^{(0)} d,$$

where the $\Pi^{(i)}$'s and $\epsilon_{\mathbf{P}}$ are from the DynamicSC data structure in MaintainZ.

The procedure runs in $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK})$ time, and $\boldsymbol{u}|_{F_H}$ is non-zero for at most $\widetilde{O}(K)$ nodes $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$.

Proof. First, we consider the runtime. We remark that the creation of vector $\boldsymbol{u}$ is for readability; the procedure can in fact be computed using $\boldsymbol{d}$ in-place.

The bottleneck of PartialProject is Line 21. For each $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$, recall from Theorem 6 that $\mathbf{L}^{(H)}$ is supported on the vertex set $F_H \cup \partial H$ and has $\widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}|F_H \cup \partial H|)$ edges. Hence, $(\mathbf{L}^{(H)}_{F_H,F_H})^{-1}\boldsymbol{u}|_{F_H}$ can be computed by a high-accuracy Laplacian solver in $\widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}|F_H \cup \partial H|)$ time, and the subsequent left-multiplying by $\mathbf{L}^{(H)}_{\partial H,F_H}$ also takes $\widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}|F_H \cup \partial H|)$ time. Finally, we can add the resulting vector to $\boldsymbol{u}$ in time linear in the sparsity. Summing this over all $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$, we get that the total runtime is $\widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}\sqrt{mK})$ by Lemma 27.

To show the correctness of PartialProject, we have the following claim:

Claim 40. *Let $\boldsymbol{u}^{(-1)} = \boldsymbol{d}$ be the value of $\boldsymbol{u}$ in PartialProject$(\boldsymbol{d}, \mathcal{H})$ before the first double for-loop. Let $\boldsymbol{u}^{(i)}$ be the value of $\boldsymbol{u}$ after iteration i of the outer loop (Line 20) for $0 \le i < \eta$. Then*

$$\boldsymbol{u}^{(i)} = \mathbf{\Pi}^{(i)} \cdots \mathbf{\Pi}^{(0)}\boldsymbol{d}.$$

Furthermore, $\boldsymbol{u}^{(i)}|_{F_H} \neq \mathbf{0}$ only if $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$.

Proof. We prove the claim by induction.

For $i = -1$, we are given $\boldsymbol{u}^{(-1)}|_{F_H} = \boldsymbol{d}|_{F_H} \neq \mathbf{0}$ exactly for all $H \in \mathcal{H} \subseteq \mathcal{P}_{\mathcal{T}}(\mathcal{H})$. For $i+1$, we have, by inductive hypothesis and definition of $\mathbf{\Pi}^{(i)}$ from Theorem 5:

$$\begin{aligned}\mathbf{\Pi}^{(i+1)}\mathbf{\Pi}^{(i)} \cdots \mathbf{\Pi}^{(0)}\boldsymbol{d} &= \mathbf{\Pi}^{(i+1)}\boldsymbol{u}^{(i)} \\ &= \left(\mathbf{I} - \sum_{H \in \mathcal{T}(i+1)} \mathbf{X}^{(H)}\right)\boldsymbol{u}^{(i)}.\end{aligned}$$

Since $\mathbf{X}^{(H)} \in \mathbb{R}^{\partial H \times F_H}$ and $\boldsymbol{u}^{(i)}|_{F_H} \neq \mathbf{0}$ only if $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$, the summation above can be taken over the smaller set $\mathcal{T}(i+1) \cap \mathcal{P}_{\mathcal{T}}(\mathcal{H}) \overset{\text{def}}{=} \mathcal{P}_{\mathcal{T}}(\mathcal{H}, i+1)$, giving

$$= \boldsymbol{u}^{(i)} - \sum_{H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H}, i+1)} \mathbf{X}^{(H)}\boldsymbol{u}^{(i)}|_{F_H}.$$

This is exactly what is computed as $\boldsymbol{u}$ after iteration i of the outer loop at Line 20. Hence, this is equal to $\boldsymbol{u}^{(i+1)}$ by definition.

For the sparsity condition, we note that if $\boldsymbol{u}^{(i+1)}|_{F'_H}$ differs from $\boldsymbol{u}^{(i)}|_{F'_H}$ at a node H', then it was changed by a term in the summation above, and so we must have $F_{H'} \cap \partial H \neq \emptyset$ for some $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H}, i+1)$. By construction of the separator tree, this occurs only if H' is an ancestor of H, which implies $H' \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$. Combined with the inductive hypothesis, we have that $\boldsymbol{u}^{(i+1)}|_{F_H} \neq \mathbf{0}$ only if $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$. □

Setting $i = \eta - 1$ in the above claim immediately shows that at the end of the first double for-loop in PartialProject, we have $\boldsymbol{u} = \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d}$.

Finally, to complete the sparsity argument, we have $|\mathcal{H}| = K$, and consequently $|\mathcal{P}_{\mathcal{T}}(\mathcal{H})| = O(K \cdot \eta) = \widetilde{O}(K)$. Combined with the claim, we get the overall sparsity guarantee. □

For the correctness of our data structure, we will need a more specific structural property of PartialProject:

Lemma 41. *Let $\mathcal{H}$ be any subset of nodes in $\mathcal{T}$. Let $H_1, \ldots, H_r$ be any permutation of all nodes from $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$ such that if H_i is an ancestor of H_j, then $i < j$. Then*

$$\textit{PartialProject}(\boldsymbol{d}, \mathcal{H}) = (\mathbf{I} - \mathbf{X}^{(H_1)}) \ldots (\mathbf{I} - \mathbf{X}^{(H_r)})\boldsymbol{d}.$$

ALGORITHM 4: Data structure to maintain the intermediate vector z, Part 2

```
33: procedure Reweight($\Delta w \in \mathbb{R}^m$)
34:     $w^{(\text{new})} \overset{\text{def}}{=} w + \Delta w$
35:     $\mathcal{H} \leftarrow$ set of leaf nodes in $\mathcal{T}$ that contain all edges whose weight has changed
36:     $\Delta u \leftarrow$ PartialProject($\mathbf{B}^\top(\mathbf{W}^{(\text{new})1/2} - \mathbf{W}^{1/2})v$)
37:     $w \leftarrow w + \Delta w$
38:     $u \leftarrow u + \Delta u$
39:     $d \leftarrow$ InversePartialProject($u, \mathcal{H}$)        ▷ revert projection with old weights
40:     dynamicSC.Reweight($w$)        ▷ update $\mathbf{L}^{(H)}$'s to use the new weights
41:                                    ▷ specifically, $\mathbf{L}^{(H)}$ changes for each $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$
42:     $u \leftarrow$ PartialProject($d, \mathcal{H}$)        ▷ apply projection with new weights
43:     $y \leftarrow z^{(\text{step})}$        ▷ backup copy of $z^{(\text{step})}$
44:     for $H$ in $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$ do
45:         $z^{(\text{step})}|_{F_H} \leftarrow (\mathbf{L}^{(H)}_{F_H,F_H})^{-1} u|_{F_H}$
46:     end for
47:     $z^{(\text{sum})} \leftarrow z^{(\text{sum})} - c \cdot (z^{(\text{step})} - y)$        ▷ update $z^{(\text{sum})}$ to maintain the invariant
48: end procedure
49:
50: procedure Move($\alpha \in \mathbb{R}, \Delta v \in \mathbb{R}^m$)
51:     $v \leftarrow v + \Delta v$
52:     $\Delta u \leftarrow$ PartialProject($\mathbf{B}^\top \mathbf{W}^{1/2} \Delta v$)
53:     $u \leftarrow u + \Delta u$
54:     $y \leftarrow z^{(\text{step})}$        ▷ backup copy of $z^{(\text{step})}$
55:     for $H$ in $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$ do
56:         $z^{(\text{step})}|_{F_H} \leftarrow (\mathbf{L}^{(H)}_{F_H,F_H})^{-1} u|_{F_H}$
57:     end for
58:     $z^{(\text{sum})} \leftarrow z^{(\text{sum})} - c \cdot (z^{(\text{step})} - y)$
59:     $c \leftarrow c + \alpha$
60: end procedure
```

Proof. First, we observe that $\mathbf{I}-\mathbf{X}^{(H_i)}$ and $\mathbf{I}-\mathbf{X}^{(H_j)}$ are commutative if H_i and H_j are not ancestor-descendants. The reason is that $\mathbf{X}^{(H_i)}\mathbf{X}^{(H_j)} = \mathbf{0}$, since $\mathbf{X}^{(H_i)} \in \mathbb{R}^{\partial H_i \times F_{H_i}}$, and $F_{H_i} \cap \partial H_j \neq \emptyset$ only if H_i is an ancestor of H_j.

From the proof of Claim 40, we observe that iteration i of the for-loop in PartialProject applies the operator

$$\mathbf{I} - \sum_{H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H},i)} \mathbf{X}^{(H)} = \prod_{H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H},i)} (\mathbf{I} - \mathbf{X}^{(H)}),$$

where the equality follows from expanding the RHS and applying the property $\mathbf{X}^{(H_i)}\mathbf{X}^{(H_j)} = \mathbf{0}$. Thus, we have a stricter version of the claim:

$$\text{PartialProject}(d, \mathcal{H}) = (\mathbf{I} - \mathbf{X}^{(H_1)}) \ldots (\mathbf{I} - \mathbf{X}^{(H_r)})d,$$

where $H_1, \ldots, H_r$ is any permutation of $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$ such that nodes at lower levels come later. Then we apply commutativity to allow $H_1, \ldots, H_r$ to be any permutation such that if H_i is an ancestor of H_j then $i < j$. □

Next, we show there is a procedure that reverses PartialProject using select nodes of $\mathcal{T}$.

Lemma 42. *Given a set of K nodes $\mathcal{H}$ in $\mathcal{T}$ and a vector u, InversePartialProject($u, \mathcal{H}$) in the MaintainZ data structure (Algorithm 4) is a procedure that returns d such that*

$$\boldsymbol{d} = \left(\mathbf{I} + \mathbf{X}^{(H_r)}\right) \cdots \left(\mathbf{I} + \mathbf{X}^{(H_1)}\right) \boldsymbol{u},$$

where $H_1, \ldots, H_r$ is any permutation of all nodes from $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$ such that if H_i is an ancestor of H_j, then $i < j$ The procedure runs in $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK})$ time, where $K = |\mathcal{H}|$.

Proof. Intuitively, observe that InversePartialProject is reversing all the operations in PartialProject. The runtime analysis is analogous to PartialProject. The proof of the equation is also analogous to PartialProject. We first observe that iteration i of the for-loop applies the operator

$$\mathbf{I} + \sum_{H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H}, i)} \mathbf{X}^{(H)} = \prod_{H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H}, i)} \left(\mathbf{I} + \mathbf{X}^{(H)}\right).$$

Then by commutativity as in Lemma 41, we have

$$\boldsymbol{d} = \left(\mathbf{I} + \mathbf{X}^{(H_r)}\right) \cdots \left(\mathbf{I} + \mathbf{X}^{(H_1)}\right) \boldsymbol{u}.$$

where $H_1, \ldots, H_r$ is any permutation of $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$ such that nodes at lower levels come later. Then we apply commutativity to allow $H_1, \ldots, H_r$ to be any permutation such that if H_i is an ancestor of H_j then $i < j$. □

Finally, we have the data structure for maintaining a vector $\boldsymbol{z}$ dependent on $\boldsymbol{v}$ throughout the IPM. For one IPM step, there is one call to Reweight followed by one call to Move.

Theorem 43 (Maintain Intermediate Vector $\boldsymbol{z}$). *Given an appropriate separator tree $\mathcal{T}$ of the IPM input graph with height η, the deterministic data structure MaintainZ (Algorithm 4) maintains the following variables correctly at the end of each IPM step:*

- *the dynamic edge weights $\boldsymbol{w}$ is and current step direction $\boldsymbol{v}$ from the IPM*
- *a DynamicSC data structure on $\mathcal{T}$ based on the current edge weights $\boldsymbol{w}$*
- *scalar c and vectors $\boldsymbol{z}^{(\text{step})}, \boldsymbol{z}^{(\text{sum})}$, which together represent $\boldsymbol{z} = c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})}$, such that at the end of IPM step k,*

$$\boldsymbol{z} = \sum_{i=1}^{k} \boldsymbol{z}^{(i)}. \tag{5.2}$$

- *$\boldsymbol{z}^{(\text{step})}$ satisfies $\boldsymbol{z}^{(\text{step})} = \widetilde{\boldsymbol{\Gamma}}\boldsymbol{\Pi}^{(\eta-1)} \cdots \boldsymbol{\Pi}^{(0)}\mathbf{B}^{\top}\mathbf{W}^{1/2}\boldsymbol{v}$.*

The data structure supports the following procedures:

- *Initialize($\mathcal{T}, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \epsilon_{\mathbf{P}} > 0$): Given a graph G, its separator tree $\mathcal{T}$, initial step direction $\boldsymbol{v}$, initial weights $\boldsymbol{w}$, and target step accuracy $\epsilon_{\mathbf{P}}$, preprocess in $\widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}m)$ time and initialize $\boldsymbol{z} = \mathbf{0}$.*
- *Reweight($\boldsymbol{w} \in \mathbb{R}^m_{>0}$ given implicitly as a set of changed coordinates): Update the current weight to $\boldsymbol{w}$ and update DynamicSC, and update the representation of $\boldsymbol{z}$. The procedure runs in $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK})$ total time, where K is the number of coordinates updated in $\boldsymbol{w}$. There are most $\widetilde{O}(K)$ nodes $H \in \mathcal{T}$ for which $\boldsymbol{z}^{(\text{step})}|_{F_H}$ and $\boldsymbol{z}^{(\text{sum})}|_{F_H}$ are updated.*
- *Move($\alpha \in \mathbb{R}$, $\boldsymbol{v} \in \mathbb{R}^n$ given implicitly as a set of changed coordinates): Update the current direction to $\boldsymbol{v}$, and set $\boldsymbol{z} \leftarrow \boldsymbol{z} + \alpha\widetilde{\boldsymbol{\Gamma}}\boldsymbol{\Pi}^{(\eta-1)} \cdots \boldsymbol{\Pi}^{(0)}\mathbf{B}^{\top}\mathbf{W}^{1/2}\boldsymbol{v}$ with the correct representation. The procedure runs in $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK})$ time, where K is the number of coordinates changed in $\boldsymbol{v}$ compared to the previous IPM step.*

Proof. If Move is implemented correctly, then by the definition of the update to $\boldsymbol{z}$, the invariant in Equation (5.2) is correctly maintained.

For the runtime analysis, recall $\{F_H : H \in \mathcal{T}\}$ partition the vertex set of G. Therefore, $\boldsymbol{v}$ has K non-zero entries, then $\boldsymbol{d} \overset{\text{def}}{=} \mathbf{B}^\top \mathbf{W}^{1/2}\boldsymbol{v}$ has $O(K)$ non-zero entries, and consequently $\boldsymbol{d}|_{F_H} \neq \mathbf{0}$ for $O(K)$ nodes H. There are $O(m)$ total nodes in the separator tree $\mathcal{T}$.

We maintain a vector $\boldsymbol{u}$ with the invariant $\boldsymbol{u} = \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)}\mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}$. We now prove the correctness and runtime of each procedure separately.

Initialize: By the guarantee of Lemma 39, at the end of Initialize, we have

$$\boldsymbol{u} = \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)}\mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}$$

and

$$\boldsymbol{z}^{(\text{step})} = \widetilde{\mathbf{\Gamma}}\boldsymbol{u} = \widetilde{\mathbf{\Gamma}}\mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)}\mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}.$$

Since c and $\boldsymbol{z}^{(\text{sum})}$ are initialized to zero, we have $\boldsymbol{z} = c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})} = \mathbf{0}$.

We initialize the DynamicSC data structure in $(\epsilon_{\mathbf{P}}{}^{-2}m)$ time. There is no sparsity guarantee for $\boldsymbol{v}$, but the call to PartialProject takes at most $O(\epsilon_{\mathbf{P}}{}^{-2}m)$ time because of the size of $\mathcal{T}$. To calculate $\widetilde{\mathbf{\Gamma}}\boldsymbol{u}$, we solve a Laplacian system $(\mathbf{L}^{(H)}_{F_H,F_H})^{-1}\boldsymbol{u}|_{F_H}$ in time $\widetilde{O}(|\mathbf{L}^{(H)}|)$ for each node H. The total time is $\widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}m)$ as well by $|\mathbf{L}^{(H)}| = \widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}|F_H \cup \partial H|)$ from Theorem 6 and by Lemma 27.

Move: Let $\boldsymbol{v}, \boldsymbol{u}$ be the variables at the start of Move, and let $\boldsymbol{v}', \boldsymbol{u}'$ denote them at the end. Similarly, let $\boldsymbol{z} = c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})}$ denote $\boldsymbol{z}$ and the respective variables at the start of Move, and let $\boldsymbol{z}' = c'\boldsymbol{z}^{(\text{step})'} + \boldsymbol{z}^{(\text{sum})'}$ denote these variables at the end.

First, after Line 53, we have

$$\begin{aligned}\boldsymbol{u}' &= \boldsymbol{u} + \Delta\boldsymbol{u}\\ &= \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)}\mathbf{B}^\top\mathbf{W}^{1/2}(\boldsymbol{v} + \Delta\boldsymbol{v})\\ &= \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)}\mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}',\end{aligned}$$

where the second equality follows from the guarantee of PartialProject and the guarantee from the previous IPM step. By Lemma 39, $\boldsymbol{u}'$ is updated only on F_H where $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$. Thus, to update $\boldsymbol{z}^{(\text{step})'} = \widetilde{\mathbf{\Gamma}}\boldsymbol{u}'$, we only need to update $\boldsymbol{z}^{(\text{step})'}|_{F_H}$ for $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$, which happens on Line 56 Observe that the update in value to $\boldsymbol{z}^{(\text{step})}$ is cancelled out by the update in $\boldsymbol{z}^{(\text{sum})}$ at Line 58, so that the value of $\boldsymbol{z}$ does not change overall up to that point But we have

$$\boldsymbol{z} = c\boldsymbol{z}^{(\text{step})'} + \boldsymbol{z}^{(\text{sum})'} = c\widetilde{\mathbf{\Gamma}}\mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)}\mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}' + \boldsymbol{z}^{(\text{sum})'}.$$

Then in Line 59, incrementing c by α represents increasing the value of $\boldsymbol{z}$ by $\alpha\boldsymbol{z}^{(\text{step})'}$, which is exactly the desired update.

For the runtime, first note $nnz(\Delta\boldsymbol{v}) = K$. So PartialProject runs in $\widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}\sqrt{mK})$ time by Lemma 39. Line 56 takes $\widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}\sqrt{mK})$ time in total by Theorem 6 and Lemma 27. The remaining operations in the procedure are adding vectors with bounded sparsity.

Reweight: Let $\boldsymbol{w}^{(\text{old})}$ denote the weight vector immediately before this procedure is called, and $\boldsymbol{w}^{(\text{new})}$ is the new weight passed in as an argument.

Let $\widetilde{\mathbf{\Gamma}}$ and $\mathbf{\Pi}^{(i)}$ denote these matrices defined using the old weights, and let $\widetilde{\mathbf{\Gamma}}'$ and $\mathbf{\Pi}^{(i)'}$ denote the matrices using the new weights. Similarly let $\boldsymbol{u}$ be the state of the vector at the start of the procedure call and $\boldsymbol{u}'$ at the end.

In Reweight, we do not change the value of $\boldsymbol{z}$, but rather update $\boldsymbol{z}^{(\text{step})}$ and $\boldsymbol{z}^{(\text{sum})}$ so that at the end of the procedure,

$$\boldsymbol{z}^{(\text{step})} = \widetilde{\mathbf{\Gamma}}'\mathbf{\Pi}^{(\eta-1)'} \cdots \mathbf{\Pi}^{(0)'}\mathbf{B}^\top\mathbf{W}^{(\text{new})1/2}\boldsymbol{v},$$

so that we maintain the invariant claimed in the theorem statement.

To see that the value of $\boldsymbol{z}$ does not change at the end of the procedure, observe that we modify $\boldsymbol{z}^{(\mathrm{step})}$ during the procedure, and cancel all the changes to $\boldsymbol{z}^{(\mathrm{step})}$ by updating $\boldsymbol{z}^{(\mathrm{sum})}$ appropriately at the last line (Line 47).

Immediately before Line 38, the algorithm invariant guarantees

$$\boldsymbol{u} = \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)} \mathbf{B}^\top \mathbf{W}^{(\mathrm{old})1/2} \boldsymbol{v}.$$

By Lemma 39,

$$\Delta \boldsymbol{u} = \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)} \mathbf{B}^\top (\mathbf{W}^{(\mathrm{new})1/2} - \mathbf{W}^{(\mathrm{old})1/2}) \boldsymbol{v}.$$

Therefore, after executing Line 38, we have

$$\boldsymbol{u} \leftarrow \boldsymbol{u} + \Delta \boldsymbol{u} = \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)} \mathbf{B}^\top \mathbf{W}^{(\mathrm{new})1/2} \boldsymbol{v}.$$

Next, we need to update $\boldsymbol{u}$ to reflect the changes to $\widetilde{\mathbf{\Gamma}}, \mathbf{\Pi}^{(i)}$. Updating these matrices is done via `dynamicSC`. However, calling PartialProject($\mathbf{B}^\top \mathbf{W}^{(\mathrm{new})1/2} \boldsymbol{v}$) afterward is too costly if done directly, since the argument is a dense vector. To circumvent this problem, we make the key observation that the change to $\boldsymbol{u}$ is restricted to a subcollection of nodes on $\mathcal{T}$ (in fact a connected subtree containing the root), and it suffices to partially reverse and reapply the operator $\widetilde{\mathbf{\Gamma}} \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)}$. Intuitively, InversePartialProject revert all computations in PartialProject that are related to the changes to $\mathbf{W}$.

Let $H_1, \ldots, H_t$ be a permutation of all nodes in $\mathcal{T}$, such that the nodes in $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$ is a prefix of the permutation, and it satisfies that for any node H_i with descendant H_j, $i < j$. Then by Lemma 41, after executing Line 38, we have

$$\begin{aligned} \boldsymbol{u} &= \text{PartialProject}(\mathbf{B}^\top \mathbf{W}^{(\mathrm{new})1/2} \boldsymbol{v}, \mathcal{T}) \\ &= (\mathbf{I} - \mathbf{X}^{(H_1)}) \ldots (\mathbf{I} - \mathbf{X}^{(H_t)}) \mathbf{B}^\top \mathbf{W}^{(\mathrm{new})1/2} \boldsymbol{v}. \end{aligned} \tag{5.3}$$

Let $r = |\mathcal{P}_{\mathcal{T}}(\mathcal{H})|$. Then InversePartialProject($\boldsymbol{u}, \mathcal{H}$) on Line 39 returns $\boldsymbol{d}$ by Lemma 42 satisfying

$$\boldsymbol{d} = (\mathbf{I} + \mathbf{X}^{(H_r)}) \ldots (\mathbf{I} + \mathbf{X}^{(H_1)}) \boldsymbol{u}.$$

Plugging in $\boldsymbol{u}$ from Equation (5.3), we have

$$\boldsymbol{d} = (\mathbf{I} + \mathbf{X}^{(H_r)}) \ldots (\mathbf{I} + \mathbf{X}^{(H_1)})(\mathbf{I} - \mathbf{X}^{(H_1)}) \ldots (\mathbf{I} - \mathbf{X}^{(H_t)}) \mathbf{B}^\top \mathbf{W}^{(\mathrm{new})1/2} \boldsymbol{v}.$$

We use the fact that each $\mathbf{I} - \mathbf{X}^{(H_i)}$ is nonsingular and has inverse $\mathbf{I} + \mathbf{X}^{(H_i)}$ to get

$$\boldsymbol{d} = (\mathbf{I} - \mathbf{X}^{(H_{r+1})}) \ldots (\mathbf{I} - \mathbf{X}^{(H_t)}) \mathbf{B}^\top \mathbf{W}^{(\mathrm{new})1/2} \boldsymbol{v}.$$

We then call `dynamicSC`.Reweight, which updates $\mathbf{L}^{(H)}$ and in turn $\mathbf{X}^{(H_i)}$ for precisely all nodes in $\mathcal{P}_{\mathcal{T}}(\mathcal{H}) = \{H_1, \ldots, H_r\}$. Let $\mathbf{X}^{(H)'}$ denote the matrix after reweight. Next, we call PartialProject again. Let us denote it by PartialProject$^{(\mathrm{new})}$ to emphasize that it runs with new weights. This gives

$$\begin{aligned} \boldsymbol{u}' &= \text{PartialProject}^{(\mathrm{new})}(\boldsymbol{d}, \mathcal{H}) \\ &= (\mathbf{I} - \mathbf{X}^{(H_1)'}) \ldots (\mathbf{I} - \mathbf{X}^{(H_r)'}) \boldsymbol{d} \\ &= (\mathbf{I} - \mathbf{X}^{(H_1)'}) \ldots (\mathbf{I} - \mathbf{X}^{(H_r)'})(\mathbf{I} - \mathbf{X}^{(H_{r+1})}) \ldots (\mathbf{I} - \mathbf{X}^{(H_t)}) \mathbf{B}^\top \mathbf{W}^{(\mathrm{new})1/2} \boldsymbol{v} \\ &= (\mathbf{I} - \mathbf{X}^{(H_1)'}) \ldots (\mathbf{I} - \mathbf{X}^{(H_t)'}) \mathbf{B}^\top \mathbf{W}^{(\mathrm{new})1/2} \boldsymbol{v} \qquad (\text{since } \mathbf{X}^{(H_i)'} = \mathbf{X}^{(H_i)} \text{ for all } i > r) \\ &= \text{PartialProject}^{(\mathrm{new})}(\mathbf{B}^\top \mathbf{W}^{(\mathrm{new})1/2} \boldsymbol{v}, \mathcal{T}). \end{aligned}$$

Because $\boldsymbol{u}'|_{F_H}$ is updated on $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$, and $\mathbf{L}^{(H)}$ is updated on $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$ by Theorem 6, running Line 45 on $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$ correctly sets $\boldsymbol{z}^{(\mathrm{step})'} = \widetilde{\mathbf{\Gamma}}' \boldsymbol{u}'$.

For the runtime, the first call to PartialProject has a vector with $O(K)$ sparsity as the argument, and therefore runs in $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK})$. Next, we know $|\mathcal{H}| = O(K)$. The call to InversePartialProject and the subsequent call to PartialProject both have $\mathcal{H}$ as an argument, so they run in $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK})$. The `DynamicSC`.Reweight call runs in $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK})$ Updating $\boldsymbol{z}^{(\text{step})}$ (Line 45) takes $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK})$ time in total by Theorem 6 and Lemma 27. And finally, we can update $\boldsymbol{z}^{(\text{sum})}$ in the same time.

We remark that although InversePartialProject returns a vector $\boldsymbol{d}$ that is not necessarily sparse, and we then assign $\boldsymbol{u} \leftarrow$ PartialProject$(\boldsymbol{d}, \mathcal{H})$, this is for readability. $\boldsymbol{d}$ is in fact an intermediate state of $\boldsymbol{u}$, on which we perform in-place operations. □

5.2 Tree Operator

At IPM step k, our goal is to write the slack update $\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}^{(k)}$ as $\mathbf{M}^{(\text{slack})}\boldsymbol{z}^{(k)}$, and similarly, write the partial flow update $\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v}^{(k)}$ approximately as $\mathbf{M}^{(\text{flow})}\boldsymbol{z}^{(k)}$, where $\boldsymbol{z}^{(k)}$ is defined in the previous subsection, and $\mathbf{M}^{(\text{slack})}$ and $\mathbf{M}^{(\text{flow})}$ are linear operators that are efficiently maintainable between IPM steps.

In this section, we define a general class of operators called *tree operators* and show how to efficiently compute and maintain them. In later sections, we show that $\mathbf{M}^{(\text{slack})}$ and $\mathbf{M}^{(\text{flow})}$ can be defined as tree operators.

We begin with the formal definitions. Recall for a tree $\mathcal{T}$ and node $H \in \mathcal{T}$, we use $\mathcal{T}_H$ to denote the subtree rooted at H.

Definition 44 (Tree Operator). Suppose $\mathcal{T}$ is a rooted tree with constant degree and root G. Let each node $H \in \mathcal{T}$ be associated with two sets $V(H)$ and $F_H \subseteq V(H)$. Let each leaf node $H \in \mathcal{T}$ be further associated with a non-empty set $E(H)$ of constant size, where *the $E(H)$'s are pairwise disjoint over all leaf nodes* For a non-leaf node H, define $E(H) \stackrel{\text{def}}{=} \bigcup_{\text{leaf } D \in \mathcal{T}_H} E(D)$. Finally, define $E \stackrel{\text{def}}{=} E(G) = \bigcup_{\text{leaf } H \in \mathcal{T}} E(H)$ and $V \stackrel{\text{def}}{=} V(G)$.

Let each node H with parent P be associated with a linear *edge operator* $\mathbf{M}_{(H,P)} : \mathbb{R}^{V(P)} \mapsto \mathbb{R}^{V(H)}$. In addition, let each leaf node H be associated with a *constant-time computable* linear *leaf operator* $\mathbf{J}_H : \mathbb{R}^{V(H)} \mapsto \mathbb{R}^{E(H)}$ We extend all these operators trivially to $\mathbb{R}^V$ and $\mathbb{R}^E$, respectively, in order to have matching dimensions overall. When an edge or leaf operator is not given, we assume it to be $\mathbf{0}$.

For a path $H_t \longrightarrow H_1 \stackrel{\text{def}}{=} (H_t, \ldots, H_1)$, where each H_i is the parent of H_{i-1} and H_1 is a leaf node (call these *tree paths*), we define

$$\mathbf{M}_{H_1 \leftarrow H_t} = \mathbf{M}_{(H_1,H_2)}\mathbf{M}_{(H_2,H_3)} \cdots \mathbf{M}_{(H_{t-1},H_t)}.$$

If $t = 1$, then $\mathbf{M}_{H_1 \leftarrow H_t} \stackrel{\text{def}}{=} \mathbf{I}$.

We define *the tree operator* $\mathbf{M} : \mathbb{R}^V \mapsto \mathbb{R}^E$ *supported on* $\mathcal{T}$ to be

$$\mathbf{M} \stackrel{\text{def}}{=} \sum_{\text{leaf } H, \text{ node } A \,:\, H \in \mathcal{T}_A} \mathbf{J}_H \mathbf{M}_{H \leftarrow A} \mathbf{I}_{F_A}. \tag{5.4}$$

We always maintain a tree operator implicitly by maintaining

$$\{\mathbf{J}_H : \text{leaf } H\} \cup \{\mathbf{M}_{(H,P)} : \text{edge } (H,P)\} \cup \{F_H : \text{node } H\}.$$

Remark 45. Although we define the tree operator in general and hope it will find applications in other problems we have used suggestive names in the definition to suit our min-cost flow setting. In particular, our tree operators will be supported on the separator tree $\mathcal{T}$. For each node H, the sets $V(H), F_H, E(H)$ associated with the tree operator are, respectively, $\partial H \cup F_H$ of region H, the eliminated vertices F_H of region H, and the edge set of region H, all from the separator tree construction.

To maintain $\mathbf{M}$ using the tree efficiently, we also need some partial operators:

Definition 46 ($\mathbf{M}^{(H)}, \overline{\mathbf{M}^{(H)}}$). For convenience of notation, define $\mathcal{T}_H$ to be the subtree of $\mathcal{T}$ rooted at node H. We define the subtree operator $\mathbf{M}^{(H)} : V(H) \mapsto E(H)$ at a node H to be

$$\mathbf{M}^{(H)} \stackrel{\text{def}}{=} \sum_{\text{leaf } D \in \mathcal{T}_H} \mathbf{J}_D \mathbf{M}_{D \leftarrow H}. \tag{5.5}$$

We also define the partial sum

$$\overline{\mathbf{M}^{(H)}} \stackrel{\text{def}}{=} \sum_{D \in \mathcal{T}_H} \mathbf{M}^{(D)} \mathbf{I}_{F_D}. \tag{5.6}$$

We state a straightforward corollary without proof.

Corollary 47. *Based on the above definitions, we have*

$$\mathbf{M} = \sum_{H \in \mathcal{T}} \mathbf{M}^{(H)} \mathbf{I}_{F_H} = \overline{\mathbf{M}^{(G)}}.$$

Furthermore, if H has children D_1, D_2, then

$$\mathbf{M}^{(H)} = \mathbf{M}^{(D_1)} \mathbf{M}_{(D_1,H)} + \mathbf{M}^{(D_2)} \mathbf{M}_{(D_2,H)}. \tag{5.7}$$

We observe an orthogonality property of the subtree operators:

Lemma 48. *For any nodes H, H' at the same level in $\mathcal{T}$,* $\mathrm{Range}(\mathbf{M}^{(H)})$ *and* $\mathrm{Range}(\mathbf{M}^{(H')})$ *are orthogonal.*

Proof. Since H and H' are at the same level in $\mathcal{T}$, we know $\mathcal{T}_H$ and $\mathcal{T}_{H'}$ have disjoint sets of leaves. The range of $\mathbf{M}^{(H)}$ is supported on edges in the regions given by leaves of $\mathcal{T}_H$, and analogously for the range of $\mathbf{M}^{(H')}$. □

We define the complexity of a tree operator to be parameterized by the number of tree edges.

Definition 49 (Complexity of Tree Operator). Let $\mathbf{M}$ be a tree operator on tree $\mathcal{T}$. We say $\mathbf{M}$ has complexity function T if for any $k > 0$, for any set S of k distinct edges in $\mathcal{T}$ and any families of vectors $\{\boldsymbol{u}_e : e \in S\}$ and $\{\boldsymbol{v}_e : e \in S\}$ the total cost of computing $\{\boldsymbol{u}_e^\top \mathbf{M}_e : e \in S\}$ and $\{\mathbf{M}_e \boldsymbol{v}_e : e \in S\}$ is bounded by $T(k)$.

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

We can show the structure of a tree operator by the procedure ComputeMz($\mathbf{M}, \boldsymbol{z}$) to compute $\mathbf{M}\boldsymbol{z}$. Intuitively, $\boldsymbol{z}$ is given as input to each node H. The edge operators are concatenated in the order of tree paths from H to a leaf, but we apply them level-wise in descending order.

Corollary 50. *Suppose $\mathbf{M} : \mathbb{R}^V \to \mathbb{R}^E$ is a tree operator on tree $\mathcal{T}$ with complexity T, where $|V| = n$ and $|E| = m$. Then for $\boldsymbol{z} \in \mathbb{R}^V$, Exact($\mathbf{M}, \boldsymbol{z}$) outputs $\mathbf{M}\boldsymbol{z}$ in $O(T(K)) = O(T(m))$ time where K is the total number of non-zero edge and leaf operators in $\mathbf{M}$.*

Proof. Note only non-zero edge and leaf operators contribute to $\mathbf{M}\boldsymbol{z}$. We omit the proof of correctness as it is simply an application of the definition.

Since $E = \cup_{\text{leaf } D} E(D)$, and each $E(D)$ has constant size, we know there are at most $O(m)$ leaves in $\mathcal{T}$ Hence, there are $O(m)$ edges in $\mathcal{T}$, and $K = O(m)$. Since we define each leaf operator to be constant time computable, applying $\mathbf{J}_H$ for leaves in $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$ costs $O(K)$ time in total. The bottleneck of the procedure is to apply the edge operator $\mathbf{M}_e$ to some vector exactly once for each edge e in $\mathcal{T}$; the time cost is $O(T(K))$ by definition of the operator complexity. □

ALGORITHM 5: Compute $\mathbf{M}z$ for a tree operator $\mathbf{M}$

```
1:  procedure ComputeMz($\mathbf{M}, z$)
2:      $\mathcal{H} \leftarrow$ set of all nodes $H$ in $\mathcal{T}$ such that $\mathbf{M}_{(H,P)}$ or $\mathbf{J}_H$ is nonzero
3:      $\mathcal{P}_{\mathcal{T}}(\mathcal{H}) \leftarrow$ set of $\mathcal{H}$ and all ancestor nodes of $\mathcal{H}$ in $\mathcal{T}$
4:      $\boldsymbol{v}_H \leftarrow \mathbf{0}$ for each $H \in \mathcal{T}$        ▷ sparse vectors for intermediate computations
5:      for each node $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$ do
6:          $\boldsymbol{v}_H \leftarrow \mathbf{I}_{F_H} z = z|_{F_H}$        ▷ apply the $\mathbf{I}_{F_H}$ part of the operator
7:      end for
8:      for each node $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$ by decreasing level do
9:          Let $P$ be the parent of $H$
10:         $\boldsymbol{v}_H \leftarrow \boldsymbol{v}_H + \mathbf{M}_{(H,P)}\boldsymbol{v}_P$        ▷ apply $\mathbf{M}_{(H,P)}$ as we move from $P$ to $H$
11:     end for
12:     for each leaf node $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$ do
13:         $\boldsymbol{x}|_{E(H)} \leftarrow \mathbf{J}_H \boldsymbol{v}_H$        ▷ apply the leaf operator
14:     end for
15:     return $\boldsymbol{x}$
16: end procedure
```

5.3 Proof of Theorem 7

Finally, we give the data structure for maintaining an implicit representation of the form $\boldsymbol{y} + \mathbf{M}z$ throughout the IPM. For an instantiation of this data structure, there is exactly one call to INITIALIZE at the very beginning, and one call to Exact at the very end. Otherwise, each step of the IPM consists of one call to REWEIGHT followed by one call to MOVE. In our pseudocode, we use a box around a vector to mean that vector is maintained using a data structure rather than directly accessible.

PROOF OF THEOREM 7. First, we discuss how $\mathbf{M}$ is stored in the data structure: Recall $\mathbf{M}$ is represented implicitly by a collection of edge operators and leaf operators on the separator tree $\mathcal{T}$, so that each edge operator is stored at a corresponding node of $\mathcal{T}$, and each leaf operator is stored at a corresponding leaf node of $\mathcal{T}$. However, the data structure *does not* store any edge or leaf operator matrix explicitly. We make a key assumption that each edge and leaf operator is computable using $O(1)$-number of $\mathbf{L}^{(H)}$ matrices from DYNAMICSC. This will be true for the slack and flow operators we define. As a result, to store an edge or leaf operator at a node, we simply store *pointers to the matrices from DYNAMICSC* required in the definition, and an $O(1)$-sized instruction for how to compute the operator. The computation time is proportional to the size of the matrices in the definitions, but crucially the instructions have only $O(1)$-size.

Now, we prove the correctness and runtime of each procedure separately. Observe that the invariants claimed in the theorem are maintained correctly if each procedure is implemented correctly.

INITIALIZE: Line 11 sets $\boldsymbol{y} \leftarrow \boldsymbol{x}^{(\text{init})}$, and $\boxed{z}$.INITIALIZE sets $z \leftarrow \mathbf{0}$. So we have $\boldsymbol{x} = \boldsymbol{y} + \mathbf{M}z$ at the end of initialization. Furthermore, the initialization of z correctly sets $z^{(\text{step})}$ in terms of $\boldsymbol{v}$.

By Theorem 43, $\boxed{z}$.INITIALIZE takes $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}m)$ time. Storing the implicit representation of $\mathbf{M}$ takes $O(m)$ time.

REWEIGHT: By Theorem 43, $\boxed{z}$.REWEIGHT updates its current weight and `DynamicSC`, and updates $z^{(\text{step})}$ correspondingly to maintain the invariant, while not changing the value of z. Because $\mathbf{M}$ is stored by instructions, no explicit update to $\mathbf{M}$ is required. Line 19 updates $\boldsymbol{y}$ to zero out the changes to $\mathbf{M}z$.

ALGORITHM 6: Implicit representation maintenance

```
1: data structure MaintainRep
2: private: member
3:     $\mathcal{T}$: separator tree
4:     $\boldsymbol{y} \in \mathbb{R}^m$: offset vector
5:     $\mathbf{M}$: instructions to compute the tree operator $\mathbf{M} \in \mathbb{R}^{m\times n}$
6:     $\boxed{z}$: an instance of MaintainZ which maintains $\boldsymbol{z} = c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})}$
7:
8: procedure Initialize($\mathcal{T}, \mathbf{M}, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \boldsymbol{x}^{(\text{init})} \in \mathbb{R}^m, \epsilon_{\mathbf{P}} > 0$)
9:     $\mathbf{M} \leftarrow \mathbf{M}$                         ▷ initialize the instructions to compute $\mathbf{M}$
10:    $\boxed{z}$.Initialize($\mathcal{T}, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \epsilon_{\mathbf{P}} > 0$)
11:    $\boldsymbol{y} \leftarrow \boldsymbol{x}^{(\text{init})}$
12: end procedure
13:
14: procedure Reweight($\boldsymbol{w}^{(\text{new})}$)
15:    Let $\mathbf{M}^{(\text{old})}$ represent the current tree operator $\mathbf{M}$
16:    $\boxed{z}$.Reweight($\boldsymbol{w}^{(\text{new})}$)       ▷ update representation of $z$ and DynamicSC
17:                                 ▷ $\mathbf{M}$ is updated as a result of reweight in DynamicSC
18:    $\Delta\mathbf{M} \leftarrow \mathbf{M} - \mathbf{M}^{(\text{old})}$                ▷ $\Delta\mathbf{M}$ is represented implicitly
19:    $\boldsymbol{y} \leftarrow \boldsymbol{y} -$ ComputeMz($\Delta\mathbf{M}, c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})}$)       ▷ Algorithm 5
20: end procedure
21:
22: procedure Move($\alpha, \boldsymbol{v}^{(\text{new})}$)
23:    $\boxed{z}$.Move($\alpha, \boldsymbol{v}^{(\text{new})}$)
24: end procedure
25:
26: procedure Exact()
27:    return $\boldsymbol{y} +$ ComputeMz($\mathbf{M}, c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})}$)       ▷ Algorithm 5
28: end procedure
```

The instructions for computing $\Delta\mathbf{M}$ require the Laplacians from DynamicSC before and after the update in Line 16 For this, we monitor the updates of dynamicSC and store the old and new values The runtime of this is bounded by the runtime of updating dynamicSC, which is in turn included in the runtime for $\boxed{z}$.Reweight.

Let K upper bound the number of coordinates changed in $\boldsymbol{w}$ and the number of edge and leaf operators changed in $\mathbf{M}$. Then Super.Reweight takes $\widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}\sqrt{mK})$ time and Exact($\Delta\mathbf{M}, z$) takes $O(T(K))$ time. Thurs, the total runtime is $\widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}\sqrt{mK} + T(m))$.

Move: The runtime and correctness follow from Theorem 43.

Exact: ComputeMz computes $\mathbf{M}z$ correctly in $O(T(m))$ time by Corollary 50. Adding the result to $\boldsymbol{y}$ takes $O(m)$ time and gives the correct value of $\boldsymbol{x} = \boldsymbol{y} + \mathbf{M}\boldsymbol{z}$. Thus, Exact returns $\boldsymbol{x}$ in $O(T(m))$ time. □

6 Maintaining Vector Approximation

Recall at every step of the IPM, we want to maintain approximate vectors $\overline{s}, \overline{\boldsymbol{f}}$ so that

$$\left\|\mathbf{W}^{-1/2}(\overline{\boldsymbol{f}} - \boldsymbol{f})\right\|_\infty \le \delta \quad \text{and} \quad \left\|\mathbf{W}^{1/2}(\overline{\boldsymbol{s}} - \boldsymbol{s})\right\|_\infty \le \delta'$$

for some additive error tolerances δ and δ'.

In the previous section, we showed how to maintain some vector $\boldsymbol{x}$ implicitly as $\boldsymbol{x} \stackrel{\text{def}}{=} \boldsymbol{y} + \mathbf{M}\boldsymbol{z}$ throughout the IPM, where $\boldsymbol{x}$ should represent $\boldsymbol{s}$ or part of $\boldsymbol{f}$. In this section, we give a data structure to efficiently maintain an approximate vector $\overline{\boldsymbol{x}}$ to the $\boldsymbol{x}$ from MaintainRep, so that at every IPM step,

$$\left\|\mathbf{D}^{1/2}\left(\overline{\boldsymbol{x}} - \boldsymbol{x}\right)\right\|_\infty \le \delta,$$

where $\mathbf{D}$ is a diagonal scaling matrix that is a fixed function of $\overline{\boldsymbol{x}}$. (It will be $\mathbf{W}^{-1}$ for the flow or $\mathbf{W}$ for the slack.)

In Section 6.1, we reduce the problem of maintaining $\overline{\boldsymbol{x}}$ to detecting coordinates in $\boldsymbol{x}$ with large changes In Section 6.2, we detect coordinates of $\boldsymbol{x}$ with large changes using a sampling technique on a binary tree, where Johnson–Lindenstrauss sketches of subvectors of $\boldsymbol{x}$ are maintained at each node of the tree In Section 6.3, we show how to compute and maintain the necessary collection of JL-sketches on the separator tree $\mathcal{T}$; in particular, we do this efficiently with only an implicit representation of $\boldsymbol{x}$. Finally, we put the three parts together to prove Theorem 8.

We use the superscript $^{(k)}$ to denote the variable at the end of the kth step of the IPM; that is, $\mathbf{D}^{(k)}$ and $\boldsymbol{x}^{(k)}$ are $\mathbf{D}$ and $\boldsymbol{x}$ at the end of the kth step. Step 0 is the state of the data structure immediately after initialization.

6.1 Reduction to Change Detection

In this subsection, we show that in order to maintain an approximation $\overline{\boldsymbol{x}}$ to some vector $\boldsymbol{x}$, it suffices to detect coordinates of $\boldsymbol{x}$ that change a lot.

To do so, we make use of dyadic intervals. At step k of the IPM, for each ℓ such that $k = 0 \bmod 2^\ell$, we find the 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 51. At step k of the IPM, for each ℓ such that $k = 0 \bmod 2^\ell$, we define

$$I_\ell^{(k)} \stackrel{\text{def}}{=} \{i \in [n] : \sqrt{\mathbf{D}_{ii}^{(k-1)}} \cdot |\boldsymbol{x}_i^{(k)} - \boldsymbol{x}_i^{(k-2^\ell)}| \ge \frac{\delta}{2\lceil \log m \rceil}$$
$$\text{and } \overline{\boldsymbol{x}}_i \text{ has not been updated since the } (k-2^\ell)\text{-th step}\}.$$

We show how to find the sets $I_\ell^{(k)}$ with high probability in the next subsection. Assuming the correct implementation, we have the following data structure for maintaining the desired approximation $\overline{\boldsymbol{x}}$:

Lemma 52 (Approximate Vector Maintenance). *Suppose FindLargeCoordinates(ℓ) is a procedure in AbstractMaintainApprox that correctly computes the set $I_\ell^{(k)}$ at the kth step. Then the deterministic data structure AbstractMaintainApprox in Algorithm 7 maintains an approximation $\overline{\boldsymbol{x}}$ of $\boldsymbol{x}$ with the following procedures:*

- *Initialize($\mathcal{T}, \boldsymbol{x} \in \mathbb{R}^m, \rho > 0, \delta > 0$): Initialize the data structure at step 0 with tree $\mathcal{T}$, initial vector $\boldsymbol{x}$, target additive approximation error δ, and success probability $1-\rho$.*
- *Approximate($\boldsymbol{x}^{(\text{new})} \in \mathbb{R}^m$): Increment the step counter and update vector $\boldsymbol{x}$. Output a vector $\overline{\boldsymbol{x}}$ such that $\|\mathbf{D}^{1/2}(\boldsymbol{x} - \overline{\boldsymbol{x}})\|_\infty \le \delta$ for the latest $\boldsymbol{x}$ and $\mathbf{D}$.*

Furthermore, if $\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\|_{\mathbf{D}^{(k-1)}} \le \beta$ for all k, then at the kth step, the data structure updates $\overline{\boldsymbol{x}}_i \leftarrow \boldsymbol{x}_i^{(k)}$ for $O(2^{2\ell_k}(\beta/\delta)^2 \log^2 m)$ coordinates, where ℓ_k is the largest integer ℓ with $k \equiv 0 \bmod 2^\ell$.

Remark 53. In our problem setting of maintaining approximate flows and slacks, we do not have full access to the exact vector The algorithms in the next two subsections; however, will refer to the exact vector $\boldsymbol{x}$ for readability and modularity We observe that access to $\boldsymbol{x}$ is limited to two

ALGORITHM 7: Data structure AbstractMaintainApprox, Part 1

```
1: data structure AbstractMaintainApprox
2: private : member
3:     $\mathcal{T}$: constant-degree rooted tree with height $\eta$ and $m$ leaves
4:     $w \overset{\text{def}}{=} \Theta(\eta^2 \log(\frac{m}{\rho}))$: sketch dimension
5:     $\mathbf{\Phi} \sim \mathrm{N}(0, \frac{1}{w})^{w \times m}$: JL-sketch matrix
6:     $\delta > 0$: additive approximation error
7:     $k$: current IPM step
8:     $\overline{\boldsymbol{x}} \in \mathbb{R}^m$: current valid approximate vector
9:     $\{\boldsymbol{x}^{(j)} \in \mathbb{R}^m\}_{j=0}^{k}$: list of previous inputs
10:    $\{\mathbf{D}^{(j)} \in \mathbb{R}^{m \times m}\}_{j=0}^{k}$: list of previous diagonal scaling matrices
11:
12: procedure Initialize($\mathcal{T}, \boldsymbol{x} \in \mathbb{R}^m, \rho > 0, \delta > 0$)
13:     $\mathcal{T} \leftarrow \mathcal{T}, \delta \leftarrow \delta, k \leftarrow 0$
14:     $\overline{\boldsymbol{x}} \leftarrow \boldsymbol{x}, \boldsymbol{x}^{(0)} \leftarrow \boldsymbol{x}, \mathbf{D}^{(0)}$ computed from $\overline{\boldsymbol{x}}$
15:     sample $\mathbf{\Phi} \sim \mathrm{N}(0, \frac{1}{w})^{w \times m}$
16: end procedure
17:
18: procedure Approximate($\boldsymbol{x}^{(\text{new})} \in \mathbb{R}^m$)
19:     $k \leftarrow k + 1$,
20:     $\boldsymbol{x}^{(k)} \leftarrow \boldsymbol{x}^{(\text{new})}$
21:     $I \leftarrow \emptyset$
22:     for all $0 \le \ell < \lceil \log m \rceil$ such that $k \equiv 0 \bmod 2^{\ell}$ do
23:         $I_{\ell}^{(k)} \leftarrow$ FindLargeCoordinates($\ell$)
24:         $I \leftarrow I \cup I_{\ell}^{(k)}$
25:     end for
26:     if $k = 0 \bmod 2^{\lceil \log m \rceil}$ then
27:         $I \leftarrow [m]$                                   ▷ Update $\overline{\boldsymbol{x}}$ in full every $2^{\lceil \log m \rceil}$ steps
28:     end if
29:     $\overline{\boldsymbol{x}}_i \leftarrow \boldsymbol{x}_i^{(k)}$ for all $i \in I$
30:     $\mathbf{D}^{(k)} \leftarrow \mathbf{D}^{(k-1)}$, then update $\mathbf{D}_{ii}^{(k)}$ for $i \in I$ from $\overline{\boldsymbol{x}}_i$
31:     return $\overline{\boldsymbol{x}}$
32: end procedure
```

types: accessing the JL-sketches of specific subvectors, and accessing exact coordinates and other specific subvectors of sufficiently small size. In later sections, we show how to implement these oracle accesses to $\boldsymbol{x}$.

Proof of Lemma 52. We first prove the correctness of Approximate in AbstractMaintainApprox Fix some coordinate $i \in [m]$ and fix some IPM step k Suppose the latest update to $\overline{\boldsymbol{x}}_i$ is $\overline{\boldsymbol{x}}_i \leftarrow \boldsymbol{x}_i^{(k')}$ We have that $\mathbf{D}_{ii}^{(d)}$ is the same for all $k > d > k'$ and that i is not in the set $I_{\ell}^{(d)}$ returned by FindLargeCoordinates for all $k > d > k'$. Since we set $\overline{\boldsymbol{x}} \leftarrow \boldsymbol{x}$ every $2^{\lceil \log m \rceil}$ steps by Line 27 we have $k - 2^{\lceil \log m \rceil} \le k' < k$. Using dyadic intervals, we can write $k' = k_0 < k_1 < k_2 < \cdots < k_s = k$ such that $k_{j+1} - k_j$ is a power of 2, and $|s| \le 2\lceil \log m \rceil$. Hence, we have that

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

We know that $\mathbf{D}_{ii}^{(d)}$ is the same for all $k > d > k'$. By the guarantees of FindLargeCoordinates, we have

$$\sqrt{\mathbf{D}_{ii}^{(k-1)}} \cdot \left|\boldsymbol{x}_i^{(k_{j+1})} - \boldsymbol{x}_i^{(k_j)}\right| = \sqrt{\mathbf{D}_{ii}^{(k_{j+1})}} \cdot \left|\boldsymbol{x}_i^{(k_{j+1})} - \boldsymbol{x}_i^{(k_j)}\right| \leq \frac{\delta}{2\lceil \log m \rceil}$$

for all $0 \leq j < s$ Summing over all $j = 0, 1, \ldots, s-1$ gives

$$\sqrt{\mathbf{D}_{ii}^{(k-1)}} \cdot \left|\boldsymbol{x}_i^{(k)} - \overline{\boldsymbol{x}}_i^{(k)}\right| \leq \delta.$$

Hence, we have $\|\mathbf{D}^{1/2}(\boldsymbol{x} - \overline{\boldsymbol{x}})\|_\infty \leq \delta$.

Next, we bound the number of coordinates changed from $\overline{\boldsymbol{x}}^{(k-1)}$ to $\overline{\boldsymbol{x}}^{(k)}$ Fix some ℓ with $k = 0 \bmod 2^\ell$. For any $i \in I_\ell^{(k)}$, we know $\mathbf{D}_{ii}^{(j)} = \mathbf{D}_{ii}^{(k-1)}$ 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

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

Using $\mathbf{D}_{ii}^{(j)} = \mathbf{D}_{ii}^{(k-1)}$ for all $j > k - 2^\ell$ again, the above inequality yields

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

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

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

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

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

Recall this expression is for a fixed ℓ At the kth step, summing over all ℓ with $k = 0 \bmod 2^\ell$, we have that the total number of coordinates changed is

$$\sum_{\ell=0}^{\ell_k} |I_\ell^{(k)}| = O(2^{2\ell_k}(\beta/\delta)^2 \log^2 m). \qquad \square$$

6.2 From Change Detection to Sketch Maintenance

Now we discuss the implementation of FindLargeCoordinates(ℓ) to find the set $I_\ell^{(k)}$ in Line 23 of Algorithm 7. We accomplish this by 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

ALGORITHM 8: Data structure AbstractMaintainApprox, Part 2

```
1:  procedure FindLargeCoordinates($\ell$)
2:                                       ▷ $\overline{\mathbf{D}}$ and $\boldsymbol{q}$ are symbolic definitions
3:      ▷ $\overline{\mathbf{D}}$: diagonal matrix such that
        $$\overline{\mathbf{D}}_{ii} = \begin{cases} \mathbf{D}_{ii}^{(k-1)} & \text{if } \overline{\boldsymbol{x}}_i \text{ has not been updated after the } (k-2^{\ell})\text{-th step} \\ 0 & \text{otherwise.} \end{cases}$$
4:      ▷ $\boldsymbol{q} \overset{\text{def}}{=} \overline{\mathbf{D}}^{1/2}(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-2^{\ell})})$      ▷ vector to sample coordinates from
5:
6:      $I \leftarrow \emptyset$                                    ▷ set of candidate coordinates
7:      for $N \overset{\text{def}}{=} \Theta(2^{2\ell}(\beta/\delta)^2 \log^2 m \log(m/\rho))$ iterations do
8:          ▷ Sample coordinate $i$ of $\boldsymbol{q}$ w.p. proportional to $\boldsymbol{q}_i^2$ by random descent down $\mathcal{T}$ to a leaf
9:          while true do
10:             $u \leftarrow \text{root}(\mathcal{T})$, $p_u \leftarrow 1$
11:             while $u$ is not a leaf node do
12:                 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 $u$
13:                 $p_u \leftarrow p_u \cdot \mathbf{P}(u \to u')$
14:                 $u \leftarrow u'$
15:             end while
16:             break with probability $p_{\text{accept}} \overset{\text{def}}{=} \left\|\boldsymbol{q}|_{E(u)}\right\|^2 / (2 \cdot p_u \cdot \|\boldsymbol{\Phi}\boldsymbol{q}\|_2^2)$
17:         end while
18:         $I \leftarrow I \cup E(u)$
19:     end for
20:     return $\{i \in I : \boldsymbol{q}_i \geq \frac{\delta}{2\lceil \log m \rceil}\}$.
21: end procedure
```

the sampling procedure, we make use of a data structure based on segment trees [16] along with sketching based on the Johnson–Lindenstrauss lemma.

Formally, we define the vector $\boldsymbol{q} \in \mathbb{R}^m$ where $\boldsymbol{q}_i \overset{\text{def}}{=} \mathbf{D}_{ii}^{(k-1)^{1/2}}(\boldsymbol{x}_i^{(k)} - \boldsymbol{x}_i^{(k-2^{\ell})})$ 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 54. *Assume that $\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\|_{\mathbf{D}^{(k)}} \leq \beta$ for all IPM steps k Let $\rho < 1$ be any given failure probability, and let $N \overset{\text{def}}{=} \Theta(2^{2\ell}(\beta/\delta)^2 \log^2 m \log(m/\rho))$ be the number of samples Algorithm 8 takes.*

Then with probability $\geq 1-\rho$, during the kth call of Approximate, *Algorithm 8 finds the set $I_\ell^{(k)}$ correctly Furthermore, the while-loop in Line 9 happens only $O(1)$ times in expectation per sample.*

Proof. The proof is similar to Lemma 6.17 in [21]. 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 16 breaks with probability at least $\frac{1}{4}$. By the choice of w, Johnson–Lindenstrauss lemma shows that $\|\Phi_{E(u)}\boldsymbol{q}\|_2^2 = (1 \pm \frac{1}{9\eta})\|\mathbf{I}_{E(u)}\boldsymbol{q}\|_2^2$ 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 16 is at least

$$p_{\text{accept}} = \frac{\left\|\boldsymbol{q}|_{E(u)}\right\|^2}{2 \cdot p_u \cdot \|\Phi\boldsymbol{q}\|_2^2} \geq \frac{\left\|\boldsymbol{q}|_{E(u)}\right\|^2}{2 \cdot (1+\frac{1}{3\eta})^\eta \frac{\left\|\boldsymbol{q}|_{E(u)}\right\|^2}{\|\boldsymbol{q}\|_2^2} \cdot \|\Phi\boldsymbol{q}\|_2^2} \geq \frac{\|\boldsymbol{q}\|_2^2}{2 \cdot (1+\frac{1}{3\eta})^\eta \|\Phi\boldsymbol{q}\|_2^2} \geq \frac{1}{4}.$$

On the other hand, we have that $p_{\text{accept}} \leq \frac{\|\boldsymbol{q}\|_2^2}{2(1-\frac{1}{3\eta})^\eta \|\Phi\boldsymbol{q}\|_2^2} < 1$ and hence this is a valid probability.

Next, we note that u is accepted on Line 16 with probability

$$p_{\text{accept}} p_u = \frac{\left\|\boldsymbol{q}|_{E(u)}\right\|^2}{2 \cdot \|\Phi\boldsymbol{q}\|_2^2}.$$

Since $\|\Phi\boldsymbol{q}\|_2^2$ 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 18, 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 kth step. Then

$$\begin{aligned}
\|\boldsymbol{q}\|_2 &= \|\mathbf{I}_S(\mathbf{D}^{(k-1)})^{1/2}(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-2^\ell)})\|_2 \\
&\leq \sum_{i=k-2^\ell}^{k-1} \|\mathbf{I}_S(\mathbf{D}^{(k-1)})^{1/2}(\boldsymbol{x}^{(i+1)} - \boldsymbol{x}^{(i)})\|_2 \\
&= \sum_{i=k-2^\ell}^{k-1} \|\mathbf{I}_S(\mathbf{D}^{(i)})^{1/2}(\boldsymbol{x}^{(i+1)} - \boldsymbol{x}^{(i)})\|_2 \\
&\leq \sum_{i=k-2^\ell}^{k-1} \|(\mathbf{D}^{(i)})^{1/2}(\boldsymbol{x}^{(i+1)} - \boldsymbol{x}^{(i)})\|_2 \\
&\leq 2^\ell \beta,
\end{aligned}$$

where we used $\mathbf{I}_S\mathbf{D}^{(i)} = \mathbf{I}_S\mathbf{D}^{(k-1)}$ Hence, each leaf u is sampled with probability at least $\left\|\boldsymbol{q}|_{E(u)}\right\|^2 / (2^\ell\beta)^2$. If $|\boldsymbol{q}_i| \geq \frac{\delta}{\lceil \log m \rceil}$, 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 55. In Algorithm 8, 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.

6.3 Sketch Maintenance

In FINDLARGECOORDINATES in the previous subsection, we assumed the existence of a constant degree 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 $u \in \mathcal{T}(0)$.

In this section, we consider when the required tree is the separator tree $\mathcal{T}$ of the overall input graph, and the vector $\boldsymbol{q}$ is of the form $\boldsymbol{q} = \boldsymbol{y} + \mathbf{M}\boldsymbol{z}$, where $\mathbf{M}$ is a tree operator supported on $\mathcal{T}$, and each of $\boldsymbol{y}, \mathbf{M}, \boldsymbol{z}$ undergo changes at every IPM step We present a data structure that implements two features efficiently on $\mathcal{T}$:

— access $(\boldsymbol{y} + \mathbf{M}\boldsymbol{z})|_{E(H)}$ at every leaf node H, where $E(H) \stackrel{\text{def}}{=} \text{Range}(\mathbf{J}_H)$.
— access $\boldsymbol{\Phi}_{E(H)}(\boldsymbol{y} + \mathbf{M}\boldsymbol{z})$ at every node H, where $\boldsymbol{\Phi}_{E(H)}$ is $\boldsymbol{\Phi}$ restricted to columns given by $E(H) \stackrel{\text{def}}{=} \bigcup_{\text{leaf } D \in \mathcal{T}_H} E(D)$.

Remark 56. As seen in the pseudocode, sketches for $\boldsymbol{y}$ and $\mathbf{M}\boldsymbol{z}$ can be maintained separately. We collected them together to represent $\boldsymbol{x}$ as a whole for simplicity.

First, we present some lemmas about the structure of the expression $\mathbf{M}\boldsymbol{z}$ which will help us to implement the requirements above. For any node $H \in \mathcal{T}$, let $\mathcal{T}_H$ be the subtree of $\mathcal{T}$ rooted at H.

LEMMA 57. *At any leaf node $H \in \mathcal{T}(0)$, we have*

$$(\mathbf{M}\boldsymbol{z})|_{E(H)} = \sum_{A: H \in \mathcal{T}_A} \mathbf{J}_H \mathbf{M}_{H \leftarrow A} \mathbf{I}_{F_A} \boldsymbol{z} = \mathbf{J}_H \mathbf{I}_{F_H} \boldsymbol{z} + \sum_{\text{ancestor } A \text{ of } H} \mathbf{J}_H \mathbf{M}_{H \leftarrow A} \mathbf{I}_{F_A} \boldsymbol{z}.$$

PROOF. Recall from the definition of the tree operator that $\text{Range}(\mathbf{J}_H)$ are disjoint. So to get $(\mathbf{M}\boldsymbol{z})|_{E(H)}$, it suffices to only consider the terms corresponding to the leaf H in the expression Equation (5.4) for $\mathbf{M}$; this gives the first equality. The second equality simply splits the sum into two parts. (We do not consider a node to be its own ancestor.) □

LEMMA 58. *At any node $H \in \mathcal{T}$, we have*

$$\boldsymbol{\Phi}_{E(H)}\mathbf{M}\boldsymbol{z} = \boldsymbol{\Phi}\overline{\mathbf{M}^{(H)}}\boldsymbol{z} + \boldsymbol{\Phi}\mathbf{M}^{(H)} \sum_{\text{ancestor } A \text{ of } H} \mathbf{M}_{H \leftarrow A} \mathbf{I}_{F_A} \boldsymbol{z}.$$

Intuitively, the lemma shows that the sketch of $\mathbf{M}\boldsymbol{z}$ restricted to $E(H)$ can be split into two parts The first part involves some sum over all nodes in $\mathcal{T}_H$, i.e., descendants of H and H itself, and the second part involves a sum over all ancestors of H.

Proof. First, note that since Φ is restricted to $E(H)$, it suffices to consider the terms in the sum for $\mathbf{M}$ that map into $E(H)$. In particular, this is the set of leaf nodes $\mathcal{T}_H$ in the subtree rooted at H.

$$\Phi_{E(H)}\mathbf{M}z = \Phi \sum_{\text{leaf } D\in\mathcal{T}_H} \sum_{A:D\in\mathcal{T}_A} \mathbf{J}_D\mathbf{M}_{D\leftarrow A}\mathbf{I}_{F_A}z.$$

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

$$\sum_{\text{leaf } D\,\in\,\mathcal{T}_H} \sum_{A\in\mathcal{T}_H} \mathbf{J}_D\mathbf{M}_{D\leftarrow H}\mathbf{I}_{F_H}z + \sum_{\text{leaf } D\,\in\,\mathcal{T}_H} \sum_{\text{ancestor } A \text{ of } H} \mathbf{J}_D\mathbf{M}_{D\leftarrow A}\mathbf{I}_{F_A}z.$$

The first term is precisely $\overline{\mathbf{M}^{(H)}}z$. For the second term, we can use the fact that A is an ancestor of H to expand $\mathbf{M}_{D\leftarrow A} = \mathbf{M}_{D\leftarrow H}\mathbf{M}_{H\leftarrow A}$. Then, the second term is

$$\begin{aligned}&\sum_{\text{leaf } D\,\in\,\mathcal{T}_H} \sum_{\text{ancestor } A \text{ of } H} \mathbf{J}_D\mathbf{M}_{D\leftarrow H}\mathbf{M}_{H\leftarrow A}\mathbf{I}_{F_A}z\\ &= \sum_{\text{leaf } D\,\in\,\mathcal{T}_H} \mathbf{J}_D\mathbf{M}_{D\leftarrow H}\left(\sum_{\text{ancestor } A \text{ of } H}\mathbf{M}_{H\leftarrow A}\mathbf{I}_{F_A}z\right)\\ &= \mathbf{M}^{(H)}\left(\sum_{\text{ancestor } A \text{ of } H}\mathbf{M}_{H\leftarrow A}\mathbf{I}_{F_A}z\right),\end{aligned}$$

by definition of $\mathbf{M}^{(H)}$. □

Lemma 59. *Let $\mathcal{T}$ be a rooted tree with height η supporting tree operator $\mathbf{M}$ with complexity T. Let $w = \Theta(\eta^2\log(\frac{m}{\rho}))$ be as defined in Algorithm 7, and let $\Phi\in\mathbb{R}^{w\times m}$ be a JL-sketch matrix. Then MaintainSketch (Algorithm 9) is a data structure that maintains $\Phi(y+\mathbf{M}z)$, as y, $\mathbf{M}$ and z undergo changes in the IPM. The data structure supports the following procedures:*

- *Initialize(rooted tree $\mathcal{T}$, $\Phi\in\mathbb{R}^{w\times m}$, tree operator $\mathbf{M}^{(\text{init})}\in\mathbb{R}^{m\times n}$, $z^{(\text{init})}\in\mathbb{R}^n$, $y^{(\text{init})}\in\mathbb{R}^m$): Initialize the data structure with tree operator $\mathbf{M}\leftarrow\mathbf{M}^{(\text{init})}$, and vectors $z\leftarrow z^{(\text{init})}$, $y\leftarrow y^{(\text{init})}$, and compute the initial sketches in $O(w\cdot m)$ time.*
- *Update($\mathbf{M}^{(\text{new})}, z^{(\text{new})}, y^{(\text{new})}$): Update $\mathbf{M}\leftarrow\mathbf{M}^{(\text{new})}$, $z\leftarrow z^{(\text{new})}$, $y\leftarrow y^{(\text{new})}$ and all the necessary sketches in $O(w\cdot T(\eta\cdot|\mathcal{S}|))$ time, where $\mathcal{S}$ is the set of all nodes H where one of $\mathbf{M}_{(H,P)}, \mathbf{J}_H, z|_{F_H}, y|_{E(H)}$ is updated.*
- *SumAncestors($H\in\mathcal{T}$): Return $\sum_{\text{ancestor } A \text{ of } H}\mathbf{M}_{H\leftarrow A}\mathbf{I}_{F_A}z$.*
- *Estimate($H\in\mathcal{T}$): Return $\Phi_{E(H)}(y+\mathbf{M}z)$.*
- *Query($H\in\mathcal{T}$): Return $(y+\mathbf{M}z)|_{E(H)}$.*

If we call Query on N nodes, the total runtime is $O(w\cdot T(\eta N))$.

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 $O(w\cdot T(\eta N))$.

Proof. First, we note that each edge operator $\mathbf{M}_e$ should be stored implicitly In particular, it suffices to only support the operation of computing $u^\top\mathbf{M}_e$ and $\mathbf{M}_e x$ for any vectors u and x.

We prove the running time and correctness for each procedure.

ALGORITHM 9: Data structure for maintaining $\Phi(\boldsymbol{y} + \mathbf{M}\boldsymbol{z})$, Part 1

```
1: data structure MaintainSketch
2: private : member
3:     $\mathcal{T}$ : rooted constant degree tree, where at every node $H$, there is
4:             $\boxed{\Phi\mathbf{M}^{(H)}}$ : sketch of partial tree operator
5:             $\boxed{\Phi\overline{\mathbf{M}^{(H)}\boldsymbol{z}}}$ : sketched vector                ▷ gives $\Phi\mathbf{M}\boldsymbol{z}$ at the root
6:             $\boxed{\Phi\boldsymbol{y}|_{E(H)}}$ : sketched subvector of $\boldsymbol{y}$
7:                                  ▷ boxes around expressions are to indicate they are sketched vectors
8:     $\Phi \in \mathbb{R}^{w\times m}$ : JL-sketch matrix
9:     $\mathbf{M}$ : tree operator on $\mathcal{T}$
10:    $\boldsymbol{z} \in \mathbb{R}^n$ : vector $\boldsymbol{z}$
11:    $\boldsymbol{y} \in \mathbb{R}^n$ : vector $\boldsymbol{y}$                ▷ $\mathbf{M}, \boldsymbol{z}, \boldsymbol{y}$ are pointers to read-only memory

12: procedure Initialize(rooted tree $\mathcal{T}$, $\Phi \in \mathbb{R}^{w\times m}$, tree operator $\mathbf{M}$, $\boldsymbol{z}$, $\boldsymbol{y}$)
13:     $\Phi \leftarrow \Phi, \mathcal{T} \leftarrow \mathcal{T}$
14:     $\boxed{\Phi\mathbf{M}^{(H)}} \leftarrow \mathbf{0}, \boxed{\Phi\overline{\mathbf{M}^{(H)}\boldsymbol{z}}} \leftarrow \mathbf{0}, \boxed{\Phi\boldsymbol{y}|_{E(H)}} \leftarrow \mathbf{0}$ for all $H \in \mathcal{T}$
15:     Update($\mathbf{M}, \boldsymbol{z}, \boldsymbol{y}, V(\mathcal{T})$)
16: end procedure

17: procedure Update($\mathbf{M}^{(\text{new})}, \boldsymbol{z}^{(\text{new})}, \boldsymbol{y}^{(\text{new})}, \mathcal{S} \overset{\text{def}}{=}$ set of nodes admitting changes)
18:     $\mathbf{M} \leftarrow \mathbf{M}^{(\text{new})}, \boldsymbol{z} \leftarrow \boldsymbol{z}^{(\text{new})}, \boldsymbol{y} \leftarrow \boldsymbol{y}^{(\text{new})}$
19:     for $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{S})$ by increasing node level do
20:         if $H$ is a leaf then
21:             $\boxed{\Phi\mathbf{M}^{(H)}} \leftarrow \Phi\mathbf{J}_H$
22:             $\boxed{\Phi\overline{\mathbf{M}^{(H)}\boldsymbol{z}}} \leftarrow \Phi\mathbf{J}_H\boldsymbol{z}|_{F_H}$
23:             $\boxed{\Phi\boldsymbol{y}|_{E(H)}} \leftarrow \Phi\boldsymbol{y}|_{E(H)}$
24:         else
25:             $\boxed{\Phi\mathbf{M}^{(H)}} \leftarrow \sum_{\text{child } D \text{ of } H} \boxed{\Phi\mathbf{M}^{(D)}}\mathbf{M}_{(D,H)}$
26:             $\boxed{\Phi\overline{\mathbf{M}^{(H)}\boldsymbol{z}}} \leftarrow \boxed{\Phi\mathbf{M}^{(H)}}\boldsymbol{z}|_{F_H} + \sum_{\text{child } D \text{ of } H} \boxed{\Phi\overline{\mathbf{M}^{(D)}\boldsymbol{z}}}$
27:             $\boxed{\Phi\boldsymbol{y}|_{E(H)}} \leftarrow \sum_{\text{child } D \text{ of } H} \boxed{\Phi\boldsymbol{y}|_{E(D)}}$
28:         end if
29:     end for
30: end procedure
```

Initialize: It sets the sketches to $\mathbf{0}$ in $O(w \cdot m)$ time It then calls Update with the initial $\mathbf{M}$, $\boldsymbol{z}$, $\boldsymbol{y}$, and updates the sketches everywhere on $\mathcal{T}$ By the runtime and correctness of Update, this step is correct and runs in $\widetilde{O}(w \cdot T(m))$ time.

Update($\mathbf{M}^{(\text{new})}, \boldsymbol{z}^{(\text{new})}, \boldsymbol{y}^{(\text{new})}$)*:* Let $\mathcal{S}$ denote the set of nodes admitting changes as defined in the theorem statement. If a node H is not in $\mathcal{S}$ and it has no descendants in $\mathcal{S}$, then by definition, $\mathbf{M}^{(H)}$ and $\overline{\mathbf{M}^{(H)}\boldsymbol{z}}$ are not affected by the updates in $\mathbf{M}$ and $\boldsymbol{z}$. Similarly, in this case, $\boldsymbol{y}|_{E(H)}$ is not affected by the updates to $\boldsymbol{y}$ Hence, it suffices to update the sketches only at all nodes in $\mathcal{P}_{\mathcal{T}}(\mathcal{S})$ We update the nodes from the bottom level of the tree upwards, so that when we're at a node H, all the sketches at its descendant nodes are correct. Hence, by definition, the sketch at H is also correct.

ALGORITHM 9: Data structure for maintaining $\Phi(\boldsymbol{y} + \mathbf{M}\boldsymbol{z})$, part 2

```
31: procedure SumAncestors($H \in \mathcal{T}$)
32:     if Update has not been called since the last call to SumAncestors($H$) then
33:         return the result of the last SumAncestors($H$)
34:     end if
35:     if $H$ is the root then return 0
36:     end if
37:     return $\mathbf{M}_{(H,P)}(z|_{F_P}$ + SumAncestors($P$))          ▷ $P$ is the parent of $H$
38: end procedure

39: procedure Estimate($H \in \mathcal{T}$)
40:     Let $\boldsymbol{u}$ be the result of SumAncestors($H$)
41:     return $\boxed{\Phi\mathbf{M}^{(H)}}\boldsymbol{u} + \boxed{\Phi\overline{\mathbf{M}^{(H)}}\boldsymbol{z}} + \boxed{\Phi\boldsymbol{y}|_{E(H)}}$
42: end procedure

43: procedure Query(leaf $H \in \mathcal{T}$)
44:     return $\boldsymbol{y}|_{E(H)} + \mathbf{J}_H(z|_{F_H}$ + SumAncestors($H$))
45: end procedure
```

To compute the runtime, first note $|\mathcal{P}_{\mathcal{T}}(\mathcal{S})| = O(\eta|\mathcal{S}|)$ since for each node $H \in \mathcal{S}$, the set includes all the $O(\eta)$ nodes on the path from H to the root For each leaf node $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{S})$, we can compute its sketches in constant time. For each non-leaf node $H \in \mathcal{S}$ with children D_1, D_2 Line 25 multiplies each row of $\boxed{\Phi\mathbf{M}^{(D_1)}}$ with $\mathbf{M}_{(D_1,H)}$, each row of $\boxed{\Phi\mathbf{M}^{(D_2)}}$ with $\mathbf{M}_{(D_2,H)}$, and sums the results. For a fixed row number, the total time over all $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{S})$ is bounded by $O(T(|\mathcal{P}_{\mathcal{T}}(\mathcal{S})|))$. So the total time for Line 25 in the procedure is $O(w \cdot T(\eta|S|))$.

Line 26 multiply each row of $\boxed{\Phi\mathbf{M}^{(H)}}$ with a vector and then performs a constant number of additions of $O(w)$-length vectors. Since $\boxed{\Phi\mathbf{M}^{(H)}}$ is computed for all $H \in T(|\mathcal{P}_{\mathcal{T}}(\mathcal{S})|)$ in $O(w \cdot T(\eta|S|))$ total time, this runtime must also be a bound on the number of total non-zero entries. Since each $\boxed{\Phi\mathbf{M}^{(H)}}$ is used once in Line 26 for a matrix-vector multiplication, the total runtime over all H is also $O(w \cdot T(\eta|S|))$. Lastly, the vector additions across all H takes $O(w \cdot \eta|S|)$ time.

Line 27 adds two vectors of length w. This is not the bottleneck.

SumAncestors(H): 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

$$\sum_{\text{ancestor } A \text{ of } H} \mathbf{M}_{H \leftarrow A}\mathbf{I}_{F_A}\boldsymbol{z} = \mathbf{M}_{(H,P)}\left(\mathbf{I}_{F_P}\boldsymbol{z} + \sum_{\text{ancestor } A \text{ of } P} \mathbf{M}_{P \leftarrow A}\mathbf{I}_{F_A}\boldsymbol{z}\right).$$

The procedure implements the right hand side, and is therefore, correct.

Estimate and Query: Their correctness follows from Lemmas 57 and 58, and the correctness of $\boxed{\Phi\boldsymbol{y}|_{E(H)}}$ maintained by Update.

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 overall cost for these calls is $O(w \cdot T(\eta N))$:

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 $\mathbf{M}_{(H,P)}$ with a vector Hence, the total runtime of SumAncestors is $T(|\mathcal{H}|)$. Furthermore, the total number of non-zero entries of the return values of these SumAncestors is also $O(T(|\mathcal{H}|))$.

Finally, each Query applies a constant-time operator $\mathbf{J}_H$ to the output of a unique SumAncestors call, so the overall runtime is certainly bounded by $O(T(|\mathcal{H}|))$. Adding a constant-sized $\boldsymbol{y}|_{E(H)}$ can be done efficiently. Similarly, each Estimate multiplies $\boxed{\boldsymbol{\Phi}\mathbf{M}^{(H)}}$ 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 $H \in \mathcal{H}$, the overall runtime is $O(w \cdot T(|\mathcal{H}|)) = O(w \cdot T(\eta N))$.

Query time on N leaves: Since this is a subset of the work described above, the runtime must also be bounded by $O(w \cdot T(\eta N))$. □

6.4 Proof of Theorem 8

We combine the previous three subsections for the overall approximation procedure. It is essentially AbstractMaintainApprox in Algorithm 7, with the abstractions replaced by a data structure implementation. The corresponding pseudocode is omitted.

Proof of Theorem 8. The data structure AbstractMaintainApprox in Algorithm 7 performs the correct vector approximation maintenance, however, it is not completely implemented. MaintainApprox simply replaces the abstractions with a concrete implementation using the data structure MaintainSketch from Algorithm 9.

First, for notation purposes, let $\boldsymbol{z} \overset{\text{def}}{=} c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})}$, and let $\boldsymbol{x} \overset{\text{def}}{=} \boldsymbol{y} + \mathbf{M}\boldsymbol{z}$, so that at step k, Approximate procedure has $\boldsymbol{x}^{(k)}$ (in implicit form) as input, and return $\overline{\boldsymbol{x}}$ Let $\ell \in \{1, \dots, O(\log m)\}$. We define a new dynamic vector $\boldsymbol{x}_\ell$ *symbolically*, which is represented at each step k for $k \geq 2^\ell$ by

$$\boldsymbol{x}_\ell^{(k)} \overset{\text{def}}{=} \boldsymbol{y}_\ell^{(k)} + \mathbf{M}_\ell^{(k)} \boldsymbol{z}_\ell^{(k)},$$

where the new tree operator $\mathbf{M}_\ell$ at step k is given by

– $\mathbf{M}_{\ell\ (H,P)}^{(k)} = \operatorname{diag}(\mathbf{M}_{(H,P)}^{(k)}, \mathbf{M}_{(H,P)}^{(k-2^\ell)})$ for each child-parent edge (H,P) in $\mathcal{T}$,
– $\mathbf{J}_{\ell\ H}^{(k)} = \overline{\mathbf{D}}_{E(H),E(H)}[\mathbf{J}_H^{(k)}\ \mathbf{J}_H^{(k-2^\ell)}]$ for each leaf node $H \in \mathcal{T}$,

where $\overline{\mathbf{D}}$ is the diagonal matrix defined in FindLargeCoordinates, with $\overline{\mathbf{D}}_{i,i} = \mathbf{D}_{i,i}^{(k)}$ at step k if $\overline{\boldsymbol{x}}_i$ has not been updated after step $k - 2^\ell$, and zero, otherwise.

At step k, the vector $\boldsymbol{y}_\ell$ is given by $\boldsymbol{y}_\ell^{(k)} = \overline{\mathbf{D}}^{1/2}(\boldsymbol{y}^{(k)} - \boldsymbol{y}^{(k-2^\ell)})$, and $\boldsymbol{z}_\ell$ by $\boldsymbol{z}_\ell^{(k)} \overset{\text{def}}{=} [\boldsymbol{z}^{(k)}\ \boldsymbol{z}^{(k-2^\ell)}]^\top$. Then, at each step k with $k \geq 2^\ell$, we have

$$\begin{aligned} \boldsymbol{x}_\ell^{(k)} &\overset{\text{def}}{=} \boldsymbol{y}_\ell^{(k)} + \mathbf{M}_\ell^{(k)} \boldsymbol{z}_\ell^{(k)} \qquad (6.1)\\ &= \left(\overline{\mathbf{D}}^{1/2}\boldsymbol{y}^{(k)} + \overline{\mathbf{D}}^{1/2}\mathbf{M}^{(k)}\boldsymbol{z}^{(k)}\right) - \left(\overline{\mathbf{D}}^{1/2}\boldsymbol{y}^{(k-2^\ell)} + \overline{\mathbf{D}}^{1/2}\mathbf{M}^{(k-2^\ell)}\boldsymbol{z}^{(k-2^\ell)}\right)\\ &= \overline{\mathbf{D}}^{1/2}(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-2^\ell)}). \end{aligned}$$

Note this is precisely the vector $\boldsymbol{q}$ for a fixed ℓ in FindLargeCoordinates in Algorithm 7. It is straightforward to see that $\mathbf{M}_\ell$ indeed satisfies the definition of a tree operator. Furthermore, $\mathbf{M}_\ell$ has the same complexity as $\mathbf{M}$. MaintainApprox will contain $O(\log m)$ copies of the MaintainSketch data structures in total, where the ℓth copy sketches $\boldsymbol{x}_\ell$ as it changes throughout the IPM algorithm.

We now describe each procedure in words, and then prove their correctness and runtime.

Initialize$(\mathcal{T}, \mathbf{M}, c, z^{(\text{step})}, z^{(\text{sum})}, y, \rho, \delta)$: This procedure implements the initialization of AbstractMaintainApprox, where the dynamic vector x to be approximated is represented by $x \stackrel{\text{def}}{=} y + \mathbf{M}(cz^{(\text{step})} + z^{(\text{sum})})$. The initialization steps described in Algorithm 7 takes $O(wm)$ time. Let $\mathbf{\Phi}$ denote the JL-sketching matrix.

We initialize two copies of the MaintainSketch data structure, `ox_cur` and `ox_prev`. At step k, `ox_cur` will maintain sketches of $\mathbf{\Phi} x^{(k)}$, and `ox_prev` will maintain sketches of $\mathbf{\Phi} x^{(k-1)}$. (The latter is initialized at step 1, but we consider it as part of initialization.)

In addition, for each $0 \le \ell \le O(m)$, we initialize a copy $\texttt{sketch}_\ell$ of MaintainSketch. These are needed for the implementation of FindLargeCoordinates(ℓ) in Approximate. Specifically, at step $k = 2^\ell$ of the IPM, we initialize $\texttt{sketch}_\ell$ by calling $\texttt{sketch}_\ell$.Initialize$(\mathcal{T}, \mathbf{\Phi}, \mathbf{M}_\ell^{(k)}, z_\ell^{(k)}, y_\ell^{(k)})$. (Although this occurs at step $k > 0$, we charge its runtime according to its function as part of initialization.)

The total initialization time is $O(wm \log m) = O(m\eta^2 \log m \log(\frac{m}{\rho}))$ by Lemma 59 By the existing pseudocode in Algorithm 7, it correctly initializes $\overline{x} \leftarrow x$.

Approximate$(\mathbf{M}^{(\text{new})}, c^{(\text{new})}, z^{(\text{step})^{(\text{new})}}, z^{(\text{sum})^{(\text{new})}}, y^{(\text{new})})$: This is Approximate in Algorithm 7 We consider when the current step is k.

First, we update the sketch data structures $\texttt{sketch}_\ell$ for each ℓ via $\texttt{sketch}_\ell$.Update. Recall at step k, $\texttt{sketch}_\ell$ maintains sketches for the vector $x_\ell^{(k)} = \overline{\mathbf{D}}^{1/2}(x^{(k)} - x^{(k-2^\ell)})$, although the actual representation in $\texttt{sketch}_\ell$ of the vector x_ℓ is given by $x_\ell = y_\ell + \mathbf{M}_\ell z_\ell$ as defined in Equation (6.1).

Next, we execute the pseudocode given in Approximate in Algorithm 7:

To update $\overline{x}_e$ to $x_e^{(k-1)}$ for a single coordinate in Algorithm 7 we find the leaf node H containing the edge e, and call `ox_cur`.Query(H) This returns the subvector $x^{(k-1)}|_{E(H)}$, from which we can make the assignment to $\overline{x}_e$.

In the subroutine FindLargeCoordinates(ℓ) the vector q defined in the pseudocode is exactly $x_\ell^{(k)}$. We get $\mathbf{\Phi}_{E(u)} q$ at a node u by calling $\texttt{sketch}_\ell$.Estimate(u), and we get the value of $q|_{E(u)}$ at a leaf node u by calling $\texttt{sketch}_\ell$.Query(u).

Number of coordinates changed in $\overline{x}$ during Approximate. The procedure collects a set of coordinates for which we update $\overline{x}$, by calling FindLargeCoordinates(ℓ) for each $0 \le \ell \le \ell_k$, where ℓ_k is defined to be the number of trailing zeros in the binary representation of k (These are exactly the values of ℓ such that $k \equiv 0 \mod 2^\ell$). In each call of FindLargeCoordinates(ℓ) There are $O(2^{2\ell}(\eta/\delta)^2 \log^2 m \log(m/\rho))$ iterations of the outer for-loop and $O(1)$ iterations of the inner while-loop by the assumption of $\|x^{(k+1)} - x^{(k)}\|_{\mathbf{D}^{(k)}} \le \beta$ and Lemma 54 Each iteration of the while-loop adds a $O(1)$ sized set to the collection I of candidate coordinates. So overall, FindLargeCoordinates(ℓ) returns a set of size $O(2^{2\ell}(\eta/\delta)^2 \log^2 m \log(m/\rho))$. Summing up over all calls of FindLargeCoordinates, the total size of the set of coordinates to update is

$$N_k \stackrel{\text{def}}{=} \sum_{\ell=0}^{\ell_k} O(2^{2\ell}(\beta/\delta)^2 \log^2 m \log(m/\rho)) = O(2^{2\ell_k}(\beta/\delta)^2 \log^2 m). \tag{6.2}$$

We define $\ell_0 = N_0 = 0$ for convenience.

Changes to sketching data structures. Let $\mathcal{S}^{(k)}$ denote the set of nodes H, where one of (when applicable) $\mathbf{M}_{(H,P)}$, $\mathbf{J}_H$, $z^{(\text{step})}|_{F_H}$, $z^{(\text{sum})}|_{F_H}$, y_{E_H} changes during step k (They are entirely induced by changes in v and w at step k.) We store $\mathcal{S}^{(k)}$ for each step.

For each ℓ, 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 $\overline{x}_i$ changes at a step Thus, the number of coordinate changes to $\overline{\mathbf{D}}$ at step k is the number of changes to $\mathbf{D}$, plus $N_{k-1} + N_{k-2^\ell}$: N_{k-1} entries are zeroed out because of updates to $\overline{\boldsymbol{x}}_i$ in step $k-1$. The N_{k-2^ℓ} entries that were zeroed out in step $k-2^\ell+1$ because of the update to $\overline{\boldsymbol{x}}_i$ in step $k-2^\ell$ are back.

Hence, at step k, the updates to $\texttt{sketch}_\ell$ are induced by updates to $\overline{\mathbf{D}}$, and the updates to $\boldsymbol{x}$ at step k, and at step $k-2^\ell$ The updates to the two $\boldsymbol{x}$ terms are restricted to the nodes $\mathcal{S}^{(k-2^\ell)} \cup \mathcal{S}^{(k)}$ in $\mathcal{T}$ for Algorithm 9. Updates to `ox_cur` and `ox_prev` can be similarly analyzed.

Runtime of Approximate. First, we consider the time to update each $\texttt{sketch}_\ell$: At step k, the analysis above combined with Lemma 59 show that $\texttt{sketch}_\ell$.Update with new iterations of the appropriate variables run in time

$$\begin{aligned}&O\left(w \cdot T\left(\eta \cdot \left(|\mathcal{S}^{(k)}| + |\mathcal{S}^{(k-2^\ell)}| + N_{k-1} + N_{k-2^\ell}\right)\right)\right)\\ &\leq w \cdot O\left(T\left(\eta \cdot \left(|\mathcal{S}^{(k)}| + N_{k-1} + N_{k-2^\ell}\right)\right)\right) + w \cdot O\left(T\left(\eta \cdot |\mathcal{S}^{(k-2^\ell)}|\right)\right),\end{aligned}$$

where we use the concavity of T The second term can be charged to step $k-2^\ell$ Thus, the amortized time cost for $\texttt{sketch}_\ell$.Update at step k is

$$w \cdot O(T(\eta \cdot (|\mathcal{S}^{(k)}| + N_{k-1} + N_{k-2^{\ell_k}}))).$$

Summing over all $0 \leq \ell \leq O(\log m)$ for the different copies of $\texttt{sketch}_\ell$, we get an extra $O(\log m)$ factor in the overall update time.

Similarly, we can update `ox_prev` and `ox_cur` in the same amortized time.

Finally, we analyze the remainder of the procedure, which consists of FindLargeCoordinates(ℓ) for each $0 \leq \ell \leq \ell_k$ and the subsequent updates to entries of $\overline{\boldsymbol{x}}$: For each FindLargeCoordinates(ℓ) call, by Lemma 54, $N_{k,\ell} \overset{\text{def}}{=} \Theta(2^{2\ell}(\beta/\delta)^2 \log^2 m \log(m/\rho))$ sampling paths are explored in the $\texttt{sketch}_\ell$ data structure, where each sampling path correspond to one iteration of the while-loop. We calculate $\|\mathbf{\Phi}_{E(H)}\boldsymbol{x}_\ell\|_2^2$ at a node H in the sampling path using $\texttt{sketch}_\ell$.Estimate(H), and at a leaf node H using $\texttt{sketch}_\ell$.Query(H) The total time is $w \cdot O(T(\eta \cdot N_{k,\ell}))$ by Lemma 59. To update a coordinate $i \in E(H)$ that was identified to be large, we can refer to the output of $\texttt{sketch}_\ell$.Query(H) from the sampling step.

Summing over each $0 \leq \ell \leq \ell_k$, we see that the total time for the FindLargeCoordinates calls and the subsequent updates fo $\overline{\boldsymbol{x}}$ is

$$\sum_{\ell=0}^{\ell_k} w \cdot O(T(\eta \cdot N_{k,\ell})) = w \cdot O(T(\eta \cdot N_k)),$$

where N_k is the number of coordinates that are updated in $\overline{\boldsymbol{x}}$ as shown in Equation (6.2).

Combined with the update times, we conclude that the total amortized cost of Approximate at step k is

$$\Theta\left(\eta^2 \log\left(\frac{m}{\rho}\right) \log m\right) \cdot T\left(\eta \cdot \left(|\mathcal{S}^{(k)}| + N_{k-1} + N_{k-2^{\ell_k}}\right)\right).$$

Observe that $N_{k-1} = N_{k-2^0}$ and N_{k-2^ℓ} are both bounded by $O(N_{k-2^{\ell_k}})$: When $\ell \neq \ell_k$, the number of trailing zeros in $k-2^\ell$ is no more than ℓ_k. When $\ell = \ell_k$, the number of trailing zeros of $k-2^{\ell_k}$ is $\ell_{k-2^{\ell_k}}$. In both cases, $\ell_{k-2^\ell} \leq \ell_{k-2^{\ell_k}}$ So we have the desired overall runtime. □

7 Slack and Flow Projection

In this section, we define the slack and flow tree operator as required to use MaintainRep, and then give the data structures for the solution maintenance.

7.1 Tree Operator for Slack

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 $\widetilde{\mathbf{L}}^{\dagger}$ denote the approximation of $\mathbf{L}^{\dagger}$ from Equation (3.6), maintained and computable with a DynamicSC data structure. If we define

$$\widetilde{\mathbf{P}}_{\boldsymbol{w}} = \mathbf{W}^{1/2}\mathbf{B}\widetilde{\mathbf{L}}^{\dagger}\mathbf{B}^{\top}\mathbf{W}^{1/2} = \mathbf{W}^{1/2}\mathbf{B}\mathbf{\Pi}^{(0)\top}\cdots\mathbf{\Pi}^{(\eta-1)\top}\widetilde{\mathbf{\Gamma}}\mathbf{\Pi}^{(\eta-1)}\cdots\mathbf{\Pi}^{(0)}\mathbf{B}^{\top}\mathbf{W}^{1/2}.$$

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 $\mathbf{M} \stackrel{\text{def}}{=} \mathbf{W}^{1/2}\mathbf{B}\mathbf{\Pi}^{(0)\top}\cdots\mathbf{\Pi}^{(\eta-1)\top}$ Then we set $\mathbf{M}^{(\text{slack})} \stackrel{\text{def}}{=} \mathbf{W}^{-1/2}\mathbf{M}$.

For the remainder of the section, we use $\boldsymbol{z}$ to mean $\boldsymbol{z}^{(k)}$ for one IPM step k.

Definition 60 (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

$$\mathbf{M}_{(H,P)} \stackrel{\text{def}}{=} \mathbf{I}_{\partial H \cup F_H} - \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1}\mathbf{L}^{(H)}_{F_H,\partial H}. \tag{7.1}$$

For notational convenience, let $\mathbf{X}^{(H)\top} \stackrel{\text{def}}{=} (\mathbf{L}^{(H)}_{F_H,F_H})^{-1}\mathbf{L}^{(H)}_{F_H,\partial H}$.

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 61. *Let* $\mathbf{M}$ *be the tree operator as defined in Definition 60. 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 62. *For any* $0 \le i < \eta$, *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* $0 \le i < j < \eta$,

$$\mathbf{\Pi}^{(i)\top}\cdots\mathbf{\Pi}^{(j-1)\top}\boldsymbol{x} = \boldsymbol{x} + \boldsymbol{y},$$

where $\boldsymbol{y}$ *is a vector supported on* $F_i \cup \cdots \cup F_{j-1} = \cup_{H:i \le \eta(H) < j} F_H$. *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 63. *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*

$$\mathbf{M}_{H \leftarrow A}\boldsymbol{z}|_{F_A} = \mathbf{I}_{\partial H \cup F_H}\mathbf{\Pi}^{(0)\top}\cdots\mathbf{\Pi}^{(\eta-1)\top}\boldsymbol{z}|_{F_A}. \tag{7.2}$$

Proof. 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 ($H_0 \overset{\text{def}}{=} H, H_1, \ldots, H_t \overset{\text{def}}{=} A$), for some $t \leq \eta(A)$. We will prove by induction for k decreasing from t to 0:

$$\mathbf{M}_{H_k\leftarrow A}\boldsymbol{z}|_{F_A} = \mathbf{I}_{V(H_k)}\mathbf{\Pi}^{(\eta(H_k))\top}\mathbf{\Pi}^{(\eta(H_k)+1)\top}\cdots\mathbf{\Pi}^{(\eta(A)-1)\top}\boldsymbol{z}|_{F_A}. \tag{7.3}$$

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

$$\mathbf{M}_{H_{k+1}\leftarrow A}\boldsymbol{z}|_{F_A} = \mathbf{I}_{V(H_{k+1})}\mathbf{\Pi}^{(\eta(H_{k+1}))\top}\cdots\mathbf{\Pi}^{(\eta(A)-1)\top}\boldsymbol{z}|_{F_A}. \tag{7.4}$$

Multiplying by the edge operator $\mathbf{M}_{(H_k,H_{k+1})}$ on both sides gives

$$\mathbf{M}_{H_k\leftarrow A}\boldsymbol{z}|_{F_A} = \mathbf{M}_{(H_k,H_{k+1})}\mathbf{I}_{V(H_{k+1})}\mathbf{\Pi}^{(\eta(H_{k+1}))\top}\cdots\mathbf{\Pi}^{(\eta(A)-1)\top}\boldsymbol{z}|_{F_A}. \tag{7.5}$$

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 $\boldsymbol{x} \overset{\text{def}}{=} \mathbf{\Pi}^{(\eta(H_{k+1}))\top}\cdots\mathbf{\Pi}^{(\eta(A)-1)\top}\boldsymbol{z}|_{F_A}$. Now, by the definition of the edge operator, the above equation becomes

$$\mathbf{M}_{H_k\leftarrow A}\boldsymbol{z}|_{F_A} = (\mathbf{I}_{V(H_k)} - \mathbf{X}^{(H_k)\top})\boldsymbol{x}. \tag{7.6}$$

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 62. In particular, $\boldsymbol{y}$ is zero on F_{H_k}. Also, $\boldsymbol{y}$ is zero on ∂H_k, since by Observation 23, $\partial H_k \subseteq \cup_{\text{ancestor } A' \text{ of } H_k} F_{A'}$, 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 Equation (7.6) 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 61, we apply the leaf operators to the result of the previous lemma and sum over all nodes and leaf nodes.

Proof of Lemma 61. Let H be a leaf node. We sum Equation (7.2) 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 62, 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 $\mathbf{I}_{\partial H\cup F_H}$ on 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

$$\sum_{H\in\mathcal{T}(0)}\sum_{A:H\in\mathcal{T}_A} \mathbf{J}_H\mathbf{M}_{H\leftarrow A}\boldsymbol{z}|_{F_A} = \mathbf{W}^{1/2}\left(\sum_{H\in\mathcal{T}(0)}\mathbf{B}[H]\right)\mathbf{\Pi}^{(0)\top}\cdots\mathbf{\Pi}^{(\eta-1)\top}\boldsymbol{z} \tag{7.7}$$
$$\mathbf{M}\boldsymbol{z} = \mathbf{W}^{1/2}\mathbf{B}\mathbf{\Pi}^{(0)\top}\cdots\mathbf{\Pi}^{(\eta-1)\top}\boldsymbol{z}.$$

□

The tree operator $\mathbf{M}$ defined in Definition 60 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 $\mathbf{M}^{(\text{slack})} \overset{\text{def}}{=} \mathbf{W}^{-1/2}\mathbf{M}$ and note it is clearly also a tree operator with the same complexity.

We now examine the slack tree operator complexity.

Lemma 64. *The complexity of the slack tree operator as defined in Definition 60 is $T(k) = \widetilde{O}(\sqrt{mk}\cdot \epsilon_{\mathbf{P}}^{-2})$, 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 $\mathbf{M}_{(D,P)} = \mathbf{I}_{\partial D} - (\mathbf{L}^{(D)}_{F_D,F_D})^{-1}\mathbf{L}^{(D)}_{F_D,\partial D}$ 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 $O(\epsilon_{\mathbf{P}}^{-2}|\partial D \cup F_D|)$ by Lemma 35 and Theorem 3.

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. To bound the total cost over k distinct edges, we apply Lemma 27, which then gives the claimed complexity. □

7.2 Additional Considerations for Flow

As discussed in Section 3.6 we intend to set the flow update $\tilde{\boldsymbol{f}} = \widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v} = \mathbf{M}\boldsymbol{z}$ where $\widetilde{\mathbf{P}}_{\boldsymbol{w}}$ is the same as was defined for slack using the tree operator $\mathbf{M}$ in Definition 60. This immediately satisfies $\|\tilde{\boldsymbol{f}} - \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}\|_2 \leq \widetilde{O}(\eta\epsilon_{\mathbf{P}})\|\boldsymbol{v}\|_2$, so one additional complication remains: feasibility constraints require the slack update to lie in Range$(\mathbf{W}^{1/2}\mathbf{B})$ while the flow update $\tilde{\boldsymbol{f}}$ needs to be a valid flow i.e., if $\boldsymbol{d} \overset{\text{def}}{=} \mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}$ denotes the vertex demands, then $\tilde{\boldsymbol{f}}$ needs to satisfy $\mathbf{B}^\top\mathbf{W}^{1/2}\tilde{\boldsymbol{f}} = \boldsymbol{d}$.

To resolve this, recall from Equation (7.7) that

$$\tilde{\boldsymbol{f}} = \mathbf{M}\boldsymbol{z} = \mathbf{W}^{1/2}\left(\sum_{\text{leaf } H}\mathbf{B}[H]\right)\mathbf{\Pi}^{(0)\top}\cdots\mathbf{\Pi}^{(\eta-1)\top}\boldsymbol{z},$$

while $\boldsymbol{d} \overset{\text{def}}{=} \mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v} = \sum_{\text{leaf } H}\mathbf{B}[H]^\top\mathbf{W}^{1/2}\boldsymbol{v}$; in other words, they both admit a decomposition at the leaf node level. We define the excess demand at each leaf node H by $\bar{\boldsymbol{d}}^{(H)} \overset{\text{def}}{=} \mathbf{B}[H]^\top\mathbf{W}^{1/2}(\boldsymbol{v} - \tilde{\boldsymbol{f}})$. The vector $\bar{\boldsymbol{d}}^{(H)}$ is indeed a demand on region H with its entries summing to 0, 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 $\|\boldsymbol{r}\|_2 \le \widetilde{O}(\eta\epsilon_{\mathbf{P}})\|\boldsymbol{v}\|_2$.

Let us first consider $\boldsymbol{r}|_{E(H)}$ for any leaf region H. We have

$$\begin{aligned}
\|\boldsymbol{r}|_{E(H)}\|_2^2 &\le O(1) \cdot \left(\|\boldsymbol{r}|_{E(H)}\|_\infty\right)^2 && \text{(regions are constant size)}\\
&\le O(1) \cdot \left(\frac{\|\bar{\boldsymbol{d}}^{(H)}\|_1}{\boldsymbol{w}_{\min}^{1/2}}\right)^2 && \text{($\boldsymbol{r}$ routes $\boldsymbol{d}$ in a weighted manner)}\\
&\le O\left(\boldsymbol{w}_{\min}^{-1}\right) \cdot \left(\|\bar{\boldsymbol{d}}^{(H)}\|_1\right)^2\\
&\le O\left(\boldsymbol{w}_{\min}^{-1}\right) \cdot \left(\|\bar{\boldsymbol{d}}^{(H)}\|_2\right)^2, && \text{(again regions are constant size)}
\end{aligned}$$

where $\boldsymbol{w}_{\min}$ is the minimum entry in $\mathbf{W}$, and $\boldsymbol{w}_{\max}$ is the maximum. Next, since $\|\tilde{\boldsymbol{f}} - \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}\|_2 \le \widetilde{O}(\eta\epsilon_{\mathbf{P}})\|\boldsymbol{v}\|_2$, and $\mathbf{B}$ is an adjacency matrix, we also know that

$$\left\|\sum_{\text{leaf } H} \bar{\boldsymbol{d}}^{(H)}\right\|_2^2 = \left\|\sum_{\text{leaf } H} \mathbf{B}[H]^\top \mathbf{W}^{1/2}(\boldsymbol{v} - \tilde{\boldsymbol{f}})\right\|_2^2 \le \widetilde{O}\left(\eta\epsilon_{\mathbf{P}}\boldsymbol{w}_{\max}\right) \cdot \|\boldsymbol{v}\|_2^2.$$

We are happy to incur poly(m) error. Hence, combining the two inequalities above with the bound on $\boldsymbol{w}_{\max}/\boldsymbol{w}_{\min}$ in (A.1), we get

$$\|\boldsymbol{r}\|_2^2 \le \widetilde{O}(\eta\epsilon_{\mathbf{P}} \cdot \text{poly}(m))\|\boldsymbol{v}\|_2^2,$$

as required.

In Appendix C, we give an alternative interpretation of the tree operator based on electrical flows.

7.3 Proof of Theorem 9

Finally, we present the overall solution maintenance data structure. We only give the flow version, as the slack is analogous and slightly simpler. In our pseudocode, we use a box around a vector to mean that vector is maintained using a data structure rather than directly accessible.

Proof of Theorem 9. We have the additional invariant that the IPM flow solution $\boldsymbol{f}$ can be recovered in the data structure by the identity

$$\boldsymbol{f} = \hat{\boldsymbol{f}} - \boldsymbol{f}^\perp, \tag{7.8}$$

where $\boldsymbol{f}^\perp$ is implicit maintained by maintainRep, and $\hat{\boldsymbol{f}}$ is implicitly maintained by the identity $\hat{\boldsymbol{f}} = \hat{\boldsymbol{f}}_0 + \hat{c}\mathbf{W}\boldsymbol{v}$.

We prove the runtime and correctness of each procedure separately Recall by Lemma 63, the tree operator $\mathbf{M}$ has complexity $T(K) = O(\epsilon_{\mathbf{P}}{}^{-2}\sqrt{mK})$.

Initialize: By the initialization of maintainRep (Theorem 7), the implicit representation of $\boldsymbol{f}^\perp$ is correct and $\boldsymbol{f}^\perp = \mathbf{0}$ We then set $\hat{\boldsymbol{f}} \stackrel{\text{def}}{=} \hat{\boldsymbol{f}}_0 + \hat{c}\mathbf{W}\boldsymbol{v} = \boldsymbol{f}^{(\text{init})}$. So overall, we have $\boldsymbol{f} \stackrel{\text{def}}{=} \hat{\boldsymbol{f}} + \boldsymbol{f}^\perp = \boldsymbol{f}^{(\text{init})}$. By the initialization of maintainApprox, $\overline{\boldsymbol{f}}$ is set to $\boldsymbol{f} = \boldsymbol{f}^{(\text{init})}$ to start.

Initialization of $\boxed{\boldsymbol{f}^\perp}$ takes $\widetilde{O}(m\epsilon_{\mathbf{P}}{}^{-2})$ time by Theorem 7, and initialization of $\boxed{\overline{\boldsymbol{f}}}$ takes $\widetilde{O}(m)$ time by Theorem 8.

ALGORITHM 10: Flow maintenance, main algorithm

```
data structure MaintainFlow
private: member
    $\boldsymbol{w} \in \mathbb{R}^m$: weight vector
    $\boldsymbol{v} \in \mathbb{R}^m$: direction vector
    $\boxed{\boldsymbol{f}^{\perp}} \in \mathbb{R}^m$: MaintainRep instance that implicitly maintains $\boldsymbol{f}^{\perp}$ where
        $$\boldsymbol{f}^{\perp} \overset{\text{def}}{=} \boldsymbol{y} + \mathbf{W}^{1/2}\mathbf{M}(c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})})$$
    $\hat{c} \in \mathbb{R}, \hat{\boldsymbol{f}}_0 \in \mathbb{R}^m$: scalar and vector to implicitly maintain $\hat{\boldsymbol{f}} \overset{\text{def}}{=} \hat{\boldsymbol{f}}_0 + \hat{c} \cdot \mathbf{W}\boldsymbol{v}$.
    $\boxed{\overline{\boldsymbol{f}}}$: MaintainApprox instance that implicitly maintains $\overline{\boldsymbol{f}}$ approximating $\boldsymbol{f}$

procedure Initialize($\mathcal{T}, \boldsymbol{f}^{(\text{init})} \in \mathbb{R}^m, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \epsilon_{\mathbf{P}} > 0, \overline{\epsilon} > 0$)
    $\boxed{\boldsymbol{f}^{\perp}}$.Initialize($\mathcal{T}, \mathbf{W}^{1/2}\mathbf{M}, \boldsymbol{v}, \boldsymbol{w}, \mathbf{0}, \epsilon_{\mathbf{P}}$)        ▷ initialize $\boldsymbol{f}^{\perp} \leftarrow \mathbf{0}$
    $\boldsymbol{w} \leftarrow \boldsymbol{w}, \boldsymbol{v} \leftarrow \boldsymbol{v}$
    $\hat{c} \leftarrow 0, \hat{\boldsymbol{f}}_0 \leftarrow \boldsymbol{f}^{(\text{init})}$        ▷ initialize $\hat{\boldsymbol{f}} \leftarrow \boldsymbol{f}^{(\text{init})}$
    $\boxed{\overline{\boldsymbol{f}}}$.Initialize($-\mathbf{W}^{1/2}\mathbf{M}, c, \boldsymbol{z}^{(\text{step})}, \boldsymbol{z}^{(\text{sum})}, -\boldsymbol{y} + \hat{\boldsymbol{f}}_0 + \hat{c} \cdot \mathbf{W}\boldsymbol{v}, \mathbf{W}^{-1}, n^{-5}, \overline{\epsilon}$)
                                                        ▷ initialize $\overline{\boldsymbol{f}} \leftarrow \boldsymbol{f}^{(\text{init})}$
end procedure

procedure Reweight($\Delta\boldsymbol{w} \in \mathbb{R}^m_{>0}$)
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta\boldsymbol{w}$
    $\boxed{\boldsymbol{f}^{\perp}}$.Reweight($\Delta\boldsymbol{w}$)
    $\hat{\boldsymbol{f}}_0 \leftarrow \hat{\boldsymbol{f}}_0 - \hat{c}(\Delta\mathbf{W})^{1/2}\boldsymbol{v}$
end procedure

procedure Move($\alpha, \Delta\boldsymbol{v} \in \mathbb{R}^m$)
    $\boldsymbol{v} \leftarrow \boldsymbol{v} + \Delta\boldsymbol{v}$
    $\boxed{\boldsymbol{f}^{\perp}}$.Move($\alpha, \Delta\boldsymbol{v}$)
    $\hat{\boldsymbol{f}}_0 \leftarrow \hat{\boldsymbol{f}}_0 - \hat{c}\mathbf{W}^{1/2}\Delta\boldsymbol{v}$
    $\hat{c} \leftarrow \hat{c} + \alpha$
end procedure

procedure Approximate( )
    return $\boxed{\overline{\boldsymbol{f}}}$.Approximate($-\mathbf{W}^{1/2}\mathbf{M}, c, \boldsymbol{z}^{(\text{step})}, \boldsymbol{z}^{(\text{sum})}, -\boldsymbol{y} + \hat{\boldsymbol{f}}_0 + \hat{c} \cdot \mathbf{W}\boldsymbol{v}, \mathbf{W}^{-1}$)
end procedure

procedure Exact( )
    $\boldsymbol{f}^{\perp} \leftarrow \boxed{\boldsymbol{f}^{\perp}}$.Exact()
    return $(\hat{\boldsymbol{f}}_0 + \hat{c} \cdot \mathbf{W}\boldsymbol{v}) - \boldsymbol{f}^{\perp}$
end procedure
```

Move: The call $\boxed{\boldsymbol{f}^{\perp}}$.Move($\alpha, \boldsymbol{v}^{(k)}$) updates the implicit representation of $\boldsymbol{f}^{\perp}$ by

$$\boldsymbol{f}^{\perp} \leftarrow \boldsymbol{f}^{\perp} + \mathbf{W}^{1/2}\mathbf{M}\alpha\boldsymbol{z}^{(k)},$$

where $\mathbf{M}$ is the flow projection tree operator defined in Definition 60. By Lemma 61, this is equivalent to the update

$$f^{\perp} \leftarrow f^{\perp} + \alpha \mathbf{W}^{1/2}\tilde{f},$$

where $\left\|\tilde{f} - \mathbf{P}_{w}\boldsymbol{v}^{(k)}\right\|_2 \leq O(\eta\epsilon_{\mathbf{P}}) \left\|\boldsymbol{v}^{(k)}\right\|_2$ and $\mathbf{B}^{\top}\mathbf{W}^{1/2}\tilde{f} = \mathbf{B}^{\top}\mathbf{W}^{1/2}\boldsymbol{v}^{(k)}$ by Theorem 71.

For the $\hat{f}$ term, let $\hat{f}', \hat{f}_0', \hat{c}', \boldsymbol{v}'$ be the state of $\hat{f}, \hat{f}_0, \hat{c}$ and $\boldsymbol{v}$ at the start of the procedure. At the end of the procedure, we have

$$\hat{f} \stackrel{\text{def}}{=} \hat{f}_0 + \hat{c}\mathbf{W}\boldsymbol{v} = \hat{f}_0' - \hat{c}'\mathbf{W}^{1/2}\Delta\boldsymbol{v} + (\hat{c}' + \alpha)\mathbf{W}\boldsymbol{v} = \hat{f}_0' + \hat{c}'\mathbf{W}^{1/2}\boldsymbol{v}' + \alpha\mathbf{W}^{1/2}\boldsymbol{v} = \hat{f}' + \alpha\mathbf{W}^{1/2}\boldsymbol{v},$$

so we have the correct update $\hat{f} \leftarrow \hat{f} + \alpha\mathbf{W}^{1/2}\boldsymbol{v}$. Combined with $f^{\perp}$, the update to f is

$$f \leftarrow f + \alpha\mathbf{W}^{1/2}\boldsymbol{v} - \alpha\mathbf{W}^{1/2}\tilde{f}.$$

By Theorem 7, if $\boldsymbol{v}^{(k)}$ differs from $\boldsymbol{v}^{(k-1)}$ on K coordinates, then the runtime of $\boxed{f^{\perp}}$.Move is $\widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}\sqrt{mK})$. Furthermore, $z^{(\text{step})}$ and $z^{(\text{sum})}$ change on F_H for at most $\widetilde{O}(K)$ nodes in $\mathcal{T}$. Updating $\hat{f}$ takes $O(K)$ time where $K \leq O(m)$, giving us the overall claimed runtime.

Exact: The runtime and correctness follow from the guarantee in Theorem 7 and the invariant that $f = \hat{f} - f^{\perp}$.

Approximate: By Theorem 8, the returned vector satisfies

$$\|\mathbf{W}^{-1/2}\left(\overline{f} - (\hat{f} - f^{\perp})\right)\|_{\infty} \leq \overline{\epsilon},$$

where $\hat{f}$ and $f^{\perp}$ are maintained in the current data structure.

Finally, we have the following lemma about the runtime for Approximate. Let $\overline{f}^{(k)}$ denote the returned approximate vector at step k.

Lemma 65. *Suppose $\alpha\|\boldsymbol{v}\|_2 \leq \beta$ for some β for all calls to Move. Let K denote the total number of coordinates changed in $\boldsymbol{v}$ and $\boldsymbol{w}$ between the $k-1$-th and kth Reweight and Move calls. Then at the kth Approximate call,*

- *The data structure first sets $\overline{f}_e \leftarrow f_e^{(k-1)}$ for all coordinates e where $\boldsymbol{w}_e$ changed in the last Reweight, then sets $\overline{f}_e \leftarrow f_e^{(k)}$ for $O(N_k \stackrel{\text{def}}{=} 2^{2\ell_k}(\frac{\beta}{\overline{\epsilon}})^2 \log^2 m)$ coordinates e, where ℓ_k is the largest integer ℓ with $k = 0 \mod 2^{\ell}$ when $k \neq 0$ and $\ell_0 = 0$.*
- *The amortized time for the kth Approximate call is $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{m(K + N_{k-2^{\ell_k}})})$.*

Proof. We apply Theorem 8 with $\boldsymbol{x} = f$ and diagonal matrix $\mathbf{D} = \mathbf{W}^{-1}$ We need to prove $\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\|_{\mathbf{D}^{(k-1)}} \leq O(\beta)$ for all k first. The constant factor in $O(\beta)$ does not affect the guarantees in Theorem 8. The left-hand side is

$$\begin{aligned}
\left\|f^{(k)} - f^{(k-1)}\right\|_{\mathbf{W}^{(k-1)^{-1}}} &= \left\|-\alpha^{(k)}\mathbf{M}z^{(k)} + \alpha^{(k)}\boldsymbol{v}^{(k)}\right\|_2 && \text{(by Move)} \\
&\leq \left\|-\alpha^{(k)}\mathbf{M}z^{(k)}\right\|_2 + \left\|\alpha^{(k)}\boldsymbol{v}^{(k)}\right\|_2 \\
&\leq (2 + O(\eta\epsilon_{\mathbf{P}}))\alpha^{(k)}\|\boldsymbol{v}^{(k)}\|_2 && \text{(by the assumption that } \alpha\|\boldsymbol{v}\|_2 \leq \beta) \\
&\leq 3\beta.
\end{aligned}$$

Now, we can apply the conclusions from Theorem 8 to get that at the kth step, the data structure first sets $\overline{f}_e \leftarrow f_e^{(k-1)}$ for all coordinates e where $\boldsymbol{w}_e$ changed in the last Reweight, then sets $\overline{f}_e \leftarrow f_e^{(k)}$ for $O(N_k \stackrel{\text{def}}{=} 2^{2\ell_k}(\frac{\beta}{\overline{\epsilon}})^2 \log^2 m)$ coordinates e, where ℓ_k is the largest integer ℓ with $k = 0 \mod 2^{\ell}$ when $k \neq 0$ and $\ell_0 = 0$.

For the second point, Move updates $\boldsymbol{z}^{(\mathrm{step})}$ and $\boldsymbol{z}^{(\mathrm{sum})}$ on F_H for $\widetilde{O}(K)$ different nodes $H \in \mathcal{T}$ by Theorem 7. Reweight then updates $\boldsymbol{z}^{(\mathrm{step})}$ and $\boldsymbol{z}^{(\mathrm{sum})}$ on F_H for $\widetilde{O}(K)$ different nodes, and updates the tree operator $\mathbf{W}^{-1/2}\mathbf{M}$ on $\widetilde{O}(K)$ different edge and leaf operators. In turn, it updates $\boldsymbol{y}$ on $E(H)$ for $\widetilde{O}(K)$ leaf nodes H. The changes of $\hat{\boldsymbol{f}}$ cause $O(K)$ changes to the vector $-\boldsymbol{y} + \hat{\boldsymbol{f}}_0 + \hat{c} \cdot \mathbf{W}\boldsymbol{v}$, which is the parameter $\boldsymbol{y}$ of Theorem 8. Now, we apply Theorem 8 and the complexity of the tree operator to conclude the desired amortized runtime. □

We have shown the correctness and runtime for each of the procedures in the data structure, so we are done. □

8 Min-cost Flow for Separable Graphs

In this section, we extend our result to α-separable graphs.

Corollary 2 (Separable Min-cost Flow). *Let $\mathcal{G}$ be an α-separable graph class, and suppose there is convex function s such that we can compute a balanced vertex-separator for any $G \in \mathcal{G}$ with m edges in $s(m)$ time. Then given connected $G \in \mathcal{G}$ with m edges, and integral demands $\boldsymbol{d}$, edge capacities $\boldsymbol{u}$, costs $\boldsymbol{c}$, all bounded by M in absolute value, there is an algorithm that computes a minimum-cost flow on G satisfying demand $\boldsymbol{d}$ in $\widetilde{O}((m + m^{1/2+\alpha}) \log^2 M + s(m))$ expected time.*

The change in running time essentially comes from the parameters of the separator tree which we discuss in Section 8.1. We then calculate the total running time and prove Corollary 2 in Section 8.2.

8.1 Separator Tree for Separable Graphs

Since our algorithm only exploits the separable property of the planar graphs, it can be applied to other separable graphs directly and yields different running times. Similar to the planar case, by adding two extra vertices to any α-separable graph, it is still α-separable with the constant c in Definition 18 increased by 2.

We define a separator tree $\mathcal{T}$ for an α-separable graph G in the same way as for a planar graph. The only two differences are their construction time and update time (for k-sparse updates). For the planar case, these are bounded by Corollary 25 and Lemma 27, respectively. We shall prove their analogs Lemma 66 and Lemma 67.

[28] showed that the separator tree can be constructed in $O(s(n) \log n)$ time for any class of 1/2-separable graphs where $s(n)$ is the time for computing the separator. The proof can be naturally extended to α-separable graphs. We include extended proofs for the following two lemmas in the appendix for completeness.

Lemma 66. *Let C be an α-separable class such that we can compute a balanced separator for any connected graph in C with n vertices and m edges in $s(m)$ time for some convex function $s(m) \geq m$. Given an α-separable graph $G \in C$, there is an algorithm that computes a separator tree $\mathcal{T}$ for G in $O(s(m) \log m)$ time.*

We then prove the update time. Same as the planar case, we define $\mathcal{P}_{\mathcal{T}}(H)$ to be the set of all ancestors of H in the separator tree and $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$ to be the union of $\mathcal{P}_{\mathcal{T}}(H)$ for all $H \in \mathcal{H}$. Then we have the following bound:

Lemma 67. *Let G be an α-separable graph with separator tree $\mathcal{T}$ constructed via Lemma 66. Let $\mathcal{H}$ be a set of K nodes in $\mathcal{T}$. Then*

$$\sum_{H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})} |\partial H| + |S(H)| \leq \widetilde{O}(K^{1-\alpha} m^{\alpha}).$$

By setting α as 1/2, we get Lemma 27 for planar graphs as a corollary.

8.2 Proof of Runtime

In this section, we prove Corollary 2. The data structures (except for the construction of the separator tree) will use exactly the same pseudocode as for the planar case. Thus, the correctness can be proven in the same way. We prove the runtimes only.

We first construct the separator tree in $O(s(m)\log m)$ time by Lemma 66. Then we can simply change the runtime from $\widetilde{O}(\sqrt{mK})$ in Lemma 27 to $\widetilde{O}(m^\alpha K^{1-\alpha})$ in Lemma 67 to all the data structures and to the complexity $T(\cdot)$ of the flow and slack tree operators.

Lemma 68. *For any α-separable graph G with separator tree $\mathcal{T}$, the flow and slack operators defined in Definition 60 both have complexity $T(K) = O(\epsilon_{\mathbf{P}}{}^{-2}K^{1-\alpha}m^\alpha)$.*

Proof. The leaf operators of both the flow and slack tree operators has constant size. Let $\mathbf{M}_{(H,P)}$ be a tree edge operator. Note that it is a symmetric matrix. For the slack operator, Applying $\mathbf{M}_{(D,P)} = \mathbf{I}_{\partial D} - (\mathbf{L}^{(D)}_{F_D,F_D})^{-1}\mathbf{L}^{(D)}_{F_D,\partial D}$ 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 $\boldsymbol{b}$. For the flow operator, $\mathbf{M}_{(H,P)}\boldsymbol{u}$ consists of multiplying with $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ and solving the Laplacian system $\mathbf{L}^{(H)}$.

Each of the steps costs time $O(\epsilon_{\mathbf{P}}{}^{-2}|\partial D \cup F_D|)$ by Lemma 35 and Theorem 3. To bound the total cost over K distinct edges, we apply Lemma 67 instead of Lemma 27, which gives the claimed complexity. □

With the updated subroutine runtimes, we can prove Corollary 2.

Proof of Corollary 2. The correctness is exactly the same as the proof for Theorem 1. For the runtime, we may assume $\alpha > 1/2$ because otherwise the graph is 1/2-separable and the runtime follows from Theorem 1. The amortized time for the kth IPM step is

$$\widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}m^\alpha(K + N_{k-2^{\ell_k}})^{1-\alpha}),$$

where $N_k \overset{\text{def}}{=} 2^{2\ell_k}(\beta/\alpha)^2\log^2 m = O(2^{2\ell_k}\log^2 m)$, where $\alpha = O(1/\log m)$ and $\epsilon_{\mathbf{P}} = O(1/\log m)$.

Observe that $K + N_{k-2^{\ell_k}} = O(N_{k-2^{\ell_k}})$. Summing over all T steps, the total time is

$$\begin{aligned} O(m^\alpha \log m)\sum_{k=1}^{T}(N_{k-2^{\ell_k}})^{1-\alpha} &= O(m^\alpha\log^2 m)\sum_{k=1}^{T}2^{2(1-\alpha)\ell_{(k-2^{\ell_k})}} \\ &= O(m^\alpha\log^2 m)\sum_{k'=1}^{T}2^{2(1-\alpha)\ell_{k'}}\sum_{k=1}^{T}[k-2^{\ell_k}=k'], \\ &= O(m^\alpha\log^2 m\log T)\sum_{k'=1}^{T}2^{2(1-\alpha)\ell_{k'}}. \qquad (8.1)\end{aligned}$$

Without $1-\alpha$ in the exponent, recall from the planar case that

$$\sum_{k'=1}^{T}2^{\ell_{k'}} = \sum_{i=0}^{\log T}2^i\cdot T/2^{i+1} = O(T\log T).$$

The summation from Equation (8.1) is

$$\begin{aligned}\sum_{k=1}^{T}2^{2(1-\alpha)\ell_k} &= \sum_{k=1}^{T}(2^{\ell_k})^{2-2\alpha} \\ &\le \left(\sum_{k=1}^{T}1^{1/(2\alpha-1)}\right)^{2\alpha-1}\left(\sum_{k=1}^{T}\left(\left(2^{\ell_k}\right)^{2-2\alpha}\right)^{1/(2-2\alpha)}\right)^{2-2\alpha} \qquad \text{(by Hölder's Inequality)}\end{aligned}$$

$$= \widetilde{O}\left(T^{2\alpha-1}(T\log T)^{2-2\alpha}\right)$$
$$= \widetilde{O}(\sqrt{m}\log M\log T),$$

where we use $T = O(\sqrt{m}\log n\log(nM))$ from Theorem 4. So the runtime for IPM is $\widetilde{O}(m^{1/2+\alpha}\log M)$ Combined with Lemma 66, the overall runtime is $\widetilde{O}(m^{1/2+\alpha}\log M + s(m))$. □

References

[1] Deeksha Adil, Brian Bullins, Rasmus Kyng, and Sushant Sachdeva. 2021. Almost-linear-time weighted ℓ_p-norm solvers in slightly dense graphs via sparsification. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming.* Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 9:1–9:15.

[2] Deeksha Adil, Rasmus Kyng, Richard Peng, and Sushant Sachdeva. 2019. Iterative refinement for p-norm regression. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms.* Timothy M. Chan (Ed.), SIAM, 1405–1424. DOI: https://doi.org/10.1137/1.9781611975482.86

[3] Deeksha Adil and Sushant Sachdeva. 2020. Faster p-norm minimizing flows, via smoothed q-norm problems. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms.* SIAM, 892–910.

[4] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. 1988. *Network Flows.* Prentice Hall.

[5] Mudabir Kabir Asathulla, Sanjeev Khanna, Nathaniel Lahn, and Sharath Raghvendra. 2018. A faster algorithm for minimum-cost bipartite perfect matching in planar graphs. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms.* SIAM, 457–476.

[6] Kyriakos Axiotis, Aleksander Mądry, and Adrian Vladu. 2020. Circulation control for faster minimum cost flow in unit-capacity graphs. In *Proceedings of the 2020 IEEE 61st Annual Symposium on Foundations of Computer Science.* IEEE, 93–104.

[7] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. 2021. Deterministic decremental SSSP and approximate min-cost flow in almost-linear time. In *Proceedings of the 62st IEEE Annual Symposium on Foundations of Computer Science.* IEEE.

[8] Glencora Borradaile. 2008. *Exploiting Planarity for Network Flow and Connectivity Problems.* Brown University.

[9] Glencora Borradaile and Philip N. Klein. 2009. An $O(n\log n)$ algorithm for maximum st-flow in a directed planar graph. *Journal of the ACM* 56, 2 (2009), 9:1–9:30.

[10] Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. 2017. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. *SIAM Journal on Computing* 46, 4 (2017), 1280–1303.

[11] Erin W. Chambers, Jeff Erickson, and Amir Nayyeri. 2012. Homology flows, cohomology cuts. *SIAM Journal on Computing* 41, 6 (2012), 1605–1634.

[12] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. 2022. Maximum flow and minimum-cost flow in almost-linear time. In *Proceedings of the 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science.* IEEE, 612–623.

[13] Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. 2011. 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.* 273–282.

[14] Michael B. Cohen, Yin Tat Lee, and Zhao Song. 2021. Solving linear programs in the current matrix multiplication time. *Journal of the ACM* 68, 1 (2021), 1–39.

[15] Michael B. Cohen, Aleksander Mądry, Piotr Sankowski, and Adrian Vladu. 2017. Negative-weight shortest paths and unit capacity minimum cost flow in $\widetilde{O}(m^{10/7}\log W)$ time. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms.* SIAM, 752–771.

[16] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms.* MIT Press.

[17] Samuel I. Daitch and Daniel A. Spielman. 2008. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing.* 451–460.

[18] Sally Dong. 2024. *Convex Optimization with Combinatorial Characteristics: New Algorithms for Linear Programming, Min-cost Flow, and Other Structured Problems.* Ph.D. Dissertation. University of Washington.

[19] Sally Dong, Gramoz Goranci, Lawrence Li, Sushant Sachdeva, and Guanghao Ye. 2024. Fast algorithms for separable linear programs. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms.* SIAM, 3558–3604.

[20] Sally Dong, Yin Tat Lee, and Guanghao Ye. 2021. 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'21).* ACM, 1784–1797.

[21] Sally Dong, Yin Tat Lee, and Guanghao Ye. 2021. A nearly-linear time algorithm for linear programs with small treewidth: A multiscale representation of robust central path. arXiv:2011.05365v2. Retrieved from https://arxiv.org/abs/2011.05365v2

[22] David Durfee, Rasmus Kyng, John Peebles, Anup B. Rao, and Sushant Sachdeva. 2017. Sampling random spanning trees faster than matrix multiplication. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. 730–742.

[23] Jittat Fakcharoenphol and Satish Rao. 2006. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences* 72, 5 (2006), 868–889.

[24] Lester R. Ford and Delbert R. Fulkerson. 1956. Maximal flow through a network. *Canadian Journal of Mathematics* 8 (1956), 399–404.

[25] Yu Gao, Yang P. Liu, and Richard Peng. 2021. Fully dynamic electrical flows: Sparse maxflow faster than goldberg-rao. In *Proceedings of the 62st IEEE Annual Symposium on Foundations of Computer Science*. IEEE.

[26] Mehrdad Ghadiri, Richard Peng, and Santosh S. Vempala. 2023. The bit complexity of efficient continuous optimization. In *Proceedings of the 2023 IEEE 64th Annual Symposium on Foundations of Computer Science*. IEEE, 2059–2070.

[27] J. R. Gilbert and R. E. Tarjan. 1987. The analysis of a nested dissection algorithm. *Numerische Mathematik* 50, 4 (1987), 377–404. DOI: https://doi.org/10.1007/BF01396660

[28] Gramoz Goranci, Monika Henzinger, and Pan Peng. 2018. Dynamic effective resistances and approximate schur complement on separable graphs. In *Proceedings of the 26th Annual European Symposium on Algorithms (LIPIcs'18)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 40:1–40:15.

[29] Keith D. Gremban. 1996. *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*. Ph.D. Dissertation. Carnegie Mellon University.

[30] Refael Hassin. 1981. Maximum flow in (s, t) planar networks. *Information Processing Letters* 13, 3 (1981), 107.

[31] Refael Hassin and Donald B. Johnson. 1985. An $O(n \log^2 n)$ algorithm for maximum flow in undirected planar networks. *SIAM Journal on Computing* 14, 3 (1985), 612–624.

[32] Monika R. Henzinger, Philip Klein, Satish Rao, and Sairam Subramanian. 1997. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences* 55, 1 (1997), 3–23.

[33] Baihe Huang, Shunhua Jiang, Zhao Song, and Runzhou Tao. 2021. Solving tall dense SDPs in the current matrix multiplication time. arXiv:2101.08208. Retrieved from https://arxiv.org/abs/2101.08208

[34] Hiroshi Imai and Kazuo Iwano. 1990. Efficient sequential and parallel algorithms for planar minimum cost flow. In *Proceedings of the Algorithms, International Symposium SIGAL'90 (Lecture Notes in Computer Science)*. Springer, 21–30.

[35] Alon Itai and Yossi Shiloach. 1979. Maximum flow in planar networks. *SIAM Journal on Computing* 8, 2 (1979), 135–150.

[36] Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. 2011. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing*. ACM, 313–322.

[37] Donggu Kang and James Payor. 2015. Flow rounding. arXiv:1507.08139. Retrieved from https://arxiv.org/abs/1507.08139

[38] Haim Kaplan and Yahav Nussbaum. 2013. Min-cost flow duality in planar networks. arXiv:1306.6728. Retrieved from https://arxiv.org/abs/1306.6728

[39] Adam Karczmarz and Piotr Sankowski. 2019. Min-cost flow in unit-capacity planar graphs. In *Proceedings of the 27th Annual European Symposium on Algorithms (LIPIcs'19)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 66:1–66:17.

[40] Tarun Kathuria, Yang P. Liu, and Aaron Sidford. 2020. Unit capacity maxflow in almost $O(m^{4/3})$ time. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science*. 119–130. DOI: https://doi.org/10.1109/FOCS46700.2020.00020

[41] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. 2014. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 217–226.

[42] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. 2013. A simple, combinatorial algorithm for solving SDD systems in nearly-linear time. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*. 911–920.

[43] Samir Khuller, Joseph Naor, and Philip Klein. 1993. The lattice structure of flow in planar graphs. *SIAM Journal on Discrete Mathematics* 6, 3 (1993), 477–490.

[44] Valerie King, Satish Rao, and Rorbert Tarjan. 1994. A faster deterministic maximum flow algorithm. *Journal of Algorithms* 17, 3 (1994), 447–474.

[45] Rasmus Kyng. 2017. *Approximate Gaussian Elimination*. Ph.D. Dissertation. Yale University.

[46] Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman. 2016. Sparsified Cholesky and multigrid solvers for connection laplacians. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 842–850.

[47] Rasmus Kyng, Richard Peng, Sushant Sachdeva, and Di Wang. 2019. Flows in almost linear time via adaptive preconditioning. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. 902–913.

[48] Rasmus Kyng and Sushant Sachdeva. 2016. Approximate gaussian elimination for Laplacians—fast, sparse, and simple. In *Proceedings of the 2016 IEEE 57th Annual Symposium on Foundations of Computer Science*. IEEE, 573–582.

[49] Nathaniel Lahn and Sharath Raghvendra. 2019. A faster algorithm for minimum-cost bipartite matching in minor-free graphs. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 569–588.

[50] R. J. Lipton and Robert Tarjan. 1979. A planar separator theorem. *SIAM Journal of Applied Mathematics* 36, 2 (1979), 177–189.

[51] Richard J. Lipton, Donald J. Rose, and Robert Endre Tarjan. 1979. Generalized nested dissection. *SIAM Journal on Numerical Analysis* 16, 2 (1979), 346–358.

[52] Aleksander Madry. 2013. Navigating central path with electrical flows: From flows to matchings, and back. In *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. IEEE, 253–262.

[53] Aleksander Madry. 2016. Computing maximum flow with augmenting electrical flows. In *Proceedings of the 2016 IEEE 57th Annual Symposium on Foundations of Computer Science*. IEEE, 593–602.

[54] Gary L. Miller and Joseph Naor. 1995. Flow in planar graphs with multiple sources and sinks. *SIAM Journal on Computing* 24, 5 (1995), 1002–1017.

[55] Gary L. Miller and Richard Peng. 2013. Approximate maximum flow on separable undirected graphs. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1151–1170.

[56] James Orlin. 1988. A faster strongly polynomial minimum cost flow algorithm. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*. 377–387.

[57] John H. Reif. 1983. Minimum s-t cut of a planar undirected network in $O(n \log^2 n)$ time. *SIAM Journal on Computing* 12, 1 (1983), 71–81.

[58] Jonah Sherman. 2013. Nearly maximum flows in nearly linear time. In *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. IEEE, 263–269.

[59] Jonah Sherman. 2017. Area-convexity, linf regularization, and undirected multicommodity flow. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. 452–460.

[60] Aaron Sidford and Kevin Tian. 2018. Coordinate methods for accelerating linf regression and faster approximate maximum flow. In *Proceedings of the 2018 IEEE 59th Annual Symposium on Foundations of Computer Science*. IEEE, 922–933.

[61] Daniel A. Spielman and Shang-Hua Teng. 2004. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*. 81–90.

[62] Robert E. Tarjan. 1971. *An Efficient Planarity Algorithm*. Technical Report.

[63] Balachandran Vaidyanathan and Ravindra K. Ahuja. 2010. Fast algorithms for specially structured minimum cost flow problems with applications. *Operations Research* 58, 6 (2010), 1681–1696.

[64] Jan van den Brand. 2020. A deterministic linear program solver in current matrix multiplication time. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 259–278.

[65] Jan van den Brand. 2021. Unifying matrix data structures: Simplifying and speeding up iterative algorithms. In *Proceedings of the Symposium on Simplicity in Algorithms*. SIAM, 1–13.

[66] Jan van den Brand, Yu Gao, Arun Jambulapati, Yin Tat Lee, Yang P. Liu, Richard Peng, and Aaron Sidford. 2022. Faster maxflow via improved dynamic spectral vertex sparsifiers. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. 543–556.

[67] Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. 2021. Minimum cost flows, MDPs, and ℓ1-regression in nearly linear time for dense instances. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. 859–869.

[68] Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. 2020. Solving tall dense linear programs in nearly linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. 775–788.

[69] Nisheeth K. Vishnoi. 2013. $Lx = b$ Laplacian solvers and their algorithmic applications. *Foundations and Trends in Theoretical Computer Science* 8, 1–2 (2013), 1–141.

[70] Karsten Weihe. 1997. Maximum (s, t)-flows in planar networks in $O(|V| \log |V|)$ time. *Journal of Computer and System Sciences* 55, 3 (1997), 454–475.

Appendices

A Inexact Laplacian Solver

As part of computing the overall runtime of our algorithm, we bound the runtime of the SDD-solver from Theorem 3; 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

$$\mathbf{W}_{ii} = ((\boldsymbol{u}_i - \overline{\boldsymbol{f}}_i)^{-2} + (\overline{\boldsymbol{f}}_i - \boldsymbol{l}_i)^{-2})^{-1} \geq d_i^2/2, \tag{A.1}$$

where d_i is the distance from $\overline{\boldsymbol{f}}_i$ to the boundary, i.e., $\boldsymbol{l}_i$ and $\boldsymbol{u}_i$. In Lemma 69, we prove that d_i is polynomially bounded in terms of the polytope parameters L, R, and the central path parameter 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.

Lemma 69. *Given $(\boldsymbol{x}, \boldsymbol{s})$ in the robust central path region of the LP with central path parameter t, let η be the distance from $\boldsymbol{x}$ to the boundary of the feasible region. Then η is polynomially bounded from below as a function of t.*

Proof. We roughly follow the proof of [21, Theorem A.18] using the same notation. Consider the function $f(\boldsymbol{x}) = \boldsymbol{c}^\top \boldsymbol{x} + t\phi(\boldsymbol{x})$ over the domain $\mathcal{P} \stackrel{\text{def}}{=} \{\boldsymbol{x} : \mathbf{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq 0\}$, where ϕ is ν-self-concordant. We define the corresponding dual set by $\mathcal{D} \stackrel{\text{def}}{=} \{\boldsymbol{y} : \mathbf{A}^\top \boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c}\}$.

Given $(\boldsymbol{x}, \boldsymbol{s})$ with central path parameter t, we note that $\boldsymbol{x}$ must be optimal point of the function $g(\boldsymbol{x}) \stackrel{\text{def}}{=} \tilde{\boldsymbol{c}}^\top \boldsymbol{x} + t \cdot \phi(\boldsymbol{x})$ over $\mathcal{P}$, where $\tilde{\boldsymbol{c}} = \boldsymbol{c} - \boldsymbol{s} - t \cdot \nabla\phi(\boldsymbol{x})$ Indeed, by definition, we have

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

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.

Recall that as part of the initial set up, $\mathcal{P}$ contains a ball of radius r around some point $\boldsymbol{z}$ Define the line segment between $\boldsymbol{x}$ and $\boldsymbol{z}$:

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

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

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

Define $\alpha = \boldsymbol{s}/t + \nabla\phi(\boldsymbol{x})$. Then, we have $\tilde{\boldsymbol{c}} = \boldsymbol{c} - t \cdot \alpha$. By assumption of RIPM, we have

$$\|\alpha_i\|^*_{\boldsymbol{x}_i} \leq 1/16 \text{ for all } i. \tag{A.2}$$

Then,

$$\begin{aligned} &(\tilde{\boldsymbol{c}} + t\nabla\phi(\boldsymbol{x}))^\top (\boldsymbol{z} - \boldsymbol{x}) \\ &= (\boldsymbol{c} - t\alpha + t\nabla\phi(\boldsymbol{x}))^\top (\boldsymbol{z} - \boldsymbol{x}) \end{aligned}$$

$$\leq 2LR + \tfrac{t}{16}\sum_i \|\boldsymbol{z}_i - \boldsymbol{x}_i\|_{\boldsymbol{x}_i} + t\nabla\phi(\boldsymbol{x})^\top(z - \boldsymbol{x}), \tag{A.3}$$

where the last inequality follows from the $\|\boldsymbol{c}\|_2 \leq L$, $\|\boldsymbol{x} - z\|_2 \leq 2R$, and Equation (A.2).

To bound the last two terms, let us first fix $i \in [m]$, and define $\widetilde{\phi}$ to beϕ_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 the 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 $\alpha, \boldsymbol{x}_i, \boldsymbol{z}_i, \beta$ on the line p, and we view them in 1 dimension, denoted, respectively, by α, x, z, β. Then, we have

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

For most $i \in [m]$, we simply use the bound $u_i \leq 4\nu_i^2$.

For each $i \in m$, let η_i be the distance from $\boldsymbol{x}_i$ to the boundary of $[\boldsymbol{\ell}_i, \boldsymbol{u}_i]$. Since η is the distance from $\boldsymbol{x}$ to the boundary, we must have $\eta = \eta_i$ for some i. For this fixed dimension, we use a tighter bound that includes the second term in the above expression. Let $\boldsymbol{q} \overset{\text{def}}{=} \arg\min_{\boldsymbol{v} \in \partial K_i} \|\boldsymbol{v} - \boldsymbol{x}_i\|_2$. 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, we know $\boldsymbol{z}' \notin K_i$. Finally, an argument of similar triangles shows

$$\frac{\|\boldsymbol{q} - \boldsymbol{x}_i\|_2}{\|\boldsymbol{x} - \boldsymbol{\alpha}\|_2} = \frac{\|\boldsymbol{z}' - \boldsymbol{z}_i\|_2}{\|\boldsymbol{z} - \boldsymbol{\alpha}\|_2}.$$

Hence,

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

This gives us the tighter bound $u_i \leq 4\nu_i^2 - \frac{r}{64\eta}$ for the i with $\eta_i = \eta$.

Putting it back to Equation (A.3), we have

$$\begin{aligned} (\tilde{c} + t\nabla\phi(\boldsymbol{x}))^\top(\boldsymbol{z} - \boldsymbol{x}) &\leq 2LR + \tfrac{t}{16}\sum_i \|\boldsymbol{z}_i - \boldsymbol{x}_i\|_{\boldsymbol{x}_i} + t\nabla\phi(\boldsymbol{x})^\top(\boldsymbol{z} - \boldsymbol{x}) \\ &\leq 2LR + t\sum_i u_i \\ &\leq 2LR + 4t\nu^2 - \frac{rt}{16\eta}. \end{aligned}$$

Since $2LR + 4t\nu^2 - \frac{rt}{16\eta} \geq 0$, we have $\eta \geq \frac{rt}{64LR + 64t\nu^2}$. □

The RIPM additionally guarantees that the approximation $\overline{\boldsymbol{x}}$ satisfies $\|\mathbf{W}^{-1/2}(\boldsymbol{x} - \overline{\boldsymbol{x}})\|_\infty \leq O(1/\log m)$. Hence, we also have

Corollary 70. *Let $\overline{\boldsymbol{x}}$ be the approximation of $\boldsymbol{x}$ guaranteed by the RIPM. Then the distance from $\overline{\boldsymbol{x}}$ to the boundary of the feasible region is also bounded from below with polynomial dependence on t.*

B Deferred Graph Lemmas

Lemma 27. *Suppose $\mathcal{T}$ is the appropriate separator tree for the IPM input graph. Let $\mathcal{H}$ be a set of K nodes in $\mathcal{T}$. Then*

$$\sum_{H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})} |\partial H| + |F_H| \leq \widetilde{O}\left(\sqrt{mK}\right).$$

Proof. Note that F_H is always a subset of $S(H)$. We will instead prove

$$\sum_{H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})} |\partial H| + |S(H)| \leq \widetilde{O}(\sqrt{mK}).$$

First, we decompose the quantity we want to bound by levels in $\mathcal{T}$:

$$\sum_{H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})} |\partial H| + |S(H)| = \sum_{i=0}^{\eta} \sum_{H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H}, i)} |\partial H| + |S(H)|. \tag{B.1}$$

We first bound $\sum_{H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H}, i)} |\partial H| + |S(H)|$ for a fixed i. Our main observation is that we can bound the total number of boundary vertices of nodes at level i by the number of boundary and separator vertices of nodes at level $(i+1)$ Formally, our key claim is the following:

$$\sum_{H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H}, i)} |\partial H| \leq \sum_{H' \in \mathcal{P}_{\mathcal{T}}(\mathcal{H}, i+1)} (|\partial H'| + 2|S(H')|)\,. \tag{B.2}$$

Without loss of generality, we may assume that if node H is included in the left hand sum, then its sibling is included as well. Next, recall by the definition of $\mathcal{T}$, for siblings H_1, H_2 with parent H' their boundaries are defined as

$$\partial H_i = (S(H') \cup \partial H') \cap V(H_i) = (S(H') \cap V(H_i)) \cup ((\partial H' \setminus S(H')) \cap V(H_i)),$$

for $i = 1, 2$. Furthermore, $V(H_1) \cup V(H_2) = V(H)$. Another crucial observation is that a vertex from $\partial H'$ exists in both H_1 and H_2 if and only if that vertex belongs to the separator $S(H')$.

$$\begin{aligned} |\partial H_1| + |\partial H_2| &\leq |S(H')| + |(\partial H' \setminus S(H')) \cap V(H_1)| + |S(H')| + |(\partial H' \setminus S(H')) \cap V(H_2')| \\ &\leq |\partial H'| + 2|S(H')|. \end{aligned} \tag{B.3}$$

By summing Equation (B.3) over all pairs of siblings in $\mathcal{P}_{\mathcal{T}}(\mathcal{H}, i)$, we get Equation (B.2). By repeatedly applying Equation (B.2) until we reach the root at height η, we have

$$\sum_{H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H}, i)} |\partial H| \leq 2 \sum_{j=i+1}^{\eta} \sum_{H' \in \mathcal{P}_{\mathcal{T}}(\mathcal{H}, j)} |S(H')|. \tag{B.4}$$

Summing over all the levels in $\mathcal{T}$, we have

$$\begin{aligned} \sum_{i=0}^{\eta} \sum_{H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H}, i)} (|\partial H| + |S(H)|) &\leq 2 \sum_{j=0}^{\eta} (j+1) \sum_{H' \in \mathcal{P}_{\mathcal{T}}(\mathcal{H}, j)} |S(H')| && \text{(by Equation (B.4))} \\ &\leq 2c \sum_{j=0}^{\eta} (j+1) \sum_{H' \in \mathcal{P}_{\mathcal{T}}(\mathcal{H}, j)} \sqrt{|E(H')|}, \end{aligned} \tag{B.5}$$

where c is the constant such that $|S(H')| \leq c(|E(H')|)^{1/2}$ in the definition of being 1/2-separable. Furthermore, the set of ancestors of $\mathcal{H}$ at level j has size $|\mathcal{P}_{\mathcal{T}}(\mathcal{H}, j)| \leq |\mathcal{H}| = K$. Applying the

Cauchy-Schwarz inequality, we get that

$$\begin{aligned}\sum_{H\in\mathcal{P}_{\mathcal{T}}(\mathcal{H})}(|\partial H|+|S(H)|) &\leq 2c\sum_{j=0}^{\eta}(j+1)\sqrt{|\mathcal{P}_{\mathcal{T}}(\mathcal{H},j)|}\cdot\left(\sum_{H'\in\mathcal{P}_{\mathcal{T}}(\mathcal{H},j)}|E(H')|\right)^{1/2}\\ &\leq 2c\sum_{j=0}^{\eta}(j+1)\sqrt{K}\cdot\left(\sum_{H'\in\mathcal{P}_{\mathcal{T}}(\mathcal{H},j)}|E(H')|\right)^{1/2}\\ &\leq 2c\eta\sqrt{K}\sum_{j=0}^{\eta}\left(\sum_{H'\in\mathcal{P}_{\mathcal{T}}(\mathcal{H},j)}|E(H')|\right)^{1/2}\\ &\leq O(\eta^2\sqrt{mK}),\end{aligned}$$

where the final inequality follows from the fact that nodes at the same level form an edge partition of G. As $\eta = O(\log m)$, the lemma follows. □

Lemma 66. *Let C be an α-separable class such that we can compute a balanced separator for any connected graph in C with n vertices and m edges in $s(m)$ time for some convex function $s(m) \geq m$. Given an α-separable graph $G \in C$, there is an algorithm that computes a separator tree $\mathcal{T}$ for G in $O(s(m)\log m)$ time.*

Proof. First, we let G be the root node of $\mathcal{T}(G)$. Let G_1 and G_2 be the two disjoint components of G obtained after the removal of the vertices in $S(G)$. We define the children $c_1(G), c_2(G)$ of G as follows: $V(c_i(G)) = V(G_i)\cup S(G)$ and $E(c_i(G)) = E(G_i)$, for $i = 1, 2$. Edges connecting some vertex in G_i and another vertex in $S(G)$ are added to $E(c_i(G))$. For each edge connecting two vertices in $S(G)$, we append it to $E(c_1(G))$ or $E(c_2(G))$, whichever has less edges. We continue by repeatedly splitting each child $c_i(G)$ that has at least one edge in the same way as we did for G, whenever possible. There are $O(m)$ components, each containing exactly 1 edge. The components containing exactly 1 edge form the *leaf nodes* of $\mathcal{T}(G)$. Note that the height of $\mathcal{T}(G)$ is bounded by $O(\log m) = O(\log m)$ as for any child H' of a node H, $|E(H')| \leq b|E(H)|$.

The runtime of the algorithm is bounded by the total time to construct the separator for all nodes in the tree. Because the tree has height $O(\log m)$ and nodes with the same depth does not share any edge, the sum of edges over all tree nodes is $O(m\log m)$. Since $s(m)$ is convex, the algorithm runs in no more than $O(s(m)\log m)$ time. □

Lemma 67. *Let G be an α-separable graph with separator tree $\mathcal{T}$ constructed via Lemma 66. Let $\mathcal{H}$ be a set of K nodes in $\mathcal{T}$. Then*

$$\sum_{H\in\mathcal{P}_{\mathcal{T}}(\mathcal{H})}|\partial H|+|S(H)| \leq \widetilde{O}(K^{1-\alpha}m^{\alpha}).$$

Proof. Using the separator tree, we have Equation (B.5) in exactly the same way as for the planar case.

$$\sum_{H\in\mathcal{P}_{\mathcal{T}}(\mathcal{H})}(|\partial H|+|S(H)|) \leq 2c\sum_{j=0}^{\eta}(j+1)\sum_{H'\in\mathcal{P}_{\mathcal{T}}(\mathcal{H},j)}\sqrt{|E(H')|}$$

Applying Hölder's Inequality instead of Cauchy–Schwarz for the planar case, we get

$$
\begin{aligned}
&\leq 2c\sum_{j=0}^{\eta}(j+1)|\mathcal{P}_{\mathcal{T}}(\mathcal{H},j)|^{1-\alpha}\cdot\left(\sum_{H'\in\mathcal{P}_{\mathcal{T}}(\mathcal{H},j)}|E(H')|\right)^{\alpha}\\
&\leq 2c\sum_{j=0}^{\eta}(j+1)K^{1-\alpha}\cdot\left(\sum_{H'\in\mathcal{P}_{\mathcal{T}}(\mathcal{H},j)}|E(H')|\right)^{\alpha}\\
&\leq 2c\eta K^{1-\alpha}\sum_{j=0}^{\eta}\left(\sum_{H'\in\mathcal{P}_{\mathcal{T}}(\mathcal{H},j)}|E(H')|\right)^{\alpha}\\
&\leq O(\eta^2K^{1-\alpha}m^{\alpha}),
\end{aligned}
$$

where the final inequality follows from the fact that nodes at the same level form an edge partition of G. As $\eta = O(\log m)$, the lemma follows. □

C Interpretation of the Flow Operator

Suppose we have an exact SDD-solver, i.e., given an SDD matrix $\mathbf{L}$ and $\boldsymbol{d}$, we can compute $\boldsymbol{x}$ such that $\mathbf{L}\boldsymbol{x} = \boldsymbol{d}$. Then this section provides an intuitive interpretation of how the flow update is computed at every step via electrical flows.

Recall the definition of the slack projection tree operator in Definition 60. On an edge from node H to parent P, we have the edge operator

$$
\mathbf{M}_{(H,P)} \overset{\text{def}}{=} \mathbf{I}_{\partial H\cup F_H} - \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1}\mathbf{L}^{(H)}_{F_H,\partial H} = \begin{bmatrix}\mathbf{I}_{F_H} & -\left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1}\mathbf{L}^{(H)}_{F_H,\partial H}\\ \mathbf{0} & \mathbf{I}_{\partial H}\end{bmatrix}.
$$

As shown in [66, Lemma 5.2], this linear operator satisfies

$$
\begin{bmatrix}\mathbf{I}_{F_H} & -\left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1}\mathbf{L}^{(H)}_{F_H,\partial H}\\ \mathbf{0} & \mathbf{I}_{\partial H}\end{bmatrix}\begin{bmatrix}\mathbf{0}\\ \mathbf{I}_{\partial H} - \mathbf{1}\mathbf{1}^\top/|\partial H|\end{bmatrix} = (\mathbf{L}^{(H)})^{\dagger}\mathbf{Sc}(\mathbf{L}^{(H)},\partial H).
$$

We only apply $\mathbf{M}_{(H,P)}$ to vectors that are zero on F_H, and are orthogonal to $\mathbf{1}$ on ∂H; hence, over this domain,

$$
\mathbf{M}_{(H,P)} = (\mathbf{L}^{(H)})^{\dagger}\mathbf{Sc}(\mathbf{L}^{(H)},\partial H).
$$

We use this version of the edge operator, with the additional modification of using approximate Schur complements instead of exact. The remainder of the section proves the following theorem of correctness:

Theorem 71. *Let $\boldsymbol{v}\in\mathbb{R}^m$, and $\boldsymbol{z}\overset{\text{def}}{=}\widetilde{\mathbf{\Gamma}}\mathbf{\Pi}^{(\eta-1)}\cdots\mathbf{\Pi}^{(0)}\mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}$. Let $\mathbf{M}$ be the flow projection tree operator as defined above, and let $\epsilon_{\mathbf{P}}$ be the overall target step accuracy from* DynamicSC. *Then $\tilde{\boldsymbol{f}}\overset{\text{def}}{=}\mathbf{M}\boldsymbol{z}$ satisfies $\mathbf{B}^\top\mathbf{W}^{1/2}\tilde{\boldsymbol{f}} = \mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}$ and $\left\|\tilde{\boldsymbol{f}} - \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}\right\|_2 \leq O(\eta\epsilon_{\mathbf{P}})\,\|\boldsymbol{v}\|_2$.*

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 $(\mathbf{W}^{1/2}\mathbf{B})^\top\boldsymbol{f} = \boldsymbol{d}$.

Next, we introduce some definitions and properties of electrical flow, which we use in later to prove Theorem 71. 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 72. 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 $\mathbf{L} \overset{\text{def}}{=} \mathbf{B}^\top\mathbf{W}\mathbf{B}$ be the Laplacian. Let $\boldsymbol{d} \overset{\text{def}}{=} \mathbf{L}\boldsymbol{z}$ 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 $\boldsymbol{f}^\star = \mathbf{W}^{1/2}\mathbf{B}\mathbf{L}^\dagger\boldsymbol{d}$. Hence, we can refer to its energy as the energy of the demand $\boldsymbol{d}$ on the graph of $\mathbf{L}$, given by

$$\mathcal{E}_{\mathbf{L}}(\boldsymbol{d}) \overset{\text{def}}{=} \min_{(\mathbf{W}^{1/2}\mathbf{B})^\top \boldsymbol{f}=\boldsymbol{d}} \|\boldsymbol{f}\|_2^2 = \boldsymbol{d}^\top(\mathbf{B}^\top\mathbf{W}\mathbf{B})^\dagger\boldsymbol{d} = \boldsymbol{d}^\top\mathbf{L}^\dagger\boldsymbol{d} = \boldsymbol{z}^\top\mathbf{L}\boldsymbol{z}. \tag{C.1}$$

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 73. *We continue using the notation from Definition 72. For any flow $\tilde{\boldsymbol{f}}$ routing $\boldsymbol{d}$ on G, if $\|\tilde{\boldsymbol{f}}\|_2^2 \leq_\varepsilon \mathcal{E}_{\mathbf{L}}(\boldsymbol{d})$, then $\|\tilde{\boldsymbol{f}} - \boldsymbol{f}^\star\|_2^2 \leq O(\varepsilon)\left\|\boldsymbol{f}^\star\right\|_2^2$.*

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

$$\boldsymbol{f}^{\star\top}(\tilde{\boldsymbol{f}} - \boldsymbol{f}^\star) = \boldsymbol{d}^\top\mathbf{L}^\dagger\mathbf{B}^\top\mathbf{W}^{1/2}(\tilde{\boldsymbol{f}} - \boldsymbol{f}^\star) = \boldsymbol{d}^\top\mathbf{L}^\dagger(\boldsymbol{d} - \boldsymbol{d}) = \mathbf{0}.$$

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

$$\|\tilde{\boldsymbol{f}} - \boldsymbol{f}^\star\|^2 \leq (e^\varepsilon - 1)\cdot\left\|\boldsymbol{f}^\star\right\|_2^2 = O(\varepsilon)\cdot\left\|\boldsymbol{f}^\star\right\|_2^2. \qquad \square$$

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 74. *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 $\boldsymbol{d} \overset{\text{def}}{=} \mathbf{L}\boldsymbol{z}$ 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,

$$\begin{aligned}\mathcal{E}_{\mathbf{L}^{(1)}}\left(\boldsymbol{d}^{(1)}\right) + \mathcal{E}_{\mathbf{L}^{(2)}}\left(\boldsymbol{d}^{(2)}\right) &= \boldsymbol{z}^\top\mathbf{L}^{(1)}\boldsymbol{z} + \boldsymbol{z}^\top\mathbf{L}^{(2)}\boldsymbol{z} \\ &= \boldsymbol{z}^\top\mathbf{L}\boldsymbol{z} \\ &= \mathcal{E}_{\mathbf{L}}(\boldsymbol{d}). \qquad \square\end{aligned}$$

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 75. *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,*

$$\mathcal{E}_{\mathbf{L}}(\boldsymbol{d}) \approx_\varepsilon \mathcal{E}_{\mathbf{L}'}(\boldsymbol{d}).$$

Proof. We have, by definition,

$$\mathcal{E}_{\mathbf{L}}(\boldsymbol{d}) = \boldsymbol{d}^\top\mathbf{L}^\dagger\boldsymbol{d} = \boldsymbol{d}^\top\mathrm{Sc}(\mathbf{L}, C)^\dagger\boldsymbol{d} \approx_\varepsilon \boldsymbol{d}^\top\mathbf{L}'^\dagger\boldsymbol{d} = \mathcal{E}_{\mathbf{L}'}(\boldsymbol{d}),$$

where the second equality follows from the fact that $\boldsymbol{d}$ is supported on C combined with the formula for $\mathbf{L}^\dagger$ in Equation (3.4). Note that we may omit the projection $\mathbf{P}_{\mathbf{L}}$ onto $\mathbf{L}$'s kernel in Equation (3.4), as $\boldsymbol{d}$ is a demand and therefore already in the kernel. $\square$

We are now ready to prove Theorem 71.

Let G denote the input graph with weights $\mathbf{W}$ and Laplacian $\mathbf{L}$. Let $\boldsymbol{d} \stackrel{\text{def}}{=} \mathbf{B}^\top \mathbf{W}^{1/2}\boldsymbol{v}$ be the demand vector Let $\boldsymbol{f}^\star \stackrel{\text{def}}{=} \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v} = \mathbf{W}^{1/2}\mathbf{B}\mathbf{L}^\dagger \boldsymbol{d}$, that is, $\boldsymbol{f}^\star$ is the electrical flow routing $\boldsymbol{d}$. In the first part of the proof, we show that $\tilde{\boldsymbol{f}}$ routes the demand $\boldsymbol{d}$ (Lemma 79). In the second part of the proof, we show that $\tilde{\boldsymbol{f}}$ is close to $\boldsymbol{f}^\star$.

We note the following fact: if $\mathbf{L}$ and $\mathbf{L}'$ are two Laplacians, then $\mathbf{L}\mathbf{L}^\dagger\mathbf{L}' = \mathbf{L}'$, since the range of $\mathbf{L}'$ is orthogonal to the kernel of $\mathbf{L}$. In our proofs, we will often apply this to products of Schur complements (which are themselves Laplacians).

Lemma 76. *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 71. 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 $\boldsymbol{d}^{(H)} \stackrel{\text{def}}{=} \mathbf{L}^{(H)}\boldsymbol{z}|_{F_H}$. 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

$$\mathbf{X}^{(i)} \stackrel{\text{def}}{=} \sum_{H\in\mathcal{T}(i)} \mathbf{X}^{(H)} \text{ where } \mathbf{X}^{(H)} \stackrel{\text{def}}{=} \mathbf{L}^{(H)}_{\partial H, F_H}\left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1},$$

So that $\mathbf{\Pi}^{(i)} = \mathbf{I} - \mathbf{X}^{(i)}$. Suppose H is at level i of $\mathcal{T}$. We have

$$\begin{aligned}\boldsymbol{z}|_{F_H} &= \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1}\mathbf{\Pi}^{(\eta-1)}\cdots\mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d}\\ &= \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1}\mathbf{\Pi}^{(i-1)}\cdots\mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d}, \end{aligned} \tag{C.2}$$

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 Equation (C.2). 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

$$\begin{aligned}\sum_{H\in\mathcal{T}}\boldsymbol{d}^{(H)} &= \sum_{i=0}^{\eta}\sum_{H\in\mathcal{T}(i)}\mathbf{X}^{(H)}\mathbf{\Pi}^{(i-1)}\cdots\mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d} + \sum_{i=0}^{\eta}\sum_{H\in\mathcal{T}(i)}(\mathbf{\Pi}^{(\eta-1)}\cdots\mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d})|_{F_H}\\ &= \left(\sum_{i=0}^{\eta}\mathbf{X}^{(i)}\mathbf{\Pi}^{(i-1)}\cdots\mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d}\right) + \mathbf{\Pi}^{(\eta-1)}\cdots\mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d} && (F_H \text{ partition } V(G))\\ &= \left(\sum_{i=0}^{\eta-1}(\mathbf{I}-\mathbf{\Pi}^{(i)})\mathbf{\Pi}^{(i-1)}\cdots\mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d}\right) + \mathbf{\Pi}^{(\eta-1)}\cdots\mathbf{\Pi}^{(1)}\mathbf{\Pi}^{(0)}\boldsymbol{d}\\ &= \boldsymbol{d}, && \text{(telescoping sum)}\end{aligned}$$

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 77. Let $\mathbf{M}^{(H)}$ be the flow tree operator supported on the tree $\mathcal{T}_H$ (Definition 46). We define the flow $\tilde{\boldsymbol{f}}^{(H)} \stackrel{\text{def}}{=} \mathbf{M}^{(H)}\boldsymbol{z}|_{F_H}$.

Lemma 78. *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

$$\begin{aligned}
\mathbf{B}^\top\mathbf{W}^{1/2}\mathbf{M}^{(H)} &= \mathbf{B}^\top\mathbf{W}^{1/2}\left(\mathbf{M}^{(D_1)}\mathbf{M}_{(D_1,H)} + \mathbf{M}^{(D_2)}\mathbf{M}_{(D_2,H)}\right) \\
&= \mathbf{L}^{(D_1)}\mathbf{M}_{(D_1,H)} + \mathbf{L}^{(D_2)}\mathbf{M}_{(D_2,H)} && \text{(by induction)} \\
&= \mathbf{L}^{(D_1)}(\mathbf{L}^{(D_1)})^\dagger\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)}, \partial D_1) + \mathbf{L}^{(D_2)}(\mathbf{L}^{(D_2)})^\dagger\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)}, \partial D_2) \\
&= \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)}, \partial D_1) + \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)}, \partial D_2) \\
&= \mathbf{L}^{(H)}.
\end{aligned}$$

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 79. *$\tilde{\boldsymbol{f}}$ 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 76. 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 78. 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 80. *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

$$\left\|\mathbf{M}^{(H)}\boldsymbol{z}\right\|_2^2 = \boldsymbol{z}^\top(\mathbf{B}[H])^\top\mathbf{W}\mathbf{B}[H]\boldsymbol{z} = \boldsymbol{z}^\top\mathbf{L}[H]\boldsymbol{z} = \mathcal{E}_{\mathbf{L}[H]}(\boldsymbol{d}).$$

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

$$\begin{aligned}
&= \left\|\mathbf{M}^{(D_1)}\mathbf{M}_{(D_1,H)}\boldsymbol{z}\right\|_2^2 + \left\|\mathbf{M}^{(D_2)}\mathbf{M}_{(D_2,H)}\boldsymbol{z}\right\|_2^2 \\
&\leq_{(i-1)\epsilon_\mathbf{P}} \mathcal{E}_{\mathbf{L}[D_1]}\left(\mathbf{L}^{(D_1)}\mathbf{M}_{(D_1,H)}\boldsymbol{z}\right) + \mathcal{E}_{\mathbf{L}[D_2]}\left(\mathbf{L}^{(D_2)}\mathbf{M}_{(D_2,H)}\boldsymbol{z}\right) \\
&\hspace{20em}\text{(by inductive hypothesis with } \boldsymbol{z} = \mathbf{M}_{(D_i,H)}\boldsymbol{z}) \\
&= \mathcal{E}_{\mathbf{L}[D_1]}\left(\mathbf{L}^{(D_1)}(\mathbf{L}^{(D_1)})^\dagger\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)},\partial D_1)\boldsymbol{z}\right) + \mathcal{E}_{\mathbf{L}[D_2]}\left(\mathbf{L}^{(D_2)}(\mathbf{L}^{(D_2)})^\dagger\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)},\partial D_2)\boldsymbol{z}\right) \\
&\hspace{20em}\text{(substituting the definition of } \mathbf{M}_{(D_i,H)}) \\
&= \mathcal{E}_{\mathbf{L}[D_1]}\left(\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)},\partial D_1)\boldsymbol{z}\right) + \mathcal{E}_{\mathbf{L}[D_2]}\left(\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)},\partial D_2)\boldsymbol{z}\right).
\end{aligned}$$

Since the demand vectors are supported on ∂D_1 and ∂D_2, respectively, we may take *exact* Schur complements and apply Lemma 75 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 6 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 75,

$$\begin{aligned}
&\leq_{\epsilon_\mathbf{P}} \mathcal{E}_{\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)},\partial D_1)}\left(\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)},\partial D_1)\boldsymbol{z}\right) + \mathcal{E}_{\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)},\partial D_2)}\left(\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)},\partial D_2)\boldsymbol{z}\right) \\
&= \mathcal{E}_{\mathbf{L}^{(H)}}(\mathbf{L}^{(H)}\boldsymbol{z}). \qquad\qquad \text{(by Lemma 74)}
\end{aligned}$$

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 81. *We have the following approximation of the energy of routing* $\boldsymbol{d}$*:*

$$\sum_{H\in\mathcal{T}} \mathcal{E}_{\mathbf{L}[H]}\left(\boldsymbol{d}^{(H)}\right) \approx_{(\eta+1)\epsilon_\mathbf{P}} \mathcal{E}_{\mathbf{L}}(\boldsymbol{d}).$$

Proof. We need the following matrix multiplication property: For any matrices $\mathbf{A}, \mathbf{B}, \mathbf{D}$,

$$\begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^\top & \mathbf{D} \end{bmatrix}\begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{C.3}$$

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.

Let $\boldsymbol{\beta} \stackrel{\text{def}}{=} \boldsymbol{\Pi}^{(\eta-1)} \cdots \boldsymbol{\Pi}^{(1)}\boldsymbol{\Pi}^{(0)}\boldsymbol{d}$ for simplicity of notation, so that $\boldsymbol{z}|_{F_H} = (\mathbf{L}^{(H)}_{F_H,F_H})^{-1}\boldsymbol{\beta}$. Then,

$$\begin{aligned}
\mathcal{E}_{\mathbf{L}[H]}\left(\boldsymbol{d}^{(H)}\right) &\approx_{\epsilon_{\mathrm{P}}} \mathcal{E}_{\mathbf{L}^{(H)}}\left(\boldsymbol{d}^{(H)}\right) && \text{(by Lemma 75)}\\
&= \boldsymbol{z}^\top|_{F_H}\mathbf{L}^{(H)}\boldsymbol{z}|_{F_H}\\
&= \boldsymbol{\beta}^\top\left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1}\mathbf{L}^{(H)}\left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1}\boldsymbol{\beta}\\
&= \boldsymbol{\beta}^\top\left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1}\boldsymbol{\beta}. && \text{(by Equation (C.3))}
\end{aligned}$$

Summing over all $H \in \mathcal{T}$, we get

$$\begin{aligned}
\sum_{H\in\mathcal{T}}\mathcal{E}_{\mathbf{L}[H]}\left(\boldsymbol{d}^{(H)}\right) &\approx_{\epsilon_{\mathrm{P}}} \boldsymbol{\beta}^\top\sum_{H\in\mathcal{T}}(\mathbf{L}^{(H)}_{F_H,F_H})^{-1}\boldsymbol{\beta}\\
&= \boldsymbol{d}^\top\boldsymbol{\Pi}^{(0)\top}\cdots\boldsymbol{\Pi}^{(\eta-1)\top}\left[\sum_{H\in\mathcal{T}}(\mathbf{L}^{(H)}_{F_H,F_H})^{-1}\right]\boldsymbol{\Pi}^{(\eta-1)}\cdots\boldsymbol{\Pi}^{(0)}\boldsymbol{d}\\
&\approx_{\eta\epsilon_{\mathrm{P}}} \boldsymbol{d}^\top\mathbf{L}^\dagger\boldsymbol{d}\\
&= \mathcal{E}_{\mathbf{L}}(\boldsymbol{d}).
\end{aligned}$$

□

Finally, we conclude the proof of Theorem 71 by showing $\tilde{\boldsymbol{f}}$ is close to $\boldsymbol{f}^\star \stackrel{\text{def}}{=} \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}$.

Lemma 82. *We have* $\left\|\tilde{\boldsymbol{f}} - \boldsymbol{f}^\star\right\|_2 \leq O(\eta\epsilon_{\mathrm{P}})\left\|\boldsymbol{v}\right\|_2$.

Proof. We know $\tilde{\boldsymbol{f}}$ routes $\boldsymbol{d}$ on G. Its energy is bounded by

$$\begin{aligned}
\left\|\tilde{\boldsymbol{f}}\right\|_2^2 &= \left\|\sum_{H\in\mathcal{T}}\tilde{\boldsymbol{f}}^{(H)}\right\|_2^2\\
&\leq \left(\sum_{i=0}^{\eta}\left\|\sum_{H\in\mathcal{T}(i)}\tilde{\boldsymbol{f}}^{(H)}\right\|_2\right)^2 && \text{(by triangle inequality)}\\
&\leq (\eta+1)\cdot\sum_{i=0}^{\eta}\left\|\sum_{H\in\mathcal{T}(i)}\tilde{\boldsymbol{f}}^{(H)}\right\|_2^2 && \text{(by Cauchy-Schwarz)}\\
&\leq (\eta+1)\cdot\sum_{i=0}^{\eta}\sum_{H\in\mathcal{T}(i)}\left\|\tilde{\boldsymbol{f}}^{(H)}\right\|_2^2 && \text{(by orthogonality, Lemma 48)}\\
&\leq (\eta+1)\cdot\sum_{i=0}^{\eta}\sum_{H\in\mathcal{T}(i)}e^{i\epsilon_{\mathrm{P}}}\cdot\mathcal{E}_{\mathbf{L}[H]}\left(\boldsymbol{d}^{(H)}\right) && \text{(by Lemma 80)}\\
&\approx_{O(\eta\epsilon_{\mathrm{P}})}\mathcal{E}_{\mathbf{L}}\left(\boldsymbol{d}\right). && \text{(by Lemma 81)}
\end{aligned}$$

Now, we apply Lemma 73 to get that

$$\left\|\tilde{\boldsymbol{f}} - \boldsymbol{f}^\star\right\|_2^2 \leq O(\eta\epsilon_{\mathrm{P}})\cdot\left\|\boldsymbol{f}^\star\right\|_2^2 \leq O(\eta\epsilon_{\mathrm{P}})\cdot\left\|\boldsymbol{v}\right\|_2^2,$$

where the last inequality follows from the fact that $\mathbf{P}_{\boldsymbol{w}}$ is an orthogonal projection matrix. □

Received 11 August 2022; revised 15 March 2025; accepted 30 April 2025