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

Sally Dong
University of Washington
sallyqd@uw.edu

Yu Gao
Georgia Tech
ygao380@gatech.edu

Gramoz Goranci †
University of Glasgow
gramoz.goranci@glasgow.ac.uk

Yin Tat Lee ‡
University of Washington
yintat@uw.edu

Richard Peng §
University of Waterloo
y5peng@uwaterloo.ca

Sushant Sachdeva ¶
University of Toronto
sachdeva@cs.toronto.edu

Guanghao Ye‖
Massachusetts Institute of Technology
ghye@mit.edu

Abstract

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 $O(n^{1.5}\text{poly}(\log n))$ 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, JSTOR'79] and approximate Schur complements [Kyng-Sachdeva, FOCS'16]. This implicit representation permits us to design a data structure to route an electrical flow with sparse demands in roughly $\sqrt{n}$ update time, resulting in a total running time of $O(n \cdot \text{poly}(\log n))$.

Our results immediately extend to all families of separable graphs.

*A preliminary version of this work was published at SODA 2022.

†Part of this work was done while the author was at University of Toronto.

‡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.

§Supported by NSF award CCF-1846218 and CCF-2106444. Part of this work was done while the author was at Georgia Tech.

¶Supported by a Discovery grant awarded by NSERC, and an Ontario Early Researcher Award.

‖Supported by an MIT Presidential Fellowship. Part of this work was done while the author was a student at the University of Washington.

Contents

1 Introduction

The minimum cost flow problem on planar graphs is a foundational problem in combinatorial optimization studied since the 1950's. It has diverse applications including network design, VLSI layout, and computer vision. The seminal paper of Ford and Fulkerson in the 1950's [FF56] 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 [Rei83, HJ85], planar graphs by Borradaile-Klein [BK09], and finally bounded genus graphs by Chambers-Erickson-Nayyeri [CEN12]. 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 paper, 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 $\widetilde{O}(n \log M)$ [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: Given a class $\mathcal{C}$ of graphs closed under taking subgraphs, we say it is *α-separable* if there are constants $0 < b < 1$ and $c > 0$ such that every graph in $\mathcal{C}$ with n vertices and m edges has a balanced vertex separator with at most cm^α vertices, and both components obtained after removing the separator have at most bm edges. Then, our algorithm generalizes as follows:

Corollary 2 (Separable min-cost flow)**.** *Let $\mathcal{C}$ be an α-separable graph class such that we can compute a balanced separator for any graph in $\mathcal{C}$ with m edges in $s(m)$ time for some convex function s. Given a graph $G \in \mathcal{C}$ with n vertices and m edges, integer demands $\boldsymbol{d}$, edge capacities $\boldsymbol{u}$ and 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 $\widetilde{O}((m + m^{1/2+\alpha}) \log M + s(m))$ expected time.*

Beyond the study of structured graphs, we believe our paper 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 [Tar71], and using geometric structures to find faster solvers for structured linear systems provided foundations of Laplacian algorithms as well as combinatorial scientific computing [LRT79, Gre96]. Several surprising insights from our nearly-linear time algorithm are:

1. We are able to design a data structure for maintaining a feasible primal-dual (flow/slack) solution that allows sublinear time updates – requiring $\widetilde{O}(\sqrt{nK})$ time for a batch update consisting of updating the flow value of K edges. This ends up not being a bottleneck for the overall performance because the interior point method only takes roughly $\sqrt{n}$ iterations and makes K-sparse updates roughly $\sqrt{n/K}$ times, resulting in a total running time of $\widetilde{O}(n)$.

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

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 devising 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	Time bound	Reference
Strongly polytime	$O(m^2 \log n + mn \log^2 n)$	[Orl88]
Weakly polytime	$\widetilde{O}((m + n^{3/2}) \log^2 M)$	[vdBLL+21]
Unit-capacity	$m^{\frac{4}{3}+o(1)} \log M$	[AMV20]
Planar graph	$\widetilde{O}(n \log M)$	**this paper**
Unit-capacity planar graph	$O(n^{4/3} \log M)$	[KS19]
Graph with treewidth τ	$\widetilde{O}(n\tau^2 \log M)$	[DLY21a]
Outerplanar graph	$O(n \log^2 n)$	[KN13]
Unidirectional, bidirectional cycle	$O(n)$, $O(n \log n)$	[VA10]

Table 1: Fastest known exact algorithms for the min-cost flow problem, ordered by the generality of the result. Here, n is the number of vertices, m is the number of edges, and M is the maximum of edge capacity and cost value. After the preliminary version of this work was published at SODA 2022, the best weakly polytime algorithm was improved to $\widetilde{O}(m^{1+o(1)} \log^2 M)$ by [CKL+22].

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 [KRT94, AMO88]. For the approximate max-flow problem, we refer the reader to the recent papers [CKM+11, She13, KLOS14, She17, ST18, BGS21].

To understand the 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, (shortest path) 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 [ST04]. 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 interior point methods (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 $\widetilde{O}(m^{3/2} \log M)$ runtime [DS08]. 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 [Mad13]. It was later improved and generalized to $O(m^{4/3+o(1)} \log M)$ [AMV20] for the min-cost flow problem. Kathuria et al. [KLS20] 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 [KPSW19].

For 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 [vdBLL+21], and $\widetilde{O}(m^{\frac{3}{2}-\frac{1}{328}}\log M)$ for max-flow on sparse graphs [GLP21]. 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, [vdBGJ+21] gave a runtime of $\widetilde{O}(m^{\frac{3}{2}-\frac{1}{58}}\log^2 M)$ for general min-cost flow following the dynamic electrical flow framework. Most recently, [CKL+22] improved the runtime for general min-cost flow to $\widetilde{O}(m^{1+o(1)}\log^2 M)$ by solving a sequence of approximate undirected minimum-ratio cycles.

Max-flow on planar graphs. The planar max-flow problem has an equally long history. We refer the reader to the thesis [Bor08] for a detailed exposition. In the seminal work of Ford and Fulkerson that introduced the max-flow min-cut theorem, they also gave a max-flow algorithm for s,t-planar graphs (planar graphs where the source and sink lie on the same face)[FF56]. 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 [IS79]. 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 [Has81]. Building on Hassin's work, the current best runtime is $O(n)$ by Henzinger, Klein, Rao, and Subramanian [HKRS97].

For undirected planar graphs, Reif first gave an $O(n\log^2 n)$ time algorithm for finding the max-flow value [Rei83]. Hassin and Johnson then showed how to compute the flow in the same runtime [HJ85]. The current best runtime is $O(n\log\log n)$ by Italiano, Nussbaum, Sankowski, and Wulff-Nilsen [INSW11].

For general planar graphs, Weihe gave the first $O(n\log n)$ time algorithm, assuming the graph satisfies certain connectivity conditions [Wei97]. Later, Borradaile and Klein gave an $O(n\log n)$ time algorithm for any planar graph [BK09].

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 [MN95]. This was then improved to $O(n\log^3 n)$ in [BKM+17].

For generalizations of planar graphs, Chambers, Ericskon and Nayyeri gave the first nearly-linear time algorithm for max-flow on graphs embedded on bounded-genus surfaces [CEN12]. 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 [MP13], although this is superseded by the previously mentioned works for general graphs [She13, KLOS14].

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 [II90]. 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, [AKLR18, LR19] 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 [KS19]. Currently, bounded treewidth graphs is the only graph family we know that admits min-cost flow algorithms that run in nearly-linear time [DLY21a].

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 [KNK93]. Prior to our result, the fastest min-cost flow algorithm for planar graphs is based on interior point methods (IPMs) and takes $\widetilde{O}(n^{3/2} \log M)$ time [DS08]. 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 [CKM$^+$11] 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 [**?** KPSW19, AS20, ABKS21], 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 [Mad13, Mad16, CMSV17, KLS20, AMV20], 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 on utilizing data structures for designing faster IPM algorithms for general linear programs and flow problems on general graphs. For general linear programs, robust interior point methods have been developed recently with running times that essentially match the matrix multiplication cost [CLS21, vdB20, vdBLSS20, HJST21, vdB21]. 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 i-th iteration of a robust IPM involves computing an electrical flow on some graph G_i. The edge support remains unchanged between iterations, though the edge weights change. Further, if K_i is the number of edges with weight changes between G_i and G_{i+1}, then robust IPMs guarantee that

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

Roughly, this says that, on average, each edge weight changes only poly-log many times throughout the algorithm. Unfortunately, any sparsity bound is not enough to achieve nearly-linear time. Unlike the shortest path problem, changing *any* edge in a connected graph will result in the electrical flow changing on essentially *every* edge. Therefore, it is very difficult to implement (robust) IPMs in sublinear time per iteration, even if the subproblem barely changes every iteration. On moderately dense graphs with $m = \Omega(n^{1.5})$, this issue can be avoided by first approximating the graph by sparse graphs and solving the electrical flow on the sparse graphs. This leads to $\widetilde{O}(n) \ll \widetilde{O}(m)$ time cost per step [vdBLSS20]. 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 [GLP21, vdBGJ$^+$21]. Unfortunately, this still requires $m^{1-\frac{1}{58}}$ time per iteration.

Finally, we note that [DLY21a] 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, 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 paper follows the approach in [DLY21a] 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 [KLS20, AMV20], resulting in super-linear time and polynomial dependence on capacities, rather than logarithmic.

1.3 Our approaches

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

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

where $\mathbf{B} \in \mathbb{R}^{m\times n}$ is an edge-vertex incidence matrix of the graph, $\boldsymbol{f}$ is the flow and $\boldsymbol{s}$ is the slack (or adjusted cost vector). The primal is the min-cost circulation problem 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 2.1) and new data structures (Section 2.3). 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 roughly in $\widetilde{O}(\sqrt{m})$ 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 $\boldsymbol{f}$ to 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 $\mathbf{W}$ and direction $\boldsymbol{v}$ in each projection step. Both $\mathbf{W}$ and $\boldsymbol{v}$ are roughly chosen from some explicit entry-wise formula of $\boldsymbol{f}$ and $\boldsymbol{s}$, namely, $\mathbf{W}_{ii} = \psi_1(\boldsymbol{f}_i, \boldsymbol{s}_i)$ and $\boldsymbol{v}_i = \psi_2(\boldsymbol{f}_i, \boldsymbol{s}_i)$. Hence, the main bottleneck is to implement the projection step (computing $\mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}$). For the min-cost flow problem, this projection step corresponds to an electrical flow computation.

Recently, it has been observed that there is a lot of freedom in choosing the weight $\mathbf{W}$ and the direction $\boldsymbol{v}$ (see for example [CLS21]). Instead of computing them exactly, we maintain some entry-wise approximation $\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}$ of $\boldsymbol{f}, \boldsymbol{s}$ and use them to compute $\mathbf{W}$ and $\boldsymbol{v}$. By updating $\overline{\boldsymbol{f}}_i, \overline{\boldsymbol{s}}_i$ only when $\boldsymbol{f}_i, \boldsymbol{s}_i$ changed significantly, we can ensure $\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}$ has mostly sparse updates. Since $\mathbf{W}$ and $\boldsymbol{v}$ are given by some entry-wise formula of $\overline{\boldsymbol{f}}$ and $\overline{\boldsymbol{s}}$, this ensures that $\mathbf{W}, \boldsymbol{v}$ change sparsely and in turn allows us to maintain the corresponding projection $\mathbf{P}_{\boldsymbol{w}}$ via low-rank updates.

We refer to IPMs that use approximate $\overline{\boldsymbol{f}}$ and $\overline{\boldsymbol{s}}$ as robust IPMs. In this paper, we apply the version given in [DLY21a] in a black-box manner. In Section 2.1, we state the IPM we use. 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 [LRT79, GT87]. By recursively partitioning the graph into edge-disjoint subgraphs (i.e. regions) using balanced vertex separators, we can construct a hierarchical decomposition of a planar

graph G which is called a *separator tree* [FR06]. This is a binary search tree over the edges in G. Each node in the separator tree represents a region in G. In planar graphs, for a region H with $|H|$ vertices, an $O(\sqrt{|H|})$-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 subregions partition the edges in H and share only vertices in the separator. We call the set of vertices in a region H that appear in the separators of its ancestors the *boundary* 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 [LRT79, GT87] essentially replace each region by a graph involving 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 as shortest path, this primitive involving dense graphs can be further accelerated using additional properties of distance matrices [FR06].

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 [MP13]. However, that work predated (and in some sense motivated) subsequent works on nearly-linear time approximations of Schur complements on general graphs [KLP$^+$16, KS16, Kyn17]. 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 an 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 (which is required in implementating the IPM step), 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 (regions) 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 coordinates changed in $\mathbf{W}$ and $\boldsymbol{v}$ in the step. Such a recursive approximate Schur complement structure was used in [GHP18], where the authors achieved a running time 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 changes (by more than a constant factor) in $\mathbf{W}$ 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 $\sqrt{m}\left(\sum_{i=1}^{\widetilde{O}(\sqrt{m})}\sqrt{K_i}\right) = \widetilde{O}(m^{1.25})$. 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 $(i-2^l)$-th step for some $l \le \ell_i$, 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 prove 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, this gives 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 Overview

In this section, we give formal statements of the main theorems proved in the paper, along with the proof for our main result. We provide a high-level explanation of the algorithm, sometimes using a simplified setup.

The main components of this paper are: the IPM from [DLY21a] (Section 2.1); a data structure to maintain a collection of Schur complements via nested dissection of the graph (Section 2.2); abstract data structures to maintain the solutions $\boldsymbol{s}, \boldsymbol{f}$ implicitly, notably using an abstract tree operator (Section 2.3); a sketching-based data structure to maintain the approximations $\overline{\boldsymbol{s}}$ and $\overline{\boldsymbol{f}}$ needed in the IPM (Section 2.4); and finally, the definition of the tree operators for slack and flow corresponding to the IPM projection matrices onto their respective feasible subspaces, along with the complete IPM data structure for slack and flow (Sections 2.5 and 2.6).

We extend our result to α-separable graphs in Section 9.

2.1 Robust interior point method

In this subsection, we explain the robust interior point method developed in [DLY21a], which is a refinement of the methods in [CLS21, vdB20]. Although there are many other robust interior point methods, we simply refer to this method as RIPM. Consider a linear program of the form[2]

$$\min_{\boldsymbol{f} \in \mathcal{F}} \boldsymbol{c}^\top \boldsymbol{f} \quad \text{where} \quad \mathcal{F} = \{\mathbf{B}^\top \boldsymbol{f} = \boldsymbol{b},\ \boldsymbol{l} \le \boldsymbol{f} \le \boldsymbol{u}\} \tag{2.1}$$

for some matrix $\mathbf{B} \in \mathbb{R}^{m \times n}$. As with many other IPMs, RIPM follows the central path $\boldsymbol{f}(t)$ from an interior point ($t \gg 0$) to the optimal solution ($t = 0$):

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

where the term ϕ controls how close the flow $\boldsymbol{f}_i$ can be to the capacity constraints $\boldsymbol{u}_i$ and $\boldsymbol{l}_i$. Following the central path exactly is expensive. Instead, RIPM maintains feasible primal and dual solution $(\boldsymbol{f}, \boldsymbol{s}) \in \mathcal{F} \times \mathcal{S}$, where $\mathcal{S}$ is the dual space given by $\mathcal{S} = \{\boldsymbol{s} : \mathbf{B}\boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c} \text{ for some } \boldsymbol{y}\}$, and ensures $\boldsymbol{f}(t)$ is an approximate minimizer. Specifically, the optimality condition for $\boldsymbol{f}(t)$ is given by

$$\begin{aligned} \mu^t(\boldsymbol{f}, \boldsymbol{s}) &\stackrel{\text{def}}{=} \boldsymbol{s}/t + \nabla\phi(\boldsymbol{f}) = \mathbf{0} \\ (\boldsymbol{f}, \boldsymbol{s}) &\in \mathcal{F} \times \mathcal{S} \end{aligned} \tag{2.2}$$

[2] Although the min-cost flow problem can be written as a one-sided linear program, it is more convenience 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.

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

$$\|\gamma^t(\boldsymbol{f}, \boldsymbol{s})\|_\infty \leq \frac{1}{C \log m} \text{ where } \gamma^t(\boldsymbol{f}, \boldsymbol{s})_i = \frac{\mu^t(\boldsymbol{f}, \boldsymbol{s})_i}{(\nabla^2 \phi(\boldsymbol{f}))_{ii}^{1/2}}, \tag{2.3}$$

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 key subroutine CENTERING 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 subroutine 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. [DLY21a] proved that even if each step is computed approximately, CENTERING 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. See Algorithm 1 for a simplified version.

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

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

where $\boldsymbol{c}^{(1)} = \boldsymbol{c}$, 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. The second call to CENTERING finds an approximate solution by following the central path of the original linear program. Note that both calls run the same algorithm on essentially the same graph: The only difference is that in the first call to CENTERING, each edge e of G becomes three copies of the edge with flow value $\boldsymbol{f}_e^{(1)}, \boldsymbol{f}_e^{(2)}, \boldsymbol{f}_e^{(3)}$. Note that this edge duplication does not affect planarity.

We note that the IPM algorithm only 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. We only require $(\boldsymbol{f}, \boldsymbol{s})$ explicitly when returning the approximately optimal solution at the end of the algorithm Line 26.

Theorem 3. *Consider the linear program*

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

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* $\boldsymbol{l} + r \leq \boldsymbol{f}_\circ \leq \boldsymbol{u} - r$.[3] *Let* $L = \|\boldsymbol{c}\|_2$ *and* $R = \|\boldsymbol{u} - \boldsymbol{l}\|_2$. *For any* $0 < \epsilon \leq 1/2$, *the algorithm* RIPM *(Algorithm 1) finds* $\boldsymbol{f}$ *such that* $\mathbf{B}^\top \boldsymbol{f} = \boldsymbol{b}$, $\boldsymbol{l} \leq \boldsymbol{f} \leq \boldsymbol{u}$ *and*

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

Furthermore, the algorithm has the following properties:

- *Each call of* CENTERING *involves* $O(\sqrt{m} \log m \log(\frac{mR}{\epsilon r}))$ *many steps, and* $\overline{t}$ *is only updated* $O(\log m \log(\frac{mR}{\epsilon r}))$ *times.*

[3] For any vector $\boldsymbol{v}$ and scalar x, we define $\boldsymbol{v} + x$ to be the vector obtained by adding x to each coordinate of $\boldsymbol{v}$. We define $\boldsymbol{v} - x$ to be the vector obtained by subtracting x from each coordinate of $\boldsymbol{v}$.

Algorithm 1 Robust Interior Point Method from [DLY21a]

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

▷ Modify the linear program and obtain an initial (x, s) for modified linear program
  Let $t = 2^{21} m^5 \cdot \frac{LR}{128} \cdot \frac{R}{r}$
  Compute $\boldsymbol{f}_c = \arg\min_{\boldsymbol{l} \le \boldsymbol{f} \le \boldsymbol{u}} \boldsymbol{c}^\top \boldsymbol{f} + t\phi(\boldsymbol{f})$ and $\boldsymbol{f}_\circ = \arg\min_{\mathbf{B}^\top \boldsymbol{f} = \boldsymbol{b}} \|\boldsymbol{f} - \boldsymbol{f}_c\|_2$
  Let $\boldsymbol{f} = (\boldsymbol{f}_c, 3R + \boldsymbol{f}_\circ - \boldsymbol{f}_c, 3R)$ and $\boldsymbol{s} = (-t\nabla\phi(\boldsymbol{f}_c), \frac{t}{3R + \boldsymbol{f}_\circ - \boldsymbol{f}_c}, \frac{t}{3R})$
  Let the new matrix $\mathbf{B}^{\text{new}} \stackrel{\text{def}}{=} [\mathbf{B}; \mathbf{B}; -\mathbf{B}]$, the new barrier

$$\phi_i^{\text{new}}(x) = \begin{cases} \phi_i(x) & \text{if } i \in [m], \\ -\log x & \text{else.} \end{cases}$$

▷ Find an initial $(\boldsymbol{f}, \boldsymbol{s})$ for the original linear program
  $((\boldsymbol{f}^{(1)}, \boldsymbol{f}^{(2)}, \boldsymbol{f}^{(3)}), (\boldsymbol{s}^{(1)}, \boldsymbol{s}^{(2)}, \boldsymbol{s}^{(3)})) \leftarrow$ CENTERING($\mathbf{B}^{\text{new}}, \phi^{\text{new}}, \boldsymbol{f}, \boldsymbol{s}, t, LR$)
  $(\boldsymbol{f}, \boldsymbol{s}) \leftarrow (\boldsymbol{f}^{(1)} + \boldsymbol{f}^{(2)} - \boldsymbol{f}^{(3)}, \boldsymbol{s}^{(1)})$

▷ Optimize the original linear program
  $(\boldsymbol{f}, \boldsymbol{s}) \leftarrow$ CENTERING($\mathbf{B}, \phi, \boldsymbol{f}, \boldsymbol{s}, LR, \frac{\epsilon}{4m}$)
  return $\boldsymbol{f}$
end procedure

procedure CENTERING($\mathbf{B}, \phi, \boldsymbol{f}, \boldsymbol{s}, t_{\text{start}}, t_{\text{end}}$)
  Let $\alpha = \frac{1}{2^{20}\lambda}$ and $\lambda = 64 \log(256m^2)$ where m is the number of rows in $\mathbf{B}$
  Let $t \leftarrow t_{\text{start}}$, $\overline{\boldsymbol{f}} \leftarrow \boldsymbol{f}, \overline{\boldsymbol{s}} \leftarrow \boldsymbol{s}, \overline{t} \leftarrow t$
  while $t \ge t_{\text{end}}$ **do**
    Set $t \leftarrow \max((1 - \frac{\alpha}{\sqrt{m}})t, t_{\text{end}})$
    Update $h = -\alpha / \|\cosh(\lambda\gamma^{\overline{t}}(\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}))\|_2$ where γ is defined in Eq. (2.2)
    Update the diagonal weight matrix $\mathbf{W} = \nabla^2\phi(\overline{\boldsymbol{f}})^{-1}$
    Update the direction $\boldsymbol{v}$ where $\boldsymbol{v}_i = \sinh(\lambda\gamma^{\overline{t}}(\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}})_i)$
    Pick $\boldsymbol{v}^{\|}$ and $\boldsymbol{v}^{\perp}$ such that $\mathbf{W}^{-1/2}\boldsymbol{v}^{\|} \in \text{Range}(\mathbf{B})$, $\mathbf{B}^\top \mathbf{W}^{1/2} \boldsymbol{v}^{\perp} = \mathbf{0}$ and

$$\|\boldsymbol{v}^{\|} - \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}\|_2 \le \alpha\|\boldsymbol{v}\|_2,$$
$$\|\boldsymbol{v}^{\perp} - (\mathbf{I} - \mathbf{P}_{\boldsymbol{w}})\boldsymbol{v}\|_2 \le \alpha\|\boldsymbol{v}\|_2 \qquad (\mathbf{P}_{\boldsymbol{w}} \stackrel{\text{def}}{=} \mathbf{W}^{1/2}\mathbf{B}(\mathbf{B}^\top\mathbf{W}\mathbf{B})^{-1}\mathbf{B}^\top\mathbf{W}^{1/2})$$

    Implicitly update $\boldsymbol{f} \leftarrow \boldsymbol{f} + h\mathbf{W}^{1/2}\boldsymbol{v}^{\perp}$, $\boldsymbol{s} \leftarrow \boldsymbol{s} + \overline{t}h\mathbf{W}^{-1/2}\boldsymbol{v}^{\|}$
    Explicitly maintain $\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}$ such that $\|\mathbf{W}^{-1/2}(\overline{\boldsymbol{f}} - \boldsymbol{f})\|_\infty \le \alpha$ and $\|\mathbf{W}^{1/2}(\overline{\boldsymbol{s}} - \boldsymbol{s})\|_\infty \le \overline{t}\alpha$
    Update $\overline{t} \leftarrow t$ if $|\overline{t} - t| \ge \alpha\overline{t}$
  end while
  return $(\boldsymbol{f}, \boldsymbol{s})$
end procedure

- *In each step of* CENTERING*, the coordinate i in $\mathbf{W}, \boldsymbol{v}$ changes only if $\overline{\boldsymbol{f}}_i$ or $\overline{\boldsymbol{s}}_i$ changes.*
- *In each step of* CENTERING*, $h\|\boldsymbol{v}\|_2 = O(\frac{1}{\log m})$.*
- *Line 18 to Line 20 takes $O(K)$ time in total, where K is the total number of coordinate changes in $\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}$.*

Proof. The number of steps follows from Theorem A.1 in [DLY21b], with the parameter $w_i = \nu_i = 1$ for all i. The number of coordinate changes in $\mathbf{W}, \boldsymbol{v}$ and the runtime of Line 18 to Line 20 follows directly from the formula of $\mu^t(\boldsymbol{f}, \boldsymbol{s})_i$ and $\gamma^t(\boldsymbol{f}, \boldsymbol{s})_i$. For the bound for $h\|\boldsymbol{v}\|_2$, it follows from

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

□

A key idea in our paper involves the computation of projection matrices required for the RIPM. Recall from the definition of $\mathbf{P}_{\boldsymbol{w}}$ in Algorithm 1, the true projection matrix is

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

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

$$\mathbf{P}_{\boldsymbol{w}} = \mathbf{W}^{1/2}\mathbf{B}\mathbf{L}^{-1}\mathbf{B}^\top\mathbf{W}^{1/2}. \tag{2.4}$$

Lemma 4. *To implement Line 21 in Algorithm 1, it suffices to find an approximate slack projection matrix $\widetilde{\mathbf{P}}_{\boldsymbol{w}}$ satisfying $\left\|\left(\widetilde{\mathbf{P}}_{\boldsymbol{w}} - \mathbf{P}_{\boldsymbol{w}}\right)\boldsymbol{v}\right\|_2 \leq \alpha\|\boldsymbol{v}\|_2$ and $\mathbf{W}^{-1/2}\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v} \in \text{Range}(\mathbf{B})$; and an approximation flow projection matrix $\widetilde{\mathbf{P}}'_{\boldsymbol{w}}$ satisfying $\left\|\left(\widetilde{\mathbf{P}}'_{\boldsymbol{w}} - \mathbf{P}_{\boldsymbol{w}}\right)\boldsymbol{v}\right\|_2 \leq \alpha\|\boldsymbol{v}\|_2$ and $\mathbf{B}^\top\mathbf{W}^{1/2}\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v} = \mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}$.*

Proof. We simply observe that setting $\boldsymbol{v}^\parallel = \widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}$ and $\boldsymbol{v}^\perp = \boldsymbol{v} - \widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v}$ suffices. □

In finding these approximate projection matrices, we apply ideas from nested dissection and approximate Schur complements to the matrix $\mathbf{L}$.

2.2 Nested dissection and approximate Schur complements

In this subsection, we discuss nested dissection and the corresponding Schur complements, and explain how it relates to our goal of finding the approximate projection matrices for Lemma 4.

As we will discuss later in the main proof, our LP formulation for the IPM uses a *modified planar graph* which includes two additional vertices and $O(n)$ additional edges to the original planar graph. Although the modified graph is no longer planar, it has only two additional vertices. We may add these two vertices to any relevant sets defined in nested dissection without changing the overall complexity. As such, we can apply nested dissection as we would for planar graphs.

We first illustrate the key ideas using a two-layer nested dissection scheme. By the well-known planar separator theorem [LT79], a planar graph G can be decomposed into two edge-disjoint (not vertex-disjoint) subgraphs H_1 and H_2 called *regions*, such that each subgraph has at most $2n/3$ vertices. 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 bounded by $O(\sqrt{n})$. 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, and let $F = F_{H_1} \cup F_{H_2}$ be the disjoint union of the two interior sets. Note that C is a *balanced vertex separator* of G, with size

$$|C| \leq |\partial H_1| + |\partial H_2| = O(\sqrt{n}).$$

Furthermore, F and C give a natural partition of the vertices of G. Using block Cholesky decomposition, we can now write[4]

$$\mathbf{L}^{-1} = \begin{bmatrix} \mathbf{I} & -\mathbf{L}_{F,C}{\mathbf{L}_{F,F}}^{-1} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} {\mathbf{L}_{F,F}}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{Sc}(\mathbf{L}, C)^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{L}_{F,C}{\mathbf{L}_{F,F}}^{-1} & \mathbf{I} \end{bmatrix}, \tag{2.5}$$

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

The IPM in Algorithm 1 involves updating $\mathbf{L}^{-1}$ in every step; written as the above decomposition, we must in turn update the Schur complement $\mathbf{Sc}(\mathbf{L}, C)$ in 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

$$\widetilde{\mathbf{Sc}}(\mathbf{L}, C) \stackrel{\text{def}}{=} \widetilde{\mathbf{Sc}}(\mathbf{L}[H_1], C) + \widetilde{\mathbf{Sc}}(\mathbf{L}[H_2], C). \tag{2.6}$$

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 $\widetilde{\mathbf{Sc}}(\mathbf{L}, C)^{-1}\boldsymbol{x}$ for some vector $\boldsymbol{x}$, we use a generic 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 edge-disjoint union of the regions of 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,

[4]To keep notation simple, $\mathbf{M}^{-1}$ will denote the Moore-Penrose pseudo-inverse for non-invertible matrices.

analogous to what was shown in the two-layer dissection. Moreover, if we let F_i denote the disjoint union of sets F_H over all nodes H at level i, and C_i denote the union of sets ∂H, then we essentially generalize the set C from the two-layer dissection to $V(G) = C_{-1} \supset C_0 \supset \cdots \supset C_{\eta-1} \supset C_\eta = \emptyset$, where each C_i is some vertex separator of $G \backslash C_{i-1}$, and generalize the set F to $F_0, \ldots, F_\eta$ partitioning $V(G)$, where $F_i \stackrel{\text{def}}{=} C_{i-1} \setminus C_i$. With a height-$\eta$ separator tree, we can write

$$\mathbf{L}^{-1} = \mathbf{U}^{(0)\top} \cdots \mathbf{U}^{(\eta-1)\top} \begin{bmatrix} \mathbf{Sc}(\mathbf{L}, C_{-1})_{F_0,F_0}{}^{-1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{Sc}(\mathbf{L}, C_{\eta-1})_{F_\eta,F_\eta}{}^{-1} \end{bmatrix} \mathbf{U}^{(\eta-1)} \cdots \mathbf{U}^{(0)}, \quad (2.7)$$

for some explicit upper triangular matrices $\mathbf{U}^{(i)}$. Here, $\mathbf{Sc}(\mathbf{L}, C_i)_{F_{i+1},F_{i+1}}$ denotes the $F_{i+1} \times F_{i+1}$ submatrix of $\mathbf{Sc}(\mathbf{L}, C_i)$.

In the expression Eq. (2.7), the Schur complement term $\mathbf{Sc}(\mathbf{L}, C_i)$ at level i can further be decomposed at according to the nodes at the level. Then, we can obtain an approximation to $\mathbf{L}^{-1}$ by using approximate Schur complements as follows:

Theorem 5 ($\mathbf{L}^{-1}$ approximation)**.** *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

$$\mathbf{L}^{-1} \approx_{\eta\epsilon_\mathbf{P}} \mathbf{\Pi}^{(0)\top} \cdots \mathbf{\Pi}^{(\eta-1)\top} \widetilde{\mathbf{\Gamma}} \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)}, \quad (2.8)$$

where

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

and

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

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.

Compared to *Eq.* (2.7), we see that $\widetilde{\mathbf{\Gamma}}$ approximates the middle block-diagonal matrix, and $\mathbf{\Pi}^{(i)}$ approximates $\mathbf{U}^{(i)}$.

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 modified planar graph G with m edges and its separator tree $\mathcal{T}$ with height $\eta = O(\log m)$, the deterministic data structure* DynamicSC *(Algorithm 3) maintains the edge weights $\boldsymbol{w}$ from the IPM, and at every node $H \in \mathcal{T}$, maintains two vertex sets F_H and ∂H, and two Laplacians $\mathbf{L}^{(H)}$ and $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H \cup F_H)$ dependent on $\boldsymbol{w}$. It supports the following procedures:*

- Initialize$(G, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \epsilon_\mathbf{P} > 0)$*: Given a graph G, initial weights $\boldsymbol{w}$, projection matrix approximation accuracy $\epsilon_\mathbf{P}$, preprocess in $\widetilde{O}({\epsilon_\mathbf{P}}^{-2} m)$ time.*

- REWEIGHT($\boldsymbol{w} \in \mathbb{R}^m_{>0}$, *given implicitly as a set of changed coordinates*): *Update the weights to* $\boldsymbol{w}$*, and update the relevant Schur complements in* $\widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}\sqrt{mK})$ *time, where* K *is the number of coordinates changed in* $\boldsymbol{w}$*.*

  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 time* $\widetilde{O}\left(\epsilon_{\mathbf{P}}{}^{-2}|\partial H \cup F_H|\right)$*.*

- *Access to Laplacian* $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ *at any node* $H \in \mathcal{T}$ *in time* $\widetilde{O}\left(\epsilon_{\mathbf{P}}{}^{-2}|\partial H|\right)$*.*

Furthermore, the $\mathbf{L}^{(H)}$*'s maintained by the data structure satisfy*

$$\mathbf{L}^{(H)} \approx_{\epsilon_{\mathbf{P}}} \mathbf{Sc}(\mathbf{L}[H], \partial H \cup F_H), \tag{2.9}$$

for all $H \in \mathcal{T}$ *with high probability. The* $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$*'s maintained satisfy*

$$\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H) \approx_{\epsilon_{\mathbf{P}}} \mathbf{Sc}(\mathbf{L}[H], \partial H) \tag{2.10}$$

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

2.3 Implicit representations using tree operator

In this section, we outline the data structures for maintaining the flow and slack solutions $\boldsymbol{f}, \boldsymbol{s}$ as needed in Algorithm 1, Line 22. Recall from Lemma 4, at IPM step k with step direction $\boldsymbol{v}^{(k)}$, we want to update

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

for some approximate projection matrices $\widetilde{\mathbf{P}}_{\boldsymbol{w}}$ and $\widetilde{\mathbf{P}}'_{\boldsymbol{w}}$ satisfying $\text{Range}(\mathbf{W}^{-1/2}\widetilde{\mathbf{P}}_{\boldsymbol{w}}) \subseteq \text{Range}(\mathbf{B})$ 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 $\mathbf{M}^{(\text{slack})}, \mathbf{M}^{(\text{flow})}$ 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

$$\begin{aligned}\boldsymbol{s} &= \boldsymbol{s}_0 + \mathbf{M}^{(\text{slack})}\boldsymbol{z}\\ \boldsymbol{f}^\perp &= \boldsymbol{f}_0^\perp + \mathbf{M}^{(\text{flow})}\boldsymbol{z}.\end{aligned}$$

In this subsection, we abstract away the difference between slack and flow, and give a general data structure MAINTAINREP to maintain $\boldsymbol{x} = \boldsymbol{y} + \mathbf{M}\boldsymbol{z}$ for $\mathbf{M}$ with a special tree structure.

At a high level, MAINTAINREP implements 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 first computes $\boldsymbol{z}^{(k)}$ as a function of $\boldsymbol{v}^{(k)}$, then updates $\boldsymbol{z} \leftarrow \boldsymbol{z} + \alpha^{(k)}\boldsymbol{z}^{(k)}$, which translates to the desired overall update in $\boldsymbol{x}$ of $\boldsymbol{x} \leftarrow \boldsymbol{x} + \mathbf{M}(\alpha^{(k)}\boldsymbol{z}^{(k)})$. To reweight with new weights $\boldsymbol{w}^{(\text{new})}$ (which does not change the value of $\boldsymbol{x}$), the data structure first computes $\mathbf{M}^{(\text{new})}$ using $\boldsymbol{w}^{(\text{new})}$ and $\Delta\mathbf{M} \stackrel{\text{def}}{=} \mathbf{M}^{(\text{new})} - \mathbf{M}$, then updates

$\mathbf{M} \leftarrow \mathbf{M}^{(\text{new})}$. This causes an increase in value in the $\mathbf{M}\boldsymbol{z}$ term by $\Delta\mathbf{M}\boldsymbol{z}$, which is then offset in the $\boldsymbol{y}$ term with $\boldsymbol{y} \leftarrow \boldsymbol{y} - \Delta\mathbf{M}\boldsymbol{z}$.

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

Let us now discuss the definition of $\boldsymbol{z}$, 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 $\boldsymbol{\Pi}^{(0)}, \cdots, \boldsymbol{\Pi}^{(\eta-1)}, \widetilde{\boldsymbol{\Gamma}}$ based on the current weights $\boldsymbol{w}$, which together give an $\eta\epsilon_{\mathbf{P}}$-approximation of $\mathbf{L}^{-1}$. MaintainRep will contain a DynamicSC data structure, so we can use these Laplacians in the definition of $\boldsymbol{z}$:

At step k, let

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

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

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

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

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

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

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

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

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

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

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

Let us briefly discuss how to compute $\widetilde{\boldsymbol{\Gamma}}\boldsymbol{\Pi}^{(\eta-1)} \cdots \boldsymbol{\Pi}^{(0)}\boldsymbol{d}$ for some vector $\boldsymbol{d}$. We use the two-layer nested dissection setup from Section 2.2 for intuition, so

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

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 $\widetilde{\mathbf{Sc}}(\mathbf{L}, C)^{-1}$, 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 $\mathbb{R}^{V(G)}$ to $\mathbb{R}^{E(G)}$. 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 $\partial P \cup F_P$ to vectors supported on $\partial H \cup F_H$. If H is a leaf node, the leaf operator for H will map vectors on $\partial H \cup F_H$ 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)**.** *Given a modified planar graph G with n vertices and m edges, and its separator tree $\mathcal{T}$ 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 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,*

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

  where $\alpha^{(i)}$ is the step size α given in Move *for step i,*
- *$\boldsymbol{z}^{(\text{step})}$ satisfies $\boldsymbol{z}^{(\text{step})} = \widetilde{\boldsymbol{\Gamma}}\boldsymbol{\Pi}^{(\eta-1)} \cdots \boldsymbol{\Pi}^{(0)} \mathbf{B}^\top \mathbf{W}^{1/2} \boldsymbol{v}$,*
- *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$(G, \mathcal{T}, \mathbf{M}, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \boldsymbol{x}^{(\text{init})} \in \mathbb{R}^m, \epsilon_{\mathbf{P}} > 0)$*: Given a graph G, its separator tree $\mathcal{T}$, 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 projection matrix 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$(\boldsymbol{w} \in \mathbb{R}^m_{>0}$ *given implicitly as a set of changed coordinates): Update the weights to $\boldsymbol{w}$. 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 the number of coordinates changed in $\boldsymbol{w}$ 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($\alpha \in \mathbb{R}$, $\boldsymbol{v} \in \mathbb{R}^n$ *given implicitly as a set of changed coordinates): Update the current direction to $\boldsymbol{v}$, and then $\boldsymbol{z}^{(\text{step})}$ to maintain the claimed invariant. 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, where K is the number of coordinates changed in $\boldsymbol{v}$ compared to the previous IPM step.

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

2.4 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 [DLY21a] 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 $\frac{\delta}{2\lceil \log m \rceil}$ from $\boldsymbol{x}$, it is updated. We can prove that each coordinate of $\overline{\boldsymbol{x}}$ has additive error at most δ 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 $\mathbf{M}, c, \boldsymbol{z}^{(\text{step})}, \boldsymbol{z}^{(\text{sum})}, \boldsymbol{y}, \mathbf{D}$ at every IPM step, and maintains the approximation $\overline{\boldsymbol{x}}$ to $\boldsymbol{x} \stackrel{\text{def}}{=} \boldsymbol{y} + \mathbf{M}\boldsymbol{z} = \boldsymbol{y} + \mathbf{M}(c \cdot \boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})})$ satisfying $\left\|\mathbf{D}^{1/2}(\boldsymbol{x} - \overline{\boldsymbol{x}})\right\|_\infty \leq \delta$. It supports the following procedures:*

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

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

Suppose $\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\|_{\mathbf{D}^{(k+1)}} \leq \beta$ 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 first updates* $\overline{\boldsymbol{x}}_i \leftarrow \boldsymbol{x}_i^{(k-1)}$ *for the coordinates* i *with* $\mathbf{D}_{ii}^{(k)} \neq \mathbf{D}_{ii}^{(k-1)}$*, then updates* $\overline{\boldsymbol{x}}_i \leftarrow \boldsymbol{x}_i^{(k)}$ *for* $O(N_k \stackrel{\text{def}}{=} 2^{2\ell_k}(\beta/\delta)^2 \log^2 m)$ *coordinates, where* ℓ_k *is the largest integer* ℓ *with* $k = 0 \bmod 2^\ell$*.*

- *The amortized time cost of* APPROXIMATE *is*

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

  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* $\boldsymbol{y}_e$ *or* $\mathbf{D}_{e,e}$ *changed for some edge* e *in* H*, compared to the* $(k-1)$*-th step.*

2.5 Slack projection

We want to use a MAINTAINREP data structure to implicitly maintain the slack solution $\boldsymbol{s}$ throughout the IPM, and use a MAINTAINAPPROX data structure to explicitly maintain the approximate slack solution $\overline{\boldsymbol{s}}$.

To use MAINTAINREP, it remains to define a suitable tree operator $\mathbf{M}^{(\text{slack})}$, so that at IPM step k, the update in MAINTAINREP is the correct IPM slack update; that is:

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

Let $\widetilde{\mathbf{L}}^{-1}$ denote the approximation of $\mathbf{L}^{-1}$ from Eq. (2.8), maintained and computable with a DYNAMICSC data structure. We define

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

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, this suffices as our approximate slack projection matrix.

Using Section 2.3, we can write

$$\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}^{(k)} = \mathbf{W}^{1/2}\mathbf{B}\boldsymbol{\Pi}^{(0)\top} \cdots \boldsymbol{\Pi}^{(\eta-1)\top}\boldsymbol{z}^{(\text{step})}, \tag{2.11}$$

where $\boldsymbol{z}^{(\text{step})} = \widetilde{\boldsymbol{\Gamma}}\boldsymbol{\Pi}^{(\eta-1)} \cdots \boldsymbol{\Pi}^{(0)}\mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}^{(k)}$ at the end of IPM step k, as defined in the previous section. The remaining matrix multiplication on the left in Eq. (2.11) can indeed be represented by a tree operator $\mathbf{M}$ on the tree $\mathcal{T}$. Intuitively, observe that each $\boldsymbol{\Pi}^{(i)}$ operates on level i of $\mathcal{T}$ and can be decomposed to be written in terms of the nodes at level i. Furthermore, the $\boldsymbol{\Pi}^{(i)}$'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 $\mathbf{M}^{(\text{slack})}$ to be $\mathbf{W}^{1/2}\mathbf{M}$, which is also a valid tree operator.

Now, we state the full data structure for maintaining slack.

Theorem 9 (Slack maintenance)**.** *Given a modified planar graph G with m edges and its separator tree $\mathcal{T}$ with height η, the randomized data structure* MaintainSlack *(Algorithm 9) implicitly maintains the slack solution $\boldsymbol{s}$ undergoing IPM changes, and explicitly maintains its approximation $\overline{\boldsymbol{s}}$, and supports the following procedures with high probability against an adaptive adversary:*

- Initialize($G, \boldsymbol{s}^{(\text{init})} \in \mathbb{R}^m, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \epsilon_{\mathbf{P}} > 0, \overline{\epsilon} > 0$)*: Given a graph G, initial solution $\boldsymbol{s}^{(\text{init})}$, 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 representations $\boldsymbol{s} \leftarrow \boldsymbol{s}^{(\text{init})}$ and $\overline{\boldsymbol{x}} \leftarrow \boldsymbol{s}$.*
- Reweight($\boldsymbol{w} \in \mathbb{R}^m_{>0}$, *given implicitly as a set of changed weights): Set the current weights to $\boldsymbol{w}$ in $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK})$ time, where K is the number of coordinates changed in $\boldsymbol{w}$.*
- Move($\alpha \in \mathbb{R}, \boldsymbol{v} \in \mathbb{R}^m$ *given implicitly as a set of changed coordinates): Implicitly update $\boldsymbol{s} \leftarrow \boldsymbol{s} + \alpha\mathbf{W}^{-1/2}\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}$ for some $\widetilde{\mathbf{P}}_{\boldsymbol{w}}$ with $\|(\widetilde{\mathbf{P}}_{\boldsymbol{w}} - \mathbf{P}_{\boldsymbol{w}})\boldsymbol{v}\|_2 \le \eta\epsilon_{\mathbf{P}} \|\boldsymbol{v}\|_2$, and $\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v} \in \text{Range}(\mathbf{B})$. The total runtime is $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK})$ where K is the number of coordinates changed in $\boldsymbol{v}$.*
- Approximate() $\to \mathbb{R}^m$*: Return the vector $\overline{\boldsymbol{s}}$ such that $\|\mathbf{W}^{1/2}(\overline{\boldsymbol{s}} - \boldsymbol{s})\|_\infty \le \overline{\epsilon}$ for the current weight $\boldsymbol{w}$ and the current vector $\boldsymbol{s}$.*
- Exact() $\to \mathbb{R}^m$*: Output the current vector $\boldsymbol{s}$ 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 k-th* Reweight *and* Move *calls. Then at the k-th* Approximate *call,*

- *the data structure first sets $\overline{\boldsymbol{s}}_e \leftarrow \boldsymbol{s}_e^{(k-1)}$ for all coordinates e where $\boldsymbol{w}_e$ changed in the last* Reweight*, then sets $\overline{\boldsymbol{s}}_e \leftarrow \boldsymbol{s}_e^{(k)}$ 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 k-th* Approximate *call is $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{m(K + N_{k-2^{\ell_k}})})$.*

2.6 Flow projection

Similar to slack, we want to use a MaintainRep data structure to implicitly maintain the flow solution $\boldsymbol{f}$ throughout the IPM, and use a MaintainApprox data structure to explicitly maintain the approximate flow solution $\overline{\boldsymbol{f}}$. For the overview, we focus on the non-trivial part of the flow update at every step given by

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

To use MaintainRep, it remains to define a suitable tree operator $\mathbf{M}^{(\text{flow})}$ so that at IPM step k, the update in MaintainRep is the correct IPM flow update; that is:

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

Rather than finding an explicit $\widetilde{\mathbf{P}}'_{\boldsymbol{w}}$ as we did for slack, observe it suffices to find some *weighted flow* $\tilde{\boldsymbol{f}} \approx \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}$ satisfying $\mathbf{B}^\top\mathbf{W}^{1/2}\tilde{\boldsymbol{f}} = \mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}$. (We use the term "weighted flow" to mean it is obtained by multiplying the edge weights $\mathbf{W}$ to some valid flow.) Then the IPM update becomes

$$h\mathbf{W}^{1/2}\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v} = h\mathbf{W}^{1/2}\tilde{\boldsymbol{f}}.$$

Hence, our goal is to write $\mathbf{W}^{1/2}\tilde{\boldsymbol{f}} = \mathbf{M}^{(\text{flow})}\boldsymbol{z}^{(\text{step})}$ for an appropriate weighted flow $\tilde{\boldsymbol{f}}$.

Let us define demands on vertices by $\boldsymbol{d} \stackrel{\text{def}}{=} \mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}$. Unwrapping the definition of $\mathbf{P}_{\boldsymbol{w}}$, we see that the condition of $\tilde{\boldsymbol{f}} \approx \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}$ is actually $\tilde{\boldsymbol{f}} \approx \mathbf{W}^{1/2}\mathbf{B}\mathbf{L}^{-1}\boldsymbol{d}$. The second condition says $\tilde{\boldsymbol{f}}$ is a weighted flow routing demand $\boldsymbol{d}$. Suppose we had $\tilde{\boldsymbol{f}} = \mathbf{W}^{1/2}\mathbf{B}\mathbf{L}^{-1}\boldsymbol{d}$ exactly, then we see immediately that the second condition is satisfied with $\mathbf{B}^\top\mathbf{W}^{1/2}\tilde{\boldsymbol{f}} = \mathbf{B}^\top\mathbf{W}\mathbf{B}\mathbf{L}^{-1}\boldsymbol{d} = \boldsymbol{d}$. To realize the approximation, we make use of the approximation of $\mathbf{L}^{-1}$ from Eq. (2.8). Hence, one important fact about our construction is that when the Schur complements are exact, our flow $\tilde{\boldsymbol{f}}$ agrees with the true electrical flow routing the demand.

In constructing $\tilde{\boldsymbol{f}}$ to route the demand $\boldsymbol{d}$, we show that $\tilde{\boldsymbol{f}}$ can be written as $\mathbf{M}\boldsymbol{z}^{(\text{step})}$, where $\mathbf{M}$ is a tree operator on the tree $\mathcal{T}$, and $\boldsymbol{z}^{(\text{step})}$ is from MaintainRep, and in fact correspond to electric potentials. Here we explain what $\mathbf{M}$ captures intuitively. For simplicity, 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 electric flows using only the corresponding region H, rather than the entire graph. The recursive nature of the decomposition allows us to bound the overall runtime. To show that the resulting flow $\tilde{\boldsymbol{f}}$ indeed is close to the electric 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 67).

Let us illustrate this partially using the two-layer decomposition scheme from Section 2.2: Suppose we have a demand term $\boldsymbol{d}$ *that is non-zero only on vertices of* C. Then, observe that

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

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 Eq. (2.6), 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 electric 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 $\tilde{\boldsymbol{f}} = \sum \tilde{\boldsymbol{f}}_i$. By construction, this $\tilde{\boldsymbol{f}}$ satisfies $\mathbf{B}^\top\mathbf{W}^{1/2}\tilde{\boldsymbol{f}} = \boldsymbol{d} = \mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}$.

In Section 8, we show that this recursive operation can be realized using a tree operator. We then present the full proof for Theorem 10 below, and implement the data structure.

Theorem 10 (Flow maintenance)**.** *Given a modified planar graph G with m edges and its separator tree $\mathcal{T}$ with height η, the randomized 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($G, \boldsymbol{f}^{(\text{init})} \in \mathbb{R}^m, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \epsilon_{\mathbf{P}} > 0, \overline{\epsilon} > 0$)*: Given a graph G, 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($\boldsymbol{w} \in \mathbb{R}^m_{>0}$ *given implicitly as a set of changed weights): Set the current weights to $\boldsymbol{w}$ in $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK})$ time, where K is the number of coordinates changed in $\boldsymbol{w}$.*
- Move($\alpha \in \mathbb{R}, \boldsymbol{v} \in \mathbb{R}^m$ *given implicitly as a set of changed coordinates): 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 $\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v}$, where $\|\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v} - \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}\|_2 \le O(\eta\epsilon_{\mathbf{P}})\|\boldsymbol{v}\|_2$ 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 is the number of coordinates changed in $\boldsymbol{v}$.*
- 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 k-th* Reweight *and* Move *calls. Then at the k-th* Approximate *call,*

- *the data structure first sets $\overline{\boldsymbol{f}}_e \leftarrow \boldsymbol{f}_e^{(k-1)}$ for all coordinates e where $\boldsymbol{w}_e$ changed in the last* Reweight*, then sets $\overline{\boldsymbol{f}}_e \leftarrow \boldsymbol{f}_e^{(k)}$ 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 $k \ne 0$ and $\ell_0 = 0$.*
- *The amortized time for the k-th* Approximate *call is $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{m(K + N_{k-2^{\ell_k}})})$.*

2.7 Main proof

We are now ready to prove our main result. Algorithm 2 presents the implementation of RIPM Algorithm 1 using our data structures.

We first prove a lemma about how many coordinates change in $\boldsymbol{w}$ and $\boldsymbol{v}$ in each step. This is useful for bounding the complexity of each iteration.

Lemma 11. *When updating $\boldsymbol{w}$ and $\boldsymbol{v}$ at the $(k+1)$-th step of the* CenteringImpl *algorithm, $\boldsymbol{w}$ and $\boldsymbol{v}$ change in $O(2^{2\ell_{k-1}}\log^2 m + 2^{2\ell_k}\log^2 m)$ coordinates, where ℓ_k is the largest integer ℓ with $k \equiv 0 \mod 2^\ell$.*

Proof. Since both $\boldsymbol{w}$ and $\boldsymbol{v}$ are an entry-wise function of $\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}$ 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 CenteringImpl run, $\overline{t}$ 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.

Algorithm 2 Implementation of Robust Interior Point Method

procedure CENTERINGIMPL($\mathbf{B}, \phi, \boldsymbol{f}, \boldsymbol{s}, t_{\text{start}}, t_{\text{end}}$)
  G: graph on n vertices and m edges with incidence matrix $\mathbf{B}$
  $\mathcal{S}, \mathcal{F}$: data structures for slack and flow maintenance ▷ Theorems 9 and 10
  $\alpha \stackrel{\text{def}}{=} \frac{1}{2^{20}\lambda}, \lambda \stackrel{\text{def}}{=} 64 \log(256m^2)$
  $t \leftarrow t_{\text{start}}$, $\overline{\boldsymbol{f}} \leftarrow \boldsymbol{f}, \overline{\boldsymbol{s}} \leftarrow \boldsymbol{s}, \overline{t} \leftarrow t$, $\mathbf{W} \leftarrow \nabla^2\phi(\overline{\boldsymbol{f}})^{-1}$ ▷ variable initialization
  $\boldsymbol{v}_i \leftarrow \sinh(\lambda\gamma^{\overline{t}}(\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}})_i)$ for all $i \in [n]$
  ▷ data structure initialization
  $\mathcal{F}$.INITALIZE($G, \boldsymbol{f}, \boldsymbol{v}, \mathbf{W}, \epsilon_{\mathbf{P}} = O(\alpha/\log m), \overline{\epsilon} = \alpha$) ▷ choose $\epsilon_{\mathbf{P}}$ so $\eta\epsilon_{\mathbf{P}} \le \alpha$ in Theorem 9
  $\mathcal{S}$.INITALIZE($G, \overline{t}^{-1}\boldsymbol{s}, \boldsymbol{v}, \mathbf{W}, \epsilon_{\mathbf{P}} = O(\alpha/\log m), \overline{\epsilon} = \alpha$) ▷ and $O(\eta\epsilon_{\mathbf{P}}) \le \alpha$ in Theorem 10
  while $t \ge t_{\text{end}}$ **do**
    $t \leftarrow \max\{(1 - \frac{\alpha}{\sqrt{m}})t, t_{\text{end}}\}$
    Update $h = -\alpha/\|\cosh(\lambda\gamma^{\overline{t}}(\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}))\|_2$ ▷ γ as defined in Eq. (2.2)
    Update the diagonal weight matrix $\mathbf{W} = \nabla^2\phi(\overline{\boldsymbol{f}})^{-1}$
    $\mathcal{F}$.REWEIGHT($\mathbf{W}$) ▷ update the implicit representation of $\boldsymbol{f}$ with new weights
    $\mathcal{S}$.REWEIGHT($\mathbf{W}$) ▷ update the implicit representation of $\boldsymbol{s}$ with new weights
    $\boldsymbol{v}_i \leftarrow \sinh(\lambda\gamma^{\overline{t}}(\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}})_i)$ for all i where $\overline{\boldsymbol{f}}_i$ or $\overline{\boldsymbol{s}}_i$ has changed ▷ update direction $\boldsymbol{v}$
    ▷ $\mathbf{P}_{\boldsymbol{w}} \stackrel{\text{def}}{=} \mathbf{W}^{1/2}\mathbf{B}(\mathbf{B}^\top\mathbf{W}\mathbf{B})^{-1}\mathbf{B}^\top\mathbf{W}^{1/2}$
    $\mathcal{F}$.MOVE($h, \boldsymbol{v}$) ▷ Update $\boldsymbol{f} \leftarrow \boldsymbol{f} + h\mathbf{W}^{1/2}\boldsymbol{v} - h\mathbf{W}^{1/2}\tilde{\boldsymbol{f}}$ with $\tilde{\boldsymbol{f}} \approx \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}$
    $\mathcal{S}$.MOVE($h, \boldsymbol{v}$) ▷ Update $\boldsymbol{s} \leftarrow \boldsymbol{s} + \overline{t}h\mathbf{W}^{-1/2}\tilde{\boldsymbol{s}}$ with $\tilde{\boldsymbol{s}} \approx \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}$
    $\overline{\boldsymbol{f}} \leftarrow \mathcal{F}$.APPROXIMATE() ▷ Maintain $\overline{\boldsymbol{f}}$ such that $\|\mathbf{W}^{-1/2}(\overline{\boldsymbol{f}} - \boldsymbol{f})\|_\infty \le \alpha$
    $\overline{\boldsymbol{s}} \leftarrow \bar{t}\mathcal{S}$.APPROXIMATE() ▷ Maintain $\overline{\boldsymbol{s}}$ such that $\|\mathbf{W}^{1/2}(\overline{\boldsymbol{s}} - \boldsymbol{s})\|_\infty \le \overline{t}\alpha$
    if $|\overline{t} - t| \ge \alpha\overline{t}$ **then**
      $\boldsymbol{s} \leftarrow \overline{t}\mathcal{S}$.EXACT()
      $\overline{t} \leftarrow t$
      $\mathcal{S}$.INITALIZE($G, \overline{t}^{-1}\boldsymbol{s}, \boldsymbol{v}, \mathbf{W}, \epsilon_{\mathbf{P}} = O(\alpha/\log m), \overline{\epsilon} = \alpha$)
    end if
  end while
  return ($\mathcal{F}$.EXACT(), $\overline{t}\mathcal{S}$.EXACT())
end procedure

By Theorem 3, we have $h\|\boldsymbol{v}\|_2 = O(\frac{1}{\log m})$ at all steps. So we apply Theorem 9 and Theorem 10 both with parameters $\beta = O(\frac{1}{\log m})$ and $\overline{\epsilon} = \alpha = \Theta(\frac{1}{\log m})$. We use their conclusions in the following argument. Let the superscript $^{(k)}$ denote the variable at the end of the k-th step.

By definition, $\boldsymbol{w}^{(k+1)}$ is an entry-wise function of $\overline{\boldsymbol{f}}^{(k)}$, and recursively, $\overline{\boldsymbol{f}}^{(k)}$ is an entry-wise function of $\boldsymbol{w}^{(k)}$. We first prove inductively that at step k, $O(2^{2\ell_k}\log^2 m)$ coordinates of $\overline{\boldsymbol{f}}$ change to $\boldsymbol{f}^{(k)}$ where $\boldsymbol{f}^{(k)}$ is the exact solution, and there are no other changes. This allows us to conclude that $\boldsymbol{w}^{(k+1)}$ differ from $\boldsymbol{w}^{(k)}$ on $O(2^{2\ell_k}\log^2 m)$ coordinates.

In the base case at step $k = 1$, because $\boldsymbol{w}^{(1)}$ is equal to the initial weights $\boldsymbol{w}^{(0)}$, only $O(2^{2\ell_1}\log^2 m)$ coordinates $\overline{\boldsymbol{f}}_e$ change to $\boldsymbol{f}_e^{(1)}$. Suppose at step k, a set S of $O(2^{2\ell_k}\log^2 m)$ coordinates of $\overline{\boldsymbol{f}}$ change; that is, $\overline{\boldsymbol{f}}|_S$ is updated to $\boldsymbol{f}^{(k)}|_S$, and there are no other changes. Then at step $k+1$, by definition, $\boldsymbol{w}^{(k+1)}$ differ from $\boldsymbol{w}^{(k)}$ exactly on S, and in turn, $\overline{\boldsymbol{f}}^{(k+1)}|_S$ is set to $\boldsymbol{f}^{(k)}|_S$ again (Line 20 of Algorithm 7). In other words, there is no change from this operation. Then, $O(2^{2\ell_{k+1}}\log^2 m)$ additional coordinates $\overline{\boldsymbol{f}}_e$ change to $\boldsymbol{f}_e^{(k+1)}$.

Now, we bound the change in $\overline{\boldsymbol{s}}$: Theorem 9 guarantees that in the k-th step, there are $O(2^{2\ell_k}\log^2 m)+D$ coordinates in $\overline{\boldsymbol{s}}$ that change, where D is the number of changes between $\boldsymbol{w}^{(k-1)}$ and $\boldsymbol{w}^{(k)}$ and is equal to $O(2^{2\ell_{k-1}}\log^2 m)$ as shown above.

Finally, $\boldsymbol{v}^{(k+1)}$ is an entry-wise function of $\overline{\boldsymbol{f}}^{(k)}$ and $\overline{\boldsymbol{s}}^{(k)}$, so we conclude that $\boldsymbol{v}^{(k+1)}$ and $\boldsymbol{v}^{(k)}$ differ on at most $O(2^{2\ell_k}\log^2 m) + 2\cdot O(2^{2\ell_{k-1}}\log^2 m)$ coordinates. □

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 $\widetilde{O}(n\log M)$ [5] expected time.*

Proof. The proof is structured as follows. We first write the minimum cost flow problem as a linear program of the form Eq. (2.1). We prove the linear program has an interior point and is bounded, so to satisfy the assumptions in Theorem 3. Then, we implement the IPM algorithm using the data structures from Sections 2.3 to 2.6. Finally, we bound the cost of each operations of the data structures.

To write down the min-cost flow problem as a linear program of the form Eq. (2.1), we add extra vertices s and t. Let $\boldsymbol{d}$ be the demand vector of the min-cost flow problem. 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 modified graph is no longer planar but it has only two extra vertices s and t.

The cost and capacity on the $t \to s$ edge is chosen such that the minimum cost flow problem on the original graph is equivalent to the minimum cost circulation on this new graph. Namely, if the minimum cost circulation in this new graph satisfies all the demand $\boldsymbol{d}_v$, then this circulation (ignoring the flow on the new edges) is the minimum cost flow in the original graph.

Since Theorem 3 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

$$\min_{\mathbf{B}^\top \boldsymbol{f} = \mathbf{0},\, \boldsymbol{l}^{\text{new}} \le \boldsymbol{f} \le \boldsymbol{u}^{\text{new}}} (\boldsymbol{c}^{\text{new}})^\top \boldsymbol{f}$$

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

where $\mathbf{B}$ is the signed incidence matrix of the new graph, $\boldsymbol{c}^{\text{new}}$ is the new cost vector (with cost on extra edges), and $\boldsymbol{l}^{\text{new}}, \boldsymbol{u}^{\text{new}}$ are the new capacity constraints. If an edge e has only one direction, we set $\boldsymbol{l}_e^{\text{new}} = 0$ and $\boldsymbol{u}_e^{(\text{new})} = \boldsymbol{u}_e$, otherwise, we orient the edge arbitrarily and set $-\boldsymbol{l}_e^{\text{new}} = \boldsymbol{u}_e^{\text{new}} = \boldsymbol{u}_e$.

Now, we bound the parameters L, R, r in Theorem 3. 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 is an "interior" flow $\boldsymbol{f}$ 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 $r \geq \frac{1}{4m}$.

The RIPM in Theorem 3 runs the subroutine CENTERING 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 use the implementation given in CENTERINGIMPL (Algorithm 2) for both runs.

By the guarantees of Theorem 9 and Theorem 10, we correctly maintain $\boldsymbol{s}$ and $\boldsymbol{f}$ at every step in CENTERINGIMPL, and the requirements on $\overline{\boldsymbol{f}}$ and $\overline{\boldsymbol{s}}$ for the RIPM are satisfied. Hence, Theorem 3 shows that we can find a circulation $\boldsymbol{f}$ 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. Note that $\boldsymbol{f}$, when restricted to the original graph, is almost a flow routing the required demand with flow value off by at most $\frac{1}{2nM}$. This is because sending extra k units of fractional flow from s to t gives extra negative cost $\leq -knM$. Now we can round $\boldsymbol{f}$ to an integral flow $\boldsymbol{f}^{\text{int}}$ with same or better flow value using no more than $\widetilde{O}(m)$ time [KP15]. Since $\boldsymbol{f}^{\text{int}}$ is integral with flow value at least the total demand minus $\frac{1}{2}$, $\boldsymbol{f}^{\text{int}}$ routes the demand completely. Again, since $\boldsymbol{f}^{\text{int}}$ is integral with cost at most $\text{OPT} - \frac{1}{2}$, $\boldsymbol{f}^{\text{int}}$ must have the minimum cost.

Finally, we bound the runtime of one call to CENTERINGIMPL. We initialize the data structures for flow and slack by INITIALIZE. Here, the data structures are given the first IPM step direction $\boldsymbol{v}$ for preprocessing; the actual step is taken in the first iteration of the main while-loop. At each step of CENTERINGIMPL, we perform the implicit update of $\boldsymbol{f}$ and $\boldsymbol{s}$ using MOVE; we update $\mathbf{W}$ in the data structures using REWEIGHT; and we construct the explicit approximations $\overline{\boldsymbol{f}}$ and $\overline{\boldsymbol{s}}$ using APPROXIMATE; each in the respective flow and slack data structures. We return the true $(\boldsymbol{f}, \boldsymbol{s})$ by EXACT. The total cost of CENTERINGIMPL is dominated by MOVE, REWEIGHT, and APPROXIMATE.

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 3 guarantees that there are $T = O(\sqrt{m} \log n \log(nM))$ total APPROXIMATE calls. Lemma 11 shows that at the k-th call, the number of coordinates changed in $\boldsymbol{w}$ and $\boldsymbol{v}$ is bounded by $K \stackrel{\text{def}}{=} O(2^{2\ell_{k-1}} \log^2 m + 2^{2\ell_{k-2}} \log^2 m)$, where ℓ_k is the largest integer ℓ with $k \equiv 0 \mod 2^\ell$, or equivalently, the number of trailing zeros in the binary representation of k. Theorem 3 further guarantees we can apply Theorem 9 and Theorem 10 with parameter $\beta = O(1/\log m)$, which in turn shows the amortized time for the k-th call is

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

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

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

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

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

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

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 $O(\sqrt{m}T\log m\log^2 T)$. Combined with Theorem 3's guarantee of $T = O(\sqrt{m}\log n\log(nM))$, we conclude the overall runtime is $\widetilde{O}(m\log M)$. □

3 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 abuse of notation is unfortunately unavoidable as we will be considering lots of submatrices and subvectors.

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 guarantees by constant factors.

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 $\boldsymbol{v}^\top\mathbf{M}\boldsymbol{v}$. For any vector $\boldsymbol{v}$ and scalar x, we define $\boldsymbol{v}+x$ to be the vector obtained by adding x to each coordinate of $\boldsymbol{v}$ and similarly $\boldsymbol{v}-x$ to be the vector obtained by subtracting x from each coordinate of $\boldsymbol{v}$. We use $\mathbf{0}$ for all-zero vectors and matrices where dimensions are determined by context. We use $\mathbf{1}_A$ for the vector with value 1 on coordinates in A and 0 everywhere else. We use $\mathbf{I}$ for the identity matrix and $\mathbf{I}_S$ for the identity matrix in $\mathbb{R}^{S\times S}$. For any vector $\boldsymbol{x}\in\mathbb{R}^S$, $\boldsymbol{x}|_C$ denotes the sub-vector of $\boldsymbol{x}$ supported on $C\subseteq S$; *more specifically,* $\boldsymbol{x}|_C\in\mathbb{R}^S$, *where* $\boldsymbol{x}_i=0$ *for all* $i\notin C$.

For any matrix $\mathbf{M}\in\mathbb{R}^{A\times B}$, we use the convention that $\mathbf{M}_{C,D}$ denotes the sub-matrix of $\mathbf{M}$ supported on $C\times D$ where $C\subseteq A$ and $D\subseteq B$. When $\mathbf{M}$ is not symmetric and only one subscript is specified, as in $\mathbf{M}_D$, this denotes the sub-matrix of $\mathbf{M}$ supported on $A\times D$. To keep notations simple, $\mathbf{M}^{-1}$ will denote the inverse of $\mathbf{M}$ if it is an invertible matrix and the Moore-Penrose pseudo-inverse otherwise.

For two positive semi-definite matrices $\mathbf{L}_1$ and $\mathbf{L}_2$, we write $\mathbf{L}_1\approx_t\mathbf{L}_2$ if $e^{-t}\mathbf{L}_1\preceq\mathbf{L}_2\preceq e^t\mathbf{L}_1$, where $\mathbf{A}\preceq\mathbf{B}$ means $\mathbf{B}-\mathbf{A}$ is positive semi-definite. Similarly we define $\ge_t$ and $\le_t$ for scalars, that is, $x\le_t y$ if $e^{-t}x\le y\le e^t x$.

Graphs and Trees. We define *modified planar graph* to mean a graph obtained from a planar graph by adding 2 new vertices s, t and any number of edges incident to the new vertices. We allow distinguishable parallel edges in our graphs. We assume the input graph is connected.

We use n for the number of vertices and m for the number of edges in the input graph. We will use $\boldsymbol{w}$ for the vector of edge weights in a graph. We define $\mathbf{W}$ as the diagonal matrix $\mathrm{diag}(\boldsymbol{w})$.

We define $\mathbf{L} = \mathbf{B}^\top \mathbf{W}\mathbf{B}$ be the Laplacian matrix associated with an undirected graph G with non-negative edge weights $\mathbf{W}$. We at times use a graph and its Laplacian interchangeably. For a subgraph $H \subseteq G$, we use $\mathbf{L}[H]$ to denote the weighted Laplacian on H, and $\mathbf{B}[H]$ to denote the incidence matrix of H.

For a tree $\mathcal{T}$, we write $H \in \mathcal{T}$ to mean H is a node in $\mathcal{T}$. 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 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.

For binary tree data structures, we assume there is constant time access to each node.

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 k-th iteration of the while-loop in Centering (Algorithms 1 and 2); that is, it is the k-th 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 k-th step.

In all procedures in these data structures, we assume inputs are given by the set of changed coordinates and their values, *compared to the previous input*. Similarly, we output a vector by the set of changed coordinates and their values, compared to the previous output. This can be implemented by checking memory for changes.

We use smallCaps to denote function names and data structure classes, and `typewriterFont` to denote an instantiation of a data structure.

We say a data structure B *extends* A in the object-oriented sense. Inside data structure B, we directly access functions and variables of A when the context is clear, or use the keyword `super`.

In the data structure where we write $\mathbf{L}^{-1}\boldsymbol{x}$ for some Laplacian $\mathbf{L}$ and vector $\boldsymbol{x}$, we imply the use of an SDD-solver as a black box in nearly-linear time:

Theorem 12 ([ST04, JS21])**.** *There is a randomized algorithm which is an ε-approximate Laplacian system solver for the any input n-vertex m-edge graph and $\varepsilon \in (0,1)$ and has the following runtime $O(m\mathrm{poly}(\log\log n)\log(1/\varepsilon))$.*

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 necessary for approximating $\mathbf{P}_{\boldsymbol{w}} \stackrel{\text{def}}{=} \mathbf{W}^{1/2}\mathbf{B}(\mathbf{B}^\top\mathbf{W}\mathbf{B})^{-1}\mathbf{B}^\top\mathbf{W}^{1/2}$ as needed in the robust IPM. In particular, we are interested in the weighted Laplacian matrix $\mathbf{L} \stackrel{\text{def}}{=} \mathbf{B}^\top\mathbf{W}\mathbf{B}$.

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

4.1 Cholesky decomposition and Schur complement

Let G be a 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 13 (Block Cholesky decomposition)**.** The *block Cholesky decomposition* of a symmetric $\mathbf{L}$ with blocks indexed by F and C defined as above is:

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

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 14 (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 15. *Let $V(G) = \{v_1, \ldots, v_n\}$. Let $C = V(G) - v_1$. Let $\boldsymbol{w}_{ij}$ 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 $\boldsymbol{w}_{ij} = \boldsymbol{w}_{1i}\boldsymbol{w}_{1j}/\boldsymbol{w}_1$, where $\boldsymbol{w}_1$ is the total weight of edges incident to v_1 in G. Note that on the right hand side, we use a graph to mean its Laplacian. □

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

Lemma 16. *If $X \subseteq Y \subseteq V(G)$, then $\mathbf{Sc}(\mathbf{Sc}(\mathbf{L}, Y), X) = \mathbf{Sc}(\mathbf{L}, X)$.* □

Lemma 17. *Let $\boldsymbol{w}_e$ denote the weight of edge e in G. Suppose $C \subseteq V(G)$, and H is a subgraph of G on the vertex set C with edge weights $\boldsymbol{w}'_e \leq \boldsymbol{w}_e$ for all edges in $G[C]$. Let $\mathbf{L}'$ denote the Laplacian of H. Then, $\mathbf{Sc}(\mathbf{L} - \mathbf{L}', C) = \mathbf{Sc}(\mathbf{L}, C) - \mathbf{L}'$.* □

Lemma 18. *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

$$\begin{aligned}
\mathbf{Sc}(\mathbf{L}, C) &= \mathbf{Sc}(\mathbf{L}_1 + \mathbf{L}_2, C) \\
&= \mathbf{Sc}(\mathbf{Sc}(\mathbf{L}_1 + \mathbf{L}_2, C \cup V_2), C) \\
&= \mathbf{Sc}(\mathbf{Sc}(\mathbf{L}_1, C \cup V_2) + \mathbf{L}_2, C) && \text{(by Lemma 17)} \\
&= \mathbf{Sc}(\mathbf{Sc}(\mathbf{L}_1, C) + \mathbf{L}_2, C) && \text{(since } (C \cup V_2) \cap V_1 \subseteq C\text{)} \\
&= \mathbf{Sc}(\mathbf{L}_1, C) + \mathbf{Sc}(\mathbf{L}_2, C), && \text{(by Lemma 17)} \\
&= \mathbf{Sc}(\mathbf{L}_1, C \cap V_1) + \mathbf{Sc}(\mathbf{L}_2, C \cap V_2) && \text{(since } \mathbf{L}_i \text{ is supported on } V_i \text{ for } i = 1, 2\text{)}
\end{aligned}$$

as desired. □

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 19 (Separable graph)**.** A graph $G = (V, E)$ is α-separable if there exists two constants $c > 0$ and $b \in (0, 1)$ such that every nonempty subgraph $H = (V(H) \subseteq V, E(H) \subseteq E)$ with $|E(H)| \geq 2$ of G can be partitioned into H_1 and H_2 such that

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

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

It is known that any planar graph is 1/2-separable.

Remark 20. As we discussed in Section 2.7, our LP formulation for the IPM uses a *modified planar graph* which is the original planar graph with two additional vertices and $O(n)$ additional edges incident to them. By adding two vertices and edges incident to them to a planar graph, the modified graph is also 1/2-separable with the constant c in Definition 19 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 21 (Separator tree $\mathcal{T}$)**.** Let G be a modified planar graph. A separator tree $\mathcal{T}$ 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 19, 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. Further, 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 22. *Using the above definition, $\{F_H : H \in \mathcal{T}\}$ partitions the vertex set $V(G)$.*

Observation 23. *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 24. *Suppose H is a node in $\mathcal{T}$. Then $\partial H \subseteq \cup_{\text{ancestor } A \text{ of } H} F_A$.*

Fakcharoenphol and Rao [FR06] gave an algorithm that computes the separator tree for any planar graph.

Theorem 25 (Separator tree construction [FR06])**.** *Given a planar graph G, there is an algorithm that computes a separator tree $\mathcal{T}$ of G of height $\eta = O(\log n)$ in $O(n \log n)$ time.*

For computing the separator tree $\mathcal{T}$ of a modified planar 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 the following corollary:

Corollary 26 (Separator tree construction for modified planar graph)**.** *Given a* modified planar graph *G, there is an algorithm that computes a separator tree $\mathcal{T}$ of G of 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 27. 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 [FR06, 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 Appendix A of a slightly weaker bound that in addition requires bounding the number of separator vertices.

Lemma 28. *Let G be a modified planar graph with separator tree $\mathcal{T}$. Let $\mathcal{H}$ be a set of K nodes in $\mathcal{T}$. Then*

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

4.3 Approximating $\mathbf{L}^{-1}$ using the separator tree

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

Definition 29 (C_i, F_i)**.** Let $\mathcal{T}$ be the separator tree from Corollary 26. For all $0 \leq i \leq \eta$, we define $F_i = \bigcup_{H \in \mathcal{T}(i)} F_H$ to be the vertices eliminated at level i. For all $0 \leq i \leq \eta$, we 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 22, 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 $C_i = \cup_{j>i} F_j$.

Now, the decomposition from Eq. (4.1) can be extended and inverted as follows:

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

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

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

where we assume all matrices are $n \times n$ by padding zeroes when required. To efficiently compute parts of $\mathbf{L}^{-1}$, we use approximate Schur complements instead of exact ones in Eq. (4.2).

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 $\widetilde{\mathbf{Sc}}(\mathbf{L}, C) \in \mathbb{R}^{C \times C}$ 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.

Definition 31 ($\mathbf{L}^{(H)}$)**.** Let $\epsilon_{\mathbf{P}} > 0$. For each $H \in \mathcal{T}$, let $\mathbf{L}^{(H)}$ be 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.

Here, we define the necessary approximate matrices and show how to approximate $\mathbf{L}^{-1}$.

Definition 32 ($\mathbf{\Pi}^{(i)}, \mathbf{X}^{(H)}, \widetilde{\mathbf{\Gamma}}$)**.** To approximate $\mathbf{U}^{(i)}$, we define

$$\mathbf{\Pi}^{(i)} = \mathbf{I} - \sum_{H \in \mathcal{T}(i)} \mathbf{X}^{(H)}, \quad (4.3)$$

where

$$\mathbf{X}^{(H)} = \mathbf{L}^{(H)}_{\partial H, F_H} \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1} \quad (4.4)$$

for each $H \in \mathcal{T}$.

To approximate the block diagonal matrix in Eq. (4.2), we define

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

Theorem 33 ($\mathbf{L}^{-1}$ approximation)**.** *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

$$\mathbf{L}^{-1} \approx_{\eta\epsilon_{\mathbf{P}}} \mathbf{\Pi}^{(0)\top} \cdots \mathbf{\Pi}^{(\eta-1)\top} \widetilde{\mathbf{\Gamma}} \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)}. \tag{4.5}$$

Proof. Let C_i, F_i be defined for each i according to Definition 29. Let $\mathbf{L}^{(i)} \stackrel{\text{def}}{=} \sum_{H \in \mathcal{T}(i)} \mathbf{L}^{(H)}$.

Note that $\mathbf{L}^{(i)}_{F_i,F_i} \stackrel{\text{def}}{=} \sum_{H \in \mathcal{T}(i)} \mathbf{L}^{(H)}_{F_H,F_H}$ is a block-diagonal matrix with blocks indexed by $H \in \mathcal{T}(i)$, since F_i is a disjoint union over F_H for $H \in \mathcal{T}(i)$, and only $\mathbf{L}^{(H)}$ is supported on F_H. Hence, $\mathbf{L}^{(i)}_{F_i,F_i}{}^{-1} = \sum_{H \in \mathcal{T}(i)} \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1}$.

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 18 to get

$$\begin{aligned} \mathbf{Sc}(\mathbf{L}, C_{i-1}) &= \sum_{H \in \mathcal{T}(i)} \mathbf{Sc}(\mathbf{L}[H], C_{i-1} \cap V(H)) \\ &\approx_{\epsilon_{\mathbf{P}}} \sum_{H \in \mathcal{T}(i)} \widetilde{\mathbf{Sc}}(\mathbf{L}[H], \partial H \cup F_H) = \sum_{H \in \mathcal{T}(i)} \mathbf{L}^{(H)} = \mathbf{L}^{(i)}. \end{aligned} \tag{4.6}$$

Now, we prove inductively that

$$\mathbf{L}^{-1} \approx_{i\epsilon_{\mathbf{P}}} \mathbf{\Pi}^{(0)\top} \cdots \mathbf{\Pi}^{(i-1)\top} \begin{bmatrix} \left(\mathbf{L}^{(0)}_{F_0,F_0}\right)^{-1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \left(\mathbf{L}^{(i-1)}_{F_{i-1},F_{i-1}}\right)^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \left(\mathbf{L}^{(i)}\right)^{-1} \end{bmatrix} \mathbf{\Pi}^{(i-1)} \cdots \mathbf{\Pi}^{(0)}, \tag{4.7}$$

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

For general i, we factor $\mathbf{L}^{(i)}$ in Eq. (4.7) 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,

$$\mathbf{L}^{(i)} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{L}^{(i)}_{C_i,F_i}(\mathbf{L}^{(i)}_{F_i,F_i})^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{L}^{(i)}_{F_i,F_i} & \mathbf{0} \\ \mathbf{0} & \mathbf{Sc}(\mathbf{L}^{(i)}, C_i) \end{bmatrix} \begin{bmatrix} \mathbf{I} & (\mathbf{L}^{(i)}_{F_i,F_i})^{-1}\mathbf{L}^{(i)}_{F_i,C_i} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}. \tag{4.8}$$

For the Schur complement term in the factorization, we have

$$\begin{aligned} \mathbf{Sc}(\mathbf{L}^{(i)}, C_i) &\approx_{i\epsilon_{\mathbf{P}}} \mathbf{Sc}(\mathbf{Sc}(\mathbf{L}, C_{i-1}), C_i) && \text{(by Eq. (4.6))} \\ &= \mathbf{Sc}(\mathbf{L}, C_i) && \text{(by transitivity of Schur complements)} \\ &\approx_{\epsilon_{\mathbf{P}}} \mathbf{L}^{(i)}. && \text{(by Eq. (4.6))} \end{aligned}$$

So we can use $\mathbf{L}^{(i)}$ in place of the Schur complement term, and the equality becomes an approximation with factor $(i+1)\epsilon_{\mathbf{P}}$. Furthermore, in Eq. (4.8), we can rewrite

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

Plugging the inverse of Eq. (4.8) into Eq. (4.7), we get the correct recursive approximation.

Finally, we note that at the η-th level, $\mathbf{L}^{(\eta)}_{F_\eta,F_\eta} = \mathbf{L}^{(\eta)}$ since $C_\eta = \emptyset$. So we have the overall expression. ∎

4.4 Recursive Schur complements on separator tree

In this section, we prove Theorem 6 which maintains approximate Schur complements onto the boundary vertices of each node H in $\mathcal{T}$.

We use the following result as a black-box for computing sparse approximate Schur complements:

Lemma 34 (ApproxSchur procedure [DKP+17]). *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). We say ApproxSchurNode(H) runs 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,
- $\mathbf{L}^{(H)} \approx_{(i-1)\delta} \mathbf{Sc}(\mathbf{L}[H], \partial H \cup F_H)$,
- $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H) \approx_{i\delta} \mathbf{Sc}(\mathbf{L}[H], \partial H)$, and the graph is on ∂H with $\widetilde{O}(\delta^{-2}|\partial H|)$ edges.

Lemma 35. *Suppose $\mathbf{L}^{(D)}$ and $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D)}, \partial D)$ are computed 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 34.

Otherwise, suppose H is at level i with children D_1 and D_2. By construction of the separator tree and Observation 23, 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

$$\begin{aligned}
\mathbf{L}^{(H)} &= \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)}, \partial D_1) + \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)}, \partial D_2) \\
&\approx_{(i-1)\delta} \mathbf{Sc}(\mathbf{L}[D_1], \partial D_1) + \mathbf{Sc}(\mathbf{L}[D_2], \partial D_2) \\
&= \mathbf{Sc}(\mathbf{L}[D_1], (\partial H \cup F_H) \cap V(D_1)) + \mathbf{Sc}(\mathbf{L}[D_2], (\partial H \cup F_H) \cap V(D_2)) \\
&\quad \text{(by construction of the separator tree, } \partial D_j = (\partial H \cup F_H) \cap V(D_j) \text{ for } j = 1, 2) \\
&= \mathbf{Sc}(\mathbf{L}[H], \partial H \cup F_H), \qquad \text{(by Lemma 18)}
\end{aligned}$$

so we have the second correctness property.

Line 43 returns $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ with $\widetilde{O}(\delta^{-2}|\partial H|)$ edges by Lemma 34. Also,

$$\begin{aligned}
\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H) &\approx_{\delta} \mathbf{Sc}(\mathbf{L}^{(H)}, \partial H) \\
&\approx_{(i-1)\delta} \mathbf{Sc}(\mathbf{Sc}(\mathbf{L}[H], \partial H \cup F_H), \partial H) \\
&= \mathbf{Sc}(\mathbf{L}[H], \partial H), \qquad \text{(by Lemma 16)}
\end{aligned}$$

Algorithm 3 Data structure to maintain dynamic approximate Schur complements

data structure DynamicSC
private: member
  Graph G with incidence matrix $\mathbf{B}$
  $\boldsymbol{w} \in \mathbb{R}^m$, $\mathbf{W} \in \mathbb{R}^{m \times m}$: Weight vector and diagonal weight matrix, used interchangeably
  $\epsilon_{\mathbf{P}} > 0$: Overall approximation factor
  $\delta > 0$: Fast Schur complement approximation factor
  $\mathcal{T}$: Separator tree of height η. Every node H of $\mathcal{T}$ stores:
    F_H, ∂H: Interior and boundary vertices of region H
    $\mathbf{L}^{(H)} \in \mathbb{R}^{m \times m}$: Laplacian supported on $F_H \cup \partial H$
    $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H) \in \mathbb{R}^{m \times m}$: δ-approximate Schur complement of $\mathbf{L}^{(H)}$

procedure Initialize(G, $\boldsymbol{w} \in \mathbb{R}^m$, $\epsilon_{\mathbf{P}} > 0$)
  $\mathbf{B} \leftarrow$ incidence matrix of G
  $\mathcal{T} \leftarrow$ separator tree of G of height η constructed by Theorem 25
  $\delta \leftarrow \epsilon_{\mathbf{P}}/(\eta + 1)$
  $\boldsymbol{w} \leftarrow \boldsymbol{w}$
  for $i = 0, \ldots, \eta$ **do**
    for each node H at level i in $\mathcal{T}$ **do**
      ApproxSchurNode(H)
    end for
  end for
end procedure

procedure Reweight($\boldsymbol{w}^{(\text{new})} \in \mathbb{R}^m$)
  $\mathcal{H} \leftarrow$ set of leaf nodes in $\mathcal{T}$ that contain each edge e whose weight is updated
  $\boldsymbol{w} \leftarrow \boldsymbol{w}^{(\text{new})}$
  $\mathcal{P}_{\mathcal{T}}(\mathcal{H}) \leftarrow$ set of all ancestor nodes of $\mathcal{H}$ in $\mathcal{T}$ and $\mathcal{H}$
  for $i = 0, \ldots, \eta$ **do**
    for each node H at level i in $\mathcal{P}_{\mathcal{T}}(\mathcal{H})$ **do**
      ApproxSchurNode(H)
    end for
  end for
end procedure

procedure ApproxSchurNode($H \in \mathcal{T}$)
  if H is a leaf node **then**
    ▷ $\mathbf{B}[H]$ is the incidence matrix for the induced subgraph H with edge set $E(H)$
    $\mathbf{L}^{(H)} \leftarrow (\mathbf{B}[H])^\top \mathbf{W}_{E(H)} \mathbf{B}[H]$
    $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H) \leftarrow$ ApproxSchur($\mathbf{L}^{(H)}, \partial H, \delta$) ▷ Lemma 34
  else
    Let D_1, D_2 be the children of H
    $\mathbf{L}^{(H)} \leftarrow \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)}, \partial D_1) + \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)}, \partial D_2)$
    $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H) \leftarrow$ ApproxSchur($\mathbf{L}^{(H)}, \partial H, \delta$)
  end if
end procedure

giving us the third correctness property. ☐

Lemma 36. *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 34.

Otherwise, when H has children D_1, D_2, computing $\mathbf{L}^{(H)}$ requires accessing $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_j)}, \partial D_j)$ for $j = 1, 2$ and summing them together, in time $\widetilde{O}(|\partial D_1| + |\partial D_2|) = \widetilde{O}(|\partial H \cup F_H|)$. Then, computing $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ take $\widetilde{O}(\delta^{-2}|\partial H \cup F_H|)$ by Lemma 34. ☐

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

Theorem 6 (Schur complements maintenance)**.** *Given a modified planar graph G with m edges and its separator tree $\mathcal{T}$ with height $\eta = O(\log m)$, the deterministic data structure* DynamicSC *(Algorithm 3) maintains the edge weights $\boldsymbol{w}$ from the IPM, and at every node $H \in \mathcal{T}$, maintains two vertex sets F_H and ∂H, and two Laplacians $\mathbf{L}^{(H)}$ and $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H \cup F_H)$ dependent on $\boldsymbol{w}$. It supports the following procedures:*

- Initialize$(G, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \epsilon_{\mathbf{P}} > 0)$*: Given a graph G, initial weights $\boldsymbol{w}$, projection matrix approximation accuracy $\epsilon_{\mathbf{P}}$, preprocess in $\widetilde{O}({\epsilon_{\mathbf{P}}}^{-2}m)$ time.*
- Reweight$(\boldsymbol{w} \in \mathbb{R}^m_{>0}$*, given implicitly as a set of changed coordinates*$)$*: Update the weights to $\boldsymbol{w}$, and update the relevant Schur complements in $\widetilde{O}({\epsilon_{\mathbf{P}}}^{-2}\sqrt{mK})$ time, where K is the number of coordinates changed in $\boldsymbol{w}$.*

  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 time $\widetilde{O}\left({\epsilon_{\mathbf{P}}}^{-2}|\partial H \cup F_H|\right)$.*
- *Access to Laplacian $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ at any node $H \in \mathcal{T}$ in time $\widetilde{O}\left({\epsilon_{\mathbf{P}}}^{-2}|\partial H|\right)$.*

Furthermore, the $\mathbf{L}^{(H)}$'s maintained by the data structure satisfy

$$\mathbf{L}^{(H)} \approx_{\epsilon_{\mathbf{P}}} \mathbf{Sc}(\mathbf{L}[H], \partial H \cup F_H), \tag{2.9}$$

for all $H \in \mathcal{T}$ with high probability. The $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$'s maintained satisfy

$$\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H) \approx_{\epsilon_{\mathbf{P}}} \mathbf{Sc}(\mathbf{L}[H], \partial H) \tag{2.10}$$

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 35, 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 Theorem 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 35. The runtime of Initialize is bounded by running ApproxSchurNode on each node, i.e:

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

Where we bound the sum using Lemma 28 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 28, 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.

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 $\boldsymbol{x}\stackrel{\text{def}}{=}\boldsymbol{y}+\mathbf{M}\boldsymbol{z}$. 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 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 $\mathbf{L}^{-1}$. MaintainRep will contain a copy of the DynamicSC data structure; therefore, the remainder of this section will freely refer to $\widetilde{\mathbf{\Gamma}}$ and $\mathbf{\Pi}^{(0)},\cdots,\mathbf{\Pi}^{(\eta-1)}$.

5.1 Maintaining the intermediate vector $\boldsymbol{z}$

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

Definition 37 ($\boldsymbol{z}^{(k)}$)**.** At the k-th step of the IPM, let $\boldsymbol{v}^{(k)}$ be the step direction. Let $\boldsymbol{d}\stackrel{\text{def}}{=}\mathbf{B}^{\top}\mathbf{W}^{1/2}\boldsymbol{v}^{(k)}$. Define $\boldsymbol{z}^{(k)}$ to be the partial computation

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

Observe that this is a partial projection: If we apply $\mathbf{W}^{1/2}\mathbf{B}\mathbf{\Pi}^{(0)\top}\cdots\mathbf{\Pi}^{(\eta-1)\top}$ to $\boldsymbol{z}^{(k)}$, then by Theorem 33, the result is an approximation to $\mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}^{(k)}$.

We first show how to multiply $\widetilde{\mathbf{\Gamma}}\mathbf{\Pi}^{(\eta-1)}\cdots\mathbf{\Pi}^{(0)}$ 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 $\boldsymbol{d}$ 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.

Algorithm 4 Data structure to maintain the intermediate vector $\boldsymbol{z}$, Part 1

data structure MAINTAINZ
private: member
  G: input graph G with incidence matrix $\mathbf{B}$
  $\mathcal{T}$: separator tree of G of height η
  $c \in \mathbb{R}, \boldsymbol{z}^{(\text{step})}, \boldsymbol{z}^{(\text{sum})} \in \mathbb{R}^n$: coefficient and vectors to be maintained
  $\boldsymbol{u} \in \mathbb{R}^n$: vector to be maintained such that $\boldsymbol{u} = \mathbf{\Pi}^{(\eta-1)}\cdots\mathbf{\Pi}^{(0)}\mathbf{B}^\top\mathbf{W}\boldsymbol{v}$
  $\boldsymbol{v} \in \mathbb{R}^m$: direction vector from the current iteration
  $\boldsymbol{w} \in \mathbb{R}^m$: weight vector ▷ we sometimes also use $\mathbf{W} \stackrel{\text{def}}{=} \text{diag}(\boldsymbol{w})$
  `dynamicSC`: an instance of DYNAMICSC struct ▷ gives read access to $\mathbf{L}^{(H)}$ for $H \in \mathcal{T}$

procedure INITIALIZE($G, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \epsilon_{\mathbf{P}} > 0$)
  $\boldsymbol{w} \leftarrow \boldsymbol{w}$, $\boldsymbol{v} \leftarrow \boldsymbol{v}$
  `dynamicSC`.INITIALIZE($G, \boldsymbol{w}, \epsilon_{\mathbf{P}}$)
  $\boldsymbol{u} \leftarrow$ PARTIALPROJECT($\mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}$)
  $\boldsymbol{z}^{(\text{step})} \leftarrow \widetilde{\mathbf{\Gamma}}\boldsymbol{u}$
  $\boldsymbol{z}^{(\text{sum})} \leftarrow \mathbf{0}$
  $c \leftarrow 0$
end procedure

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

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

Lemma 38. *Given a vector $\boldsymbol{d} \in \mathbb{R}^n$, let $\mathcal{H} \supseteq \{H \in \mathcal{T} : \boldsymbol{d}|_{F_H} \neq \mathbf{0}\}$ and suppose $|\mathcal{H}| = K$. Then the procedure* PARTIALPROJECT($\boldsymbol{d}, \mathcal{H}$) *in the* MAINTAINZ *data structure (Algorithm 4) returns the*

vector

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

where the $\boldsymbol{\Pi}^{(i)}$*'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 24. 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 an exact 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 28.

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

Claim 39. *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 23) for* $0 \le i < \eta$. *Then*

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

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 $\boldsymbol{\Pi}^{(i)}$,

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

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 $\mathcal{T}(i+1) \cap \mathcal{P}_{\mathcal{T}}(\mathcal{H}) \stackrel{\text{def}}{=} \mathcal{P}_{\mathcal{T}}(\mathcal{H}, i+1)$, 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 23. 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 $\boldsymbol{u} = \boldsymbol{\Pi}^{(\eta-1)} \cdots \boldsymbol{\Pi}^{(1)}\boldsymbol{\Pi}^{(0)}\boldsymbol{d}$.

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 40. *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*

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

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 39, 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:

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

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

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

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

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

Then by commutativity as in Lemma 40, we have

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

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.

Algorithm 4 Data structure to maintain the intermediate vector $\boldsymbol{z}$, Part 2

procedure REWEIGHT($\boldsymbol{w}^{(\mathrm{new})} \in \mathbb{R}^m_{>0}$)
  $\boldsymbol{w} \leftarrow \boldsymbol{w}^{(\mathrm{new})}$
  $\mathcal{H} \leftarrow$ set of leaf nodes in $\mathcal{T}$ that contain all the edges of G whose weight has changed
  $\Delta\boldsymbol{u} \leftarrow \textsc{PartialProject}(\mathbf{B}^\top(\mathbf{W}^{(\mathrm{new})1/2} - \mathbf{W}^{1/2})\boldsymbol{v})$
  $\boldsymbol{u} \leftarrow \boldsymbol{u} + \Delta\boldsymbol{u}$
  $\boldsymbol{d} \leftarrow \textsc{InversePartialProject}(\boldsymbol{u}, \mathcal{H})$ ▷ revert projection with old weights
  `dynamicSC`.REWEIGHT($\boldsymbol{w}^{(\mathrm{new})}$) ▷ update $\mathbf{L}^{(H)}$'s to use the new weights
  ▷ specifically, $\mathbf{L}^{(H)}$ changes for each $H \in \mathcal{P}_\mathcal{T}(\mathcal{H})$
  $\boldsymbol{u} \leftarrow \textsc{PartialProject}(\boldsymbol{d}, \mathcal{H})$ ▷ apply projection with new weights
  $\boldsymbol{y} \leftarrow \boldsymbol{z}^{(\mathrm{step})}$ ▷ backup copy of $\boldsymbol{z}^{(\mathrm{step})}$
  for H in $\mathcal{P}_\mathcal{T}(\mathcal{H})$ **do**
    $\boldsymbol{z}^{(\mathrm{step})}|_{F_H} \leftarrow (\mathbf{L}^{(H)}_{F_H,F_H})^{-1}\boldsymbol{u}|_{F_H}$
  end for
  $\boldsymbol{z}^{(\mathrm{sum})} \leftarrow \boldsymbol{z}^{(\mathrm{sum})} - c \cdot (\boldsymbol{z}^{(\mathrm{step})} - \boldsymbol{y})$ ▷ update $\boldsymbol{z}^{(\mathrm{sum})}$ to maintain the invariant
end procedure

procedure MOVE($\alpha \in \mathbb{R}, \boldsymbol{v}^{(\mathrm{new})} \in \mathbb{R}^m$)
  $\Delta\boldsymbol{v} \leftarrow \boldsymbol{v}^{(\mathrm{new})} - \boldsymbol{v}$
  $\boldsymbol{v} \leftarrow \boldsymbol{v}^{(\mathrm{new})}$
  $\Delta\boldsymbol{u} \leftarrow \textsc{PartialProject}(\mathbf{B}^\top\mathbf{W}^{1/2}\Delta\boldsymbol{v})$
  $\boldsymbol{u} \leftarrow \boldsymbol{u} + \Delta\boldsymbol{u}$
  $\boldsymbol{y} \leftarrow \boldsymbol{z}^{(\mathrm{step})}$ ▷ backup copy of $\boldsymbol{z}^{(\mathrm{step})}$
  for H in $\mathcal{P}_\mathcal{T}(\mathcal{H})$ **do**
    $\boldsymbol{z}^{(\mathrm{step})}|_{F_H} \leftarrow (\mathbf{L}^{(H)}_{F_H,F_H})^{-1}\boldsymbol{u}|_{F_H}$
  end for
  $\boldsymbol{z}^{(\mathrm{sum})} \leftarrow \boldsymbol{z}^{(\mathrm{sum})} - c \cdot (\boldsymbol{z}^{(\mathrm{step})} - \boldsymbol{y})$
  $c \leftarrow c + \alpha$
end procedure

Theorem 42 (Maintain intermediate vector $\boldsymbol{z}$)**.** *Given a modified planar graph G with n vertices and m edges and its separator tree $\mathcal{T}$ 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(G, *separator tree* $\mathcal{T}, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \epsilon_{\mathbf{P}} > 0$)*: Given a graph G, its separator tree $\mathcal{T}$, initial step direction $\boldsymbol{v}$, initial weights $\boldsymbol{w}$, and target projection matrix 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 Eq. (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 $\boldsymbol{d} \stackrel{\text{def}}{=} \mathbf{B}^\top \mathbf{W}^{1/2}\boldsymbol{v}$ 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 $\boldsymbol{u} = \boldsymbol{\Pi}^{(\eta-1)} \cdots \boldsymbol{\Pi}^{(0)}\mathbf{B}^\top \mathbf{W}^{1/2}\boldsymbol{v}$. We now prove the correctness and runtime of each procedure separately.

Initialize: By the guarantee of Lemma 38, at the end of Initialize, we have

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

and

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

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 $\widetilde{\boldsymbol{\Gamma}}\boldsymbol{u}$, 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 $|\mathbf{L}^{(H)}| = \widetilde{O}\left({\epsilon_{\mathbf{P}}}^{-2}|F_H \cup \partial H|\right)$ from Theorem 6 and by Lemma 28.

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 56, we have

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

where the second equality follows from the guarantee of PartialProject and the guarantee from the previous IPM step. By Lemma 38, $\boldsymbol{u}'$ is updated only on F_H where $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$. Thus, to update $\boldsymbol{z}^{(\text{step})'} = \widetilde{\boldsymbol{\Gamma}}\boldsymbol{u}'$, we only need to update $\boldsymbol{z}^{(\text{step})'}|_{F_H}$ for $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{H})$, which happens on Line 59. Observe that the update in value to $\boldsymbol{z}^{(\text{step})}$ is cancelled out by the update in $\boldsymbol{z}^{(\text{sum})}$ at Line 61, so that the value of $\boldsymbol{z}$ does not change overall up to that point. But we have

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

Then in Line 62, 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 38. Line 59 takes $\widetilde{O}({\epsilon_{\mathbf{P}}}^{-2}\sqrt{mK})$ time in total by Theorem 6 and Lemma 28. 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 $\widetilde{\boldsymbol{\Gamma}}$ and $\boldsymbol{\Pi}^{(i)}$ denote these matrices defined using the old weights, and let $\widetilde{\boldsymbol{\Gamma}}'$ and $\boldsymbol{\Pi}^{(i)'}$ 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,

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

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 $\boldsymbol{z}^{(\text{step})}$ during the procedure, and cancel all the changes to $\boldsymbol{z}^{(\text{step})}$ by updating $\boldsymbol{z}^{(\text{sum})}$ appropriately at the last line (Line 49).

Immediately before Line 40, the algorithm invariant guarantees

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

By Lemma 38,

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

Therefore, after executing Line 40, we have

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

Next, we need to update $\boldsymbol{u}$ to reflect the changes to $\widetilde{\boldsymbol{\Gamma}}, \boldsymbol{\Pi}^{(i)}$. Updating these matrices is done via `dynamicSC`. However, calling PartialProject($\mathbf{B}^\top \mathbf{W}^{(\text{new})1/2} \boldsymbol{v}$) afterwards 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 $\widetilde{\boldsymbol{\Gamma}}\boldsymbol{\Pi}^{(\eta-1)}\cdots\boldsymbol{\Pi}^{(0)}$. 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 40, after executing Line 40, we have

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

Let $r=|\mathcal{P}_{\mathcal{T}}(\mathcal{H})|$. Then $\textsc{InversePartialProject}(\boldsymbol{u},\mathcal{H})$ on Line 41 returns $\boldsymbol{d}$ by Lemma 41 satisfying

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

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

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

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

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

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 $\textsc{PartialProject}^{(\text{new})}$ to emphasize that it runs with new weights. This gives

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

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 47 on $H\in\mathcal{P}_{\mathcal{T}}(\mathcal{H})$ correctly sets $\boldsymbol{z}^{(\text{step})'}=\widetilde{\boldsymbol{\Gamma}}'\boldsymbol{u}'$.

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 47) takes $\widetilde{O}({\epsilon_{\mathbf{P}}}^{-2}\sqrt{mK})$ time in total by Theorem 6 and Lemma 28. 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 $\boldsymbol{u}\leftarrow\textsc{PartialProject}(\boldsymbol{d},\mathcal{H})$, 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 43 (Tree operator)**.** Suppose $\mathcal{T}$ is a rooted tree with constant degree. 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 $V \stackrel{\text{def}}{=} V(G) = \bigcup_{H \in \mathcal{T}} V(H)$, where G is the root node of $\mathcal{T}$.

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 a edge or leaf operator is not given, we assume it to be $\mathbf{0}$.

For a path $H_t \to H_1 \stackrel{\text{def}}{=} (H_t, \ldots, H_1)$, 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 44. 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 45 ($\mathbf{M}^{(H)}, \overline{\mathbf{M}^{(H)}}$)**.** For notational convenience, define $\mathcal{T}_H$ to be the subtree of $\mathcal{T}$ rooted at H.

We define the subtree operator $\mathbf{M}^{(H)} : V(H) \mapsto E(H)$ at each 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 based on the definitions without proof.

Corollary 46. *For any node $H \in \mathcal{T}$,*

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

where G is the root node of $\mathcal{T}$.

Furthermore, if H has with 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 define the complexity of a tree operator to be parameterized by the number of tree edges.

Definition 47 (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.

Algorithm 5 Compute $\mathbf{M}\boldsymbol{z}$ for a tree operator $\mathbf{M}$

procedure ComputeMz($\mathbf{M}, \boldsymbol{z}$)
  $\mathcal{H} \leftarrow$ set of all nodes H in $\mathcal{T}$ such that $\mathbf{M}_{(H,P)}$ or $\mathbf{J}_H$ is nonzero
  $\mathcal{P}_\mathcal{T}(\mathcal{H}) \leftarrow$ set of $\mathcal{H}$ and all ancestor nodes of $\mathcal{H}$ in $\mathcal{T}$
  $\boldsymbol{v}_H \leftarrow \mathbf{0}$ for each $H \in \mathcal{T}$ ▷ sparse vectors for intermediate computations
  for each node $H \in \mathcal{P}_\mathcal{T}(\mathcal{H})$ **do**
    $\boldsymbol{v}_H \leftarrow \mathbf{I}_{F_H} \boldsymbol{z} = \boldsymbol{z}|_{F_H}$ ▷ apply the $\mathbf{I}_{F_H}$ part of the operator
  end for
  for each node $H \in \mathcal{P}_\mathcal{T}(\mathcal{H})$ by decreasing level **do**
    Let P be the parent of H
    $\boldsymbol{v}_H \leftarrow \boldsymbol{v}_H + \mathbf{M}_{(H,P)} \boldsymbol{v}_P$ ▷ apply $\mathbf{M}_{(H,P)}$ as we move from P to H
  end for
  for each leaf node $H \in \mathcal{P}_\mathcal{T}(\mathcal{H})$ **do**
    $\boldsymbol{x}|_{E(H)} \leftarrow \mathbf{J}_H \boldsymbol{v}_H$ ▷ apply the leaf operator
  end for
  return $\boldsymbol{x}$
end procedure

Corollary 48. *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. □

5.3 Proof of Theorem 7

Finally, we give the data structure for maintaining an implicit representation of the form $\boldsymbol{y} + \mathbf{M}\boldsymbol{z}$ 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. Note that this data structure *extends* MaintainZ in the object-oriented programming sense.

Theorem 7 (Implicit representation maintenance)**.** *Given a modified planar graph G with n vertices and m edges, and its separator tree $\mathcal{T}$ 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 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,*

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

  where $\alpha^{(i)}$ is the step size α given in Move *for step i,*
- *$\boldsymbol{z}^{(\text{step})}$ satisfies $\boldsymbol{z}^{(\text{step})} = \widetilde{\boldsymbol{\Gamma}}\boldsymbol{\Pi}^{(\eta-1)} \cdots \boldsymbol{\Pi}^{(0)} \mathbf{B}^\top \mathbf{W}^{1/2} \boldsymbol{v}$,*
- *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$(G, \mathcal{T}, \mathbf{M}, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \boldsymbol{x}^{(\text{init})} \in \mathbb{R}^m, \epsilon_{\mathbf{P}} > 0)$*: Given a graph G, its separator tree $\mathcal{T}$, 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 projection matrix 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($\boldsymbol{w} \in \mathbb{R}^m_{>0}$ *given implicitly as a set of changed coordinates): Update the weights to* $\boldsymbol{w}$*. 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 the number of coordinates changed in* $\boldsymbol{w}$ *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($\alpha \in \mathbb{R}$, $\boldsymbol{v} \in \mathbb{R}^n$ *given implicitly as a set of changed coordinates): Update the current direction to* $\boldsymbol{v}$*, and then* $\boldsymbol{z}^{(\text{step})}$ *to maintain the claimed invariant. 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, where* K *is the number of coordinates changed in* $\boldsymbol{v}$ *compared to the previous IPM step.*

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

Proof. 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 12 sets $\boldsymbol{y} \leftarrow \boldsymbol{x}^{(\text{init})}$, and `Super`.INITIALIZE sets $\boldsymbol{z} \leftarrow \mathbf{0}$. So we have $\boldsymbol{x} = \boldsymbol{y} + \mathbf{M}\boldsymbol{z}$ at the end of initialization. Furthermore, the initialization of $\boldsymbol{z}$ correctly sets $\boldsymbol{z}^{(\text{step})}$ in terms of $\boldsymbol{v}$.

By Theorem 42, `Super`.INITIALIZE takes $\widetilde{O}({\epsilon_{\mathbf{P}}}^{-2}m)$ time. Storing the implicit representation of $\mathbf{M}$ takes $O(m)$ time.

REWEIGHT: By Theorem 42, `Super`.REWEIGHT updates its current weight and `DynamicSC`, and updates $\boldsymbol{z}^{(\text{step})}$ correspondingly to maintain the invariant, while not changing the value of $\boldsymbol{z}$. Because $\mathbf{M}$ is stored by instructions, no explicit update to $\mathbf{M}$ is required. Line 20 updates $\boldsymbol{y}$ to zero out the changes to $\mathbf{M}\boldsymbol{z}$.

The instructions for computing $\Delta\mathbf{M}$ require the Laplacians from `DynamicSC` before and after the update in Line 17. For this, we monitor the updates of `dynamicSC` and stores 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 `Super`.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($\Delta\mathbf{M}, \boldsymbol{z}$) takes $O(T(K))$ time. Thurs, the total runtime is $\widetilde{O}({\epsilon_{\mathbf{P}}}^{-2}\sqrt{mK} + T(m))$.

Algorithm 6 Implicit representation maintenance

data structure MAINTAINREP **extends** MAINTAINZ
private: member
    $\mathcal{T}$: separator tree
    $\boldsymbol{y} \in \mathbb{R}^m$: offset vector
    $\mathbf{M}$: *instructions to compute the tree operator* $\mathbf{M} \in \mathbb{R}^{m \times n}$
▷ $\boldsymbol{z} = c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})}$ maintained by MAINTAINZ, accessable in this data structure
▷ `DynamicSC`: an accessable instance of DYNAMICSC maintained by MAINTAINZ

procedure INITIALIZE($G, \mathcal{T}, \mathbf{M}, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \boldsymbol{x}^{(\text{init})} \in \mathbb{R}^m, \epsilon_{\mathbf{P}} > 0$)
    $\mathbf{M} \leftarrow \mathbf{M}$ ▷ initialize the instructions to compute $\mathbf{M}$
    `Super`.INITIALIZE($G, \mathcal{T}, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \epsilon_{\mathbf{P}} > 0$) ▷ initialize $\boldsymbol{z}$
    $\boldsymbol{y} \leftarrow \boldsymbol{x}^{(\text{init})}$
end procedure

procedure REWEIGHT($\boldsymbol{w}^{(\text{new})}$)
    Let $\mathbf{M}^{(\text{old})}$ represent the current tree operator $\mathbf{M}$
    `Super`.REWEIGHT($\boldsymbol{w}^{(\text{new})}$) ▷ update representation of $\boldsymbol{z}$ and `DynamicSC`
    ▷ $\mathbf{M}$ is updated as a result of reweight in `DynamicSC`
    $\Delta\mathbf{M} \leftarrow \mathbf{M} - \mathbf{M}^{(\text{old})}$ ▷ $\Delta\mathbf{M}$ is represented implicitly
    $\boldsymbol{y} \leftarrow \boldsymbol{y} -$ COMPUTEMZ($\Delta\mathbf{M}, c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})}$) ▷ Algorithm 5
end procedure

procedure MOVE($\alpha, \boldsymbol{v}^{(\text{new})}$)
    `Super`.MOVE($\alpha, \boldsymbol{v}^{(\text{new})}$)
end procedure

procedure EXACT()
    return $\boldsymbol{y} +$ COMPUTEMZ($\mathbf{M}, c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})}$) ▷ Algorithm 5
end procedure

Move: The runtime and correctness follow from Theorem 42.

Exact: ComputeMz computes $\mathbf{M}\boldsymbol{z}$ correctly in $O(T(m))$ time by Corollary 48. 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 $\overline{\boldsymbol{s}}, \overline{\boldsymbol{f}}$ so that

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

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,

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

where $\mathbf{D}$ is a dynamic diagonal scaling matrix. (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 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 k-th step of the IPM; that is, $\mathbf{D}^{(k)}$ and $\boldsymbol{x}^{(k)}$ are $\mathbf{D}$ and $\boldsymbol{x}$ at the end of the k-th step. Step 0 is the state of the data structure immediately after initialization.

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.

Here, we make use of dyadic intervals, and 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 49. At step k of the IPM, for each ℓ such that $k = 0 \bmod 2^\ell$, we define

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

We say that $\overline{\boldsymbol{x}}_i$ has not been updated since the $(k-2^\ell)$-th step if $\overline{\boldsymbol{x}}_i^{(j)} = \overline{\boldsymbol{x}}_i$ and $\mathbf{D}_{ii}^{(j)} = \mathbf{D}_{ii}^{(k-2^\ell)}$ for $j \geq k - 2^\ell$, i.e. $\overline{\boldsymbol{x}}_i$ was not updated by Line 20 or Line 29 in the $(k-2^\ell+1), \ldots, (i-1)$-th steps.

Algorithm 7 Data structure AbstractMaintainApprox, Part 1

data structure AbstractMaintainApprox
private : member
    $\mathcal{T}$: constant-degree rooted tree with height η and m leaves ▷ leaf i corresponds to $\boldsymbol{x}_i$
    $w \stackrel{\text{def}}{=} \Theta(\eta^2 \log(\frac{m}{\rho}))$: sketch dimension
    $\boldsymbol{\Phi} \sim \mathbf{N}(0, \frac{1}{w})^{w \times m}$: JL-sketch matrix
    $\delta > 0$: additive approximation error
    k: current IPM step
    $\overline{\boldsymbol{x}} \in \mathbb{R}^m$: current valid approximate vector
    $\{\boldsymbol{x}^{(j)} \in \mathbb{R}^m\}_{j=0}^k$: list of previous inputs
    $\{\mathbf{D}^{(j)} \in \mathbb{R}^{m \times m}\}_{j=0}^k$: list of previous diagonal scaling matrices

procedure Initialize($\mathcal{T}, \boldsymbol{x} \in \mathbb{R}^m, \mathbf{D} \in \mathbb{R}_{>0}^{m \times m}, \rho > 0, \delta > 0$)
    $\mathcal{T} \leftarrow \mathcal{T}$, $\delta \leftarrow \delta$, $k \leftarrow 0$
    $\overline{\boldsymbol{x}} \leftarrow \boldsymbol{x}, \boldsymbol{x}^{(0)} \leftarrow \boldsymbol{x}, \mathbf{D}^{(0)} \leftarrow \mathbf{D}$
    sample $\boldsymbol{\Phi} \sim \mathbf{N}(0, \frac{1}{w})^{w \times m}$
end procedure

procedure Approximate($\boldsymbol{x}^{(\text{new})} \in \mathbb{R}^m, \mathbf{D}^{(\text{new})} \in \mathbb{R}_{>0}^{m \times m}$)
    $k \leftarrow k + 1$, $\boldsymbol{x}^{(k)} \leftarrow \boldsymbol{x}^{(\text{new})}$, $\mathbf{D}^{(k)} \leftarrow \mathbf{D}^{(\text{new})}$
    $\overline{\boldsymbol{x}}_i \leftarrow \boldsymbol{x}_i^{(k-1)}$ for all i such that $\mathbf{D}_{ii}^{(k)} \neq \mathbf{D}_{ii}^{(k-1)}$
    $I \leftarrow \emptyset$
    for all $0 \leq \ell < \lceil \log m \rceil$ such that $k \equiv 0 \bmod 2^\ell$ **do**
        $I_\ell^{(k)} \leftarrow$ FindLargeCoordinates(ℓ)
        $I \leftarrow I \cup I_\ell^{(k)}$
    end for
    if $k = 0 \bmod 2^{\lceil \log m \rceil}$ **then**
        $I \leftarrow [m]$ ▷ Update $\overline{\boldsymbol{x}}$ in full every $2^{\lceil \log m \rceil}$ steps
    end if
    $\overline{\boldsymbol{x}}_i \leftarrow \boldsymbol{x}_i^{(k)}$ for all $i \in I$
    return $\overline{\boldsymbol{x}}$
end procedure

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

- Initialize($\mathcal{T}, \boldsymbol{x} \in \mathbb{R}^m$, $\mathbf{D} \in \mathbb{R}^{m\times m}_{>0}$, $\rho > 0$, $\delta > 0$)*: Initialize the data structure at step 0 with tree $\mathcal{T}$, initial vector $\boldsymbol{x}$, initial diagonal scaling matrix $\mathbf{D}$, target additive approximation error δ, and success probability $1-\rho$.*
- Approximate($\boldsymbol{x}^{(\text{new})} \in \mathbb{R}^m$, $\mathbf{D}^{(\text{new})} \in \mathbb{R}^{m\times m}_{>0}$)*: Increment the step counter and update vector $\boldsymbol{x}$ and diagonal scaling matrix $\mathbf{D}$. 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 $\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\|_{\mathbf{D}^{(k)}} \le \beta$ for all k, then at the k-th step, the data structure first updates $\overline{\boldsymbol{x}}_i \leftarrow \boldsymbol{x}_i^{(k-1)}$ for the coordinates i with $\mathbf{D}_{ii}^{(k)} \neq \mathbf{D}_{ii}^{(k-1)}$, then 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 51. 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 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 50. 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 $\overline{\boldsymbol{x}}_i \leftarrow \boldsymbol{x}_i^{(k')}$. This may happen in Line 29 at step k' or in Line 20 at step $k'+1$. In both case, we have that $\mathbf{D}_{ii}^{(d)}$ is the same for all $k \ge d > k'$ and that i is not in the set $I_\ell^{(d)}$ returned by FindLargeCoordinates for all $k \ge d > k'$. (In the former case, we further have $\mathbf{D}_{ii}^{(k'+1)} = \mathbf{D}_{ii}^{(k')}$ but this is not required in the proof.) 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, $k_{j+1} - k_j$ divides k_{j+1}, and $|s| \le 2\lceil \log m \rceil$. Hence, we have that

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

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

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

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

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

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

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 $\mathbf{D}_{ii}^{(j)} = \mathbf{D}_{ii}^{(k)}$ 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

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

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

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

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

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

where we use $\|\boldsymbol{x}^{(j+1)} - \boldsymbol{x}^{(j)}\|_{\mathbf{D}^{(j+1)}} \leq \beta$ 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 k-th step, summing over all ℓ with $k = 0 \bmod 2^\ell$, we have that the total number of coordinates changed, excluding those induced by a change in $\mathbf{D}$, is

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

□

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 $\mathbf{D}_{ii}^{(k)} \cdot |\boldsymbol{x}_i^{(k)} - \boldsymbol{x}_i^{(k-2^\ell)}|^2$, among all coordinates i where $\overline{\boldsymbol{x}}_i$ has not been updated since 2^ℓ steps ago. With high probability, we can find all $i \in I_\ell^{(k)}$ in this way efficiently. To implement the sampling procedure, we make use of a data structure based on segment trees [CLRS09] along with sketching based on the Johnson-Lindenstrauss lemma.

Formally, we define the vector $\boldsymbol{q} \in \mathbb{R}^m$ where $\boldsymbol{q}_i \stackrel{\text{def}}{=} \mathbf{D}_{ii}^{(k)^{1/2}}(\boldsymbol{x}_i^{(k)} - \boldsymbol{x}_i^{(k-2^\ell)})$ 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.

Algorithm 7 Data structure AbstractMaintainApprox, Part 2

procedure FindLargeCoordinates(ℓ)

▷ $\overline{\mathbf{D}}$ and $\boldsymbol{q}$ are symbolic definitions

▷ $\overline{\mathbf{D}}$: diagonal matrix such that

$$\overline{\mathbf{D}}_{ii} = \begin{cases} \mathbf{D}_{ii}^{(k)} & \text{if } \overline{\boldsymbol{x}}_i \text{ has not been updated after the } (k-2^\ell)\text{-th step} \\ 0 & \text{otherwise.} \end{cases}$$

▷ $\boldsymbol{q} \stackrel{\text{def}}{=} \overline{\mathbf{D}}^{1/2}(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-2^\ell)})$ ▷ vector to sample coordinates from

$I \leftarrow \emptyset$ ▷ set of candidate coordinates

for $N \stackrel{\text{def}}{=} \Theta(2^{2\ell}(\beta/\delta)^2 \log^2 m \log(m/\rho))$ iterations **do**

▷ Sample coordinate i of $\boldsymbol{q}$ w.p. proportional to $\boldsymbol{q}_i^2$ by random descent down $\mathcal{T}$ to a leaf

while true do

$u \leftarrow \text{root}(\mathcal{T})$, $p_u \leftarrow 1$

while u is not a leaf node **do**

Sample a child u' of u with probability

$$\mathbf{P}(u \to u') \stackrel{\text{def}}{=} \frac{\|\boldsymbol{\Phi}_{E(u')}\boldsymbol{q}\|_2^2}{\sum_{\text{child } u'' \text{ of } u} \|\boldsymbol{\Phi}_{E(u'')}\boldsymbol{q}\|_2^2}$$

▷ let $\boldsymbol{\Phi}_{E(u)} \stackrel{\text{def}}{=} \boldsymbol{\Phi}\mathbf{I}_{E(u)}$ for each node u

$p_u \leftarrow p_u \cdot \mathbf{P}(u \to u')$

$u \leftarrow u'$

end while

break with probability $p_{\text{accept}} \stackrel{\text{def}}{=} \left\|\boldsymbol{q}|_{E(u)}\right\|^2 / (2 \cdot p_u \cdot \|\boldsymbol{\Phi}\boldsymbol{q}\|_2^2)$

end while

$I \leftarrow I \cup E(u)$

end for

return $\{i \in I \ : \ \boldsymbol{q}_i \geq \frac{\delta}{2\lceil \log m \rceil}\}$.

end procedure

Lemma 52. *Assume that $\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\|_{\mathbf{D}^{(k+1)}} \leq \beta$ for all IPM steps k. Let $\rho < 1$ be any given*

failure probability, and let $N \stackrel{\text{def}}{=} \Theta(2^{2\ell}(\beta/\delta)^2 \log^2 m \log(m/\rho))$ *be the number of samples Algorithm 7 takes. Then with probability* $\geq 1-\rho$*, during the* k*-th call of* APPROXIMATE*, Algorithm 7 finds the set* $I_\ell^{(k)}$ *correctly. Furthermore, the while-loop in Line 40 happens only* $O(1)$ *times in expectation per sample.*

Proof. The proof is similar to Lemma 6.17 in [DLY21b]. 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 47 breaks with probability at least $\frac{1}{4}$. By the choice of w, Johnson–Lindenstrauss lemma shows that $\|\mathbf{\Phi}_{E(u)}\boldsymbol{q}\|_2^2 = (1 \pm \frac{1}{9\eta})\|\mathbf{I}_{E(u)}\boldsymbol{q}\|_2^2$ 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 47 is at least

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

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

Next, we note that u is accepted on Line 47 with probability

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

Since $\|\mathbf{\Phi}\boldsymbol{q}\|_2^2$ 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 49, u is chosen with probability exactly $\left\|\boldsymbol{q}|_{E(u)}\right\|^2 / \|\boldsymbol{q}\|^2$.

Now, we want to show the output set is exactly $\{i \in [n] : |\boldsymbol{q}_i| \geq \frac{\delta}{2\lceil \log m \rceil}\}$. Let S denote the set of indices where $\overline{\boldsymbol{x}}$ did not update between the $(k-2^\ell)$-th step and the current k-th step. Then

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

where we used $\mathbf{I}_S\mathbf{D}^{(i+1)} = \mathbf{I}_S\mathbf{D}^{(k)}$, because $\overline{\boldsymbol{x}}_i$ changes whenever $\mathbf{D}_{ii}$ changes at a step. Hence, each leaf u is sampled with probability at least $\left\|\boldsymbol{q}|_{E(u)}\right\|^2/(2^\ell\beta)^2$. If $|\boldsymbol{q}_i| \geq \frac{\delta}{2\lceil\log m\rceil}$, and $i \in E(u)$ for a leaf node u, then the coordinate i is not in I with probability at most

$$\left(1 - \frac{\left\|\boldsymbol{q}|_{E(u)}\right\|^2}{(2^\ell\beta)^2}\right)^N \leq \left(1 - \frac{1}{2^{2\ell+2}(\beta/\delta)^2\lceil\log m\rceil^2}\right)^N \leq \frac{\rho}{m},$$

by our choice of N. Hence, all i with $|\boldsymbol{q}_i| \geq \frac{\delta}{2\lceil\log m\rceil}$ lies in I with probability at least $1-\rho$. This proves that the output set is exactly $I_\ell^{(k)}$ with probability at least $1-\rho$. □

Remark 53. In Algorithm 7, 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 54. 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 55. *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 Eq. (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 56. *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 $\mathcal{T}_H$, ie. descendants of H and H itself, and the second part involves a sum over all ancestors of H.

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

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

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

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

The first term is precisely $\overline{\mathbf{M}^{(H)}}\boldsymbol{z}$. 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

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

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

Lemma 57. *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 $\boldsymbol{\Phi} \in \mathbb{R}^{w \times m}$ be a JL-sketch matrix. Then* MaintainSketch *(Algorithm 8) is a data structure that maintains $\boldsymbol{\Phi}(\boldsymbol{y} + \mathbf{M}\boldsymbol{z})$, as $\boldsymbol{y}$, $\mathbf{M}$ and $\boldsymbol{z}$ undergo changes in the IPM. The data structure supports the following procedures:*

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

Algorithm 8 Data structure for maintaining $\boldsymbol{\Phi}(\boldsymbol{y} + \mathbf{M}\boldsymbol{z})$, Part 1

data structure MAINTAINSKETCH
private : member
    $\mathcal{T}$: rooted constant degree tree, where at every node H, there is
        $\boxed{\boldsymbol{\Phi}\mathbf{M}^{(H)}}$: sketch of partial tree operator
        $\boxed{\boldsymbol{\Phi}\overline{\mathbf{M}^{(H)}\boldsymbol{z}}}$: sketched vector ▷ This gives $\boldsymbol{\Phi}\mathbf{M}\boldsymbol{z}$ at the root
        $\boxed{\boldsymbol{\Phi}\boldsymbol{y}|_{E(H)}}$: sketched subvector of $\boldsymbol{y}$
    $\boldsymbol{\Phi} \in \mathbb{R}^{w \times m}$: JL-sketch matrix
    $\mathbf{M}$: tree operator on $\mathcal{T}$
    $\boldsymbol{z} \in \mathbb{R}^n$: vector $\boldsymbol{z}$
    $\boldsymbol{y} \in \mathbb{R}^n$: vector $\boldsymbol{y}$ ▷ $\mathbf{M}, \boldsymbol{z}, \boldsymbol{y}$ are pointers to read-only memory

procedure INITIALIZE(rooted tree $\mathcal{T}$, $\boldsymbol{\Phi} \in \mathbb{R}^{w \times m}$, tree operator $\mathbf{M}$, $\boldsymbol{z}$, $\boldsymbol{y}$)
    $\boldsymbol{\Phi} \leftarrow \boldsymbol{\Phi}$, $\mathcal{T} \leftarrow \mathcal{T}$
    $\boxed{\boldsymbol{\Phi}\mathbf{M}^{(H)}} \leftarrow \mathbf{0}$, $\boxed{\boldsymbol{\Phi}\overline{\mathbf{M}^{(H)}\boldsymbol{z}}} \leftarrow \mathbf{0}$, $\boxed{\boldsymbol{\Phi}\boldsymbol{y}|_{E(H)}} \leftarrow \mathbf{0}$ for all $H \in \mathcal{T}$
    UPDATE($\mathbf{M}, \boldsymbol{z}, \boldsymbol{y}, V(\mathcal{T})$)
end procedure

procedure UPDATE($\mathbf{M}^{(\text{new})}, \boldsymbol{z}^{(\text{new})}, \boldsymbol{y}^{(\text{new})}, \mathcal{S} \stackrel{\text{def}}{=}$ set of nodes admitting changes)
    $\mathbf{M} \leftarrow \mathbf{M}^{(\text{new})}$, $\boldsymbol{z} \leftarrow \boldsymbol{z}^{(\text{new})}$, $\boldsymbol{y} \leftarrow \boldsymbol{y}^{(\text{new})}$
    for $H \in \mathcal{P}_{\mathcal{T}}(\mathcal{S})$ by increasing node level **do**
        if H is a leaf **then**
            $\boxed{\boldsymbol{\Phi}\mathbf{M}^{(H)}} \leftarrow \boldsymbol{\Phi}\mathbf{J}_H$
            $\boxed{\boldsymbol{\Phi}\overline{\mathbf{M}^{(H)}\boldsymbol{z}}} \leftarrow \boldsymbol{\Phi}\mathbf{J}_H\boldsymbol{z}|_{F_H}$
            $\boxed{\boldsymbol{\Phi}\boldsymbol{y}|_{E(H)}} \leftarrow \boldsymbol{\Phi}\boldsymbol{y}|_{E(H)}$
        else
            $\boxed{\boldsymbol{\Phi}\mathbf{M}^{(H)}} \leftarrow \sum_{\text{child } D \text{ of } H} \boxed{\boldsymbol{\Phi}\mathbf{M}^{(D)}}\mathbf{M}_{(D,H)}$
            $\boxed{\boldsymbol{\Phi}\overline{\mathbf{M}^{(H)}\boldsymbol{z}}} \leftarrow \boxed{\boldsymbol{\Phi}\mathbf{M}^{(H)}}\boldsymbol{z}|_{F_H} + \sum_{\text{child } D \text{ of } H} \boxed{\boldsymbol{\Phi}\overline{\mathbf{M}^{(D)}\boldsymbol{z}}}$
            $\boxed{\boldsymbol{\Phi}\boldsymbol{y}|_{E(H)}} \leftarrow \sum_{\text{child } D \text{ of } H} \boxed{\boldsymbol{\Phi}\boldsymbol{y}|_{E(D)}}$
        end if
    end for
end procedure

procedure SUMANCESTORS($H \in \mathcal{T}$)
    if UPDATE has not been called since the last call to SUMANCESTORS(H) **then**
        return the result of the last SUMANCESTORS(H)
    end if
    if H is the root **then return 0**
    end if
    return $\mathbf{M}_{(H,P)}(\boldsymbol{z}|_{F_P} + \text{SUMANCESTORS}(P))$ ▷ P is the parent of H
end procedure

Algorithm 8 Data structure for maintaining $\boldsymbol{\Phi}(\boldsymbol{y} + \mathbf{M}\boldsymbol{z})$, part 2

41: **procedure** ESTIMATE($H \in \mathcal{T}$)
42: Let $\boldsymbol{u}$ be the result of SUMANCESTORS(H)
43: **return** $\boxed{\boldsymbol{\Phi}\mathbf{M}^{(H)}}\boldsymbol{u} + \boxed{\boldsymbol{\Phi}\overline{\mathbf{M}^{(H)}\boldsymbol{z}}} + \boxed{\boldsymbol{\Phi}\boldsymbol{y}|_{E(H)}}$
44: **end procedure**
45:
46: **procedure** QUERY(leaf $H \in \mathcal{T}$)
47: **return** $\boldsymbol{y}|_{E(H)} + \mathbf{J}_H(\boldsymbol{z}|_{F_H} + \text{SUMANCESTORS}(H))$
48: **end procedure**

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 $\boldsymbol{u}^\top \mathbf{M}_e$ and $\mathbf{M}_e \boldsymbol{x}$ for any vectors $\boldsymbol{u}$ and $\boldsymbol{x}$.

We prove the running time and correctness for each 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.

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 26 multiplies each row of $\boxed{\boldsymbol{\Phi}\mathbf{M}^{(D_1)}}$ with $\mathbf{M}_{(D_1,H)}$, each row of $\boxed{\boldsymbol{\Phi}\mathbf{M}^{(D_2)}}$ 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 26 in the procedure is $O(w \cdot T(\eta|S|))$.

Line 27 multiply each row of $\boxed{\boldsymbol{\Phi}\mathbf{M}^{(H)}}$ with a vector and then performs a constant number of additions of $O(w)$-length vectors. Since $\boxed{\boldsymbol{\Phi}\mathbf{M}^{(H)}}$ 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 $\boxed{\boldsymbol{\Phi}\mathbf{M}^{(H)}}$ is used once in Line 27 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 28 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 follow from Lemmas 55 and 56, and the correctness of $\boxed{\boldsymbol{\Phi y}|_{E(H)}}$ 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. We did not provide the corresponding pseudocode.

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 $\mathbf{M}, c, \boldsymbol{z}^{(\text{step})}, \boldsymbol{z}^{(\text{sum})}, \boldsymbol{y}, \mathbf{D}$ at every IPM step, and maintains the approximation $\overline{\boldsymbol{x}}$ to $\boldsymbol{x} \stackrel{\text{def}}{=} \boldsymbol{y} + \mathbf{M}\boldsymbol{z} = \boldsymbol{y} + \mathbf{M}(c \cdot \boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})})$ satisfying $\left\|\mathbf{D}^{1/2}(\boldsymbol{x} - \overline{\boldsymbol{x}})\right\|_\infty \leq \delta$. It supports the following procedures:*

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

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

Suppose $\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\|_{\mathbf{D}^{(k+1)}} \leq \beta$ 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 first updates $\overline{\boldsymbol{x}}_i \leftarrow \boldsymbol{x}_i^{(k-1)}$ for the coordinates i with $\mathbf{D}_{ii}^{(k)} \neq \mathbf{D}_{ii}^{(k-1)}$, then updates $\overline{\boldsymbol{x}}_i \leftarrow \boldsymbol{x}_i^{(k)}$ for $O(N_k \stackrel{\text{def}}{=} 2^{2\ell_k}(\beta/\delta)^2 \log^2 m)$ coordinates, where ℓ_k is the largest integer ℓ with $k = 0 \bmod 2^\ell$.*
- *The amortized time cost of* APPROXIMATE *is*

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

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 $\boldsymbol{y}_e$ or $\mathbf{D}_{e,e}$ changed for some edge e in H, compared to the $(k-1)$-th step.

Proof. 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 8.

First, for notation purposes, let $\boldsymbol{z} \stackrel{\text{def}}{=} c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})}$, and let $\boldsymbol{x} \stackrel{\text{def}}{=} \boldsymbol{y} + \mathbf{M}\boldsymbol{z}$, so that at step k, APPROXIMATE procedure has $\boldsymbol{x}^{(k)}$ (in implicit form) as input, and return $\overline{\boldsymbol{x}}$.

Let $\ell \in \{1, \ldots, O(\log m)\}$. We define a new dynamic vector $\boldsymbol{x}_\ell$ *symbolically*, which is represented at each step k for $k \geq 2^\ell$ by

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

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

- $\mathbf{M}_\ell^{(k)}{}_{(H,P)} = \text{diag}\left(\mathbf{M}_{(H,P)}^{(k)}, \mathbf{M}_{(H,P)}^{(k-2^\ell)}\right)$ for each child-parent edge (H, P) in $\mathcal{T}$,
- $\mathbf{J}_\ell^{(k)}{}_H = \overline{\mathbf{D}}_{E(H),E(H)} \left[\mathbf{J}_H^{(k)} \ \mathbf{J}_H^{(k-2^\ell)}\right]$ 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 $\boldsymbol{y}_\ell^{(k)} = \overline{\mathbf{D}}^{1/2}\left(\boldsymbol{y}^{(k)} - \boldsymbol{y}^{(k-2^\ell)}\right)$, and $\boldsymbol{z}_\ell$ by $\boldsymbol{z}_\ell^{(k)} \stackrel{\text{def}}{=} \left[\boldsymbol{z}^{(k)} \ \boldsymbol{z}^{(k-2^\ell)}\right]^\top$. Then, at each step k with $k \geq 2^\ell$, we have

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

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 ℓ-th 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$(\mathcal{T}, \mathbf{M}, c, \boldsymbol{z}^{(\text{step})}, \boldsymbol{z}^{(\text{sum})}, \boldsymbol{y}, \mathbf{D}, \rho, \delta)$**:** This procedure implements the initialization of ABSTRACTMAINTAINAPPROX, where the dynamic vector $\boldsymbol{x}$ to be approximated is represented by $\boldsymbol{x} \stackrel{\text{def}}{=} \boldsymbol{y} + \mathbf{M}(c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})})$. The initialization steps described in Algorithm 7 takes $O(wm)$ time. Let $\boldsymbol{\Phi}$ 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 $\boldsymbol{\Phi}\boldsymbol{x}^{(k)}$, and `ox_prev` will maintain sketches of $\boldsymbol{\Phi}\boldsymbol{x}^{(k-1)}$. (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$(\mathcal{T}, \boldsymbol{\Phi}, \mathbf{M}_\ell^{(k)}, \boldsymbol{z}_\ell^{(k)}, \boldsymbol{y}_\ell^{(k)})$. (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 57. By the existing pseudocode in Algorithm 7, it correctly initializes $\overline{\boldsymbol{x}} \leftarrow \boldsymbol{x}$.

APPROXIMATE$(\mathbf{M}^{(\text{new})}, c^{(\text{new})}, \boldsymbol{z}^{(\text{step})^{(\text{new})}}, \boldsymbol{z}^{(\text{sum})^{(\text{new})}}, \boldsymbol{y}^{(\text{new})}, \mathbf{D}^{(\text{new})})$**:** This procedure implements APPROXIMATE in Algorithm 7. We consider when the current step is k below.

First, we update the sketch data structures $\texttt{sketch}_\ell$ for each ℓ by calling $\texttt{sketch}_\ell$.UPDATE. Recall at step k, $\texttt{sketch}_\ell$ maintains sketches for the vector $\boldsymbol{x}_\ell^{(k)} = \overline{\mathbf{D}}^{1/2}(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-2^\ell)})$, although the actual representation in $\texttt{sketch}_\ell$ of the vector $\boldsymbol{x}_\ell$ is given by $\boldsymbol{x}_\ell = \boldsymbol{y}_\ell + \mathbf{M}_\ell \boldsymbol{z}_\ell$ as defined in Eq. (6.1).

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

To update $\overline{\boldsymbol{x}}_e$ to $\boldsymbol{x}_e^{(k-1)}$ for a single coordinate (Line 20 of Algorithm 7), we find the leaf node H containing the edge e, and call `ox_prev`.QUERY(H). This returns the subvector $\boldsymbol{x}^{(k-1)}|_{E(H)}$, from which we can make the assignment to $\overline{\boldsymbol{x}}_e$. To update $\overline{\boldsymbol{x}}_e$ to $\boldsymbol{x}_e^{(k)}$ for single coordinates (Line 29 of Algorithm 7), we do the same as above, except using the data structure `ox_cur`.

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

Number of coordinates changed in $\overline{\boldsymbol{x}}$ during APPROXIMATE. In Line 20 of APPROXIMATE in Algorithm 7, $\overline{\boldsymbol{x}}$ is updated in every coordinate e where $\mathbf{D}_e$ differs compared to the previous step.

Next, the procedure collect a set of coordinates for which we update $\overline{\boldsymbol{x}}$, 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 $\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\|_{\mathbf{D}^{(k+1)}} \le \beta$ and Lemma 52. 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) $\mathbf{M}_{(H,P)}$, $\mathbf{J}_H$, $\boldsymbol{z}^{(\text{step})}|_{F_H}$, $\boldsymbol{z}^{(\text{sum})}|_{F_H}$, $\boldsymbol{y}_{F_H}$, $\mathbf{D}_{E(H)}$ changes during step k. (They are entirely induced by changes in $\boldsymbol{v}$ and $\boldsymbol{w}$ 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 $\overline{\boldsymbol{x}}_i$ 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 8. 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 57 show that $\texttt{sketch}_\ell$.Update with new iterations of the appropriate variables run in time

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

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.

Next, we consider the runtime for Line 20 in Algorithm 7: The number of coordinate accesses to $\boldsymbol{x}^{(k-1)}$ is $|\{i : \mathbf{D}_{ii}^{(k)} - \mathbf{D}_{ii}^{(k-1)} \neq 0\}| = O(\mathcal{S}^{(k)})$. Each coordinate is computed by calling `ox_cur`.Query, and by Lemma 57, the total time for these updates is $w \cdot O(T(\eta \cdot |\mathcal{S}^{(k)}|)$.

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 52, $N_{k,\ell} \stackrel{\text{def}}{=} \Theta(2^{2\ell}(\beta/\delta)^2 \log^2 m \log(m/\rho))$ 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 $\|\boldsymbol{\Phi}_{E(H)}\boldsymbol{x}_\ell\|_2^2$ 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 57. 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 Eq. (6.2).

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

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

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 projection

In this section, we define the slack tree operator as required to use MaintainRep. We then give the full slack maintenance data structure.

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 $\widetilde{\mathbf{L}}^{-1}$ denote the approximation of $\mathbf{L}^{-1}$ from Eq. (2.8), maintained and computable with a DynamicSC data structure. If we define

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

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 $\boldsymbol{z}^{(k)} \stackrel{\text{def}}{=} \widetilde{\boldsymbol{\Gamma}}\boldsymbol{\Pi}^{(\eta-1)}\cdots\boldsymbol{\Pi}^{(0)}\mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}^{(k)}$ at IPM step k. We proceed by defining a tree operator $\mathbf{M}$ satisfying $\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}^{(k)} = \mathbf{M}\boldsymbol{z}^{(k)}$. Namely, we show that $\mathbf{M} \stackrel{\text{def}}{=} \mathbf{W}^{1/2}\mathbf{B}\boldsymbol{\Pi}^{(0)\top}\cdots\boldsymbol{\Pi}^{(\eta-1)\top}$ is indeed a tree operator. Then we set $\mathbf{M}^{(\text{slack})} \stackrel{\text{def}}{=} \mathbf{W}^{-1/2}\mathbf{M}$.

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

Definition 58 (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:

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

where $\mathbf{X}^{(H)}$ is defined in Eq. (4.4).

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 59. *Let* $\mathbf{M}$ *be the tree operator as defined in Definition 58. We have*

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

We begin with a few observations about the $\mathbf{\Pi}^{(i)}$'s:

Observation 60. *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 61. *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. For simplicity of notation, let $V(H) \stackrel{\text{def}}{=} \partial H \cup F_H$ for a node H.

To start, observe that for a node A at level $\eta(A)$, we have $\mathbf{\Pi}^{(i)}\boldsymbol{z}|_{F_A} = \boldsymbol{z}|_{F_A}$ for all $i \ge \eta(A)$. So it suffices to prove

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

Let the path from leaf H up to node A in $\mathcal{T}$ be denoted ($H_0 \stackrel{\text{def}}{=} H, H_1, \ldots, H_t \stackrel{\text{def}}{=} A$), for some $t \le \eta(A)$. 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 $\boldsymbol{x} \stackrel{\text{def}}{=} \mathbf{\Pi}^{(\eta(H_{k+1}))\top}\cdots\mathbf{\Pi}^{(\eta(A)-1)\top}\boldsymbol{z}|_{F_A}$. Now, by the definition of the edge operator, the above equation becomes

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

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 60. In particular, $\boldsymbol{y}$ is zero on F_{H_k}. Also, $\boldsymbol{y}$ is zero on ∂H_k, since by Observation 24, $\partial H_k \subseteq \cup_{\text{ancestor } A' \text{ of } H_k} F_{A'}$, and ancestors of H_k are at level $\eta(H_{k+1})$ or higher. Then $\boldsymbol{y}$ is zero on $V(H_k) = \partial H_k \cup F_{H_k}$, and the right hand side is

$$= (\mathbf{I}_{V(H_k)} - \mathbf{X}^{(H_k)\top})\boldsymbol{x},$$

where we apply the definition of $\mathbf{\Pi}^{(\eta(H_k))^\top}$ and expand the left-multiplication by $\mathbf{I}_{V(H_k)}$.

Combining with Eq. (7.6) 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 59, we apply the leaf operators to the result of the previous lemma and sum over all nodes and leaf nodes.

Proof of Lemma 59. Let H be a leaf node. We sum Eq. (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 60, for any A with $H \notin \mathcal{T}_A$, we simply have $\mathbf{I}_{\partial H\cup F_H}\mathbf{\Pi}^{(0)^\top}\cdots\mathbf{\Pi}^{(\eta-1)^\top}\boldsymbol{z}|_{F_A} = \mathbf{0}$. Next, we apply the leaf operator $\mathbf{J}_H = \mathbf{W}^{1/2}\mathbf{B}[H]$ to both sides to get

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

Since $\mathbf{B}[H]$ is zero on columns supported on $V(G)\setminus(\partial H\cup F_H)$, we can simply drop the $\mathbf{I}_{\partial H\cup F_H}$ in the right hand side.

Finally, we sum up the equation above over all leaf nodes. The left hand side is precisely the definition of $\mathbf{M}\boldsymbol{z}$. Recall the regions of the leaf nodes partition the original graph G, so we have

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

□

We now examine the slack tree operator complexity.

Lemma 62. *The complexity of the slack tree operator as defined in Definition 65 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 $\mathbf{M}_{(D,P)} = \mathbf{I}_{\partial D} - \left(\mathbf{L}^{(D)}_{F_D,F_D}\right)^{-1}\mathbf{L}^{(D)}_{F_D,\partial D}$ to the left or right consists of three steps which are applying $\mathbf{I}_{\partial D}$, applying $\mathbf{L}^{(D)}_{F_D,\partial D}$ and solving for $\mathbf{L}^{(D)}_{F_D,F_D}\boldsymbol{v} = \boldsymbol{b}$ for some vectors $\boldsymbol{v}$ and $\boldsymbol{b}$. Each of the three steps costs time $O(\epsilon_{\mathbf{P}}{}^{-2}|\partial D\cup F_D|)$ by Lemma 34 and Theorem 12.

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 28, which then gives the claimed complexity. □

7.2 Proof of Theorem 9

Finally, we give the full data structure for maintaining the slack solution.

The tree operator $\mathbf{M}$ defined in Definition 58 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 $\boldsymbol{s} \leftarrow \boldsymbol{s} + \overline{t}h\mathbf{W}^{-1/2}\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}^{(k)}$, we define $\mathbf{M}^{(\text{slack})} \stackrel{\text{def}}{=} \mathbf{W}^{-1/2}\mathbf{M}$ and note it is also a tree operator:

Lemma 63. *Suppose $\mathbf{M}$ is a tree operator supported on $\mathcal{T}$ with complexity $T(K)$. Let $\mathbf{D}$ be a diagonal matrix in $\mathbb{R}^{E\times E}$ where $E = \bigcup_{\text{leaf } H\in\mathcal{T}} E(H)$. Then $\mathbf{DM}$ can be represented by a tree operator with complexity $T(K)$.*

Proof. Suppose $\mathbf{M} \in \mathbb{R}^{E\times V}$. For any vector $\boldsymbol{z} \in \mathbb{R}^V$, $\mathbf{DM}\boldsymbol{z} = \mathbf{D}(\mathbf{M}\boldsymbol{z})$. Thus, to compute $\mathbf{DM}\boldsymbol{z}$, we may first compute $\mathbf{M}\boldsymbol{z}$ and then multiply the i-th entry of $\mathbf{M}\boldsymbol{z}$ with $\mathbf{D}_{i,i}$. This can be achieved by defining a new tree operator $\mathbf{M}'$ with leaf operators $\mathbf{J}'$ such that $\mathbf{J}'_H = \mathbf{D}_{E(H),E(H)}\mathbf{J}_H$ and $\mathbf{M}'_{(H,P)} = \mathbf{M}_{(H,P)}$. The size of each leaf operator remains constant. All edge operators do not change from $\mathbf{M}$. Thus, the new operator $\mathbf{M}'$ has the same complexity as $\mathbf{M}$. □

With the lemma above, we can use MaintainRep (Algorithm 6) to maintain the implicit representation of $\boldsymbol{s}$ and Theorem 8 to maintain an approximate vector $\overline{\boldsymbol{s}}$ as required in Algorithm 2. A single IPM step calls the procedures Reweight, Move, Approximate in this order once. Note that we reinitialize the data structure when $\overline{t}$ changes, so within each instantiation, may assume $\overline{t} = 1$ by scaling. $\overline{t}$ changes only $\widetilde{O}(1)$ times in the IPM.

Theorem 9 (Slack maintenance)**.** *Given a modified planar graph G with m edges and its separator tree $\mathcal{T}$ with height η, the randomized data structure* MaintainSlack *(Algorithm 9) implicitly maintains the slack solution $\boldsymbol{s}$ undergoing IPM changes, and explicitly maintains its approximation $\overline{\boldsymbol{s}}$, and supports the following procedures with high probability against an adaptive adversary:*

- Initialize($G, \boldsymbol{s}^{(\text{init})} \in \mathbb{R}^m, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \epsilon_{\mathbf{P}} > 0, \overline{\epsilon} > 0$)*: Given a graph G, initial solution $\boldsymbol{s}^{(\text{init})}$, 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 representations $\boldsymbol{s} \leftarrow \boldsymbol{s}^{(\text{init})}$ and $\overline{\boldsymbol{x}} \leftarrow \boldsymbol{s}$.*
- Reweight($\boldsymbol{w} \in \mathbb{R}^m_{>0}$, *given implicitly as a set of changed weights): Set the current weights to $\boldsymbol{w}$ in $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK})$ time, where K is the number of coordinates changed in $\boldsymbol{w}$.*
- Move($\alpha \in \mathbb{R}, \boldsymbol{v} \in \mathbb{R}^m$ *given implicitly as a set of changed coordinates): Implicitly update $\boldsymbol{s} \leftarrow \boldsymbol{s} + \alpha\mathbf{W}^{-1/2}\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}$ for some $\widetilde{\mathbf{P}}_{\boldsymbol{w}}$ with $\|(\widetilde{\mathbf{P}}_{\boldsymbol{w}} - \mathbf{P}_{\boldsymbol{w}})\boldsymbol{v}\|_2 \le \eta\epsilon_{\mathbf{P}}\|\boldsymbol{v}\|_2$, and $\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v} \in \text{Range}(\mathbf{B})$. The total runtime is $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK})$ where K is the number of coordinates changed in $\boldsymbol{v}$.*
- Approximate() $\to \mathbb{R}^m$*: Return the vector $\overline{\boldsymbol{s}}$ such that $\|\mathbf{W}^{1/2}(\overline{\boldsymbol{s}} - \boldsymbol{s})\|_\infty \le \overline{\epsilon}$ for the current weight $\boldsymbol{w}$ and the current vector $\boldsymbol{s}$.*
- Exact() $\to \mathbb{R}^m$*: Output the current vector $\boldsymbol{s}$ 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 k-th* Reweight *and* Move *calls. Then at the k-th* Approximate *call,*

- *the data structure first sets $\overline{\boldsymbol{s}}_e \leftarrow \boldsymbol{s}_e^{(k-1)}$ for all coordinates e where $\boldsymbol{w}_e$ changed in the last* Reweight*, then sets $\overline{\boldsymbol{s}}_e \leftarrow \boldsymbol{s}_e^{(k)}$ 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 $\ell_0 = 0$.*

Algorithm 9 Slack Maintenance, Main Algorithm

data structure MAINTAINSLACK **extends** MAINTAINREP
private: member
    MAINTAINREP `maintainRep`: data structure to implicitly maintain

$$\boldsymbol{s} = \boldsymbol{y} + \mathbf{W}^{-1/2}\mathbf{M}(c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})}).$$

▷ $\mathbf{M}$ is defined by Definition 58
    MAINTAINAPPROX `bar_s`: data structure to maintain approximation $\overline{\boldsymbol{s}}$ to $\boldsymbol{s}$ (Theorem 8)

procedure INITIALIZE($G, \boldsymbol{s}^{(\text{init})} \in \mathbb{R}^m, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \epsilon_{\mathbf{P}} > 0, \overline{\epsilon} > 0$)
    Build the separator tree $\mathcal{T}$ by Theorem 25
    `maintainRep`.INITIALIZE($G, \mathcal{T}, \mathbf{W}^{-1/2}\mathbf{M}, \boldsymbol{v}, \boldsymbol{w}, \boldsymbol{s}^{(\text{init})}, \epsilon_{\mathbf{P}}$) ▷ initialize $\boldsymbol{s} \leftarrow \boldsymbol{s}^{(\text{init})}$
    `bar_s`.INITIALIZE($\mathbf{W}^{-1/2}\mathbf{M}, c, \boldsymbol{z}^{(\text{step})}, \boldsymbol{z}^{(\text{sum})}, \boldsymbol{y}, \mathbf{W}, n^{-5}, \overline{\epsilon}$) ▷ initialize $\overline{\boldsymbol{s}}$ approximating $\boldsymbol{s}$
end procedure

procedure REWEIGHT($\boldsymbol{w}^{(\text{new})} \in \mathbb{R}^m_{>0}$)
    `maintainRep`.REWEIGHT($\boldsymbol{w}^{(\text{new})}$)
end procedure

procedure MOVE($\alpha, \boldsymbol{v}^{(\text{new})} \in \mathbb{R}^m$)
    `maintainRep`.MOVE($\alpha, \boldsymbol{v}^{(\text{new})}$)
end procedure

procedure APPROXIMATE( )
    ▷ the variables in the argument are accessed from `maintainRep`
    return $\overline{\boldsymbol{s}}$ = `bar_s`.APPROXIMATE($\mathbf{W}^{-1/2}\mathbf{M}, c, \boldsymbol{z}^{(\text{step})}, \boldsymbol{z}^{(\text{sum})}, \boldsymbol{y}, \mathbf{W}$)
end procedure

procedure EXACT( )
    return `maintainRep`.EXACT()
end procedure

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

Proof of Theorem 9. We prove the runtime and correctness of each procedure separately.

Recall by Lemma 61, 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 $\boldsymbol{s}$ in `maintainRep` is correct and $\boldsymbol{s} = \boldsymbol{s}^{(\text{init})}$. By the initialization of `bar_f`, $\overline{\boldsymbol{s}}$ is set to $\boldsymbol{s}$ to start.

Initialization of `maintainRep` takes $\widetilde{O}(m\epsilon_{\mathbf{P}}{}^{-2})$ time by Theorem 7, and the initialization of `bar_s` takes $\widetilde{O}(m)$ time by Theorem 8.

Reweight: In Reweight, the value of $\boldsymbol{s}$ does not change, but all the variables in `MaintainRep` are updated to depend on the new weights. The correctness and runtime follow from Theorem 7.

Move: `maintainRep`.Move$(\alpha, \boldsymbol{v}^{(k)})$ updates the implicit representation of $\boldsymbol{s}$ by

$$\boldsymbol{s} \leftarrow \boldsymbol{s} + \mathbf{W}^{-1/2}\mathbf{M}\alpha\boldsymbol{z}^{(k)}.$$

By the definition of the slack projection tree operator $\mathbf{M}$ and Lemma 59, this is equivalent to the update

$$\boldsymbol{s} \leftarrow \boldsymbol{s} + \alpha\mathbf{W}^{-1/2}\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}^{(k)},$$

where $\widetilde{\mathbf{P}}_{\boldsymbol{w}} = \mathbf{W}^{1/2}\mathbf{B}\boldsymbol{\Pi}^{(0)}\cdots\boldsymbol{\Pi}^{(\eta-1)}\widetilde{\boldsymbol{\Gamma}}\boldsymbol{\Pi}^{(\eta-1)}\cdots\boldsymbol{\Pi}^{(0)}\mathbf{B}^\top\mathbf{W}^{1/2}$. By Theorem 33, $\|\widetilde{\mathbf{P}}_{\boldsymbol{w}} - \mathbf{P}_{\boldsymbol{w}}\|_{\text{op}} \le \eta\epsilon_{\mathbf{P}}$. From the definition, $\text{Range}(\mathbf{W}^{1/2}\widetilde{\mathbf{P}}_{\boldsymbol{w}}) \subseteq \text{Range}(\mathbf{B})$.

By the guarantees of `maintainRep`, if $\boldsymbol{v}^{(k)}$ differs from $\boldsymbol{v}^{(k-1)}$ on K coordinates, then the runtime is $\widetilde{O}(\epsilon_{\mathbf{P}}{}^{-2}\sqrt{mK})$. Furthermore, $\boldsymbol{z}^{(\text{step})}$ and $\boldsymbol{z}^{(\text{sum})}$ change on F_H for at most $\widetilde{O}(K)$ nodes in $\mathcal{T}$.

Approximate: The returned vector $\overline{\boldsymbol{s}}$ satisfies $\|\mathbf{W}^{1/2}(\overline{\boldsymbol{s}} - \boldsymbol{s})\|_\infty \le \overline{\epsilon}$ by the guarantee of `bar_s`.Approximate from Theorem 8.

Exact: The runtime and correctness directly follow from the guarantee of `maintainRep`.Exact given in Theorem 7.

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

Lemma 64. *Suppose $\alpha\|\boldsymbol{v}\|_2 \le \beta$ 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 k-th* Reweight *and* Move *calls. Then at the k-th* Approximate *call,*

- *The data structure first sets $\overline{\boldsymbol{s}}_e \leftarrow \boldsymbol{s}_e^{(k-1)}$ for all coordinates e where $\boldsymbol{w}_e$ changed in the last* Reweight*, then sets $\overline{\boldsymbol{s}}_e \leftarrow \boldsymbol{s}_e^{(k)}$ 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 k-th* Approximate *call is* $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{m(K+N_{k-2^{\ell_k}})})$.

Proof. Since $\overline{\boldsymbol{s}}$ is maintained by `bar_s`, we apply Theorem 8 with $\boldsymbol{x} = \boldsymbol{s}$ and diagonal matrix $\mathbf{D} = \mathbf{W}$. We need to prove $\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\|_{\mathbf{D}^{(k)}} \le O(\beta)$ for all k first. The constant factor in $O(\beta)$

does not affect the guarantees in Theorem 8. The left-hand side is

$$\begin{aligned}\left\|\boldsymbol{s}^{(k)}-\boldsymbol{s}^{(k-1)}\right\|_{\mathbf{W}^{(k)}} &= \left\|\alpha^{(k)}\mathbf{W}^{(k)^{-1/2}}\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}^{(k)}\right\|_{\mathbf{W}^{(k)}} && \text{(by MOVE)}\\ &= \left\|\alpha^{(k)}\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}^{(k)}\right\|_2\\ &\le (1+\eta\epsilon_{\mathbf{P}})\alpha^{(k)}\|\boldsymbol{v}^{(k)}\|_2 && \text{(by the assumption that } \alpha\|\boldsymbol{v}\|_2\le\beta)\\ &\le 2\beta.\end{aligned}$$

Where the second last step follows from $\|\widetilde{\mathbf{P}}_{\boldsymbol{w}}-\mathbf{P}_{\boldsymbol{w}}\|_{\mathrm{op}}\le\eta\epsilon_{\mathbf{P}}$ and the fact that $\mathbf{P}_{\boldsymbol{w}}$ is an orthogonal projection. Now, we can apply Theorem 8 to conclude that at each step k, `bar_s`.APPROXIMATE first sets $\overline{\boldsymbol{s}}_e\leftarrow\boldsymbol{s}_e^{(k-1)}$ for all coordinates e where $\boldsymbol{w}_e$ changed in the last REWEIGHT, then set $\overline{\boldsymbol{s}}_e\leftarrow\boldsymbol{s}_e^{(k)}$ for $O(N_k\stackrel{\text{def}}{=}2^{2\ell_k}(\frac{\beta}{\epsilon})^2\log^2 m)$ 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 $\boldsymbol{z}^{(\text{step})}$ and $\boldsymbol{z}^{(\text{sum})}$ on F_H for $\widetilde{O}(K)$ different nodes $H\in\mathcal{T}$ by Theorem 7. REWEIGHT then updates $\boldsymbol{z}^{(\text{step})}$ and $\boldsymbol{z}^{(\text{sum})}$ 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. Now, we apply Theorem 8 and the complexity of the tree operator to conclude the desired amortized runtime. □

□

8 Flow projection

In this section, we define the flow tree operator as required to use MAINTAINREP. We then give the full flow maintenance data structure.

During the IPM, we maintain $\boldsymbol{f}\stackrel{\text{def}}{=}\hat{\boldsymbol{f}}-\boldsymbol{f}^\perp$ by maintaining the two terms separately. For IPM step k with direction $\boldsymbol{v}^{(k)}$ and step size h, we update them as follows:

$$\begin{aligned}\hat{\boldsymbol{f}}&\leftarrow\hat{\boldsymbol{f}}+h\mathbf{W}^{1/2}\boldsymbol{v}^{(k)},\\ \boldsymbol{f}^\perp&\leftarrow\boldsymbol{f}^\perp+h\mathbf{W}^{1/2}\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v}^{(k)},\end{aligned}$$

where $\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v}^{(k)}$ satisfies $\left\|\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v}^{(k)}-\mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}^{(k)}\right\|_2\le\varepsilon\left\|\boldsymbol{v}^{(k)}\right\|_2$ for some factor ε, and $\mathbf{B}^\top\mathbf{W}^{1/2}\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v}^{(k)}=\mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}^{(k)}$. We will include the initial value of $\boldsymbol{f}$ in $\hat{\boldsymbol{f}}$.

Maintaining $\hat{\boldsymbol{f}}$ is straightforward; in the following section, we focus on $\boldsymbol{f}^\perp$.

8.1 Tree operator for flow

We hope to use MAINTAINREP to maintain $\boldsymbol{f}^\perp$ throughout the IPM. In order to do so, it remains to define a flow tree operator $\mathbf{M}^{(\text{flow})}$ so that

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

where $\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v}$ satisfies the constraints mentioned above, and $\boldsymbol{z}^{(k)}\stackrel{\text{def}}{=}\widetilde{\boldsymbol{\Gamma}}\boldsymbol{\Pi}^{(\eta-1)}\cdots\boldsymbol{\Pi}^{(0)}\mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}^{(k)}$. We will define a flow projection tree operator $\mathbf{M}$ so that $\tilde{\boldsymbol{f}}\stackrel{\text{def}}{=}\mathbf{M}\boldsymbol{v}^{(k)}$ satisfies $\left\|\tilde{\boldsymbol{f}}-\mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}\right\|_2\le O(\eta\epsilon_{\mathbf{P}})\left\|\boldsymbol{v}\right\|_2$ and $\mathbf{B}^\top\mathbf{W}^{1/2}\tilde{\boldsymbol{f}}=\mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}^{(k)}$. This means it is feasible to set $\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v}^{(k)}=\tilde{\boldsymbol{f}}$. Then, we define $\mathbf{M}^{(\text{flow})}\stackrel{\text{def}}{=}\mathbf{W}^{-1/2}\mathbf{M}$.

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

Definition 65 (Flow projection tree operator)**.** Let $\mathcal{T}$ be the separator tree from data structure DynamicSC, with Laplacians $\mathbf{L}^{(H)}$ and $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ at each node $H \in \mathcal{T}$. We use $\mathbf{B}[H]$ to denote the adjacency matrix of G restricted to the region.

To define the flow projection tree operator $\mathbf{M}$, we proceed as follows: The tree operator is supported on the tree $\mathcal{T}$. For a node $H \in \mathcal{T}$ with parent P, define the tree edge operator $\mathbf{M}_{(H,P)}$ as:

$$\mathbf{M}_{(H,P)} \stackrel{\text{def}}{=} (\mathbf{L}^{(H)})^{-1}\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H). \tag{8.1}$$

At each node H, we let F_H in the tree operator be the set F_H of eliminated vertices defined in the separator tree. At each leaf node H of $\mathcal{T}$, we have the leaf operator $\mathbf{J}_H = \mathbf{W}^{1/2}\mathbf{B}[H]$.

Before we give intuition and formally prove the correctness of the flow tree operator, we examine its complexity.

Lemma 66. *The complexity of the flow tree operator as defined in Definition 65 is* $T(k) = \widetilde{O}(\sqrt{mk} \cdot {\epsilon_{\mathbf{P}}}^{-2})$*, where* $\epsilon_{\mathbf{P}}$ *is the overall approximation factor from data structure* DynamicSC*.*

Proof. Let $\mathbf{M}_{(H,P)}$ be a tree edge operator. Note that it is a symmetric matrix. 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.

If H is not a leaf node, then $\mathbf{M}_{(H,P)}\boldsymbol{u}$ consists of multiplying with $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ and solving the Laplacian system $\mathbf{L}^{(H)}$. By Lemma 34 and Theorem 12, this can be done in $\widetilde{O}({\epsilon_{\mathbf{P}}}^{-2} \cdot |\partial H|)$ time. To bound the total cost over k distinct edges, we apply Lemma 28, which gives the claimed complexity. □

Theorem 67. *Let* $\boldsymbol{v} \in \mathbb{R}^m$*, and let* $\boldsymbol{z} = \widetilde{\mathbf{\Gamma}}\mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)}\mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}$*. Let* $\mathbf{M}$ *be the flow projection tree operator from Definition 65. Suppose* $\epsilon_{\mathbf{P}} = O(1/\log m)$ *is the overall approximation factor from* DynamicSC*. Then* $\tilde{\boldsymbol{f}} \stackrel{\text{def}}{=} \mathbf{M}\boldsymbol{z}$ *satisfies* $\mathbf{B}^\top\mathbf{W}^{1/2}\tilde{\boldsymbol{f}} = \mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}$ *and* $\left\|\tilde{\boldsymbol{f}} - \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}\right\|_2 \leq O(\eta\epsilon_{\mathbf{P}}) \left\|\boldsymbol{v}\right\|_2$*.*

The remainder of the section is dedicated to proving this theorem.

Fix $\boldsymbol{v}$ for the remainder of this section. Let $\boldsymbol{d} \stackrel{\text{def}}{=} \mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v} \in \mathbb{R}^n$; since it is supported on the vertices of G and its entries sum to 0, it is a *demand* vector. In the first part of the proof, we show that $\tilde{\boldsymbol{f}}$ routes the demand $\boldsymbol{d}$. Let $\boldsymbol{f}^\star \stackrel{\text{def}}{=} \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v} = \mathbf{W}^{1/2}\mathbf{B}\mathbf{L}^{-1}\boldsymbol{d}$. In the second part of the proof, we show that $\tilde{\boldsymbol{f}}$ is close to $\boldsymbol{f}^\star$. Finally, a remark about terminology:

Remark 68. If $\mathbf{B}$ is the incidence matrix of a graph, then any vector of the form $\mathbf{B}\boldsymbol{x}$ is a flow by definition. Often in this section, we have vectors of the form $\mathbf{W}^{1/2}\mathbf{B}\boldsymbol{x}$. In this case, we refer to it as a *weighted flow*. We say a weighted flow $\boldsymbol{f}$ routes a demand $\boldsymbol{d}$ if $(\mathbf{W}^{1/2}\mathbf{B})^\top\boldsymbol{f} = \boldsymbol{d}$.

We proceed with a series of lemmas and their intuition, before tying them together in the overall proof at the end of the section.

Lemma 69. *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 67. 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, $\boldsymbol{z}|_{F_H}$ below should be viewed as an n-dimensional vector supported on F_H. Define

$$\mathbf{X}^{(i)} = \sum_{H \in \mathcal{T}(i)} \mathbf{X}^{(H)}.$$

We have

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

Suppose H is at level i of $\mathcal{T}$. We have

$$\begin{aligned} \boldsymbol{z}|_{F_H} &= (\mathbf{L}^{(H)}_{F_H,F_H})^{-1} \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(1)} \mathbf{\Pi}^{(0)} \boldsymbol{d} \\ &= (\mathbf{L}^{(H)}_{F_H,F_H})^{-1} \mathbf{\Pi}^{(i-1)} \cdots \mathbf{\Pi}^{(1)} \mathbf{\Pi}^{(0)} \boldsymbol{d}, \end{aligned} \tag{8.2}$$

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 Eq. (8.2). 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

$$\begin{aligned} \sum_{H \in \mathcal{T}} \boldsymbol{d}^{(H)} &= \sum_{i=0}^{\eta} \sum_{H \in \mathcal{T}(i)} \mathbf{X}^{(H)} \mathbf{\Pi}^{(i-1)} \cdots \mathbf{\Pi}^{(1)} \mathbf{\Pi}^{(0)} \boldsymbol{d} + \sum_{i=0}^{\eta} \sum_{H \in \mathcal{T}(i)} (\mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(1)} \mathbf{\Pi}^{(0)} \boldsymbol{d})|_{F_H} \\ &= \left(\sum_{i=0}^{\eta} \mathbf{X}^{(i)} \mathbf{\Pi}^{(i-1)} \cdots \mathbf{\Pi}^{(1)} \mathbf{\Pi}^{(0)} \boldsymbol{d} \right) + \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(1)} \mathbf{\Pi}^{(0)} \boldsymbol{d} \qquad (F_H \text{ partition } V(G)) \\ &= \left(\sum_{i=0}^{\eta-1} (\mathbf{I} - \mathbf{\Pi}^{(i)}) \mathbf{\Pi}^{(i-1)} \cdots \mathbf{\Pi}^{(1)} \mathbf{\Pi}^{(0)} \boldsymbol{d} \right) + \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(1)} \mathbf{\Pi}^{(0)} \boldsymbol{d} \\ &= \boldsymbol{d}, \qquad \text{(telescoping sum)} \end{aligned}$$

completing our proof. □

Next, we examine the feasibility of $\tilde{\boldsymbol{f}}$. To begin, we introduce a decomposition of $\tilde{\boldsymbol{f}}$ based on the decomposition of $\boldsymbol{d}$, and prove its feasibility.

Definition 70. Let $\mathbf{M}^{(H)}$ be the flow tree operator supported on the tree $\mathcal{T}_H \in \mathcal{F}$ (Definition 45). We define the flow $\tilde{\boldsymbol{f}}^{(H)} \stackrel{\text{def}}{=} \mathbf{M}^{(H)} \boldsymbol{z} = \mathbf{M}^{(H)} \boldsymbol{z}|_{F_H}$.

Lemma 71. *We have that* $(\mathbf{W}^{1/2}\mathbf{B})^\top \tilde{\boldsymbol{f}}^{(H)} = \boldsymbol{d}^{(H)}$. *In other words, the weighted flow* $\tilde{\boldsymbol{f}}^{(H)}$ *routes the demand* $\boldsymbol{d}^{(H)}$ using the edges of the original graph G.

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 $\mathcal{F}_H$ 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 other case, H is not a leaf node of $\mathcal{T}$. Let D_1, D_2 be the two children of H. Then

$$\begin{aligned}
\mathbf{B}^\top \mathbf{W}^{1/2} \mathbf{M}^{(H)} &= \mathbf{B}^\top \mathbf{W}^{1/2} \left(\mathbf{M}^{(D_1)} \mathbf{M}_{(D_1,H)} + \mathbf{M}^{(D_2)} \mathbf{M}_{(D_2,H)} \right) \\
&= \mathbf{L}^{(D_1)} \mathbf{M}_{(D_1,H)} + \mathbf{L}^{(D_2)} \mathbf{M}_{(D_2,H)} && \text{(by induction)} \\
&= \mathbf{L}^{(D_1)} (\mathbf{L}^{(D_1)})^{-1} \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)}, \partial D_1) + \mathbf{L}^{(D_2)} (\mathbf{L}^{(D_2)})^{-1} \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)}, \partial D_2) \\
&= \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)}, \partial D_1) + \widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)}, \partial D_2) \\
&= \mathbf{L}^{(H)}.
\end{aligned}$$

Finally, we conclude that $\mathbf{B}^\top \mathbf{W}^{1/2} \tilde{\boldsymbol{f}}^{(H)} = \mathbf{B}^\top \mathbf{W}^{1/2} \mathbf{M}^{(H)} \boldsymbol{z}|_{F_H} = \mathbf{L}^{(H)} \boldsymbol{z}_{F_H} = \boldsymbol{d}^{(H)}$, where the last inequality follows by definition of $\boldsymbol{d}^{(H)}$. ☐

We observe an orthogonality property of the flows, which will become useful later:

Lemma 72. *For any nodes H, H' at the same level in $\mathcal{T}$,* Range$(\mathbf{M}^{(H)})$ *and* Range$(\mathbf{M}^{(H')})$ *are disjoint. Consequently, the flows $\tilde{\boldsymbol{f}}^{(H)}$ and $\tilde{\boldsymbol{f}}^{(H')}$ are orthogonal.*

Proof. Recall leaves of $\mathcal{T}$ correspond to pairwise edge-disjoint, constant-sized regions of the original graph G. 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 $\mathbf{M}^{(H')}$. ☐

Next, we set up the tools for bounding $\left\| \tilde{\boldsymbol{f}} - \mathbf{P}_{\boldsymbol{w}} \boldsymbol{v} \right\|_2$, involving an energy analysis drawing inspiration from electric flow routing. We begin with the canonical definitions and properties of electric-flow energy.

Definition 73. Let $\mathbf{W}^{1/2}\mathbf{B}$ be the edge-weighted incidence matrix of some graph G, and let $\mathbf{L} \stackrel{\text{def}}{=} \mathbf{B}^\top \mathbf{W} \mathbf{B}$ be the Laplacian. Let $\boldsymbol{d} \stackrel{\text{def}}{=} \mathbf{L}\boldsymbol{z}$ be a demand and $\boldsymbol{f}$ be any weighted 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 electric flows, we know $\boldsymbol{f}^\star = \mathbf{W}^{1/2}\mathbf{B}\mathbf{L}^{-1}\boldsymbol{d}$. Hence, we can refer to its energy as the energy of the demand $\boldsymbol{d}$ on the graph of $\mathbf{L}$, given by

$$\mathcal{E}_{\mathbf{L}}(\boldsymbol{d}) \stackrel{\text{def}}{=} \min_{(\mathbf{W}^{1/2}\mathbf{B})^\top \boldsymbol{f} = \boldsymbol{d}} \|\boldsymbol{f}\|_2^2 = \boldsymbol{d}^\top (\mathbf{B}^\top \mathbf{W} \mathbf{B})^{-1} \boldsymbol{d} = \boldsymbol{d}^\top \mathbf{L}^{-1} \boldsymbol{d} = \boldsymbol{z}^\top \mathbf{L} \boldsymbol{z}. \tag{8.3}$$

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 $\boldsymbol{d} \stackrel{\text{def}}{=} \mathbf{L}\boldsymbol{z}$ is a demand on the vertices of G. Then if we decompose $\boldsymbol{d} = \boldsymbol{d}^{(1)} + \boldsymbol{d}^{(2)}$, where $\boldsymbol{d}^{(i)} = \mathbf{L}^{(i)}\boldsymbol{z}$, then the energies are related as:*

$$\mathcal{E}_{\mathbf{L}}(\boldsymbol{d}) = \mathcal{E}_{\mathbf{L}^{(1)}}(\boldsymbol{d}^{(1)}) + \mathcal{E}_{\mathbf{L}^{(2)}}(\boldsymbol{d}^{(2)}).$$

Proof. We have, by definition,

$$\begin{aligned}\mathcal{E}_{\mathbf{L}^{(1)}}(\boldsymbol{d}^{(1)}) + \mathcal{E}_{\mathbf{L}^{(2)}}(\boldsymbol{d}^{(2)}) &= \boldsymbol{z}^\top \mathbf{L}^{(1)} \boldsymbol{z} + \boldsymbol{z}^\top \mathbf{L}^{(2)} \boldsymbol{z} \\ &= \boldsymbol{z}^\top \mathbf{L} \boldsymbol{z} \\ &= \mathcal{E}_{\mathbf{L}}(\boldsymbol{d}).\end{aligned}$$

□

The following lemma shows if G' is a graph derived from G by taking Schur complement on a subset of the vertices C, and $\boldsymbol{d}$ is a demand supported on C, then the flow routing $\boldsymbol{d}$ on G will have lower energy than the flow routing $\boldsymbol{d}$ on G'.

Lemma 75. *Suppose G is a weighted graph with Laplacian $\mathbf{L}$. Let C be a subset of vertices of G. Let $\mathbf{L}' = \widetilde{\mathbf{Sc}}(\mathbf{L}, C)$ be an ε-approximate Schur complement. Then for the demand $\boldsymbol{d} = \mathbf{L}'\boldsymbol{z}$ supported on C,*

$$\mathcal{E}_{\mathbf{L}}(\boldsymbol{d}) \leq_\varepsilon \mathcal{E}_{\mathbf{L}'}(\boldsymbol{d}).$$

Proof. We have, by definition,

$$\begin{aligned}\mathcal{E}_{\mathbf{L}}(\mathbf{L}'\boldsymbol{z}) &= \boldsymbol{z}^\top \mathbf{L}' \mathbf{L}^{-1} \mathbf{L}' \boldsymbol{z} \\ &\leq \boldsymbol{z}^\top \mathbf{L}' \mathbf{Sc}(\mathbf{L}, C)^{-1} \mathbf{L}' \boldsymbol{z} && (\text{since } \mathbf{Sc}(\mathbf{L}, C) \preccurlyeq \mathbf{L}) \\ &\approx_\varepsilon \boldsymbol{z}^\top \mathbf{L}' \mathbf{L}'^{-1} \mathbf{L}' \boldsymbol{z} \\ &= \mathcal{E}_{\mathbf{L}'}(\mathbf{L}'\boldsymbol{z}).\end{aligned}$$

□

For any $H \in \mathcal{T}$, we know $\tilde{\boldsymbol{f}}^{(H)}$ routes $\boldsymbol{d}^{(H)}$ using the original graph G. Furthermore, we know the graph of $\mathbf{L}^{(H)}$ is related to G using the graph operations considered above. Suppose $\boldsymbol{f}^{(H)\star}$ is the energy-minimizing flow routing $\boldsymbol{d}^{(H)}$ on the graph of $\mathbf{L}^{(H)}$. Then we want to relate the energies of $\tilde{\boldsymbol{f}}^{(H)}$ and $\boldsymbol{f}^{(H)\star}$:

Lemma 76. *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 weighted 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, $\left\|\tilde{\boldsymbol{f}}^{(H)}\right\|_2^2 \leq_{i\epsilon_{\mathbf{P}}} \mathcal{E}_{\mathbf{L}^{(H)}}(\boldsymbol{d}^{(H)})$.

Proof. We proceed by induction. 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 Range($\mathbf{M}^{(D_1)}$) and Range($\mathbf{M}^{(D_2)}$) are orthogonal, we have

$$
\begin{aligned}
&= \left\|\mathbf{M}^{(D_1)}\mathbf{M}_{(D_1,H)}\boldsymbol{z}\right\|_2^2 + \left\|\mathbf{M}^{(D_2)}\mathbf{M}_{(D_2,H)}\boldsymbol{z}\right\|_2^2 \\
&\leq_{(i-1)\epsilon_{\mathbf{P}}} \mathcal{E}_{\mathbf{L}^{(D_1)}}\left(\mathbf{L}^{(D_1)}\mathbf{M}_{(D_1,H)}\boldsymbol{z}\right) + \mathcal{E}_{\mathbf{L}^{(D_2)}}\left(\mathbf{L}^{(D_2)}\mathbf{M}_{(D_2,H)}\boldsymbol{z}\right) \\
&\qquad\qquad \text{(by inductive hypothesis with } \boldsymbol{z} = \mathbf{M}_{(D_i,H)}\boldsymbol{z}) \\
&= \mathcal{E}_{\mathbf{L}^{(D_1)}}\left(\mathbf{L}^{(D_1)}(\mathbf{L}^{(D_1)})^{-1}\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)},\partial D_1)\boldsymbol{z}\right) + \mathcal{E}_{\mathbf{L}^{(D_2)}}\left(\mathbf{L}^{(D_2)}(\mathbf{L}^{(D_2)})^{-1}\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)},\partial D_2)\boldsymbol{z}\right) \\
&\leq_{\epsilon_{\mathbf{P}}} \mathcal{E}_{\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)},\partial D_1)}\left(\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_1)},\partial D_1)\boldsymbol{z}\right) + \mathcal{E}_{\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)},\partial D_2)}\left(\widetilde{\mathbf{Sc}}(\mathbf{L}^{(D_2)},\partial D_2)\boldsymbol{z}\right) \\
&= \mathcal{E}_{\mathbf{L}^{(H)}}(\mathbf{L}^{(H)}\boldsymbol{z}),
\end{aligned}
$$

where the last two inequalities follow from Lemmas 74 and 75. □

Next, we want to relate the energy of routing $\boldsymbol{d}^{(H)}$ on the graph G and the energy on the graph of $\mathbf{L}^{(H)}$.

Lemma 77. *For a node H at level i in $\mathcal{T}$,*

$$\mathcal{E}_{\mathbf{L}}(\boldsymbol{d}^{(H)}) \approx_{\epsilon_{\mathbf{P}}} \mathcal{E}_{\mathbf{L}^{(H)}}(\boldsymbol{d}^{(H)}).$$

□

Proof. For one direction, we have

$$
\begin{aligned}
\mathcal{E}_{\mathbf{L}^{(H)}}(\boldsymbol{d}^{(H)}) &= {\boldsymbol{d}^{(H)}}^{\top}{\mathbf{L}^{(H)}}^{-1}\boldsymbol{d}^{(H)} \\
&\approx_{\epsilon_{\mathbf{P}}} {\boldsymbol{d}^{(H)}}^{\top}\mathbf{Sc}(\mathbf{L}[H],\partial H \cup F_H)^{-1}\boldsymbol{d}^{(H)} && \text{(by Theorem 6)} \\
&\leq {\boldsymbol{d}^{(H)}}^{\top}\mathbf{L}^{-1}\boldsymbol{d}^{(H)} && (\text{since } \mathbf{Sc}(\mathbf{L},C) \succcurlyeq \mathbf{L}) \\
&= \mathcal{E}_{\mathbf{L}}(\boldsymbol{d}^{(H)}).
\end{aligned}
$$

In the other direction, we note that $\tilde{\boldsymbol{f}}^{(H)}$ is a weighted flow routing $\boldsymbol{d}^{(H)}$ on G. By Lemma 76 and the definition of energy,

$$\mathcal{E}_{\mathbf{L}}(\boldsymbol{d}^{(H)}) \leq \left\|\tilde{\boldsymbol{f}}^{(H)}\right\|_2^2 \approx_{i\epsilon_{\mathbf{P}}} \mathcal{E}_{\mathbf{L}^{(H)}}(\boldsymbol{d}^{(H)}).$$

□

We need to further bound the sum of energies:

Lemma 78. *We have the following approximation of the energy of $\boldsymbol{d}$ on graph G:*

$$\sum_{H\in\mathcal{T}} \mathcal{E}_{\mathbf{L}^{(H)}}(\boldsymbol{d}^{(H)}) \approx_{\eta\epsilon_{\mathbf{P}}} \mathcal{E}_{\mathbf{L}}(\boldsymbol{d}).$$

Proof. We need the following matrix multiplication property: For any matrices $\mathbf{A},\mathbf{B},\mathbf{D}$,

$$\begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^{\top} & \mathbf{D} \end{bmatrix}\begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{8.4}$$

Recall in our setting, all matrices are padded with zeros so that their dimension is $n \times n$, and vectors padded with zeros so their dimension is n.

Define $\boldsymbol{\beta} \stackrel{\text{def}}{=} \boldsymbol{\Pi}^{(\eta-1)} \cdots \boldsymbol{\Pi}^{(1)}\boldsymbol{\Pi}^{(0)}\boldsymbol{d}$ for simplicity. Recall $\boldsymbol{z} \stackrel{\text{def}}{=} \widetilde{\boldsymbol{\Gamma}}\boldsymbol{\Pi}^{(\eta-1)} \cdots \boldsymbol{\Pi}^{(0)}\mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}$. We can write

$$\boldsymbol{z}|_{F_H} = \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1}\boldsymbol{\beta}.$$

Then,

$$\begin{aligned}
\mathcal{E}_{\mathbf{L}^{(H)}}(\boldsymbol{d}^{(H)}) &= \boldsymbol{z}^\top|_{F_H}\mathbf{L}^{(H)}\boldsymbol{z}|_{F_H} \\
&= \boldsymbol{\beta}^\top\left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1}\mathbf{L}^{(H)}\left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1}\boldsymbol{\beta} \\
&= \boldsymbol{\beta}^\top\left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1}\boldsymbol{\beta}. && \text{(by Eq. (8.4))}
\end{aligned}$$

Summing over all $H \in \mathcal{T}$, we get

$$\begin{aligned}
\sum_{H\in\mathcal{T}}\mathcal{E}_{\mathbf{L}^{(H)}}(\boldsymbol{d}^{(H)}) &= \boldsymbol{\beta}^\top\sum_{H\in\mathcal{T}}(\mathbf{L}^{(H)}_{F_H,F_H})^{-1}\boldsymbol{\beta} \\
&= \boldsymbol{d}^\top\boldsymbol{\Pi}^{(0)\top}\cdots\boldsymbol{\Pi}^{(\eta-1)\top}\left[\sum_H(\mathbf{L}^{(H)}_{F_H,F_H})^{-1}\right]\boldsymbol{\Pi}^{(\eta-1)}\cdots\boldsymbol{\Pi}^{(0)}\boldsymbol{d} \\
&\approx_{\eta\epsilon_{\mathbf{P}}} \boldsymbol{d}^\top\mathbf{L}^{-1}\boldsymbol{d} \\
&= \mathcal{E}_{\mathbf{L}}(\boldsymbol{d}).
\end{aligned}$$

where the last second step follows by Theorem 33. □

Lastly, the following lemma shows that our weighted flow $\tilde{\boldsymbol{f}}$ routing $\boldsymbol{d}$ can be orthogonally decomposed in terms of the unique energy minimizer $\boldsymbol{f}^\star$, which in turn allows us to bound $\|\tilde{\boldsymbol{f}}-\boldsymbol{f}^\star\|_2^2$.

Lemma 79. *Let $\mathbf{L}$ be a weighted Laplacian as above, and let $\boldsymbol{d}$ be a demand. Let $\boldsymbol{f}^\star = \mathbf{W}^{1/2}\mathbf{B}\mathbf{L}^{-1}\boldsymbol{d}$ be the weighted electric flow routing $\boldsymbol{d}$ attaining the minimum energy $\mathcal{E}_{\mathbf{L}}(\boldsymbol{d})$. For any other weighted flow $\tilde{\boldsymbol{f}}$ satisfying $\mathbf{B}^\top\mathbf{W}^{1/2}\tilde{\boldsymbol{f}} = \boldsymbol{d}$, if $\|\tilde{\boldsymbol{f}}\|_2^2 \le_\varepsilon \mathcal{E}_{\mathbf{L}}(\boldsymbol{d})$, then*

$$\|\tilde{\boldsymbol{f}} - \boldsymbol{f}^\star\|_2^2 \le (e^\varepsilon - 1)\,\|\boldsymbol{f}^\star\|_2^2\,.$$

Proof. Observe that

$$\boldsymbol{f}^{\star\top}(\tilde{\boldsymbol{f}} - \boldsymbol{f}^\star) = \boldsymbol{d}^\top\mathbf{L}^{-1}\mathbf{B}^\top\mathbf{W}^{1/2}(\tilde{\boldsymbol{f}} - \boldsymbol{f}^\star) = \boldsymbol{d}^\top\mathbf{L}^{-1}(\boldsymbol{d} - \boldsymbol{d}) = \mathbf{0}.$$

Hence, we have an orthogonal decomposition of $\tilde{\boldsymbol{f}}$:

$$\|\tilde{\boldsymbol{f}}\|_2^2 = \|\tilde{\boldsymbol{f}}^\star\|_2^2 + \|\tilde{\boldsymbol{f}} - \boldsymbol{f}^\star\|_2^2.$$

It follows that

$$\|\boldsymbol{f} - \boldsymbol{f}^\star\|^2 \le (e^\varepsilon - 1)\cdot\|\boldsymbol{f}^*\|_2^2\,.$$

□

Finally, we put all the lemmas together for the overall proof that $\tilde{\boldsymbol{f}}$ is the desired weighted flow.

Proof of Theorem 67. We first decompose $\boldsymbol{d} = \sum_{H\in\mathcal{T}}\boldsymbol{d}^{(H)}$ according to Lemma 69. 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 $\tilde{\boldsymbol{f}}^{(H)} \stackrel{\text{def}}{=} \mathbf{M}^{(H)}\boldsymbol{z}|_{F_H}$ routes demand $\boldsymbol{d}^{(H)}$ by Lemma 71. Hence,

$$(\mathbf{W}^{1/2}\mathbf{B})^\top \tilde{\boldsymbol{f}} = \sum_{H\in\mathcal{T}} (\mathbf{W}^{1/2}\mathbf{B})^\top \tilde{\boldsymbol{f}}^{(H)} = \sum_{H\in\mathcal{T}} \boldsymbol{d}^{(H)} = \boldsymbol{d},$$

meaning $\tilde{\boldsymbol{f}}$ is feasible for routing $\boldsymbol{d}$ on G.

For each demand term $\boldsymbol{d}^{(H)}$, let $\boldsymbol{f}^{(H)\star}$ be the weighted flow on G that attains the minimum energy $\mathcal{E}_{\mathbf{L}}(\boldsymbol{d}^{(H)})$ for routing it. By Definition 73 , $\boldsymbol{f}^{(H)\star} = \mathbf{W}^{1/2}\mathbf{B}\mathbf{L}^{-1}\boldsymbol{d}^{(H)}$. Recall $\boldsymbol{f}^\star \stackrel{\text{def}}{=} \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v} = \mathbf{W}^{1/2}\mathbf{B}\mathbf{L}^{-1}\boldsymbol{d}$. Hence,

$$\boldsymbol{f}^\star = \sum_{H\in\mathcal{T}} \boldsymbol{f}^{(H)\star}.$$

By Lemma 77, we know if H is at level i in $\mathcal{T}$, then $\tilde{\boldsymbol{f}}^{(H)}$ satisfies

$$\left\|\tilde{\boldsymbol{f}}^{(H)}\right\|_2^2 \leq_{i\epsilon_{\mathbf{P}}} \mathcal{E}_{\mathbf{L}^{(H)}}(\boldsymbol{d}^{(H)}) \approx_{i\epsilon_{\mathbf{P}}} \mathcal{E}_{\mathbf{L}}(\boldsymbol{d}^{(H)}) = \left\|\boldsymbol{f}^{(H)\star}\right\|_2^2. \tag{8.5}$$

This shows that in the flow tree operator, the output $\tilde{\boldsymbol{f}}^{(H)}$ of each tree operator $\mathbf{M}^{(H)}$ is close to the natural corresponding term $\boldsymbol{f}^{(H)\star}$. Finally, we bound the overall approximation error:

$$\begin{aligned}
\left\|\tilde{\boldsymbol{f}} - \boldsymbol{f}^\star\right\|_2^2 &= \left\|\sum_{H\in\mathcal{T}} \left(\tilde{\boldsymbol{f}}^{(H)} - \boldsymbol{f}^{(H)\star}\right)\right\|_2^2 \\
&\leq \left(\sum_{H\in\mathcal{T}} \left\|\tilde{\boldsymbol{f}}^{(H)} - \boldsymbol{f}^{(H)\star}\right\|_2\right)^2 \\
&= \sum_{i=0}^{\eta} \sum_{H\in\mathcal{T}(i)} (e^{2i\epsilon_{\mathbf{P}}} - 1)\mathcal{E}_{\mathbf{L}}(\boldsymbol{d}^{(H)}) && \text{(by Lemma 79 and Eq. (8.5))} \\
&\leq \sum_{i=0}^{\eta} \sum_{H\in\mathcal{T}(i)} (e^{2i\epsilon_{\mathbf{P}}} - 1)e^{i\epsilon_{\mathbf{P}}}\mathcal{E}_{\mathbf{L}^{(H)}}(\boldsymbol{d}^{(H)}) && \text{(by Lemma 77)} \\
&\leq e^{4\eta\epsilon_{\mathbf{P}}} \sum_{H\in\mathcal{T}} \mathcal{E}_{\mathbf{L}}(\boldsymbol{d}) && \text{(by Lemma 78)} \\
&= O(\eta\epsilon_{\mathbf{P}}) \left\|\boldsymbol{f}^\star\right\|^2,
\end{aligned}$$

which concludes the overall proof. □

8.2 Proof of Theorem 10

Finally, we present the overall flow maintenance data structure. It is analogous to slack, except during each MOVE operation, there is an additional term of $\alpha\mathbf{W}^{1/2}\boldsymbol{v}$.

Theorem 10 (Flow maintenance)**.** *Given a modified planar graph G with m edges and its separator tree $\mathcal{T}$ with height η, the randomized 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$(G, \boldsymbol{f}^{(\text{init})} \in \mathbb{R}^m, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \epsilon_{\mathbf{P}} > 0, \overline{\epsilon} > 0)$*: Given a graph G, 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($\boldsymbol{w} \in \mathbb{R}^m_{>0}$ *given implicitly as a set of changed weights): Set the current weights to* $\boldsymbol{w}$ *in* $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK})$ *time, where* K *is the number of coordinates changed in* $\boldsymbol{w}$*.*

- MOVE($\alpha \in \mathbb{R}, \boldsymbol{v} \in \mathbb{R}^m$ *given implicitly as a set of changed coordinates): 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* $\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v}$*, where* $\|\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v} - \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}\|_2 \leq O(\eta\epsilon_{\mathbf{P}})\|\boldsymbol{v}\|_2$ *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 *is the number of coordinates changed in* $\boldsymbol{v}$*.*

- APPROXIMATE() $\to \mathbb{R}^m$*: Output the vector* $\overline{\boldsymbol{f}}$ *such that* $\|\mathbf{W}^{-1/2}(\overline{\boldsymbol{f}} - \boldsymbol{f})\|_\infty \leq \overline{\epsilon}$ *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 $\alpha\|\boldsymbol{v}\|_2 \leq \beta$ *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* k*-th* REWEIGHT *and* MOVE *calls. Then at the* k*-th* APPROXIMATE *call,*

- *the data structure first sets* $\overline{\boldsymbol{f}}_e \leftarrow \boldsymbol{f}_e^{(k-1)}$ *for all coordinates* e *where* $\boldsymbol{w}_e$ *changed in the last* REWEIGHT*, then sets* $\overline{\boldsymbol{f}}_e \leftarrow \boldsymbol{f}_e^{(k)}$ *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* k*-th* APPROXIMATE *call is* $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{m(K + N_{k-2^{\ell_k}})})$*.*

Proof of Theorem 10. We have the additional invariant that the IPM flow solution $\boldsymbol{f}$ can be recovered in the data structure by the identity

$$\boldsymbol{f} = \hat{\boldsymbol{f}} - \boldsymbol{f}^\perp, \tag{8.6}$$

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 61, 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 $\boldsymbol{f}^\perp$ in `maintainRep` 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 `bar_f`, $\overline{\boldsymbol{f}}$ is set to $\boldsymbol{f} = \boldsymbol{f}^{(\text{init})}$ to start.

Initialization of `maintainRep` takes $\widetilde{O}(m{\epsilon_{\mathbf{P}}}^{-2})$ time by Theorem 7, and the initialization of `bar_f` takes $\widetilde{O}(m)$ time by Theorem 8.

REWEIGHT: The change to the representation in $\boldsymbol{f}^\perp$ is correct via `maintainRep` in exactly the same manner as the proof for the slack solution. For the representation of $\hat{\boldsymbol{f}}$, the change in value caused by the update to $\boldsymbol{w}$ is subtracted from the $\hat{\boldsymbol{f}}_0$ term, so that the representation is updated while the overall value remains the same.

Algorithm 10 Flow Maintenance, Main Algorithm

data structure MAINTAINFLOW **extends** MAINTAINZ
private: member
  $\boldsymbol{w} \in \mathbb{R}^m$: weight vector ▷ we use the diagonal matrix $\mathbf{W}$ interchangeably
  $\boldsymbol{v} \in \mathbb{R}^m$: direction vector
  MAINTAINREP `maintainRep`: data structure to implicitly maintain

$$\boldsymbol{f}^\perp \stackrel{\text{def}}{=} \boldsymbol{y} + \mathbf{W}^{1/2}\mathbf{M}(c\boldsymbol{z}^{(\text{step})} + \boldsymbol{z}^{(\text{sum})}).$$

▷ $\mathbf{M}$ is defined by Definition 65
  $\hat{c} \in \mathbb{R}, \hat{\boldsymbol{f}}_0 \in \mathbb{R}^m$: scalar and vector to implicitly maintain

$$\hat{\boldsymbol{f}} \stackrel{\text{def}}{=} \hat{\boldsymbol{f}}_0 + \hat{c} \cdot \mathbf{W}\boldsymbol{v}.$$

  MAINTAINAPPROX `bar_f`: data structure to maintain approximation $\overline{\boldsymbol{f}}$ to $\boldsymbol{f}$ (Theorem 8)

procedure INITIALIZE($G, \boldsymbol{f}^{(\text{init})} \in \mathbb{R}^m, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \epsilon_{\mathbf{P}} > 0, \overline{\epsilon} > 0$)
  Build the separator tree $\mathcal{T}$ by Theorem 25
  `maintainRep`.INITIALIZE($G, \mathcal{T}, \mathbf{W}^{1/2}\mathbf{M}, \boldsymbol{v}, \boldsymbol{w}, \mathbf{0}, \epsilon_{\mathbf{P}}$) ▷ 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 $\hat{\boldsymbol{f}} \leftarrow \boldsymbol{f}^{(\text{init})}$
  `bar_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($\boldsymbol{w}^{(\text{new})} \in \mathbb{R}^m_{>0}$)
  `maintainRep`.REWEIGHT($\boldsymbol{w}^{(\text{new})}$)
  $\Delta\boldsymbol{w} \leftarrow \boldsymbol{w}^{(\text{new})} - \boldsymbol{w}$
  $\boldsymbol{w} \leftarrow \boldsymbol{w}^{(\text{new})}$
  $\hat{\boldsymbol{f}}_0 \leftarrow \hat{\boldsymbol{f}}_0 - \hat{c}(\Delta\mathbf{W})^{1/2}\boldsymbol{v}$
end procedure

procedure MOVE($\alpha, \boldsymbol{v}^{(\text{new})} \in \mathbb{R}^m$)
  `maintainRep`.MOVE($\alpha, \boldsymbol{v}^{(\text{new})}$)
  $\Delta\boldsymbol{v} \leftarrow \boldsymbol{v}^{(\text{new})} - \boldsymbol{v}$
  $\boldsymbol{v} \leftarrow \boldsymbol{v}^{(\text{new})}$
  $\hat{\boldsymbol{f}}_0 \leftarrow \hat{\boldsymbol{f}}_0 - \hat{c}\mathbf{W}^{1/2}\Delta\boldsymbol{v}$
  $\hat{c} \leftarrow \hat{c} + \alpha$
end procedure

procedure APPROXIMATE( )
  ▷ the variables in the argument are accessed from `maintainRep`
  return `bar_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( )
  $\boldsymbol{f}^\perp \leftarrow$ `maintainRep`.EXACT()
  return $(\hat{\boldsymbol{f}}_0 + \hat{c} \cdot \mathbf{W}\boldsymbol{v}) - \boldsymbol{f}^\perp$
end procedure

Move: This is similar to the proof for the slack solution. `maintainRep`.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 65. By Lemma 59, this is equivalent to the update

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

where $\left\|\tilde{\boldsymbol{f}} - \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}^{(k)}\right\|_2 \le O(\eta\epsilon_{\mathbf{P}})\left\|\boldsymbol{v}^{(k)}\right\|_2$ and $\mathbf{B}^\top\mathbf{W}^{1/2}\tilde{\boldsymbol{f}} = \mathbf{B}^\top\mathbf{W}^{1/2}\boldsymbol{v}^{(k)}$ by Theorem 67.

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

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

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

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

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

Approximate: By the guarantee of `bar_f`.Approximate from Theorem 8, the returned vector satisfies $\|\mathbf{W}^{-1/2}\left(\overline{\boldsymbol{f}} - (\hat{\boldsymbol{f}} - \boldsymbol{f}^\perp)\right)\|_\infty \le \overline{\epsilon}$, where $\hat{\boldsymbol{f}}$ and $\boldsymbol{f}^\perp$ are maintained in the current data structure.

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

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

Lemma 80. *Suppose $\alpha\|\boldsymbol{v}\|_2 \le \beta$ 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 k-th* Reweight *and* Move *calls. Then at the k-th* Approximate *call,*

- *The data structure first sets $\overline{\boldsymbol{f}}_e \leftarrow \boldsymbol{f}_e^{(k-1)}$ for all coordinates e where $\boldsymbol{w}_e$ changed in the last* Reweight*, then sets $\overline{\boldsymbol{f}}_e \leftarrow \boldsymbol{f}_e^{(k)}$ 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 $k \ne 0$ and $\ell_0 = 0$.*
- *The amortized time for the k-th* Approximate *call is $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{m(K + N_{k-2^{\ell_k}})})$.*

Proof. The proof is similar to the one for slack. Since $\overline{\boldsymbol{f}}$ is maintained by `bar_f`, we apply Theorem 8 with $\boldsymbol{x} = \overline{\boldsymbol{f}}$ and diagonal matrix $\mathbf{D} = \mathbf{W}^{-1}$. We need to prove $\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\|_{\mathbf{D}^{(k)}} \le O(\beta)$ for all k first. The constant factor in $O(\beta)$ does not affect the guarantees in Theorem 8. The left-hand side is

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

Now, we can apply the conclusions from Theorem 8 to get that at the k-th step, the data structure first sets $\overline{\boldsymbol{f}}_e \leftarrow \boldsymbol{f}_e^{(k-1)}$ for all coordinates e where $\boldsymbol{w}_e$ changed in the last REWEIGHT, then sets $\overline{\boldsymbol{f}}_e \leftarrow \boldsymbol{f}_e^{(k)}$ for $O(N_k \stackrel{\text{def}}{=} 2^{2\ell_k}(\frac{\beta}{\epsilon})^2 \log^2 m)$ 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 $\boldsymbol{z}^{(\text{step})}$ and $\boldsymbol{z}^{(\text{sum})}$ on F_H for $\widetilde{O}(K)$ different nodes $H \in \mathcal{T}$ by Theorem 7. REWEIGHT then updates $\boldsymbol{z}^{(\text{step})}$ and $\boldsymbol{z}^{(\text{sum})}$ 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. □

□

9 Min-Cost Flow for Separable Graphs

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

Corollary 2 (Separable min-cost flow)**.** *Let $\mathcal{C}$ be an α-separable graph class such that we can compute a balanced separator for any graph in $\mathcal{C}$ with m edges in $s(m)$ time for some convex function s. Given a graph $G \in \mathcal{C}$ with n vertices and m edges, integer demands $\boldsymbol{d}$, edge capacities $\boldsymbol{u}$ and 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 $\widetilde{O}((m + m^{1/2+\alpha}) \log M + s(m))$ expected time.*

The change in running time essentially comes from the parameters of the separator tree which we shall discuss in Section 9.1. We then calculate the total running time and prove Corollary 2 in Section 9.2.

9.1 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 19 increased by 2.

Recall the definition of separable graphs:

Definition 19 (Separable graph)**.** A graph $G = (V, E)$ is α-separable if there exists two constants $c > 0$ and $b \in (0, 1)$ such that every nonempty subgraph $H = (V(H) \subseteq V, E(H) \subseteq E)$ with $|E(H)| \geq 2$ of G can be partitioned into H_1 and H_2 such that

- $E(H_1) \cup E(H_2) = E(H)$, $E(H_1) \cap E(H_2) = \emptyset$,
- $|V(H_1) \cap V(H_2)| \leq c\lceil |E(H)|^\alpha \rceil$,
- $|E(H_i)| \leq b|E(H)|$, for $i = 1, 2$.

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

We define a separator tree $\mathcal{T}$ for an α-separable graph G in the same way as for a planar graph.

Definition 81 (Separator tree $\mathcal{T}$ for α-separable graph)**.** Let G be an α-separable graph. A separator tree $\mathcal{T}$ 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, and their vertex sets intersect on the balanced separator $S(H)$ of H. 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. Further, 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 $\eta(H) = i$.

The only two differences between the separator trees for planar and α-separable graphs are their construction time and update time (for k-sparse updates). For the planar case, these are bounded by Theorem 25 and Lemma 28 respectively. We shall prove their analogs Lemma 82 and Lemma 83.

[GHP18] 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 the extended proofs in Appendix A for completeness.

Lemma 82. *Let $\mathcal{C}$ be an α-separable class such that we can compute a balanced separator for any graph in $\mathcal{C}$ with n vertices and m edges in $s(m)$ time for some convex function $s(m) \geq m$. Given an α-separable graph, there is an algorithm that computes a separator tree $\mathcal{T}$ in $O(s(m) \log m)$ time.*

Note that $s(\cdot)$ does not depend on n because we may assume the graph is connected so that $n = O(m)$.

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 83. *Let G be an α-separable graph with separator tree $\mathcal{T}$. 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 28 for planar graphs as a corollary.

9.2 Proof of Running time

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.

For the planar case, after constructing the separator tree by Theorem 25, Lemma 28 is the lemma that interacts with other parts of the algorithm. For α-separable graphs, we first construct the separator tree in $O(s(m)\log m)$ time by Lemma 82. Then we propagate the change in runtime ($\widetilde{O}(\sqrt{mK})$ from Lemma 28 to $\widetilde{O}(m^{\alpha}K^{1-\alpha})$ from Lemma 83) to all the data structures and to the complexity $T(\cdot)$ of the flow and slack tree operators.

We first propagate the change to the implicit representation maintenance data structure, which is the common component for maintaining the flow and the slack vectors.

Theorem 84. *Given an α-separable graph G with n vertices and m edges, and its separator tree $\mathcal{T}$ 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 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,*

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

  where $\alpha^{(i)}$ is the step size α given in Move *for step i,*
- *$\boldsymbol{z}^{(\text{step})}$ satisfies $\boldsymbol{z}^{(\text{step})} = \widetilde{\boldsymbol{\Gamma}}\boldsymbol{\Pi}^{(\eta-1)} \cdots \boldsymbol{\Pi}^{(0)}\mathbf{B}^{\top}\mathbf{W}^{1/2}\boldsymbol{v}$,*
- *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$(G, \mathcal{T}, \mathbf{M}, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \boldsymbol{x}^{(\text{init})} \in \mathbb{R}^m, \epsilon_{\mathbf{P}} > 0)$*: Given a graph G, its separator tree $\mathcal{T}$, 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 projection matrix 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$(\boldsymbol{w} \in \mathbb{R}^m_{>0}$ *given implicitly as a set of changed coordinates): Update the weights to $\boldsymbol{w}^{(\text{new})}$. 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 $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}K^{1-\alpha}m^{\alpha} + T(K))$ total time, where K is an upper bound on the number of coordinates changed in $\boldsymbol{w}$ 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($\alpha \in \mathbb{R}$, $\boldsymbol{v} \in \mathbb{R}^n$ *given implicitly as a set of changed coordinates): Update the current direction to* $\boldsymbol{v}$*, and then* $\boldsymbol{z}^{(\text{step})}$ *to maintain the claimed invariant. 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 $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2} K^{1-\alpha} m^{\alpha})$ *time, where* K *is the number of coordinates changed in* $\boldsymbol{v}$ *compared to the previous IPM step.*

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

Proof. The bottlenecks of MOVE is PARTIALPROJECT. 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 an exact 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. By Lemma 83, PARTIALPROJECT takes $\widetilde{O}({\epsilon_{\mathbf{P}}}^{-2}K^{1-\alpha}m^{\alpha})$ time. MOVE also runs in $\widetilde{O}({\epsilon_{\mathbf{P}}}^{-2}K^{1-\alpha}m^{\alpha})$ time.

REWEIGHT calls PARTIALPROJECT and REVERSEPARTIALPROJECT for $O(1)$ times and COMPUTEMZ once. REVERSEPARTIALPROJECT costs the same as PARTIALPROJECT. The runtime of COMPUTEMZ is still bounded by the complexity of the tree operator, $O(T(K))$. Thus, PARTIALPROJECT takes $\widetilde{O}({\epsilon_{\mathbf{P}}}^{-2}K^{1-\alpha}m^{\alpha})$ time. MOVE also runs in $\widetilde{O}({\epsilon_{\mathbf{P}}}^{-2}K^{1-\alpha}m^{\alpha} + T(K))$ time.

Runtimes of other procedures and correctness follow from the same argument as in the proof for Theorem 7. □

Then we may use Theorem 84 and Theorem 8 to maintain vectors $\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}$, with the updated complexity of the operators.

Lemma 85. *For any* α*-separable graph* G *with separator tree* $\mathcal{T}$*, the flow and slack operators defined in Definitions 58 and 65 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 $\mathbf{M}_{(D,P)} = \mathbf{I}_{\partial D} - \left(\mathbf{L}^{(D)}_{F_D,F_D}\right)^{-1}\mathbf{L}^{(D)}_{F_D,\partial D}$ to the left or right consists of three steps which are applying $\mathbf{I}_{\partial D}$, applying $\mathbf{L}^{(D)}_{F_D,\partial D}$ and solving for $\mathbf{L}^{(D)}_{F_D,F_D}\boldsymbol{v} = \boldsymbol{b}$ for some vectors $\boldsymbol{v}$ and $\boldsymbol{b}$. 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 34 and Theorem 12. To bound the total cost over K distinct edges, we apply Lemma 83 instead of Lemma 28, which gives the claimed complexity. □

We then have the following lemmas for maintaining the flow and slack vectors:

Theorem 86 (Slack maintenance for α-separable graphs)**.** *Given a modified planar graph* G *with* n *vertices and* m *edges, and its separator tree* $\mathcal{T}$ *with height* η*, the randomized data structure* MAINTAINSLACK *(Algorithm 9) implicitly maintains the slack solution* $\boldsymbol{s}$ *undergoing IPM changes, and explicitly maintains its approximation* $\overline{\boldsymbol{s}}$*, and supports the following procedures with high probability against an adaptive adversary:*

- INITIALIZE($G, \boldsymbol{s}^{(\text{init})} \in \mathbb{R}^m, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \epsilon_{\mathbf{P}} > 0, \overline{\epsilon} > 0$)*: Given a graph* G*, initial solution* $\boldsymbol{s}^{(\text{init})}$*, 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 representations* $\boldsymbol{s} \leftarrow \boldsymbol{s}^{(\text{init})}$ *and* $\overline{\boldsymbol{x}} \leftarrow \boldsymbol{s}$*.*

- REWEIGHT($\boldsymbol{w} \in \mathbb{R}^m_{>0}$, *given implicitly as a set of changed weights): Set the current weights to* $\boldsymbol{w}$ *in* $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2} K^{1-\alpha} m^{\alpha})$ *time, where* K *is the number of coordinates changed in* $\boldsymbol{w}$.

- MOVE($t \in \mathbb{R}, \boldsymbol{v} \in \mathbb{R}^m$ *given implicitly as a set of changed coordinates): Implicitly update* $\boldsymbol{s} \leftarrow \boldsymbol{s} + t\mathbf{W}^{-1/2}\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v}$ *for some* $\widetilde{\mathbf{P}}_{\boldsymbol{w}}$ *with* $\|(\widetilde{\mathbf{P}}_{\boldsymbol{w}} - \mathbf{P}_{\boldsymbol{w}})\boldsymbol{v}\|_2 \leq \eta\epsilon_{\mathbf{P}} \|\boldsymbol{v}\|_2$, *and* $\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v} \in \mathrm{Range}(\mathbf{B})$. *The total runtime is* $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2} K^{1-\alpha} m^{\alpha})$ *where* K *is the number of coordinates changed in* $\boldsymbol{v}$.

- APPROXIMATE() $\to \mathbb{R}^m$: *Return the vector* $\overline{\boldsymbol{s}}$ *such that* $\|\mathbf{W}^{1/2}(\overline{\boldsymbol{s}} - \boldsymbol{s})\|_\infty \leq \overline{\epsilon}$ *for the current weight* $\boldsymbol{w}$ *and the current vector* $\boldsymbol{s}$.

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

Suppose $t\|\boldsymbol{v}\|_2 \leq \beta$ *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* k*-th* REWEIGHT *and* MOVE *calls. Then at the* k*-th* APPROXIMATE *call,*

- *the data structure first sets* $\overline{\boldsymbol{s}}_e \leftarrow \boldsymbol{s}_e^{(k-1)}$ *for all coordinates* e *where* $\boldsymbol{w}_e$ *changed in the last* REWEIGHT, *then sets* $\overline{\boldsymbol{s}}_e \leftarrow \boldsymbol{s}_e^{(k)}$ *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* k*-th* APPROXIMATE *call is* $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}(m^\alpha(K + N_{k-2^{\ell_k}})^{1-\alpha}))$.

Proof. Because $T(m) = \widetilde{O}({\epsilon_{\mathbf{P}}}^{-2}m)$ (Lemma 85), the runtime of INITIALIZE is still $\widetilde{O}({\epsilon_{\mathbf{P}}}^{-2}m)$ by Theorem 84 and Theorem 8. The runtime of REWEIGHT, MOVE, and EXACT follow from the guarantees of Theorem 84. The runtime of APPROXIMATE follows from Theorem 8 with $T(K) = O(K^{1-\alpha}m^\alpha)$ (Lemma 85). □

Theorem 87 (Flow maintenance for α-separable graphs)**.** *Given a* α*-separable graph* G *with* n *vertices and* m *edges, and its separator tree* $\mathcal{T}$ *with height* η, *the randomized 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($G, \boldsymbol{f}^{(\text{init})} \in \mathbb{R}^m, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \epsilon_{\mathbf{P}} > 0, \overline{\epsilon} > 0$): *Given a graph* G, *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($\boldsymbol{w} \in \mathbb{R}^m_{>0}$ *given implicitly as a set of changed weights): Set the current weights to* $\boldsymbol{w}$ *in* $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\alpha)$ *time, where* K *is the number of coordinates changed in* $\boldsymbol{w}$.

- MOVE($t \in \mathbb{R}, \boldsymbol{v} \in \mathbb{R}^m$ *given implicitly as a set of changed coordinates): Implicitly update* $\boldsymbol{f} \leftarrow \boldsymbol{f} + t\mathbf{W}^{1/2}\boldsymbol{v} - t\mathbf{W}^{1/2}\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v}$ *for some* $\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v}$, *where* $\|\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v} - \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}\|_2 \leq O(\eta\epsilon_{\mathbf{P}}) \|\boldsymbol{v}\|_2$ *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* $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}K^{1-\alpha}m^\alpha)$, *where* K *is the number of coordinates changed in* $\boldsymbol{v}$.

- APPROXIMATE() $\to \mathbb{R}^m$: *Output the vector* $\overline{\boldsymbol{f}}$ *such that* $\|\mathbf{W}^{-1/2}(\overline{\boldsymbol{f}} - \boldsymbol{f})\|_\infty \leq \overline{\epsilon}$ *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 $t\|\boldsymbol{v}\|_2 \le \beta$ 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 k-th* Reweight *and* Move *calls. Then at the k-th* Approximate *call,*

- *the data structure first sets $\overline{\boldsymbol{f}}_e \leftarrow \boldsymbol{f}_e^{(k-1)}$ for all coordinates e where $\boldsymbol{w}_e$ changed in the last* Reweight, *then sets $\overline{\boldsymbol{f}}_e \leftarrow \boldsymbol{f}_e^{(k)}$ for $O(N_k \stackrel{\text{def}}{=} 2^{2\ell_k}(\frac{\beta}{\epsilon})^2 \log^2 m)$ 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 k-th* Approximate *call is $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}(m^\alpha(K + N_{k-2^{\ell_k}})^{1-\alpha}))$.*

The proof is the same as Theorem 86.

Finally, 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 use the data structure runtimes given in Theorem 86 and Theorem 87. 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 k-th IPM step is

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

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

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

$$\begin{aligned} O(m^\alpha \log m) \sum_{k=1}^{T} (N_{k-2^{\ell_k}})^{1-\alpha} &= O(m^\alpha \log^2 m) \sum_{k=1}^{T} 2^{2(1-\alpha)\ell_{(k-2^{\ell_k})}} \\ &= O(m^\alpha \log^2 m) \sum_{k'=1}^{T} 2^{2(1-\alpha)\ell_{k'}} \sum_{k=1}^{T} [k - 2^{\ell_k} = k'], \\ &= O(m^\alpha \log^2 m \log T) \sum_{k'=1}^{T} 2^{2(1-\alpha)\ell_{k'}}. \end{aligned} \tag{9.1}$$

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 Eq. (9.1) is

$$\begin{aligned} \sum_{k=1}^{T} 2^{2(1-\alpha)\ell_k} &= \sum_{k=1}^{T} (2^{\ell_k})^{2-2\alpha} \\ &\le \left(\sum_{k=1}^{T} 1^{1/(2\alpha-1)} \right)^{2\alpha-1} \left(\sum_{k=1}^{T} \left(\left(2^{\ell_k}\right)^{2-2\alpha} \right)^{1/(2-2\alpha)} \right)^{2-2\alpha} && \text{(by Hölder's Inquality)} \\ &= \widetilde{O}\left(T^{2\alpha-1} (T \log T)^{2-2\alpha} \right) \\ &= \widetilde{O}(\sqrt{m} \log M \log T), \end{aligned}$$

where we use $T = O(\sqrt{m} \log n \log(nM))$ from Theorem 3. So the runtime for CenteringImpl is $\widetilde{O}(m^{1/2+\alpha} \log M)$. By Lemma 82, the overall runtime is $\widetilde{O}(m^{1/2+\alpha} \log M + s(m))$. ∎

References

[ABKS21] Deeksha Adil, Brian Bullins, Rasmus Kyng, and Sushant Sachdeva. Almost-Linear-Time Weighted ℓ_p-norm Solvers in Slightly Dense Graphs via Sparsification. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198, pages 9:1–9:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. 6

[AKLR18] Mudabir Kabir Asathulla, Sanjeev Khanna, Nathaniel Lahn, and Sharath Raghvendra. A Faster Algorithm for Minimum-Cost Bipartite Perfect Matching in Planar Graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 457–476. SIAM, 2018. 5

[AMO88] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network Flows*. Prentice Hall, 1988. 4

[AMV20] Kyriakos Axiotis, Aleksander Mądry, and Adrian Vladu. Circulation control for faster minimum cost flow in unit-capacity graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 93–104. IEEE Computer Society, 2020. 4, 6, 7

[AS20] Deeksha Adil and Sushant Sachdeva. Faster p-norm minimizing flows, via smoothed q-norm problems. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 892–910. SIAM, 2020. 6

[BGS21] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental SSSP and approximate min-cost flow in almost-linear time. In *62st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*. IEEE, 2021. 4

[BK09] Glencora Borradaile and Philip N. Klein. An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. *J. ACM*, 56(2):9:1–9:30, 2009. 3, 5

[BKM+17] Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. *SIAM J. Comput.*, 46(4):1280–1303, 2017. 5

[Bor08] Glencora Borradaile. *Exploiting Planarity for Network Flow and Connectivity Problems*. Brown University, 2008. 5

[CEN12] Erin W. Chambers, Jeff Erickson, and Amir Nayyeri. Homology flows, cohomology cuts. *SIAM J. Comput.*, 41(6):1605–1634, 2012. 3, 5

[CKL+22] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. *CoRR*, abs/2203.00671, 2022. 4, 5

[CKM+11] Paul Christiano, Jonathan A Kelner, Aleksander Madry, Daniel A Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 273–282, 2011. 4, 6

[CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009. 52

[CLS21] Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *Journal of the ACM (JACM)*, 68(1):1–39, 2021. 6, 7, 9

[CMSV17] Michael B Cohen, Aleksander Mądry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in $\widetilde{O}(m^{10/7} \log w)$ time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 752–771. SIAM, 2017. 6

[DKP+17] David Durfee, Rasmus Kyng, John Peebles, Anup B. Rao, and Sushant Sachdeva. Sampling random spanning trees faster than matrix multiplication. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 730–742, 2017. 33

[DLY21a] Sally Dong, Yin Tat Lee, and Guanghao Ye. A nearly-linear time algorithm for linear programs with small treewidth: A multiscale representation of robust central path. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 1784–1797. ACM, 2021. 4, 5, 6, 7, 9, 10, 11, 18

[DLY21b] Sally Dong, Yin Tat Lee, and Guanghao Ye. A nearly-linear time algorithm for linear programs with small treewidth: A multiscale representation of robust central path. *arXiv preprint arXiv:2011.05365v2*, 2021. 12, 54

[DS08] Samuel I Daitch and Daniel A Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 451–460, 2008. 4, 6

[FF56] Lester R Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956. 3, 5

[FR06] Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences*, 72(5):868–889, 2006. 8, 30

[GHP18] Gramoz Goranci, Monika Henzinger, and Pan Peng. Dynamic effective resistances and approximate Schur Complement on separable graphs. In *26th Annual European Symposium on Algorithms, ESA 2018*, volume 112 of *LIPIcs*, pages 40:1–40:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 8, 81

[GLP21] Yu Gao, Yang P. Liu, and Richard Peng. Fully dynamic electrical flows: Sparse maxflow faster than Goldberg-Rao. In *62st IEEE Annual Symposium on Foundations of Computer Science, FOCS2021*. IEEE, 2021. 5, 6

[Gre96] Keith D Gremban. *Combinatorial preconditioners for sparse, symmetric, diagonally dominant linear systems*. PhD thesis, Carnegie Mellon University, 1996. 3

[GT87] J. R. Gilbert and R. E. Tarjan. The analysis of a nested dissection algorithm. *Numer. Math.*, 50(4):377–404, February 1987. 7, 8

[Has81] Refael Hassin. Maximum flow in (s, t) planar networks. *Information Processing Letters*, 13(3):107, 1981. 5

[HJ85] Refael Hassin and Donald B Johnson. An $O(n \log^2 n)$ algorithm for maximum flow in undirected planar networks. *SIAM Journal on Computing*, 14(3):612–624, 1985. 3, 5

[HJST21] Baihe Huang, Shunhua Jiang, Zhao Song, and Runzhou Tao. Solving tall dense SDPs in the current matrix multiplication time. *arXiv preprint arXiv:2101.08208*, 2021. 6

[HKRS97] Monika R Henzinger, Philip Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997. 5

[II90] Hiroshi Imai and Kazuo Iwano. Efficient sequential and parallel algorithms for planar minimum cost flow. In *Algorithms, International Symposium SIGAL '90, Tokyo, Japan*, volume 450 of *Lecture Notes in Computer Science*, pages 21–30. Springer, 1990. 5

[INSW11] Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011*, pages 313–322. ACM, 2011. 5

[IS79] Alon Itai and Yossi Shiloach. Maximum flow in planar networks. *SIAM Journal on Computing*, 8(2):135–150, 1979. 5

[JS21] Arun Jambulapati and Aaron Sidford. Ultrasparse ultrasparsifiers and faster laplacian system solvers. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 540–559. SIAM, 2021. 27

[KLOS14] Jonathan A Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on discrete algorithms*, pages 217–226. SIAM, 2014. 4, 5

[KLP+16] Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman. Sparsified cholesky and multigrid solvers for connection laplacians. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 842–850. ACM, 2016. 8

[KLS20] Tarun Kathuria, Yang P. Liu, and Aaron Sidford. Unit capacity maxflow in almost $o(m^{4/3})$ time. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 119–130, 2020. 4, 6, 7

[KN13] Haim Kaplan and Yahav Nussbaum. Min-cost flow duality in planar networks. *arXiv preprint arXiv:1306.6728*, 2013. 4

[KNK93] Samir Khuller, Joseph Naor, and Philip Klein. The lattice structure of flow in planar graphs. *SIAM Journal on Discrete Mathematics*, 6(3):477–490, 1993. 6

[KP15] Donggu Kang and James Payor. Flow Rounding. *arXiv preprint arXiv:1507.08139*, 2015. 25

[KPSW19] Rasmus Kyng, Richard Peng, Sushant Sachdeva, and Di Wang. Flows in almost linear time via adaptive preconditioning. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 902–913, 2019. 5, 6

[KRT94] Valerie King, Satish Rao, and Rorbert Tarjan. A faster deterministic maximum flow algorithm. *Journal of Algorithms*, 17(3):447–474, 1994. 4

[KS16] Rasmus Kyng and Sushant Sachdeva. Approximate gaussian elimination for laplacians-fast, sparse, and simple. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 573–582. IEEE, 2016. 8

[KS19] Adam Karczmarz and Piotr Sankowski. Min-cost flow in unit-capacity planar graphs. In *27th Annual European Symposium on Algorithms, ESA 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 66:1–66:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. 4, 5

[Kyn17] Rasmus Kyng. *Approximate Gaussian Elimination*. PhD thesis, Yale University, 2017. 8

[LR19] Nathaniel Lahn and Sharath Raghvendra. A faster algorithm for minimum-cost bipartite matching in minor-free graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 569–588. SIAM, 2019. 5

[LRT79] Richard J Lipton, Donald J Rose, and Robert Endre Tarjan. Generalized nested dissection. *SIAM journal on numerical analysis*, 16(2):346–358, 1979. 3, 7, 8

[LT79] RJ Lipton and Robert Tarjan. A Planar Separator Theorem. *SIAM Journal of Applied Mathematics*, 36(2):177–189, 1979. 12

[Mad13] Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 253–262. IEEE, 2013. 4, 6

[Mad16] Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 593–602. IEEE, 2016. 6

[MN95] Gary L. Miller and Joseph Naor. Flow in planar graphs with multiple sources and sinks. *SIAM J. Comput.*, 24(5):1002–1017, 1995. 5

[MP13] Gary L. Miller and Richard Peng. Approximate maximum flow on separable undirected graphs. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013*, pages 1151–1170. SIAM, 2013. 5, 8

[Orl88] James Orlin. A faster strongly polynomial minimum cost flow algorithm. In *Proceedings of the Twentieth annual ACM symposium on Theory of Computing*, pages 377–387, 1988. 4

[Rei83] John H Reif. Minimum s-t cut of a planar undirected network in $O(n \log^2 n)$ time. *SIAM Journal on Computing*, 12(1):71–81, 1983. 3, 5

[She13] Jonah Sherman. Nearly maximum flows in nearly linear time. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 263–269. IEEE, 2013. 4, 5

[She17] Jonah Sherman. Area-convexity, linf regularization, and undirected multicommodity flow. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 452–460, 2017. 4

[ST04] Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90, 2004. 4, 27

[ST18] Aaron Sidford and Kevin Tian. Coordinate methods for accelerating linf regression and faster approximate maximum flow. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 922–933. IEEE, 2018. 4

[Tar71] Robert E Tarjan. An efficient planarity algorithm. Technical report, 1971. 3

[VA10] Balachandran Vaidyanathan and Ravindra K Ahuja. Fast algorithms for specially structured minimum cost flow problems with applications. *Operations Research*, 58(6):1681–1696, 2010. 4

[vdB20] Jan van den Brand. A deterministic linear program solver in current matrix multiplication time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 259–278. SIAM, 2020. 6, 9

[vdB21] Jan van den Brand. Unifying matrix data structures: Simplifying and speeding up iterative algorithms. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 1–13. SIAM, 2021. 6

[vdBGJ+21] Jan van den Brand, Yu Gao, Arun Jambulapati, Yin Tat Lee, Yang P. Liu, Richard Peng, and Aaron Sidford. Faster maxflow via improved dynamic spectral vertex sparsifiers. *CoRR*, abs/2112.00722, 2021. 5, 6

[vdBLL+21] Jan van den Brand, Yin Tat Lee, Yang P Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, MDPs, and $\ell 1$-regression in nearly linear time for dense instances. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 859–869, 2021. 4, 5

[vdBLSS20] Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 775–788, 2020. 6

[Wei97] Karsten Weihe. Maximum (s, t)-flows in planar networks in $O(|v| \log |v|)$ time. *Journal of Computer and System Sciences*, 55(3):454–475, 1997. 5

A Appendix

Lemma 28. *Let G be a modified planar graph with separator tree $\mathcal{T}$. Let $\mathcal{H}$ be a set of K nodes in $\mathcal{T}$. Then*

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

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 $\mathcal{T}$:

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

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

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

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

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

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 $S(H')$.

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

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

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

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

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

where c is the constant such that $|S(H')| \leq c\left(|E(H')|\right)^{1/2}$ 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 82. *Let $\mathcal{C}$ be an α-separable class such that we can compute a balanced separator for any graph in $\mathcal{C}$ with n vertices and m edges in $s(m)$ time for some convex function $s(m) \geq m$. Given an α-separable graph, there is an algorithm that computes a separator tree $\mathcal{T}$ 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 $\mathrm{c}_1(G), \mathrm{c}_2(G)$ of G as follows: $V(\mathrm{c}_i(G)) = V(G_i) \cup S(G)$ and $E(\mathrm{c}_i(G)) = E(G_i)$, for $i = 1, 2$. Edges connecting some vertex in G_i and another vertex in $S(G)$ are added to $E(\mathrm{c}_i(G))$. For each edge connecting two vertices in $S(G)$, we append it to $E(\mathrm{c}_1(G))$ or $E(\mathrm{c}_2(G))$, whichever has less edges. By construction, property 2 in the definition of $\mathcal{T}(G)$ holds. We continue by repeatedly splitting each child $\mathrm{c}_i(G)$ 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 running time 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 83. *Let G be an α-separable graph with separator tree $\mathcal{T}$. 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 Eq. (A.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. □