Faster Min-Cost Flow and Approximate Tree Decomposition on Bounded Treewidth Graphs

Sally Dong
University of Washington, Seattle, WA, USA

Guanghao Ye
Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

We present an algorithm for min-cost flow in graphs with n vertices and m edges, given a tree decomposition of width τ and size S, and polynomially bounded, integral edge capacities and costs, running in $\widetilde{O}(m\sqrt{\tau}+S)$ time. This improves upon the previous fastest algorithm in this setting achieved by the bounded-treewidth linear program solver of [26, 16], which runs in $\widetilde{O}(m\tau^{(\omega+1)/2})$ time, where $\omega \approx 2.37$ is the matrix multiplication exponent. Our approach leverages recent advances in structured linear program solvers and robust interior point methods (IPM). In general graphs where treewidth is trivially bounded by n, the algorithm runs in $\widetilde{O}(m\sqrt{n})$ time, which is the best-known result without using the Lee-Sidford barrier or ℓ_1 IPM, demonstrating the surprising power of robust interior point methods.

As a corollary, we obtain a $\widetilde{O}(\mathrm{tw}^3 \cdot m)$ time algorithm to compute a tree decomposition of width $O(\mathrm{tw} \cdot \log(n))$, given a graph with m edges.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Network optimization

Keywords and phrases Min-cost flow, tree decomposition, interior point method, bounded treewidth graphs

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.49

Related Version *Full Version*: `https://arxiv.org/abs/2308.14727` [18]

Funding *Guanghao Ye*: Supported by NSF awards CCF-1955217 and DMS-2022448.

1 Introduction

An active area of research in recent years is the advancement of interior point methods (IPM) for linear and convex programs, with its origins tracing back to the works of [30, 34]. This, along with the design of problem-specific data structures supporting the IPM, had led to breakthroughs in faster linear program solvers [13] and faster max flow algorithms [31, 10, 24, 4, 12], among others.

One line of research, inspired by nested dissection from [33] and methods in numerical linear algebra [14], focuses on exploiting separable structures in the constraint matrix of the linear program, which can be captured succinctly by associating the matrix with a graph. Several classical graph-theoretic concepts are appropriate here: Given a graph $G = (V, E)$ on n vertices, we say $S \subseteq V$ is a *(b-)balanced vertex separator* if there exists some constant $b \in (0, 1)$ such that every connected component of $G \setminus S$ has size at most bn. For a function f, we say G is *f-separable* if any subgraph H of G has a balanced separator of size $f(|V(H)|)$. The *treewidth* of a graph informally measures how close a graph is to a tree, and is closely related to separability. If G has treewidth τ, then any subgraph of G has a 1/2-balanced separator of size $\tau + 1$; conversely, if G is τ-separable, then G has treewidth at most $\tau \log n$ (c.f. [7]). Treewidth has been a important focus in the study of parametrized complexity [19, 8].

Formally,

▶ **Definition 1.** *A* tree decomposition *of a graph G is a pair (X, T) where T is a tree, and $X : V(T) \mapsto 2^{V(G)}$ is a family of subsets of $V(G)$ labelling the nodes of T called* bags*, such that*

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

Given a tree decomposition (X, T), its width is defined to be $\max\{|X(t)| - 1 : t \in T\}$*; and its* size *is $\sum_{t\in V(T)} |X(t)|$. The* treewidth *of G is the minimum width over all tree decompositions of G.*

By leveraging various separable properties of graphs, the sequence of papers [17, 15, 16] have iteratively refined the robust IPM framework and associated data structures for solving structured linear programs. [17] gave the first general linear program solver parametrized by treewidth: Given an LP of the form $\min\{\boldsymbol{c}^\top \boldsymbol{x} : \mathbf{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \mathbf{0}\}$ and a width τ decomposition of the *dual graph*[1] $G_{\mathbf{A}}$ of the constraint matrix $\mathbf{A}$, suppose the feasible region has inner radius r and outer radius R, and the costs are polynomially bounded. Then the LP can be solved to ε-accuracy in $\widetilde{O}(m\tau^2 \log(R/(\varepsilon r))$ time[2]. The τ^2 factor arises from carefully analyzing the sparsity-pattern of the *Cholesky factorization* of $\mathbf{A}\mathbf{A}^\top$ and the associated matrix computation times. This runtime dependence on τ was improved from quadratic to $(\omega + 1)/2 \approx 1.68$ in [26], by means of a coordinate-grouping technique applied to the updates during the IPM. [26] further combined [17] with [36] to give the current best algorithm for semidefinite programs with bounded treewidth. The result of [17] is now subsumed by [16], which shows that under the same setup, except where $G_{\mathbf{A}}$ is known to be a κn^α-separable graph[3] the LP can be solved in $\widetilde{O}\left(\left(\kappa^2(m + m^{2\alpha+0.5}) + m\kappa^{(\omega-1)/(1-\alpha)} + \kappa^\omega n^{\alpha\omega}\right) \cdot \log(\frac{mR}{\varepsilon r})\right)$ time. Taking $\kappa = \tau$ and $\alpha = 0$ for bounded treewidth graphs recovers the [17] result. For n^α-separable graphs, the runtime expression simplifies to $\widetilde{O}\left((m + m^{1/2+2\alpha}) \log(\frac{mR}{\varepsilon r})\right)$.

In the special setting of flow problems, the constraint matrix $\mathbf{A}$ is the vertex-edge incidence matrix of a graph and $\mathbf{A}\mathbf{A}^\top$ is corresponding graph Laplacian. Fast Laplacian solvers [35, 28] and approximate Schur complements [20] can be combined with the separable structure of the graph to further speed up computations at every IPM step. Using these ideas, [15] gave a nearly-linear time min-cost flow algorithm for planar graphs, which are $O(\sqrt{n})$-separable. Notably, the combination of nested dissections with fast approximate Schur complements was explored before in [25], where they demonstrated how to maintain dynamic effective resistances in separable graphs.

In this work, we expand the landscape by giving the first specific result for min-cost flow on bounded treewidth graphs. Previously, [23] showed how to compute a max vertex flow in $O(\tau^2 \cdot n \log n)$ time given a directed graph with unit capacities and its tree decomposition of width τ; their approach is combinatorial in nature.

▶ **Theorem 2** (Main result). *Let $G = (V, E)$ be a directed graph with n vertices and m edges. Assume that the demands $\boldsymbol{d}$, edge capacities $\boldsymbol{u}$ and costs $\boldsymbol{c}$ are all integers and bounded by M in absolute value. Given a tree decomposition of G with width τ and size S, there is an algorithm that computes a minimum-cost flow in $\widetilde{O}(m\sqrt{\tau} \log M + S)$ expected time.*

[1] The dual graph of a matrix $\mathbf{A} \in \mathbb{R}^{n\times m}$ is a graph on n vertices, where each column of $\mathbf{A}$ gives rise to a clique (or equivalently, a hyper-edge). Importantly, when the linear program is a flow problem on a graph, $G_{\mathbf{A}}$ is precisely the graph in the original problem.

[2] Throughout the paper, we use $\widetilde{O}(\cdot)$ to hide polylog m and polylog M factors.

[3] Here, κ is a constant expression, but could be a function of the size of $G_{\mathbf{A}}$ (which is considered fixed).

As a direct corollary of Theorem 2, we can solve min-cost flow on any graph with n vertices, m edges and integral polynomially-bounded costs and constraints in $\widetilde{O}(m\sqrt{n})$ expected time, as the treewidth of any n-vertex graph is at most $\tau = n$, and the tree decomposition is trivially the graph itself. This result matches that of [31] obtained using the Lee-Sidford barrier for the IPM, which requires $\widetilde{O}(\sqrt{n})$ iterations. In contrast, we use the standard log barrier which requires $\widetilde{O}(\sqrt{m})$ iterations, and leverage the robustness of the IPM and custom data structures to reduce the amortized cost per iteration. We find it noteworthy that all other max flow results matching or beating our time require either the Lee-Sidford barrier ([10]) or significantly divergent IPM techniques ([12]).

Using our faster max-flow algorithm as a subroutine, we can efficiently compute a tree decomposition of any given graph, where the width is within a $O(\log n)$-factor of the optimal:

▶ **Corollary 3** (Approximating treewidth). *Let $G = (V, E)$ be a graph with n vertices, m edges, and treewidth $\mathrm{tw}(G)$. There is an algorithm to find a tree decomposition of G with width at most $O(\mathrm{tw}(G) \cdot \log n)$ in $\widetilde{O}(\mathrm{tw}(G)^3 \cdot m)$ expected time.*

It is well known that computing the treewidth of a graph is NP-hard [1], and there is conditional hardness result for even constant-factor approximation algorithm [3]. For polynomial time algorithms, the best known result is a $O(\sqrt{\log \mathrm{tw}})$-approximation algorithm by [22], which involves solving a semidefinite program with n^2 variables.

There is a series of works focused on computing approximate treewidth for small treewidth graph in nearly-linear time; we refer the readers to [6] for a more detailed survey. Notably, [23] showed for any graph G, there is an algorithm to compute a tree decomposition of width $O(\mathrm{tw}(G)^2)$ in $\widetilde{O}(\mathrm{tw}(G)^7 \cdot n)$ time. [11] improved the running time to $\widetilde{O}(\mathrm{tw}(G)^3 \cdot m)$ with slightly compromised approximation ratio $O(\mathrm{tw}(G)^2 \cdot \log^{1+o(1)} n)$. More recently, [5] showed how to compute a tree decomposition of width $O(\mathrm{tw}(G) \cdot \log^3 n)$ in $O(m^{1+o(1)})$ time.

1.1 Overview of techniques

The foundation of our algorithm is the planar min-cost flow algorithm from [15]. We begin with the identical robust IPM algorithm in abstraction, as given in Algorithm 1. [15] first defines a separator tree $\mathcal{T}$ for the input graph, and uses it as the basis for the data structures in the IPM. We modify the separator tree construction, so that instead of recursively decomposing the input planar graph which is $O(\sqrt{n})$-separable, we recursively decompose the bounded treewidth graph. The leaf nodes of $\mathcal{T}$ partition the edges of the input graph; We guarantee that each leaf node contains $O(\tau)$-many edges, compared to constantly-many in the planar case.

There are two main components to the data structures from [15]:

1. A data structure DynamicSC (Theorem 10) is used to maintain approximate Schur complement matrices at every node of of the separator tree $\mathcal{T}$. It implicitly represents the matrix $(\mathbf{B}^\top \mathbf{W} \mathbf{B})^{-1}$, where $\mathbf{W}$ are edge weights changing at every step of the IPM. We use DynamicSC in exactly the same way.
2. A data structure MaintainSoln (Theorem 11) using $\mathcal{T}$ to implicitly maintain the primal and dual solutions $\boldsymbol{f}$ and $\boldsymbol{s}$, and explicitly maintain approximate solutions $\overline{\boldsymbol{f}}$ and $\overline{\boldsymbol{s}}$ at the current IPM iteration. The approximate solutions are used in the subsequent iteration to compute the step direction $\boldsymbol{v}$ and edge weights $\boldsymbol{w}$.

In [15], the approximate solutions $\overline{\boldsymbol{f}}$ and $\overline{\boldsymbol{s}}$ are updated coordinate-wise whenever a coordinate is sufficiently far from $\overline{\boldsymbol{f}}$ and $\overline{\boldsymbol{s}}$ respectively. An update induces changes in the data structures as follows: If $\overline{\boldsymbol{f}}_e$ or $\overline{\boldsymbol{s}}_e$ is updated for an edge e (subsequently, $\boldsymbol{w}_e$ and $\boldsymbol{v}_e$

are updated), then we find the unique leaf node H in $\mathcal{T}$ containing the edge e, and update DynamicSC and MaintainSoln at all nodes along the path from H to the root of $\mathcal{T}$. [15] shows this runtime depends on the sizes of the nodes visited.

[26] introduced a natural grouping technique for the coordinate updates, where coordinates are grouped into blocks, and updates are performed block-wise. Since the leaves of our separator tree $\mathcal{T}$ contain $O(\tau)$ edges, it is natural for us to also incorporate a blocking scheme, where the blocks are given by the edge partition according to the leaves of $\mathcal{T}$. In this case, the runtime for data structure updates is the same whether we update a single coordinate or a block containing said coordinate, since they affect the same path from the leaf node to the root of $\mathcal{T}$. We bound the overall runtime expression using properties of the new separator tree.

Lastly, the RIPM guarantees that for each k iterations, $\widetilde{O}(k^2)$-many blocks of $\overline{\boldsymbol{f}}$ and $\overline{\boldsymbol{s}}$ are updated, meaning that running our data structures for many IPM iterations leads to superlinear runtime scaling. We can, however, restart our data structures at any point, by explicitly computing the current exact solutions $\boldsymbol{f}, \boldsymbol{s}$ and reinitializing the data structures with these values in $\widetilde{O}(m)$ total time. On balance, we choose to restart the data structures every $\sqrt{m/\tau}$-many iterations, for a total restarting cost of $\widetilde{O}(m\sqrt{\tau})$. Between each restart, we make $O(m/\tau)$-many block updates using $\widetilde{O}(\tau)$ time each, for a total update cost of $\widetilde{O}(m\sqrt{\tau})$. Hence, the overall runtime is $\widetilde{O}(m\sqrt{\tau})$.

2 Robust interior point method

For the sake of completion, we give the robust interior point method from [17, 16], which is a refinement of the methods in [13, 9], for solving linear programs of the form

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

for some matrix $\mathbf{B} \in \mathbb{R}^{m \times n}$.

▶ **Theorem 4** ([17]). *Consider the linear program*

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

with $\mathbf{B} \in \mathbb{R}^{m\times n}$. Suppose there exists some interior point $\boldsymbol{f}_\circ$ satisfying $\mathbf{B}^\top \boldsymbol{f}_\circ = \boldsymbol{b}$ and $\boldsymbol{l} + r \le \boldsymbol{f}_\circ \le \boldsymbol{u} - r$,[4] *for some scalar $r > 0$. Let $L = \|\boldsymbol{c}\|_2$ and $R = \|\boldsymbol{u} - \boldsymbol{l}\|_2$. For any $0 < \epsilon \le 1/2$, the algorithm RIPM (Algorithm 1) finds $\boldsymbol{f}$ such that $\mathbf{B}^\top \boldsymbol{f} = \boldsymbol{b}$, $\boldsymbol{l} \le \boldsymbol{f} \le \boldsymbol{u}$, and*

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

Furthermore, the algorithm has the following properties:

- *Each call of Solve involves $O(\sqrt{m} \log m \log(\frac{mR}{\epsilon r}))$ many steps, and $\bar{t}$ is updated for $O(\log m \log(\frac{mR}{\epsilon r}))$ times total.*
- *In each step of Solve, 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 Solve, $h\|\boldsymbol{v}\|_2 = O(\frac{1}{\log m})$.*
- *Algorithm 1 to Algorithm 1 takes $O(K)$ time in total, where K is the total number of coordinate changes in $\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}$.* ◀

We note that this algorithm only requires access to $(\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}})$, but not $(\boldsymbol{f}, \boldsymbol{s})$ during the main while-loop in Solve. Hence, $(\boldsymbol{f}, \boldsymbol{s})$ can be implicitly maintained via any data structure.

[4] 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 [17].

```
 1: procedure RIPM($\mathbf{B} \in \mathbb{R}^{m\times n}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{l}, \boldsymbol{u}, \epsilon$)
 2:     Let $L = \|\boldsymbol{c}\|_2$ and $R = \|\boldsymbol{u} - \boldsymbol{l}\|_2$
 3:     Define $\phi_i(x) \overset{\text{def}}{=} -\log(\boldsymbol{u}_i - x) - \log(x - \boldsymbol{l}_i)$
 4:     Define $\mu_i^t(\boldsymbol{f}_i, \boldsymbol{s}_i) \overset{\text{def}}{=} \boldsymbol{s}_i/t + \nabla\phi_i(\boldsymbol{f}_i)$
 5:     Define $\gamma^t(\boldsymbol{f}, \boldsymbol{s})_i \overset{\text{def}}{=} \|(\nabla^2\phi_i(\boldsymbol{f}_i))^{-1/2}\mu_i^t(\boldsymbol{f}_i, \boldsymbol{s}_i)\|_2$

    ▷ Modify the linear program and obtain an initial $(\boldsymbol{f}, \boldsymbol{s})$ for modified linear program
 6:     Let $t = 2^{21}m^5 \cdot \frac{LR}{128} \cdot \frac{R}{r}$
 7:     Compute $\boldsymbol{f}_c = \arg\min_{\boldsymbol{l}\le\boldsymbol{f}\le\boldsymbol{u}} \boldsymbol{c}^\top\boldsymbol{f} + t\phi(\boldsymbol{f})$ and $\boldsymbol{f}_\circ = \arg\min_{\mathbf{B}^\top\boldsymbol{f}=\boldsymbol{b}} \|\boldsymbol{f} - \boldsymbol{f}_c\|_2$
 8:     Let $\boldsymbol{f} = (\boldsymbol{f}_c, 3R + \boldsymbol{f}_\circ - \boldsymbol{f}_c, 3R)$ and $\boldsymbol{s} = (-t\nabla\phi(\boldsymbol{f}_c), \frac{t}{3R+\boldsymbol{f}_\circ-\boldsymbol{f}_c}, \frac{t}{3R})$
 9:     Let the new matrix $\mathbf{B}^{\text{new}} \overset{\text{def}}{=} [\mathbf{B}; \mathbf{B}; -\mathbf{B}]$, 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
10:     $((\boldsymbol{f}^{(1)}, \boldsymbol{f}^{(2)}, \boldsymbol{f}^{(3)}), (\boldsymbol{s}^{(1)}, \boldsymbol{s}^{(2)}, \boldsymbol{s}^{(3)})) \leftarrow \textsc{Solve}(\mathbf{B}^{\text{new}}, \phi^{\text{new}}, \boldsymbol{f}, \boldsymbol{s}, t, LR)$
11:     $(\boldsymbol{f}, \boldsymbol{s}) \leftarrow (\boldsymbol{f}^{(1)} + \boldsymbol{f}^{(2)} - \boldsymbol{f}^{(3)}, \boldsymbol{s}^{(1)})$

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

15: procedure Solve($\mathbf{B}, \phi, \boldsymbol{f}, \boldsymbol{s}, t_{\text{start}}, t_{\text{end}}$)
16:     Let $\alpha = \frac{1}{2^{20}\lambda}$ and $\lambda = 64\log(256m^2)$ where $m$ is the number of rows in $\mathbf{B}$
17:     Let $t \leftarrow t_{\text{start}}$, $\overline{\boldsymbol{f}} \leftarrow \boldsymbol{f}, \overline{\boldsymbol{s}} \leftarrow \boldsymbol{s}, \overline{t} \leftarrow t$
18:     while $t \ge t_{\text{end}}$ do
19:         $t \leftarrow \max\left\{(1 - \frac{\alpha}{\sqrt{m}})t, t_{\text{end}}\right\}$
20:         Update $h = -\alpha/\|\cosh(\lambda\gamma^{\overline{t}}(\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}))\|_2$
21:         Update the diagonal weight matrix $\mathbf{W} = \nabla^2\phi(\overline{\boldsymbol{f}})^{-1}$
22:         Update the direction $\boldsymbol{v}$ where $\boldsymbol{v}_i = \sinh(\lambda\gamma^{\overline{t}}(\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}})_i)\mu^{\overline{t}}(\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}})_i$
23:         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

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

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

3 Nested dissection on bounded treewidth graphs

In this section, we show how to leverage the structural properties of bounded treewidth graphs to find a sparse Cholesky factorization of $\mathbf{L} \stackrel{\text{def}}{=} \mathbf{B}^\top \mathbf{W} \mathbf{B}$, and hence implicitly maintain an approximation of $\mathbf{L}^{-1}$ as part of the projection matrix $\mathbf{P}_{\boldsymbol{w}} \stackrel{\text{def}}{=} \mathbf{W}^{1/2}\mathbf{B}(\mathbf{B}^\top\mathbf{W}\mathbf{B})^{-1}\mathbf{B}^\top\mathbf{W}^{1/2}$ used in the RIPM.

3.1 Separator tree for bounded treewidth graph

The notion of using a *separator tree* to represent the recursive decomposition of a separable graph is well-established in literature, c.f [21, 27]. In [15], the authors show that the separator tree can be used to construct a sparse approximate projection matrix for RIPM. Here, we extends their result to bounded treewidth graphs.

▶ **Definition 5** (Separator tree). *Let G be a graph with n vertices and m edges. A* separator tree *$\mathcal{T}$ of G is a rooted constant-degree tree whose nodes represent a recursive decomposition of G based on vertex separators.*

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

The nodes of $\mathcal{T}$ satisfy the following recursive definition:

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 $H_1, H_2 \in \mathcal{T}$ that form a edge-disjoint partition of H, and $S(H) \stackrel{\text{def}}{=} V(H_1) \cap V(H_2)$ is a vertex separator of H. Define the set of eliminated vertices at H to be $F_H \stackrel{\text{def}}{=} S(H) \setminus \partial H$.*
   By definition of boundary vertices, we have $\partial H_1 \stackrel{\text{def}}{=} (\partial H \cup S(H)) \cap V(H_1)$, and $\partial H_2 \stackrel{\text{def}}{=} (\partial H \cup S(H)) \cap V(H_2)$.
3. *If H is a leaf node, define $S(H) = \emptyset$ and $F_H = V(H) \setminus \partial H$.*

By construction, the leaf nodes of $\mathcal{T}$ partition the edges of G. If H is a leaf node, let $E(H)$ denote the edges contained in H. If H is not a leaf node, let $E(H)$ denote the union of all the edges in the leaf nodes in the subtree $\mathcal{T}_H$. Let η denote the height of $\mathcal{T}$.

Next, we show how to construct an appropriate separator tree for bounded treewidth graphs.

▶ **Theorem 6.** *Let G be a graph with n vertices and m edges. Given a tree decomposition of G with width τ and size s, we can construct a separator tree $\mathcal{T}$ of G in $\widetilde{O}(s)$ time, such that $\mathcal{T}$ has height $\eta = O(\log m)$, each node $H \in \mathcal{T}$ satisfies $|F_H \cup \partial H| \le O(\tau \operatorname{poly}\log(m))$, and each leaf node $H \in \mathcal{T}$ has $|E(H)| \le \Theta(\tau)$.*

To prove the theorem above, we need a lemma about balanced vertex separators. Let us adopt the notation (A, S, B) to mean that S is a vertex separator such that its removal from the graph leaves two disjoint subgraphs A and B.

▶ **Lemma 7** (Slight modification of [17, Theorem 4.17]). *Let (X, T) be a tree decomposition of a graph G on n vertices with width τ and size s. Then in $O(s)$ time, we can find a 2/3-balanced vertex separator (A, S, B) of G and compute tree decompositions (X_1, T_1) of $G[A \cup S]$ with width τ and size s_1, and (X_2, T_2) of $G[B \cup S]$ with width τ and size s_2, such that $s_1 + s_2 \le s + \tau$.*

Proof. By scanning through the bags of T in $O(s)$ time, we can find a node $t \in T$ such that $T \setminus t$ is two disjoint subtrees T_1' and T_2', with $|\bigcup_{u \in T_1'} X(s) \setminus X(t)| \leq 2/3n$, and similarly for T_2'. Then $S \overset{\text{def}}{=} X(t) \subseteq V(G)$ is a 2/3-balanced vertex separator of G, and $A \overset{\text{def}}{=} \bigcup_{u \in T_1'} X(u)$ and $B \overset{\text{def}}{=} \bigcup_{u \in T_2'} X(u)$ partitions the remaining vertices of G.

Observe that (X, T) restricted to the nodes $T_1 \cup t$ yields a tree decomposition of $G[A \cup S]$ (with vertices of B deleted from the bags), and similarly when restricted to $T_2 \cup t$ yields a tree decomposition of $G[B \cup S]$. Both trivially have width τ. Since only the node t is shared between the two, and t has size at most τ, we see that $s_1 + s_2 \leq s + \tau$. ◀

Proof of Theorem 6. We construct the separator tree $\mathcal{T}$ by recursively decomposing the graph starting with subgraph $H = G$:

1. Given a subgraph H of G, if $|E(H)| \leq \Theta(\tau)$, then set H as a leaf node.
2. Else if $|V(H)| > 2\tau$, find a 2/3-balanced vertex separator (A, S, B) of H using the tree decomposition of H by Lemma 7. Let (X_1, T_1) and (X_2, T_2) be the resulting tree decompositions of $G[A \cup S]$ and $G[B \cup S]$.
   Let H_1 and H_2 be subgraphs of H where $V(H_1) = A \cup S$ and $V(H_2) = B \cup S$. Partition $E(H)$ so that edges incident to A are in H_1 and edges incident to B are in H_2, and edges in $S(H)$ can be arbitrarily assigned to H_1 or H_2. Observe that (X_1, T_1) and (X_2, T_2) are indeed tree decompositions of H_1 and H_2 respectively. Set H_1 and H_2 as the children of H.
3. Else if $|V(H)| \leq 2\tau$, then we partition the edges of H into two sets each of size at most $\frac{2}{3}|E(H)|$, and let H_1 and H_2 be graphs on $V(H)$ with their respective edge sets.[5]

Consider a non-leaf node H with children H_1 and H_2, we note that

$$|V(H_i)| \cdot |E(H_i)| \leq \frac{2}{3}|V(H)| \cdot |E(H)| \quad \text{for } i \in \{1, 2\}.$$

This directly shows the height of the separator tree $\mathcal{T}$ is $\eta = O(\log m)$. At any node H, the cardinality of $F_H \cup \partial H$ is bounded by the number of vertices contained in any ancestor node of H in $\mathcal{T}$, which is at most $O(\tau \log m)$.

The runtime is bounded by the total size of the tree decompositions across node generated by option 2 in the above construction. [16] shows that there are $\widetilde{O}(n/\tau)$-many interior nodes in $\mathcal{T}$ generated this way. Each such node contributes a term of τ to the total size, while each layer in the separator tree contributes a term of s. In total, this gives a size of $s \log m + (n/\tau) \cdot \tau = \widetilde{O}(s + n)$, as desired. ◀

3.2 Nested dissection using a separator tree

▶ **Definition 8** (Block Cholesky decomposition). *The* block Cholesky decomposition *of a symmetric matrix* $\mathbf{L}$ *with two blocks indexed by* F *and* C *is:*

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

where the middle matrix in the decomposition is a block-diagonal matrix with blocks indexed by F *and* C*, with the lower-right block being the* Schur complement $\mathbf{Sc}(\mathbf{L}, C)$ *of* $\mathbf{L}$ *onto* C*:*

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

[5] We use 2τ here to differentiate between the second and third case, instead of the more natural τ, in order to avoid any infinite-loop edge cases in the recursive process arising from division and rounding.

We use the separator tree structure to factor the matrix $\mathbf{L}^{-1} \stackrel{\text{def}}{=} (\mathbf{B}^\top \mathbf{W}\mathbf{B})^{-1}$:

▶ **Theorem 9** (Approximate $\mathbf{L}^{-1}$ factorization, c.f. [15, Theorem 33]). *Let $\mathcal{T}$ be the separator tree of G with height η. For each node $H \in \mathcal{T}$ with edges $E(H)$, let $\mathbf{B}[H]$ denote the matrix $\mathbf{B}$ restricted to rows indexed by $E(H)$, and define $\mathbf{L}[H] \stackrel{\text{def}}{=} \mathbf{B}[H]^\top \mathbf{W}\mathbf{B}[H]$.*

Given approximation parameter $\epsilon_{\mathbf{P}}$, suppose for each node H at level i of $\mathcal{T}$, we have a matrix $\mathbf{L}^{(H)}$ satisfying the $e^{i\epsilon_{\mathbf{P}}}$-spectral approximation

$$\mathbf{L}^{(H)} \approx_{i\epsilon_{\mathbf{P}}} \mathbf{Sc}(\mathbf{L}[H], \partial H \cup F_H). \tag{3.1}$$

Then, we can approximate $\mathbf{L}^{-1}$ by

$$\mathbf{L}^{-1} \approx_{\eta\epsilon_{\mathbf{P}}} \mathbf{\Pi}^{(0)\top} \cdots \mathbf{\Pi}^{(\eta-1)\top} \mathbf{\Gamma} \mathbf{\Pi}^{(\eta-1)} \cdots \mathbf{\Pi}^{(0)}, \tag{3.2}$$

where

$$\mathbf{\Gamma} \stackrel{\text{def}}{=} \begin{bmatrix} \sum_{H\in\mathcal{T}(0)} \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \sum_{H\in\mathcal{T}(\eta)} \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1} \end{bmatrix}, \tag{3.3}$$

and for $i = 0, \ldots, \eta - 1$,

$$\mathbf{\Pi}^{(i)} \stackrel{\text{def}}{=} \mathbf{I} - \sum_{H\in\mathcal{T}(i)} \mathbf{X}^{(H)}, \tag{3.4}$$

where $\mathcal{T}(i)$ is the set of nodes at level i in $\mathcal{T}$, the matrix $\mathbf{\Pi}^{(i)}$ is supported on $\bigcup_{H\in\mathcal{T}(i)} \partial H \cup F_H$ and padded with zeros to full dimension, and for each $H \in \mathcal{T}$,

$$\mathbf{X}^{(H)} \stackrel{\text{def}}{=} \mathbf{L}^{(H)}_{\partial H, F_H} \left(\mathbf{L}^{(H)}_{F_H,F_H}\right)^{-1}. \tag{3.5}$$

[15] gives a data structure to maintain an implicit representation of $\mathbf{L}^{-1}$ as the weights $\boldsymbol{w}$ undergoes changes in the IPM:

▶ **Theorem 10** ([15, Theorem 6]). *Given a graph G with m edges and its $\widetilde{O}(\tau)$-separator tree $\mathcal{T}$ with height $\eta = O(\log m)$, there is a deterministic data structure* DynamicSC *which maintains the edge weights $\boldsymbol{w}$ from the IPM, and at every node $H \in \mathcal{T}$, maintains two Laplacians $\mathbf{L}^{(H)}$ and $\widetilde{\mathbf{Sc}}(\mathbf{L}^{(H)}, \partial H)$ 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 $\boldsymbol{w}$ in $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2} \sum_{H\in\mathcal{H}} |F_H \cup \partial H|)$ time, where $\mathcal{H} \stackrel{\text{def}}{=} \{H \in \mathcal{T} : (\boldsymbol{w} - \boldsymbol{w}^{(\text{prev})})|_{E(H)} \neq \mathbf{0}\}$ is the set of nodes containing an edge with updated weight. (Note that $\mathcal{H}$ is a union of paths from leaf nodes to the root.)*
- *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, with high probability, for any node H in $\mathcal{T}$, if H is at level i, then

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

4 Solution maintenance

Assuming the correct maintenance of Laplacians and Schur complements along a recursive separator tree, [15] gave detailed data structures for maintaining the exact and approximate flow and slack solutions throughout the RIPM. Recall that the leaf nodes of the separator tree $\mathcal{T}$ form a partition of the edges of G. In [15], each leaf node of the separator tree contains $O(1)$-many edges, whereas in our case, each leaf node contains $O(\tau)$-many edges, and therefore we update the approximate solution in a block manner. The data structures in [15] naturally generalized from coordinate-wise updates to the block-wise case, so we use their implementation in a black-box manner. Here, we state a combined version of their main theorems.

We use $\boldsymbol{f}_{[i]}$ to denote the subvector of $\boldsymbol{f}$ indexed by edges in the i-th leaf node of $\mathcal{T}$, and similarly $\boldsymbol{s}_{[i]}$. We use $\boldsymbol{f}_{[i]}^{(k)}$ and $\boldsymbol{s}_{[i]}^{(k)}$ to denote the vector $\boldsymbol{f}_{[i]}$ and $\boldsymbol{s}_{[i]}$ at the k-th step of the IPM. Each block contains $O(\tau)$-many variables.

▶ **Theorem 11** ([15, Theorems 9, 10]). *Given a graph G with m edges and its separator tree $\mathcal{T}$ with height η, there is a randomized data structure implicitly maintains the IPM solution pair $(\boldsymbol{f}, \boldsymbol{s})$ undergoing IPM changes, and explicitly maintains its approximation $(\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}})$, and supports the following procedures with high probability against an adaptive adversary:*

- *Initialize*$(G, \boldsymbol{f}^{(\text{init})} \in \mathbb{R}^m, \boldsymbol{s}^{(\text{init})} \in \mathbb{R}^m, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^m_{>0}, \epsilon_{\mathbf{P}} > 0, \overline{\epsilon} > 0)$*: Given a graph G, initial solutions $\boldsymbol{f}^{(\text{init})}, \boldsymbol{s}^{(\text{init})}$, 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, set the implicit representations $\boldsymbol{f} \leftarrow \boldsymbol{f}^{(\text{init})}, \boldsymbol{s} \leftarrow \boldsymbol{s}^{(\text{init})}$, and set the approximations $\overline{\boldsymbol{f}} \leftarrow \boldsymbol{f}, \overline{\boldsymbol{s}} \leftarrow \boldsymbol{s}$.*
- *Reweight$(\boldsymbol{w} \in \mathbb{R}^m_{>0}$, given implicitly as a set of changed weights): Set the current weights to $\boldsymbol{w}$ in $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\tau K)$[6] time, where K is the number of blocks 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*

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

  for some $\widetilde{\mathbf{P}}_{\boldsymbol{w}}$ satisfying $\|(\widetilde{\mathbf{P}}_{\boldsymbol{w}} - \mathbf{P}_{\boldsymbol{w}})\boldsymbol{v}\|_2 \leq \eta\epsilon_{\mathbf{P}} \|\boldsymbol{v}\|_2$ and $\widetilde{\mathbf{P}}_{\boldsymbol{w}}\boldsymbol{v} \in \operatorname{Range}(\mathbf{B})$, and some other $\widetilde{\mathbf{P}}'_{\boldsymbol{w}}$ satisfying $\|\widetilde{\mathbf{P}}'_{\boldsymbol{w}}\boldsymbol{v} - \mathbf{P}_{\boldsymbol{w}}\boldsymbol{v}\|_2 \leq O(\eta\epsilon_{\mathbf{P}}) \|\boldsymbol{v}\|_2$ 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 total runtime is $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\tau K)$, where K is the number of blocks changed in $\boldsymbol{v}$.
- *Approximate$() \to \mathbb{R}^{2m}$: Return the vector pair $\overline{\boldsymbol{f}}, \overline{\boldsymbol{s}}$ implicitly as a set of changed coordinates, satisfying $\|\mathbf{W}^{-1/2}(\overline{\boldsymbol{f}} - \boldsymbol{f})\|_\infty \leq \overline{\epsilon}$ and $\|\mathbf{W}^{1/2}(\overline{\boldsymbol{s}} - \boldsymbol{s})\|_\infty \leq \overline{\epsilon}$, for the current weight $\boldsymbol{w}$ and the current solutions $\boldsymbol{f}, \boldsymbol{s}$.*
- *Exact$() \to \mathbb{R}^{2m}$: Output the current vector $\boldsymbol{f}, \boldsymbol{s}$ 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 blocks 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 sets $\overline{\boldsymbol{f}}_{[i]} \leftarrow \boldsymbol{f}_{[i]}^{(k)}, \overline{\boldsymbol{s}}_{[i]} \leftarrow \boldsymbol{s}_{[i]}^{(k)}$ for $O(N_k \overset{\text{def}}{=} 2^{2\ell_k}(\frac{\beta}{\overline{\epsilon}})^2 \log^2 m)$ blocks i, where ℓ_k is the largest integer ℓ with $k \equiv 0 \mod 2^\ell$ when $k \neq 0$ and $\ell_0 = 0$, and*
- *the amortized time for the k-th Approximate call is $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\tau(K + N_{k-2^{\ell_k}}))$.*

[6] The original bound here is $\widetilde{O}(\epsilon_{\mathbf{P}}^{-2}\sqrt{mK})$, where the $\sqrt{mK}$ factor comes from the fact that they can bound $\sum_{H\in\mathcal{H}} |F_H \cup \partial H| \leq \sqrt{mK}$ and $\mathcal{H} = \{H \in \mathcal{T} : (\boldsymbol{w} - \boldsymbol{w}^{(\text{prev})})|_{E(H)} \neq \mathbf{0}\}$. See [15, Section 9] for more details. Here, we replace it by $\widetilde{O}(\tau K)$ by the guarantees of Theorem 6.

An alternative derivation of the running time for Approximate shows that 2^ℓ steps take $\widetilde{O}(2^{2\ell}\tau)$ time. For $2^\ell = \Omega(\sqrt{m})$ steps of IPM, this would take $\widetilde{O}(m\tau)$ total time, which is too expensive for our use case. Since initialization only takes $\widetilde{O}(m)$ time, we reinitialize the data structure every $2^\ell = \Omega(\sqrt{m/\tau})$ steps, allowing us to avoid running the same data structure for too many steps and accuring large superlinear costs.

5 Proof of main theorems

Proof of Theorem 2. We apply RIPM from Theorem 4 combined with the data structures from Theorem 11.

First, we reduce the min-cost flow problem to min-cost circulation, for the sake of simplicity in showing the existence of an interior point in the polytope. We begin by adding extra vertices s and t to the input graph G, which increases the treewidth of G by at most 2. For every vertex v with demand $\boldsymbol{d}_v < 0$, we add a directed edge from s to v with capacity $-\boldsymbol{d}_v$ and cost 0. For every vertex v with $\boldsymbol{d}_v > 0$, we add a directed edge from v to t with capacity $\boldsymbol{d}_v$ and cost 0. Then, we add a directed edge from t to s with capacity $4nM$ and cost $-4nM$. The cost and capacity on the $t \to s$ edge is chosen such that the min-cost flow problem on the original graph is equivalent to the min-cost circulation on this new graph.

We apply Algorithm 1 in Theorem 4 to find a circulation $\boldsymbol{f}$ in the new graph such that $(\boldsymbol{c}^{\text{new}})^\top \boldsymbol{f} \le \text{OPT} + \frac{1}{2}$ by setting $\epsilon = \frac{1}{CM^2m^2}$ for some large constant C. The other parameters L, R, r in the theorem can be bounded by $L = \|\boldsymbol{c}^{\text{new}}\|_2 = O(Mm)$, $R = \|\boldsymbol{u}^{\text{new}} - \boldsymbol{l}^{\text{new}}\|_2 = O(Mm)$, and $r \ge \frac{1}{4m}$, as shown in [15]. Now, the solution $\boldsymbol{f}$, when restricted to the original graph, is almost a flow routing the required demand, with flow value off by at most $\frac{1}{2nM}$ from the optimal. This is because sending extra k units of fractional flow from s to t gives extra negative cost $\le -knM$. 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 [29]. 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.

The RIPM given in Algorithm 1 runs the subroutine Solve twice. In the first run, the constraint matrix is the incidence matrix of the input graph G copied three times. Copying edges does not affect treewidth, and our data structures allow for duplicate edges, so we may treat the two runs identically. We construct a separator tree $\mathcal{T}$ based on the input graph in $\widetilde{O}(S)$ time using Theorem 6, and implement the IPM data structures on top of $\mathcal{T}$ using Theorem 11.

Finally, we bound the IPM runtime. 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, 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 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 4 guarantees that there are $T = O(\sqrt{m} \log n \log(nM))$ total Approximate calls. We implement reinitialize the data structures every $\sqrt{m/\tau}$ steps.

At the k-th step after any initialization, the number of blocks changed in $\boldsymbol{w}$ and $\boldsymbol{v}$ is bounded by $K \stackrel{\text{def}}{=} O(2^{2\ell_{k-1}} \log^2 m)$, where ℓ_k is the largest integer ℓ with $k \equiv 0 \mod 2^\ell$, or equivalently, the number of trailing zeros in the binary representation of k. Theorem 4 guarantees we can apply Theorem 11 with parameter $\beta = O(1/\log m)$, which in turn shows the amortized time for the k-th call is $\widetilde{O}({\epsilon_{\mathbf{P}}}^{-2}\tau(K + N_{k-2^{\ell_k}}))$, where $N_k \overset{\text{def}}{=} 2^{2\ell_k}(\beta/\alpha)^2 \log^2 m = O(2^{2\ell_k}\log^2 m)$, with $\alpha = O(1/\log m)$ and $\epsilon_{\mathbf{P}} = O(1/\log m)$. Observe that $K + N_{k-2^{\ell_k}} = O(N_{k-2^{\ell_k}})$. Now, summing over $T = \sqrt{m/\tau}$ steps, the total runtime between each restart is

$$O(\sqrt{m/\tau})\sum_{k=1}^{T}\widetilde{O}(\tau N_{k-2^{\ell_k}}) = O(\sqrt{m/\tau})\sum_{\ell=0}^{\log T}\frac{T}{2^\ell}\cdot\widetilde{O}(2^{2\ell}\tau) = \widetilde{O}(m).$$

Theorem 4 guarantees $O(\sqrt{m}\log n\log(nM))$-many IPM steps in total. The data structure restarts $\widetilde{O}(\sqrt{\tau})$-many times, and each initialization time is $\widetilde{O}(m)$. Hence, the total runtime for the RIPM is $\widetilde{O}(m\sqrt{\tau}\log M)$. ◀

▶ **Corollary 3** (Approximating treewidth). *Let $G = (V, E)$ be a graph with n vertices, m edges, and treewidth $\mathrm{tw}(G)$. There is an algorithm to find a tree decomposition of G with width at most $O(\mathrm{tw}(G)\cdot\log n)$ in $\widetilde{O}(\mathrm{tw}(G)^3\cdot m)$ expected time.*

Our algorithm for Corollary 3 requires some tree decomposition of the graph as input. We use the following lemma to construct the initial tree decomposition.

▶ **Lemma 12** ([11]). *For any $\frac{2}{3} < \alpha < 1$ and $0 < \epsilon < 1 - \alpha$, given a graph G with n vertices and m edges, if the graph G contains an α-balanced vertex separator of size K, then there is a randomized algorithm that finds a balanced vertex separator of size $\widetilde{O}(K^2/\epsilon)$ in $\widetilde{O}(mK^3/\epsilon)$ expected time. The algorithm does not require knowledge of K.*

Next, the lemma below establishes the relationship between max flow and balanced edge separators. We first give the relevant definitions. For a given constant $c \le 1/2$, a directed edge-cut $(S, \overline{S})$ is called a *c-balanced edge separator* if both $|S| \ge cn$ and $|\overline{S}| \ge cn$. The capacity of the cut $(S, \overline{S})$ is the total capacity of all edges crossing the cut. The *minimum c-balanced edge separator* is the c-balanced edge separator with minimum capacity. A λ *pseudo-approximation* to the minimum c-balanced edge separator is a c'-balanced cut $(S, \overline{S})$ for some other constant c', whose capacity is within a factor of λ of that of the minimum c-balanced edge separator.

▶ **Lemma 13** ([2]). *An $O(\log n)$ pseudo-approximation to the minimum c-balanced edge separator in directed graphs can be computed using* polylog n *single-commodity flow computations on the same graph.*

Proof of Corollary 3. It is well known that given a $O(\log n)$ approximation algorithm for finding a balanced vertex separator, one can construct a tree decomposition of width $O(\mathrm{tw}(G)\log n)$. Specifically, the algorithm of [7] finds such a tree decomposition by recursively using a balanced vertex separator algorithm and requires only an additional log factor in the runtime.

Now, it suffices to show we can find a $\log(n)$ pseudo-approximation balanced vertex separator in $\widetilde{O}(m\cdot\mathrm{tw}(G)^3)$ expected time. Using the reduction from [32], we reduce the balanced vertex separator to directed edge separator on graph $G^* = (V^*, E^*)$, where

$$V^* = \{v \mid v \in V\} \cup \{v' \mid v \in V\},$$

and

$$E^* = \{\,(v, v') \mid v \in V\} \cup \{\,(u', v) \mid (u, v) \in E\} \cup \{\,(v', u) \mid (u, v) \in E\}.$$

We note that $\mathrm{tw}(G^*) = O(\mathrm{tw}(G))$. This shows G^* has a 2/3-balanced vertex separator of size $O(\mathrm{tw}(G))$. We first use Lemma 12 to construct a $\widetilde{O}(\mathrm{tw}(G)^2)$-separator tree for G^*. Then, we use the algorithm in [2] combined with our flow algorithm to find a balanced edge separator in $\widetilde{O}(m \cdot \mathrm{tw}(G))$ expected time. Hence, we can find a balanced vertex separator in $\widetilde{O}(m \cdot \mathrm{tw}(G)^3)$ expected time. ◀

References

1 Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987. doi:10.1137/0608024.

2 Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. *Journal of the ACM (JACM)*, 63(2):1–35, 2016. doi:10.1145/2837020.

3 Per Austrin, Toniann Pitassi, and Yu Wu. Inapproximability of treewidth, one-shot pebbling, and related layout problems. In *International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 13–24. Springer, 2012. doi:10.1007/978-3-642-32512-0_2.

4 Jan van den Band, 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. doi:10.48550/arXiv.2112.00722.

5 Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental sssp and approximate min-cost flow in almost-linear time. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1000–1008. IEEE, 2022. doi:10.1109/FOCS52979.2021.00100.

6 Hans L Bodlaender, Pål Grønås Drange, Markus S Dregi, Fedor V Fomin, Daniel Lokshtanov, and Michał Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM Journal on Computing*, 45(2):317–378, 2016. doi:10.1137/130947374.

7 Hans L Bodlaender, John R Gilbert, Hjálmtyr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995. doi:10.1006/jagm.1995.1009.

8 Hans L Bodlaender and Arie MCA Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008. doi:10.1093/comjnl/bxm037.

9 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. doi:10.1137/1.9781611975994.16.

10 Jan van den Brand, Yin Tat Lee, Yang P Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, MDPs, and ℓ1-regression in nearly linear time for dense instances. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 859–869, 2021. doi:10.1145/3406325.3451108.

11 Sebastian Brandt and Roger Wattenhofer. Approximating small balanced vertex separators in almost linear time. *Algorithmica*, 81:4070–4097, 2019. doi:10.1007/s00453-018-0490-x.

12 Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623. IEEE, 2022. doi:10.1109/FOCS54457.2022.00064.

13 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. doi:10.1145/3424305.

14 Timothy A Davis. *Direct methods for sparse linear systems.* SIAM, 2006.

15 Sally Dong, Yu Gao, Gramoz Goranci, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Guanghao Ye. Nested dissection meets ipms: Planar min-cost flow in nearly-linear time. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 124–153. SIAM, 2022. doi:10.1137/1.9781611977073.7.

16 Sally Dong, Gramoz Goranci, Lawrence Li, Sushant Sachdeva, and Guanghao Ye. Fast algorithms for separable linear programs. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3558–3604. SIAM, 2024. doi:10.1137/1.9781611977912.127.

17 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. doi:10.1145/3406325.3451056.

18 Sally Dong and Guanghao Ye. Faster min-cost flow and approximate tree decomposition on bounded treewidth graphs. *CoRR*, abs/2308.14727, 2023. doi:10.48550/arXiv.2308.14727.

19 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*, volume 4 of *Texts in Computer Science*. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

20 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. doi:10.1145/3055399.3055499.

21 David Eppstein, Zvi Galil, Giuseppe F Italiano, and Thomas H Spencer. Separator based sparsification: I. planarity testing and minimum spanning trees. *journal of computer and system sciences*, 52(1):3–27, 1996. doi:10.1006/jcss.1996.0002.

22 Uriel Feige, MohammadTaghi Hajiaghayi, and James R Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38(2):629, 2008. doi:10.1137/05064299X.

23 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, Michał Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Transactions on Algorithms (TALG)*, 14(3):1–45, 2018. doi:10.1145/3186898.

24 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. doi:10.1109/FOCS52979.2021.00058.

25 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. doi:10.4230/LIPIcs.ESA.2018.40.

26 Yuzhou Gu and Zhao Song. A faster small treewidth SDP solver. *arXiv preprint arXiv:2211.06033*, 2022. doi:10.48550/arXiv.2211.06033.

27 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. doi:10.1006/jcss.1997.1493.

28 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. doi:10.1137/1.9781611976465.33.

29 Donggu Kang and James Payor. Flow rounding. *CoRR*, abs/1507.08139, 2015. doi:10.48550/arXiv.1507.08139.

30 N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, December 1984. doi:10.1007/BF02579150.

31 Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $\tilde{O}(\sqrt{rank})$ iterations and faster algorithms for maximum flow. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 424–433. IEEE, 2014. doi:10.1109/FOCS.2014.52.

32 Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM (JACM)*, 46(6):787–832, 1999. doi:10.1145/331524.331526.

33 Richard J Lipton, Donald J Rose, and Robert Endre Tarjan. Generalized nested dissection. *SIAM journal on numerical analysis*, 16(2):346–358, 1979. doi:10.1137/0716027.

34 James Renegar. A polynomial-time algorithm, based on Newton's method, for linear programming. *Mathematical programming*, 40(1-3):59–93, 1988. doi:10.1007/BF01580724.

35 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. doi:10.1145/1007352.1007372.

36 Richard Y Zhang and Javad Lavaei. Sparse semidefinite programs with near-linear time complexity. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 1624–1631. IEEE, 2018. doi:10.1109/CDC.2018.8619478.