DOI:10.1145/3610940

To view the accompanying Technical Perspective, visit doi.acm.org/10.1145/3623277 tp

Almost-Linear-Time Algorithms for Maximum Flow and Minimum-Cost Flow

By Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva

Abstract

We present an algorithm that computes exact maximum flows and minimum-cost flows on directed graphs with m edges and polynomially bounded integral demands, costs, and capacities in $m^{1+o(1)}$ time. Our algorithm builds the flow through a sequence of $m^{1+o(1)}$ approximate undirected minimum-ratio cycles, each of which is computed and processed in amortized $m^{o(1)}$ time using a new dynamic graph data structure.

Our framework extends to algorithms running in $m^{1+o(1)}$ time for computing flows that minimize general edge-separable convex functions to high accuracy. This gives almost-linear time algorithms for several problems including entropy-regularized optimal transport, matrix scaling, p-norm flows, and p-norm isotonic regression on arbitrary directed acyclic graphs.

1. INTRODUCTION

The maximum flow problem and its generalization, the minimum-cost flow problem, are classic combinatorial graph problems that find numerous applications in engineering and scientific computing. These problems have been studied extensively over the last seven decades, starting from the work of Dantzig and Ford-Fulkerson. Several important algorithmic problems can be reduced to minimum-cost flows, for example, max-weight bipartite matching, min-cut, and Gomory-Hu trees. The origin of numerous significant algorithmic developments such as the simplex method, graph sparsification, and link-cut trees can be traced back to seeking faster algorithms for maximum flow and minimum-cost flow.

1.1. Problem formulation

Formally, we are given a directed graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges, upper/lower edge capacities $\boldsymbol{u}^+, \boldsymbol{u}^- \in \mathbb{R}^E$, edge costs $\boldsymbol{c} \in \mathbb{R}^E$, and vertex demands $\boldsymbol{d} \in \mathbb{R}^V$ with $\sum_{v \in V} \boldsymbol{d}_v = 0$. Our goal is to solve the following linear program for the *minimum-cost flow problem*

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^\top \boldsymbol{f} \\ \text{subject to} \quad & \boldsymbol{u}_e^- \le \boldsymbol{f}_e \le \boldsymbol{u}_e^+ \text{ for all } e \in E \\ & \boldsymbol{B}^\top \boldsymbol{f} = \boldsymbol{d}, \end{aligned} \tag{1}$$

where the last constraint, $\boldsymbol{B}^\top \boldsymbol{f} = \boldsymbol{d}$, succinctly captures the requirement that the flow $\boldsymbol{f}$ satisfies vertex demands $\boldsymbol{d}$, where $\boldsymbol{B} \in \mathbb{R}^{E \times V}$ is the edge-vertex incidence matrix defined as $\boldsymbol{B}_{((a,b),v)}$ is 1 if $v = a$, –1 if $v = b$, and 0 otherwise.

In this extended abstract, we assume that all $\boldsymbol{u}_e^+$, $\boldsymbol{u}_e^-$, $\boldsymbol{c}_e$ and $\boldsymbol{d}_v$ are integral and polynomially bounded in n since this paper focuses on weakly-polynomial algorithms for the maximum flow and minimum-cost flow problems.

1.2. Previous work

There has been extensive work on maximum flow and minimum-cost flow. Here, we briefly discuss some highlights from this work to help place our work in context.

Starting with the first pseudo-polynomial time algorithm by Dantzig[14] for maximum-flow that ran in $O(mn^2U)$ time where U denotes the maximum absolute capacity, approaches to designing faster flow algorithms were primarily combinatorial, working with various adaptations of augmenting paths, cycle canceling, blocking flows, and capacity/cost scaling. A long line of work led to a running time of $\tilde{O}(m \min\{m^{1/2}, n^{2/3}\})$[15, 18, 19, 21] for maximum flow, and $\tilde{O}(mn)$[17] for minimum-cost flow. These bounds stood for decades.

In a breakthrough work on solving Laplacian linear systems and computing electrical flows, Spielman and Teng[34] introduced a novel set of ideas and tools for solving flow problems using combinatorial techniques in conjunction with continuous optimization methods. To deploy these methods, flow algorithms researchers have used graph-algorithmic techniques to solve increasingly difficult subproblems that drive powerful continuous methods.

In the context of maximum flow and minimum-cost flow, Daitch and Spielman[13] demonstrated the power of this paradigm by using a path-following interior point method (IPM) to reduce the minimum-cost flow problem to solving a sequence of roughly $\sqrt{m}$ electrical flow (ℓ_2) problems. Since each of these ℓ_2 problems could then be solved in nearly-linear time using the fast Laplacian solver by Spielman-Teng, they achieved an $\tilde{O}(m^{1.5})$ time algorithm for minimum-cost flow, the first progress in two decades. They showed that a key advantage of IPMs is that they reduce flow problems on directed graphs to flow problems on undirected graphs, which are easier to work with.

While other continuous optimization methods have been used in the context of maximum flow, even leading

The original version of this paper titled "Maximum Flow and Minimum-Cost Flow in Almost-Linear Time" was published in *Proceedings of the 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science*, 612–623.

to nearly-linear time $(1+\epsilon)$ -approximate undirected maximum flow and multicommodity flow algorithms,[12, 23, 30, 32, 33] to date all approaches for exact maximum flow and minimum-cost flow rely on the framework suggested by Daitch and Spielman of using a path-following IPM to reduce to a small but polynomial number of convex optimization problems. Notable achievements include an $m^{4/3+o(1)}U^{1/3}$ time algorithm for bipartite matching and unit-capacity maximum flow.[4, 22, 27, 28, 29] Further, for general capacities, minimum-cost flow algorithms were given with runtimes $\tilde{O}(m+n^{1.5})$[8, 9, 10] and $\tilde{O}(m^{3/2-1/58})$.[5, 6, 16, 35] In both of these results, the development of efficient data structures to solve the sequence of ℓ_2 subproblems in amortized time sub-linear in m has played a key role in obtaining these runtimes. Yet, despite this progress, the best running time bounds remain far from linear.

1.3. Our result

We give the first almost-linear time algorithm for minimum-cost flow, achieving the optimal running time up to subpolynomial factors.

THEOREM 1.1. *There is an algorithm that, on a graph $G = (V, E)$ with m edges, vertex demands, upper/lower edge capacities, and edge costs, all integral with capacities and costs bounded by a polynomial in n, computes an exact minimum-cost flow in $m^{1+o(1)}$ time with high probability.*

Our algorithm can be extended to work with capacities and costs that are not polynomially bounded at the cost of an additional logarithmic dependency in both the maximum absolute capacity and the maximum absolute cost.

We make two key contributions to achieve our result. First, we develop a novel potential-reduction IPM, similar to Karmarkar's original work.[20] Our IPM is *worse* in some ways than existing path-following IPMs because it needs more updates to converge to a good solution. However, our new IPM reduces the minimum-cost flow problem to a sequence of update subproblems that have a more combinatorial structure than those studied before. This enables our second key contribution, a data structure that solves our sequence of update subproblems extremely quickly.

In addition to the use of highly combinatorial updates, our new IPM has three crucial properties. The IPM is (a) *robust* to approximation error in subproblems, (b) *stable* in terms of the subproblems it defines, and (c) *stable* in terms of a good solution to the subproblems. These features allow us to solve the sequence of update subproblems much faster by developing a powerful data structure, yielding a much faster algorithm overall. Thus, instead of making graph algorithms more suitable for continuous optimization, we made continuous optimization more suitable for graph algorithms.

We call the update subproblem that our IPM yields *min-ratio cycle*: This problem is specified by a graph where every edge has a "gradient" and a "length." The problem asks us to find a cycle that minimizes the sum of (signed, directed) edge gradients relative to its (undirected) length.

Our data structure for solving the sequence of min-ratio cycle problems is our second key contribution. As a first observation, we show that if we sample a random "low-stretch" spanning tree of the graph, then with constant probability, some *fundamental tree cycle* approximately solves the min-ratio cycle problem. Recall a fundamental tree cycle is a cycle defined by a single non-tree edge and the unique tree path between its endpoints. Unfortunately, this simple approach fails after a single flow update, as the IPM requires us to change the gradients and lengths after each update.

To maintain a set of trees that repeatedly allow us to identify good update cycles, we develop a hierarchical construction based on a recursive approach. This requires us to repeatedly construct and contract a random forest (which partially defines our tree), and then recurse on the resulting smaller graph, which we call a *core graph*. Furthermore, to enable recursion, we need to reduce the edge count in the core graph. We achieve this using a new *spanner* construction, which identifies a sparse subgraph of the core graph on which to recurse *and* detects if the removed edges damage the min-ratio cycle. Maintaining the core graphs and spanners in our recursive construction requires us to develop an array of novel dynamic graph techniques, which may be of independent interest.

1.4. Applications

Our result in Theorem 1.1 has a wide range of applications. By standard reductions, it gives the first $m^{1+o(1)}$ time algorithms for bipartite matching, worker assignment, negative-lengths single-source shortest paths, and several other problems. For the negative-lengths shortest path problem, Bernstein, Nanongkai, and Wulff-Nilsen obtained the first $m \cdot \mathrm{poly}(\log m)$ time algorithm in an independent and concurrent work.[7]

Using recent reductions from various connectivity problems to maximum flow, we also obtain the first $m^{1+o(1)}$ time algorithms to compute vertex connectivity and Gomory-Hu trees in undirected, unweighted graphs,[1,25] and $(1+\varepsilon)$-approximate Gomory-Hu trees in undirected weighted graphs.[26] We also obtain the current fastest algorithm to find the global min-cut in a directed graph.[11] Finally, we obtain the first almost-linear-time algorithm to compute the approximate sparsest cuts in directed graphs.

Additionally, our algorithm extends to computing flows that minimize general edge-separable convex objectives.

INFORMAL THEOREM 1.2. *Consider a graph G with demands $\boldsymbol{d}$, and an edge-separable convex cost function $\mathrm{cost}(\boldsymbol{f}) = \sum_e \mathrm{cost}_e(\boldsymbol{f}_e)$ for "computationally efficient" edge costs cost_e. Then in $m^{1+o(1)}$ time, we can compute a (fractional) flow $\boldsymbol{f}$ that routes demands $\boldsymbol{d}$ and $\mathrm{cost}(\boldsymbol{f}) \le \mathrm{cost}(\boldsymbol{f}^*) + \exp(-\log^C m)$ for any constant $C > 0$, where $\boldsymbol{f}^*$ minimizes $\mathrm{cost}(\boldsymbol{f}^*)$ over flows with demands $\boldsymbol{d}$.*

This generalization gives the first almost-linear-time algorithms for solving entropy-regularized optimal transport (equivalently, matrix scaling), p-norm flow problems, and p-norm isotonic regression for $p \in [1, \infty]$.

2. OVERVIEW

2.1. Computing minimum-cost flows via undirected min-ratio cycles

Recall the linear program for minimum-cost flow given in Equation (1). We assume that this LP has a unique optimal solution at $\boldsymbol{f}^* \in \mathbb{R}^E$ and let $F^* = \boldsymbol{c}^\top \boldsymbol{f}^*$ (this can be achieved by a negligible perturbation using the famous Isolation Lemma).

For our algorithm, we use a potential-reduction interior point method,[20] where in each iteration we measure progress by reducing the value of the potential function

$$\Phi(\boldsymbol{f}) \stackrel{\text{def}}{=} 20m \log(\boldsymbol{c}^\top \boldsymbol{f} - F^*) + \sum_{e\in E}\left((\boldsymbol{u}_e^+ - \boldsymbol{f}_e)^{-\alpha} + (\boldsymbol{f}_e - \boldsymbol{u}_e^-)^{-\alpha}\right)$$

for $\alpha = 1/(1000 \log mU)$. The reader can think of the barrier $x^{-\alpha}$ as the more standard $-\log x$ for simplicity instead. We use it for technical reasons beyond the scope of this paper.

Using standard techniques, one can add $O(n)$ additional, artificial edges of large capacity and cost to the graph G such that the optimal solution to the minimum-cost flow problem remains unchanged (and in particular is not supported on the artificial edges) and such that one can easily find a *feasible* flow $\boldsymbol{f}$ on the artificial edges such that $\boldsymbol{B}^\top \boldsymbol{f} = \boldsymbol{d}$ with bounded potential, that is, $\Phi(\boldsymbol{f}) = O(m \log m)$.

Given the current *feasible solution* $\boldsymbol{f}$, the potential reduction interior point method asks to find a circulation $\boldsymbol{\Delta}$, that is, a flow that satisfies $\boldsymbol{B}^\top \boldsymbol{\Delta} = \boldsymbol{0}$, such that $\Phi(\boldsymbol{f} + \boldsymbol{\Delta}) \le \Phi(\boldsymbol{f}) - m^{-o(1)}$. Given $\boldsymbol{\Delta}$, it then sets $\boldsymbol{f} \leftarrow \boldsymbol{f} + \boldsymbol{\Delta}$ and repeats. When $\Phi(\boldsymbol{f}) \le -200m \log mU$, we can terminate because then $\boldsymbol{c}^\top \boldsymbol{f} - F^* \le (mU)^{-10}$, at which point standard techniques let us round to an exact optimal flow.[13] Thus if we can reduce the potential by $m^{-o(1)}$ per iteration, the method terminates in $m^{1+o(1)}$ iterations.

Let us next describe how to find a circulation $\boldsymbol{\Delta}$ that reduces the potential sufficiently. Given the current flow $\boldsymbol{f}$, defining the gradient and lengths $\boldsymbol{g}(\boldsymbol{f}) \stackrel{\text{def}}{=} \nabla\Phi(\boldsymbol{f})$ and $\boldsymbol{\ell}(\boldsymbol{f})_e \stackrel{\text{def}}{=} (\boldsymbol{u}_e^+ - \boldsymbol{f}_e)^{-1-\alpha} + (\boldsymbol{f}_e - \boldsymbol{u}_e^-)^{-1-\alpha}$, and we let $\boldsymbol{L} \stackrel{\text{def}}{=} \text{diag}(\boldsymbol{\ell})$ be the matrix with these lengths on the diagonal and zeros elsewhere. We can then define the *min-ratio cycle* problem

$$\begin{aligned} &\text{minimize} \quad \boldsymbol{g}^\top \boldsymbol{\Delta} / \|\boldsymbol{L}\boldsymbol{\Delta}\|_1 \\ &\text{subject to} \quad \boldsymbol{B}^\top \boldsymbol{\Delta} = \boldsymbol{0} \end{aligned} \qquad (2)$$

Given any solution $\boldsymbol{\Delta}$ to this problem with $\boldsymbol{g}(\boldsymbol{f})^\top \boldsymbol{\Delta} / \|\boldsymbol{L}\boldsymbol{\Delta}\|_1 \le -\kappa$ for some $\kappa < 1/100$, scaled so that $\|\boldsymbol{L}\boldsymbol{\Delta}\|_1 = \kappa/50$. Then a direct Taylor expansion shows that $\Phi(\boldsymbol{f} + \boldsymbol{\Delta}) \le \Phi(\boldsymbol{f}) - \kappa^2/500$. Hence, it suffices to show that such a $\boldsymbol{\Delta}$ exists with $\kappa = \tilde{\Omega}(1)$, because then a data structure that returns an $m^{o(1)}$-approximate solution still has $\kappa = m^{-o(1)}$, which suffices. Fortunately, the *witness circulation* $\boldsymbol{\Delta}(\boldsymbol{f})^* = \boldsymbol{f}^* - \boldsymbol{f}$ satisfies $\boldsymbol{g}(\boldsymbol{f})^\top \boldsymbol{\Delta} / \|\boldsymbol{L}\boldsymbol{\Delta}\|_1 \le -\tilde{\Omega}(1)$.

We emphasize that it is essential for our data structure that the witness circulation $\boldsymbol{f}^* - \boldsymbol{f}$ yields a sufficiently good solution. This assumption ensures that good solutions to the min ratio cycle instances do not change arbitrarily between iterations. Further, even though the algorithm does not have access to the witness circulation $\boldsymbol{f}^* - \boldsymbol{f}$, it still knows how it changes between iterations as it can track changes in $\boldsymbol{f}$. We are able to leverage this guarantee to ensure our data structure succeeds for the updates coming from the IPM.

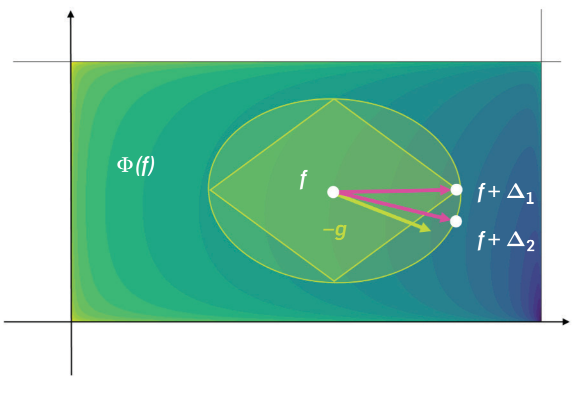
Finally, let us contrast our approach with previous approaches: previous analyses of IPMs solved ℓ_2 problems, that is, problems of the form given in Equation (2) but with the ℓ_1 norm replaced by a ℓ_2 norm (see Figure 1), which can be solved using a linear system. Karmarkar[20] shows that repeatedly solving ℓ_2 subproblems, the IPM converges in $\tilde{O}(m)$ iterations. Later analyses of path-following IPMs[31] showed that a sequence of $\tilde{O}(\sqrt{m})$ ℓ_2 subproblems suffices to give a high-accuracy solution. Surprisingly, we are able to argue that a solving sequence of $\tilde{O}(m)\ell_1$ minimizing subproblems of the form in Equation (2) suffice to give a high-accuracy solution to Equation (1). In other words, changing the ℓ_2 norm to an ℓ_1 norm does not increase the number of iterations in a potential reduction IPM. The use of an ℓ_1-norm-based subproblem gives us a crucial advantage: Problems of this form must have optimal solutions in the form of simple cycles—and our new algorithm finds approximately optimal cycles vastly more efficiently than any known approaches for ℓ_2 subproblems.

2.2. High-level overview of the data structure for dynamic min-ratio cycle

As discussed in the previous section, our algorithm computes a minimum-cost flow by solving a sequence of $m^{1+o(1)}$ min-ratio cycle problems $\min_{\boldsymbol{B}^\top\boldsymbol{\Delta}=0} \boldsymbol{g}^\top \boldsymbol{\Delta} / \|\boldsymbol{L}\boldsymbol{\Delta}\|_1$ to $m^{o(1)}$ multiplicative accuracy. Because our IPM ensures stability for lengths and gradients, and is even robust to approximations of lengths and gradients, we can show that over the course of the algorithm, we only need to update the entries of the gradients $\boldsymbol{g}$ and lengths $\boldsymbol{\ell}$ at most $m^{1+o(1)}$ total times.

Warm-up: A simple, static ALGORITHM. A simple approach to finding an $\tilde{O}(1)$ -approximate min-ratio cycle is the following: given our graph G, we find a probabilistic low stretch spanning tree T, that is, a tree such that for each edge $e = (u,v) \in G$, the stretch of e, defined as $\text{str}_e^{T,\ell} \stackrel{\text{def}}{=} \frac{\sum_{f\in T[u,v]} \ell(f)}{\ell(e)}$ where $T[u, v]$ is the unique path from u to v along the tree T, is $\tilde{O}(1)$ in expectation. Such a tree can be found in $\tilde{O}(m)$ time.[2, 3] This fact will allow us to argue that with probability

Figure 1. The IPM takes steps to minimize the potential $\Phi(f)$ by updating f to $f + \Delta$. Previous approaches suggest obtaining Δ by solving an ℓ_2 subproblem (here finding Δ_2 as the optimal step on an ellipsoid), but our approach obtains Δ by minimizing an ℓ_1 problem (illustrated by finding Δ_1 as the optimal step in a box). While the new strategy possibly makes less progress at a step, this allows us to find the step Δ more efficiently.



at least $\frac{1}{2}$, one of the tree cycles is an $\tilde{O}(1)$ -approximate solution to Equation (2).

Let $\boldsymbol{\Delta}^*$ be the optimal circulation that minimizes Equation (2), and assume w.l.o.g. that $\boldsymbol{\Delta}^*$ is a cycle that routes one unit of flow along the cycle. We assume for convenience, that edges on $\boldsymbol{\Delta}^*$ are oriented along the flow direction of $\boldsymbol{\Delta}^*$, that is, $\boldsymbol{\Delta}^* \in \mathbb{R}^E_{\geq 0}$. Then, for each edge $e = (u, v)$, the *fundamental tree cycle* of e in T, denoted $e \oplus T[v, u]$, is formed by edge e concatenated with the path in T from its endpoint v to u. To work again with vector notation, we denote by $\boldsymbol{p}(e \oplus T[v, u]) \in \mathbb{R}^E$ the vector that sends one unit of flow along the cycle $e \oplus T[v, u]$ in the direction that aligns with the orientation of e. A classic fact from graph theory now states that $\boldsymbol{\Delta}^* = \sum_{e:\boldsymbol{\Delta}^*_e>0} \boldsymbol{\Delta}^*_e \cdot \boldsymbol{p}(e \oplus T[v, u])$ (note that the tree paths used by adjacent off-tree edges cancel out, see Figure 2). In particular, this implies that $\boldsymbol{g}^\top \boldsymbol{\Delta}^* = \sum_{e:\boldsymbol{\Delta}^*_e>0} \boldsymbol{\Delta}^*_e \cdot \boldsymbol{g}^\top \boldsymbol{p}(e \oplus T[v, u])$.

From the guarantees of the low-stretch tree distribution, we know that the circulation $\boldsymbol{\Delta}^*$ is not stretched by too much in expectation. Thus, by Markov's inequality, with probability at least $\frac{1}{2}$, the circulation $\boldsymbol{\Delta}^*$ is not stretched by too much. Formally, we have that $\sum_{e:\boldsymbol{\Delta}^*_e>0} \boldsymbol{\Delta}^*_e \cdot \|\boldsymbol{L}\,\boldsymbol{p}(e \oplus T[v, u])\|_1 \leq \gamma \|\boldsymbol{L}\boldsymbol{\Delta}^*\|_1$ for $\gamma = \tilde{O}(1)$. Combining these insights, we can derive that

$$\frac{\boldsymbol{g}^\top \boldsymbol{\Delta}^*}{\|\boldsymbol{L}\boldsymbol{\Delta}^*\|_1} \geq \frac{1}{\gamma} \cdot \frac{\sum_{e:\boldsymbol{\Delta}^*_e>0} \boldsymbol{\Delta}^*_e \cdot \boldsymbol{g}^\top \boldsymbol{p}(e \oplus T[v, u])}{\sum_{e:\boldsymbol{\Delta}^*_e>0} \boldsymbol{\Delta}^*_e \cdot \|\boldsymbol{L}\,\boldsymbol{p}(e \oplus T[v, u])\|_1}$$
$$\geq \frac{1}{\gamma} \min_{e:\boldsymbol{\Delta}^*_e>0} \frac{\boldsymbol{g}^\top \boldsymbol{p}(e \oplus T[v, u])}{\|\boldsymbol{L}\,\boldsymbol{p}(e \oplus T[v, u])\|_1}$$

where the last inequality follows from the fact that $\min_{i\in[n]} \frac{\boldsymbol{x}_i}{\boldsymbol{y}_i} \leq \frac{\sum_{i\in[n]} \boldsymbol{x}_i}{\sum_{i\in[n]} \boldsymbol{y}_i}$ (recall also that $\boldsymbol{g}^\top \boldsymbol{\Delta}^*$ is negative). This tells us that for the edge e minimizing the expression on the right, the tree cycle $e \oplus T[v, u]$ is a γ-approximate solution to Equation (2), as desired. We can boost the probability of success of the above algorithm by sampling $\tilde{O}(1)$ trees $T_1, T_2, \ldots, T_s$ independently at random and conclude that w.h.p. one of the fundamental tree cycles approximately solves Equation (2).

Unfortunately, after updating the flow $\boldsymbol{f}$ to $\boldsymbol{f}'$ along such a fundamental tree cycle, we cannot reuse the set of trees $T_1, T_2, \ldots, T_s$ because the next solution to Equation (2) has to be found with respect to gradients $\boldsymbol{g}(\boldsymbol{f}')$ and lengths $\boldsymbol{\ell}(\boldsymbol{f}')$ depending on $\boldsymbol{f}'$ (instead of $\boldsymbol{g} = \boldsymbol{g}(\boldsymbol{f})$ and $\boldsymbol{\ell} = \boldsymbol{\ell}(\boldsymbol{f})$). But $\boldsymbol{g}(\boldsymbol{f}')$ and $\boldsymbol{\ell}(\boldsymbol{f}')$ depend on the randomness used in trees $T_1, T_2, \ldots, T_s$. Thus, naively, we have to recompute all trees, spending again $\Omega(m)$ time. But this leads to run-time $\Omega(m^2)$ for our overall algorithm which is far from our goal.

A dynamic approach. Thus we consider the data structure problem of maintaining an $m^{o(1)}$ approximate solution to Equation (2) over a sequence of at most $m^{1+o(1)}$ changes to entries of $\boldsymbol{g}$, $\boldsymbol{\ell}$. To achieve an almost linear time algorithm overall, we want our data structure to have an amortized $m^{o(1)}$ update time. Motivated by the simple construction above, our data structure will ultimately maintain a set of $s = m^{o(1)}$ spanning trees $T_1, \ldots, T_s$ of the graph G. Each cycle $\boldsymbol{\Delta}$ that is returned is represented by $m^{o(1)}$ off-tree edges and paths connecting them on some T_i.

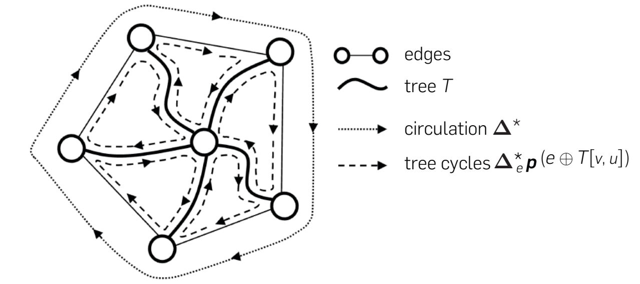
To obtain an efficient algorithm to maintain these trees T_i, we turn to a recursive approach. Each level of the recursion will *partially* construct a tree, by choosing a spanning forest of the vertices, and contracting the connected components of the forest. We obtain a tree by repeating forest selection-and-contraction until only a single vertex is left. Then, we compose the forest edges obtained at different levels, yielding a spanning tree of the original graph. At each level of the construction, our forest is probabilistic and only succeeds with constant probability at preserving the hidden witness circulation well enough. To preserve the witness with high probability, we construct $O(\log n)$ different forests at each level and recurse on them separately.

In each level of our recursion, we first reduce the number of vertices using a forest contraction and then the number of edges by making the contracted graph sparse. To reduce the number of vertices, we produce a *core graph* (the result of contracting forest components) on a subset of the original vertex set, and we then compute a *spanner* of the core graph which reduces the number of edges. The edge-reduction step is important to ensure the overall recursion reduces the graph size in each step, which is essential to obtaining almost linear running time in our framework.

Both the *core graph* and *spanner* at each level need to be maintained dynamically, and we ensure they are very stable under changes in the graphs at shallower levels in the recursion. In both cases, our notion of stability relies on some subtle properties of the interaction between data structure and hidden witness circulation.

We maintain a recursive hierarchy of graphs. At the top level of our hierarchy, for the input graph G, we produce $B = O(\log n)$ core graphs. To obtain each such core graph, for each $i \in [B]$, we sample a (random) forest F_i with $\tilde{O}(m/k)$ connected components for some size reduction parameter k. The associated core graph is the graph G/F_i which denotes G after contracting the vertices in the same components of F_i. We can define a map that lifts circulations $\hat{\boldsymbol{\Delta}}$ in the core graph G/F_i, to circulations $\boldsymbol{\Delta}$ in the graph G by routing flow along the contracted paths in F_i. The lengths in the core graph $\hat{\boldsymbol{\ell}}$ (again let $\hat{\boldsymbol{L}} = \mathrm{diag}(\hat{\boldsymbol{\ell}})$) are chosen to upper bound the length of circulations when mapped back into G such that $\|\hat{\boldsymbol{L}}\hat{\boldsymbol{\Delta}}\|_1 \geq \|\boldsymbol{L}\boldsymbol{\Delta}\|_1$. Crucially, we must ensure these new lengths $\hat{\boldsymbol{\ell}}$ do not stretch the witness circulation $\boldsymbol{\Delta}^*$ when mapped into G/F_i by too much, so we can recover it from G/F_i. To achieve this goal, we choose F_i to be a low-stretch forest, that is, a forest with properties similar to those of a low-stretch tree. In Section 2.3, we summarize the central aspects of our core graph construction.

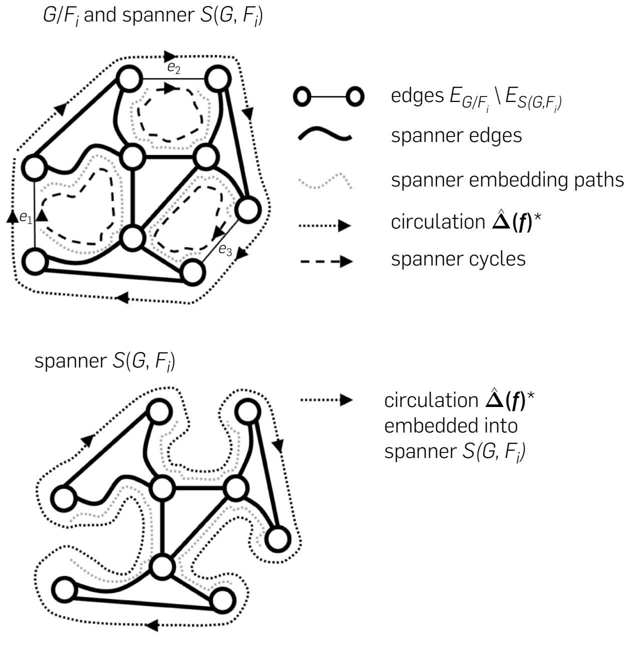
Figure 2. Illustrating the decomposition $\boldsymbol{\Delta}^* = \Sigma_{e:\boldsymbol{\Delta}^*_e>0} \boldsymbol{\Delta}^*_e \cdot \boldsymbol{p}(e \oplus T[v,u])$ of a circulation into fundamental tree cycles.


edges
tree T
circulation $\boldsymbol{\Delta}^*$
tree cycles $\boldsymbol{\Delta}^*_e \boldsymbol{p}(e \oplus T[v,u])$

While each core graph G/F_i now has only $\tilde{O}(m/k)$ vertices, it still has m edges which are too large for our recursion. To overcome this issue we build a spanner $\mathcal{S}(G, F_i)$ on G/F_i to reduce the number of edges to $\tilde{O}(m/k)$, which guarantees that for every edge $e = (u, v)$ that we remove from G/F_i to obtain $\mathcal{S}(G, F_i)$, there is a u-to-v path in $\mathcal{S}(G, F_i)$ of length $m^{o(1)}$. Ideally, we would now recurse on each spanner $\mathcal{S}(G, F_i)$, again approximating it with a collection of smaller core graphs and spanners. However, we face an obstacle: removing edges could destroy the witness circulation so that possibly no good circulation exists in any $\mathcal{S}(G, F_i)$. To solve this problem, we compute an explicit embedding $\Pi_{G/F_i \rightarrow \mathcal{S}(G,F_i)}$ that maps each edge $e = (u, v) \in G/F_i$ to a short u-to-v path in $\mathcal{S}(G, F_i)$. We can then show the following dichotomy: Let $\hat{\boldsymbol{\Delta}}^*$ denote the witness circulation when mapped into the core graph G/F_i. Then, *either* one of the edges $e \in E_{G/F_i} \setminus E_{\mathcal{S}(G,F_i)}$ has a spanner cycle consisting of e combined with $\Pi_{G/F_i \rightarrow \mathcal{S}(G,F_i)}(e)$ which is almost as good as $\hat{\boldsymbol{\Delta}}^*$, *or* re-routing $\hat{\boldsymbol{\Delta}}^*$ into $\mathcal{S}(G, F_i)$ roughly preserves its quality. Figure 3 illustrates this dichotomy. Thus, either we find a good cycle using the spanner, or we can recursively find a solution $\mathcal{S}(G, F_i)$ onthat almost matches $\hat{\boldsymbol{\Delta}}^*$ in quality. To construct our dynamic spanner with its efficient updates and strong stability guarantees under changes in the input graph, we design a new approach that diverges from other recent works on dynamic spanners.

Our recursion uses d levels, where we choose the size reduction factor k such that $k^d \approx m$ and the bottom level graphs have $m^{o(1)}$ edges. Note that since we build B trees on G and recurse on the spanners of $G/F_1, G/F_2, \ldots, G/F_B$, our recursive hierarchy has a branching factor of $B = O(\log n)$ at each level of recursion. Thus, choosing $d \leq \sqrt{\log n}$, we get $B^d = m^{o(1)}$ leaf nodes in our recursive hierarchy. Now, consider the forests $F_{i_1}, F_{i_2}, \ldots, F_{i_d}$ on the path from the top of our recursive hierarchy to a leaf node. We can patch these forests together to form a tree associated with the leaf node. For each of these trees, we maintain a link-cut tree data structure. Using this data structure, whenever we find a good cycle, we can route flow along it and detect edges where the flow has changed significantly. The cycles are either given by an off-tree edge or a collection of $m^{o(1)}$ off-tree edges coming from a spanner cycle. We call the entire construction a *branching tree chain*. In Section 2.4, we elaborate on the overall composition of the data structure.

Figure 3. Illustration of a dichotomy: either one of the edges $e \in E_{G/F_i} \setminus E_{S(G,F_i)}$ has a spanner cycle consisting of *e* combined with $\Pi_{G/F_i \rightarrow S(G,F_i)}(e)$ which is almost as good as $\hat{\Delta}(f)^*$, *or* re-routing $\hat{\Delta}(f)^*$ into $S(G, F_i)$ roughly preserves its quality.


G/F_i and spanner $S(G, F_i)$
e_2
e_1
e_3
edges $E_{G/F_i} \setminus E_{S(G,F_i)}$
spanner edges
spanner embedding paths
circulation $\hat{\Delta}(f)^*$
spanner cycles
spanner $S(G, F_i)$
circulation $\hat{\Delta}(f)^*$ embedded into spanner $S(G, F_i)$

What have we achieved using this hierarchical construction compared to our simple, static algorithm? First, consider the setting of an *oblivious adversary*, where the gradient and length update sequences and the optimal circulation after each update are fixed in advance (i.e., the adversary is oblivious of the algorithm's random choices). In this setting, we can show that our spanner-of-core graph construction can survive through $m^{1-o(1)}/k^i$ updates at level i. Meanwhile, we can rebuild these constructions in time $m^{1+o(1)}/k^{i-1}$, leading to an amortized cost per update of $km^{o(1)} \leq m^{o(1)}$ at each level. This gives the first dynamic data structure for our undirected min-ratio problem with $m^{o(1)}$ query time against an oblivious adversary.

However, our real problem is harder: the witness circulation in each round is $\boldsymbol{\Delta}(\boldsymbol{f})^* = \boldsymbol{f}^* - \boldsymbol{f}$ and depends on the updates we make to $\boldsymbol{f}$, making our sequence of subproblems adaptive. While we cannot show that our data structure succeeds against an adaptive adversary, we give a data structure that works against a restricted adaptive adversary. We show that the witness circulation $\boldsymbol{f}^* - \boldsymbol{f}$ lets us express the IPM as such a restricted adversary.

2.3. Building core graphs

In this section, we describe our core graph construction, which maps our dynamic undirected min-ratio cycle problem on a graph G with at most m edges and vertices into a problem of the same type on a graph with only $\tilde{O}(m/k)$ vertices and m edges, and handles $\tilde{O}(m/k)$ updates to the edges before we need to rebuild it.

Forest routings and stretches. To understand how to define the stretch of an edge e with respect to a forest F, it is useful to define how to *route* an edge e in F. Given a spanning forest F, every path and cycle in G can be mapped to G/F naturally (where we allow G/F to contain self-loops). On the other hand, if every connected component in F is rooted, where root^F_u denotes the root corresponding to a vertex $u \in V$, we can map every path and cycle in G/F back to G as follows. Let $P = e_1, \ldots, e_k$ be any (not necessarily simple) path in G/F where the preimage of every edge e_i is $e_i^G = (u_i^G, v_i^G) \in G$. The preimage of P, denoted P^G, is defined as the following concatenation of paths:

$$P^G \stackrel{\text{def}}{=} \bigoplus_{i=1}^{k} F[\mathrm{root}^F_{u_i^G}, u_i^G] \oplus e_i^G \oplus F[v_i^G, \mathrm{root}^F_{u_i^G}],$$

where we use $A \oplus B$ to denote the concatenation of paths A and B, and $F[a, b]$ to denote the unique ab-path in the forest F. When P is a circuit (that is, a not necessarily simple cycle), P^G is a circuit in G as well. One can extend these maps linearly to all flow vectors and denote the resulting operators as $\mathbf{\Pi}_F : \mathbb{R}^{E(G)} \to \mathbb{R}^{E(G/F)}$ and $\mathbf{\Pi}_F^{-1} : \mathbb{R}^{E(G/F)} \to \mathbb{R}^{E(G)}$. Since we let *G*/*F* have self-loops, there is a bijection between the edges of *G* and *G*/*F*, and thus $\mathbf{\Pi}_F$ acts like the identity function. Related routing schemes date back to Spielman-Teng[34] and are generally known as *portal routing*.

To make our core graph construction dynamic, the key operation we need to support is the dynamic addition of more root nodes, which results in forest edges being deleted to maintain the invariant each connected component has a root node. Whenever an edge is changing in *G*, we ensure that *G*/*F* approximates the changed edge well by forcing both its endpoints to become root nodes, which in turn makes the portal routing of the new edge trivial and this guarantees its stretch is 1.

For any edge $e^G = (u^G, v^G)$ in *G* with image *e* in *G*/*F*, we set $\hat{\boldsymbol{\ell}}_e^F$, the edge length of *e* in *G*/*F*, to be *an upper bound* on the length of the *forest routing* of *e*, that is, the path $F[\mathsf{root}^F_{u^G}, u^G] \oplus e^G \oplus F[v^G, \mathsf{root}^F_{u^G}]$. Meanwhile, we define $\widetilde{\mathsf{str}}_e \overset{\mathrm{def}}{=} \hat{\boldsymbol{\ell}}^F_e / \boldsymbol{\ell}_e$, as an overestimate on the stretch of *e* w.r.t. the forest routing. A priori, it is unclear how to provide a single upper bound on the stretch of every edge, as the root nodes of the endpoints are changing over time. Providing such a bound for every edge is important for us as the lengths in *G*/*F* could otherwise be changing too often when the forest changes. We guarantee these bounds by a scheme that makes auxiliary edge deletions in the forest in response to external updates, with the resulting additional roots chosen carefully to ensure the length of upper bounds.

Now, for any flow $\boldsymbol{f}$ in *G*/*F*, its length in *G*/*F* is at least the length of its pre-image in *G*, that is, $\left\|\boldsymbol{L}\mathbf{\Pi}_F^{-1}\boldsymbol{f}\right\|_1 \le \left\|\hat{\boldsymbol{L}}^F\boldsymbol{f}\right\|_1$. Let $\mathbf{\Delta}^*$ be the optimal solution to Equation (2). We will show later how to build *F* such that with constant probability $\|\hat{\boldsymbol{L}}^F\mathbf{\Delta}^*\|_1 \le \gamma\|\boldsymbol{L}\mathbf{\Delta}^*\|_1$ holds for some $\gamma = m^{o(1)}$, solving Equation (2) on *G*/*F* with edge length $\hat{\boldsymbol{\ell}}$ and properly defined gradient $\hat{\boldsymbol{g}}$ on *G*/*F* yields an $\frac{1}{\gamma}$-approximate solution for *G*. The gradient $\hat{\boldsymbol{g}}$ is defined so that the total gradient of any circulation $\mathbf{\Delta}$ on *G*/*F* and its preimage $\mathbf{\Pi}_F^{-1}\mathbf{\Delta}$ in *G* is the same, that is, $\hat{\boldsymbol{g}}^\top\mathbf{\Delta} = \boldsymbol{g}^\top\mathbf{\Pi}_F^{-1}\mathbf{\Delta}$. The idea of incorporating gradients into portal routing was introduced in Kyng et al.[24]; our version of this construction is somewhat different to allow us to make it dynamic efficiently.

Collections of low stretch decompositions (LSD). The first component of the data structure is constructing and maintaining forests of *F* that form a *Low Stretch Decomposition (LSD)* of *G*. Informally, a *k*-LSD is a rooted forest $F \subseteq G$ that decomposes *G* into *O*(*m*/*k*) vertex disjoint components. Given some positive edge weights $\boldsymbol{v} \in \mathbb{R}^E_{>0}$ and reduction factor *k* > 0, we compute a *k*-LSD *F* and length upper bounds $\hat{\boldsymbol{\ell}}^F$ of *G*/*F* that satisfy two properties:

1. $\widetilde{\mathsf{str}}^F_e = \hat{\boldsymbol{\ell}}^F_e / \boldsymbol{\ell}_{e^G} = \tilde{O}(k)$ for any edge $e^G \in G$ with image *e* in *G*/*F*, and
2. The weighted average of $\widetilde{\mathsf{str}}^F_e$ w.r.t. $\boldsymbol{v}$ is only $\tilde{O}(1)$, that is, $\sum_{e^G \in G} \boldsymbol{v}_{e^G} \cdot \widetilde{\mathsf{str}}^F_e \le \tilde{O}(1) \cdot \|\boldsymbol{v}\|_1$.

Item 1 guarantees that the solution to Equation (2) for *G*/*F* yields a $\tilde{O}(k)$-approximate one for *G*. However, this guarantee is not sufficient for our data structure, as our *B*-branching tree chain has $d \approx \log_k m$ levels of recursion and the quality of the solution from the deepest level would only be $\tilde{O}(k)^d \approx m^{1+o(1)}$-approximate.

Instead, we compute *k* different edge weight assignments $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ via multiplicative weight updates so that the LSDs $F_1, \ldots, F_k$ have $\tilde{O}(1)$ an average stretch on every edge in $G : \sum_{j=1}^k \widetilde{\mathsf{str}}^{F_j}_e = \tilde{O}(k)$, for all $e^G \in G$ with image *e* in *G*/*F*.

By Markov's inequality, for any fixed flow $\boldsymbol{f}$ in G, $\left\|\hat{\boldsymbol{L}}^{F_j}\boldsymbol{f}\right\|_1 \le \tilde{O}(1)\|\boldsymbol{L}\boldsymbol{f}\|_1$ holds for at least half the LSDs corresponding to $F_1, \ldots, F_k$. Taking *O*(log *n*) samples uniformly from $F_1, \ldots, F_k$, say $F_1, \ldots, F_B$ for *B* = *O*(log *n*) we get that with high probability

$$\min_{j \in [B]} \left\|\widetilde{\mathsf{str}}^{F_j} \circ \boldsymbol{L}\mathbf{\Delta}^*\right\|_1 \le \tilde{O}(1)\|\boldsymbol{L}\mathbf{\Delta}^*\|_1. \qquad (3)$$

That is, it suffices to solve Equation (2) on $G/F_1, \ldots, G/F_B$ to find an $\tilde{O}(1)$-approximate solution for *G*.

2.4. Maintaining a branching tree chain

Our branching chain is constructed as follows:

1. Sample and maintain *B* = *O*(log *n*) *k*-LSDs $F_1, F_2, \ldots, F_B$, and their associated core graphs G/F_i. Across *O*(*m*/*k*) updates at the top level, the forests F_i are *decremental*, that is, only undergo edge deletions (from root insertions), and will have $\tilde{O}(m/k)$ connected components.
2. Maintain spanners $\mathcal{S}(G, F_i)$ of the core graphs G/F_i, and embeddings $\Pi_{E(G/F_i) \to \mathcal{S}(G, F_i)}$, say with length increase $\gamma_\ell = m^{o(1)}$.
3. Recursively process the graphs $\mathcal{S}(G, F_i)$, that is, maintain LSDs and core graphs on those, and spanners on the contracted graphs, etc., for *d* total levels, with $k^d = m$.
4. Whenever a level *i* accumulates m/k^i total updates, hence doubling the number of edges in the graphs at that level, we rebuild levels *i*, *i* + 1, …, *d*.

Recall that on average, the LSDs stretch lengths by $\tilde{O}(1)$, and the spanners $\mathcal{S}(G, F_i)$ stretch lengths by γ_ℓ. Hence, the overall data structure stretches lengths by $\tilde{O}(\gamma_\ell)^d = m^{o(1)}$ (for appropriately chosen *d*).

We now discuss how to update the forests G/F_i and spanners $\mathcal{S}(G, F_i)$. Intuitively, every time an edge *e* = (*u*, *v*) is changed in *G*, we will delete $\tilde{O}(1)$ additional edges from F_i. This ensures that no edge's total stretch/routing-length increases significantly due to the deletion of *e*. As the forest F_i undergoes edge deletions, the graph G/F_i undergoes *vertex splits*, where a vertex has a subset of its edges moved to a newly created vertex. Thus, a key component of our data structure is to maintain spanners and embed-dings of graphs undergoing vertex splits (and edge insertions/ deletions). Importantly, the amortized recourse (number of changes) to the spanner $\mathcal{S}(G, F_i)$ is $m^{o(1)}$ independent of *k*, even though the average degree of G/F_i is Ω(*k*). Hence, on average Ω(*k*) edges will move in G/F_i per vertex split.

Overall, let every level have recourse $\gamma_r = m^{o(1)}$ (independent of *k*) per tree. Then each update at the top level induces $O(B\gamma_r)^d$ (as we branch using *B* forests/core graphs at each level of the recursion) updates in the data structure overall. Intuitively, for the proper choice of *d* = ω(1), both the total recourse $O(B\gamma_r)^d$ and approximation factor $\tilde{O}(\gamma_\ell)^d$ are $m^{o(1)}$ as desired.

2.5. Going beyond oblivious adversaries by using IPM guarantees

The precise data structure in the previous section only works for *oblivious adversaries*, because we used that if we sampled $B = O(\log n)$ LSDs, then w.h.p. there is a tree whose average stretch is $\tilde{O}(1)$ with respect to a *fixed flow* $\boldsymbol{f}$. However, since we are updating the flow along the circulations returned by our data structure, we influence future updates, so the optimal circulations our data structure needs to preserve, are not independent of the randomness used to generate the LSDs. To overcome this issue, we leverage the key fact that the flow $\boldsymbol{f}^* - \boldsymbol{f}$ is a good witness for the min-ratio cycle problem at each iteration.

To simplify our discussion, we focus on the role of the witness in ensuring the functioning of a single layer of core graph construction, which already captures the main ideas. We can prove that for any flow $\boldsymbol{f}$, $\boldsymbol{g}(\boldsymbol{f})^\top \boldsymbol{\Delta}(\boldsymbol{f})/(100m+\|\boldsymbol{L}(\boldsymbol{f})\boldsymbol{\Delta}(\boldsymbol{f})\|_1) \le -\tilde{\Omega}(1)$ holds where $\boldsymbol{\Delta}(\boldsymbol{f}) = \boldsymbol{f}^* - \boldsymbol{f}$. Then, the best solution to Equation (2) among the LSDs $G/F_1, \ldots, G/F_B$ maintains an $\tilde{O}(1)$ -approximation of the quality of the witness $\boldsymbol{\Delta}(\boldsymbol{f}) = \boldsymbol{f}^* - \boldsymbol{f}$ as long as

$$\min_{j\in[B]} \left\|\hat{\boldsymbol{L}}^{F_j}\boldsymbol{\Delta}(\boldsymbol{f})\right\|_1 \le \tilde{O}(1)\|\boldsymbol{L}(\boldsymbol{f})\boldsymbol{\Delta}(\boldsymbol{f})\|_1 + \tilde{O}(m). \quad (4)$$

In this case, let $\hat{\boldsymbol{\Delta}}$ be the best solution obtained from graphs $G/F_1, \ldots, G/F_B$. We have

$$\frac{\boldsymbol{g}(\boldsymbol{f})^\top\hat{\boldsymbol{\Delta}}}{\|\boldsymbol{L}(\boldsymbol{f})\hat{\boldsymbol{\Delta}}\|_1} \le \frac{\boldsymbol{g}(\boldsymbol{f})^\top\boldsymbol{\Delta}(\boldsymbol{f})}{\tilde{O}(1)\|\boldsymbol{L}(\boldsymbol{f})\boldsymbol{\Delta}(\boldsymbol{f})\|_1+\tilde{O}(m)} = -\tilde{\Omega}(1).$$

The additive $\tilde{O}(m)$ term is there for technical reasons that can be ignored for now. We define the *width* $\boldsymbol{w}(\boldsymbol{f})$ of $\boldsymbol{\Delta}(\boldsymbol{f})$ as $\boldsymbol{w}(\boldsymbol{f}) = 100 \cdot \mathbf{1} + |\boldsymbol{L}(\boldsymbol{f})\boldsymbol{\Delta}(\boldsymbol{f})|$. The name comes from the fact that $\boldsymbol{w}(\boldsymbol{f})_e$ is always at least $|\boldsymbol{\ell}(\boldsymbol{f})_e(\boldsymbol{f}^*_e - \boldsymbol{f}_e)|$ for any edge e. We show that the width is also slowly changing across IPM iterations, in that if the width changed by a lot, then the residual capacity of e must have changed significantly. This gives our data structure a way to predict which edges' contribution to the length of the witness flow $\boldsymbol{f}^* - \boldsymbol{f}$ could have significantly increased.

Observe that for any forest F_j in the LSD of G, we have $\left\|\hat{\boldsymbol{L}}^{F_j}\boldsymbol{\Delta}(\boldsymbol{f})\right\|_1 \le \left\|\widetilde{\mathsf{str}}^{F_j} \circ \boldsymbol{w}(\boldsymbol{f})\right\|_1$. Thus, we can strengthen Equation (4) and show that the IPM potential can be decreased by $m^{-o(1)}$ if

$$\min_{j\in[B]} \left\|\widetilde{\mathsf{str}}^{F_j} \circ \boldsymbol{w}(\boldsymbol{f})\right\|_1 \le \tilde{O}(1)\|\boldsymbol{w}(\boldsymbol{f})\|_1. \quad (5)$$

Equation (5) also holds with w.h.p. if the collection of LSDs is built after knowing $\boldsymbol{f}$. However, this does not necessarily hold after augmenting with $\boldsymbol{\Delta}$, an approximate solution to Equation (2).

Due to the stability of $\boldsymbol{w}(\boldsymbol{f})$, we have $\boldsymbol{w}(\boldsymbol{f}+\boldsymbol{\Delta})_e \approx \boldsymbol{w}(\boldsymbol{f})_e$ for every edge e whose length does not change a lot. For other edges, we update their edge length and force the stretch to be 1, that is, $\widetilde{\mathsf{str}}_e^{F_j} = 1$ via the dynamic LSD maintenance, by shortcutting the routing of the edge e at its endpoints. To distinguish between the earlier stretch values and those after updating edges, let us denote the former by $\widetilde{\mathsf{str}}_{\text{old}}^{F_j}$ and the latter by $\widetilde{\mathsf{str}}_{\text{new}}^{F_j}$. This gives that for any $j \in [B]$, the following holds:

$$\left\|\widetilde{\mathsf{str}}_{\text{new}}^{F_j} \circ \boldsymbol{w}(\boldsymbol{f}+\boldsymbol{\Delta})\right\|_1 \lesssim \left\|\widetilde{\mathsf{str}}_{\text{old}}^{F_j} \circ \boldsymbol{w}(\boldsymbol{f})\right\|_1 + \|\boldsymbol{w}(\boldsymbol{f}+\boldsymbol{\Delta})\|_1.$$

Using the fact that $\min_{j\in[B]}\left\|\widetilde{\mathsf{str}}_{\text{old}}^{F_j} \circ \boldsymbol{w}(\boldsymbol{f})\right\|_1 \le \tilde{O}(\|\boldsymbol{w}(\boldsymbol{f})\|_1)$, we have the following:

$$\min_{j\in[B]}\left\|\widetilde{\mathsf{str}}_{\text{new}}^{F_j} \circ \boldsymbol{w}(\boldsymbol{f}+\boldsymbol{\Delta})\right\|_1 \lesssim \tilde{O}(\|\boldsymbol{w}(\boldsymbol{f})\|_1) + \|\boldsymbol{w}(\boldsymbol{f}+\boldsymbol{\Delta})\|_1.$$

Thus, solving Equation (2) on the updated $G/F_1, \ldots, G/F_B$ yields a good enough solution for reducing IPM potential as long as the width of $\boldsymbol{w}(\boldsymbol{f}+\boldsymbol{\Delta})$ has not decreased significantly, that is, $\|\boldsymbol{w}(\boldsymbol{f})\|_1 \le \tilde{O}(1)\|\boldsymbol{w}(\boldsymbol{f}+\boldsymbol{\Delta})\|_1$.

If the solution on the updated graphs $G/F_1, \ldots, G/F_B$ does not have a good enough quality, we know by the above discussion that $\|\boldsymbol{w}(\boldsymbol{f}+\boldsymbol{\Delta})\|_1 \le 0.5\|\boldsymbol{w}(\boldsymbol{f})\|_1$ must hold. Then, we re-compute the collection of LSDs of G and solve Equation (2) on the new collection of $G/F_1, \ldots, G/F_B$ again. Because each recomputation reduces the ℓ_1 norm of the width by a constant factor, and all the widths are bounded by $\exp(\log^{O(1)} m)$ (which can be guaranteed by the IPM), there can be at most $\tilde{O}(1)$ such recomputations. At the top level, this only increases our runtime by $\tilde{O}(1)$ factors.

The full construction is along these lines, but more complicated since we recursively maintain the solutions on the spanners of each core graph $G/F_1, \ldots, G/F_B$. In the full version of the paper, we describe and analyze a multi-level rebuilding scheme that extends the above reasoning to our full data structure.

3. CONCLUSION

In this paper, we presented an almost-linear time algorithm for minimum-cost flow, maximum flow, and more generally, all convex single-commodity flows. Our work essentially settles the complexity of several fundamental and intensely studied problems in algorithms design. We hope that the ideas introduced in this work will spur further research in several directions, including simpler and faster algorithms for flow problems; hopefully resulting in a significant impact on algorithms in practice.

Acknowledgments

Li Chen was supported by NSF Grant CCF-2106444. Rasmus Kyng and Maximilian Probst Gutenberg have received funding from the grant "Algorithms and complexity for high-accuracy flows and convex optimization" (no. 200021 204787) of the Swiss National Science Foundation. Yang P. Liu was supported by NSF CAREER Award CCF-1844855 and NSF Grant CCF-1955039. Richard Peng was partially supported by NSF CAREER Award CCF-1846218 and NSERC Discovery Grant RGPIN-2022–03207. Sushant Sachdeva's research is supported by a Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant RGPIN-2018–06398 and an Ontario Early Researcher Award (ERA) ER21–16–284.

The authors thank the 2021 Hausdorff Research Institute for Mathematics Program Discrete Optimization. The authors are very grateful to Yin Tat Lee and Aaron Sidford for several useful discussions.

References

1. Abboud, A., Krauthgamer, R., Li, J., Panigrahi, D., Saranurak, T., Trabelsi, O. Breaking the cubic barrier for all-pairs max-flow: Gomory-hu tree in nearly quadratic time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022* (Denver, CO, USA, October 31–November 3, 2022), IEEE, NY, 884–895.
2. Abraham, I., Neiman, O. Using petal-decompositions to build a low stretch spanning tree. *SIAM J. Comput. 48*, 2 (2019), 227–248.
3. Alon, N., Karp, R.M., Peleg, D., West, D. A graph-theoretic game and its application to the k-server problem. *SIAM J. Comput. 24*, 1 (1995), 78–100.
4. Axiotis, K., Mądry, A., Vladu, A. Circulation control for faster minimum cost flow in unit-capacity graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)* (2020), IEEE, NY, 93–104.
5. Axiotis, K., Mądry, A., Vladu, A. Faster sparse minimum cost flow by electrical flow localization. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)* (2022), IEEE, NY, 528–539.
6. Bernstein, A., Gutenberg, M.P., Saranurak, T. Deterministic decremental sssp and approximate min-cost flow in almost-linear time. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)* (2022), IEEE, NY, 1000–1008.
7. Bernstein, A., Nanongkai, D., Wulff-Nilsen, C. Negative-weight single-source shortest paths in near-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)* (2022), IEEE, NY, 600–611.
8. Brand, J.v.d., Lee, Y.T., Liu, Y.P., Saranurak, T., Sidford, A., Song, Z., et al. Minimum cost flows, mdps, and ℓ_1-regression in nearly linear time for dense instances. In *STOC* (2021), ACM, NY, 859–869.
9. Brand, J.v.d., Lee, Y.-T., Nanongkai, D., Peng, R., Saranurak, T., et al. Bipartite matching in nearly-linear time on moderately dense graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)* (2020), IEEE, NY, 919–930.
10. Brand, J.v.d., Lee, Y.T., Sidford, A., Song, Z. Solving tall dense linear programs in nearly linear time. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)* (2020), ACM, NY, 775–788.
11. Cen, R., Li, J., Nanongkai, D., Panigrahi, D., Saranurak, T., Quanrud, K. Minimum cuts in directed graphs via partial sparsification. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)* (2021), IEEE, Denver, CO, 1147–1158.
12. Christiano, P., Kelner, J.A., Mądry, A., Spielman, D.A., Teng, S. Electrical flows, Laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)* (2011), ACM, NY, 273–282.
13. Daitch, S.I., Spielman, D.A. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the fortieth annual ACM symposium on Theory of computing* (2008), ACM, NY, 451–460.
14. Dantzig, G.B. Application of the simplex method to a transportation problem. In *Activity Analysis and Production and Allocation*, T.C. Koopmans (ed.), (1951), John Wiley and Sons, New York, 359–373.
15. Even, S., Tarjan, R.E. Network flow and testing graph connectivity. *SIAM J. Comput. 4*, 4 (1975), 507–518.
16. Gao, Y., Liu, Y.P., Peng, R. Fully dynamic electrical flows: Sparse maxflow faster than goldberg-rao. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)* (2022), IEEE, NY, 516–527.
17. Goldberg, A., Tarjan, R. Finding minimum-cost circulation by successive approximation. *Math. Oper. Res.* 15 (1990), 430–466.
18. Goldberg, A.V., Rao, S. Beyond the flow decomposition barrier. *J. ACM 45*, 5 (1998), 783–797. Announced at FOCS'97.
19. Hopcroft, J.E., Karp, R.M. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput. 2*, 4 (Dec 1973), 225–231.
20. Karmarkar, N. A new polynomial-time algorithm for linear programming. In *STOC* (1984), ACM, NY, 302–311.
21. Karzanov, A.V. On finding maximum flows in networks with special structure and some applications. *Matematicheskie Voprosy Upravleniya Proizvodstvom 5*, (1973), 81–94.
22. Kathuria, T., Liu, Y.P., Sidford, A. Unit capacity maxflow in almost $O(m^{4/3})$ time. In *61st IEEE Annual Symposium on Foundations of Computer Science (FOCS)* (2020), IEEE, NY, 119–130.
23. Kelner, J.A., Lee, Y.T., Orecchia, L., Sidford, A. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2014), Society for Industrial and Applied Mathematics, Portland, OR, 217–226.
24. Kyng, R., Peng, R., Sachdeva, S., Wang, D. Flows in almost linear time via adaptive preconditioning. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing* (2019), ACM, NY, 902–913.
25. Li, J., Nanongkai, D., Panigrahi, D., Saranurak, T., Yingchareonthawornchai, S. Vertex connectivity in poly-logarithmic max-flows. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing* (2021), ACM, NY, 317–329.
26. Li, J., Panigrahi, D. Approximate gomory–hu tree is faster than n–1 max-flows. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing* (2021), ACM, NY, 1738–1748.
27. Liu, Y.P., Sidford, A. Faster energy maximization for faster maximum flow. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2020)* (2020), ACM, NY, 803–814.
28. Mądry, A. Navigating central path with electrical flows: From flows to matchings, and back. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science* (2013), IEEE, NY, 253–262.
29. Mądry, A. Computing maximum flow with augmenting electrical flows. In *57th IEEE Annual Symposium on Foundations of Computer Science (FOCS)* (2016), IEEE Computer Society, NY, 593–602.
30. Peng, R. Approximate undirected maximum flows in o(mpolylog(n)) time. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms* (2016), SIAM, Philadelphia, PA, 1862–1867.
31. Renegar, J. A polynomial-time algorithm, based on newton's method, for linear programming. *Math. Program. 40*, 1 (1988), 59–93.
32. Sherman, J. Nearly maximum flows in nearly linear time. In *54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (2013), IEEE Computer Society, NY, 263–269.
33. Sherman, J. Area-convexity, ℓ_∞ regularization, and undirected multicommodity flow. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing* (2017), ACM, NY, 452–460.
34. Spielman, D.A., Teng, S. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)* (2004), ACM, NY, 81–90.
35. van den Brand, J., Gao, Y., Jambulapati, A., Lee, Y.T., Liu, Y.P., Peng, R., et al. Faster maxflow via improved dynamic spectral vertex sparsifiers. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2022)* (2022), ACM, NY, 543–556.

Li Chen (lichen@gatech.edu), Georgia Institute of Technology, Atlanta, GA, USA.

Rasmus Kyng and Maximilian Probst Gutenberg ({kyng, maxprobst }@inf.ethz.ch), ETH Zurich, Zurich, Switzerland.

Yang P. Liu (yangpliu@stanford.edu), Stanford University, Stanford, CA, USA.

Richard Peng (y5peng@uwaterloo.ca), University of Waterloo, Waterloo, ON, Canada.

Sushant Sachdeva (sachdeva@cs.toronto.edu), University of Toronto, Toronto, ON, Canada.