# Algorithms: New Techniques for Old Problems

Jason Li

September 11, 2020

## 1 Introduction

Algorithms is a broad, rich, and fast-growing field. For the latter half of last century, many subfields have emerged and grown in popularity, including fast sequential algorithms, dynamic algorithms, and approximation algorithms. Each developed its own tools and techniques to solve its main open problems, but for the most part, the methods between different areas remained separate.

More recently, as the whole of algorithms matured as a field, so has the line between different fields thinned over time. Problems were investigated in multiple settings at once, and techniques from multiple subfields were combined to navigate the uncharted territory. Amazingly, this cross-pollination of techniques has given birth to new, interdisciplinary methods and insights that have advanced our understanding of algorithms as a whole in unparalleled ways.

One illustrative case study is the NP-hard problem Sparsest Cut, one of the central problems in approximation algorithms with rich connections to other topics such as multicommodity flow, spectral graph theory, and geometric embeddings. In the early days of the Sparsest Cut problem, all of the action was concentrated on improving the approximation factor, producing a long line of work culminating in the celebrated $O(\sqrt{\log n})$-approximation algorithm of Arora, Rao, and Vazirani. Since then, faced with the difficulty of any further improvement, the trend has shifted towards approximation algorithms that also run in near-linear time, bringing the problem into the context of fast graph algorithms, a field which has historically remained largely separate from traditional approximation algorithms. Along the way, the technique of *expander decompositions* was developed as a way to incorporate sparsest cut-based ideas in fast graph algorithms, and has proved central to recent major developments in fast algorithms in the sequential, dynamic, and distributed settings.

This thesis focuses on this modern approach to algorithm design—drawing techniques from multidisciplinary fields—with the goal of resolving long-standing open problems that have resisted individual approaches in the past. In doing so, we simultaneously demonstrate the power of unifying algorithmic techniques as a whole, and grant closure to problems that have baffled researchers in the past. Our work can be viewed a step towards a revolution in algorithm design, where distinct areas of algorithms are no longer held separate from each other, but now join forces to unite the entire field of algorithms.

### 1.1 Problems Studied

In this proposal, all graphs are undirected with $n$ vertices and $m$ edges. They will always either be unweighted or weighted by nonnegative edge weights.

**Deterministic expander decomposition.** In Section 2.1, we consider the problem of computing an *expander decomposition* of a graph deterministically, and provide the first such algorithm in near-linear time. The problem definition and algorithm are as follows:

**Definition 1.1** (($\epsilon, \phi$)-expander decomposition)**.** *Given an unweighted graph $G = (V, E)$, a partition*
$\{V_1, \ldots, V_k\}$ *of $V$ is an* $(\epsilon, \phi)$-expander decomposition *of $G$ if*

1. *For all $1 \le i \le k$, the graph $G[V_i]$ has conductance at least $\phi$,*
2. *The total number of inter-cluster edges, $|E[V_1, \ldots, V_k]|$, is at most $\epsilon m$.*

**Theorem 1.2** (Deterministic expander decomposition)**.** *There is a deterministic algorithm that, given an unweighted graph $G = (V, E)$ and parameters $\epsilon \in (0, 1]$ and $1 \le r \le O(\log m)$, computes a $(\epsilon, \phi)$-expander decomposition of $G$ with $\phi = \Omega(\epsilon/(\log m)^{O(r^2)})$, in time $O\left(m^{1+O(1/r)+o(1)} \cdot (\log m)^{O(r^2)}\right)$. Setting $r = (\log n)^{1/3}$, we obtain $\phi = \epsilon/n^{o(1)}$ and time $O(m^{1+o(1)})$.*

In the past decade, expander decompositions have seen a surge in applications to fast graph algorithms, including the min-cut problem, which we consider next.

**Deterministic min-cut.** In Section 2.2, we investigate the problem of computing the (global) mincut of a graph deterministically. Using our results on expander decomposition, we improve upon the current, long-standing bounds for the problem to polylog$(n)$ many $s$–$t$ max-flow calls, and then all the way to near-linear time.

**Theorem 1.3** (Deterministic min-cut)**.** *Fix any constant $\epsilon > 0$. There is a deterministic min-cut algorithm for weighted undirected graphs that makes* polylog$(n)$ *calls to $s$–$t$ max-flow on a weighted undirected graph with $O(n)$ vertices and $O(m)$ edges, and runs in $O(m^{1+\epsilon})$ time outside these max-flow calls.*[1] *If the original graph $G$ is unweighted, then the inputs to the max-flow calls are also unweighted. Using the current fastest deterministic max-flow algorithms on unweighted (multi-)graphs (Liu and Sidford [LS20]) and weighted graphs (Goldberg and Rao [GR98]) respectively, this implies a deterministic min-cut algorithm for unweighted (multi-)graphs in $m^{4/3+o(1)}$ time and for weighted graphs in $\tilde{O}(m \cdot \min(\sqrt{m}, n^{2/3}))$ time.*

**Theorem 1.4** (Deterministic min-cut)**.** *There is a deterministic algorithm that computes the min-cut of a weighted, undirected graph in $m^{1+o(1)}$ time.*

**Shortest path in parallel.** In Section 3, we consider the problem of computing $(1+\epsilon)$-approximate single-source shortest paths (SSSP) in the parallel PRAM model. By viewing the problem from the perspective of *continuous optimization*, we provide the first truly work-efficient algorithm for this problem.

**Theorem 1.5** (Parallel SSSP)**.** *There exists a parallel algorithm that, given an undirected graph with nonnegative weights, computes a $(1+\epsilon)$-approximate single-source shortest path tree in $m$* polylog$(n)\,\epsilon^{-2}$ *work and* polylog$(n)\,\epsilon^{-2}$ *time in the PRAM model.*

[1]The exponent in the polylog$(n)$ term denoting the number of max-flow computations is a function of $\epsilon$.

# 2 Graph Conductance, Balance, and Cut Problems

Historically, the study of graph cut/flow problems has mainly fallen under two separate categories: fast, exact algorithms and polynomial-time approximation algorithms. For problems which admit polynomial-time algorithms, such as $s$–$t$ max-flow, global (edge)-min-cut, and global vertex-min-cut, the aim was towards faster algorithms for these problems, with near-linear time algorithms as the ultimate goal. On the other hand, for NP-hard problems such as conductance, minimum bisection, and multi-commodity flow, efforts were concentrated towards approximation factors and hardness of approximation, with landmark results including the $O(\sqrt{\log n})$-approximate SDP-based algorithm for conductance by Arora, Rao, and Vazirani.

In the last two decades, there has been a revolutionary trend in adopting techniques from both branches simultaneously, using expanders and conductance-based techniques to devise new, fast algorithms for (exact) graph cut/flow problems. This movement was pioneered by Spielman and Teng [ST04], who developed one of the first variants of *expander decompositions* to solve Laplacian systems and $\ell_2$-cost flows (exactly) in near-linear time. In hindsight, it was not at all obvious why polylogarithmic approximations to conductance-related problems would help at all in the exact setting, which makes the field and its recent popularity all the more remarkable.

This chapter focuses on our contributions in this area.

## 2.1 Deterministic Expander Decomposition

Of course, in order to apply conductance-based algorithms in the fast, exact setting, the algorithms need to be fast themselves. The most well-known breakthrough in the setting of fast conductance algorithms was the *cut-matching game* framework of Khandekar, Rao and Vazirani [KKOV07], who showed that a polylogarithmic approximation to conductance can be computed using only $O(\log n)$ calls to (approximate) max-flow. Together with the near-linear time algorithms for approximate max-flow in the past decade, this opened the door to near-linear time algorithms for graph conductance-based problems. The one that has seen the most applications is computing an *expander decomposition* of a graph: removing a small number of edges so that each connected component has large conductance.

**Definition 2.1** (Conductance). *Given a graph $G = (V, E)$, the conductance $\Phi(G)$ is the expression*

$$\min_{\substack{S\subseteq V:\\ 0<\mathbf{vol}(S)<\mathbf{vol}(V)/2}} \frac{|\partial S|}{\mathbf{vol}(S)},$$

*where* $\mathbf{vol}(S) := \sum_{v\in S}\deg(v)$.

**Definition 1.1** (($\epsilon, \phi$)-expander decomposition). *Given an unweighted graph $G = (V, E)$, a partition*
$\{V_1, \ldots, V_k\}$ *of $V$ is an* $(\epsilon, \phi)$-expander decomposition *of $G$ if*

1. *For all $1 \le i \le k$, the graph $G[V_i]$ has conductance at least $\phi$,*
2. *The total number of inter-cluster edges, $|E[V_1, \ldots, V_k]|$, is at most $\epsilon m$.*

This simple concept, first popularized by Spielman and Teng in their seminal paper on solving Laplacian systems, has proven invaluable in recent breakthroughs in fast graph algorithms in many areas, from sequential to dynamic to distributed algorithms. The simple, tried-and-true method in applying expander decompositions is to first consider the special case when the input graph is an expander, which is often a much easier problem. Then, for general graphs, one would commonly run expander decomposition, solve the problem separately on each expander, and apply recursion on an instance a constant factor smaller. Recent landmark results that successfully apply this technique include an algorithm by Nanongkai et al. [NS17] for dynamic connectivity with improved worst-case update time, and a fast algorithm for $(1+\epsilon)$-approximate $\ell_p$-cost flows by Adil et al. [AKPS19].

Prior to our work, the biggest drawback to expander decomposition-based algorithms was that the only near-linear time algorithms to compute an expander decomposition were randomized. In joint work with Chuzhoy, Gao, Nanongkai, Peng, and Saranurak [CGL$^+$20], we develop the first near-linear time *deterministic* algorithm for expander decomposition. Previously, not even a near-linear time algorithm for approximate *conductance* was known.

**Theorem 1.2** (Deterministic expander decomposition)**.** *There is a deterministic algorithm that, given an unweighted graph $G = (V, E)$ and parameters $\epsilon \in (0, 1]$ and $1 \le r \le O(\log m)$, computes a $(\epsilon, \phi)$-expander decomposition of $G$ with $\phi = \Omega(\epsilon/(\log m)^{O(r^2)})$, in time $O\left(m^{1+O(1/r)+o(1)} \cdot (\log m)^{O(r^2)}\right)$. Setting $r = (\log n)^{1/3}$, we obtain $\phi = \epsilon/n^{o(1)}$ and time $O(m^{1+o(1)})$.*

Our strategy, at a high level, is to derandomize the aforementioned cut-matching game. The algorithm is recursive, making a single cut-matching game call many recursive cut-matching games in “parallel”, so that the sizes of the recursive instances do not blow up.

Together with the existing framework of Nanongkai et al., we automatically obtain a *deterministic* algorithm for dynamic connectivity with subpolynomial worst-case update time, breaking the $\tilde{O}(\sqrt{n})$ barrier for this problem since the 1980s.

**Corollary 2.2.** *There is a deterministic algorithm that, given an $n$-vertex graph $G$ undergoing edge insertions and deletions, maintains a minimum spanning forest of $G$ with $n^{o(1)}$ worst-case update time.*

Through other standard techniques, we also obtain faster deterministic algorithms for other fundamental graph problems for which the only previous near-linear time algorithms were randomized:

**Corollary 2.3.** *There are deterministic algorithms that, given an undirected graph, compute*

1. *$(1+\epsilon)$-approximate max-flow/min-cut in $m^{1+o(1)}\epsilon^{-2}$ time,*
2. *$n^{o(1)}$-approximate sparsest cut in $m^{1+o(1)}$ time,*
3. *$(1+\epsilon)$-approximate Laplacian solvers in $m^{1+o(1)} \log(1/\epsilon)$ time.*

Finally, in subsequent work, we obtain many new algorithms centered around expander decomposition, which we discuss in the next section.

## 2.2 Deterministic Min-cut

The (global) min-cut of an undirected, weighted graph is a minimum weight subset of edges whose removal disconnects the graph. Finding the min-cut of a graph is one of the central problems in combinatorial optimization and has been extensively studied since the pioneering work of Gomory and Hu. A landmark result of Karger [Kar00] established a near-linear time randomized algorithm for this problem, and the author famously posed as an open question whether a fast, *deterministic* algorithm exists. For almost twenty years, no progress had been made towards even partially answering this question, until the breakthrough result of Kawarabayashi and Thorup [KT19] who achieved a deterministic, near-linear time algorithm on *simple*, unweighted graphs. Their key conceptual contribution introduces a simple but surprisingly deep connection between two popular graph parameters: they relate the min-cut of a graph to its *conductance*, a concept popularized by the sparsest cut problem in approximation algorithms.

### 2.2.1 Distinguishing Unbalanced and Balanced Cuts

In a joint paper with Debmalya Panigrahi [LP20], we further investigate this new, promising connection, and establish the first improvement to the problem for undirected, weighted graphs since the early 1990s. We show that it can be solved in polylogarithmic calls to $s$–$t$ max-flow. Note that a reduction from global min-cut to $s$–$t$ max-flow/min-cut is not at all obvious, even in the randomized setting: finding two vertices on opposite sides of the min-cut is challenging, especially if the global min-cut only has a few vertices on one side.

**Theorem 1.3** (Deterministic min-cut). *Fix any constant $\epsilon > 0$. There is a deterministic min-cut algorithm for weighted undirected graphs that makes* $\operatorname{polylog}(n)$ *calls to $s$–$t$ max-flow on a weighted undirected graph with $O(n)$ vertices and $O(m)$ edges, and runs in $O(m^{1+\epsilon})$ time outside these max-flow calls.*[2] *If the original graph $G$ is unweighted, then the inputs to the max-flow calls are also unweighted. Using the current fastest deterministic max-flow algorithms on unweighted (multi-)graphs (Liu and Sidford [LS20]) and weighted graphs (Goldberg and Rao [GR98]) respectively, this implies a deterministic min-cut algorithm for unweighted (multi-)graphs in $m^{4/3+o(1)}$ time and for weighted graphs in $\tilde{O}(m \cdot \min(\sqrt{m}, n^{2/3}))$ time.*

Our main insight can be summarized as follows: either the global min-cut in the graph is *unbalanced*, in that it has few—say, polylogarithmic—vertices on one side, for which we develop a "local" algorithm for this case, or the graph has low *conductance*—less than $1/\operatorname{polylog}(n)$—in which case computing an expander decomposition will break the graph into smaller pieces. One surprising feature of our algorithm is that, like Kawarabayashi and Thorup, we only need to consider the *approximate* conductance of a graph, even though our initial problem concerns the *exact* min-cut of the graph.

**Unbalanced case.** For the unbalanced case, our main technical contribution is an algorithm that solves the following problem, which we name the *minimum isolating cuts*, for any subset $R \subseteq V$ of vertices:

**Definition 2.4** (Minimum isolating cuts). *Consider a weighted, undirected graph $G = (V, E)$ and a subset $R \subseteq V$ ($|R| \geq 2$). The* minimum isolating cuts *for $R$ is a collection of sets $\{S_v : v \in R\}$ such that for each vertex $v \in R$, the set $S_v$ satisfies $S_v \cap R = \{v\}$ and has the minimum value of $w(\partial S'_v)$ over all sets $S'_v$ satisfying $S'_v \cap R = \{v\}$.*

**Theorem 2.5.** *Given a weighted, undirected graph $G = (V, E)$ and a subset $R \subseteq V$ ($|R| \geq 2$), there is an algorithm that computes the minimum isolating cuts for $R$ using $\lceil \log_2 |R| \rceil$ calls to $s$–$t$*

[2]The exponent in the $\operatorname{polylog}(n)$ term denoting the number of max-flow computations is a function of $\epsilon$.

*max-flow on unweighted graphs of $O(n)$ vertices and $O(m)$ edges, and takes $\tilde{O}(m)$ deterministic time outside of the max-flow calls.*

Using Theorem 2.5, we can now solve the unbalanced case by the lemma below, where we set $k = \text{polylog}(n)$, and imagine $U = V$ for now.

**Lemma 2.6** (Unbalanced case)**.** *Consider a graph $G = (V, E)$, a parameter $k \geq 1$, and a set $U \subseteq V$ of vertices such that $|U| > k$ and the smaller side $S$ for some min-cut satisfies $|S \cap U| \leq k$. Then, we can compute the min-cut in $k^{O(1)}\text{polylog}(n)$ many $s$–$t$ max-flow computations plus $\tilde{O}(m)$ deterministic time.*

For the proof of this lemma, observe that if we sample each vertex in $U$ with probability $1/k$ and let $R$ be our sample, then with constant probability, $|R \cap U| = 1$. Moreover, this random sampling can be efficiently derandomized with a factor $k^{O(1)}\text{polylog}(n)$ overhead using standard derandomization techniques. Finally, we need to ensure that $|R| \geq 2$, which is not too difficult.

**Balanced case.** For the balanced case, assume that each side $S$ of every min-cut satisfies $|S \cap R| > \text{polylog}(n)$. Here, our solution is not to solve the min-cut outright, but to make "progress" in a different way: we "sparsify" $R$ by replacing it with a subset $R' \subseteq R$ of at most half the size, such that if $R$ intersects both sides of some target min-cut in more than $\text{polylog}(n)$ vertices, then $R'$ intersects both sides of the same min-cut in at least 1 vertex. Our main insight is that computing an appropriate expander decomposition and selecting an arbitrary subset of vertices in each expander produces the desired sparsification.

**Lemma 2.7** (Balanced case: sparsification of $U$)**.** *Fix any constant $\epsilon > 0$. Then, there is a constant $C > 0$ (depending on $\epsilon$) such that the following holds. Consider a graph $G = (V, E)$, a parameter $\phi \leq 1/(C \log^C n)$, and a set $U \subseteq V$ of vertices such that some min-cut $(S_1, S_2)$ satisfies $|S_i \cap U| \geq (1 + 1/\phi)^3$ for $i = 1, 2$. Then, we can compute in deterministic $O(m^{1+\epsilon})$ time a set $U' \subseteq U$ with $|U'| \leq |U|/2$ such that $S_i \cap U' \neq \emptyset$ for $i = 1, 2$.*

With Lemmas 2.6 and 2.7, the deterministic algorithm for Theorem 1.3 proceeds as follows. We begin the algorithm with $R = V$ and repeatedly sparsify $R$ according to Lemma 2.7, which can be performed at most $O(\log n)$ times before $|R| \leq \text{polylog}(n)$. If, for each intermediate $R$, we always have $|S \cap R| > \text{polylog}(n)$ for each side $S$ of some min-cut, then by the sparsification guarantee, the final $R$ still intersects both sides of some target min-cut. But since $|R| \leq \text{polylog}(n)$ now, we can simply iterate over all pairs $s, t \in R$ and compute a min $s$–$t$ cut in $G$ for each such pair. Of course, it might also happen that in an intermediate step, $|S \cap R| < \text{polylog}(n)$, and we are in the unbalanced case at that stage. So, we also run the algorithm of Lemma 2.6 for the unbalanced case described earlier in each step of the sparsification procedure.

### 2.2.2 Deterministic Skeleton Graph and a Near-Linear Time Algorithm

We further explore the min-cut problem from the direction of graph conductance, and in joint work with my team at Microsoft Research Redmond (not yet submitted), we manage to fully resolve Karger's long-standing open problem, designing an algorithm that solves deterministic min-cut for undirected, weighted graphs in $m^{1+o(1)}$ time.

**Theorem 1.4** (Deterministic min-cut)**.** *There is a deterministic algorithm that computes the min-cut of a weighted, undirected graph in $m^{1+o(1)}$ time.*

This algorithm deviates from our previous min-cut algorithm, and tackles the problem through Karger's original approach. Karger's min-cut algorithm has a single randomized component: computing a *skeleton* of the original graph, after which the algorithm is deterministic and near-linear time.

**Definition 2.8** (Skeleton)**.** *Given a weighted, undirected graph $G$, an unweighted graph $H$ is called a* skeleton *of $G$ if*

1. *$H$ has $m^{1+o(1)}$ edges,*
2. *the min-cut of $H$ is $m^{o(1)}$,*
3. *the min-cut in $G$ corresponds (under the same vertex partition) to a $(7/6)$-approximate min-cut of $H$.*

**Theorem 2.9** ([Kar00])**.** *Given a weighted, undirected graph $G$ and a skeleton graph $H$, there is a deterministic algorithm that computes the min-cut of $G$ in $m^{1+o(1)}$ time.*

To compute the skeleton graph $H$, Karger actually computes a $(1+\epsilon)$-approximate cut-sparsifier (for $\epsilon = 1/6$) whose edges are all weighted the same weight $\Omega(\lambda/\log n)$, where $\lambda$ is the min-cut of $G$. It is easy to see that such a uniformly-weighted cut-sparsifier is also a skeleton graph (after unweighting its edges).

**Definition 2.10.** *A graph $H$ is a $(1+\epsilon)$-approximate cut-sparsifier of a weighted, undirected graph $G = (V, E)$ if, for all subsets $\emptyset \subsetneq S \subsetneq V$,*

$$(1-\epsilon)w(\partial_G S) \leq w(\partial_H S) \leq (1+\epsilon)w(\partial_G S). \tag{1}$$

**Theorem 2.11** ([BK15])**.** *Given a weighted, undirected graph $G$, sample each edge $e$ with probability $\frac{C\log n}{\epsilon^2} \cdot \frac{w(e)}{\lambda}$ for a large enough constant $C > 0$, and weight all sampled edges by $\frac{\epsilon^2\lambda}{C\log n}$. Then, the graph $H$ is a $(1+\epsilon)$-approximate cut sparsifier of $G$ w.h.p.*

The classic *cut-counting* analysis of Benczur and Karger takes a union bound over all cuts $\emptyset \subsetneq S \subsetneq V$ in a clever way. For each $\alpha \geq 1$, there are at most $\binom{n}{2\alpha}$ many cuts $\emptyset \subsetneq S \subsetneq V$ with $w(\partial_G S) \approx \alpha\lambda$, and each of these fails (1) with probability at most $n^{-C'\alpha}$ for an arbitrarily large constant $C' > 0$ (depending on $C$). A union bound is therefore first taken over each $\alpha$, and then over a collection of $\alpha \geq 1$ that covers all the cuts.

Derandomizing cut-sparsifier constructions has been notoriously difficult in the past, since there are an exponential number of cuts to union bound over; even the cut-counting analysis of Benczur-Karger has at least a polynomial number of approximate min-cuts to consider. In a random sampling algorithm, one does not need to explicitly compute these approximate min-cuts in order to establish a union bound, while for a deterministic algorithm, even *checking* whether a computed subset of edges is a sparsifier requires iterating over the approximate min-cuts, which is too slow.

Our solution is to develop a *structural representation* of all approximate min-cuts of a graph which allows us to establish a union-bound proof that is more efficient and amenable to derandomization. We compute a *hierarchy* of *expander decompositions* of the graph, and represent each approximate min-cut by how it evolves when traveling down the hierarchy. We show that on each level of the hierarchy, the cut can only "shift" in a small number of vertices, thus developing a compact representation of all approximate min-cuts of a graph, a concept that extends beyond the specific problem of deterministic min-cut, and is bound to have further applications elsewhere.

**Theorem 2.12.** *There is a deterministic algorithm to compute a skeleton graph $H$ in $m^{1+o(1)}$ time.*

Combining the above theorem with Theorem 2.9 proves Theorem 1.4.

## 2.3 Future work

**Deterministic graph sparsification.** One immediate future question in the area of deterministic cut algorithms is on deterministic graph *sparsification*, whether a deterministic variant of Theorem 2.11 exists. A skeleton graph is useful for the min-cut problem, but graph sparsifiers have many more applications to other fundamental graph problems.

**Question 2.13.** *Is there a deterministic algorithm to compute a $(1+\epsilon)$-approximate cut sparsifier with $n^{1+o(1)}$ edges of a given weighted, undirected graph in $m^{1+o(1)}$ time? What about a $(1+\epsilon)$-approximate spectral sparsifier?*

The sparsifier produced in our deterministic min-cut result only preserve *large* cuts up to $n^{o(1)}$ factor, not a $(1+\epsilon)$-factor, since the structural representation via expander decompositions is too expensive for them. Hence, we need a new technique to handle these cuts. Like with the deterministic min-cut problem, solving this one would also introduce new insights about the structure of graph cuts that has potential application elsewhere. Our hope is that *spectral*-based algorithms could be useful here, since they capture the exponentially many large cuts in a compact, matrix-based way. At the same time, current derandomization methods for spectral-based algorithms are generally slow, many of them requiring matrix multiplication at least, so any new techniques developed may have further applications to matrix-based derandomization.

**Vertex connectivity.** There are many other graph cut problems whose optimal running times are not known even in the randomized setting. One that we hope to make progress on, using techniques we have developed for deterministic min-cut, is *(global) vertex connectivity.* The current best algorithm, even for the unweighted setting, is $\tilde{O}(mn)$ from the 1990s.

**Question 2.14.** *Is there a (possibly randomized) algorithm to compute the global vertex connectivity of a given unweighted, undirected graph in $o(mn)$ time?*

Here, we believe that the idea of distinguishing balanced versus unbalanced, an idea that has only recently been truly explored, will prove fruitful towards breaking this long-standing barrier. One source of difficulty for vertex cuts, as opposed to edge cuts, is that a vertex cut is no longer a bipartition of the vertices: it is a tripartition $(A, S, B)$ where $S$ is the vertex cut. In our edge min-cut problem, we wanted our set $R$ to intersect both sides of some min-cut, but in the vertex min-cut setting, we want $R$ to intersect the two sides $A$ and $B$ *and be disjoint from $S$.* The latter property is harder to ensure, but we remain optimistic that there is a solution around it.

**Directed (edge) min-cut.** Another similar problem in this vein is global min-cut for *directed* graphs, which we also hope to investigate.

**Question 2.15.** *Is there an algorithm to compute the global vertex connectivity of a given unweighted, undirected graph in $o(mn)$ time?*

The main challenge in the direction version is that the minimum isolating cuts algorithm of Theorem 2.5 breaks down. We believe establishing a fast minimum isolating cuts algorithm in directed graphs for the unbalanced case becomes the main challenge. For the balanced case, directed expander decompositions have been studied before, though they come with extra hurdles to overcome.

**Gomory-Hu tree.** Finally, a more ambitious problem to consider is computing the *Gomory-Hu tree* for unweighted graphs; in particular, proving that a sublinear number of max-flows is sufficient to compute one would be the first improvement in over fifty years for weighted graphs. (Faster algorithms are known if the graph is unweighted.)

**Question 2.16.** *Is there an algorithm to compute the Gomory-Hu tree of a given weighted, undirected graph in $o(n)$ calls to $s$–$t$ max-flow?*

Again, we hope that the minimum isolating cuts procedure will help. In particular, if all $s$–$t$ min-cuts are unbalanced between any $s, t \in V$, then we can almost directly apply Theorem 2.5 to obtain a faster algorithm. Handling the balanced case seems to be the main challenge: now, we are not keeping track of just the (global) min-cut, but *all* $s$–$t$ min-cuts.

## 3 Iterative Optimization, Flows, and Parallel Algorithms

In the past, the maximum flow and shortest path problems have largely been regarded as *discrete optimization* problems. Indeed, when all flow constraints are integral, the maximum flow can also be assumed to be integral, so it can be decomposed into a number of discrete paths. Classic, flow-augmenting algorithms such as Ford-Fulkerson produce this decomposition inherently as part of their algorithm.

Only in the past decade has there been a shift towards viewing these problems *continuously*, and employing *iterative* methods from *continuous optimization* to solve these problems faster. This new perspective has led to many ground-breaking results, from Sherman's $(1+\epsilon)$-approximate max-flow algorithm based on gradient descent to Lee and Sidford's interior-point-based algorithm for (exact) min-cost flow.

This chapter focuses on our work in adapting continuous optimization techniques to the *shortest path* problem in the PRAM setting. From the discrete viewpoint, the max-flow and shortest path problems may seem widely different from each other. However, as we will see, they are closely related when viewed continuously, with one measuring the $\ell_\infty$-cost flow and the other the $\ell_1$-cost flow.

### 3.1 Shortest Path in Parallel

The single-source shortest path problem in the parallel PRAM model is one of the central problems in parallel computing. It is also one of the most notorious, since all fast, well-known algorithms for SSSP, such as Dijkstra's algorithm, are inherently sequential in nature. A more successful direction of research has focused on the $(1+\epsilon)$-approximate SSSP problem for undirected graphs, for which a breakthrough result of Cohen [Coh00] developed an algorithm based on the concept of *hopsets* that runs in $O(m^{1+\epsilon_0})$ work and polylogarithmic time for any constant $\epsilon_0 > 0$, which is optimal up to the $m^{\epsilon_0}$ factor. The natural follow-up question is whether the work can be improved to

$m \operatorname{polylog}(n)$, but this question has resisted two decades of attempts. Surprisingly, a recent work of Abboud et al. partially explains why: they present a lower-bound graph for which any purely hopset-based algorithm running in polylogarithmic time requires $\Omega(m^{1+\epsilon_0})$ work for some $\epsilon_0 > 0$.

In our paper [Li20], we tackle this problem from a continuous perspective, in particular the methods pioneered by Sherman [She13] for the max-flow problem. We reduce the SSSP problem to the more continuous *minimum transshipment* problem and provide the first $(1+\epsilon)$-approximate algorithm for both problems in $m \operatorname{polylog}(n)$ work and polylogarithmic time.

**Theorem 1.5** (Parallel SSSP)**.** *There exists a parallel algorithm that, given an undirected graph with nonnegative weights, computes a $(1+\epsilon)$-approximate single-source shortest path tree in $m \operatorname{polylog}(n)\, \epsilon^{-2}$ work and $\operatorname{polylog}(n)\, \epsilon^{-2}$ time in the PRAM model.*

**Theorem 3.1** (Parallel transshipment)**.** *There exists a parallel algorithm that, given an undirected graph with nonnegative weights and polynomial aspect ratio, computes a $(1+\epsilon)$-approximation to minimum transshipment in $m \operatorname{polylog}(n)\, \epsilon^{-2}$ work and $\operatorname{polylog}(n)\, \epsilon^{-2}$ time in the PRAM model.*

Sherman reduces the $(1+\epsilon)$-approximate transshipment problem to polylogarithmic calls to the $\ell_1$-*oblivious routing* problem, which is similar to standard oblivious routing for max-flow, except that we measure the $\ell_1$-cost of the flow instead of the $\ell_\infty$-cost.

**Definition 3.2** ($\ell_1$-oblivious routing)**.** *Given a weighted, undirected graph $G$, a matrix $R$ is a $\kappa$-approximate* $\ell_1$-oblivious routing matrix *for $G$ if for all demand vectors $b \in \mathbb{R}^n$, $\|Rb\|_1$ is a $\kappa$-approximation to the minimum transshipment cost for demands $b$.*

Here, the matrix $R$ essentially encodes an oblivious routing in matrix form. Sherman's algorithm is based on gradient descent, so a matrix representation is the most convenient.

**Theorem 3.3** (Sherman, paraphrased)**.** *Given a graph $G$, a $\kappa$-approximate $\ell_1$-oblivious routing matrix for $G$ with $M$ nonzero entries, and a demand vector $b$, we can compute a $(1+\epsilon)$-approximate minimum transshipment in $\kappa^2\epsilon^{-2}M \operatorname{polylog}(n)$ work and $\kappa^2\epsilon^{-2}\operatorname{polylog}(n)$ time.*

The beauty of Sherman's framework, and the benefit of considering the $\ell_1$-*oblivious routing* problem—which actually looks harder at first glance—is that only a *polylogarithmic* approximation to this problem is sufficient to obtain a $(1+\epsilon)$-approximation to the original transshipment problem. Moreover, the iterative procedure at the heart of the framework only requires polylogarithmic (sequential) steps, so as long as each step can be computed efficiently in parallel, so can the entire algorithm.

Our main technical contribution is a fast, parallel algorithm for the $\ell_1$-*oblivious routing* problem, at the expense of requiring an initial $\ell_1$-*embedding* of the graph.

**Theorem 3.4.** *Given a weighted, undirected graph $G$ and an $\ell_1$-embedding of $G$ with* $\operatorname{polylog}(n)$ *distortion in $O(\log n)$ dimensions, there is a parallel algorithm to compute a $(1+\epsilon)$-approximate minimum transshipment in $\tilde{O}(m\epsilon^{-2})$ work and* $\operatorname{polylog}(n)\epsilon^{-2}$ *time.*

Fast, sequential algorithms for $\ell_1$-embeddings typically require distance computations as subroutines, such as exact SSSP, and no work-efficient parallel algorithm for $\ell_1$-embeddings are known. Here, we aim for a recursive approach, making recursive calls to $(1+\epsilon)$-approximate SSSP on smaller graphs to compute the $\ell_1$-embedding. To ensure that the total size of the recursive instances goes down, we compute an *ultra-sparse* spanner of the original graph, and $\ell_1$-embed the spanner instead. This approach is inspired by Peng's recursive algorithm for max-flow, which uses ultra-sparsifiers instead of spanners [Pen16].

## 3.2 Future work

**Exact SSSP.** The biggest problem in this area is still the *exact* SSSP problem in parallel, which we hope to tackle next via exact transshipment and continuous optimization.

**Question 3.5.** *Is there a parallel algorithm for exact SSSP in $m^{1+o(1)}$ work and $n^{o(1)}$ time?*

Recent iterative methods for max-flow have successfully obtained algorithms for exact max-flow, and we believe many of the methods will translate over to exact transshipment as well. We even expect that these techniques will turn out simpler and more natural in the $\ell_1$-setting. For example, the SSSP problem itself admits fast, classic algorithms in the sequential model, in contrast to max-flow, for which no near-linear time algorithm is yet known in any model. Also, our $\ell_1$-oblivious routing algorithm is considerably simpler than the (standard) oblivious routing algorithms for fast, $(1+\epsilon)$-approximate max-flow, which is further evidence that the $\ell_1$-setting is cleaner than the $\ell_\infty$ one.

**Exact max-flow.** Ultimately, a simpler setting may allow us to understand these techniques better at an intuitive level, and in turn develop new insights to the original max-flow problem. Indeed, the recent advances in exact max-flow (on sparse graphs)—the current fastest of which runs in $m^{4/3+o(1)}$ time [LS20], and only on unweighted graphs—do not come close to the ultimate $m^{1+o(1)}$ time, suggesting that significantly new ideas may be necessary. Work on the transshipment problem has been sparse, even in the sequential setting, leaving many potentially fruitful ideas yet to be discovered.

**Question 3.6.** *Is there a (sequential) algorithm for exact max-flow in $m^{1+o(1)}$ time?*

**All-pairs shortest paths** One could generalize the SSSP problem to $S$-SSSP for some subset $S \subseteq V$, requiring us to solve the SSSP problem for each source in $S$. When $S = V$, this becomes the all-pairs shortest paths (APSP) problem, whose fastest running time is still the easy $O(n^3)$, unless the graph is unweighted and matrix multiplication is allowed, in which case the algorithm takes $O(n^\omega)$ time. Improving upon the general $O(n^3)$ time algorithm is a major open problem, and it has even been conjectured to be impossible. Nevertheless, continuous optimization techniques have not seen wide use in shortest path problems, and tackling the problem from this new perspective is always worth a shot.

Currently, it is not known how to adapt continuous optimization techniques for max-flow or SSSP to solve multiple instances (on the same graph) faster than it takes to solve each one separately. Here, what we need is an offline dynamic algorithm that takes sublinear time per instance. The intersection of continuous optimization and dynamic algorithms is still a wide open field, and there is plenty of room for new, exciting developments.

**Question 3.7.** *Is there a (sequential) algorithm for exact APSP in $O(n^{3-\epsilon})$ time for any constant $\epsilon > 0$?*

# 4 Related Work

In this section, we discuss our other related work.

### 4.1 Minimum $k$-cut.

In the (global) minimum $k$-cut problem, the goal is to remove the minimum weight set of edges to break the input graph into at least $k$ connected components. The famous contraction algorithm of Karger and Stein also works for min $k$-cut, and their original analysis produced to an algorithm running in time $\tilde{O}(n^{2k-2})$. As a corollary, they also show that there are at most $\binom{n}{2k-2}$ many min $k$-cuts in any graph (on $n$ vertices). In joint work with Anupam Gupta, David Harris, and Euiwoong Lee [GHLL20], we improved the running time to $\tilde{O}(n^k)$, and improved the extremal bound on the number of min $k$-cuts to $O_k(n^k)$, where $O_k$ hides factors depending only on $k$. The algorithm matches the lower bound of $n^{(1-o(1))k}$ from a reduction from max-weight $k$-clique, and the extremal bound is within constant factor of the lower bound $\binom{n}{k}$, which is achieved when the graph is an unweighted cycle.

**Theorem 4.1.** *Given a graph $G$ and parameters $k$ and $\alpha$, the Karger-Stein algorithm outputs any fixed $\alpha$-approximate $k$-cut in $G$ with probability at least $n^{-\alpha k}e^{-O(\alpha k^2 \log k)}$.*

**Corollary 4.2.** *For any graph (on $n$ vertices) and parameters $k$ and $\alpha$, there are at most $n^{\alpha k}e^{O(\alpha k^2 \log k)}$ many $\alpha$-approximate $k$-cuts.*

Our main insight is a *continuous* view of the Karger-Stein random contraction algorithm, viewing it as a continuous-time dynamical process and tracking its trajectory with a differential equation.

### 4.2 $k$-median in the FPT setting.

Given a metric space on clients and facilities, the $k$-median is a clustering problem in which one selects $k$ facilities to minimize the sum of distances from each client to its closest selected facility. This NP-hard problem has been extensively studied in the approximation algorithms setting, with a long line of work improving the approximation factor to the current best BLAH, which is still far from the $(1 + 2/e)$-approximation hardness of BLAH.

In joint work with Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, and Euiwoong Lee [CAGK+19], we investigate the problem from the fixed-parameter tractability (FPT) setting, where one is allowed $f(k) \cdot n^{O(1)}$ time for any function of $k$. Here, we present a $(1 + 2/e + \epsilon)$-approximation algorithm and show that it is tight (among even FPT algorithms).

**Theorem 4.3.** *Given a metric space on clients and facilities and any constant $\epsilon > 0$, there is a $(1 + 2/e + \epsilon)$-approximation algorithm for $k$-median that runs in $f(k) \cdot n^{O(1)}$ time, and under the Gap-ETH, there is no $(1 + 2/e - \epsilon)$-approximation algorithm that runs in $f(k) \cdot n^{O(1)}$ time.*

Our algorithm makes several new clustering insights for the $k$-median problem that may have further applications in the original, polynomial-time setting.

## References

[AKPS19] Deeksha Adil, Rasmus Kyng, Richard Peng, and Sushant Sachdeva. Iterative refinement for $\ell_p$-norm regression. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1405–1424. SIAM, 2019.

[BK15] András A Benczúr and David R Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM Journal on Computing*, 44(2):290–319, 2015.

[CAGK$^+$19] Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight fpt approximations for $k$-median and $k$-means. *arXiv preprint arXiv:1904.12334*, 2019.

[CGL$^+$20] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *Symp. Foundations of Computer Science (FOCS)*, 11 2020.

[Coh00] Edith Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *Journal of the ACM (JACM)*, 47(1):132–166, 2000.

[GHLL20] Anupam Gupta, David G Harris, Euiwoong Lee, and Jason Li. Optimal bounds for the $k$-cut problem. *arXiv preprint arXiv:2005.08301*, 2020.

[GR98] Andrew V Goldberg and Satish Rao. Beyond the flow decomposition barrier. *Journal of the ACM (JACM)*, 45(5):783–797, 1998.

[Kar00] David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000.

[KKOV07] Rohit Khandekar, Subhash Khot, Lorenzo Orecchia, and Nisheeth K Vishnoi. On a cut-matching game for the sparsest cut problem. *Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2007-177*, 2007.

[KT19] Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *J. ACM*, 66(1):4:1–4:50, 2019.

[Li20] Jason Li. Faster parallel algorithm for approximate shortest path. In *ACM Symposium on the Theory of Computing (STOC)*, 5 2020.

[LP20] Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *Symp. Foundations of Computer Science (FOCS)*, 11 2020.

[LS20] Yang P Liu and Aaron Sidford. Faster divergence maximization for faster maximum flow. *arXiv preprint arXiv:2003.08929*, 2020.

[NS17] Danupon Nanongkai and Thatchaphol Saranurak. Dynamic spanning forest with worst-case update time: adaptive, Las Vegas, and $O(n^{1/2-\epsilon})$-time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1122–1129, 2017.

[Pen16] Richard Peng. Approximate undirected maximum flows in $O(m\operatorname{poly}\log(n))$ time. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1862–1867, 2016.

[She13] Jonah Sherman. Nearly maximum flows in nearly linear time. In *FOCS*, pages 263–269. IEEE Computer Society, 2013.

[ST04] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC*, pages 81–90. ACM, 2004.