Cuts on Graphs: Applications and Variations

Josh Jones* Brandon Tran† Laryn Qi ‡

Abstract

In this paper we survey several recent results related to cuts on graphs. We begin with an introduct to cuts and general results an definitions, before covering Gomory-Hu (cut-equivalent) trees, expander graphs, and a result by Jason Li et al. We also introduce the Minimum Balanced Cut problem and recent progress made by Julia Chuzhoy et al. with the first deterministic, almost-linear time approximation algorithm for the problem. Finally, we explore sparse cuts in greater depth.

*joshuajones@berkeley.edu
†b.tran@berkeley.edu
‡larynqi@berkeley.edu

1 Introduction

Of considerable importance, both to modern algorithms and to real-world applications, is the idea of a cut on a graph.

Definition 1.1 (Cut)**.** A cut of a graph G with vertex set V is a disjoint partition (A, B) of the vertex set.

There are several types of cuts. For our purposes, the most important is an (s,t)-cut.

Definition 1.2. (Minimum (s,t)-cut problem). Given a weighted graph $G = (V, E, c)$, where $c : E \to \mathbb{R}$ is a capacity function, a *source* vertex $s \in V$, and a *sink* vertex $t \in V$, an (s,t)-cut is a cut (S,T) of V such that $s \in S$ and $t \in T$. Let $E(S,T) = \{e = (u,v) : u \in S, v \in T$ be the set of edges *crossing* the cut, and define the cost of the cut to be

$$c(S,T) = \sum_{e \in E(S,T)} c(e).$$

The cut cost may also be denoted $c(S)$ or $c(T)$. The minimum (s,t)-cut problem is then to find an (s,t)-cut with minimum cost.

A minimum cut, also known as a min cut or min-cut, between vertices s and t may be of practical importance. If the graph represents a transportation network, for example, then the minimum cut represents the minimum cost of disconnecting the network. One of the most interesting applications of the min cut problem is its dual problem, max flow, proved by Ford and Fulkerson in 1962 [FF62].

Definition 1.3 (Flow)**.** A flow f in a graph $G = (V, E, c)$ is a function $f : E \to \mathbb{R}_{\geq 0}$ that obeys two constraints:

i. (Capacity). $\forall e \in E$, $f(e) \leq c(e)$.

ii. (Conservation of flow). $\forall v \in V \setminus \{s,t\}$, $\sum_{e=(v,\cdot)} f(e) = \sum_{e=(\cdot,v)} f(e)$.

Then f is called *feasible*.

Theorem 1.4 (Max flow min cut theorem)**.** *The size of the maximum flow in a network equals the capacity of the smallest (s,t)-cut.*

This theorem can be viewed as a realization of the duality between the max flow and min cut problems. Max flow can be recognized as the following LP by simply writing down the definition of a flow:

$$\begin{aligned} \text{maximize} \quad & \sum_{e=(s,v) \in E} f(e) \\ \text{subject to} \quad & \sum_{e=(v,\cdot)} f(e) = \sum_{e=(\cdot,v)} f(e) \quad \forall v \in V \setminus \{s,t\} \\ & f(e) \leq c(e) \quad \forall e \in E \\ & f(e) \geq 0 \quad \forall e \in E \end{aligned}$$

And min cut can be similarly recognized as the following dual LP:

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} c(e) y_e \\ \text{subject to} \quad & \sum_{e \in p} y_e \geq 1 \quad \forall p \in P \\ & y_e \geq 0 \quad \forall e \in E \end{aligned}$$

where P is the set of of all possible simple paths from s to t. The constraints ensure that along every (s,t)-path, there exists some edge that "breaks" the connectivity and crosses the cut. Then, the objective function serves to minimize the cost of all the edges crossing the cut. With these two LPs, the max flow min cut theorem follows from the strong duality of linear programs.

We define here some important terminology and notation related to cuts.

Definition 1.5 (Volume)**.** Given a graph $G = (V, E)$, take a subset $S \subset V$. We define the volume of the cut to be

$$\text{vol}(S) = \sum_{v \in S} \deg v.$$

Definition 1.6 (Induced subgraph)**.** If $S \subset V$, then the subgraph of $G = (V, E)$ denoted by S, denoted $G[S]$, is the graph $G' = (S, E')$, where E' consists of the edges between vertices in S:

$$E' = \{(u, v) \in E : u, v \in S\}.$$

In many applications it is useful to have a measure of how dense a cut or graph is.

Definition 1.7 (Conductance)**.** The conductance of a cut $(S, V/S)$ of $G = (V, E)$ is defined as

$$\Phi_G(S) = \frac{c(S)}{\min\{\text{vol}(S), \text{vol}(V \setminus S)\}}.$$

The conductance of a graph is

$$\Phi(G) = \min_{S \subset V} \Phi(S).$$

If a cut has high conductance, then it has a high ratio of boundary edges to interior edges, i.e. its interior is not strongly connected compared to the rest of the graph. If a graph has high conductance, then there are no cuts with low conductance, i.e. no cuts that are extremely isolated from the rest of the graph.

Definition 1.8 (Sparsity)**.** The sparsity of a cut S of the graph G is defined as

$$\Psi_G(S) = \frac{c(S)}{\min\{|S|, |V \setminus S|\}}$$

Claim 1.9 (Conductance and Sparsity)**.** *The relationship between conductance and sparsity can be quantified via the maximum vertex degree, Δ, of the graph as follows*

$$\frac{\Psi_G(S)}{\Delta} \leq \Phi_G(S) \leq \Psi_G(S)$$

The proof follows from the fact that $\forall S \subset V$, $|S| \leq \text{vol}(S) \leq \Delta \cdot |S|$. Conductance and sparsity both measure how connected one side of a cut is to the other side, relative to the sizes of the sides.

Definition 1.10 (Expanders)**.** The expansion (sometimes also referred to as the Cheeger constant) of a graph $G = (V, E)$ on n vertices is defined as

$$\Psi(G) = \min_{\emptyset \neq S \subset V} \Psi_G(S)$$

G is a ψ-expander if $\Psi(G) \geq \psi$. We also say G is an expander if $\Psi(G) \geq \frac{1}{n^{o(1)}}$.

Intuitively, expander graphs have strong connectivity and yet are sparse. No two disjoint sets of vertices can be disconnected from each other without having to remove many edges, and yet there are not too many edges within each set.

2 Applications of Cuts

2.1 Gomory-Hu Trees

In many contexts it may be important to know the maximum flow or minimum cut between many or all pairs of vertices $u, v \in V$, rather than just between the pair (s, t). Clearly these max flows can be calculated with $\binom{n}{2} = O(n^2)$ max flow calculations, and then stored in a table of size $O(n^2)$ for lookup. It would therefore take $\tilde{O}(m^{1+o(1)}n^2)$ time to construct the table, which is too slow to be of practical use in many applications. Luckily, there is a much more efficient solution.

Definition 2.1 (Gomory-Hu tree)**.** Given a graph $G = (V, E, c)$, let $\lambda(s, t)$ be the value of the minimum cut between s and t. Then the Gomory-Hu tree for G is a tree $T = (V, E')$ with edge weights w such that for any $u, v \in V$ two properties are satisfied:

1. A minimum (u, v) cut in T is a min (u, v) cut in G, and
2. The value of this min (u, v) cut in T is the same as the value of the min (u, v) cut in G.

In particular, if e is an edge with minimum weight along the unique path in T connecting u and v, then $c(e) = \lambda(u, v)$, and furthermore the connected components of $T/\{e\}$ form the min cut (U, V).

This tree provides an efficient $O(n)$ space and $\tilde{O}(1)$ lookup method of storing the min cuts between all pairs of vertices. Furthermore, it is efficient to compute:

Theorem 2.2 (Gomory-Hu [GH61])**.** *For a graph $G = (V, E)$, a Gomory-Hu tree that satisfies the above properties always exists. Moreover, there exists an algorithm that computes the tree with only $|V|-1$ max-flow computations.*

This implies that if $r(m, n)$ is the runtime of a max-flow algorithm, then a Gomory-Hu tree can be constructed in $O\big(n \cdot r(m, n)\big)$ time. Chen et al's breakthrough max flow algorithm [CKL+22] therefore implies that exact GH trees can be constructed with high probability in $O(m^{1+o(1)}n)$ time for general weighted graphs; the GH tree of an unweighted graph can be constructed in $\tilde{O}(mn)$ time [HKPB07]. While the Gomory-Hu algorithm with advanced max-flow algorithms blackboxed remained the state-of-the-art for 60 years, a recent advance obtained independently by Zhang [Zha21] and Abboud et al [AKL+22] yields an $\tilde{O}(n^2)$ time algorithm for computing the GH tree. This algorithm is randomized and uses a fast reduction to the single-source min cut problem, rather than a direct application to Gomory-Hu's max-flow contraction algorithm. Since max-flow computations provably take $\Omega(m)$ time, the Gomory-Hu algorithm takes $\Omega(mn)$ time, and since $m = \Omega(n)$, this algorithm matches Gomory-Hu with optimal max-flow in all cases up to polylogarithmic factors, and supersedes it in graphs asymptotically denser than trees.

Definition 2.3 (Single-source min-cut)**.** Given a graph $G = (V, E)$ and a fixed vertex $s \in V$, the single-source min-cut problem (SSMC) is to compute the min (s, v)-cut for each $v \in V$.

When $m = \Theta(n^2)$, this algorithm takes $\tilde{O}(m)$ time. Any algorithm that constructs a Gomory-Hu tree must take at least $\Omega(m)$ time, so in this case Abboud's algorithm seems close to optimal. It remains to be seen whether a stronger lower bound can be proved, or whether a faster algorithm exists, especially in the case of sparser graphs.

If we relax our restriction that the GH tree must be exact, and instead only wish to compute an α-approximate GH tree, then we can reach even faster asymptotic complexities.

Definition 2.4 (Approximate Gomory-Hu tree)**.** Let $\lambda_G(u, v)$ represent the value of the min (u, v)-cut computed on graph G. Then for any parameter $\alpha \geq 1$, a tree $T = (V, E')$ is an α-approximate Gomory-Hu tree for the graph $G = (V, E)$ if for every pair of vertices $u, v \in V$,

$$\lambda_G(u, v) = \lambda_T(u, v)$$

and moreover the (u, v) cut (S, T) on the tree is an α-approximate min cut on G.

The approximate GH problem admits a significantly faster solution using the fair cut technique we introduce later.

Theorem 2.5 (Fast approximate GH tree [LNPS23])**.** *There exists a Monte Carlo $\tilde{O}(m \cdot poly(1/\epsilon))$ algorithm that, for any graph G and any $\epsilon > 0$ constructs a $(1+\epsilon)$-approximate GH tree.*

2.2 Expanders and Expander Decomposition

Expander graphs are highly useful in many fields of computer science and mathematics, and especially in pseudorandomness, cryptography, and hashing. To formalize this, we review some Markov chain terminology; refer to [LP17] for a more complete exposition.

Definition 2.6 (Markov chain)**.** We say that a stochastic process $(X_n)_{n\geq 0}$ is a Markov chain if for all $n \geq 0$,

$$P(X_{n+1} = x_{n+1} | X_n = x_n, \ldots, X_0 = x_0) = P(X_{n+1} = x_{n+1} | X_n = x_n).$$

The set of possible values that X_n can take on is the state space $\mathcal{X}$. Then

$$P^t(x, y) = P(X_t = x | X_0 = y)$$

is the t-step transition probability, and $P^t(x, \cdot)$ is the distribution of X_t on $\mathcal{X}$ given that $X_0 = x$.

Definition 2.7 (Stationary distribution)**.** For a Markov chain $(X_n)_{n\geq 0}$ with state space $\mathcal{X}$, we say that π is a stationary distribution if

$$\lim_{t\to\infty} P^t(x, \cdot) = \pi$$

for all $x \in \mathcal{X}$.

Definition 2.8 (Total variation norm)**.** The *total variation norm* of probability distributions μ and ν, defined on state space $\mathcal{X}$ and event space $\mathcal{F}$ is

$$\|\mu - \nu\|_{TV} = \max_{A\in F} |\mu(A) - \nu(A)| .$$

Definition 2.9 (Mixing time)**.** Given a Markov chain with stationary distribution π and starting distribution μ, the mixing time is defined as

$$t_{\text{mix}} = \inf\{t \geq 0 : \sup_{x\in\mathcal{X}} \|P^t(x, \cdot) - \pi\|_{TV} \leq 1/4\}.$$

Informally, the mixing time therefore represents the time until we are approximately at the stationary distribution. Treating a random walk on an expander as a Markov chain, the sparsity and connectedness of an expander graph imply that the mixing time is low; this was shown explicitly by Gillman [Gil93].

Therefore, after a relatively short random walk, no matter the initial starting distribution, we are randomly distributed throughout the graph. This has broad applications; a hash function, for example, can be defined by a graph, with the value to be hashed interpreted as instructions to walk through the graph. The hash is then the final vertex in the walk. If the graph is a good expander, then the walk need not be long to get good pseudorandom behavior, and sparsity implies that the graph is not too expensive to store, so this encryption scheme can be made efficient [CLG09].

More applicable to algorithmically solving cut problems is the expander decomposition. Intuitively, it is a decomposition of a graph into components such that each component is sparse, and there are not too many edges between partitions. The definition was introduced by Kannan et al [VKV00] to define good decompositions for clustering and sparsification.

Definition 2.10 (Expander decomposition)**.** For a graph $G = (V, E, c)$, we say that a partition $V_1, \ldots, V_k$ of V is an (ϵ, ϕ)-expander decomposition if

1. For each i,

$$\Phi(G[V_i]) \geq \phi.$$

2. We have

$$\sum_i c(V_i) \leq \epsilon \cdot \text{vol}(V_i).$$

The expander decomposition is useful in a wide variety of contexts, one of the most important of which is sparsification. If graph G' can be constructed from G such that G' is much sparser than G, and yet G' still has many of the same properties as G, then to solve many important problems on G we may first solve them on G' and then adjust the result. Moreover, since G' is sparser than G, this process can asymptotically improve runtimes.

Expander decompositions were used independently by [She13] and [KLOS14] in this manner to to achieve approximate max flow algorithms in almost and nearly linear time, and recently played a role in almost linear max flow algorithm discovered by Chen et al [CKL+22]. Similarly, this method can be efficiently used to dynamically maintain a spanning forest on a graph [NS17]. It critical in efficiently solving the bipartite matching problem, giving a runtime of $\tilde{O}(m + n^{1.5})$ [vdBLN+20]. Additional applications include solving linear and Laplacian equations [ST04, CKP+17].

Kannan et al proved that fixing ϕ and finding the decomposition that minimizes ϵ, or fixing ϵ and finding the decomposition that maximizes ϕ, is NP-hard, since finding the conductance of a graph is NP-hard, but gave an approximation algorithm running in time $\tilde{O}(n^3)$ [VKV00]. Significant progress led to $(\phi n^{o(1)}, \phi)$ expander decompositions in $m^{1+o(1)}$ time [NS17] and $(\tilde{O}(\phi), \phi)$ decompositions in $\tilde{O}(m\phi)$ time [SW18]. This result was improved upon by [LNPS23], producing a $(\tilde{O}(\phi), \phi)$-expander decomposition in $\tilde{O}(m)$ time.

It is unknown whether there are better algorithms for finding ϕ-expander decompositions; perhaps a fast $(O(\text{polylog}(\phi), \phi)$-expander decomposition algorithm exists. There are no proven lower bounds.

2.3 Fair Cuts

Li et al recently introduced the idea of a fair cut as a more restrictive approximate min cut that is not dominated by just a few edges with extreme capacities; instead, the corresponding flow must use all edges in the cut.

Definition 2.11 (Fair cut [LNPS23])**.** On an undirected graph $G = (V, E, c)$, with source $s \in V$ and sink $t \in V$, an α-fair (s, t)-cut is an (s, t)-cut (S, T) such that there exists an (s, t) flow f_0, called a *fair flow* with

$$f_0(e) \geq \frac{c(e)}{\alpha}$$

for all $e \in E(S, T)$.

Lemma 2.12. *An α-fair cut is also an α-approximate min cut.*

Proof. Let (S, T) be an α-fair (s, t) cut, and let the corresponding fair flow be f. Then

$$|f| = \sum_{e \in E(S,T)} f(e) \geq \sum_{e \in E(S,T)} \frac{c(e)}{\alpha} = \frac{c(S, T)}{\alpha}.$$

Since $|f^*| \geq |f|$, where f^* is the optimal (max) flow, this implies that

$$c(S, T) \leq \alpha\, |f^*|$$

so (S, T) is α-approximate. □

It turns out that there exists an efficient algorithm to compute α-fair cuts.

Theorem 2.13. *Let $G = (V, E)$ be an undirected, capacitated graphs, and let $s, t \in V$. Then for any $0 < \epsilon \leq 1$ there exists an algorithm that returns, with high probability, a $(1+\epsilon)$-fair (s, t) cut. Moreover, this algorithm runs in $\tilde{O}(m/\epsilon^2)$ time.*

The algorithm has several moving parts. It initializes an arbitrary (s, t) cut (S, T), and then refines it iteratively. At each step, on S and then on T, we call a subroutine AlmostFair that computes a partition of the input set into a "fair" component and an "unfair" component that the algorithm has trouble routing flow across fairly. Then, depending on the values of the corresponding cuts, these "unfair" components can be transferred across the cut. Eventually the algorithm terminates with a fair cut.

The function AlmostFair uses the MWU framework. It initializes weights for each set in a precomputed "representative" laminar collection of sets $\mathcal{S}$, defines a potential function based on these weights, and then defines flow values based on the potential function. Finally, "unfair" vertices are selected based on whether the flow can satisfy them and pruned out, and the weights are updated.

We have introduced already a number of applications of fair cuts. The crucial property that allows fair cuts to be used in applications where other approximate max flow/min cut algorithms lies within the uncrossing property:

Theorem 2.14 (Uncrossing property)**.** *For any (s, t) min cut (S, T) of a graph $G = (V, E)$, take $u, v \in S$. Then there exists $U \subset S$ such that $(U, V \setminus U)$ is a (u, v) mincut.*

That is, min cuts are naturally recursive. Unfortunately, this is not true of α-approximate min cuts; it is possible that, in the above setup, all α-approximate min cuts $(U, V \setminus U)$ have $U \not\subset S$. On the other hand, fair cuts do induce the uncrossing property: if (S, T) is an α-fair min cut, and $u, v \in S$, then there exists $U \subset S$ such that $(U, V \setminus U)$ is an α-fair min cut also. This property generalizes the application of fair cuts to algorithms that employ recursion on min cuts.

3 Balanced Cuts

In the traditional Minimum Balanced Cut problem, given a graph $G = (V, E)$, our goal is to find a cut of the graph (A, B) such that $\text{vol}(A)$, $\text{vol}(B) \geq \frac{\text{vol}(V)}{3}$ while minimizing the number of edges crossing the cut.

3.1 Graph Embeddings and Congestion

To explain the premise of Chuzhoy et al.'s algorithm, we first need a few additional definitions.

Definition 3.1 (Embedding)**.** Given two graphs $G = (V, E)$, $H = (V, E')$, an embedding of H into G is a collection of paths in G, $\mathcal{P} = \{P(e') | e' \in E'\}$, where every edge $e' \in E'$ is mapped to a path in G, $P(e')$.

From the definition, we can see it's only interesting to embed more connected graphs into sparser graphs.

Definition 3.2 (Congestion)**.** The congestion, η, of a given embedding $\mathcal{P}$ is the maximum number of paths that any single edge in E is a part of in $\mathcal{P}$.

We can formalize the relationship between the connectivity of H and the connectivity G using this congestion quantity, η.

Lemma 3.3 (Expansion ratios of embeddings [LR99])**.** *Given two graphs $G = (V, E)$, $H = (V, E')$ and an embedding of H into G, $\mathcal{P} = \{P(e') | e' \in E'\}$, with congestion $\eta \geq 1$, G is a $\frac{\psi}{\eta}$-expander if H is a ψ-expander.*

Proof. Let (A, B) be a partition of V. WLOG, say $|A| \leq |B|$. Let $c_H(A)$ be the cost of (# of edges crossing) the cut in H. Since H is a ψ-expander, $c_H(A) \geq \Psi_H(A) \cdot |A| \geq \psi \cdot |A|$ where the first inequality follows from the definition of sparsity and the second follows from the definition of expansion. Additionally, the path

associated with each edge crossing the cut in H, $P(e')$, must contain an edge crossing the same cut in G, e, since the paths by definition must connect the endpoints of e' in G. Since the embedding of H in to G causes congestion η, each edge in G participates in at most η paths. Thus, we have $c_H(A) = |\mathcal{P}| \leq \eta \cdot c_G(A)$. Finally, putting together the two inequalities, we have $\frac{\psi \cdot |A|}{\eta} \leq \frac{c_H(A)}{\eta} \leq c_G(A)$. $\Psi_G(A) := \frac{c_G(A)}{|A|} \geq \frac{\psi}{\eta} \implies$ G is a $\frac{\psi}{\eta}$-expander. □

3.2 The Cut-Matching Game

The state-of-the-art almost-linear time approximation algorithm for Minimum Balanced Cut by Chuzhoy et al. formulates the problem in terms of the *cut-matching game*. The game involves an intermediate graph H, the cut player who continually computes a small (sparse) balanced cut on H until H becomes an expander, and the matching player who tries to delay the construction of the expander by updating H with new edges according to a matching between vertices in an even partition of H. Specifically, in a given iteration of the game, the cut player first finds a balanced cut (A, B) such that $|A|, |B| \geq \frac{n}{4}$ with the cost of the cut being at most $\frac{n}{100}$. Then, the matching player computes their own partition (A', B') such that $|A'| = |B'|$ and $A \subseteq A'$. They make a perfect matching between A' and B' such that the matching can be embedded into G with low congestion, and the edges of the matching are added into H.

For the purposes of finding a minimum balanced cut in G, we try to embed an expander graph W into G. The game terminates once either

- the cut player can no longer find such a balanced sparse cut and can thus determine that W is a ψ-expander. In this case, by Lemma 3.1, G is also a ψ'-expander and must also not have a balanced sparse cut.
- or the matching player can fail to update H using a new, large matching that can be embedded into G with low congestion. In this case, there must be a balanced sparse cut in G which is returned [CGL+20].

It was shown in [KKOV07] that the game terminates in $\Theta(\log n)$ iterations.

Algorithms have been developed previously for both the cut and matching players. In a variation of the game, Khandekar et al. came up with a randomized algorithm that works by computing a max flow/min cut in $\tilde{O}(n^{\frac{3}{2}})$ [KRV06].

Chuzhoy et al. propose a deterministic recursive approach that computes a balanced sparse cut on W approximately by running multiple instances of the cut-matching games in parallel on smaller graphs. The main result of their paper is the following theorem which serves as the cut player's algorithm:

Theorem 3.4 (CutOrCertify [CGL+20]). *There is an universal constant N_0, and a deterministic algorithm, that we call* CutOrCertify*, that given an n-vertex graph $G = (V, E)$ with max vertex degree $O(\log n)$, and a parameter $r \geq 1$, such that $n^{\frac{1}{r}} \geq N_0$, returns one of the following:*

- *either a cut (A, B) in G with $|A|, |B| \geq \frac{n}{4}$ and $c_G(A) \leq \frac{n}{100}$; or*
- *a subset $S \subseteq V$ of at least $\frac{n}{2}$ vertices, such that $\Psi(G[S]) \geq \frac{1}{\log^{O(r)} n}$.*

The running time of the algorithm is $O(n^{1+O(\frac{1}{r})} \cdot (\log n)^{O(r^2)})$

The idea of this algorithm, CutOrCertify, is to split the vertices of G into k subsets, $V_1, V_2, \ldots, V_k$ of roughly equal size where $k = n^{o(1)}$. First, the algorithm tries to construct k expander graphs, W_i, on each V_i and embed them all into G simultaneously by running k cut-matching games in parallel. Each instance of the game starts with no edges and recursively calls CutOrCertify $O(\log n)$ times. Assuming the smaller instances all return a balanced sparse cut, we can hand all k cuts to the matching player to compute k matchings, M_i. If the matching player is able to embed all k matchings into G, we augment each

W_i according to each M_i. If the matching player fails and instead returns a balanced sparse cut, we do the same. After $O(\log n)$ iterations, if we still do not receive a balanced sparse cut, we know that each W_i are all expander graphs.

Next, we construct one final expander graph W^* on k vertices where each vertex of W^*, v_i, represents a set of the actual vertices in G, V_i. Again, we call CutOrCertify recursively on this smaller graph starting with the graph with no edges. Similarly to the previous step, we send the outputted partition from the recursive call to the matching player. If the matching player is able to compute a large matching and embed it with log congestion, we augment W^* and continue. If the matching player fails and instead returns a balanced sparse cut, we terminate and return the same. After $O(\log n)$ iterations, all W_i and W^* graphs will be expanders. We can compose all $k+1$ smaller graphs into a single $\hat{W}$ graph [CGL+20] that can be embedded into G with low congestion, certifying that G itself is an expander graph.

Finally, Chuzhoy et al. also adapt the traditional matching player algorithm of solving a single-commodity max flow problem to solving the multi-commodity flow problem (see Section 4.2 below) using Even-Shiloach trees [ES81] in order to accommodate the new cut player algorithm, but this is not their main result [CGL+20]. The new matching player algorithm must compute k different matchings between k pairs of subsets of vertices. Together, these algorithms for the cut and matching player yield an almost-linear time approximation algorithm for Minimum Balanced Cut.

3.3 Open Problems & Applications Related to Balanced Cuts

It it still unknown if Minimum Balanced Cut can be approximated with a polylogarithmic ratio in time $O(m^{1+o(1)})$. Deterministically approximating Minimum Balanced Cut with a $n^{o(1)}$ ratio in time $\tilde{O}(m)$ is also still an open problem.

One of the most immediate applications of approximating Minimum Balanced Cut deterministically is a deterministic approximation algorithm for the Sparsest Cut problem that runs in the same amount of time. However, the best known approximation algorithm for Sparsest Cut is still due to [ARV09], which the following section will cover in depth.

4 Sparse Cuts

The sparsest cut in a graph is the cut in the graph with the minimum sparsity. Using definition 1.8, we can write this as

$$\min_{S \subseteq V} \Psi_G(S).$$

The first method for approximating the sparsest cut in a graph was developed by Leighton and Rao in 1988. They formulated an LP relaxation of the problem, which when solved, achieved an $O(\log n)$ approximation of the sparsest cut [LR88]. Arora, Rao, and Vazirani improved this to a $O(\sqrt{\log n})$ approximation using an SDP relaxation [ARV09], which is currently the best known approximation. Arora, Hazan, and Kale later improved upon the SDP algorithm by utilizing expander flows which leads to the same approximation ratio of $O(\sqrt{\log n})$, but in $\tilde{O}(n^2)$ time.

4.1 SDP for $O(\sqrt{\log n})$ Approximation of Sparsest Cut

The $O(\sqrt{\log n})$ approximation algorithm proposed by Arora et al. depends on a geometric representation of the graph G.

Definition 4.1 (ℓ^2 representation)**.** An assignment of points to each node in a graph such that for all i, j, k we have

$$|v_i - v_j|^2 + |v_j - v_k|^2 \geq |v_i - v_k|^2$$

where v_i, v_j, v_k are the points assigned to nodes i, j, k, respectively.

Using this notion of an ℓ^2 representation, we can define an SDP as a relaxation of the sparsity of a graph:

$$\begin{aligned} \min \quad & \sum_{i,j \in E} |v_i - v_j|^2 \\ \text{s.t.} \quad & |v_i - v_j|^2 + |v_j - v_k|^2 \geq |v_i - v_k|^2 \qquad \forall i, j, k \\ & \sum_{i<j} |v_i - v_j|^2 = 1 \end{aligned}$$

To see that this is indeed a relaxation of sparsest cut, let $\bar{S} := V \setminus S$ and consider a cut $(S, \bar{S})$. Place all points in S on a single point of a sphere of radius $\frac{1}{\sqrt{2|S||\bar{S}|}}$. Place the points in $\bar{S}$ on the diametrically opposite point on the sphere. Then we can see that the value of the SDP will be $\frac{|E(S,\bar{S})|}{|S||\bar{S}|}$. Noting that $|\bar{S}|$ must be between $\frac{n}{2}$ and n, the optimal value of the SDP times $\frac{n}{2}$ must be a lower bound for the value of the sparsest cut problem. Next, we state an important theorem:

Theorem 4.2. *There is a polynomial-time algorithm that, given a feasible SDP solution with value β, produces a cut $(S, \bar{S})$ satisfying $|E(S, \bar{S})| = O(\beta |S| n \sqrt{\log n})$*

Combined with the fact that the value of the SDP is at least $O(n \frac{|E(S,\bar{S})|}{|S||\bar{S}|})$, this implies that the integrality gap is $O(\sqrt{\log n})$. The proof of the theorem involves two cases. Let us denote $d(v_i, A) := \min_{j \in A} |v_i - v_j|^2$ and $B(A, r) := \{i \in V | d(i, A) \leq r\}$. Then it can be shown that if $|A| \geq cn$ and $\sum_{i \in V} d(i, A) \geq \tau/n$, then $B(A, r)$ is a $O(1/c\tau)$ approximation for sparsest cut. In the first case, we assume that we have a v_i with radius $\frac{1}{8n^2}$ that contains at least $\frac{n}{4}$ vertices. We can easily verify that it suffices to choose $c = 1/4$ and $\tau = 7/8$ to satisfy the conditions. Since c and τ are constants, this means we in fact have a $O(1)$ approximation for sparsest cut in this special case.

Outside of this case, we can show that there exists $S, T \subseteq V$ such that $|S|, |T| = \Omega(n)$ and $d(S, T) \geq \Delta/n^2$ where $\Delta = O(1/\sqrt{\log n})$. The outline of the proof is to randomly select a hyperplane defining vector. Using a randomly selected hyperplane, $|S|, |T| = \Omega(n)$ with high probability, but the sets might not be sufficiently separated. Thus, we iterative remove pairs for which $d(s, t) < \Delta/n^2$. This process is analogous to removing a maximal matching from a bipartite graph, and it turns out, removing these vertices, we will not asymptotically decrease the size of S and T.

4.2 Maximum Multicommodity Flow

One application of sparsest cut is to the multicommodity flow problem. This is a generalization of our standard maximum (s, t) flow problem, but we now have multiple pairs of terminal vertices. The problem can be stated as follows: let $G = (V, E)$ be an undirected graph. Each edge $e \in E$ has capacity given by $c(e)$, and we have k pairs of terminal vertices (s_i, t_i) for $i \in \{1, \dots k\}$. We wish to assign a flow f to each path $P \in \mathcal{P}_i$ where $\mathcal{P}_i$, $i \in \{1, \dots k\}$ denotes all paths from s_i to t_i such that the total flow is maximized. The dual of this problem is the minimum multicut problem. In the case of one sink/source pair, strong duality holds, though this is not necessarily the case with more than one pair of commodities. It turns out, the minimum multicut problem is closely related to finding the sparsest cut in a graph, and Kahale showed that sparsest cut reduces to minimum multicut [Kah93].

4.3 Maximum Concurrent Flow

The maximum concurrent flow problem is a variation on the maximum multicommodity flow problem. It can be formulated as a linear program:

$$\begin{aligned} \max \quad & \gamma \\ \text{s.t.} \quad & f(P) \geq \gamma d(i) \qquad \forall i, \forall P \in \mathcal{P}_i \\ & f(P) \leq c(e) \qquad \forall e \in E \\ & f(P) \geq 0 \qquad \forall P \in \mathcal{P}_i \end{aligned}$$

Here, $d(i)$ is the flow demand for each source/sink, and we wish to maximize γ such that a fraction γ of each source/sink's demand is met. As presented in section 4.1, the sparsest cut problem assumes uniform demand, but can be modified to capture the more general case. Note that the result that we presented by Arora et al. only applies to the uniform case, and the more general case has an approximation factor of $O(\sqrt{\log n} \log \log n)$ which was due to Arora et al. [ALN05]. The dual of the maximum concurrent flow problem is the sparsest cut problem. Thus, an approximation for the sparsest cut of a graph yields an approximation for the maximum concurrent flow.

4.4 Open Problems Related to Sparse Cuts

There are many open problems associated with sparse cuts. One such problem is to achieve a polylogarithmic approximation in near linear time. Although Arora et al achieve this approximation in $\tilde{O}(n^2)$ time (compared to the SDP algorithm which takes $O(n^{4.5})$ time), we believe that there is still room for improvement. Another question that remains to be answered is whether or not it is possible to achieve a similar approximation ratio in the directed case. Finally, we are interested in whether there is a min-cut max-flow for multicommodity flows in the directed case with general edge costs and demands.

References

[AKL+22] A. Abboud, R. Krauthgamer, J. Li, D. Panigrahi, T. Saranurak, and O. Trabelsi. Breaking the cubic barrier for all-pairs max-flow: Gomory-hu tree in nearly quadratic time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 884–895, Los Alamitos, CA, USA, nov 2022. IEEE Computer Society.

[ALN05] Sanjeev Arora, James R. Lee, and Assaf Naor. Euclidean distortion and the sparsest cut, 2005.

[ARV09] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM (JACM)*, 56(2):1–37, 2009.

[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 Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1158–1167. IEEE, 2020.

[CKL+22] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623, 2022.

[CKP+17] Michael B. Cohen, Jonathan Kelner, John Peebles, Richard Peng, Anup B. Rao, Aaron Sidford, and Adrian Vladu. Almost-linear-time algorithms for markov chains and new spectral primitives for directed graphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, page 410–419, New York, NY, USA, 2017. Association for Computing Machinery.

[CLG09] Denis X. Charles, Kristin E. Lauter, and Eyal Z. Goren. Cryptographic hash functions from expander graphs. *Journal of Cryptology*, 22:93–113, 2009.

[ES81] Shimon Even and Yossi Shiloach. An on-line edge-deletion problem. *J. ACM*, 28:1–4, 1981.

[FF62] L. R. FORD and D. R. FULKERSON. *Flows in Networks*. Princeton University Press, 1962.

[GH61] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

[Gil93] D. Gillman. A chernoff bound for random walks on expander graphs. In *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, pages 680–691, 1993.

[HKPB07] Ramesh Hariharan, Telikepalli Kavitha, Debmalya Panigrahi, and Anand Bhalgat. An $\tilde{O}(mn)$ gomory-hu tree construction algorithm for unweighted graphs. pages 605–614, 06 2007.

[Kah93] Nabil Kahale. On reducing the cut ratio to the multicut problem. 01 1993.

[KKOV07] Rohit Khandekar, Subhash A. Khot, Lorenzo Orecchia, and Nisheeth K. Vishnoi. On a cut-matching game for the sparsest cut problem. Technical Report UCB/EECS-2007-177, EECS Department, University of California, Berkeley, Dec 2007.

[KLOS14] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, page 217–226, USA, 2014. Society for Industrial and Applied Mathematics.

[KRV06] Rohit Khandekar, Satish Rao, and Umesh V. Vazirani. Graph partitioning using single commodity flows. In *Symposium on the Theory of Computing*, 2006.

[LNPS23] Jason Li, Danupon Nanongkai, Debmalya Panigrahi, and Thatchaphol Saranurak. *Near-Linear Time Approximations for Cut Problems via Fair Cuts*, pages 240–275. 2023.

[LP17] David A. Levin and Yuval Peres. Markov chains and mixing times: Second edition. 2017.

[LR88] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, pages 422–431, 1988.

[LR99] Frank Thomson Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46:787–832, 1999.

[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, page 1122–1129, New York, NY, USA, 2017. Association for Computing Machinery.

[She13] Jonah Sherman. Nearly maximum flows in nearly linear time. *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, 04 2013.

[ST04] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, page 81–90, New York, NY, USA, 2004. Association for Computing Machinery.

[SW18] Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *ACM-SIAM Symposium on Discrete Algorithms*, 2018.

[vdBLN+20] Jan van den Brand, Yin-Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 919–930, 2020.

[VKV00] S. Vempala, R. Kannan, and A. Veta. On clusterings-good, bad and spectral. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, page 367, Los Alamitos, CA, USA, nov 2000. IEEE Computer Society.

[Zha21] Tianyi Zhang. Gomory-hu trees in quadratic time. *ArXiv*, abs/2112.01042, 2021.