COS 597B: Recent Advances in Graph Algorithms (Fall '24) November 18, 2024

Lecture 18: Deterministic Near-Linear Time Minimum Cut on Weighted Graphs

Presenters: *Frederick Qiu, Chenxiao Tian* Scribe notes by: *Akhil Jakatdar*

Preliminary

In this lecture, the presenters begin discussing the paper, Deterministic Near-Linear Time Minimum Cut on Weighted Graphs [2]. The presenters began by discussing the problem statement of the paper, followed by a brief summary of the history of this problem, followed by describing some important notation used in the rest of the lecture. Finally, they discussed the algorithm introduced in the paper and a brief introduction to the three different types of decomposition used in the algorithm. In these notes, I will describe all aspects of the paper covered in this lecture by the presenters.

Problem Statement. Let $G = (V, E, w)$ be an undirected, weighted graph with $|V| = n$ vertices and $E \subseteq V \times V, |E| = m$ edges, and a weight function $w : E \to \mathbb{R}$ for all edges $e \in E$. We can define the minimum cut $\lambda(G)$ of our graph as a partition (A, B) of V such that the following are true:

1. A, B form a non-empty bipartition of V. That is $A \cap B = \emptyset$, $A \cup B = V$ and $A, B \neq \emptyset$.
2. the total weight of edges crossing the cut (A, B) is minimized.
   That is, $\lambda(G) = \min_{(A,B)} \sum_{u \in A, v \in B, (u,v) \in E} w(u, v)$.

History Review. In order to better understand the history of finding the minimum cut of a weighted graph G in near-linear time, we must first discuss prior works on this problem. In 1996, David Karger [3] introduced a randomized algorithm to find the minimum cut of a graph in $O(m \log^3 n)$ time. He coins the term "near-linear" in describing the runtime of his algorithm has being linear in the size of the input multiplied by some polylogarithmic factor. Later work by Kawarabayashi and Thorup [4] introduce a near-linear time algorithm for simple graphs where they use a clustering procedure that maintains the true minimum cut of the graph. Work by Henzinger, Rao, and Wang [1] improved this algorithm to run in $O(m \log^2 n \log \log n)$ time. Recently, work by Li [5] introduces an "almost linear" $m^{1+o(1)}$ deterministic minimum cut algorithm for weighted graphs that "derandomizes" the random step introduces in Karger's algorithm by introducing an expander decomposition that does not perfectly preserve the true minimum cut.

Important Notation. The presenters then discussed some important notation used in the paper.

Weights. The first set of notations concerns the weights of the graph G.

1. For vertex sets $A, B \subseteq V$ such that $A \cap B = \emptyset$, we define the function $w_G(A, B) = \sum_{(u,v) \in E, u \in A, v \in B} w(u, v)$
2. For a vertex set $A \subseteq V$, we define the function $w_G(A) = \sum_{(u,v) \in E, u, v \in A} w(u, v)$

3. For a vertex $v \in V$, the function $d(v)$ returns the number of edges adjacent to v in G. Alternatively this represents the degree of v.
4. For a vertex $v \in V$, the function $d^W(v)$ returns the total edge weight of all edges adjacent to v in G.
5. For a vertex set $A \subseteq V$, $\mathbf{vol}(A) = \sum_{v \in A} d(v)$.
6. For a vertex set $A \subseteq V$, $\mathbf{vol}^W(A) = \sum_{v \in A} d^W(v)$.
7. We define W to be the sum of weights across all edges in G. That is $W = \sum_{e \in E} w(e)$.

Subgraphs. The next set of notations concerns how we define subgraphs on G.

1. A subgraph G induced on a set of vertices A will be denoted as $G[A]$.
2. A subgraph G induced on a set of vertices A such that $\forall u \in A$ s.t. $w_G(u, G \setminus A) > 0$, a self-loop is constructed with weight $w_G(u, G \setminus A)$ will be denoted as $G\{A\}$.

One important remark on the construction of $G\{A\}$ is that the added self-loops make $d^W(u)$ the same in the induced subgraph as in G for all vertices $u \in A$.

Components. The final set of notations define components of G.

1. A component C is a connected component of $G \setminus E'$ for some edgeset $E' \subseteq E$.
2. For a component C, the edgeset ∂C is the set of edges in G with exactly one endpoint in C. It is also called the boundary of component C.
3. An s-strong component C for a given $s, \tilde{\delta} \geq 0$ is a component such that for all cuts S with cut-size $w(S, G \setminus S) \leq 1.1\tilde{\delta}$, the following is satisfied:

$$\min(\mathbf{vol}^W_{G\{S\}}(S \cap C), \mathbf{vol}^W_{G\{S\}}(C \setminus S)) \leq s$$

From the definition of an s-strong component, we can note that the smaller the value of s is, the more closely connected the component C is and thus the harder it is to separate the partitions within C. Furthermore, this definition of an s-strong component provides us with the following useful fact.

Fact 2.2 Any component that is a subset of an s-strong component for some $s, \tilde{\delta} \geq 0$ is s-strong for the same $s, \tilde{\delta}$.

β-boundary sparse sets. The last important definition used in the paper is one for a β-boundary sparse set. For a component $A \subset V$ and a parameter $\beta \geq 1$, a set $U \subseteq A$ is β-boundary sparse iff

$$w(U, A \setminus U) < \beta \min(w(U, V \setminus A), w(A \setminus U, V \setminus A))$$

If U is β-boundary sparse set in A, then $A \setminus U$ is also β-boundary sparse and both $\emptyset$ and A are not β-boundary sparse in A. From the definition of β-boundary sparse sets, we can note that the smaller the β parameter is, the easier it is for U to be separated.

Main Result

The presenters then briefly outlined the algorithm presented in the paper that "derandomizes" Karger's algorithm using through a customized graph decomposition procedure that creates a minimum cut sparsifier called a skeleton graph H. The customized graph decomposition at a high level reduces the runtime from the expander decomposition used in [5] to be near-linear while maintaining the property that the resulting decomposition clusters the vertices such that there exists a cut of size $(1+\epsilon)\lambda(G)$ that does not cross between any of the clusters, where $\epsilon = \frac{1}{\text{polylog}\, n}$. The algorithm is shown below:

Near-Linear Time Minimum Cut Algorithm.

1. We first run Matula's $(2+\epsilon)$-approximation algorithm [6] (which can be easily extended to the weighted setting) to get a value λ_0 such that $\lambda \leq \lambda_0 \leq 3\lambda$.
2. For all powers $\tilde{\lambda}$ of 1.01 between $\lambda_0/3$ and $1.01\lambda_0$:
   (a) Initialize $j \leftarrow 0$ and $G_0 \leftarrow G$ with vertex set V_0 and edge set E_0.
   (b) While there are at least 2 nodes in $G_j = (V_j, E_j)$:
      i. Partition V_j into subsets called clusters that are s_0-strong using the algorithm of Lemma 5.1, for parameters $\tilde{\delta} = \tilde{\lambda}/1.01$ and $s_0 = 10^{11}\tilde{\delta}^2 r^2$. Let $C_j^{s_0}$ be the set of s_0-strong clusters.
      ii. Let $\epsilon = 1/\log^{1.1} n$.
      iii. For each resulting cluster C, we further decompose C according to the large cluster decomposition with values $\tilde{\lambda}$ and ϵ. Note that all clusters except possibly A_0 (if it exists) is a small cluster.
      iv. For each small cluster, we further decompose it according to small cluster decomposition with values $\tilde{\lambda}$ and ϵ.
      v. Let $C_j = \{C_{j,1}, C_{j,2}, \dots\}$ be the resulting set of clusters. Collapse every cluster $C \in C_j$ with $|C| > 1$ into one node and call the resulting graph $G_{j+1} = (V_{j+1}, E_{j+1})$.
      vi. $j \leftarrow j+1$.
   (c) Build a skeleton graph H using the clusters $\{C_j\}_j$ by using the algorithm from Section 7.
   (d) Run Karger's tree packing and dynamic programming algorithm on the skeleton graph to find a cut S.
3. Output the cut S with the minimum value found across all iterations of $\tilde{\lambda}$.

This algorithm runs a linear time 2-approximation algorithm to initialize a best-guess minimum cut value λ_0 such that $\lambda(G) \leq \lambda_0 \leq 3\lambda(G)$. We then iterate through the range $\frac{\lambda_0}{3} \leq \tilde{\lambda} \leq 1.01\lambda_0$ and perform three different types of graph decompositions. After the decomposition step, we construct our skeleton graph H from the resulting clusters and run Karger's tree packing and DP algorithm for each estimate $\tilde{\lambda}$ on H to get a cut S. Finally, we return the smallest S for all $\tilde{\lambda}$.

Graph Decompositions. The presenters lastly discussed the three different types of graph decompositions introduced in the paper. These are the main technical contributions of this paper and allow for the near-linear runtime guarantee that was not possible in [5].

s_0-strong partition. The presenter calls these $\tilde{\Theta}(\lambda)$-strong partitions. These partitions can be constructed in $\tilde{O}(m)$ time such that the intercluster total edge weight is less than $\frac{1}{\text{polylog } m} \cdot \mathbf{vol}^W(V)$. These partitions are referred to as clusters in the aforementioned algorithm. A complete definition of this decomposition is given below.

> **(s_0-strong partition)** Given a weighted graph $G = (V, E, w)$, a parameter $\tilde{\delta}$ such that $\tilde{\delta} \leq \min_{v \in V} d^W(v)$ and a parameter $s_0 = \Theta(\tilde{\delta} \log^c n)$ for some $c \geq 2$, there is an algorithm that runs in $\tilde{O}(m)$ time and partitions the vertex set V into s_0-strong components for $\tilde{\delta}$ such that the total weight of the edges between distinct components is at most
>
> $$O\left(\frac{\sqrt{\tilde{\delta} \log n}}{\sqrt{s_0}}\right) \cdot \text{vol}^W(V).$$

Large cluster decomposition. These s_0-strong clusters are then decomposed further into larger clusters and small clusters. Given some s_0-strong cluster, there exists an algorithm that runs in $\tilde{O}(E(G[A]))$ time that partitions A into $A_0, A_1, \ldots, A_k$, such that there are $O(\partial A \text{ polylog } |A|)$ intercluster edges. From this decomposition, there is at most one "large" cluster A_0 which is "uncrossed". That is, for every cut S of size at most $1.01\tilde{\lambda}$, S can be transformed to a cut S^* such that S^* does not cross A_0 and S^* is of size at most $(1+\epsilon)S$. For all remaining "small" clusters $A_1, \ldots, A_k$, we have $\mathbf{vol}^W(A_i) = \tilde{O}(\lambda)$. It is important to note that because this decomposition happens to all s_0-strong clusters, by Fact 2.2, these further decomposed clusters are also s_0-strong. A complete definition ofthe large cluster decomposition is given below.

> **(Large cluster decomposition)** Let $A \subseteq V$ be a cluster, let $0 < \epsilon \leq 0.1$ be a parameter, and let $\tilde{\lambda} \leq 1.01\lambda$ be an approximate lower bound to the minimum cut known to the algorithm. There is an algorithm that partitions A into a (potentially "large") set A_0 and "small" sets $A_1, \ldots, A_k$ such that:
>
> 1. For each $i \in \{1, 2, \ldots, r\}$, we have
>
> $$\text{vol}_G^W(A_i) \leq \left(\frac{10 s_0}{\epsilon \tilde{\lambda}}\right) \cdot s_0.$$
>
> 2. The total weight $w(A_0, A_1, \ldots, A_k)$ of inter-cluster edges is at most $O(\epsilon^{-1}) \cdot \partial A$.
>
> 3. For any set $S \subseteq A$ with $\partial_{G[A]} S \leq 1.01\tilde{\lambda}$ and $\text{vol}_G^W(S) \leq s_0$, there exists $S^* \subseteq S$ with $S^* \cap A_0 = \emptyset$ and
>
> $$w(S \setminus S^*, V \setminus A) + \partial_{G[A]} S^* \leq (1+\epsilon)\partial_{G[A]} S.$$

Small cluster decomposition. The remaining small partitions can be further decomposed. For a cluster A, there exists an algorithm that runs in poly$|A|$ time that partitions A into $A_1, \ldots, A_k$ such that the intercluster edges have weight $O(\partial A \text{ polylog } |A|)$ and for a set $S \subseteq A$ there is a grouping of $\{A_i\}$

into $P_1, \ldots, P_\ell$ such that there exists P_j such that $S \cap P_j$ is not $(1 - \frac{1}{\operatorname{polylog} m})$-boundary sparse in P_j. A complete definition of the small cluster decomposition is given below.

(Small cluster decomposition) Let $A \subseteq V$ be a cluster, let $0 < \epsilon \leq 0.01$ be a parameter, and let $\tilde{\lambda} \leq 1.01\lambda$ be an approximate lower bound to the minimum cut known to the algorithm. There is an algorithm that partitions A into a disjoint union of clusters $A_1 \cup A_2 \cup \cdots \cup A_k = A$ such that:

1. For any set $S \subseteq A$ of G with $\partial_{G[A]} S \leq 1.01\tilde{\lambda}$, there is a partition $\mathcal{P}$ of the set $\{A_1, A_2, \ldots, A_k\}$ such that for each part $\mathcal{P} = \{A_{i_1}, \ldots, A_{i_\ell}\} \in \mathcal{P}$, the set $S \cap \bigcup_{A' \in \mathcal{P}} A'$ is non-$(1-\epsilon)$-boundary-sparse in $\bigcup_{A' \in \mathcal{P}} A'$.
2. The sum of boundaries $\sum_i \partial A_i$ is at most

$$O(\epsilon^{-3}(\log |A|)^{O(1)} \partial A).$$

This concludes the notes on all aspects of the paper discussed in Lecture 18.

References

[1] Monika Henzinger, Satish Rao, and Di Wang. "Local flow partitioning for faster edge connectivity". In: *SIAM Journal on Computing* 49.1 (2020), pp. 1–36.

[2] Monika Henzinger et al. "Deterministic near-linear time minimum cut in weighted graphs". In: *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2024, pp. 3089–3139.

[3] David R Karger. "Minimum cuts in near-linear time". In: *Journal of the ACM (JACM)* 47.1 (2000), pp. 46–76.

[4] Ken-ichi Kawarabayashi and Mikkel Thorup. "Deterministic edge connectivity in near-linear time". In: *Journal of the ACM (JACM)* 66.1 (2018), pp. 1–50.

[5] Jason Li. "Deterministic mincut in almost-linear time". In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. 2021, pp. 384–395.

[6] David W Matula. "A linear time 2+ ε approximation algorithm for edge connectivity". In: *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*. 1993, pp. 500–504.