CS 594: Representations in Algorithm Design **Spring 2022**

Lecture 14: Feb. 24, 2022

Lecturer: Xiaorui Sun *Scribe: Wenyu Jin*

1 Last time: vertex sparsifier and c-edge connectivity

- Vertex sparsification: Given a graph G and a set of vertices T called the terminals, we aim to find a graph H that also contains vertex set T such that c-connectivity between any pair of terminals are the same on G and on H. This means for any $T' \subsetneq T$ such that $\text{mincut}_G(T', T \setminus T') \leq c$,

$$\text{mincut}_G(T', T \setminus T') = \text{mincut}_H(T', T \setminus T')$$

- We can construct a vertex sparsifier in the following way
  - Let E' be a c-cut containment set of G, meaning for all T' such that $\text{mincut}_G(T', T \setminus T') \leq c$, there exists a minimum cut using only edges from E'.
  - Remove E' from G.
  - Contract all resulting connected components.
  - Add E' back to the contracted graph.
- The cut containment set can be constructed from intersecting-all (IA) sets.

  Correction: In the last lecture, IA set was defined as a set of edges F such that for each terminal partition with minimum cut size $\leq c$, there is a cut that has an edge in F. This definition is wrong. The correct definition should be the following:

  Definition 1 (IA set)**.** *Given $G = (V, E)$, terminal set $T \subseteq V$ and threshold c, we say a set of edges $F \subseteq E$ is an IA(c) set if after removing F from G, for any bipartition $(T', T \setminus T')$ with mincut$_G(T', T \setminus T') \leq c$, there exists a minimum cut C separating T' and $T \setminus T'$ such that no connected component of $G \setminus F$ contains all edges of in the cut-set of C.*

2 How to find an IA set

Claim 2. *There always exists a an IA set of size $O(|T| \cdot c)$. Below is the algorithm to find such an IA set:*

- *Find an arbitrary minimum terminal cut of size $\leq c$*
- *Remove it*
- *On each connected components caused by removing the cut, go back to the first line and repeat until there is no terminal cut of size at most c or there is only one terminal left in a connected component.*
- *Return the union of these edges.*

Proof. By the description of the algorithm, if there is a c-cut separating terminals such that all edges are contained in one connected component after removing all edges, then the algorithm would find this cut and would not terminate. Therefore, when the algorithm terminates, the returned edge set is indeed an IA set.

The size of this edge set can be proven by induction on $|T|$.

When $|T| = 1$, the IA set contains 0 edges because there is no terminal cut.

When $|T| = 2$, the algorithm terminates after at most 1 iterations since after removing a cut separating the two terminals, each connected component contains at most 1 terminal.

Suppose the proposition holds for all $|T| \leq k$. When $|T| = k+1$, after removing an arbitrary cut of size at most c, let $CC_1, \ldots, CC_\ell$ be the resulting connected components ($\ell \geq 2$), and has $k_1, \ldots, k_\ell$ terminals each, then $k_1, \ldots, k_\ell \leq k$. Therefore, the number of edges in total is at most

$$c(k_1 - 1) + \cdots c(k_\ell - 1) + c \leq c(k - 1)$$ □

When the graph is a good expander, this can be done efficiently.

3 Application: Dynamic c- edge connectivity

Definition 3 (s-t c-edge connectivity)**.** *Two vertices u and v are c0edge connected if the minimum cut separating them has size at least c, or equivalently, there are c edge-disjoint paths connecting u and v.*

Work	c	Complexity	Amortized?
KKM13	1	polylog(n)	No
CGL+20	1	$n^{o(1)}$	No
Fre97	2	$O(\sqrt{m})$	No
EGIN97	2	$O(\sqrt{n}\log(m/n))$	No
HdLT01	2	polylog(n)	Yes
GI91	3	$O(m^{2/3})$	No
EGIN97	3	$O(n^{2/3})$	No

Updates: Edge insertions/deletions

Queries: For an arbitrary pair of vertices, decide whether they are c-edge connected or not.

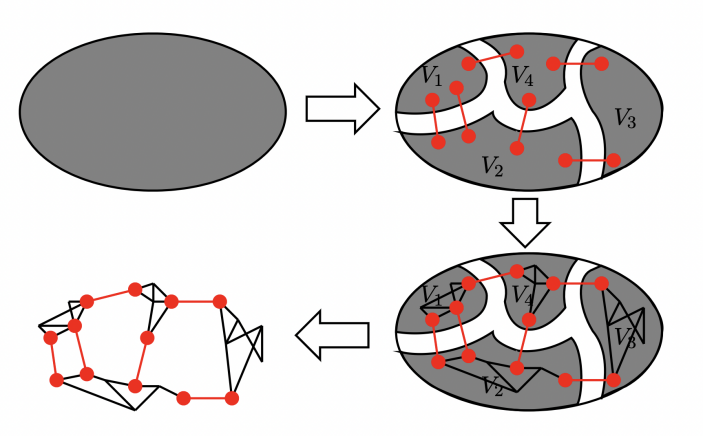
For c being small constants, there are the following results:

Theorem 4. *s-t c-edge connectivity can be maintained in deterministic $n^{o(1)}$ update and query time for $c = (\log n)^{o(1)}$.*

Problem: we can only make c-edge connectivity queries for vertices in the terminal set.

Solution: maintain a vertex sparsifier that supports adding terminals.

To initialize, given an arbitrary input graph G, we run expander decomposition on it. On each piece, we compute a vertex sparsifier (the boundary vertices of the expander decomposition are terminals). Finally, we put the boundary edges back with the smaller sparsifiers to obtain a sparsifier of G. See figure below. This process is repeated to obtain a multi-level sparsifier.



To update the sparsifier, we need to be able to update the cut containment set, we have the following result

Theorem 5. *Let G be an induced subgraph of G, a (c^2+2c)-cut containment set of G can be updated to a c-cut containment set of G'.*