Advanced Algorithms 13 November 2018

Lecture 9: Streaming and Sketching Algorithms III

Lecturer: Mohsen Ghaffari *Scribe: Davin Choo*

1 Graph sketching

Definition 1 (Streaming connected components problem)**.** *Consider a graph of n vertices and a stream S of edge updates $\{\langle e_t, \pm\rangle\}_{t\in\mathbb{N}^+}$, where edge e_t is either added (+) or removed (-). Assume that S is "well-behaved" where existing edges are not added and edge deletions can only occur after additions.*

At time t, the edge set E_t of the graph $G_t = (V, E_t)$ is the set of edges present after accounting for all stream updates up to time t. How much memory do we need if we want to be able to query the connected components for G_t for any $t \in \mathbb{N}^+$?

Let m be the total number of distinct edges that appear in the stream up to time t. There are two ways to represent connected components on a graph:

1. Every vertex stores a label where vertices in the same connected component has the same label
2. Explicitly build a tree for each connected component — This yields a *maximal forest*

For today, we are interested in building a maximal forest for G_t. This can be done with memory size of $\mathcal{O}(m)$ words[1], or — in the special case of *only edge additions* — $\mathcal{O}(n)$ words[2]. However, these are unsatisfactory as $m \in \mathcal{O}(n^2)$ on a complete graph, and we may have edge deletions. In this lecture, we will show that one can maintain a data structure with $\mathcal{O}(n \log^4 n)$ memory, with a randomized algorithm that succeeds in building the maximal forest with success probability $\geq 1 - \frac{1}{n^{10}}$.

Remark We say event X holds *with high probability* (w.h.p.) if $\Pr[X] \geq 1 - \frac{1}{n^c}$ for some constant $c \geq 2$.

1.1 Coordinator model

For a change in perspective[3], consider the following computation model where each vertex acts independently from each other. Then, upon request of connected components, each vertex sends some information to a centralized coordinator to perform computation and outputs the maximal forest.

This model will be helpful in our analysis of the algorithm later as each vertex will send $\mathcal{O}(\log^4 n)$ amount of data (a local sketch of the graph) to the coordinator, totalling $\mathcal{O}(n \log^4 n)$ memory as required.

1.2 Warm up: Finding the single cut

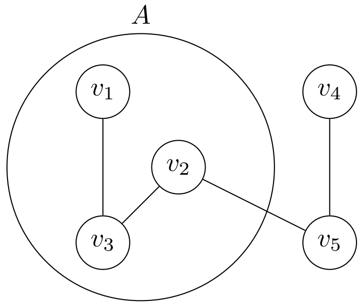
Definition 2 (The single cut problem)**.** *Fix an arbitrary subset $A \subseteq V$. Suppose there is exactly 1 cut edge (u, v) between A and $V \setminus A$. How do we output the cut edge (u, v) using $\mathcal{O}(\log n)$ bits of memory?*

Without loss of generality, assume $u \in A$ and $v \in V \setminus A$. Note that this is not a trivial problem on first glance since it already takes $\mathcal{O}(n)$ bits for any vertex to enumerate all adjacent edges. To solve the problem, we use a *bit trick* which exploits the fact that any edge $(a, b) \in A$ will be considered twice by vertices in A. Since one can uniquely identify each vertex with $\mathcal{O}(\log n)$ bits, consider the following:

- Identify an edge by the concatenation the identifiers of its endpoints (say, $u \circ v$ if $id(u) < id(v)$)
- Locally at every vertex u, maintain $XOR_u = \oplus\{id(e_i) : e_i \in S \wedge e_i$ has a endpoint u$\}$
- Vertices send the coordinator their sum and the coordinator computes $XOR_A \oplus \{XOR_u : u \in A\}$

[1] Toggle edge additions/deletion per update. Compute connected components on demand.
[2] Use the Union-Find data structure. See https://en.wikipedia.org/wiki/Disjoint-set_data_structure
[3] In reality, the algorithm simulates all the vertices' actions so it is not a real multi-party computation setup.

Example Suppose $V = \{v_1, v_2, v_3, v_4, v_5\}$ where $id(v_1) = 000$, $id(v_2) = 001$, $id(v_3) = 010$, $id(v_4) = 011$, and $id(v_5) = 100$. Then, $id((v_1, v_3)) = id(v_1) \circ id(v_3) = 000010$, and so on. Suppose $S = \{\langle (v_1, v_2), + \rangle, \langle (v_2, v_3), + \rangle, \langle (v_1, v_3), + \rangle, \langle (v_4, v_5), + \rangle, \langle (v_2, v_5), + \rangle, \langle (v_1, v_2), - \rangle\}$ and we query for the cut edge (v_2, v_5) with $A = \{v_1, v_2, v_3\}$ at $t = |S|$. The figure below shows the graph G_6 when $t = 6$:



Vertex v_1 sees $\{\langle (v_1, v_2), + \rangle$, $\langle (v_1, v_3), + \rangle$, and $\langle (v_1, v_2), - \rangle\}$. So,

$$\begin{array}{rcll} XOR_1 & \Rightarrow & 000000 & \text{Initialize} \\ & \Rightarrow & 000000 \oplus id((v_1, v_2)) = 000000 \oplus 000001 = 000001 & \text{Due to } \langle (v_1, v_2), + \rangle \\ & \Rightarrow & 000001 \oplus id((v_1, v_3)) = 000001 \oplus 000010 = 000011 & \text{Due to } \langle (v_1, v_3), + \rangle \\ & \Rightarrow & 000011 \oplus id((v_1, v_2)) = 000011 \oplus 000001 = 000010 & \text{Due to } \langle (v_1, v_2), - \rangle \end{array}$$

Repeating the simulation for all vertices,

$$\begin{array}{rcl} XOR_1 = 000010 & = & id((v_1), (v_2)) \oplus id((v_1), (v_3)) \oplus id((v_1), (v_2)) \\ & = & 000001 \oplus 000010 \oplus 000001 \\ XOR_2 = 000110 & = & id((v_1), (v_2)) \oplus id((v_2), (v_3)) \oplus id((v_2), (v_5)) \oplus id((v_1), (v_2)) \\ & = & 000001 \oplus 001010 \oplus 001100 \oplus 000001 \\ XOR_3 = 001000 & = & id((v_2), (v_3)) \oplus id((v_1), (v_3)) \\ & = & 001010 \oplus 000010 \\ XOR_4 = 011100 & = & id((v_4), (v_5)) \\ & = & 011100 \\ XOR_5 = 010000 & = & id((v_4), (v_5)) \oplus id((v_2), (v_5)) \\ & = & 011100 \oplus 001100 \end{array}$$

Thus, $XOR_A = XOR_1 \oplus XOR_2 \oplus XOR_3 = 000010 \oplus 000110 \oplus 001000 = 001100 = id((v_2, v_5))$ as expected. Notice that adding and deleting edges both add the edge ID to each vertex's XOR sum, and every edge in A contributes an even number of times to the coordinator's XOR sum.

Claim 3. $XOR_A = \oplus\{XOR_u : u \in A\}$ *is the identifier of the cut edge.*

Proof. For any edge (a, b) such that $a, b \in A$, $id((a, b))$ is in both XOR_a and XOR_b. So, $XOR_a \oplus XOR_b$ will cancel out the contribution of $id((a, b))$. Hence, the only remaining value in $XOR_A = \oplus\{XOR_u : u \in A\}$ will be the cut edge since only one endpoint lies in A. $\square$

Remark Bit tricks are often used in the random linear network coding literature (e.g. [HMK+06]).

1.3 Warm up 2: Finding one cut out of $k > 1$ cut edges

Definition 4 (The k cut problem)**.** *Fix an arbitrary subset $A \subseteq V$. Suppose there is exactly k cut edge (u, v) between A and $V \setminus A$, and we are given an estimate $\widehat{k}$ such that $\frac{\widehat{k}}{2} \leq k \leq \widehat{k}$. How do we output a cut edge (u, v) using $\mathcal{O}(\log n)$ bits of memory, with high probability?*

A straight-forward idea is to independently mark each edge, each with probability $1/\widehat{k}$. In expectation, we expect one edge to be marked. Denote the set of marked cut edges by E'.

$$\begin{array}{cll} & \Pr[|E'| = 1] & \\ = & k \cdot \Pr[\text{Cut edge } (u, v) \text{ is marked, and others are not}] & \\ = & k \cdot (1/\widehat{k})(1 - (1/\widehat{k}))^{k-1} & \text{Each edge is marked independently w.p. } 1/\widehat{k} \\ \geq & (\widehat{k}/2)(1/\widehat{k})(1 - (1/\widehat{k}))^{\widehat{k}} & \text{Since } \frac{\widehat{k}}{2} \leq k \leq \widehat{k} \\ \geq & \frac{1}{2} \cdot 4^{-1} & \text{Since } 1 - x \geq 4^{-x} \text{ for } x \leq 1/2 \\ \geq & \frac{1}{10} & \end{array}$$

Remark The above analysis assumes that the vertices can mark the edges locally so that it is consistent globally (i.e. both endpoints of any edge make the same decision whether to mark the edge or not). This can be done with a sufficiently large string of *shared randomness*. We discuss this in Section 1.4.

From above, we know that $\Pr[|E'| = 1] \geq 1/10$. If $|E'| = 1$, we can re-use the idea from Section 1.2. However, if $|E'| \neq 1$, then XOR_A may correspond erroneously to another edge in the graph. In the above example, $id((v_1, v_2)) \oplus id((v_2, v_4)) = 000001 \oplus 001011 = 001010 = id((v_2, v_3))$.

To fix this, we use random bits as edge IDs instead of simply concatenating vertex IDs. That is, randomly assign (in a consistent manner) each edge with a random ID of $k = 20 \log n$ bits. Since the XOR of random bits is random, for any edge e, $\Pr[XOR_A = id(e) \mid |E'| \neq 1] = (\frac{1}{2})^k = (\frac{1}{2})^{20 \log n}$. Hence,

$$\begin{aligned}
& \Pr[XOR_A = id(e) \text{ for some edge } e \mid |E'| \neq 1] & \\
\leq\ & \textstyle\sum_{e \in \binom{V}{2}} \Pr[XOR_A = id(e) \mid |E'| \neq 1] & \text{Union bound over all possible edges} \\
=\ & \binom{n}{2} (\frac{1}{2})^{20 \log n} & \text{There are } \binom{n}{2} \text{ possible edges} \\
=\ & 2^{-18 \log n} & \text{Since } \binom{n}{2} \leq n^2 = 2^{2 \log n} \\
=\ & \frac{1}{n^{18}} & \text{Rewriting}
\end{aligned}$$

Now, we can correctly distinguish $|E'| = 1$ from $|E'| \neq 1$ and $\Pr[|E'| = 1] \geq \frac{1}{10}$. For any given $\epsilon > 0$, there exists a constant $C(\epsilon)$ such that if we repeat $t = C(\epsilon) \log n$ times, the probability that *all* t tries fail to extract a single cut is $(1 - \frac{1}{10})^t \leq \frac{1}{n^{1+\epsilon}}$.

1.4 Maximal forest with $\mathcal{O}(n \log^4 n)$ memory in streaming edge updates

Recall that Borůvka's algorithm[4] builds a minimum spanning tree by iteratively finding the cheapest edge leaving connected components and adding them into the MST. The number of connected components decreases by at least half per iteration, so it converges in $\mathcal{O}(\log n)$ iterations.

For any arbitrary cut, the number of edge cuts is $k \in [0, n]$. Iterating through $\widehat{k} = 2^0, 2^1, \ldots, 2^{\lceil \log n \rceil}$, one can use Section 1.3 to find one such cut edge:

- If the guess $\widehat{k} \ll k$, the marking probability will select nothing (in expectation).
- If $\widehat{k} \gg k$, more than one edge will get marked, which we will then detect (and ignore) since XOR_A will likely not be a valid edge ID.

Algorithm 1 COMPUTESKETCHES($S = \{\langle e, \pm \rangle, \ldots\}, \epsilon, \mathcal{R}$)

```
for i = 1, ..., n do
    XOR_i ← 0^{(20 log n)*log^3 n}                               ▷ Initialize log^3 n copies
end for
for Edge update {⟨e = (u, v), ±⟩} ∈ S do          ▷ Edge updates arrive in streaming fashion
    for b = log n times do                                               ▷ Simulate Borůvka
        for i ∈ {1, 2, ..., log n} do                                   ▷ log n guesses of k̂
            for t = C(ε) log n times do                         ▷ Amplify success probability
                R_{b,i,t} ← Randomness for this specific instance based on R
                if Edge e is marked with probability k̂ = 2^i, according to R_{b,i,t} then
                    Compute id(e) using R
                    XOR_u[b, i, t] ← XOR_u[b, i, t] ⊕ id(e)
                    XOR_v[b, i, t] ← XOR_v[b, i, t] ⊕ id(e)
                end if
            end for
        end for
    end for
end for
return XOR_1, ..., XOR_n
```

Using a source of randomness $\mathcal{R}$, every vertex in COMPUTESKETCHES maintains $\mathcal{O}(\log^3 n)$ copies of edge XORs using random (but consistent) edge IDs and marking probabilities:

- $\lceil \log n \rceil$ times for Borůvka simulation later

[4]See https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm

Algorithm 2 STREAMINGMAXIMALFOREST($S = \{\langle e, \pm\rangle, \dots\}, \epsilon$)

```
R ← Generate O(log² n) bits of shared randomness          ▷ For edge IDs and marking probabilities
XOR_1, ..., XOR_n ← COMPUTESKETCHES(S, ε, R)
F ← (V_F = V, E_F = ∅)                                     ▷ Initialize empty forest
for b = log n times do                                     ▷ Simulate Borůvka
    C ← ∅                                                  ▷ Candidate edges
    for Every connected component A in F do
        for i ∈ {1, 2, ..., ⌈log n⌉} do                    ▷ Guess that A has [2^{i-1}, 2^i] cut edges
            for t = C(ε) log n times do                    ▷ Amplify success probability
                R_{b,i,t} ← Randomness for this specific instance
                XOR_A ← ⊕{XOR_u[b, i, t] : u ∈ A}
                if XOR_A = id(e) for some edge e = (u, v) then
                    C ← C ∪ {(u, v)}                       ▷ Add cut edge (u, v) to candidate edges
                    Go to next connected component in F
                end if
            end for
        end for
    end for
    E_F ← E_F ∪ C, removing cycles in O(1) if necessary     ▷ Add candidate edges
end for
return F
```

- $\lceil \log n \rceil$ times for guesses of cut size k
- $C(\epsilon) \cdot \log n$ times to amplify success probability of Section 1.3

Then, STREAMINGMAXIMALFOREST simulates Borůvka using the output of COMPUTESKETCHES:

- Find one out-going edge from every connected component A using the idea from Section 1.3
- Join connected components by adding edges to graph

Since each edge ID uses $\mathcal{O}(\log n)$ memory and $\mathcal{O}(\log^3 n)$ copies were maintained per vertex, a total of $\mathcal{O}(n \log^4 n)$ memory suffices. At each step, we fail to find one cut edge leaving a connected component with probability $\leq (1 - \frac{1}{10})^t$, which can be be made to be in $\mathcal{O}(\frac{1}{n^{10}})$. Applying union bound over all $\mathcal{O}(\log^3 n)$ computations of XOR_A, we see that $\Pr[\textit{Any } XOR_A \text{ corresponds wrongly some edge ID}] \leq \mathcal{O}(\frac{\log^3 n}{n^{18}}) \subseteq \mathcal{O}(\frac{1}{n^{10}})$. So, STREAMINGMAXIMALFOREST succeeds with high probability.

Remark One can drop the memory constraint per vertex from $\mathcal{O}(\log^4 n)$ to $\mathcal{O}(\log^3 n)$ by using a constant t instead of $t \in \mathcal{O}(\log n)$ such that the success probability is a constant larger than 1/2. Then, simulate Borůvka for $\lceil 2 \log n \rceil$ steps. See [AGM12] (Note that they use a slightly different sketch).

Theorem 5. *Any randomized distributed sketching protocol for computing spanning forest with success probability $\epsilon > 0$ must have expected average sketch size $\Omega(log^3 n)$, for any constant $\epsilon > 0$.*

Proof. See [NY18]. □

Claim 6. *Polynomial number of bits provide sufficient independence for the procedure described above.*

Remark One can generate polynomial number of bits of randomness with $\mathcal{O}(\log^2 n)$ bits. Interested readers can check out small-bias sample spaces[5]. The construction is out of the scope of the course, but this implies that the shared randomness $\mathcal{R}$ can be obtained within our memory constraints.

References

[AGM12] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 459–467. SIAM, 2012.

[5]See https://en.wikipedia.org/wiki/Small-bias_sample_space

[HMK+06] Tracey Ho, Muriel Médard, Ralf Koetter, David R Karger, Michelle Effros, Jun Shi, and Ben Leong. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, 52(10):4413–4430, 2006.

[NY18] Jelani Nelson and Huacheng Yu. Optimal lower bounds for distributed and streaming spanning forest computation. *arXiv preprint arXiv:1807.05135*, 2018.