Lecture 14

Finding Maximum Matchings and Counting Triangles

Scribe: Ranganath Kondapally

[The rest is unchecked; use at your own risk.]

14.1 The Problems

The number of graph problems that one could try to solve is huge and we will only be able to cover a small fraction. In the previous lecture, we considered some very basic graph problems that we could solve easily in streaming fashion. Now we turn to *maximum matching*, which is a more sophisticated problem. It is well-studied in the classical theory of algorithm design, and Edmonds's polynomial-time algorithm for the problem [**?**] remains one of the greatest achievements in the field. For graphs described by streams, we cannot afford computations of the type performed by Edmonds's algorithm. But it turns out that we can achieve low space (in the semi-streaming sense) if we settle for an approximation algorithm.

Before moving on to other topics, we shall consider the problem of counting (or estimating) the number of *triangles* in a graph. This number becomes meaningful if one thinks of the input graph as a social network, for instance. It turns out that this estimation problem is a rare example of a natural graph problem where a truly low-space (as opposed semi-streaming) solution is possible. Moreover, a *sketching* algorithm is possible, which means that we can solve the problem even in a turnstile model.

14.2 Maximum Cardinality Matching

A matching is defined to be a graph where every vertex has degree at most 1. Having fixed a vertex set, one can think of a matching as a set of edges no two of which share a vertex. The maximum cardinality matching (or simply maximum matching) problem, abbreviated as MCM, is to find a largest sized subgraph M of a given graph G such that M is a matching.

Problem: We are given a graph stream (stream of edges). We want to find a maximum matching.

There are two types of maximum matchings that we will consider:

- Maximum cardinality matching(MCM) : We want to maximize the number of edges in the matching.

DRAFT

- Maximum weight matching(MWM) : In this case, edges are assigned weights and we want to maximize the sum of weights of the edges in the matching.

"Semi-Streaming" model : aim for space $\tilde{O}(n)$ or thereabouts [we just want to do better than $O(n^2)$ space usage]

Both the algorithms for MCM and MWM have the following common characteristics:

- Approximation improves with more number of passes
- Maintain a matching (in memory) of the subgraph seen so far

Input: Stream of edges of a n-vertex graph

Without space restrictions: We can compute a MCM, starting with an empty set, and try to increase the size of matching : By finding an *augmenting path*.

Definition 14.2.1. Augmenting Path: Path whose edges are alternately in M and not in M, beginning and ending in unmatched vertices.

Theorem 14.2.2. *Structure theorem: If there is no augmenting path, then the matching is maximum.*

So, when we can no longer find an augmenting path, the matching we have is an MCM by the above theorem.

Streaming Algorithm for MCM: Maintain a *maximal* matching (cannot add any more edges without violating the matching property). In more detail:

- **Initialize:** $M \leftarrow \emptyset$
- **Process** (u, v)**:** If $M \cup \{(u, v)\}$ is a matching, $M \leftarrow M \cup \{(u, v)\}$.
- **Output:** $|M|$

Theorem 14.2.3. *Let $\hat{t}$ denote the output of the above algorithm. If t is the size of MCM of G, then $t/2 \leq \hat{t} \leq t$.*

Proof. Let M^* be a MCM and $|M^*| = t$. Suppose $|M| < t/2$. Each edge in M "kills" (prevents from being added to M) at most two edges in M^*. Therefore, there exists an unkilled edge in M^* that could have been added to M. So, M is not maximal which is a contradiction. □

State of the art algorithm for MCM: For any ε, can find a matching M such that $(1-\varepsilon)t \leq |M| \leq t$, using constant (depends on ε) number of passes.

Outline: Find a matching, as above, in the first pass. Passes $2, 3, \ldots$ find a "short" augmenting path (depending on ε) and increase the size of the matching. Generalized version of the above structure theorem for matchings gives us the quality guarantee.

MWM algorithm:

- **Initialize:** $M \leftarrow \emptyset$
- **Process** (u, v)**:** If $M \cup \{(u, v)\}$ is a matching, then $M \leftarrow M \cup \{(u, v)\}$. Else, let $C = \{$edges of M conflicting with $(u, v)\}$ (note $|C| = 1$ or 2). If $wt(u, v) > (1 + \alpha)\ wt(C)$, then $M \leftarrow (M - C) \cup \{(u, v)\}$.
- **Output:** $\hat{w} = wt(M)$

Space usage of the above algorithm is $O(n \log n)$.

Theorem 14.2.4. *Let M^* be a MWM. Then, $k \cdot wt(M^*) \leq \hat{w} \leq wt(M^*)$ where k is a constant.*

DRAFT

With respect to the above algorithm, we say that edges are "born" when we add them to M. They "die" when they are removed from M. They "survive" if they are present in the final M. Any edge that "dies" has a well defined "killer" (the edge whose inclusion resulted in its removal from M).

We can associate a ("Killing") tree with each survivor where survivor is the root, and the edge(s) that may have been "killed" by the survivor (at most 2), are its child nodes. The subtrees under the child nodes are defined recursively.

Note: The above trees, so defined, are node disjoint (no edge belongs to more than one tree) as every edge has a unique "killer", if any.

Let $S = \{\text{survivors}\}$, $T(S) = \bigcup_{e \in S}[\text{ edges in the Killing tree}(e) \text{ not including } e]$ (descendants of e)

Claim 1: $wt(T(S)) \leq wt(S)/\alpha$

Proof. Consider one tree, rooted at $e \in S$.

$$wt(\text{level}_i \text{ descendants}) \leq \frac{wt(\text{level}_{i-1} \text{ descendants})}{1+\alpha}$$

Because, $wt(\text{edge}) \geq (1+\alpha)wt(\{\text{edges killed by it}\})$

$$\begin{aligned}
\Rightarrow wt(\text{level}_i \text{ descendants}) &\leq \frac{wt(e)}{(1+\alpha)^i} \\
wt(\text{descendants}) &\leq wt(e)\Big(\frac{1}{1+\alpha} + \frac{1}{(1+\alpha)^2} + \ldots \infty\Big) \\
&= wt(e)\frac{1}{1+\alpha}\Big(\frac{1}{1-\frac{1}{1+\alpha}}\Big) \\
&= wt(e)\frac{1}{1+\alpha-1} \\
&= \frac{wt(e)}{\alpha}
\end{aligned}$$

$wt(T(S)) = \sum_{e \in S} wt(\text{descendants of } e) \leq \sum_{e \in S} \frac{wt(e)}{\alpha} = \frac{wt(S)}{\alpha}$

□

Claim 2: $wt(M^*) \leq (1+\alpha)(wt(T(S)) + 2 \cdot wt(S))$

Proof. Let $e_1^*, e_2^*, \ldots$ be the edges in M^* in the stream order. We will prove the claim by using the following charging scheme:

- If e_i^* is born, then charge $wt(e_i^*)$ to e_i^* which is in $T(S) \cup S$
- If e_i^* is not born, this is because of 1 or 2 conflicting edges
  - One conflicting edge, e: Note: $e \in S \cup T(S)$. Charge $wt(e_i^*)$ to e. Since e_i^* could not kill e, $wt(e_i^*) \leq (1+\alpha)wt(e)$
  - Two conflicting edges e_1, e_2: Note: $e_1, e_2 \in T(S) \cup S$. Charge $wt(e_i^*)\frac{wt(e_j)}{wt(e_1)+wt(e_2)}$ to e_j for $j = 1, 2$. Since e_i^* could not kill e_1, e_2, $wt(e_i^*) \leq (1+\alpha)(wt(e_1) + wt(e_2))$. As before, we maintain the property that weight charged to an edge $e \leq (1+\alpha)wt(e)$.
- If an edge is killed by e', transfer charge from e to e'. Note: $wt(e) \leq wt(e')$, so e' can indeed absorb this charge.

Edges in S may have to absorb $2(1+\alpha)$ times their weight (conflict two edges in M^*, etc). This proves the claim. □

By claim 1&2,

$$\begin{aligned} wt(M^*) &\leq (1+\alpha)\Big(\frac{wt(S)}{\alpha} + 2 \cdot wt(S)\Big) \\ &= (1+\alpha)\Big(\frac{1+2\alpha}{\alpha}\Big) wt(S) \\ &= \Big(\frac{1}{\alpha} + 3 + 2\alpha\Big) wt(S) \end{aligned}$$

Best choice for α (which minimizes the above expression) is $1/\sqrt{2}$. This gives

$$\frac{wt(M^*)}{3+2\sqrt{2}} \leq \hat{w} \leq wt(M^*)$$

Open question: Can we improve the above result (with a better constant)?

14.3 Triangle Counting:

Given a graph stream, we want to estimate the number of triangles in the graph. Some known results about this problem:

- We can't multiplicatively approximate the number of triangles in $o(n^2)$ space.
- We can approximate the number of triangles up to some *additive* error
- If we are given that the number of triangles $\geq t$, then we can multiplicatively approximate it.

Given a input stream of m edges of a graph on n vertices with at least t triangles, we can compute (ε, δ)-approximation using space $O(\frac{1}{\varepsilon^2} \log \frac{1}{\delta} \cdot \frac{mn}{t})$ as follows:

1. Pick an edge (u, v) uniformly at random from the stream
2. Pick a vertex w uniformly at random from $V \setminus \{u, v\}$
3. If (u, w) and (v, w) appear after (u, v) in the stream, then output $m(n-2)$ else output 0.

It can easily be shown that the expectation of the output of the above algorithm is equal to the number of triangles. As before we run copies of the above algorithm in parallel and take the average of their output to be the answer. Using Chebyshev's inequality, from the variance bound, we get the space usage to be $O(\frac{1}{\varepsilon^2} \log \frac{1}{\delta} \cdot \frac{mn}{t})$.

Bar-Yossef, Kumar, Sivakumar [BKS02] algorithm: Uses space $\tilde{O}(\frac{1}{\varepsilon^2} \log \frac{1}{\delta} \cdot (\frac{mn}{t})^2)$. Even though the space usage of this algorithm is more than the previous one, the advantage of this algorithm is it is a "sketching" algorithm (computes not exactly a linear transformation of the stream but we can compose "sketches" computed by the algorithm of any two streams and it can handle edge deletions).

Algorithm: Given actual stream of edges, pretend that we are seeing virtual stream of triples $\{u, v, w\}$ where $u, v, w \in V$. Specifically,

$$\begin{array}{ccc} \text{actual token} & & \text{Virtual token} \\ \{u, v\} & \rightarrow & \{u, v, w_1\}, \{u, v, w_2\}, \ldots, \{u, v, w_{n-2}\} \end{array}$$

where $\{w_1, w_2, \ldots, w_{n-2}\} = V \setminus \{u, v\}$.

Let F_k be the k^{th} frequency moment of virtual stream and

$$T_i = \Big|\Big\{\{u, v, w\} : u, v, w \text{ are distinct vertices and } \exists \text{ exactly } i \text{ edges amongst } u, v, w\Big\}\Big|$$

DRAFT

Note: $T_0 + T_1 + T_2 + T_3 = \binom{n}{3}$.

$$\begin{aligned} F_2 &= \sum_{u,v,w} (\text{number of occurrences of } \{u, v, w\} \text{ in the virtual stream})^2 \\ &= 1^2 \cdot T_1 + 2^2 \cdot T_2 + 3^2 \cdot T_3 \\ &= T_1 + 4T_2 + 9T_3 \end{aligned}$$

Similarly, $F_1 = T_1 + 2T_2 + 3T_3$ (Note: $F_1 =$ length of virtual stream $= m(n-2)$) and $F_0 = T_1 + T_2 + T_3$.

If we had estimates for F_0, F_1, F_2, we could compute T_3 by solving the above equations. So, we need to compute two sketches of the virtual stream, one to estimate F_0 and another to estimate F_2.

DRAFT