Algorithms for Big Data (VI) (Fall 2020)

Instructor: Chihao Zhang
Scribed by: Guoliang Qiu

Last modified on Nov 9, 2020

Today we continue to study algorithms for graph streams.

1 Shortest Path

In a unweighted graph, the shortest path between two vertices is the one containing minimum number of edges. Given an undirected simple graph $G = (V, E)$ in the streaming fashion, we want to answer the query "what is the minimum distance between u and v for $u, v \in V$".

Before introducing the algorithm, we clarify some notations. We let $d_G(x, y)$ be the minimum distance between u and v in the graph G and we call a subgraph $H = (V, E_H)$ of $G = (V, E)$ a t-spanner if

$$\forall x, y \in V : d_G(x, y) \le d_H(x, y) \le t \cdot d_G(x, y),$$

for some constant $t \ge 1$.

The idea of the algorithm is to maintain a t-spanner of G and output $\hat{d}_G(x, y) = d_H(x, y)$. We describe it in Algorithm 1.

Algorithm 1 Shortest Path Algorithm

Init:
$E_H \leftarrow \emptyset$.
On Input (u, v):
if $d_H(u, v) \ge t + 1$ **then**
    $E_H \leftarrow E_H \cup \{\{u, v\}\}$
end if
Output: On query (x, y)
Output $\hat{d}_G(x, y) = d_H(x, y)$.

Theorem 1. *The graph H maintained in Algorithm 1 is a t-spanner of G at any stage of the stream.*

Proof. Clearly, $d_H(x, y) \ge d_G(x, y)$ because H contains less edges. Consider the shortest path from x to y in G:

$$x = x_1, x_2, \cdots, x_k = y$$

and denote it as $P_{x,y}$. Then $d_G(x, y) = \sum_{i=1}^{k-1} d(x_i, x_{i+1})$. For every $\{u, v\} \in P_{x,y}$, if $\{u, v\} \in E_H$, then $d_H(u, v) = d_G(u, v)$; if $\{u, v\} \notin E_H$, then we know that when we are trying to insert $\{u, v\}$ into E_H, it must hold that $d_H(u, v) \le t$. Therefore, we have $d_H(x, y) \le t \cdot d_G(x, y)$. □

The algorithm costs $O(|E_H| \log n)$ bits of memory. However, we have not yet mentioned how large $|E_H|$ will be. To analyze the space consumption of Algorithm 1, we need to introduce some notions and facts in the graph theory.

The *girth* $g(G)$ of a graph G is the length of its shortest simple cycle. Clearly, $H = (V, E_H)$ is a graph with $g(H) \geq t+2$ because in order to add the new arrived edge $\{u, v\}$, $d_H(u, v)$ should be at least $t+1$. The reason for analyzing the girth of a graph is that in graphs with large girth, the number of edges is small. The fact is captured in the following theorem:

Theorem 2. *Let $G = (V, E)$ be a sufficiently large graph with $g(G) \geq k$. Let $n = |V|$ and $m = |E|$. Then*

$$m \leq n + n^{1+\frac{1}{\lfloor \frac{k-1}{2} \rfloor}}.$$

Proof. The main idea is as follows: If the girth of a graph is large, there exists a large induced tree in the graph. The number of edges in the induced tree is linear to its number of vertices. Therefore, the larger the induced tree, the less the number of total edges in the graph. We proceed to describe a deterministic way to find a large induced tree.

First we remove all vertices with small degrees. Let $d = \frac{2m}{n}$ be the average degree of G. If $d \leq 3$, we have $m \leq \frac{3}{2}n = O(n)$ and the theorem already holds. Otherwise, we repeatedly remove vertices with degree less than $\frac{d}{2}$. The number of edges removed is less than $\frac{d}{2} \cdot n \leq m$. This means that the remaining graph is nonempty. Moreover, its girth is still at least k and it contains a induced spanning tree of depth $\ell = \lfloor \frac{k-1}{2} \rfloor$ which can be found in a BFS manner. Since the degree of each vertex in the remaining graph is at least $\frac{d}{2}$, the number of vertices in the induced tree is larger than

$$1 + \left(\frac{d}{2} - 1\right) + \left(\frac{d}{2} - 1\right)^2 + \cdots + \left(\frac{d}{2} - 1\right)^\ell \geq \left(\frac{d}{2} - 1\right)^\ell = \left(\frac{m}{n} - 1\right)^\ell.$$

Therefore, $m \leq n + n^{1+\frac{1}{\ell}}$. □

According to Theorem 2 and the fact that $g(H) \geq t+2$, we know that $|E_H| \leq n^{1+\frac{2}{t}}$. It means that there is a tradeoff between the accuracy and the space memory in Algorithm 1.

2 Maximum Matching

Given a graph $G = (V, E)$ where each edge $e \in E$ has a weight $w(e) \in \mathbb{R}_{\geq 0}$. The Maximum Weighted Matching problem is to find a matching $M \subseteq E$ maximizing the weight $\sum_{e \in M} w(e)$. In this lecture, we design an algorithm to find an approximate maximum weighted matching in the streaming model.

We first consider the case when the graph is unweighted. A simple greedy algorithm described in Algorithm 2 works well in the case.

Let M be our estimate and M^* be the maximum matching.

Theorem 3. *M is a 2-approximation of M^*, that is,*

$$\frac{1}{2}|M^*| \leq |M| \leq |M^*|$$

Proof. It follows from Algorithm 2 that M is a maximal matching [1]. We now show that the size of every maximal matching M is at least $\frac{|M^*|}{2}$. Notice that each edge $e \in M$ can intersect with at most 2 edges in

[1] A matching M is a *maximal* matching of G if for any $e \in E(G)$, it holds that $M \cup \{e\}$ is no longer a matching.

Algorithm 2 Maximum Matching in Unweighted Graph

Init: $M \leftarrow \emptyset$
On Input $\{u, v\}$:
if $M \cup \{\{u, v\}\}$ is a matching **then**
    $M \cup \{\{u, v\}\}$
end if
Output:
Output M.

M^*, so if $|M| < \frac{|M^*|}{2}$, some edge $e' \in M^*$ does not intersect with any edge in M. We can then enlarge M by adding e', a contradiction to the fact that M is maximal. □

We need more effort to handle the weighted case. An algorithm is described in Algorithm 3 where α is a parameter to be determined later. The basic strategy of the algorithm is to include a new edge only if its weight is sufficiently large.

Algorithm 3 Maximum Weighted Matching

Init: $M \leftarrow \emptyset$
On Input $(\{u, v\}, w(u, v))$:
$C \leftarrow \{e \in M : \{u, v\} \cap e \neq \emptyset\}$
if $w(u, v) > (1 + \alpha)w(C)$ **then**
    $M \leftarrow M \setminus C \cup \{\{u, v\}\}$
end if
Output:
Output M.

We claim that the following fact holds for Algorithm 3.

Theorem 4. *For some constant $C_\alpha > 0$,*

$$\frac{w(M^*)}{C_\alpha} \leq w(M) \leq w(M^*)$$

To prove Theorem 4, we use a charging argument. First, we introduce the nomenclature for the *killing tree* with respect to Algorithm 3.

- An edge e is "born" when it is added to M.
- An edge e is "killed" by an edge f if it is removed while adding edge f to M.
- An edge e is "survived" if it is in M when the algorithm terminates.

The killing tree witnesses the "killing" relations between all born edges. It is a directed tree or forest which includes all "born" edges as its weighted vertices whose weights are their edge weights in the original graph. For two vertices e and f, there is a directed edge from f to e if the edge f kills the edge e. It follows from the definition that an edge e survives if and only if its corresponding vertex in the killing tree is a root. We further introduce the following notations:

- S := set of survivors.
- $T(e)$:= descendants of edge e in the killing tree.
- $T(S) := \sum_{e \in S} T(e)$.

Lemma 5. *The killing tree satisfies* $w(T(S)) \leq \frac{w(S)}{\alpha}$.

Proof. When an edge e is born, $w(e) \geq (1+\alpha)\sum_{f:e\to f} w(f)$. The same property holds for the weight of the edge f and its children. We can inductively show that

$$w(T(S)) \leq \sum_{e \in S} \sum_{i=1}^{\infty} \frac{w(e)}{(1+\alpha)^i} = \frac{w(s)}{\alpha}.$$

□

Lemma 5 gives an upper bound for $w(T(S))$ in terms of $w(S)$, i.e., the output of our algorithm. Now we relates $w(M^*)$ and $w(T(S))$.

Lemma 6.

$$w(M^*) \leq (1+\alpha)(w(T(S)) + 2w(S)).$$

Proof. The proof uses a neat charging argument. In the following discussion, a pair $\langle e, v\rangle$ consists of an edge e and a vertex v, and $E(\langle e, v\rangle)$ is the energy stored in the pair $\langle e, v\rangle$. During the charging process, we only charge energy for pairs $\langle e, x\rangle$ where $x \in e$ and maintain the following invariant properties:

- $\forall x \in V$, only one pair $\langle e, x\rangle$ including x is charged.
- $\forall \langle e, x\rangle$, it is charged at most $(1+\alpha)w(e)$.

We denote these two properties by $(\star)$.

It is ready to describe our charging procedure. We should notice that the energy only comes from the edges in the maximum weighted matching M^* and each edge e charges $w(e)$ amount of energy. While an edge $e = \{u, v\} \in M^*$ comes in:

1. If the edge e is born, charge (e, u) and (e, v) with $\frac{w(e)}{2}$ respectively.
2. If the edge e is not born because it is blocked by a single edge $f = \{v, w\}$, then we charge (f, v) with $w(e)$.
3. If the edge e is not born because it is blocked by two edges f_1, f_2, then we charge $(f_i, v_i)(i = 1, 2)$ with $\frac{w(f_i)}{w(f_i)+w(f_{3-i})} \cdot w(e)$ where v_i is the single vertex in $f_i \cap e$.

It is easy to check that the above three charge operations maintain the properties $(\star)$ because M^* is a maximum weighted matching.

Next, we describe the charging procedure for the edges $e = \{u, v\} \notin M^*$ as follows:

1. If e is not born, do nothing.
2. If e is born, absorb all energy of the edges it kills. For example, if an edge $f = \{u, u'\}$ is killed by e, we move the energy of (f, u) to (e, u).

Clearly ($\star$) is maintained by above operations as well.

We know that the charging procedure generates $w(M^*)$ energy and it is stored only in the vertices of the killing tree $T(S) \cup S$. We analyze their energy respectively. The followings are consequences of ($\star$).

- For $e = \{u, v\} \in T(S)$, its charge $\sum_{v_i \in \{u,v\}} E(\langle e, v_i \rangle) \leq (1+\alpha)w(e)$. This is because $E\langle e, v_i \rangle \leq (1+\alpha)w(e)$, for $v_i \in \{u, v\}$ and meanwhile, the edge has been killed so at most one of its end can store energy.
- For $e \in S$, its charge $\sum_{v_i \in \{u,v\}} E\langle e, v_i \rangle \leq 2(1+\alpha)w(e)$.

Therefore,

$$w(M^*) \leq (1+\alpha)(w(T(S)) + 2w(S)) \leq (1+\alpha)\left(\frac{w(S)}{\alpha} + 2w(S)\right) = \left(\frac{1}{\alpha} + 3 + 2\alpha\right) w(S).$$

We can then choose $\alpha = 1/\sqrt{2}$ to obtain a $(3+2\sqrt{2})$-approximation algorithm.

□

3 Counting Triangle

An important topic is to compute the number of some fixed subgraphs in a graph in the streaming setting. We will introduce a simple algorithm to count triangles.

We now define a vector $\boldsymbol{f} = (f_T)$ for every $T \in \binom{[n]}{3}$. For $T = \{x, y, z\}$, let $f_T = \left|\binom{T}{2} \cap E\right|$. Our goal is to compute $T_3 = |\{T | f_T = 3\}|$. We can similarly define T_0, T_1, T_2. The algorithm simply outputs $F_0 - 1.5F_1 + 0.5F_2$, where $F_i = \|\boldsymbol{f}\|_i^i$.

To see the correctness of the algorithm, note that we have

$$\begin{aligned}
F_0 &= T_1 + T_2 + T_3. \\
F_1 &= T_1 + 2T_2 + 3T_3. \\
F_2 &= T_1 + 4T_2 + 9T_3. \\
\binom{n}{3} &= T_0 + T_1 + T_2 + T_3.
\end{aligned}$$

Solving the equations, we obtain

$$T_3 = F_0 - 1.5F_1 + 0.5F_2.$$

So the problem of counting triangles can be reduced to frequency estimation of a data stream, which we already know how to do.

The analysis of the performance has been left as an exercise.