Streaming Algorithms

Meng-Tsung Tsai

04/13/2018

Reminder

Written Assignment #1 is due by tonight. You need to LaTex your solution and submit it on New E3 (https://e3new.nctu.edu.tw).

You are encouraged to discuss with your classmates, TA, or me. However, the writeup shall be your own.

I will place the solutions online after the submission deadline. If you need more time to work on it, please let me know.

Midterm Exam

There is no class next week, as we agreed at the beginning of the semester. No office hour next week either.

We have a take-home midterm exam (the 2nd written assignment), whose weight is 20%. It will be annonced by Apr 15, and due on Apr 24.

You are encouraged to discuss with your classmates and TA. However, the writeup shall be your own.

References

- "Analyzing graph structure via linear measurements," Ahn, Guha, McGregor (2012)
- "A linear-time algorithm for finding a sparse *k*-connected subgraph of a *k*-connected graphs," Nagamochi and Ibaraki (1992)
- "Sparsification - a technique for speeding up dynamic graph algorithms," Eppstein, Galil, Italiano, and Nissenzweig (1997)
- "Vertex and Hyperedge Connectivity in Dynamic Graph Streams," Guha, McGregor, Tench (2015)

Spanning Trees

Problem Definition

Input: a sequence of edges $e_i = \{u_i, v_i\}$ where $u_i, v_i \in [n]$ and $i \in [m]$, i.e. a representation of an n-node m-edge undirected graph G.

Output: a spanning forest of G, where a spanning forest consists of a spanning tree in each connected component of G.

Goal: use O(n polylog n) space. // called semi-streaming model

Note that the space usage for graph problems can be as large as $\Omega(n^2)$. In the semi-streaming model (graph streaming), we require the space usage bounded by O(n polylog n).

Algorithm (first attempt)

```
F ← ∅;

foreach incoming edge e_i {
    if (F ∪ {e} is acyclic){
        F ← F ∪ {e_i};
    }else{
        discard e_i; // i.e. we don't keep discarded edges in memory
    }
}

output F;
```

This algorithm uses O(n) space and **$O(m\ \alpha(n))$** time.

Algorithm

```
F ← ∅; B ← ∅;

foreach incoming edge e_i {
    if (|B| < n){
        B ← B ∪ {e_i};
    }
    if(|B| equals n or EOF){
        F ← spanning-forest (F ∪ B);
    }
}

output F;
```

This algorithm uses O(n) space and **O(m)** time.

Sampling-based algorithm

It is possible to sample a "random" spanning forest from a given graph G in the semi-streaming model. Because the sampling appeals to ℓ_0-samplers, we defer the details to the lecture of ℓ_p-samplers. [ref. 1]

The sampling algorithm is nontrivial. Think about that the input graph has an **bridge**.

Sampling-based algorithm

It is possible to sample a "random" spanning forest from a given graph G in the semi-streaming model. Because the sampling appeals to ℓ_0-samplers, we defer the details to the lecture of ℓ_p-samplers. [ref. 1]

Why do we need a **complicated** sampling-based algorithm? We already has a simple determinstic algorithm.

Sampling-based algorithm

It is possible to sample a "random" spanning forest from a given graph G in the semi-streaming model. Because the sampling appeals to ℓ_0-samplers, we defer the details to the lecture of ℓ_p-samplers. [ref. 1]

The main reason is that the deterministic algorithm cannot handle edge deletions, but the sampling-based algorithm can.

Applications

Minimum Spanning Trees (MST)

Input: a sequence of edges $e_i = \{u_i, v_i\}$ and the edge weight $\omega(e_i)$ where $u_i, v_i \in [n]$, $i \in [m]$, and $\omega(e_i) \in R$, i.e. a representation of an n-node m-edge undirected edge-weighted graph G.

Output: a minimum spanning forest of G, where a minimum spanning forest consists of a MST in each connected component of G. // a spanning tree T whose $\sum_{e \in T} \omega(e)$ is smallest among all spanning trees is called the MST

Goal: use O(n polylog n) space.

Algorithm

$F \leftarrow \emptyset$;

foreach incoming edge e_i {
    if ($F \cup \{e\}$ is acyclic){
        $F \leftarrow F \cup \{e_i\}$;
    }else{
        discard e_k whose $\omega(e_k)$ is largest among all edges on the cycle;
        // tie-breaking by edge index
    }
}

output F;

This algorithm uses O(n) space and **O(m n)** time (naively).

Correctness

Cycle Property: for any cycle C, if an edge e has the largest weight on C (tie-breaking by edge index), then e cannot be an edge in MST.

In the algorithm, we only discard edges that are impossible to be included in MST. The correctness clearly follows.

Exercise 1

Devise an algorithm to reduce the running time.

k-EC

Input: an integer k, a sequence of edges $e_i = \{u_i, v_i\}$ where $u_i, v_i \in [n]$, $i \in [m]$, i.e. a representation of an n-node m-edge undirected graph G.

Output: "Yes" if one can disconnect the graph by a removal of k edges. // For k = 1, it is equivalent to determining whether a graph has a bridge.

Goal: use O(kn polylog n) space.

Graph Property

Let $F_0 = \emptyset$; F_i = spanning-forest($G \setminus F_0 \setminus ... \setminus F_{i-1}$);

Theorem 1. G is k-EC if and only if $F_1 \cup F_2 \cup ... \cup F_k$ is k-EC. [ref. 2]

Algorithm

$F_1 \leftarrow \emptyset$; $F_2 \leftarrow \emptyset$; ...; $F_k \leftarrow \emptyset$;

```
foreach incoming edge e_i {
    for(j = 1; j ≤ k; ++j){
        if (F_j ∪ {e_i} is acyclic){
            F_j ← F_j ∪ {e_i};
            break;
        }
    }
}
```

output k-EC($F_1 \cup F_2 \cup ... \cup F_k$);

This algorithm uses O(kn) space and $O(k(m+n)+T_{kEC})$ time where T_{kEC} is the best time complexity to determine k-EC, on a RAM.

Sampling-based algorithm

Sample a spanning forest F_1 from G; // i.e. apply the sampling algorithm on the input edge set

Sample a spanning forest F_2 from $G \setminus F_1$; // i.e. apply the sampling algorithm on the discarded edges in the first round

...

Sample a spanning forest F_k from $G \setminus F_1 \setminus F_2 \setminus ... \setminus F_{k-1}$; // i.e. apply the sampling algorithm on the discarded edges in the (k-1)-th round

Again, why do need the sampling-based algorithm?

Exercise 2

Determining whether an n-node m-edge graph has a bridge can be done in $O(n+m)$ time by DFS.

Devise an $O(n^2+m)$-time algorithm determining whether an n-node m-edge graph G is 2-EC.

k-VC

Input: an integer k, a sequence of edges $e_i = \{u_i, v_i\}$ where $u_i, v_i \in [n]$, $i \in [m]$, i.e. a representation of an n-node m-edge undirected graph G.

Output: "Yes" if one can disconnect the graph by a removal of k nodes. // For $k = 1$, it is equivalent to determining whether a graph has an articulation point.

Goal: use $O(kn \text{ polylog } n)$ space.

Graph Property

Let $F_0 = \varnothing$; $F_i = \text{BFS-forest}(G \setminus F_0 \setminus \ldots \setminus F_{i-1})$;

Theorem 2. G is k-VC if and only if $F_1 \cup F_2 \cup \ldots \cup F_k$ is k-VC. [ref. 2]

Algorithm [ref. 3]

```
F0 ← ∅; F1 ← ∅; F2 ← ∅; ...; Fk ← ∅; B ← ∅;

foreach incoming edge ei {
    if(|B| < n){
        B ← B ∪ {ei};
    }
    if(|B| equals n or EOF){
        B ← B ∪ F1 ∪ F2 ∪ ... ∪ Fk;
        for(j = 1; j ≤ k; ++j){
            Fj ← BFS-forest(B \ F0 \ ... \Fj-1);
        }
    }
}

output k-VC(F1 ∪ F2 ∪ ... ∪ Fk);
```

This algorithm uses $O(kn)$ space and $O(k(m+n)+T_{kVC})$ time where T_{kVC} is the best time complexity to determine k-VC, on a RAM.

Sampling-based algorithm [ref. 4]

input: an n-node m-edge graph G

for($i = 1$; $i \leq R$; ++i){ // $R = 16\, k^2 \ln(n)$

Define (not compute) G_i be a random induced subgraph of G so that each node in G is included in G_i with probability $1/k$.

$T_i \leftarrow$ arbitrary spanning tree of G_i; // sampling a spanning tree of G_i

}

Define $H = T_1 \cup T_2 \cup \ldots \cup T_R$;

return k-VC(H);

H is k-VC iff G is k-VC w.h.p.

Correctness

Because H is a subgraph of G, we get H is k-VC $\Rightarrow$ G is k-VC. It remains to show that G is k-VC $\Rightarrow$ H is k-VC w.h.p.

First of all, H has the same node set as G w.h.p.

$$\Pr[\text{node } v \text{ is not in } H] = (1-1/k)^R \leq e^{-16 \ln(n)} = 1/n^{16}.$$

$$\Pr[\text{all nodes are in } H] \geq 1-1/n^{15}.$$

Let S be an arbitary node subset of size k. Our strategy is to show that $G \setminus S$ is connected $\Rightarrow$ $H \setminus S$ is connected w.h.p.

Correctness

Consider an arbitrary pair of nodes $s, t \notin S$, there is a path $s = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_p = t$. (Why?) Suppose that for each i, if v_i, v_{i+1} are in G_j and $G_j \cap S = \varnothing$, then there is a path from v_i to v_{i+1} in G_j. (Why?)

$$\Pr[G_j \text{ has } v_i \text{ and } v_{i+1} \text{ and } G_j \cap S = \varnothing] = (1/k)^2(1-1/k)^k$$

$\Pr[\text{no path from } v_i \text{ to } v_{i+1} \text{ in } H \setminus S]$
$= \Pr[\text{no } G_j \text{ has } v_i \text{ and } v_{i+1} \text{ and } G_j \cap S = \varnothing] = (1-(1/k)^2(1-1/k)^k)^R \leq 1/n^4$

By union bound on the p edges on the path $s \rightsquigarrow t$, we get

$$\Pr[s \text{ and } t \text{ is disconnected in } H \setminus S] \leq p/n^4 \leq 1/n^3.$$

By union bound on all possible t's, we get

$$\Pr[s \text{ and all other } t \text{ in } H \setminus S \text{ are connected}] \geq 1-1/n^2.$$

Space Usage

Each G_i has $O(n/k)$ nodes w.h.p. Each of them needs $O(n/k \text{ polylog } n)$ space to sample a spanning tree. Therefore, the space usage is

$$O(R\, n/k \text{ polylog } n) = O(kn \text{ polylog } n).$$