6.889 Sublinear Time Algorithms February 19, 2019

Lecture 4

Lecturer: Ronitt Rubinfeld *Scribe: Allison Tam*

1 Overview

- Distributed algorithms overview
- Parnas-Ron Framework
- Vertex cover approximations (sublinear and distributed)

Today, we covered how to use distributed local algorithms to inspire sublinear time sequential algorithms. The overall idea is to use the insights and understandings leveraged in distributed algorithms. Historically, this has led to a blending of ideas in the two fields. In lecture, we focus on a sublinear approximation algorithm of a vertex cover.

2 Distributed Algorithms

2.1 Overview

The input to a distributed algorithms problem is a network, consisisting of processors (nodes) connected by links (edges). Unlike in parallel algorithms, computation is performed by the communication network **on** the communication network. In other words, the input graph is the network itself. For the purpose of this class, we assume the following properties of the network.

1. max degree d
2. each processor knows its neighbors
3. synchronous computation – processors reliably send messages to each other during synchronized rounds of communication

Each round operates in the following fashion.

1. Nodes perform computation on their input bits, random bits, and history of communication.
2. Nodes send messages to neighbors. Messages don't necessarily have to be the same.
3. Nodes receive messages.

2.2 Connection to Sublinear Time

In the above setup, we can simulate the output of the distributed algorithm. Suppose we are interested in node v's state. After one round, v's output is based on its information and its immediate neighbors'. After two rounds, v implicitly learns information from its neighbors' neighbors. Using induction, after k rounds of an algorithm, node v knows and uses information from nodes at most k links away.

This implies that you can know v's output using a sequential simulation that uses $O(d^k)$ queries. The simulation only requires us to look at nodes within a k-radius ball of v, because information from elsewhere in the graph won't otherwise play a role in v's output. There are lots of useful k-round distributed algorithms for k constant or $\log \log n$.

In sublinear time algorithms, we often sample and test for some property. We can abstract the fast distributed algorithm as an oracle that simply tells us if a node has some property.

3 Vertex Cover

Given $G = (V, E)$, $V' \subseteq V$ is a **vertex cover** if $\forall(u, v) \in E$, either u or $v \in V'$. From 6.046, we know that finding the minimum vertex cover size is NP-hard. We also know that we can find a 2-approximation of vertex cover size in polynomial time.

Assume that we have a bounded degree d graph, where $d \geq 2$ and is a constant. Without this, the problem becomes much harder.

Let $|V'|$ be the size of the vertex cover of G. For a connected, bounded graph, we can conclude that $|V'| \geq \frac{m}{d} \geq \frac{n}{d}$. This can be shown through contradiction. Each vertex in V' can cover at most d edges, which in total must exceed m.

3.1 Distributed Subroutine

Now, we go over a distributed algorithm that returns a valid vertex cover.

Algorithm 1: Distributed Algorithm for Vertex Cover

```
Input  : Graph $G = (V, E)$
Output: Approximated vertex cover $A$
1 $i \leftarrow 0$ ;
2 while edges remain in $E$ do
3     Let $U$ be the set of nodes in $V$ with degree $\geq \frac{d}{2^i}$ ;
4     $A \leftarrow A \cup U$ ;
5     Remove $U$ from $V$ and incident edges from $E$ ;
6     Update degrees of remaining nodes in graph ;
7     $i \leftarrow i + 1$ ;
8 Return $A$ ;
```

Example Walkthrough

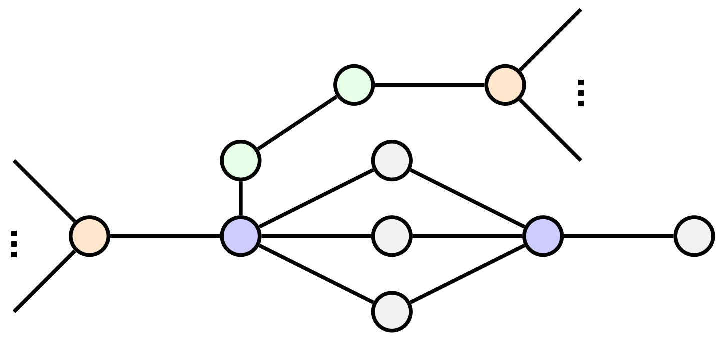

Figure 1: Consider a graph with bounded degree 8. Orange nodes have degree 8.

In round 0, the orange nodes are added to A, because they have degree ≥ 8. In round 1, the blue nodes, with degree 4, are added to A. In round 2, no modifications are made. In round 3, the green nodes, now only connected to one another, are both added to A.

After the algorithm terminates at $\log_2 d = 4$ rounds, each node will know if it is in A.

Claim 1 *A is a valid vertex cover.*

Proof At the end of algorithm 1, no edges remain. All removed edges must have at least one node added to A. ■

Theorem 2 *Let θ be any vertex cover of graph G. Algorithm 1 will output a valid vertex cover A, such that $|\theta| \leq |A| \leq (1 + 2\log d)|\theta|$.*

Proof Each round in algorithm 1 adds at most $2|\theta|$ nodes that are not in θ to A. Let X be the set of nodes that are removed during round i and are not in θ. Observe that for all nodes in X, every one of their neighbors must be in θ, since θ is a vertex cover and X has no overlap with θ.

Let $|E_X|$ be the edges incident to X. On the ith iteration, all newly removed nodes have degree $\geq \frac{d}{2^i}$, so $|E_X| \geq |X|\frac{d}{2^i}$. Furthermore, because previous iterations removed high degree nodes, *all* remaining nodes have degree $\leq \frac{d}{2^{i-1}}$. That combined with the above observation (all E_X end in θ) implies that $|E_X|$ has an upper bound of $|\theta|\frac{d}{2^{i-1}}$ when all edges are connected to vertices not yet removed.

$$|X|\frac{d}{2^i} \leq |E_X| \leq |\theta|\frac{d}{2^{i-1}}$$

$$|X| \leq 2|\theta|$$

If each round adds at most $2|\theta|$ nodes not in θ, then at most $2\log d$ additional nodes can be added to A, and the theorem follows directly. ■

The distributed algorithm achieves an $O(\log d)$ approximation in simulated $O(d^{\log d})$ queries.

3.2 Using Parnas-Ron Framework

The **Parnas-Ron framework** samples nodes, simulates the distributed algorithm to check if a node is in the vertex cover, and extrapolates and estimate from the results. We pick r based on desired ϵ. Using Chernoff, we require $r = O(\frac{1}{\epsilon^2})$.

Algorithm 2: Sublinear Vertex Cover (Parnas-Ron Framework)

```
  Input  : Graph $G$, $\epsilon$
  Output: $\hat{v}$
1 $i \leftarrow 0$ ;
2 $r \leftarrow \frac{1}{\epsilon^2}$ ;
3 Sample $r$ vertices $v_1, \cdots, v_r$ from $G$ uniformly with replacement ;
4 for $k = 1$ to $r$ do
5     Simulate algorithm 1 to see if $v_k$ is in the vertex cover;
6     if $v_k$ in vertex cover then
7         $i \leftarrow i + 1$ ;
8 Return $\hat{v} = (\frac{i}{r})n$ ;
```

Overall runtime is $O(rd^{\log d}) = O(\frac{1}{\epsilon^2}d^{\log d})$, which doesn't depend on n. Additive error comes from using sampling. Multiplicative error comes from the fact that the subroutine is an approximation.

Definition 3 *$\hat{y}$ is a (α, ϵ)-approximation of true value y for a minimization problem if $y \leq \hat{y} \leq \alpha y + \epsilon n$.*

Algorithm 2 gives an $(O(\log d, \epsilon n))$-approximation in $d^{O(\log d)}$ queries. As food for thought, another algorithm uses the Parnas-Ron framework to achieve a $(2, \epsilon n)$-approximation in $d^{O(\frac{1}{\epsilon}\log d)}$ queries.