6.889 Sublinear Time Algorithms September 10, 2020

Lecture 4

Lecturer: Ronitt Rubinfeld *Scribe: Henry Hu*

1 Overview

Today, we covered the following topics:

- Vertex Cover
- Distributed model (Local model)
- Parnas-Ron Framework for reducing sublinear to distributed approximations
- Sublinear algorithm for Approximate Vertex Cover (Parnas-Ron)

2 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.

2.1 Example cases

1. Graph with no edges
   - The size of the minimum vertex cover is 0.
2. Graph with one edge
   - The size of the minimum vertex cover is 1 (by taking either of the endpoints).
3. Star: $|V| - 1$ vertices, each of degree 1, connected to a central node.
   - The size of the minimum vertex cover is 1 (by taking the central node).
4. k-clique
   - The size of the minimum vertex cover is $k - 1$ (by taking any less vertices we would miss an edge between the remaining vertices).
5. n-cycle
   - The size of the minimum vertex cover is $\lceil \frac{n}{2} \rceil$ (by taking every other vertex).
6. Graph with nodes of degree $\leq \Delta$, where $\Delta \geq 2$ is a constant
   - The size of the minimum vertex cover is $\geq \frac{|E|}{\Delta} \geq \frac{|V|}{\Delta}$. This can be shown through contradiction. Each vertex in the vertex cover can cover at most Δ edges, which in total must exceed $|E|$.

2.2 Multiplicative approximation

In example case 2.1.1, the multiplicative approximation must return 0. But in example case 2.1.2, the multiplicative approximation must *not* return 0. Therefore, to return a multiplicative approximation requires $\Omega(n)$ queries (to distinguish between the two cases), and no sublinear multiplicative approximation is immediately possible.

2.3 Additive approximation

There are theoretical results stating that it is computationally hard to approximate a vertex cover to a factor of better than 1.36 (and maybe even 2). Because an additive approximation is even harder than a multiplicative approximation at large input sizes, no sublinear additive approximation is immediately possible.

2.4 Combined Multiplicative and Additive approximation

Definition 1 *$\hat{y}$ is a (α, ϵ)-approximation of true value y for a minimization problem if $y \leq \hat{y} \leq \alpha y + \epsilon$.*

By relaxing the multiplicative definition, we avoid the problem of distinguishing 0 edges and 1 edge. In section 4, we prove an $(O(\log \Delta), \epsilon n)$-approximation for the minimum vertex cover problem.

3 Local Model for Distributed Algorithms

3.1 Overview

The input to a distributed algorithms problem is a network, consisisting of processors (nodes) connected by links (edges). In the "local model", computations run in communication rounds which operate in the following fashion.

- Nodes perform computation on their input bits, random bits, received messages, and history of received messages.
- Nodes send messages to neighbors. Messages don't necessarily have to be the same.
- Nodes receive messages from neighbors.

A distributed local algorithm describes computations to perform and messages to send for a node in each round. For the purposes of this class, we also assume the following properties of the network.

- max degree Δ
- each processor knows its neighbors
- synchronous computation – processors reliably send messages to each other during synchronized rounds of communication

3.2 Vertex Cover in the local distributed model

Unlike in parallel algorithms, computation is performed by the communication network **on** the communication network. In other words, the input graph for the problem is the network itself, where nodes represent processors and edges represent links.

In the local model, the input to the vertex cover problem is the network graph, and the output for each node is whether it is "in" or "out" of the vertex cover.

3.3 Connection from fast distributed algorithms to sublinear-time sequential algorithms

We can simulate the output of any distributed algorithm using a sequential 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 $\leq \Delta^k$ 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 many incredible k-round distributed algorithms for k constant or $\log \log n$.

To simulate the output for vertex v after a k-round distributed algorithm, we run k iterations. In each iteration $i \in \{0, 1, ...k-1\}$, we sequentially simulate the computation, messages sent, and messages received for each of the nodes in the Δ^{k-i} ball surrounding vertex v. Any information calculated at a node $\geq i$ away from v after iteration i will never reach v, so does not need to be simulated. Finally, we simulate vertex v.

This idea is formalized in the **Parnas-Ron Framework** for reducing distributed to sequential algorithms, which is applied to the Vertex Cover problem in section 4.2.

4 Sublinear-time algorithm for Vertex Cover

4.1 Distributed Subroutine

Now, we go over a distributed algorithm that returns a valid vertex cover as described in section 2.

Algorithm 1: Distributed Algorithm for Vertex Cover

Input : Graph $G = (V, E)$
Output: Approximated vertex cover A

$i \leftarrow 0$;
$A \leftarrow \emptyset$;
while *edges remain in* E **do**
  Let U be the set of nodes in V with degree $\geq \frac{\Delta}{2^i}$;
  $A \leftarrow A \cup U$;
  Remove U from V and incident edges from E ;
  Update degrees of remaining nodes in graph ;
  $i \leftarrow i + 1$;
Return A ;

Example Walkthrough

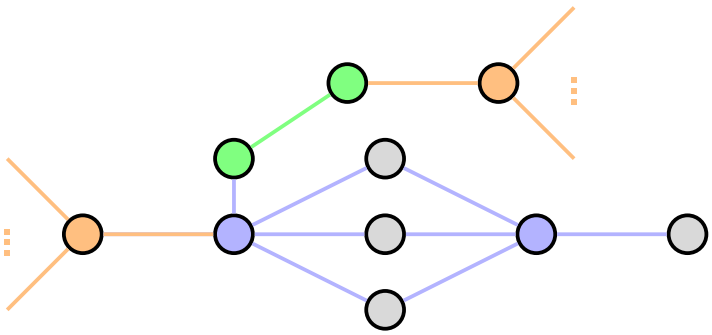

Figure 1: Consider a graph with bounded degree 8. Orange nodes have degree 8.

In round 0, the orange nodes are added to A, because they have degree ≥ 8. Then, the orange edges to be removed. For example, this leads to one of the green nodes being bumped down to degree 1.

In round 1, the blue nodes and edges, with degree 4, are added to A. In round 2, no modifications are made (since no remaining nodes have updated degree two or more). In round 3, the green nodes, now only connected to one another, are both added to A and the green edge is removed.

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

Claim 2 *Algorithm 1 terminates in* $\log_2 \Delta$ *rounds.*

Proof The degree threshold for removing nodes $\frac{\Delta}{2^i}$ becomes less than one after $i = \log_2 \Delta$ rounds. Since any nodes of degree less than one have no edges, all edges will have been removed, and the algorithm will have terminated. ■

Claim 3 *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 by the algorithm's design. ■

Claim 4 *Let θ be any vertex cover of graph G. Each round of Algorithm 1 adds $\leq 2|\theta|$ new nodes to the outputted vertex cover that are not in θ.*

Proof Let Δ denote the maximum degree of a node in G. Let V_i denote the set of all nodes remaining at the start of round i, and let $R_i = V_{i+1} \setminus V_i$ denote the set of all nodes to be removed in round i.

At the start of the ith round, every node in V_i has a maximum updated degree $\leq \frac{\Delta}{2^{i-1}} = 2 \cdot \frac{\Delta}{2^i}$, because larger degree nodes would have been removed in an earlier iteration. Furthermore, every node in R_i has a minimum updated degree $\geq \frac{\Delta}{2^i}$ by the design of the algorithm.

Let $X_i = R_i \setminus \theta$ be the set of nodes removed in round i that are *not* in θ, and let E_{X_i} be the set of edges touching X_i.

The nodes in R_i have minimum updated degree $\geq \frac{\Delta}{2^i}$, so

$$|E_{X_i}| \geq \frac{\Delta}{2^i}|X_i|. \tag{1}$$

Since θ is a vertex cover, every edge in E_{X_i} must have an endpoint in θ. In order for all edges in E_{X_i} to have an endpoint in θ, the number of such edges cannot exceed the total degree of the remaining nodes in θ. The updated degree of *any* remaining node is at most $2 \cdot \frac{\Delta}{2^i}$, so

$$|E_{X_i}| \leq 2 \cdot \frac{\Delta}{2^i}|\theta \cap V_i| \leq 2 \cdot \frac{\Delta}{2^i}|\theta|. \tag{2}$$

Using the transitivity of inequality on equations 1 and 2,

$$\frac{\Delta}{2^i}|X_i| \leq 2 \cdot \frac{\Delta}{2^i}|\theta| \tag{3}$$

$$|X_i| \leq 2|\theta| \tag{4}$$

■

Theorem 5 *Let θ be any optimal vertex cover of graph G. Algorithm 1 will output a valid vertex cover A, such that $|\theta| \leq |A| \leq (1 + 2\log \Delta)|\theta|$.*

Proof The validity of the vertex cover follows from claims 2 and 3, and the statement that $|\theta| \leq |A|$ follows from the optimality of θ.

The proof that $|A| \leq (1 + 2\log \Delta)|\theta|$ follows from claim 4. If each round adds at most $2|\theta|$ nodes not in θ, then the algorithm adds at most $\log \Delta \cdot 2|\theta|$ extra nodes to A overall, and the total number of nodes in the outputted vertex cover is at most $|\theta| + \log \Delta \cdot 2|\theta|$. ■

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

4.2 Using Parnas-Ron Framework

The **Parnas-Ron framework** samples nodes, simulates the distributed algorithm to find the output of those nodes, and extrapolates and estimates using the simulation results. We pick a sample size r based on the desired ϵ approximation precision. Using Chernoff, we require $r = O(\frac{1}{\epsilon^2})$.

The following is an application of the Parnas-Ron framework to the problem of Approximate Vertex Cover.

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

Input : Graph G, ϵ
Output: the estimated size $\hat{v}$ of the minimum vertex cover of G

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

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

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