**6.889 Sublinear Time Algorithms** April 17, 2019

# Lecture 18

*Lecturer: Ronitt Rubinfeld* *Scribe: Spencer Compton*

## 1 Outline

Today we covered the following topics:

- A review of local computation algorithms and maximal independent set.
- A distributed algorithm for maximal independent set (Luby's Algorithm).
- A local computation algorithm for determining the answer of Luby's Algorithm.

## 2 Review of Local Computation Algorithms (LCA) and Maximal Independent Set (MIS)

### 2.1 Local Computation Algorithm: a Model

A local computation algorithm (LCA) outputs a bit $y_i$ of the output if an index $i$ is queries, and it probes several bits $x_j$ of the input. The queries $i_1, i_2, \ldots$ can happen in either sequential or parallel.

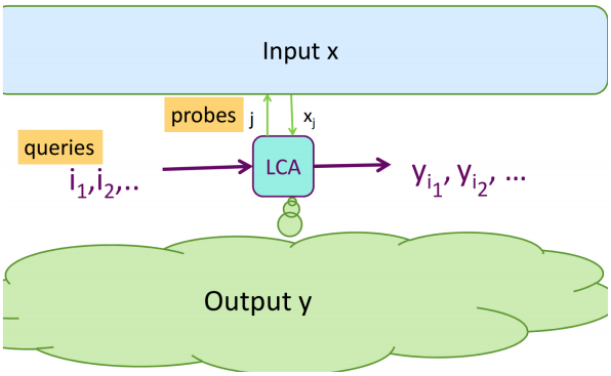


**Figure 1**: A graphical illustration of the model.

Usually, there can be multiple LCAs that process queries at the same time. One difficulty in this model is that there can be more than one valid output. For instance, if we query whether some vertex is in a maximal independent set, we can always answer "yes" to each individual queries, since every vertex is in some maximal independent set. Therefore, we need some way for all LCAs to have a consistent view.

One common method to overcome this issue is to share some (short) random strings among all instances of LCAs at the beginning. Afterwards, each instance needs to compute independently.

### 2.2 Maximal Independent Set

We are focusing on local computation algorithms for the maximal independent set of an undirected graph $G = (V, E)$. Note that we are focusing on computing a *maximal* independent set, not a *maximum* independent set (which is NP-complete).

A subset of nodes $U \subseteq V$ is a maximal independent set (MIS) if and only if:

- $\forall u_1, u_2 \in U : (u_1, u_2) \notin E$ (means $U$ is independent).
- $\nexists w \notin U$ such that $U \cup \{w\}$ is also independent (means $U$ is maximal).

# 3 Distributed Algorithm for Maximal Independent Set: "Luby's Algorithm"

**Algorithm 1** Luby's Algorithm

All nodes start out set to "live"
**for** *k rounds in parallel* **do**
  Every node $v$ tosses a coin and "selects" self with probability $= \frac{1}{2d}$

  **if** *v is live* ***and*** *v selects self* ***and*** *no neighbor of v selects itself* **then**
    $v$ is added to the MIS
    $v$ and its neighbors are removed from the graph (set to "dead")

For purposes of analyses, we will have nodes continue to flip a coin (and possibly select themselves) even after they "die".

If our goal is to "kill" the whole graph then we can make claims about the number of required phases.

**Theorem 1** *Pr[# phases* $\geq 8 \log n] \leq \frac{1}{n}$

**Lemma 2** *Pr[v adds self to MIS in one round* $|v$ *is live]* $\geq \frac{1}{4d}$.

**Proof** $\forall v$ that are live:

Pr[$v$ selects self] $= \frac{1}{2d}$.

Pr[any $w \in N(v)$ selects self] $\leq \sum_{w \in N(v)} \frac{1}{2d} \leq \frac{d}{2d} \leq \frac{1}{2}$. (Where $N(v)$ denotes the set of nodes that are adjacent to $v$).

Thus, Pr[$v$ selects self and no neighbor selects self] $\geq \frac{1}{2d}(1 - \frac{1}{2}) = \frac{1}{4d}$. ■

**Corollary 3** *Pr[v alive after* $4kd$ *rounds]* $\leq (1 - \frac{1}{4d})^{4kd} \leq e^{-k}$.

If we use $k = O(\log n)$, by union bound all nodes will die with high probability. Thus, since we do $4kd$ rounds, we have $O(d \log n)$ phases. Note that it is possible to avoid dependence on $d$ using a smarter analysis.

# 4 Local Computation Algorithm for Luby's Answer

A common way to design a local computation algorithm is to simulate a distributed algorithm. If there is a $k$ round distributed algorithm for maximal independent set, then the output of a particular vertex $v$ depends only on inputs and computations of the $k$-radius ball around $v$. Since there are at most $O(d^k)$ vertices in the $k$-radius ball, we can simulate this algorithm in $O(d^k)$ probes. Unfortunately, the best algorithm runs in $O(\log n)$ rounds, which yields an $O(d^{c \log n}) = 2^{O(\log d \log n)}$ LCA (not sublinear).

We consider running Luby's algorithm for $O(d \log d)$ rounds, with the hope that most (but not all) nodes will be dead. After the $O(d \log d)$ rounds a node will either be live (meaning it may be in the MIS or it may not), or dead (meaning we already know whether or not it is in the MIS).

Using the "Parnas Ron" reduction, on input $v$ we will simulate $v$'s view of computation using $d^{O(\log d)}$ probes to input. We will output whether $v$ is still alive, in, or not in the MIS. We call this our subroutine Lubystatus($v$).

If $v$ is in or out, then we know the answer for our LCA! But we don't know the answer yet if $v$ is still alive.

We will use the following LCA for MIS($v$):

**Algorithm 2** MIS(v)

**if** *Lubystatus*($v$) *returns in or out, return Lubystatus*($v$) **then**

**else**

Do a breadth-first search to find $v$'s connected component of live nodes
Compute lexicographically first MIS $M'$ for $v$'s connected component
Output whether $v$ is in or out of $M'$

In total, this takes $O(d^{O(d \log d)} \times$(size of $v$'s connected component). This is because our most expensive action is the breadth-first search where we call Lubystatus on each node in the connected component. Thus, we need to bound the size of live components.

## 4.1 Bounding Live Connected Component Sizes

If a node $v$ survives all rounds, then there is no round such that $v$ picks itself and $w \in N(v)$ picks themselves.

Let $d(u, v)$ denote the minimum distance between two nodes $u$ and $v$. Note that the survival of any pair of nodes $u, v$ such that $d(u, v) \geq 3$ are independent of each other. This is because the survival of $u$ only depends on the coin flips of $u$ and nodes in $N(u)$. Similarly, the survival of $v$ only depends on the coin flips of $v$ and $N(v)$. Since $d(u, v) \geq 3$, we know $(u \cup N(u)) \cap (v \cup N(v)) = \emptyset$.

Additionally, the survival of $v$ for $c \times d \log d$ rounds is rare.

$Pr[v \text{ survives}] \leq (1 - \frac{1}{4d})^{cd \log d}$

$\leq \frac{1}{8d^3}$ (if we use $c \geq 20$).

This gives us some intuition why the size of connected components might be very small. Survival is very rare, and a large component will have a large set of nodes whose survivals are all independent.

**Theorem 4** *Let $w$ be the size of the largest connected component of survivors. After $O(d \log d)$ rounds, $w \leq O(polylog(d) \log n)$ with high probability.*

**Proof** Our main idea is that any connected component that is large has many nodes that are independent (all distance $\geq 3$ from each other). These independent nodes are very unlikely to all simultaneously survive. However, if we needed to union bound over all sets of nodes of size $w$, this would be hard. Instead, we will want a tighter bound for the number of connected components of size $w$.

Let $H^{(3)}$ be a graph such that two nodes $u, v$ have an edge between them iff $d(u, v) = 3$ in the original graph $G$. Based on this definition, clearly $deg(H^{(3)}) \leq d^3$. Additionally, any subset of nodes that is an independent set in $H^{(3)}$ all have independent survivals.

How do we bound the number of connected components of size $w$ in $G$? Let's first bound the number of size $w$ subtrees in $H^{(3)}$.

**Lemma 5** *The number of size $w$ subtrees in $H^{(3)}$ is $\leq n(4d^3)^w$.*

**Proof** Consider the following process of building a tree:

1. Choose a root. ($n$ choices)

2. Choose structure of the tree. Consider the process of choosing the structure. We represent this with a binary string of length $2(w-1)$. At every point, you can choose to go "down" adding a child (which we represent as a "1") and "up" going to the parent of the current node (which we represent as a "0"). Valid shapes never have more 0's than 1's in a prefix, and have an equal number of 0's and 1's. This is upper bounded by $4^w$ ($4^w$ choices). .

3. Every time we go "down" choose what node we are going to. From our current node, we only have $\deg(H^{(3)}) \leq d^3$ options ($(d^3)^w$ total choices).

Thus, in total, we upper bound that there are $\leq n(4d^3)^w$ subtrees! ■

With an independent set $I$ in $H^{(3)}$ of size $w$, we know that $I$'s nodes are $\geq$ distance 3 in G and thus have independent survival.

$\Rightarrow$ Pr[independent set $I$ survives in $G$] $\leq (\frac{1}{8d^3})^{|I|}$

$\Rightarrow$ Pr[specific size $w$ tree survives in $H^{(3)}$] $\leq (\frac{1}{8d^3})^w$

$\Rightarrow$ Pr[$\exists$ size $w$ tree surviving in $H^{(3)}$] $\leq \frac{n(4d^3)^w}{(8d^3)^w} \leq \frac{n}{2^w}$

If we use $w = O(\log n)$, Pr[$\exists$size $w$ tree surviving in $H^{(3)}$] $\leq \frac{1}{n}$

$\Rightarrow$ Pr[$\exists$ size $wd^3$ component surviving in $G$] $\leq \frac{1}{n}$

Thus, it is unlikely to have any surviving component of size $\Omega(d^3 \log n)$!

■

Earlier we showed our algorithm runs in $O(d^{O(d \log d)} \times$(size of $v$'s connected component). Thus, we now know our LCA for MIS runs in $O(d^{O(d \log d)} \log n)$!