CS 514: Advanced Algorithms II – Sublinear Algorithms **Rutgers: Spring 2020**

Lecture 5

February 18, 2020

Instructor: Sepehr Assadi *Scribe: Aditi Dudeja*

1 Local Computation Algorithms

Motivation Behind The Model

So far the problems we have dealt with had small output, for instance: is a graph connected or not? what is the approximate average degree of the graph? does a given object or distribution have some certain property?

Of course, having a small output did not necessarily mean that we could right away obtain a sublinear time algorithm for these problems. However, one thing should be clear: if the output itself is "large", i.e., proportional to the input size, then there should be no hope of getting a sublinear time algorithm for the problem as simply writing down the output would be too time consuming. But what if we indeed aim to solve a problem with a large output, say, output a proper coloring of a given graph?

This is the motivation behind Local Computation Algorithms introduced originally by Rubinfeld, Tamir, Vardi, and Xie [5]. In this model, the local computation algorithm (LCA) allows the user to *probe* several bits of the output – we then require that the answer to each probe to be determined in a sublinear time.

The LCA Model

For concreteness, we are going to focus on the *maximal independent set (MIS)* problem in this lecture and define LCAs with respect to this problem directly.

Definition 1 (**Maximal Independent Set**)**.** For a graph $G = (V, E)$, a maximal independent set $\mathcal{M}$ of G is any set of vertices such that: (1) $\mathcal{M}$ is an independent set, namely, no vertices in $\mathcal{M}$ are neighbor to each other, and (2) $\mathcal{M}$ is maximal, i.e., no other independent set of G contains $\mathcal{M}$ as a proper subset (alternatively, any vertex not in $\mathcal{M}$ has a neighbor inside $\mathcal{M}$).

Remark. A closely problem related to MIS (at least on the surface) is that of computing a *maximum* independent set. Note however that even though the latter problem is NP-hard (and even hard to approximate to within almost any non-trivial factor), the MIS problem can be easily solved in polynomial time by a greedy algorithm: Simply go over the vertices of the graph one by one in an arbitrary order, add the first available vertex to the MIS and remove all its neighbors from the graph, and continue.

An LCA for MIS? *Ideally*, we would like an LCA to, given a graph $G = (V, E)$, compute some (fixed) MIS $\mathcal{M}$ of G to answer the following probes "quickly":

- `in-MIS`(v): Is v part of the MIS $\mathcal{M}$?

One way of solving the above problem is to take the graph G, compute an MIS of the graph, and store it explicitly; we can then answer the probes accordingly based on this stored MIS. However, this approach is too costly both *time*-wise (linear time to create the MIS) and *space*-wise (linear space to store the MIS). The goal of the LCA is to provide such an abstraction implicitly, *as if*, the LCA indeed stored a *fixed* MIS and is answering all probes based on that. Formally, we require the LCA to satisfy the following properties:

(i) The answer to each probe `in-MIS`(v) should be *only* a function of v. This in particular means that LCA should be memoryless across the probes in the sense of that running `in-MIS`(v) followed by `in-MIS`(u) give the same output as running `in-MIS`(u) first and `in-MIS`(v) next. In other words, one should be able to run multiple probes *in parallel* and obtain the same result.

(ii) The answer to the probes should be *consistent*, i.e., correspond to a *fixed* MIS of the graph. In other words, if we run `in-MIS`(v_1), ..., `in-MIS`(v_n) for all $v_1, \ldots, v_n \in V$, we obtain a single MIS of G.

(iii) The LCA should answer each probe fast, i.e., in *sublinear* time (we will quantify this further later).

(iv) Finally, these constraints are very strict and hard to satisfy and hence to ensure consistency across different probes, we also provide a string of *shared random bits* to be used across all probes. I.e., even though the LCA should be able to answer the probes in parallel and consistently, it is *allowed* to use these shared random bits across different probes (again, we will quantify this further below).

It is again worth emphasizing that the requirements above are in place to give the user this abstraction that a single MIS of the graph G is *already* computed and the user can probe any of its entries quickly.

Query access to the graph: We are almost always interested in designing LCAs for *bounded degree* graphs, specified in the *adjacency list* query model. In particular, we assume we are given an upper bound on the maximum degree of the graph, denoted by Δ, and that each vertex has a unique ID from $[n]$. Moreover, for answering each probe, the LCA is provided access to the graph via the adjacency list oracle, which allows for querying the degree $\deg(v)$ of a vertex v or the list of neighbors of v.

Measuring performance: The performance of an LCA is measured using the following parameters:

1. **Probe Time**. The time it takes by the LCA in order to answer a single probe made by the user.
2. **Randomness**. Number of random bits used by the LCA that needs to be stored explicitly and shared between different probes (this can also be seen as a measure of space complexity).

We would like both parameters above to be $f(\Delta) \cdot \text{polylog}(n)$, for some function f. This way, if the degree of the graph is very small, say a constant, then the total time that takes to answer each probe is only $O(\text{poly}\log(n))$ which is quite fast (and sublinear in the input size).

Before moving on, let us give some examples of what will happen if we relax the constraints of LCA above.

Remark. Consider the following natural approach for MIS problem. For a vertex v, answer `in-MIS`(v) with *Yes*, unless you answered *Yes* for some neighbor of v already in which case answer *No* instead. This trivially solves the problem using only $O(\Delta)$ time per probe. However, the answer in this case depends on the previous probes. In fact, if we change the order of the probes, we get a different answer (which is different from the abstraction we had in mind). To prevent such strategies, we insist that our LCA be memoryless and its answer to each probe to be only function of the probe (and shared randomness), i.e., satisfies constraint (i).

Similarly, one could have considered an LCA that always outputs *Yes* for each `in-MIS`(v) – after all, every vertex v is part of *some* MIS of the input graph. However, constraint (ii) of the LCA rules out the possibility of such LCAs as the answer across different probes should be consistent with a single fixed MIS of the input.

2 A Local Computation Algorithm for MIS

To get an LCA for MIS, a strategy would be to make sure the answer to each probe `in-MIS`(v) is *local* to the neighborhood of v, i.e., we could determine whether v is part of our MIS or not by examining only the

vertices in the *R-hop neighborhood* of v[1]. This way, whenever the user probes `in-MIS`(v), we simply need to consider $O(\Delta^R)$ vertices in the neighborhood of v which makes the runtime independent of n. We will examine this strategy for LCA in the following.

Distributed LOCAL Model for MIS

There is a very rich literature in TCS on studying algorithms in which decision of each vertex is local to its neighborhood, referred to as the distributed LOCAL model. Similar to LCAs, we also define this model in the context of the MIS problem for concreteness.

Distributed LOCAL Model

1. The input graph $G = (V, E)$ is the communication network itself and the goal is to find an MIS $\mathcal{M}$ inside this graph G.
2. In each round, communication happens where each vertex v is allowed to send an arbitrarily long message to its neighbors (possibly different messages to different neighbors).
3. Computation happens between rounds and based on the received messages, the vertices can decide their actions for the next round.
4. At the end of the last round, every vertex v knows whether it belongs to the MIS $\mathcal{M}$ or not.
5. The goal is to minimize the number of rounds of the algorithm.

In an R-round distributed algorithm for MIS (or any other problem), the decision of each vertex v is only a function of the R-hop neighborhood of v. This is because messages sent by vertices at distance greater than R from v never reach v in an R-round algorithm. As such, to *simulate* an R-round distributed LOCAL algorithm for a single vertex v, we only need to simulate the algorithm in the R-hop neighborhood of v which contains at most $O(\Delta^R)$ vertices. In other words, as desired, the "status" of a vertex in an R-round distributed LOCAL algorithm only depends on the R-hop neighborhood of the vertex (the output would be the same for v even if we change the entire graph outside the R-hop neighborhood of v).

The Distributed algorithm for MIS: We now propose a simple distributed LOCAL algorithm for MIS. This algorithm is a *weaker* version of the celebrated algorithm of Luby for MIS in this model [4], and as such, we are going to call it Luby's weak MIS algorithm.

Luby's Weak MIS Algorithm:

1. Initially, $Active(v) = True$ and $\mathcal{M}(v) = False$ for every vertex $v \in V$.
2. In each round, every vertex v runs the following in *parallel*:
   (a) Vertex v *selects* itself with probability $\frac{1}{2\Delta}$.
   (b) If $Active(v) = True$, v is selected, and no neighbor of v is selected, then $\mathcal{M}(v) = True$ and $Active(u) = False$ for $u \in v \cup N(v)$ (we say u is *deactivated* in this case).

We have the following lemma regarding the correctness of the algorithm.

Lemma 2. *Let $\mathcal{M}$ be the set of vertices $v \in V$ where $\mathcal{M}(v) = True$. Then: (i) $\mathcal{M}$ is always an independent set in the algorithm, and (ii) when no vertex remains active, $\mathcal{M}$ becomes an MIS of G.*

Proof. Observe that if we are adding a vertex v to $\mathcal{M}$, we are deactivating neighbors of v and therefore

[1] Recall that the R-hop neighborhood of v is the set of vertices that are within distance at most R from v in G.

eliminating their chances of ever being added to $\mathcal{M}$. So, $\mathcal{M}$ is an independent set at all times.

Further, a vertex v is deactivated only when either v or one of its neighbors is added to $\mathcal{M}$, i.e. for every deactivated v either $v \in \mathcal{M}$ or there exists a vertex $u \in N(v)$ such that $u \in \mathcal{M}$. It follows that when all vertices are deactivated, no other vertex can be added to $\mathcal{M}$, and therefore $\mathcal{M}$ is a MIS. □

Now, we are only left with task of proving that the number of rounds in this algorithm before all vertices are deactivated is small. In the following, we prove a more general statement that will be used in later part of the lecture as well.

Lemma 3. *Fix any vertex $v \in V$. For any integer $R \geq 1$, after running Luby's weak MIS algorithm:*

$$\Pr\left(v \text{ is still active after } R \text{ rounds}\right) \leq \exp\left(-\frac{R}{4\Delta}\right).$$

Proof. For any vertex $v \in V$ and round $r \geq 1$, define:

- $A_r(v)$: the event that v is active at the *end* of round r;
- $S_r(v)$ the event that v is selected in round r;
- $M_r(v)$: the event that v is added to $\mathcal{M}$ in round r;

We now have,

$$\begin{aligned}
\Pr\left(\overline{A_r(v)} \mid A_{r-1}(v)\right) &\geq \Pr\left(M_r(v) \mid A_{r-1}(v)\right) && \text{(if } v \text{ is added to } \mathcal{M} \text{ it no longer remains active)} \\
&= \Pr\left(S_r(v) \wedge \forall u \in N(v) : \overline{S_r(u)}\right) \\
& && (M_r(v) \text{ happens iff } v \text{ is selected and no neighbor of } v \text{ is selected}) \\
&= \Pr\left(S_r(v)\right) \cdot \Pr\left(\forall u \in N(v) : \overline{S_r(u)}\right) \\
& (S_r(v) \text{ is only a function of randomness of vertex } v \text{ in round } r \text{ and independent of } A_{r-1}(v)) \\
&= \frac{1}{2\Delta} \cdot \left(1 - \Pr\left(\exists u \in N(v) : S_r(u)\right)\right) && \text{(as } v \text{ is selected with probability } 1/2\Delta) \\
&\geq \frac{1}{2\Delta} \cdot \left(1 - \sum_{u \in N(v)} S_r(u)\right) && \text{(by union bound)} \\
&\geq \frac{1}{2\Delta} \cdot \left(1 - \Delta \cdot \frac{1}{2\Delta}\right) && \text{(as maximum degree is at most } \Delta) \\
&= \frac{1}{4\Delta}.
\end{aligned}$$

Hence, in the algorithm, any vertex which is not yet deactivated, will become deactivated with probability $1/4\Delta$ in this round. As such, we can write,

$$\begin{aligned}
\Pr\left(A_R(v)\right) &= \prod_{r=1}^{R} \Pr\left(A_r(v) \mid A_{r-1}(v)\right) \\
& (\text{by definition of conditional probability as } A_R(v) = A_1(v) \wedge A_2(v) \cdots \wedge A_R(v)) \\
&\leq (1 - \frac{1}{4\Delta})^R && \text{(as computed above)} \\
&\leq \exp\left(-\frac{R}{4\Delta}\right). && (\text{as } 1 - x \leq e^{-x})
\end{aligned}$$

This concludes the proof. □

Lemma 3 implies that by taking $R = 8\Delta \cdot \ln n$, any fixed vertex remains active after R rounds with probability only $1/n^2$. A union bound on all vertices then ensures that with high probability of $1 - 1/n$, Luby's weak MIS algorithm terminates after $R = 8\Delta \cdot \ln n = O(\Delta \cdot \log n)$ rounds.

Remark. A careful reader may notice that Luby's weak MIS algorithm that we described makes somewhat inefficient (and counterintuitive) decisions: for instance, we are selecting *all* vertices even those that are already deactivated and hence disallow an active and selected vertex to join MIS even if only its *deactivated* neighbors are selected in this round; or we are selecting a vertex with probability (inversely) proportional to Δ even if its actual degree (or better yet, its degree to still active neighbors) is much smaller. Indeed these choices significantly increase the number of rounds needed by the algorithm and using Luby's original algorithm [4], one can reduce the number of rounds to $O(\log n)$ (independent of Δ). That being said, we still work with this Luby's weak MIS algorithm as the analysis of Luby's original algorithm is more involved and does *not* lend itself to the application we will use in LCAs in the next part (in particular, an analogue of Lemma 3 is not known for Luby's original algorithm)[a].

[a] The recent work of Ghaffari [1] addresses this shortcoming by presenting an algorithm in-spirit-of Luby's algorithm with this additional guarantee that also finishes in $O(\log n)$ rounds, namely, achieve best of both algorithms.

A Local Computation Algorithm From Luby's Weak MIS Algorithm

Recall our plan of using Luby's weak MIS algorithm in an LCA to answer the probes `in-MIS`(v) by simulating the behavior of vertex v in this distributed algorithm. We can simulate Luby's weak MIS algorithm for $R = O(\Delta \log n)$ rounds to know for sure whether v should be part of the MIS or not. But considering the R-hop neighborhood of v in this case takes $\Delta^{O(\Delta \cdot \log n)} \geq O(n)$ time which is not sublinear. We are proposing the following fix.

Instead of running Luby's weak MIS algorithm to termination, we do the following: we run the algorithm for $R = O(\Delta \log \Delta)$ rounds which allows us to argue that most (but not all) of the vertices are deactivated already (by Lemma 3). Now, if the user probes a vertex v that is deactivated, we can output whether or not it is in the MIS already (by simulating the R-hop neighborhood in $\Delta^{O(\Delta \cdot \log \Delta)}$, *independent* of n). But what can we do with vertices that are still undecided (i.e., remain active after running Luby's weak MIS algorithm)? This is where the second main idea of the LCA kicks in.

We are going to use an idea that is typically referred to as *(graph) shattering*. Roughly speaking, after running Luby's weak MIS algorithm for a limited number of rounds, what remains of the graph (induced subgraph of G on the undecided vertices) is a collection of *small* connected components. Thus, once the graph is shattered, we can switch to a simple deterministic algorithm to finish off the problem in these remaining small components; even though the runtime of the deterministic algorithm depends linearly on the number of vertices, since size of each component is small enough ($\approx \log n$), we can still visit all these vertices when answering each probe.

We now formalize this approach and design our LCA. First, define the following subroutine.

LubyStatus(v, R):

1. Simulate Luby's weak MIS algorithm for R rounds on vertex v (we assume that the random bits used by Luby's weak MIS algorithm are written in a table and this subroutine can simply read them).
2. If $Active(v) = False$ then return IN-MIS or NOT-IN-MIS depending on the status of v in the algorithm; else return UNDECIDED.

LubyStatus(v, R) algorithm allows us to determine the outcome of Luby's weak MIS algorithm after R rounds for the vertex v. We now use this to design our LCA for response to `in-MIS`(v) probes.

MaxIndSetLCA(v): Returns the answer in response to the probe `in-MIS`(v).

1. Let $R := 12\Delta \cdot \log(8\Delta)$.
2. Check $status =$ **LubyStatus**(v, R).
3. If $status =$ IN-MIS or NOT-IN-MIS, then return $status$.
4. Else, find the connected component of *undecided* vertices to which v belongs, denoted by C_v, as follows:
   (a) Run DFS on v and for any visited vertex u in the DFS, compute **LubyStatus**(u, R) to decide whether u is still undecided or not – continue the DFS only on the undecided vertices until all of C_v is visited.
5. Compute the *lexicographically-first* MIS of C_v by running the greedy algorithm for MIS on the fixed ordering of vertices with smaller ID to larger ID. Return whether v belongs to the MIS or not accordingly.

We shall emphasize that in **MaxIndSetLCA**, we have stored the random bits used by Luby's weak MIS algorithm in a table of size $\Theta(n \cdot R)$ so they can be shared across the probes. Note that this makes the random bit complexity of our LCA *not* sublinear. We will revisit this issue in later lectures and problem sets and show how to limit the number of true random bits needed by the algorithm to something much smaller while maintaining the desired guarantees.

Remark. In general, the issue of using "large" amount of random bits can be addressed by using *pseudorandom number generators (PRG)* in a unified way[a]. However, PRGs are quite complicated (are considered a "heavy hammer" for most applications) and for our purpose we can typically get away with using simpler ideas such as *pair-wise independent hash functions* as we will see in later in the course.

[a] As PRGs are not the focus of this course, we will not get into their (technical) definition and simply mention (vaguely) that they are functions that are given a "small" number of truly random bits and generate as output a "large" number of bits that are "random enough" for most applications.

In the following, we first prove the correctness of this LCA and then bound its runtime for each probe.

Correctness of MaxIndSetLCA. The correctness of the algorithm follows from (i) the fact that Luby's weak MIS algorithm at any point of time maintains an independent set (Lemma 2), and that (ii) the undecided vertices are *not* adjacent to any vertices that are in the MIS already (otherwise they would be deactivated). So, an MIS of the entire graph can be found independently for any connected component of undecided vertices. As we are picking a fixed MIS of each connected component of undecided vertices (consistently across any probe that leads to the same connected component), the output of the algorithm is always consistent with respect to a fixed MIS of the input graph.

Runtime. We are going to prove the following lemma on the runtime of the algorithm.

Lemma 4. *With high probability of $1 - 1/n^4$, the runtime of the algorithm in answering any given probe* `in-MIS`(v) *is $\Delta^{O(\Delta \cdot \log \Delta)} \cdot \log n$ (the high probability bound holds* simultaneously *for all $v \in V$).*

We first have the following simple claim.

Claim 5. *For any $v \in V$, the time it takes to answer* `in-MIS`(v) *is $\Delta^{O(\Delta \log \Delta)} \cdot |C_v|$.*

Proof. Running **LubyStatus**(u, R) for any vertex $u \in V$ takes $\Delta^{O(\Delta \log \Delta)}$ time by the choice of $R = O(\Delta \log \Delta)$ and so there are at most Δ^R vertices in the R-hop neighborhood of u that needs to be simulated

when considering the distributed algorithm. Now, when running **MaxIndSetLCA**(v), we need to run **LubyStatus**(u, R) for $u = v$ and $u \in N(w)$ for any $w \in C_v$. As such, the total runtime is

$$O(\Delta) \cdot \Delta^{O(\Delta \log \Delta)} \cdot |C_v| = \Delta^{O(\Delta \log \Delta)} \cdot |C_v|,$$

as stated. □

By Claim 5, we only need to bound size of C_v to bound the runtime of **MaxIndSetLCA**(v). In particular, to obtain the bounds in Lemma 4, we only need $|C_v| = \text{poly}(\Delta) \cdot \log n$ for all $v \in V$. We do so in the following. For any $v \in V$, define:

- $U(v)$: the event that the status of vertex v is undecided at the end of a run of **LubyStatus**(v, R).
- For any $S \subseteq V$, we further define $U(S) := \wedge_{v \in S} U(v)$, namely, the event that all vertices in S are undecided.

As a corollary of Lemma 3 and by the choice of R, we have,

$$\Pr\left(U(v)\right) \leq \exp\left(-\frac{R}{4\Delta}\right) = \exp\left(-\frac{12\Delta \cdot \log\left(8\Delta\right)}{4\Delta}\right) = \frac{1}{(8\Delta)^3}. \tag{1}$$

Let us interpret Eq (1) as follows: the probability that a vertex of the graph *survives*, i.e., remains undecided, is only $\approx \Delta^{-3}$. *If* it was the case that the vertices were surviving *independently* of each other, then we should expect the graph of survived vertices to not have a large connected component. To formally argue this, we can simply pick a subset $S \subseteq V$ of vertices and then compute the probability that S remains a connected component of survived vertices; we can then do a union bound over all choices of S and finalize the proof (this step requires some more care as will become evident later in the proof).

The problem is that these events are not entirely independent. However, an important observation is the following claim.

Claim 6. *For any two vertices v and u that are within distance at least 3 of each other, $U(v)$ and $U(v)$ are independent.*

Proof. The event that a vertex remains active in Luby's weak MIS algorithm is only a function of randomness of $v \cup N(v)$ and since $v \cup N(v)$ is disjoint from $u \cup N(u)$ (as otherwise distance of u, v would be less than 3), we get that $U(v)$ and $U(u)$ are independent. □

Since the total number of vertices in distance less than 3 around v is small (only Δ^2), Claim 6 allows us to still consider most events $U(v)$ independent of each other and carry out the analysis.

We now formalize the above strategy. We first need the following definition:

- Define $G^{(3)}$ as a graph on vertices $V(G)$ such that $(u, v) \in E(G^{(3)})$ iff $d(u, v) = 3$ in G, i.e., we connect vertices in $G^{(3)}$ together that are within distance exactly three in G.

Claim 7. *Let $S \subseteq V$ be such that $G[S]$, the induced subgraph of G on S, is connected. Then there exists a set $T \subseteq S$ such that:*

(i) *$G^{(3)}[T]$ is connected;*

(ii) *$|T| \geq |S|/\Delta^2$;*

(iii) *$\Pr\left(U(T)\right) \leq (1/8\Delta^3)^{|T|}$.*

Proof. We construct T from S, using the following procedure:

1. Pick an arbitrary v from S and add it to T. Remove v and all u such that $d(u,v) < 3$ in G from C.
2. Pick a vertex at distance exactly 3 from some vertex in T and continue the above procedure until no vertex remains (as $G[S]$ is connected this procedure either finds a new vertex to add to T or no vertices are left anymore).

Firstly, every new picked vertex v has distance exactly 3 from some vertex already in T and thus there is an edge between v and some vertex in T in $G^{(3)}$. So, we can inductively prove that $G^{(3)}[T]$ is connected.

Secondly, each time we include a vertex v in T, we are excluding $\leq \Delta^2$ vertices that are within distance < 3 from v, so $|T| \geq |S|/\Delta^2$.

Finally, the distance between any pair of vertices in T is at least 3 (as we remove vertices with distance < 3) and thus the event $U(v)$ and $U(u)$ for $u, v \in T$ are independent by Claim 6. The bound in part (iii) now follows from Eq (1). □

Let s be an integer and define $\mathcal{T}_s$ as the set of all $T \subseteq V$ such that $G^{(3)}[T]$ is connected and has s vertices. We will need the following claim.

Claim 8. *Fix $s \geq 1$ and consider $\mathcal{T}_s$. Suppose the event $U(T)$ does* not *happen for* any *$T \in \mathcal{T}_s$. Then, the size of the largest connected component of undecided vertices in G is at most $s \cdot \Delta^2$.*

Proof. Suppose towards a contradiction that this is not rue. Let S be a connected component of size $> s \cdot \Delta^2$ in undecided vertices of G. This means that $U(S)$ has happened which means $U(v)$ has happened for all $v \in S$. But by Claim 7, there should exist a set $T \subseteq S$ such that $|T| = s$, $T \in \mathcal{T}_s$, and that $U(T)$ has happened as well. This contradicts the assumption that $U(T)$ did not happen for any $T \in \mathcal{T}_s$. □

To finalize the proof of Lemma 4, we only need to show that for some sufficiently small s, $U(T)$ does not happen for any $T \in \mathcal{T}_s$ with high probability. By Claim 7, we know that

$$\forall T \in \mathcal{T}_s \qquad \Pr\left(U(T)\right) \leq \left(\frac{1}{8\Delta^3}\right)^s. \tag{2}$$

We are going to bound the size of $\mathcal{T}_s$ and then apply a union bound using Eq (2) to finalize the proof.

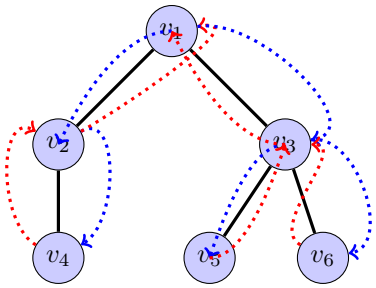
Claim 9. *For any $s \geq 1$, $|\mathcal{T}_s| \leq n \cdot (4\Delta^3)^s$.*

Proof. To construct a connected subgraph $T \in \mathcal{T}_s$ inside $G^{(3)}$, we can focus on creating a tree-subgraph of $G^{(3)}$ on s vertices. We argue that any tree of size s in $G^{(3)}$ can be created by the following procedure:

1. First choose a root of the tree. This can be done in n possible ways.
2. Next we choose the "skeleton" of the tree. This can be done by representing the DFS sequence of the tree in the form of a binary string of length $2(s-1)$. While doing a DFS of the tree, the act of going from the parent to a child is represented by 1, and going from child to parent is denoted by 0. Number of such strings is upper bounded by $2^{2(s-1)} < 4^s$. Also, in each of these strings, the number of 0's and 1's are equal, since each edge is traversed twice, once while from parent to child, and another time while going from child to parent. See Figure 1 for an example.
3. Next, we need to label the nodes of this skeleton with vertices. Each time we go from a parent to a child, we have at most Δ^3 options corresponding to at most Δ^3 vertices that are neighbor to the current vertex of $G^{(3)}$. That is, we have Δ^3 ways of filling each 1 in the string. As the number of 1's is exactly $(s-1)$, the total number of these choices is bounded by $\Delta^{3\cdot(s-1)} \leq \Delta^{3s}$.

It follows that number of such trees is at most $n(4\Delta^3)^s$. □

We can now finalize the proof of Lemma 4.



DFS sequence: $v_1, v_2, v_4, v_2, v_1, v_3, v_5, v_3, v_6, v_3, v_1$
Corresponding binary sequence: 1100110100

Figure 1: DFS sequence and binary strings

Proof of Lemma 4. Let $s = 5 \log n$. We have,

$$\begin{aligned}
\Pr\left(\exists T \in \mathcal{T}_s : U(T)\right) &\leq \sum_{T \in \mathcal{T}_s} \Pr\left(U(T)\right) && \text{(union bound)} \\
&\leq |\mathcal{T}_s| \cdot \left(\frac{1}{8\Delta^3}\right)^s && \text{(by Eq (2))} \\
&\leq n \cdot (4\Delta^3)^s \cdot \left(\frac{1}{8\Delta^3}\right)^s && \text{(by Claim 9)} \\
&\leq n \cdot \frac{1}{2}^s = n \cdot \frac{1}{n^5} = \frac{1}{n^4}.
\end{aligned}$$

By Claim 8, this implies that with probability $1 - 1/n^4$, size of the largest connected component of undecided vertices in G is at most $\Delta^2 \cdot s = 5\Delta^2 \cdot \log n = O(\Delta^2 \cdot \log n)$. By Claim 5, this implies with probability $1 - 1/n^4$, the runtime of the LCA (for each probe) is

$$\Delta^{O(\Delta \log \Delta)} \cdot O(\Delta^2 \cdot \log n) = \Delta^{O(\Delta \log \Delta)} \cdot \log n,$$

finalizing the proof. □

By putting everything together, we obtain the following theorem.

Theorem 10. *There is an LCA for the maximal independent set problem that with high probability, answers each probe $\Delta^{O(\Delta \log \Delta)} \log n$ in time.*

This theorem was first proved by Rubinfeld, Tamir, Vardi, and Xie [5]. The runtime of this LCA was improved to $\Delta^{O(\log^2 \Delta)} \cdot \log n$ by Levi, Rubinfeld, and Yodpinyanee [3] and then to $\Delta^{O(\log \Delta)} \cdot \log n$ by Ghaffari [1]. Recently, this runtime was further improved to $\Delta^{O(\log \log \Delta)} \cdot \log n$ by Ghaffari and Uitto [2]. Obtaining LCAs for this problem with runtime $\Delta^{O(1)} \cdot \text{poly} \log(n)$ (i.e., polynomial in both Δ and $\log n$) remains one of the main longstanding open problems in the LCA literature.

References

[1] Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 270–277, 2016. 5, 9

[2] Mohsen Ghaffari and Jara Uitto. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1636–1653, 2019. 9

[3] Reut Levi, Ronitt Rubinfeld, and Anak Yodpinyanee. Local computation algorithms for graphs of non-constant degrees. *Algorithmica*, 77(4):971–994, 2017. 9

[4] Michael Luby. A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 1–10, 1985. 3, 5

[5] Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In *Innovations in (Theoretical) Computer Science - I(T)CS 2010, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 223–238, 2011. 1, 9