6.5240 Sublinear Time Algorithms	November 14, 2022
Lecture 18	
Lecturer: Ronitt Rubinfeld	*Scribe: Jacob Teo*

In this lecture, we continue discussing the local computation algorithm for the maximal independent set problem.

1 Review

1.1 Maximal Independent Set (MIS)

Definition 1 *Given a graph $G = (V, E)$ with maximum degree d, a set of nodes $U \subseteq V$ is a maximal independent set (MIS) if*

- $\forall u_1, u_2 \in U, (u_1, u_2) \notin E$ *("independent")*
- $\nexists w \in V \setminus U$ *such that* $U \cup \{w\}$ *is independent ("maximal")*

Note that this is different from the *maximum* independent set, which is NP-Hard. There are many possible *maximal* independent sets, and one can be found by any simple greedy algorithm.

1.2 Local Computation Model

In a general local computation model, we are trying to compute a queried subset of a large output, given a large input. In this case, we have a large graph G and we want to answer a query like "is node i in the MIS?" with a **local computation algorithm (LCA)**. We want our algorithm to satisfy:

- Consistency: there must exist some MIS of G that is consistent with the result of all queries.
- Sublinear time: a query should only access an asymptotically small subset of the graph.

1.3 Parnas-Ron Reduction

Suppose we have a k-round distributed algorithm to compute the MIS. Then the output to a single node depends only on its radius-k ball in G, and hence the *Parnas-Ron reduction* (first covered in Lecture 4) allows us to convert a k-round distributed algorithm to a sequential d^k-time algorithm for answering a query to a single node.

1.4 Luby's Algorithm

We first present Luby's (modified) distributed algorithm for MIS.

Algorithm 1

1. MIS $\leftarrow \emptyset$
2. Set status of all nodes to *live*
3. We repeat k times in parallel:
   (a) Each *live* node v performs this independently: With probability $\frac{1}{2d}$, color self red, else color self blue. Send color to all neighbors.
   (b) If v is red and all its *live* neighbors are blue:
   - Add v to MIS
   - Set v and all its neighbors to *dead* (equivalently, remove them from the graph).

In the original version of Luby's algorithm, we repeat until all nodes are *dead*, and output the resulting MIS. For the purpose of analyzing the local computation algorithm later, we assume that nodes *continue to select their color*, but do not send the color to their neighbors.

Let X be the number of phases needed. Then we have the following (refer to Lecture 17 for more details):

Lemma 2 $\Pr[X \geq 8 \log n] \leq \frac{1}{n}$

Corollary 3 $\text{Exp}[X] = O(d \log n)$

Lemma 4 *For any node v,* $\Pr[v \text{ is live and added to MIS in a round}] \geq \frac{1}{4d}$.

Corollary 5 *For any node v,* $\Pr[v \text{ is live after } 4kd \text{ rounds}] \leq e^{-k}$.

Our goal is to use the Parnas-Ron reduction to obtain a sublinear-time LCA. However, if our number of phases has a $\log n$ term, then the resulting sequential algorithm must take $\Omega(n)$ time.

Idea: If we ran Luby's algorithm for fewer rounds, some nodes would still be *live*. Intuitively, these surviving nodes would be spread out across the graph, and hence form small (sublinear-size), independent connected components. We can then process these components individually.

2 Local Computation Algorithm for MIS

We define a new *local* algorithm, *Luby-Status* or $LS(v)$. This runs Luby's algorithm described above for $k = O(d \log d)$ rounds and returns the status of node v. There are 3 possible outputs:

- v is *live*
- v is in MIS
- v is not in MIS

With the Parnas-Ron reduction, we can compute $LS(v)$ in $d^{O(d \log d)}$ time. We now define our LCA:

Algorithm 2: MIS(v)

1. Run the sequential version of $LS(v)$. If the output is "in MIS" or "not in MIS", output that and halt.
2. Otherwise, v is *live*.
   (a) Perform a BFS to find the connected component of v consisting only of *live* nodes.
   (b) Compute lexicographically smallest MIS M' for that connected component
   (c) Output whether v is in M'.

The reason we want the lexicographically smallest MIS is because our algorithm has to be consistent, and hence the MIS that our algorithm yields should be uniquely defined by the set of nodes in the connected component, and this can be done quickly. For example, we can define a greedy algorithm where we start from the lowest-label node, and at any node we visit neighbors in increasing label order. This yields a deterministic output, and runs in $O((\text{size of component}) \cdot d \log d)$ (taking into account the need to sort neighbors by label).

2.1 Dealing with randomness

Something to note is that while the algorithm itself is randomized, the result should be consistent and reproducible (in a given round and for a given node, the color should always be the same). Thus, the output of a given MIS(v) should always be the same. To achieve this, we assume access to a "repository" of pre-generated random bits that are used for any random operation, and the same bits are accessed for each unique random operation. By doing this, the probabilistic bounds are maintained, yet the output is consistent.

2.2 Runtime analysis

Let S denote the size of the connected component of v consisting only of *live* nodes. We consider the time of each individual step:

- Step 1 takes $d^{O(d \log d)}$ time.
- In step 2a, each node visits at most d neighbors and we must run sequential-LS to check whether each visited node is *live*, hence the runtime is $(S \cdot d) \cdot d^{O(d \log d)}$.
- Step 2b runs in $S \cdot d \log d$ time.
- Step 2c in constant time.

In total, the runtime is $S \cdot d^{O(d \log d)}$ (after absorbing factors into the exponent).

2.3 Bounding component size

Now, we claim that setting $k = O(d \log d)$, then with high probability, all connected components of live nodes will be of size $O(\text{polylog}(d) \cdot \log n)$. For each node v, consider the indicator variable

$$A_v = \begin{cases} 1 & \text{if } v \text{ survives all rounds} \\ 0 & \text{otherwise} \end{cases}$$

We are interested in components of nodes where $A_v = 1$. However, this is hard to work with since there is a lot of dependence between values of A and it depends on the round history. Thus, we consider a different indicator variable defined as follows:

$$B_v = \begin{cases} 1 & \text{if } \nexists \text{ a round such that } v \text{ colors itself red, and all neighbors color themselves blue} \\ 0 & \text{otherwise} \end{cases}$$

Notice that $A_v = 1 \implies B_v = 1$. Hence, it suffices to bound the size of connected components of nodes where $B_v = 1$.

Lemma 6 *Let* $\text{dist}(u, y)$ *be the shortest distance between nodes* u *and* y*. If* $\text{dist}(u, y) \geq 3$*, then* B_u *and* B_y *are independent.*

Proof For a node u, the event B_u depends only on the randomness of itself and its neighbors $N(u)$. For two nodes u, v where $\text{dist}(u, v) \geq 3$, $\{u\} \cup N(u)$ and $\{v\} \cup N(v)$ are disjoint. ■

Lemma 7 $\Pr[B_v = 1] \leq \frac{1}{8d^3}$

Proof Making use of Lemma 4, we have: $\Pr[B_v = 1] \leq (1 - \frac{1}{4d})^{c \cdot d \log d} \leq \frac{1}{8d^3}$ if we set $c \geq 20$. ■

We now outline the idea of the proof. If we had a large connected component of live nodes ($B_v = 1$), then we would be able to find a large subset of nodes of pairwise distance at least 3, making all the individual B_v's independent. We can then apply union bound to show that the probability that all of the B_v's are 1 is very low.

Let H denote a graph on the nodes of G, where we have an edge in H between nodes u and v if $\text{dist}_G(u, v) = 3$. Note that H has max degree d^3.

Lemma 8 *If there exists a surviving connected component S of size $\geq wd^3$ in G, then there must exist a surviving connected component T of size $\geq w$ in H.*

Proof We construct this set T greedily. We first pick an arbitrary node $u \in S$. While there are still nodes in S:

- Add u to T.
- Remove from S all nodes v such that $\text{dist}_G(u, v) \leq 2$ (including u).
- Pick a new node u such that $\text{dist}(u, v) = 3$ for some $v \in T$.

Note that at most $d + d^2 < d^3 - 1$ (for $d > 1$) nodes are removed for every node added to T, and T must be connected in H. Hence we have $|T| \geq \frac{|S|}{d^3} \geq w$.
■

Note that after $k = O(d \log d)$ rounds of Luby's algorithm, the probability that a component of size w survives in H is $\leq \left(\frac{1}{8d^3}\right)^w$ (since by Lemma 6, B_v's in H are independent and $\Pr[B_v = 1] \leq \frac{1}{8d^3}$).

To bound the probability of size-wd^3 components in G, it suffices to bound the probability of size-w components in H. Notice that a size-w component can be defined by one of its spanning trees, and hence the number of possible size-w components is upper-bounded by the number of size-w subtrees.

2.4 Counting subtrees

We want to bound the number of subtrees to perform our union bound. We first begin with a known fact bounding the number of possible unique tree shapes.

Theorem 9 *The number of non-isomorphic rooted trees of w nodes $\leq 4^w$.*

Proof Consider the sequence of nodes visited in a DFS from the root, where each step either goes down or up one level of the tree, with a total of $2(w-1)$ steps. We can encode this as a length $2(w-1)$ bit string, and hence $\#\text{trees} \leq 2^{2(w-1)} \leq 4^w$. ■

Corollary 10 *The number of size-w trees in a graph G with N nodes and max degree D graph is $\leq N(4D)^w$.*

Proof Suppose we try to enumerate all trees (double-counting is fine since we want an upper bound). We choose a node in G to be the root (N choices), choose a size-w rooted tree shape ($\leq 4^w$ choices) and choose a placement of the tree on G. For the placement, consider starting with the root node and adding a child node in the tree one by one. For each node added, we have at most D choices of which node in G to assign to it. Hence, we have

$$\text{Number of choices} \leq N \cdot 4^w \cdot D^w = N(4D)^w$$

■

2.5 Final result

Now taking union bound over each possible connected component (c.c.), and using the fact that H has max degree d^3, we have:

$$\Pr[\exists \text{ c.c. of size } w \text{ surviving in } H] \leq (\#\text{ size-}w\text{ c.c. in } H) \cdot \Pr[\text{size } w \text{ c.c. survives}]$$

$$\leq (\#\text{ size-}w\text{ subtrees in } H) \cdot \left(\frac{1}{8d^3}\right)^w \leq n \cdot (4d^3)^w \cdot \left(\frac{1}{8d^3}\right)^w = \frac{n}{2^w} \leq \frac{1}{n}$$

for $w = \Omega(\log n)$. Thus,

$$\Pr[\exists \text{ surviving component of size } \Omega(d^3 \log n) \text{ in } G] \leq \frac{1}{n}$$

and hence with high probability, our algorithm runs in $d^{O(d \log d)} \cdot \log n$ time.