Advanced Distributed Algorithms **Spring 2019**

Lecture 7: June 06

Lecturer: Merav Parter

Improved Distributed LLL Algoritms [CPS14]

Recall that we consider the LLL setting in LOCAL model of distributed computing. Given is a dependency graph $G = (\mathcal{A}, E)$ along with discrete independent random variables $\mathcal{X}$ where $vbl(A_i) \subseteq \mathcal{X}$. Each event $A_i \in \mathcal{A}$ is associated with a vertex (processor) v_i, where v_i knows the distribution of the discrete random variables $vbl(A_i)$. Since each vertex is associated with an event, we sometime interchange the terminology and might refer to an event A_i by the vertex v_i and vice-versa. The goal is to find an assignment to all variables in $\mathcal{X}$ that is consistent among all nodes[1], and such that no bad event occurs.

In the previous class, we showed the distributed MT algorithm. In this algorithm, in each step an MIS M of the currently violated bad events is computed, and all the variables of this set M get resampled. The somewhat unfortunate property of this algorithm is that nodes waste a lot of time in computing the MIS, rather than in finding the desired satisfying assignment. This was the motivation for Chung, Pettie and Su [CPS14] for coming up with a new LLL algorithm that avoids the MIS computation. As we will see, this comes at the cost of requiring a stronger LLL condition.

Lemma 7.1 *[CPS14][Improved Randomized LLL] Consider an LLL instance $G = (\mathcal{A}, E)$ with n events and maximum degree d that satisfies that $e \cdot p \cdot d^2 < 1$. Then, there is a randomized distributed algorithm for solving this LLL instance within $O(\log_{1/(e \cdot p \cdot d^2)} n)$ rounds, with high probability.*

Improved Distributed LLL Algorithm:

- (1) Initialize all variables with a random assignment.
- (2) While bad event occurs do:
  - Let B be the collection of all bad events that currently hold.
  - Let $I = \{u \in B \mid ID(u) < ID(v),\ \forall v \in B \cap N(u)\}$.
  - Resample all random variables in $\bigcup_{A_i \in I} vbl(A_i)$.

The key observation for bounding the resampling steps of the algorithm is the following:

Observation 7.2 *Let I_j be the independent set that got resampled in the j^{th} step. Then for every $u \in I_j$ it holds that $N_2^+(u) \cap I_{j-1} \neq \emptyset$ where $N_2^+(u)$ contains the 2-hop neighbors of u including itself.*

Proof: The easy case is when u is good in step $(j-1)$, since by the fact that it is bad in step j we deduce that one of its neighbors in $N(u)$ got resampled in step $j-1$. Hence, assume from now on that u is bad in step $j-1$. If $u \in I_{j-1}$, then we are done. Otherwise, we have that $u \in I_j \setminus I_{j-1}$. This implies that u has at least one neighbor v that was bad in step $j-1$ but good in step j. To see that observe that since u did not join I_{j-1}, it had a neighbor v with a strictly lower ID that was bad in step $j-1$. Note that v is not necessarily in I_{j-1}, since it might be the case that it had another bad neighbor w of lower ID. Since u did join I_j we know that v is no longer in the bad guys of this step. As v got "fixed" we can conclude that either v or one of its neighbors got resampled in step $j-1$, the claim follows. ■

Note that in the distributed MT algorithm, since in every step, we resample a collection of MIS events, we had the guarantee that $u \in M_j$ has a *neighbor* in M_{j-1}. Here, since the resampled set is independent but

[1] All nodes that share variables agree on the assignment of their shared variables.

not necessarily *maximal*, the guarantee is weaker: $u \in I_j$ has a 2-hop neighbor in I_{j-1}. As will we see, this weaker guarantee is compensated by requiring a stronger LLL inequality. To bound the running time of the algorithm, we will again use the notion of witness tree with a slight modification. Recall that in the distributed MT algorithm, we showed that the witness tree of node $u \in M_j$ must be *high*, of depth $j-1$, and since the probability of having large trees is small, the probability of having a large running time is small. Here, we would like to make the same kind of arguments, and to make sure that the tree of $u \in I_j$ is large, we will build 2-*witness trees* defined as follows.

The 2-Witness Tree. Consider a log of the distributed LLL algorithm specified by an ordered list of events, where for every $j < j'$ the events of I_j appears strictly before the events for $I_{j'}$ in this ordering. As before, the internal ordering inside each independent set I_j is arbitrary, and the output ordering can also be legit for the centralized MT algorithm. For each event $A_{i,j}$ in the ordering, we build a 2-witness tree rooted at $A_{i,j}$ in an almost exact same manner as before, only that we connect events in the tree if they are at most 2-hop away rather than 1-hop away as in the standard algorithm. For each time step $j \in \{1, \ldots, \ell\}$, we define a witness tree T_j as follows:

The 2-witness tree T_j for the j^{th} sampled event:

- (1) The root of T_j is $A_{i,j}$.
- (2) Traverse $A \in \{A_{i,j-1}, \ldots, A_{i,1}\}$ (i.e., in the reverse sampling order):
  - If A has a vertex in $N_2^+(A)$ (based on the dependency graph G) in the current tree T_j, add it as a child of the deepest such vertex (breaking ties based on IDs).

For an example, see Fig. 7.1. Using the modified variant of 2-witness tree, the argument is now very similar

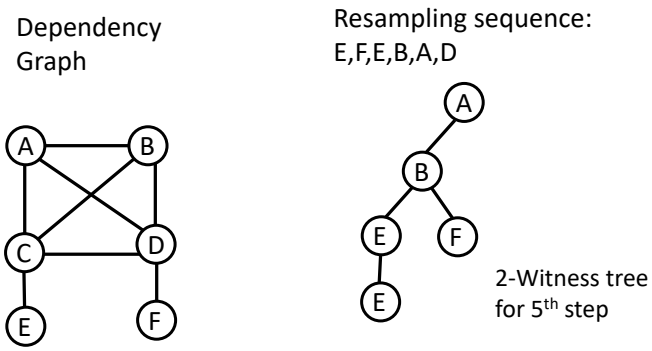

Figure 7.1: Illustration of the 2-witness trees in which nodes are connected to the deepest 2-hop neighbor.

to that of the distributed MT algorithm from last week.

Lemma 7.3 *The 2-witness tree of every node $u \in I_j$ (i.e., resampled in the j^{th} step) has depth at least $j-1$.*

Proof: Shown again by induction on j. The base of the induction $j = 1$ holds vacuously. Now, assume that the claim holds up to $j-1$ and consider $u \in I_j$. By Observation 7.2, u is connected to at least one node in $v \in N_2^+(u) \cap I_{j-1}$. The claim follows by combining with the induction assumption for $j-1$. ■

Finally, we show that large trees of size $\Omega(\log_{1/(ped^2)} n)$ do not appear throughout the execution with high probability.

Lemma 7.4 *W.h.p., there is no tree of size at least $c \cdot \log_{1/(ped^2)} n$.*

Proof: By adapting the argument from last week, one can show that there are most $n\binom{sd^2}{s-1}$ trees of size s. This is because each tree with a given root can be uniquely defined using a string of $s \cdot d^2$ bits, with exactly $s-1$ ones. In addition, by the exact same proof as from last week, each s-size tree occurs with probability at most p^s. Thus, there are $n\binom{sd^2}{s-1} \cdot p^s$ trees of size s in expectation. By plugging $s = c \cdot \log_{1/(ped^2)} n$ for some constant c, we get there are at most $1/n^{c-1}$ such trees in expectation. The proof follows by the Markov inequality. ■

Lemma 7.1 follows by combining Lemma 7.3 with Lemma 7.4.

The current state-of-the-art randomized complexity of LLL is $2^{O(\sqrt{\log \log n})}$ rounds, provided that $p \cdot e \cdot d^{32} < 1$ due to [FG17]. In addition, Brandt et al. [BFH+16] showed a lower bound of $\Omega(\log \log n)$ even if the LLL instance satisfies that $p \cdot e \cdot 2^d < 1$.

Application: Defective Coloring

We now demonstrate the power of the LLL setting for solving concrete graph problems efficiently. In the f-defective graph coloring, each node is given a palette of $\lceil 2\Delta/f \rceil$ colors $[1, \lceil 2\Delta/f \rceil]$, where Δ is the maximum degree in the graph. It is then required to color the vertices such that every vertex v with color $c(v)$ has at most f neighbors with the same color $c(v)$. The f-defective coloring is a natural generalization of the standard vertex-coloring problem, the latter is a 0-defective coloring. Throughout, for simplicity we assume that $f \geq 60 \ln \Delta$, this assumption can be removed as shown in [CPS14]. As an immediate application of Lemma 7.1, we show:

Lemma 7.5 *[Distributed Defective coloring] Given an n-vertex graph with maximum degree Δ, and integer $f \geq 60 \ln \Delta$, one can compute f-defective coloring within $O(\log n/f)$ rounds w.h.p.*

We first cast the f-defective coloring as an LLL instance, and then show how to solve it over the communication graph G. The random variables $\mathcal{X} = \{X_1, \ldots, X_n\}$ corresponds to the colors of the vertices, where each X_i corresponds to the color of v_i and it is a uniformly sampled r.v. in $[1, \lceil 2\Delta/f \rceil]$. For every vertex v, let A_v denote the bad event where u has strictly more than f neighbors with its color.

Since each X_u is chosen uniformly at random in range of size $2\Delta/f$, in expectation, every vertex v has $f/2$ neighbors with the same color of v. By applying the Chernoff bound (as all X_i's are independent), we get that $\Pr[A_v] \leq e^{-f/6}$. We next consider the degree in the dependency graph of these events. Every event A_v is defined over the variables $vbl(A_v) = \{X_u \mid u \in N^+(v)\}$ where $N^+(v)$ contains v and its neighbors. Thus, every two events A_v and A_u are dependent on each other iff $\texttt{dist}_G(u, v) \leq 2$. We get that the degree of dependency graph $d \leq \Delta^2$. To be able to apply the improved LLL algorithm, it remains to consider the 2-hop LLL criterion. We have:

$$e \cdot p \cdot d^2 \leq e \cdot e^{-f/6} \cdot \Delta^4 \leq e^{-f/12} ,$$

by using the fact that $f \geq 60 \ln \Delta$. Our next goal is to use the algorithm of Lemma 7.1 on the graph G. Observe that the communication graph G for which we want to solve the f-defective coloring problem is not the *dependency* graph of the above mentioned LLL instance. However, it is easy to see that these two graphs are quite related, where in particular the dependency graph is simply the power graph $G^{(2)}$. Since every communication round on $G^{(2)}$ can be simulated using 2 communication rounds in G, we can apply the LLL algorithm of Lemma 7.1 and compute the f-defective coloring within $O(\log n/f)$ rounds with high probability. Lemma 7.5 follows.

Deterministic Distributed LLL Algorithm [FG17]

Finally, we conculde the LLL session with deterministic algorithms provided that the LLL condition is satisfied with a *polynomial* slack, i.e., that $p \cdot (e \cdot d)^\lambda < 1$ for some constant λ. Throughout, for ease of writing, we assume that the maximum degree in the dependency graph is $d - 1$. The deterministic LLL algorithm that we will see is also used as a key component in the state-of-the-art randomized LLL algorithm due to [FG17]. The latter has the well-known two phase structure: a randomized part that sets the values

for a subset of the variables, and a deterministic part that completes the assignment by using the notion of network-decomposition (see class 1-2).

Lemma 7.6 *[FG17][Deterministic LLL] Consider an LLL instance $G = (\mathcal{A}, E)$ that satisfies that $p \cdot (e \cdot d)^\lambda < 1$ for some constant λ. Then, there is a deterministic distributed algorithm for solving this LLL instance within $\lambda \cdot n^{1/\lambda} \cdot 2^{O(\sqrt{\log n})}$ rounds.*

Reminder: Network Decomposition. Roughly speaking, a (d, c) network decomposition is a partitioning of the graph vertices into c disjoint subsets (or blocks) $V_1, \ldots, V_c$, such that each connected component of the induced subgraph $G[V_i]$ has diameter at most d. Network decomposition is the main tool for solving LCL problems deterministically. We have:

Lemma 7.7 *For every $\lambda \in [1, \log n]$, there is a deterministic algorithm that computes a $(\lambda, n^{1/\lambda} \cdot \log n)$ network decomposition in $\lambda \cdot n^{1/\lambda} \cdot 2^{O(\sqrt{\log n})}$ rounds.*

We next show how to use network decomposition to solve the LLL instance. This is considerably more involved than the standard applications for solving MIS and coloring.

Deterministic Distributed LLL Algorithm:

- (1) Compute an $(\lambda, n^{1/\lambda})$ network decomposition on the power-graph $G^{(2)}$.
- (2) Let $\mathcal{A}_1, \ldots, \mathcal{A}_\lambda$ be the output blocks of events.
- (3) Iterate over the blocks one by one, where in step i do as follows for $\mathcal{A}_i$:
  - Let $\mathcal{X}_i$ be the set of random variables of events in $\mathcal{A}_i$ that are not yet assigned.
  - Locally compute an assignment to $\mathcal{X}_i$ such that

$$\Pr[A \mid \bigcup_{j}^{i} \mathcal{X}_j] < p \cdot (e \cdot d)^i \text{ , for every event } A \in \mathcal{A} \text{ .} \tag{7.1}$$

The running time is dominated by the computation of the network decomposition. Our main goal is in showing that in each phase i, there is an assignment to the variable of $\mathcal{X}_i$ that satisfies Eq. (7.1). This will be shown by induction on i. Clearly, it holds for the first set $\mathcal{X}_1$. Assume it holds for $i-1$ and consider step i. We will show the existence of such an assignment by defining a new LLL system over the variables in $\mathcal{X}_i$. For each event A in $\mathcal{A}$, let $B(A, i)$ be the bad event where upon setting the values to the variables of $\mathcal{X}_i$, Eq. (7.1) is not satisfied. In other words, $B(A, i)$ is a function that maps each assignment of $\mathcal{X}_i$ to $\{0, 1\}$, where 1 is given to all assignments that increase the probability that A occurs to at least $p \cdot (e \cdot d)^i$.

Lemma 7.8 $\Pr[B(A,i) \mid \bigcup_{j=1}^{i-1} \mathcal{X}_j] \leq \frac{p \cdot (e \cdot d)^{i-1}}{p(e \cdot d)^i} \leq \frac{1}{e \cdot d}$.

Proof: Fix an event $A \in \mathcal{A}$ and let $\mathcal{R}_i$ be the collection of all *bad* assignment $\bar{R}$ to the variables in $\mathcal{X}_i$ satisfying that $\Pr[A \mid \bigcup_j^{i-1} \mathcal{X}_j \text{ and } \mathcal{X}_i = \bar{R}] \geq p \cdot (e \cdot d)^i$. We then have that:

$$\begin{aligned}
p(ed)^{i-1} &> \Pr[B(A,i) \mid \bigcup_{j=1}^{i-1} \mathcal{X}_j] \\
&\geq \sum_{\bar{R} \in \mathcal{R}_i} \Pr[A \mid \bigcup_{j}^{i-1} \mathcal{X}_j \text{ and } \mathcal{X}_i = \bar{R}] \cdot \Pr[\mathcal{X}_i = \bar{R} \mid \bigcup_{j}^{i-1} \mathcal{X}_j] \\
&\geq p(ed)^i \cdot \sum_{\bar{R} \in \mathcal{R}_i} \Pr[\mathcal{X}_i = \bar{R} \mid \bigcup_{j}^{i-1} \mathcal{X}_j] \\
&\geq p(ed)^i \cdot \Pr[B(A,i) \mid \bigcup_{j=1}^{i-1} \mathcal{X}_j] \text{ ,}
\end{aligned}$$

where the first inequality follows by the induction assumption for step $i-1$, and third inequality follows by the definition of the bad assignments $\mathcal{R}_i$. The claim follows. ■

The New LLL System for Step i: The events are given by the collection $B(A_1, i), \ldots, B(A_n, i)$ and the random variables $\mathcal{X}' = \mathcal{X}_i$. By Lemma 7.8, each bad event occurs with probability at most $p < 1/(ed)$. Since the dependency degree is d, we get that $p \cdot d \cdot e < 1$, therefore there exists an assignment to the variables of $\mathcal{X}_i$ that satisfies Eq. (7.1) for all $A \in \mathcal{A}$. To compute this assignment, the leader of every connected component in $G[\mathcal{A}_i]$ collects all its component graph information along with the information of the neighbors of the component. With this information, the leader can compute the assignment to the variables of $\mathcal{X}_i$ that satisfies Eq. (7.1) for all $A \in \mathcal{A}$. The values of this assignment are then sent to all the vertices in the component and to the neighbors of this component.

Note that since we compute a network decomposition in G^2 rather than in G, every vertex can have a neighboring vertex in at most *one* connected component of $G[\mathcal{A}_i]$, thus there will be at most one component that affects the "life" of a given vertex. Since every component picks the assignment that is good for all the events, the neighbors of the components are guaranteed to hold w.p at most $p \cdot (e \cdot d)^i$ at the end of this phase.

After λ phases all variables are set and Eq. (7.1) holds. Since $\Pr[A \mid \bigcup_{i=1}^{\lambda}] < p(ed)^{\lambda} < 1$ and since $\Pr[A \mid \bigcup_{i=1}^{\lambda}] \in \{0, 1\}$, we conclude that no bad event A occurs.

References

[BFH+16] Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela, and Jara Uitto. A lower bound for the distributed lovász local lemma. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 479–488. ACM, 2016.

[CPS14] Kai-Min Chung, Seth Pettie, and Hsin-Hao Su. Distributed algorithms for the lovász local lemma and graph coloring. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 134–143. ACM, 2014.

[FG17] Manuela Fischer and Mohsen Ghaffari. Sublogarithmic distributed algorithms for lovász local lemma, and the complexity hierarchy. In *31st International Symposium on Distributed Computing (DISC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.