**Advanced Distributed Algorithms** **Spring 2019**

# Lecture 6: May 30

*Lecturer: Merav Parter*

The next two classes are devoted to the Lovász Local Lemma and its incarnation in distributed algorithms. We will mainly consider the *symmetric* version of the Lovász Local Lemma due to its simplicity. The asymmetric version is stronger and can capture more problems as will be illustrated in the home exercise.

## The Lovász Local Lemma (Erdős and Lovász 1975)

Let $A_1, \ldots, A_n$ be a collection of (bad) events satisfying that (1) each event $A_i$ depends on at most $d$ other events, and (2) $\Pr[A_i] \leq p$ for every $i$. Then if $e \cdot p \cdot (d+1) < 1$, the probability that no bad event occurs is positive, that is $\Pr[\wedge \bar{A}_i] > 0$.

**Example 1: 2-Coloring of Hyper-graphs.** For a given integer $k \geq 2$, we are given a $k$-uniform $n$-vertex hyper-graph $G = (V, E)$ with $m$ edges $e_1, \ldots, e_m$, where each $e_i \subset V$ has exactly $k$ vertices. In addition, each edge $e_i$ intersects with at most $d$ other edges. Then, if $e(d+1) < 2^{k-1}$, there exists a 2-coloring of the vertices such that no edge is monochromatic. To see this, for every edge $e_i$, let $A_i$ denote the bad event where the edge $e_i$ is monochromatic. When coloring the vertices uniformly at random with two colors, we get that $\Pr[A_i] = 1/2^{k-1}$. Since $1/2^{k-1} \cdot e \cdot (d+1) < 1$, the LLL implies that such a coloring exists.

**Example 2: Satisfiability of $k$-CNF Formulas.** Any $k$-CNF formula in which each variable appears in $< 2^k/(ke)$ clauses is satisfiable. The proof again follows by considering the random assignment and denoting by $A_i$ the event where the $i^{th}$ clause is not satisfible. We have that $\Pr[A_i] \leq 1/2^k$. In addition, each clause depends on at most $k(2^k/(ke) - 1)$ other clauses. The claim follows by plugging this in the LLL formula.

The LLL as is already has a wide range of applications. However, one limitation of the lemma is that it only guarantees the existence of a good solution, but does not tell us how to find it. A-priori, it might be the case that finding the satisfying assignment for the $k$-CNF formula for example would require an exponential time. For that purpose, we will now consider the constructive LLL setting.

## The Constructive LLL [MT10]

The setting for the constructive LLL is as follows. Given is a collection of discrete and independent random variables $\mathcal{X} = \{X_1, \ldots, X_m\}$, a collection of (bad) events $\mathcal{A} = \{A_1, \ldots, A_n\}$ where $vbl(A_i)$ denote the set of variables on which $A_i$ is defined. Formally, each $A_i$ is a function that maps each assignment to $vbl(A_i)$ to a bit in $\{0, 1\}$. Two events $A_i, A_j$ are *dependent* of each other, if $vbl(A_i) \cap vbl(A_j) \neq \emptyset$. This defines the dependency graph $G = (\mathcal{A}, \{(A_i, A_j) \mid vbl(A_i) \cap vbl(A_j) \neq \emptyset\})$. Let $p$ be an upper bound on the probability that each event $A_i$ occurs. Then the dependency graph $G$ satisfies the LLL condition if $e \cdot p \cdot (d+1) < 1$ where $d$ is the maximum degree of $G$.

**Goal:** Find an assignment to all variables $X_1, \ldots, X_m$ such that no bad event occurs. In a breakthrough result, Moser and Tardos [MT10] presented a polynomial time algorithm for this task. The algorithm itself is very simple, and the key challenge is in bounding its running time.

**Moser-Tardos Algorithm (Centralized):**

- (1) Initialize all variables with a random assignment.
- (2) While bad event occurs do:
  - pick an arbitrary bad event $A_i$ and resample all its variables.

**Witness Tree.** The analysis of the MT algorithm is based on the notion of witness tree. Roughly speaking, witness trees provide a graphical representation for the log of the MT execution. Each step in the execution will be identified with a unique witness tree and thus bounding the running time will boil down into bounding the expected number of witness trees that arise throughout an execution. Consider a fixed execution of the MT algorithm and let $A_{i,1}, \ldots, A_{i,\ell}$ be the collection of events resampled in steps $1, \ldots, \ell$ of the algorithm. That is, in the first step of the algorithm, $A_{i,1}$ was resampled, then $A_{i,2}$ etc., ending with the last resampled event $A_{i,\ell}$. Note that $A_{i,j}$ might not be unique, and the same event might get resampled many times (i.e., it might be that $A_{i,3} = A_{i,5}$ and so on). For each time step $j \in \{1, \ldots, \ell\}$, we define a witness tree $T_j$ as follows:

**The witness tree $T_j$ for the $j^{th}$ resampling step:**

- (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 neighbor[a] (based on the dependency graph $G$) in the current tree $T_j$, add it as a child of the deepest such neighbor (breaking ties based on IDs).

[a] In the term neighbor we also include the event $A$ itself, i.e., all events at distance at most 1 from $A$ in the dependency graph $G$.

See Fig. 6.1 for an illustration.

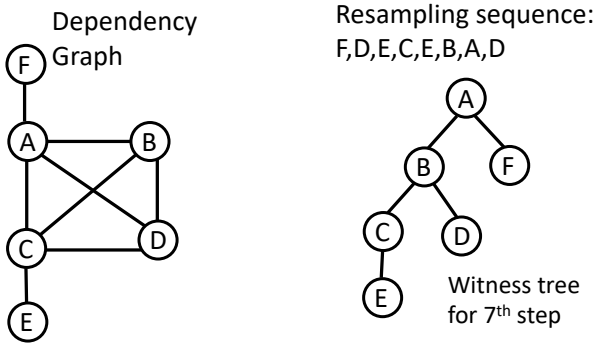


Figure 6.1: Illustration of computing the witness tree for a given log of MT-algorithm.

**Observation 6.1 (Uniqueness of trees)** *For every resampling steps $j' < j$, $T_{j'} \neq T_j$.*

**Proof:** Assume otherwise, then since the root of $T_{j'}, T_j$ is $A_{i,j'}, A_{i,j}$ (respectively), it must be that $A_{i,j} = A_{i,j'}$. This in turn implies that the number of occurrences of $A_{i,j}$ in $T_j$ must be strictly larger than in $T_{j'}$, leading to a contradiction. ■

Letting $\mathcal{T}_s$ be the collection of all possible witness trees of size $s$, we can state the following:

$$\mathsf{Exp}(\#\text{resampling steps}) = \sum_{s=1}^{\infty} \sum_{T_s \in \mathcal{T}_s} \Pr[T_s \text{ occurs during execution}] \ . \tag{6.1}$$

To bound the expected number of trees, we will first upper bound the probability that a fixed tree of size $s$ appears as a witness tree during the execution. Then we will upper bound the number of all possible trees of size $s$ in $G$. Combining these two terms will give us the desired upper bound on the number of resampling steps in the MT algorithm. The next observation follows immediately from the witness tree algorithm:

**Observation 6.2** *For every witness tree $T$, every two events $A_i, A_j$ in the same level are independent.*

The key lemma shows that in fact all events in the witness tree are independent of each other (event if they are neighbors in the dependency graph $G$).

**Lemma 6.3** *Fix a rooted tree $T$ of size $s$. Then* $\Pr[T \text{ occurs through the execution }] \leq p^s$.

**Proof Sketch:** By Obs. 6.2, every two nodes in the same level are independent. Now consider two nodes $u, v$ in levels $a < b$. This implies that $u$ was resampled strictly after $b$. We then have that by the time that $u$ is considered to be resampled, $v$ has already been resampled. Therefore even if these nodes share variables, we use fresh randomness when computing the probability that $u$ will be resampled in level $a$. Overall, the probability that events are sampled in different time steps is completely independent, since the algorithm uses distinct sets of random coins for the shared random variables. This notion is sometimes formalized by the concept of (infinite) randomness tables.

**Lemma 6.4** *The total number of trees with of size $s$ is at most $n \cdot \binom{s(d+1)}{s-1}$.*

**Proof:** There are $n$ ways of choosing the root of the tree among the $n$ events. We will now bound the number of trees of size $s$ with a fixed root $A_{i,j}$ by $\binom{s(d+1)}{s-1}$ which will conclude the proof. Each rooted tree of size $s$ can be represented by a binary string of length $s(d+1)$ as follows: order the $d+1$ neighbors of each event lexicographically. Then in the first chunk of $d+1$ bits, specify the neighbors of the root (i.e., the vertices in level-2 in the tree). Then allocate $d+1$ bits for each level-2 vertex and again use $d+1$ bits to specify their children in the tree. It is easy to see that in this way we can encode the entire tree in a unique manner using $s(d+1)$ bits. Since there is a total of $s-1$ nodes in the tree, not including the root, and since each vertex is marked only in the $(d+1)$ length bit string of its parent in the tree, there are at most $s-1$ ones in this string. Overall, there are at most $\binom{s(d+1)}{s-1}$ possible strings. ■

**Putting it all together:** By combining Eq. (6.1) with Lemma 6.3 and Lemma 6.4, get that:

$$\begin{aligned}
\mathsf{Exp}(\#\text{resampling steps}) &\leq \sum_{s=1}^{\infty} n \cdot \binom{s(d+1)}{s-1} \cdot p^s \\
&\leq n \cdot \sum_{s=1}^{\infty} \binom{s(d+1)}{s} \cdot p^s \leq n \sum_{s=1}^{\infty} (e(d+1))^s \cdot p^s \\
&\leq n \cdot \sum_{s=1}^{\infty} (pe(d+1))^s = O(n),
\end{aligned}$$

where the last inequality follows by assuming that $p \cdot e(d+1) \leq 1 - \epsilon$ for some constant $\epsilon$. This completes the description of the centralized LLL algorithm. We now turn to consider distributed solutions provided that the dependency graph is the communication graph.

## Parallel / Distributed LLL

The original paper by Moser and Tardos [MT10] also provides an efficient algorithm for the distributed or parallel setting. In the distributed setting, we consider the LOCAL model. Given is the dependency graph $G = (\mathcal{A}, E)$ along with discrete independent random variables $\mathcal{X}$ where $vbl(A_i) \subseteq \mathcal{X}$. Each event $A_i$ 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 $v_i$ that holds it.

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.

At a first glance, the distributed LLL problem might sound somewhat artificial, however, as we will see it has a wide range of applications for solving several variants of coloring problems. Recently, it was also proven

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

to be a complete problem for sublogarithmic LCL problems, and used for "speeding up" local algorithms, see [CP19, FG17]. Moser and Tardos provided the following distributed algorithm for the problem:

**The Distributed Moser-Tardos Algorithm:**

- (1) Initialize all variables with a random assignment.
- (2) While bad event occurs do:
  - Let $B$ the collection of all bad events that currently hold.
  - Let $M = MIS(G[B])$ be an MIS in the induced graph $G[B]$.
  - Resample all random varibles in $\bigcup_{A_i \in M} vbl(A_i)$.

That is, in contrast to the centerlized algorithm, where in each step, only one bad event gets resampled, in the distributed setting, we resample a maximal collection of independent events. This step is of course safe as independent events share no variable in common. The correctness is again immediate, and the main challenge is in bounding the number of MIS computations.

**Lemma 6.5** *W.h.p., the algorithm applies $O(1/\epsilon \cdot \log n)$ calls to the MIS algorithm.*

The proof uses again the notion of witness trees. The witness tree is defined exactly as before. Let $M_1, \ldots, M_\ell$ be the collection of MIS computed in each step $1, \ldots, \ell$ of the algorithm. For the purpose of computing the witness tree, we sort all the sampled nodes in non-decreasing order of the step in which they got resampled. That is, the sorting inside each independent set $M_i$ is arbitrary, and events in $M_j$ for $j < i$ appear strictly before the events of $M_i$ in this ordering. Note that this ordering of the events can be indeed a legit ordering for *centralized* MT algorithm. This is because in the latter, the resampling of bad events is arbitrary and a resampling of an event $A_i \in M_j$ cannot fix any other event $A_{i'} \in M_j$.

The key observation is that sampling the events in this specific form (i.e., as a collection of independent sets) introduces some structure, and as a result we can provide a stronger characterization for the individual witness trees.

**Lemma 6.6** *The witness tree of every node $u \in M_i$ (i.e., resampled in the $i^{th}$ step) has depth exactly $i-1$.*

**Proof:** By induction on $i$. The base of the induction $i = 1$ holds vacuously. Now, assume that the claim holds for $j \leq i-1$ and consider $u \in M_i$. Since all events in $M_i$ are independent, in its witness tree, $u$ has no neighbor in $M_i \setminus \{u\}$. We next claim that it must be connected to at least one vertex $v \in M_{i-1}$. In other words, we show that $M_{i-1} \cap N^+(u) \neq \emptyset$ where $N^+(u)$ contains all neighbors of $u$ in $G$ including itself. To show this, we consider two cases. First, assume that $u$ was not bad in step $i-1$ (i.e., the bad event associated with node $u$ does not hold). This implies that $u$ must had a neighbor that got resampled in step $i-1$ which made $u$ bad in the subsequent step. Next, assume that $u$ was bad in step $i-1$. Thus $u$ was a candidate to join the MIS. If $u \in M_{i-1}$, we are done, and otherwise it must have a neighbor in $M_{i-1}$.

Hence, there must be $v \in M_{i-1}$ that is connected as a child of $u$ in its witness tree. By induction assumption the depth of the witness tree of $v$ is $i-2$, thus the depth of the tree of $u$ is $i-1$. ■

We complete the argument by showing that w.h.p. there is no tree of depth $O(1/\epsilon \cdot \log n)$ provided that the LLL condition holds with an $\epsilon$-slack, that is that $e \cdot p \cdot (d+1) \leq 1 - \epsilon$.

**Lemma 6.7** *W.h.p. there is no tree $T_u$ of depth $O(1/\epsilon \cdot \log n)$.*

**Proof:** There are at most $n\binom{s(d+1)}{s-1}$ trees of size $s$, each occurs with probability at most $p^s$. Thus, there are $n\binom{s(d+1)}{s-1} \cdot p^s$ trees of size $s$ in expectation. By plugging $s = c \cdot 1/\epsilon \cdot \log n$ for some constant $c$, and using the fact that $e \cdot p \cdot (d+1) \leq 1 - \epsilon$, we get there are at most $1/n^{c-1}$ such trees. The proof follows by the Markov inequality. ■

Lemma 6.5 follows by combining Lemma 6.6 with Lemma 6.7. In the next class, we will see an improved distributed LLL algorithm due to [CPS14], that avoids the MIS computation all together!

## References

[CP19] Yi-Jun Chang and Seth Pettie. A time hierarchy theorem for the local model. *SIAM Journal on Computing*, 48(1):33–69, 2019.

[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.

[MT10] Robin A Moser and Gábor Tardos. A constructive proof of the general lovász local lemma. *Journal of the ACM (JACM)*, 57(2):11, 2010.