6.842 Randomness and Computation February 10, 2014

Lecture 2

Lecturer: Ronitt Rubinfeld *Scribe: Pasin Manurangsi*

1 Review

In lecture one, we showed how to prove the existence of combinatorial objects by proving that the probability that at least one of the bad events occur is less than one. We explored two bounds for this purpose. The first bound was the union bound stated as follows.

Theorem 1 *Given events $A_1, \ldots, A_n$, we have*

$$\Pr[\bigcap_{i=1}^{n} \overline{A_i}] \geq 1 - \sum_{i=1}^{n} \Pr[A_i].$$

While the union bound does not need any assumption on the independence of $A_1, \ldots, A_n$ at all, it is a very weak bound, i.e., one needs most of $\Pr[A_i]$'s to be no more than $\frac{1}{n}$ for the bound to be useful. In the previous lecture, we also took a look at another bound which is an opposite of the union bound. In this bound, a strong assumption is made and the bound is strong too. The bound is stated below.

Theorem 2 *Given events $A_1, \ldots, A_n$. If they are all independents, then*

$$\Pr[\bigcap_{i=1}^{n} \overline{A_i}] = \prod_{i=1}^{n} (1 - \Pr[A_i]).$$

In contrast to the union bound, this bound yields existence of good events requiring only that $\Pr[A_i]$'s are not one. It does, however, make a very strong assumption that all A_i's are independent.

A natural question to ask is whether we can get a better bound than the union bound when only some of the events are independent. In this lecture, we will study such bound named **Lovász Local Lemma**.

2 The Lovász Local Lemma

Before we proceed to the lemma, let us first recall the definition of independent

Definition 3 *An event A is* **independent** *of events $B_1, \ldots, B_n$ if for all nonzero subsets $J \subseteq [n]$,*

$$\Pr[A \cap \left(\bigcap_{j \in J} B_j \right)] = \Pr[A] \cdot \Pr[\bigcap_{j \in J} B_j].$$

Note that, in our notation, independent is not the same as pairwise independent. Generally speaking, our notation of independent is stronger.

Next, with our notion of independent, we will define a dependency digraph of events.

Definition 4 *Let $A_1, \ldots, A_n$ be events. An undirected graph $D = (V, E)$ with $V = [n]$ is a* ***dependency digraph*** *of $A_1, \ldots, A_n$ if each A_i is independent of the set of all A_i's that do not neighbor it in D.*

With notation of dependency graph in place, we now will state the symmetric version of the Lovász Local Lemma.

Theorem 5 (Lovász Local Lemma) *Let $A_1, \ldots, A_n$ be events such that $\Pr[A_i] \leq p$ for all $i = 1, \ldots, n$ and let D be the dependency digraph of $A_1, \ldots, A_n$. Suppose that each vertex in D is of degree at most d. If $ep(d+1) \leq 1$, then $\Pr[\bigcap_{i=1}^{n} \overline{A_i}] > 0$. [EL75] [Spe77]*

It is worth pointing out that, the condition necessary for the Lovász Local Lemma to yeild $\Pr[\bigcap_{i=1}^{n} \overline{A_i}] > 0$ is $p \leq \frac{1}{e(d+1)} = \Theta(\frac{1}{d})$, which, unlike the union bound that needs $p < \frac{1}{n}$, does not depend on n.

3 Applications

We will not show the proof of the Lovász Local Lemma but we will jump right ahead to some of its applications.

3.1 Two-Coloring Revisited

With the help of the Lovász Local Lemma, we can achieve the following result for the two-coloring problem discussed last lecture.

Theorem 6 *Given $S_1, \ldots, S_m \subseteq S$ such that $|S_i| = l$. Suppose that each S_i intersects at most d other S_i's. If $e(d+1) \leq 2^{l-1}$, then we can two-color S such that each S_i is not monochromatic.*

Note that, with the union bound, we were only able to prove the existence of two-coloring when $m < 2^{l-1}$. As a result, the theorem above covers many more cases.

Proof Color each element of S independently with red or blue each with probability $\frac{1}{2}$. Define A_i to be the event where S_i is monochromatic. We have

$$\begin{aligned}\Pr[A_i] &= \Pr[\text{all elements of } S_i \text{ are colored blue}] + \Pr[\text{all elements of } S_i \text{ are colored red}] \\ &= \frac{1}{2^l} + \frac{1}{2^l} \\ &= \frac{1}{2^{l-1}}.\end{aligned}$$

Observe that A_i is independent of all A_j's such that $S_i \cap S_j = \emptyset$. As a result, we can build a dependency digraph of $A_1, \ldots, A_n$ by connecting A_i to A_j if and only if $S_i \cap S_j \neq \emptyset$. In this digraph, degree of each vertex is at most d.

Thus, since we assume that $e(d+1) \leq 2^{l-1}$, from the Lovász Local Lemma, we can conclude that $\Pr[\bigcap_{i=1}^{n} \overline{A_i}] > 0$. Hence, there exists a two-coloring such that each S_i is not monochromatic. ■

3.2 Boolean Satisfiability Problem

The Lovász Local Lemma is useful for not only for two-coloring problem but for boolean satisfiability problem as well. Namely, if each variable does not appear in too many clauses, then there exists a satisfying assignment.

Theorem 7 *Given a conjunctive normal form formula such that each clause contains l variables and that each variable appears in at most k clauses. If $e(lk+1) \leq 2^{l-1}$, then the formula is satisfiable.*

We will not state the proof here but it is straightforward.

4 The Moser-Tardos Algorithm

While the Lovász Local Lemma can be used to prove existences of combinatorial objects, it does not give us a constructive way to find that object. In this section, we will survey the Moser-Tardos algorithm, which gives a constructive way to find a valid two-coloring.

4.1 History

Before we describe the Moser-Tardos algorithm, here is a brief timeline of significant developments from the Lovász Local Lemma to the Moser-Tardos algorithm. Note that the result below is rephrased in terms of the two-coloring problem.

- In 1975, Lovász and Erdõs proved the Lovász Local Lemma with bound $d+1 \leq 2^{l-2}$ [EL75]. Note that this is slightly weaker than the bound $d+1 \leq \frac{2^{l-1}}{e}$ we stated above; this better bound first appeared in [Spe77].
- In 1991, Beck showed, in [Bec91], a constructive algorithm to find a valid two-coloring given that $d+1 \leq 2^{\frac{l}{48}}$.
- In 2009, Moser found a constructive algorithm given that $d+1 \leq 2^{l-2}$ [Mos09]. It is later extended by Moser and Tardos to cover many more problems [MT10].

We will not show the original analysis by Moser and Tardos here. Instead, we will use a simplified analysis by Chandrasekaran, Goyal and Haeupler [CGH10], which is slightly weaker than the original.

4.2 The Moser-Tardos Algorithm

A symmetric version of the Moser-Tardos Algorithm is very simple. It can be describe as follows:

- Randomly assign a color to each element in S independently and uniformly.
- While there is still a set S_i that is monochromatic, randomly assign each element of S_i independently and uniformly.

Even though the algorithm looks very simple, its expected runtime is in fact $O(poly(m, l))$. We can state this formally as follows.

Theorem 8 *Given $S_1, \ldots, S_m \subseteq S$ such that $|S_i| = l$. Suppose that each S_i intersects at most d other S_i's. If $ce(d+1) \leq 2^{l-1}$ for some constant $c > 1$, then we can find a two-coloring of S such that each S_i is not monochromatic within $O(poly(m, l))$ time.*

We will devote the rest of this section to the proof of this theorem. We will start by defining the log of execution and witness trees.

Definition 9 *The **log of execution** is defined to be $(1, S_{i_1}), (2, S_{i_2}), \ldots$ where S_{i_j} is the set that gets resampled at the jth round of the algorithm.*

Figure 1 illustrates how the Moser-Tardos Algorithm works, together with its associated log of execution.

Definition 10 *A **witness tree for step** $j > 0$ is constructed as follows.*

1. *Start with a rooted tree with only root S_{i_j}*
2. *For each $t = j-1, j-2, \ldots,$ and 0:*

   if S_{i_t} intersects with at least one of the vertices in the current witness tree, add S_{i_t} to the witness tree by letting its parent be a vertex that intersects S_{i_t} with maximum depth. If there are more than one such vertices, just pick one arbitrarily.

In Figure 2, we show witness trees according to the Moser-Tardos Algorithm shown in Figure 1.

The algorithm's runtime depends on the log of execution's length. We will bound the expected value of this length by bounding the probability that each tree appears as a witness tree of the algorithm.

4.3 Bound on Probability of A Witness Tree

To bound the probability that a particular tree appears as a witness tree of some step j, let us define the τ-check procedure.

Definition 11 *The **τ-check** procedure is defined as follows:*

1. *Visit vertices of τ in reversed BFS order (i.e., maximum depth first)*
2. *Assign a color to each vertex in the corresponding set as done in the algorithm*
3. *Check whether the set is monochromatic*
4. *If all the sets tested are monochromatic, then the check is passed. Otherwise, the check is failed.*

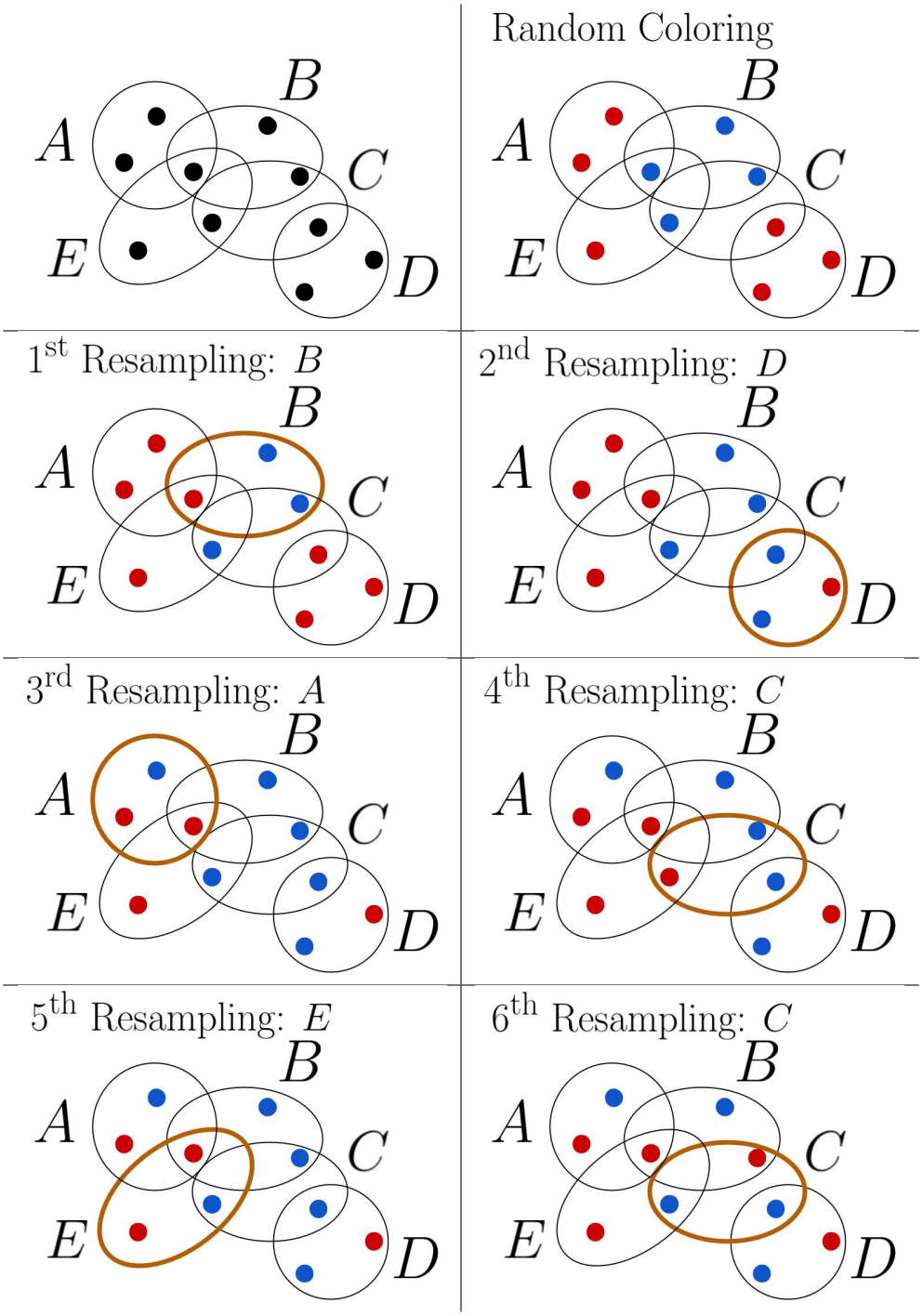

Figure 1: Illustration of The Moser-Tardos Algorithm. The set S consists of ten elements and we want to two-color S such that each of A, B, C, D and E is not monochromatic. We start with randomly colored elements as seen in the upper right picture. We then resample monochromatic sets until we find a valid two-coloring. In this case, the log of execution is $(1, B), (2, D), (3, A), (4, C), (5, E), (6, C)$.

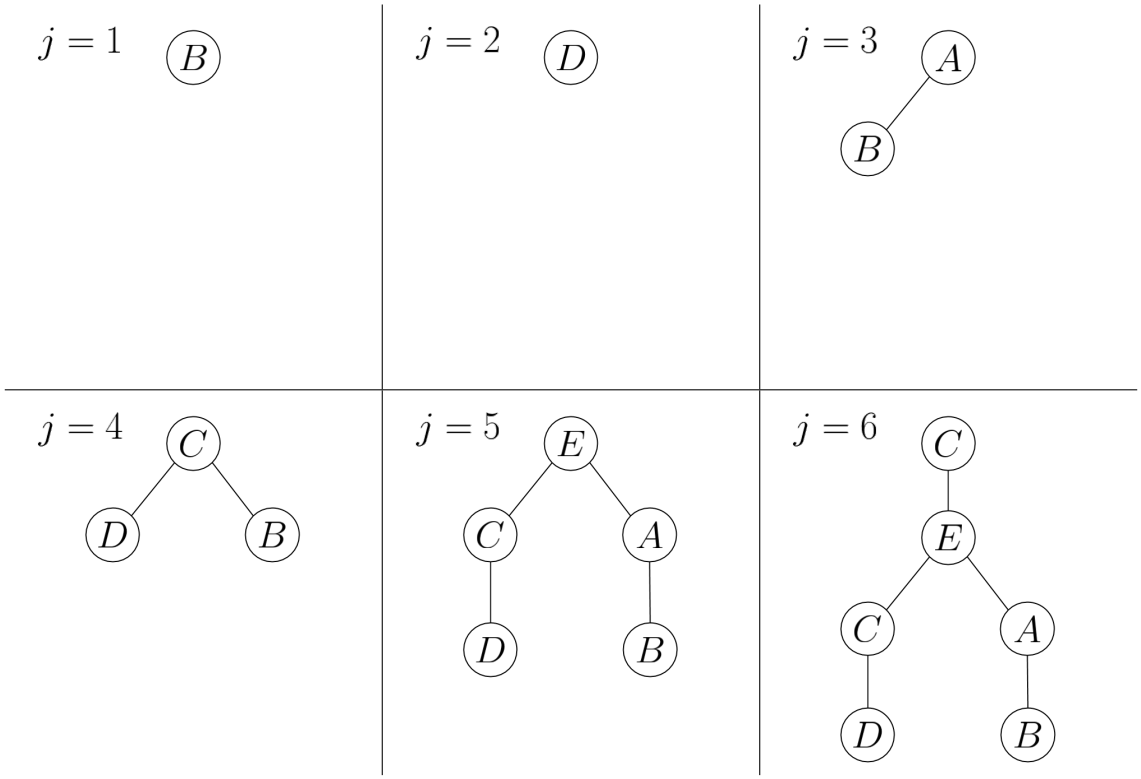

Figure 2: Illustration of Possible Witness Trees for $j = 1, \ldots, 6$ for The Moser-Tardos Algorithm in Figure 1. Note that a witness tree at each step does not need to be unique. For example, when $j = 5$, we can connect B to C instead of A and still get a witness tree.

From our definition of the log of execution and witness trees, we know that each set in a witness tree was resampled in the algorithm. This meant that it was monochromatic right before it was resampled. As a result, the witness trees will pass τ-check.

Moreover, observe that, first, if two sets have the same depth in a witness tree, they cannot share any element. This is because, if they shared an element, we would have assigned one of them to be the parent of the other in the construction of the witness tree.

Secondly, even if two sets of different depths might share elements, by the time that the τ-check checks for a set, all of the sets with higher depths got resampled. As a result, the event that each set is monochromatic when tested in τ-check is independent of each other. Thus, we can conclude that, for each tree τ,

$$\Pr[\tau\text{-check passes}] \leq \left(\frac{1}{2^{l-1}}\right)^{|\tau|}$$

where $|\tau|$ is the number of vertices of the tree.

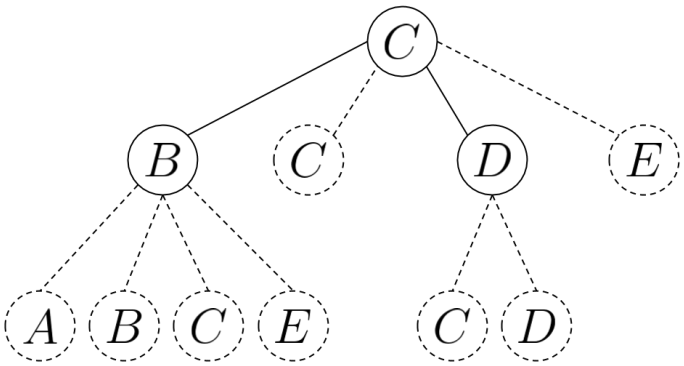

Figure 3: Illustration of Translation from a Witness Tree to a Binary String. Consider the same input from Figure 1. Suppose that we take our witness tree when $j = 4$ as illustrated in Figure 2. The graph here shows the witness tree in ordinary lines and shows other sets that intersect each vertex but not included in the witness tree with dashed lines. BFS visits C, B and D respectively. As a result, we will write out the binary string as 1010 0000 0000.

4.4 Bound on Number of Possible Witness Trees of Certain Size

Next, we will bound the number of possible withness trees of certain size with a specified root. The bound is stated and proved below.

Theorem 12 *For each $j \in [m]$, the number of possible witness trees of size s rooted at S_j is at most $\binom{s(d+1)}{s-1}$.*

Proof For each $i \in [m]$, let $S_i^{(1)}, S_i^{(2)}, \ldots, S_i^{(k_i)}$ be the sets that intersect with S_i including itself. From our hypothesis, we have $k_i \leq d + 1$ for all $i \in [m]$.

Now, let us consider any witness tree τ of size s. From our construction of witness trees, every parent and child of a tree must share an element, i.e., S_i's children must be from $S_i^{(1)}, S_i^{(2)}, \ldots, S_i^{(k_i)}$. Moreover, each vertex does not have repeated children because the construction would tell us to put one of the repeated children to be a child of another to maximize its depth.

Let $S_{\tau_1}, \ldots, S_{\tau_s}$ be the sequence of vertices visited by BFS. If a parent S_i has more than one children, always push the children in the order that they appear in $S_i^{(1)}, S_i^{(2)}, \ldots, S_i^{(k_i)}$.

Now, define a binary string $b_1 \ldots b_{s(d+1)}$ as follows.

$$b_{n(d+1)+r} = \begin{cases} 1 & \text{if } r \leq k_{\tau_{n+1}} \text{ and } S_{\tau_{n+1}}^{(r)} \text{ is a child of } S_{\tau_{n+1}} \text{ in } \tau, \\ 0 & \text{otherwise} \end{cases}$$

for all $n = 0, \ldots, s-1$ and $r = 1, \ldots, d$.

Basically, this binary string tells us which children of each node are included in τ. Figure 3 illustrates how to translate a witness tree to a binary string.

Clearly, this binary has exactly $(s-1)$ one's. Furthermore, it is easy to see that no two witness trees get mapped to the same binary string. As a result, the number of witness trees is at most the number

of binary string of length $s(d+1)$ with exactly $(s-1)$ one's, which is $\binom{s(d+1)}{s-1}$. ■

4.5 Bound on the Expected Length of the Log Execution

In this section, we will combine the results previously derived to prove a bound on the expected length of the log execution, which in turn gives us a bound on the algorithm's runtime.

From our definition of the log execution, we can conclude that

$$\text{length of log execution} = \sum_{\tau} (\text{number of times } \tau \text{ appears as a witness tree})$$

where the sum on the right runs over all possible rooted trees τ.

It is obvious from the construction that no witness tree appears twice in the tree. Thus, we have

$$\text{length of log execution} = \sum_{\tau} \mathbf{1}_{\tau}$$

where $\mathbf{1}_{\tau}$ denote an indicator variable whether τ appears as a witness tree or not. If τ is a witness tree, $\mathbf{1}_{\tau}$ is one. Otherwise, it is zero.

As a result, we have

$$\begin{aligned}
\mathbb{E}[\text{length of log execution}] &= \mathbb{E}\left[\sum_{\tau} \mathbf{1}_{\tau}\right] \\
&= \sum_{\tau} \mathbb{E}[\mathbf{1}_{\tau}] \\
&= \sum_{\tau} \Pr[\tau \text{ appears as a witness tree}] \\
&= \sum_{s=1}^{\infty} \sum_{i \in [m]} \sum_{\substack{|\tau|=s \\ \tau \text{ rooted at } S_i}} \Pr[\tau \text{ appears as a witness tree}] \\
\text{(from subsection 4.3)} &\leq \sum_{s=1}^{\infty} \sum_{i \in [m]} \sum_{\substack{|\tau|=s \\ \tau \text{ rooted at } S_i}} \left(\frac{1}{2^{l-1}}\right)^s \\
\text{(from subsection 4.4)} &\leq \sum_{s=1}^{\infty} \sum_{i \in [m]} \binom{s(d+1)}{s-1} \left(\frac{1}{2^{l-1}}\right)^s \\
&= m \sum_{s=1}^{\infty} \binom{s(d+1)}{s-1} \left(\frac{1}{2^{l-1}}\right)^s \\
\text{(Stirling's approximation)} &\leq m \sum_{s=1}^{\infty} (e(d+1))^s \left(\frac{1}{2^{l-1}}\right)^s \\
\text{(from our assumption)} &\leq m \sum_{s=1}^{\infty} \left(\frac{1}{c}\right)^s \\
&= O(m).
\end{aligned}$$

Recall that the length of the log execution equals the number of loops. Since the runtime in each loop is only $O(poly(m, l))$ and the expected length of the log execution is $O(m)$, we can conclude that the expected runtime of this algorithm is $O(poly(m, l))$ as desired.

References

[Bec91] J. Beck, An algorithmic approach to the Lovsz local lemma. In *I. Random Struct. Alg.*, 2, pages 343 - 365, 1991.

[CGH10] K. Chandrasekaran, N. Goyal and B. Haeupler, Deterministic Algorithms for the Lovász Local Lemma. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 992 - 1004, 2010.

[EL75] P. Erdõs and L. Lovász, Problems and results on 3-chromatic hypergraphs and some related questions. In *Infinite and Finite Sets. (Colloquia Math. Soc. Jnos Bolyai)*, 10, pages 609 - 627, 1975.

[Mos09] R. A. Moser, A Constructive Proof of the Lovász Local Lemma. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, pages 343 - 350, 2009.

[MT10] R. A. Moser and G. Tardos, A constructive proof of the general lovsz local lemma. In *J. ACM*, 57(2), pages 1 - 15, 2010.

[Spe77] J. Spencer, Asymptotic lower bounds for Ramsey functions. In *Discrete Mathematics*, 10, pages 69 - 76, 1977.