6.842 Randomness and Computation February 15, 2012

Lecture 3

Lecturer: Ronitt Rubinfeld *Scribe: Alan Guo*

Today we will give a constructive proof of the Lovász Local Lemma (LLL), i.e. an efficient algorithm for finding a hypergraph coloring. Note that the original proof was non-constructive. We will prove the statement about hypergraph coloring, noting that the same proof technique applies to the LLL. For convenience, we restate the Lemma.

Lemma 1 *Let $S_1, \ldots, S_m \subseteq S$ be sets such that each S_i intersects at most d other S_j, and $|S_i| = \ell$. If $ep(d+1) < 1$, where $p = 2^{-\ell-1}$, then there exists a 2-coloring of S such that no S_i is monochromatic.*

In this lecture, we prove a weaker version, which assumes $ep(d+1) < \frac{1}{2}$. The algorithm, due to Moser and Tardos, is as follows:

- Start with a random coloring of the set S
- While there is a monochromatic ("violated") set:
  - Pick an arbitrary violated S_i
  - Randomly reassign colors to elements of S_i

Clearly, if this algorithm terminates, it gives us a coloring with no monochromatic sets. The question is, then, does it terminate, and if so, after how long? To analyze this, we look at "logs of execution" and bound their length.

Definition 1 *A* log of execution *is a sequence of ordered pairs $(1, S_{i_1}), (2, S_{i_2}), \ldots$ where S_{i_j} is the set chosen to be resampled in step j of the algorithm above.*

Definition 2 *A* witness tree for step j *($j > 0$) is a rooted labeled tree constructed as follows:*

- *root vertex labeled by S_{i_j}*
- *For $t = j$ down to 1:*
  - *If S_{i_t} intersects any sets in the witness tree, then add S_{i_t} to the witness tree by pointing it to an arbitrary vertex of lowest level (max distance from root)*

The idea is to bound the expect length of the log of execution after the algorithm runs. If the expectation is finite, then there is some run which terminates in finite time, hence there is some coloring which results in no monochromatic sets. To bound the length of logs of execution, we will bound the number and size of witness trees.

1 Bounding Probability of a Specific Witness Tree

We will denote the witness tree by τ. Our goal is to define a procedure "τ-check" and then we will upper bound Pr[τ-check passes].

We now define τ-check:

1. Visit vertices of τ in reverse BFS order (max depth first)
2. For each vertex, recolor the corresponding set as done in the algorithm
3. Check that set is monochromatic
4. Pass if all checks violated (monochromatic)

There are two important points to note:

- If two sets are at the same level in τ, they cannot intersect, by construction
- If two sets are at different levels, then even though they may share elements, all shared elements get randomly recolored before the second set is evaluated by the algorithm

If τ appears in the log of execution, then it must pass the τ-check, hence

$$\Pr[\tau\text{-check passes}] \geq \Pr[\tau \text{ appears in log}].$$

What's the probability that τ-check passes? Well, the probability that a single set is monochromatic is $p \equiv 2^{-(\ell-1)}$. By the observations we made above, the event that each set in the witness tree is monochromatic is independent of the others, so

$$\Pr[\tau\text{-check passes}] = p^{|\tau|}$$

where $|\tau|$ denotes the number of vertices in τ.

2 Bounding number of witness trees of a given size

Since each witness tree (with a fixed root) is a rooted labeled subtree of a d-ary tree, it suffices to bound the number of such trees of size s. We give a one-to-one map from rooted labeled trees of size s with max degree d to the set of binary strings of length sd with $s-1$ ones. This will imply that the number of witness trees of size s is at most $\binom{sd}{s-1}$.

To map a rooted labeled tree with a fixed root to a binary string, perform the following algorithm: perform a DFS of the tree beginning with the root, and write a 1 for each child visited and a 0 for each child not there.

For example, suppose the sets are $A, B, C, D, E, F \subseteq S$, and $d = s = 3$. Suppose τ_1 has root A which has children B and C, suppose τ_2 has root A with child B, which has child E, and suppose τ_3 has root A with children B and D. The algorithm run on τ_1 runs as follows:

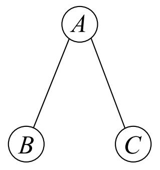

Figure 1: τ_1

1. Starting from A, visit B (write 1)
2. From B, no children (write 000, one zero per non-existing child)
3. Starting from A, visit C (write 1)
4. From C, no children (write 000)
5. From A, no third child (write 0)
6. The resulting string is **100010000**

The algorithm run on τ_2 runs as follows:

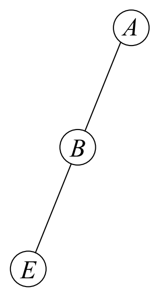

Figure 2: τ_2

1. Starting from A, visit B (write 1)
2. Starting from B, visit E (write 1)
3. From E, no children (write 000)
4. From B, no other children (write 00)
5. From A, no other children (write 00)
6. The resulting string is **110000000**

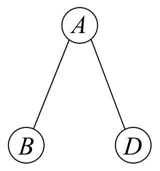

Figure 3: τ_3

The algorithm run on τ_3 runs as follows:

1. Starting from A, visit B (write 1)
2. From B, no children (write 000)
3. Starting from A, child C is missing (write 0)
4. Starting from A, visit D (write 1)
5. From D, no children (write 000)
6. The resulting string is **100001000**

In general, the string length is sd since there are s nodes, and d children per node. There are $s-1$ ones since, from the root, $s-1$ descendants will be visited. It is straightforward to see that distinct labeled trees map to distinct binary strings.

3 Bounding the expected length of log

We note that the expected length of the log of execution is equal to the expected number of witness trees that occur. Moreover, **no tree occurs twice**. Therefore

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

where 1_τ is an indicator variable which is 1 if τ occurs in the log, otherwise 0.

Then

$$\begin{aligned}
E[\text{length of log}] &= E\left[\sum_{\tau} 1_{\tau}\right] \\
&= \sum_{\tau} E[1_\tau] \\
&= \sum_{\text{roots } r} \sum_{\tau \text{ rooted at } r} E[1_\tau] \\
&= \sum_{r} \sum_{s=1}^{\infty} \sum_{|\tau| = s \text{ rooted at } r} E[1_\tau] \\
&= \sum_{r} \sum_{s=1}^{\infty} \sum_{|\tau| = s \text{ rooted at } r} p^s \\
&\leq m \sum_{s=1}^{\infty} \binom{sd}{s-1} p^s && \text{by Section 2} \\
&\leq m \sum_{s=1}^{\infty} ((d+1)ep)^s && \text{by Stirling's approximation} \\
&\leq m \sum_{s=1}^{\infty} \left(\frac{1}{2}\right)^s && \text{by assumption} \\
&= O(m).
\end{aligned}$$

In particular, the expected running time of the hypergraph coloring algorithm is polynomial, so there exists a sequence of colorings such that the algorithm terminates, hence there exists a coloring of S such that no S_i is monochromatic.