Lecture 2 : Lovász Local Lemma

Lecturer: Ronitt Rubinfeld, Scribe: Alessandre Santos

2.2.2022

1 Introduction

Last lecture, we learned that despite providing a proof for existence the probabilistic method **does not** provide a way to find the satisfying construction. However, in this lecture, we introduce the *Lovász Local Lemma(LLL)* and related algorithms that finds the diseried construction in this setting.

2 Lovász Local Lemma (LLL)

Question 1 *Given some "bad events" $A_1, ..., A_n$ that happen each with $\mathsf{Pr}[A_i] \leq p$ for some constant p for all i. Can we show that there is a non-zero probability that **none of the events** happen such that $\mathsf{Pr}(\cap_i \overline{A_i}) > 0$?*

Our first approach to the question could be to think of what different kinds of relationships could exists among these " bad events".

2.1 Case 1: all independent

If $A_i's$ are independent and non trivial $\mathsf{Pr}(A_i) \neq 1$ then we have the following relationship:

$$\mathsf{Pr}[\cup_i A_i] = 1 - \mathsf{Pr}(\cap_i \overline{A_i}) = 1 - \Pi_i \mathsf{Pr}_{>0}[\overline{A_i}] < 1 \tag{1}$$

where the first inequality comes from properties of the complement, the second inequality is just the definition of independence and the last inequality comes from the assumption that A_i's are non trivial.

2.2 Case 2: all dependent but bounded

Next, we could consider events that are all dependent among each other, but with the condition that $\sum_i \mathsf{Pr}[A_i] < 1$. Then, we have the following relationship:

Last time union bound.

$$Pr[\cup_i A_i] \leq \sum_i Pr[A_i] < 1 \tag{2}$$

In this case, the first inequality comes from union bound and the second inequality comes from the assumption on the probabilities.

2.3 Case 3

However, we can wonder if there is some intermediate case where the events are independent of some but not all events and we can bound their probabilities by some condition. Before attempting this however, it is necessary to define a condition of dependence.

Definition 2 *A is **independent** of a set of variables $B_1, ..., B_k$ if for all subsets J with $J \subseteq [k] = \{1, ..., k\}$ and $J \neq \emptyset$, we have the following relationship:*

$$\mathsf{Pr}[A \cap \bigcap_{i \in J} B_j] = \mathsf{Pr}[A] \cdot \mathsf{Pr}[\cap_{i \in J} B_j]$$

Definition 3 *Let $A_1, ..., A_n$ be events and construct the following **dependency digraph** $D = (V, E)$ such that $V = [n]$ and A_i independent of all A_j that are not neighbours in D.*

The definition of dependency digraph accounts for direction. Furthermore, this means that if there is an edge from A_i to A_j, A_i is dependent of A_j and the opposite is not necessarily true. However, the direction of the dependence will not be used in this lecture. Now we can state the full statement for the Lovász Local Lemma.

Theorem 4 *Let $A_i, ...A_n$ events such that $\mathsf{Pr}(A_i) \leq p$ with dependency graph D such that D has maximum degree d.*

Thus, if

$$e\dot{p}(d+1) \leq 1 \tag{3}$$

then we have that

$$\mathsf{Pr}(\cap_{i=1}^{n} \overline{A_i}) > 0. \tag{4}$$

3 Examples & Set Coloring

We can start with a simple example presented below. Now we can apply LLL to the 2−coloring problem to get get a condition independent of the number of sets

Theorem 5 *Let $S_1, ..., S_m \subseteq \mathbb{X}$ such that $|S_i| = l$ and each of S_i intersects $\leq d$ other S_i's. If $e(d+1) \leq 2^{l-1}$, then we can 2-color X such that no S_i is monochromatic.*

Proof

In the same way as last time, we color each element of X red or blue with probability $p = \frac{1}{2}$. We define the event A_i as $A_i = \{S_i$ **is monochromatic**$\}$. Since every element is colored independently, we get the following relationship:

$$\mathsf{Pr}[A_i] = \frac{1}{2^{l-1}} \tag{5}$$

A_i independent of all A_j such that $S_i \cap S_j = \emptyset$, thus S_i depends on at most d other A_i's (here we use our degree bounds).

Note that if the intersection of two sets is monochromatic then, these probabilities increase. But if two sets do not intersect then they are independent.

Thus, we have that $ep(d+1) = e\frac{1}{2^{l-1}}(d+1) \leq 1$. Thus, by LLL we have that:

$$Pr[\cap_i \overline{A_i}] > 0$$

■

Observation 6 *Notice that this theorem today has a less strong assumption on the sets. Instead of it being less than 2^{l-1}, we have not restriction on size but only on "number of dependencies"*

These sets S_i can thought of as the **hyper-edges** of a hyper-graph. For example in the Figure 1, We have the pink and blue sets; they both can be thought of as size three hyperedges.

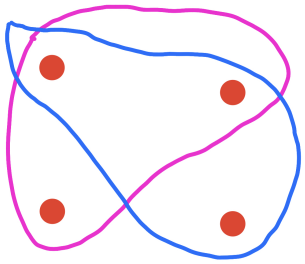

Figure 1: Hypergraph with hyperedges of size three.

One of the reasons that the LLL theorem developed to solve the problem of satisfiability.Take a CNF formula with l variables in each clause, and less than k clauses. If $\frac{e(lk+1)}{2^{l-1}} \leq 1$, then there is a satisfying assignment. This is relevant for the NP-hard $k - SAT$ problem (with $k > 2$) that determines whether a clause has a satisfying assignment. Moreover, for some inputs we can directly solve the problem just by checking clause satisfies the conditions mentioned before.

4 Algorithm and Proof

If you remember the probabilistic method, one of the main problems was that despite being able to prove the existence of a construction it did not give us a procedure to find this said construction. Thus, it was quite important when LLL was developed together with an algorithms that could find the construction.

4.1 History

The first person to make an algorithmic version of the LLL was *Beck* in 1991. However, this random algorithm made a really big assumption of $d \leq 2^{l/1000}$. Not long after, Alon made a better result with the assumption that $d \leq 2^{l/8}$. Finally, in 2009, Moser (later improved further by Moser and Tardos) were able to give a polynomial time algorithm under the assumption that $d \leq \frac{2^l}{c}$.

The algorithm developed by Moser and Tardos is the more simple one and is presented below. However, this is not the algorithm that we will prove works. It is also important to note that all the algorithms discussed hereon are solving the set 2-coloring problem.

```
2-color 𝕏 randomly ;
while some S_i is monochromatic do
    pick monochromatic S_i ;
    randomly assign colors of S_i ;
end
```

Algorithm 1: Moser-Tardos

If you look in the textbook then you can find three different versions of the algorithm for finding a 2-coloring.

4.2 Beck's Algorithms and Proof of Correctness & Running Time

4.2.1 Assumptions

Let $S_1, ..., S_m \subseteq X$ all with same size l and such that for each S_i there at most d other S_j such that $S_j \cap S_i \neq \emptyset$. Assume that l and d are constant.

Definition 7 *A set S_i with some coloring will be **bad** if at least $\leq \alpha \dot{l}$ points are all red or all blue where where $\alpha << \frac{1}{2}$ is some chosen constant*

We will consider the connected components of **bad** sets. Where two sets S_i and S_j are connected if $S_i \cap S_j \neq \emptyset$. Notice that these connected components can have any number of sets even $O(n)$.

```
(1^st pass )2-color X randomly ;
B ← {S_i | S_i is bad} ;
if all connected components of B are at most ≤ d^2 loġm then
    (2^nd pass ) brute force fix each of connected components of B without making
     neighbours monochromatic
else
    retry
end
```

Algorithm 2: Beck's Algorithm

Observation 8 *Note that sets that are **bad** are far from monochromatic. In the brute force second pass of the algorithm, we just want every set not be monochromatic.*

If the above algorithm is true, then we can use a pass of a simple randomized algorithms to do most of the work for us, and then we can do a second quick pass to fix it. This idea is very useful in randomized algorithms. An analogy in snow removal is when the snow is scattered around into lots of small patches, it melts rapidly. However, when it is heaped into a big pile, it can take a long time to melt.

4.2.2 Proof of Correctness and Run time

We have three main questions to give a full proof.

Question 9 *How do we know we reach a good solution ?*

To answer this questions we will just show that a good coloring for each of the sets considering three cases:

- Case 1: S_i **is bad**. If S_i is a bad set that is close to monochromatic, then, we will recolor everything.
- Case 2: S_i **is not bad** and has less than αl nodes in bad neighbours , then S_i will still be bi chromatic after the 2^{nd} pass (recoloring of bad components).
- Case 3: If S_i is not bad and it has more than αl nodes in bad neighbours , then $\geq \alpha l$ nodes get recolored. For this case, we just need to apply LLL again to this part of the graph together with all **bad** sets from case 1.
  - If recolor randomly the sets in case 1 and 3, then $\mathsf{Pr}[S_i \textbf{ is monochromatic}] \leq 2^{-\alpha l}$. Thus, using LLL and the assumption that $2e(d+1) \leq 2^{\alpha l}$. Then the solution exists.

Question 10 *How are you going to brute force, this is usually not good?*

Brute Force is actually not that bad in this case.

- Size of the surviving component is $O(d^2 \log m)$. The number of settings to variables in each $2^{l\dot{O}(d^2 \log m)} = m^{O(ld^2)}$

  Here is where the assumption that l and d are constants give us a polynomial time. To work with the case where they are not constants we just apply recursion. Multiple iterations with embedded logarithms that become near constants

Question 11 *How many times are we going to have to retry?*

This is the toughest part of the proof. First, we will give an expression for the probability of a set S_i being bad with the lemma below.

Lemma 12 *For each set S_i, we have that:*

$$\mathsf{Pr}[S_i \textbf{ is bad}] = 2 \cdot 2^{(H(\alpha)-1)l} = p \tag{6}$$

where $H(x) = -x \log_2 x - (1-x) \log_2 1 - x$.

Note that p as the probability of S_i being bad. With the previous property of the probability, we can get the following lemma.

Lemma 13 *Let C be some **independent** set with $S_1, ..., S_{|C|}$, then the probability that they are all bad can be given by:*

$$\mathsf{Pr}[C \subseteq B] \leq p^{|C|} \tag{7}$$

*where B is the set of all **bad** sets as defined above and p is the probability*

We will do an initial approach to proving that big bad components do not happen often.

- **Attempt 1** : Let C be a big component $C = \{S_1, ..., S_{|C|}\}$ from the dependency graph D defined above that is also contained in the set B of **bad** components. We want to show that the probability that C has a non **bad** set is high.

  To do this, we involve independent sts. Then let C' be an independent component in c. Then, if $S' \subseteq S \subseteq B$. This gives us the following

  $$Pr[S \subseteq B] \leq Pr[S' \subseteq B] \leq p^{|S'|} \tag{8}$$

  Notice that in this case $|S'|$ is the size of the set and the first inequality comes from $S' \subseteq S$ and the second one from the lemma above.

  Thus, we can use union bound for the following probability. Let $C_1, ...C_k$ be all the k big components (components that are not logarithmic in size).

  $$\mathsf{Pr}[\cap_i \{C_i \subseteq B\}] \leq \sum_i \mathsf{Pr}[C_i \subseteq B] \leq k \cdot p^{\min_i |C_i'|} \tag{9}$$

  Note that here $\min_i |C_i'|$ refers to the size of the smallest independent set in each of the set C_i's

  In this first attempt, k could be a really large number like $\binom{n}{min_i |C_i|}$. Moreover, the smallest independent set could be size 1. Thus, $k \cdot p^{\min_i |C_i'|}$ could be a number bigger than 1 which is not useful.

  However, the bounded degree of the graph can provide more information on bounding k and $\min_i |C_i'|$ to improve this result.

- **Attempt 2**: First, we will show a bound lower bound for $|C'|$

  Lemma 14 *Given a subgraph of H of size S in graph of degree $\leq \Delta$, Then, there must be an independent set in H of size $\leq \frac{|H|}{\Delta+1}$*

  Proof We will have a greedy algorithm. In each iteration, take an arbitrary node u in H and add it to the independent set. Then remove all neighbours of u from H. This removes at most $\Delta + 1$ nodes from H in each *iteration*. Repeat until H is empty Since we require at least

  $|H|/(\Delta + 1)$ iterations until H is empty, and each iteration adds a node to the independent set, the independent set is of size at least $|H|/(\Delta + 1)$

  ■

To be continued...