Randomized Algorithms

Lecture 12: proofs of the Lovasz Local Lemma

Kousha Etessami

The Lovász Local lemma

Consider a large bunch of "bad events", $E_1, E_2, \ldots, E_n$.

Suppose that in order to show existence of a desired object, using the probabilistic method, we have to avoid all these bad events. In other words, we want to show:

$$\Pr[\bigcap_{i=1}^{n} \overline{E}_i] > 0 \quad (1)$$

Supppose $\Pr[E_i] < 1$, for all i. (Otherwise, there's no hope.)

If the events $E_1, \ldots, E_n$ are mutually independent then (1) is easy, because:

$$\Pr[\bigcap_{i=1}^{n} \overline{E}_i] = \prod_{i=1}^{n} \Pr[\overline{E}_i] = \prod_{i=1}^{n} (1 - \Pr[E_i]) > 0.$$

Note that this could be a very small probability, e.g., if n is very large, but nevertheless it is a positive probability, so existence follows.

Unfortunately, often the bad events may not be independent.

The Lovasz Local Lemma allows us to establish (1) in contexts where there is some limited dependencies between the E_i's.

The Lovász Local lemma

Let us define a particular event A to be *mutually independent of* a set of events $\{E_1, E_2, \ldots, E_k\}$ if for all subsets $I \subseteq \{1, \ldots, k\}$, we have

$$\Pr\left[A \mid \bigcap_{i \in I} \overline{E}_i\right] = \Pr[A] = \Pr\left[A \mid \bigcap_{i \in I} E_i\right].$$

Definition (6.1) A *dependency graph* for a set of events $E_1, \ldots, E_n$ is a directed graph $G = (V, E)$ such that $V = \{1, \ldots, n\}$ and for each $i \in V$, the event E_i is mutually independent of the set of events $\{E_j \mid (i,j) \notin E\}$. The *degree* of G is the maximum out-degree of any vertex in G.

Theorem (Lovász Local Lemma (symmetric version))

Let $E_1, \ldots, E_n$ be a set of events. Suppose that for some $p \in (0, 1)$ and some $d \in \mathbb{N}$ the following conditions hold:

1. *For all i, $\Pr[E_i] \leq p$;*
2. *A dependency graph on $\{E_1, \ldots, E_n\}$ has degree $\leq d$;*
3. *$4dp \leq 1$.*

Then

$$\Pr\left[\bigcap_{i=1}^{n} \overline{E}_i\right] > 0.$$

Important application: satisfiability of k-CNF formulas

Recall the k**-SAT problem**: Given a k-CNF boolean formula, φ, where each clause has exactly k literals, decide whether φ is satisfiable. Recall, k**-SAT** is **NP-complete**, already for $k = 3$.

Theorem. *If no variable in a k-CNF formula φ appears in more than $\frac{2^k}{4k}$ clauses, then φ is satisfiable.*

Proof. Randomly and independently assign each boolean variable x_i either 0 or 1 with probability $1/2$ each. Suppose there are m clauses, $C_1, \ldots, C_m$, in φ. Let E_i, $i = 1, \ldots, m$, denote the event that C_i is not satisfied. Since each C_i has k literals, we have $\Pr[E_i] = 2^{-k}$.

But E_i is independent of all E_j for which C_i and C_j don't share any variables. Since each of the k variables in C_i appears in $\leq \frac{2^k}{4k}$ clauses, there is a dependency graph for the E_i's with degree $d \leq k \cdot \frac{2^k}{4k} = \frac{2^k}{4}$. Letting $p = 2^{-k}$, we have $4dp \leq 4 \cdot \frac{2^k}{4} \cdot 2^{-k} = 1$.
So we can apply the Lovasz Local Lemma to conclude:

$$\Pr[\bigcap_{i=1}^{m} \overline{E}_i] > 0,$$

meaning φ is satisfiable. □

Outlook

- In this last lecture we will prove the Lovasz Local Lemma.
- We will first give a classic, but non-constructive proof.
- Then we will describe a more recent beautiful algorithmic proof by Moser (2009) (later generalized by Moser & Tardos (2010)), which gives us, in particular, a randomized (Las Vegas) polynomial time algorithm for computing a satisfying assignment for k-SAT instances that satisfy the conditions of the theorem we stated on the prior slide.

Proof of the Lovász Local Lemma

The key to the proof is to establish the following claim, by induction.

Claim. *For $s = 0, \ldots, n-1$, if $S \subset \{1, \ldots, n\}$ & $|S| \leq s$, then:*

- $\Pr[\bigcap_{j \in S} \overline{E}_j] > 0$, *and*
- *for all $k \notin S$,* $\Pr[E_k \mid \bigcap_{j \in S} \overline{E}_j] \leq 2p$.

Using this claim, it is easy to establish the full result. Expanding $\Pr\left[\bigcap_{i=1}^{n} \overline{E}_i\right]$ using the chain rule of conditional probabilities gives:

$$\Pr\left[\bigcap_{i=1}^{n} \overline{E}_i\right] = \prod_{i=1}^{n} \Pr\left[\overline{E}_i \mid \bigcap_{j=1}^{i-1} \overline{E}_j\right] = \prod_{i=1}^{n} \left(1 - \Pr\left[E_i \mid \bigcap_{j=1}^{i-1} \overline{E}_j\right]\right)$$

$$\geq \prod_{i=1}^{n} (1-2p) = (1-2p)^n > 0.$$

In the last step, $(1-2p) > 0$ holds because $4dp \leq 1$, and hence certainly $2p < 1$, unless $d = 0$ in which case the whole Lovasz Local Lemma would hold trivially (because all E_i's would then be mutually independent).

proof of Key Claim

Base case ($s = 0$): in this case $S = \emptyset$. Hence $\Pr[\bigcap_{j \in S} \overline{E}_j] = 1 > 0$ holds vacuously, and $\Pr[E_k \mid \bigcap_{j \in S} \overline{E}_j] = \Pr[E_k] \leq 2p$ holds by the assumption that $\Pr[E_j] \leq p$ for all j.

Induction step: Assume true for $0, 1, \ldots, s-1$. We show it for s.

We first show $\Pr[\bigcap_{j \in S} \overline{E}_j] > 0$.

If $s = 1$, this follows from the assumptions: $\Pr[\overline{E}_j] \geq 1 - p$.

For $s > 1$, then we use the induction hypothesis for $(s-1), \ldots, 0$. Without loss of generality, suppose $S = \{1, \ldots, s\}$. Then, as we've already seen:

$$\begin{aligned}
\Pr\left[\bigcap_{i \in S} \overline{E}_i\right] = \prod_{i=1}^{s} \Pr\left[\overline{E}_i \mid \bigcap_{j=1}^{i-1} \overline{E}_j\right] &= \prod_{i=1}^{s}\left(1 - \Pr\left[E_i \mid \bigcap_{j=1}^{i-1} \overline{E}_j\right]\right) \\
&\geq \prod_{i=1}^{s}(1 - 2p) = (1 - 2p)^s > 0.
\end{aligned}$$

Here in the last line we have used the induction hypothesis for $(s-1), \ldots, 0$.

proof of Key Claim (cont'd.)

Induction step (cont'd): We want to show $\Pr\left[E_k \mid \bigcap_{j\in S} \overline{E}_j\right] \leq 2p$.

Let $S_1 = \{j \in S : (k,j) \in E\}$, and let $S_2 = S \setminus S_1 = \{j \in S : (k,j) \notin E\}$.

If $S_2 = S$, then E_k is mutually independent of all events $\{\overline{E}_j \mid j \in S\}$, in which case we would be done because
$\Pr\left[E_k \mid \bigcap_{j\in S} \overline{E}_j\right] = \Pr[E_k] \leq p \leq 2p$.
Otherwise, we have $|S_2| < s$, and $S_1 \neq \emptyset$. In this case, we can write

$$\Pr\left[E_k \mid \bigcap_{j\in S} \overline{E}_j\right] = \frac{\Pr\left[E_k \cap \bigcap_{i\in S_1} \overline{E}_i \;\mid\; \bigcap_{j\in S_2} \overline{E}_j\right]}{\Pr\left[\bigcap_{i\in S_1} \overline{E}_i \;\mid\; \bigcap_{j\in S_2} \overline{E}_j\right]} \qquad (2)$$

The numerator on the right of (2) is $\leq \Pr\left[E_k \mid \bigcap_{j\in S_2} \overline{E}_j\right]$ which by mutual independence is $= \Pr[E_k] \leq p$.
What's left is to lower bound the denominator

$$\Pr\left[\bigcap_{i\in S_1} \overline{E}_i \;\mid\; \bigcap_{j\in S_2} \overline{E}_j\right].$$

proof of Key Claim (cont'd.)

$$
\begin{aligned}
\Pr\left[\bigcap_{i\in S_1}\overline{E}_i \mid \bigcap_{j\in S_2}\overline{E}_j\right] &= \left(1-\Pr\left[\bigcup_{i\in S_1}E_i \mid \bigcap_{j\in S_2}\overline{E}_j\right]\right)\\
&\geq \left(1-\sum_{i\in S_1}\Pr\left[E_i \mid \bigcap_{j\in S_2}\overline{E}_j\right]\right) \quad \text{(Union bound)}\\
&\geq \left(1-\sum_{i\in S_1}2p\right) \quad \text{(induction hypothesis)}\\
&\geq 1-d2p \quad \text{(since by assumption } |S_1|\leq d\text{)}\\
&\geq \frac{1}{2} \quad \text{(since } 4pd\leq 1\text{, and hence } 2pd\leq \tfrac{1}{2}\text{)}
\end{aligned}
$$

Thus, we have

$$
\Pr\left[E_k \mid \bigcap_{j\in S}\overline{E}_j\right] \leq \frac{p}{(1/2)} = 2p.
$$

which completes the proof. □

Note: this proof is non-constructive. For example, it gives us no clue how to construct a satisfying assignment to a k-CNF formula where every variable occurs in at most $\frac{2^k}{4k}$ clauses, even though it shows that such a satisfying assignment must exist.

We're about to rectify that, with a beautiful new algorithmic proof due to Moser (2009). (This was subsequently generalized by Moser & Tardos (2010) to the setting of the general, asymmetric, Lovasz Local Lemma.)

Moser's proof (2009)

Theorem (Moser,2009)

If every clause in a k-CNF formula φ shares a variable with at most $2^{k-3} - 1$ other clauses, then φ is satisfiable.
Furthermore, there exists a (Las Vegas) randomized algorithm that, given such a φ as input, runs in expected polynomial time and outputs a satisfying assignment to φ.

Proof. The proof shows that the following amazingly simple randomized algorithm "works".

Input: A list of clauses $C_1, C_2, \ldots, C_m$ of a k-CNF formula, φ over n variables $\{x_1, x_2, \ldots, x_n\}$.

Output: A satisfying truth assignment for φ.

Main routine:

$\alpha \longleftarrow$ *a random u.a.r. truth assignment to the variables;*

while *some C_i is not satisfied by α* **do**

1. *choose the unsatisfied C_i with the smallest index i;*
2. *call* **local-correct**(C_i)*;*

local-correct(C):

$\alpha \longleftarrow$ *same as α, except with variables in C resampled u.a.r.*

while *some C_j that shares a variable with C is not satisfied by α* **do**

1. *choose such an unsatisfied C_j with the smallest index j*
2. *call* **local-correct**(C_j)*;*

Note: If this algorithm terminates, then it outputs a satisfying assignment.

Question: Why does this algorithm terminate?? And why does it terminate in expected polynomial time?

The argument for why the algorithm terminates with probability 1 (and in expected polynomial time) is based on a beautifully simple and elegant "entropy compression" argument.

The argument appeals to a basic and fundamental fact in Information Theory, namely the Noiseless Coding Theorem ([Shannon,1948]).

However, you don't need to know this theorem, nor understand any Information Theory at all, in order to understand the intuitive argument.

Let us re-state the algorithm, adding some book-keeping of “history”.

Main routine:

$\alpha \longleftarrow$ *a random u.a.r. truth assignment to the variables;*

while *some C_i is not satisfied by α* **do**

1. *choose the unsatisfied C_i with the smallest index i;*
2. *enter i in binary using $\lceil \log_2(m) \rceil$ bits in the history;*
3. *call* **local-correct**(C_i)*;*

local-correct(C):

$\alpha \longleftarrow$ *same as α, except with variables in C resampled u.a.r.*

while *some C_j that shares a variable with C is not satisfied by α* **do**

1. *choose such an unsatisfied C_j with the smallest index j;*
2. *enter “0” followed by a code for j in binary using only $k-3$ bits in the history;*
3. *call* **local-correct**(C_j)*;*

Enter “1” in the history;

Note that one way to fully describe the exact behavior of this randomized algorithm on a given k-CNF formula is by giving the exact sequence of random bits used by the algorithm.

In particular, let j be the number of "rounds" of the algorithms, meaning the number of times **local-correct** has been called by the algorithm.

Note that each "round" requires resampling the k boolean variables in a single clause, and hence requires precisely k additional u.a.r. random bits.

Then one way to describe in full the algorithm's operation up to j rounds is to specify the

$$n + (j \cdot k)$$

u.a.r. random bits that were used by the algorithm.

(Note that knowing these bits determines every step taken by the algorithm up to j rounds.)

However, there is also a different way to describe the full operation of the algorithm, up to j rounds, using the history.

A different way to describe the algorithm's behavior: History

Suppose the algorithm runs for J resampling rounds.

Consider the history that we have recorded. It records, in particular, the index i in binary using $\lceil \log_2(m) \rceil$ bits every time an unsatisfied clause C_i is resampled by the **Main routine**.

Additionally, it records a start flag bit, "0", followed by an index j recorded crucially using only $k-3$ bits in the history, every time an unsatisfied clause C_j is called inside a (recursive) invocation of **local-correct**, and it records an end flag bit "1" just before a (recursive) invocation of **local-correct** terminates.

We assume the history also keeps track of the current assigment of truth values to the n boolean variables, which requires n bits.
In total, after J rounds, the recorded history requires at most:

$$n + m\lceil \log_2(m) \rceil + J(k-1)$$

bits in total. This is because each resampling call requires at most $k-1$ bits in the history, including the two possible flag bits "0" and "1", plus the $k-3$ bits to record j for the unsatisfied C_j being resampled that shares a variable with the current clause C.

Key Claim. *After J rounds of the algorithm, given the $n + m\lceil \log_2(m) \rceil + J(k-1)$ bits of history, we can uniquely recover the $n + (J \cdot k)$ u.a.r. random bits that were generated by the algorithm.*

Proof. The key point is this: knowing the history, allows us to reconstruct the precise sequence of recursive calls of **local-correct**, and what clauses it was called on.

Then, going back through history, we know at each step which clause was being resampled.

Crucially, if we know clause C_j has just been resampled, we know exactly what the values of the variables in C_j were prior to being resampled, because there is only one assigment to those variables that does not satisfy C_j.
Hence, using the n bits of the assignment at the end of J rounds, which is part of our "history", we can "step back through time" using the history, to uniquely decipher the values of the assigment during every round of the algorithm, all the way back to the start, when the first n bits were initially sampled. □

Note that the **Key Claim** means that the $n + m\lceil\log_2(m)\rceil + J(k-1)$ bits of history constitute a uniquely decipherable code for the $n + (J \cdot k)$ u.a.r. random bits that describe the entire run of the algorithm.

However, intuitively, this means that we should have

$$n + m\lceil\log_2(m)\rceil + J(k-1) \quad \geq \quad n + (J \cdot k) \qquad (3)$$

which would imply that $J \leq m\lceil\log_2(m)\rceil$.

This is because if (3) does not hold, we would have a way of compressing $n + (J \cdot k)$ u.a.r. random bits "down to" $n + m\lceil\log_2(m)\rceil + J(k-1)$ bits.

Intuitively, this should be impossible to do on average, meaning the expected code length for N random bits cannot be strictly less than N.

Indeed, this is impossible, thanks to basic facts in Information Theory.

Basic facts from Information theory

Theorem. (cf. Noiseless Coding Theorem [Shannon,1948]) *The expected code length of any binary uniquely decipherable code for a random variable X is at least $H(X)$, where $H(X)$ is the entopy of X.*

Proposition. *For any $N \geq 1$, the entropy of N u.a.r. random bits is equal to N.*
In other words, if X denotes a random variable corresponding to sampling N u.a.r. random bits (each sequence of N bits has probability $\frac{1}{2^N}$), then $H(X) = \sum_{j=1}^{2^N} \frac{1}{2^N} \log_2(2^N) = N \sum_{j=1}^{2^N} \frac{1}{2^N} = N$.

Hence, we can conclude that with positive probability the algorithm must halt after $J \leq m\lceil \log_2(m) \rceil$ rounds, meaning that a satisfying assignment is found after J rounds. Hence a satisfying assigment must exist.

We can also show (with just a little bit more analysis) that the expected number of rounds of the algorithm is also $O(m\lceil \log_2(m) \rceil)$. Hence, we have a randomized (Las Vegas) algorithm which runs in expected polynomial time, and generates a satisfying assigment, given such a k-CNF formula. □

Conclusion

- That concluded our lectures.
- If you want to learn a bit more about entropy and Information Theory see Chapter 10 of [MU], but this is not required and not examinable.