CS265/CME309: Randomized Algorithms and Probabilistic Analysis
Lecture #12: The Constructive Lovasz Local Lemma (Moser's Entropic Proof)

Gregory Valiant*

November 2, 2019

1 Introduction

Last class we saw the statement and proof of the *existential* Lovasz Local Lemma. Today, we will see a constructive (algorithmic) version of the theorem. There has been a long progression of work towards a strong algorithmic version of the theorem over the past thirty years (e.g. [3, 2, 6, 4, 9, 7, 8, 5, 1]). In order to phrase an algorithmic version of the theorem from last class, it will be convenient to slightly restrict the set of events and probability distributions that we will consider.

Let V be a finite set of independent random variables, and let $\mathcal{A}$ denote a finite set of events that are determined by V. That is, each event $A \in \mathcal{A}$ maps the set of assignments of V to $\{0, 1\}$.

Definition 1. *Given the set of independent random variables V and set of events $\mathcal{A}$ determined by the variables of V, define the* relevant variables *for an event $A \in \mathcal{A}$, denoted $vbl(A) \subset V$ to be the smallest subset of variables that determine A. Additionally, for an event $A \in \mathcal{A}$, let $\Gamma(A) = \{B : vbl(A) \cap vbl(B) \neq \emptyset\}$ denote the set of events that share variables with A, and note that A is mutually independent from the set $\mathcal{A} \setminus \Gamma(A)$.*

The following algorithm is one extremely natural approach for finding an assignment to the variables that avoids all the events $\mathcal{A}$:

Algorithm 2. FIND ASSIGNMENT
Given V, $\mathcal{A}$:

- *Choose a random assignment σ_v for each of the random variables $v \in V$.*
- *While there exists an $A \in \mathcal{A}$ such that $A(\sigma) = 1$:*
  - *Choose (arbitrarily according to any scheme, randomized or deterministic) an event A with $A(\sigma) = 1$, and update σ by re-selecting a random assignment to the variables $vbl(A)$.*

The following theorem, due to Moser and Tardos in 2010 [8], shows that the above algorithm will, with high probability, successfully terminate quickly. The following formulation closely matches the guarantees of the "asymmetric" LLL from last lecture:

Theorem 1. *[8] Let V be a finite set of independent random variables. Let $\mathcal{A}$ be a finite set of events determined by the random variables in V. If there exists an assignment $x : \mathcal{A} \to (0, 1)$ such that for all $A \in \mathcal{A}$,*

$$\Pr[A] \leq x(A) \prod_{B \in \Gamma(A) \setminus \{A\}} (1 - x(B)),$$

then Algorithm 2 will find an assignment to the variables V such that no event of $\mathcal{A}$ occurs. Additionally, the expected number of "re-randomizations" is bounded by $\sum_{A \in \mathcal{A}} \frac{x(A)}{1-x(A)}$.

The above theorem implies the following algorithmic version of the simpler (symmetric) LLL:

Corollary 3. *Let V be a finite set of independent random variables. Let $\mathcal{A}$ be a finite set of events determined by the random variables in V. If for all $A \in \mathcal{A}$, $|\Gamma(A)| \leq d+1$, and $\Pr[A] \leq \frac{1}{e(d+1)}$, then Algorithm 2 will find an assignment to the variables V such that no event of $\mathcal{A}$ occurs. Additionally, the expected number of "re-randomizations" performed by the algorithm is bounded by $O(|\mathcal{A}|/(d+1))$.*

Proof. We apply Theorem 1 and set $x(A) = \frac{1}{d+1}$. To see why the assumptions of the theorem hold, note that

$$x(A) \prod_{B \in \Gamma(A) \setminus \{A\}} (1 - x(B)) \geq \frac{1}{d+1}(1 - \frac{1}{d+1})^d \geq \frac{1}{e(d+1)}$$

where we used our assumption that $|\Gamma(A)| \leq d+1$ and the fact that $(1 - \frac{1}{d+1})^d \geq 1/e$. To conclude, recall that we assumed that $\Pr[A] \leq 1/e(d+1)$, and hence the assumptions of the theorem are satisfied. □

The original proof of Theorem 1 proceeded by bounding the expected number of times each event $A \in \mathcal{A}$ could be selected as an event whose variables are to be re-randomized (in the third line of Algorithm 2). The proof eventually turns into an analysis of a process resembling the Galton-Watson branching process—corresponding to the process where the "offspring" of an event A whose variables are re-randomized corresponds to the events that are must now be fixed as a result of that assignment (i.e. the events that are now true because of the new assignment to $vbl(A)$). Intuitively, as long a the expected number of offspring is < 1, this process should die out, and we should end up with an assignment s.t. no event occurs. Rather than going into this rather involved proof, we will instead describe Moser's "entropic" proof, which was not contained in the original paper.

2 Moser's Entropic Proof

The core idea of the *entropic* proof is to argue that Algorithm 2 gobbles up randomness more quickly than it actually uses it, in the sense that if the algorithm were to run for too long, then we would be able to *compress* the string of random bits used by the algorithm. And, as we show below with a simple counting argument, it is impossible to significantly compress a string of random bits. This compression/entropic argument is extremely elegant—arguing that the expected runtime must be

small because otherwise, we would be able to compress the random bits used by the algorithm. I am not aware of such an argument being used to bound the runtime of an algorithm in any other setting. There is some very recent work (past 3 years) that is trying to generalize this sort of analysis—if you are interested, see [1].

Fact 4 (The incompressibility of random strings)**.** *For any function f that maps t-bit binary strings to distinct strings of (possibly variable) length, if s is a uniformly random binary string of length t, then for any integer c, $\Pr[|f(s)| \leq t - c] \leq \frac{1}{2^{c-1}}$.*

Proof. Since there are 2^i strings of length i, there are at most $\sum_{i \leq t-c} 2^i < 2^{t-c+1}$ strings, that can be mapped to strings of length at most $t - c$, and hence the probability that a random length t string is in this set is at most $\frac{2^{t-c+1}}{2^t} = \frac{1}{2^{c-1}}$. □

The "entropic" proof is especially clean in the specific setting of k-SAT (rather than in the fully general LLL setting), and we will focus on k-SAT for the rest of this lecture. Consider a k-SAT formula over n variables, $x_1, \ldots, x_n$, with clauses $A_1, \ldots, A_m$, where $vbl(A_i)$ denotes the set of variables occurring in the ith clause, and $|vbl(A_i)| = k$.

Theorem 2. *Consider a k-SAT formula with m clauses over n variables. If, for each clause C in the formula, there are at most 2^{k-3} clauses whose variable sets intersect the variables in clause C, then the formula is satisfiable, and there is a randomized algorithm that starts with any fixed assignment, iteratively re-randomizes the assignment to variables in some unsatisfied clause, such that for any integer $c > 0$, the probability that the algorithm has found a satisfying assignment after at most $n + 3 + c$ "re-randomizations" is at least $1 - \frac{1}{2^c}$.*

Proof. Ultimately, we will argue that if the algorithm were to run for too long, in expectation, then we would end up with a protocol that compresses the random bits used by the algorithm. To do this, we will imagine a game being played between the "Algorithm" who is performing the re-randomizations, and the "Listener", who is receiving updates on what the algorithm. Both the algorithm and Listener know the formula. The messages that the algorithm sends the Listener will have 2 properties:

1. After T re-randomizations of the algorithm, the Listener will be able to reconstruct the sequence of random bits used by the algorithm up until that point.

2. The total length of all the messages that the algorithm has sent to the Listener up through the Tth re-randomization, is bounded by $C + (k-1)T$, where the constant C depends on the number of variables, n but is independent of T.

Since the algorithm uses k random bits per re-randomization, if the algorithm has not already finished after $T-1$ re-randomizations, then the messages can be viewed as a way of compressing the Tk bits into a string of length $C + (k-1)T$. We can turn this into a compression scheme for all kT length bit strings as follows: if the algorithm terminates before its Tth re-randomization, then send the length kT bit string uncompressed, otherwise send the messages. Provided $C + (k-1)T = kT - c$, Fact 4 yields that the probability the algorithm has terminated before the Tth re-randomization is at least $1 - 1/2^c$.

We now instantiate the protocol that the algorithm uses to perform the re-randomizations and construct the messages. Roughly, the algorithm will start with some assignment to the variables, and

then will communicate which clause is being re-randomized at each step. After T re-randomizations, the algorithm will send the current assignment. The crux of the argument will be a clever way of *succinctly* communicating which clauses are being re-randomized using only $k-1$ bits of communication, as opposed to the $\log m$ bits that would be required naively to send the index of the clause. Below is a description of the algorithm and the messages it sends:

- At time $t = 0$, start with some known assignment to the variables (e.g. all $true$). [At this point, both the algorithm and listener know which clauses are unsatisfied by the current assignment.] Let b_i^0 indicate the index of the ith unsatisfied clause.

- We begin by re-randomizing the assignment to the variables in clause b_1^0. If that re-randomization causes any new clauses to be unsatisfied, we will refer to that set of clauses as "siblings" since they were all due to the same "parent" b_1^0. We send the Listener message 00 [indicating that the re-randomization broke something] and then communicate the identity of one of the broken clauses, say b_1^1. This just requires $\log 2^{k-3} = k-3$ bits to communicate, since at most 2^{k-3} clauses share variables with clause b_1^0.

- If no new clauses were broken, we send message 01 [indicating that no new clauses were broken, and we are now addressing a "sibling" clause, as opposed to a child of the previously re-randomized clause] and move on to clause b_2^1.

- In general, we continue in this fashion: if, in the previous re-randomization, any clauses were broken, we send message 00 and communicate one of the "children" of the clause we just re-randomized. If none were broken, we send message 01 indicating that we are moving on to re-randomize the next "sibling" of the clause we just re-randomized, and indicate which clause that is (via $\log 2^{k-3}$ bits). If there are no more "siblings", we send the message 11, indicating that we are now going to go up to the level of the parent clause.

- Finally, after either we have an empty stack (we've found a satisfying assignment), or have re-randomized T times, we send a length n message with the final/current assignment.

In the above protocol, if we reach the Tth re-randomization, the sum of the lengths of all the messages is $n + T(2 + \log 2^{k-3})$=n+T(k-1). We now argue that the listener will be able to recreate all Tk random bits used by the algorithm. To see this, the Listener will work their way backwards from the final assignment (communicated in the final message after the Tth re-randomization), and recreate the assignment that the algorithm had at each of the T steps of the algorithm. Recall that the Listener knows, at each re-randomization, exactly which clause was re-randomized. In order for a clause to be re-randomized, however, there is only one possible assignment that its k variables could have had immediately prior to its re-randomization (since that clause was not satisfied by the assignment prior to its re-randomization—that was why we re-randomized it!!) Hence, the listener can just work its way back from the end, deducing the assignment to all the variables immediately prior to each of the T re-randomizations. From this, the Listener can trivially read off the random bits used by the algorithm!

Setting $T = n + c$ yields that $n + T(k-1) = kT - c$ hence the algorithm must terminate with probability at least $1/2^{c-1}$ by the time it does $T = n + c$ re-randomizations. □

2.1 Discussion

One curious punchline that emerges from the proof is that we *want* the algorithm to use lots of randomness. If, instead of using k bits of randomness with every step, it only used $k-1$ bits, then the proof would not work, and we would not end up with any bound on the runtime. For example, consider a "greedy" algorithm which, rather than re-randomizing the variables in a given clause, tries to find an assignment to those k variables that satisfies that clause and as many other clauses are possible. Such a greedy scheme slightly reduces the amount of randomness consumed by the algorithm, and hence the proof from above would result in a worse bound on the expected runtime (or a possibly infinite bound). In practice, for some classes of formula, people have observed that this sort of greedy algorithm is actually much worse than actually re-randomizing. (At first, the greedy algorithm seems to make great progress, but then things start to stagnate/loop.) The above proof offers one conceptual explanation for why, in these settings, we want to maximize the amount of randomness the algorithm is actually using.

References

[1] Dimitris Achlioptas and Fotis Iliopoulos. Random walks that find perfect objects and the lovász local lemma. *Journal of the ACM (JACM)*, 63(3):22, 2016.

[2] N. Alon. A parallel algorithmic version of the local lemma. *Random Structures and Algorithms*, 2(4):367–378, 1991.

[3] J. Beck. An algorithmic approach to the lovasz local lemma. *Random Structures and Algorithms*, 2(4):343–366, 1991.

[4] A. Czuma and C. Scheideler. Coloring non-uniform hypergraphs: a new algorithmic approach to the general lovasz local lemma. In *SODA*, pages 30–39, 2000.

[5] B. Haeupler, B. Saha, and A. Srinivasan. new constructive aspects of the lovasz local lemma. In *FOCS*, pages 397–406, 2010.

[6] M. Malloy and B. Reed. Further algorithmic aspects of the lovasz local lemma. In *STOC*, pages 524–529, 1998.

[7] R. Moser. A constructive proof of the lovasz local lemma. In *STOC*, pages 343–350, 2009.

[8] R. Moser and G. Tardos. A constructive proof of the general lovasz local lemma. *Journal of the ACM*, 57(2), 2010.

[9] A. Srinivasan. Improved algorithmic versions of the lovasz local lemma. In *SODA*, pages 611–620, 2008.