CS265/CME309, Fall 2014. Instructor: Gregory Valiant

Constructive Lovasz Local Lemma

1 Introduction

Last class we saw the statement and proof of the original *existential* Lovasz Local Lemma. In this class 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 two decades (e.g. [2, 1, 5, 3, 8, 6, 7, 4]). Our treatment will closely follow that of [7].

2 A Constructive Lovasz Local Lemma

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.*

Definition 2. *Given the set of independent random variables V and set of events $\mathcal{A}$ determined by the variables of V, the associated* dependency graph *$G_{V,\mathcal{A}}$ is the graph with vertex set $\mathcal{A}$, and with an edge between $A, B \in \mathcal{A}$ if $A \neq B$, and $vbl(A) \cap vbl(B) \neq \emptyset$.*

For notational convenience, for an event $A \in \mathcal{A}$, let $\Gamma(A) = \{B : vbl(A) \cap vbl(B) \neq \emptyset\}$ denote the neighborhood of A in $G_{V,\mathcal{A}}$ including the vertex A.

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

Algorithm 3. 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 $vbl(A)$.*

Theorem 1. *[7] 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 3 will find an assignment to the variables V such that no event of $\mathcal{A}$ occurs. Additionally, the expected runtime of the algorithm is bounded by $\sum_{A \in \mathcal{A}} \frac{x(A)}{1-x(A)}$.

To prove the above theorem, we will end up bounding the expected number of times each event $A \in \mathcal{A}$ can be selected as an event whose variables are to be re-randomized (in the third line of Algorithm 3. To this end, we define the *execution log* of a run of the algorithm as a list $C(1), C(2), \ldots,$ where $C(i) \in \mathcal{A}$ is the event chosen at the ith iteration of the algorithm. (In the case that the algorithm has terminated after k such steps we will say that $C(j)$ is undefined for $j \geq k$.) The core of this argument will be the construction (and analysis) of a "witness tree" corresponding to each chosen event $C(i)$.

A "witness tree" will simply be a tree whose nodes consist of events of $\mathcal{A}$. We will construct a tree T_i for each $C(i)$, which one should think of as a tree that will "justify" how we ended up selecting event $C(i)$ to re-randomize $vbl(C(i))$. We will define T_i via a sequence of i trees, $T_{i,i}, T_{i,i-1}, T_{i,i-2}, \ldots, T_{i,1} = T_i$ as follows. Set $T_{i,i}$ to be the tree consisting of a single root node $C(i)$. For each $j < i$: if $C(j) \in \Gamma(A)$ for some event A that is in tree $T_{i,j+1}$, then add node $C(j)$ as a child of the deepest (furthest from the root, $C(i)$) occurrence of such an event; if $C(j)$ is not in the neighborhood of any event [node] in tree $T_{i,j+1}$, then set $T_{i,j} = T_{i,j+1}$.

We say that a given "witness tree" T *occurs* in the execution log C of the algorithm if there exists some i for which $T_i = T$.

Lemma 4. *For any witness tree T, the probability that T occurs in the execution log is at most*

$$\prod_{\text{nodes } A \text{ of } T} \Pr[A].$$

Proof. First observe that for all pairs of nodes at a given depth of a witness tree, their variable sets must be disjoint (because if two events have a variable in common, whichever is added to the tree second will have depth strictly greater than that of the event that was previously added, as it could be added as a child of that event).

Consider fixing the randomness in the resampling step. That is, for each variable p, we associate an infinite sequence of values $p^0, p^1, \ldots,$ (that were randomly generated) and we assume that the kth time the algorithm requests that variable p be resampled, it is assigned value p^k. (And the value p^0 was the value assigned at the beginning of the algorithm.) Given some node v in a witness tree T, with the node v labeled by event A, with $p \in vbl(A)$, let $S_v(p)$ denote the set of vertices in tree T that are deeper than node v and whose associated variable sets also contain variable p. Hence the value assigned to variable p at the time in the algorithm at which the event associated to v is considered, will be $p^{|S_v(p)|}$, since in the execution of the algorithm, variable p had been resampled in the steps exactly corresponding to vertices in the set $S_v(p)$. For $vbl(A) = \{p_1, \ldots, p_k\}$, observe that the assignments $\{p_i^{|S_v(p_i)|}\}$ must satisfy event A (otherwise event A could not be the event associated to node v. We now bound the probability that this occurs for all nodes in the tree T.

Consider the following process that "checks" a tree T: beginning at the deepest layer, and working our way up layer by layer towards the root, for each node v, with associated event A, randomly select values of $vbl(A)$, and say that the tree "checks" if for all nodes, the corresponding events are all satisfied. Clearly the probability that a given tree "checks" is exactly the product of the probabilities of the nodes. Now, consider performing this check using the same choice of randomness as the algorithm, that is, with p^k representing the kth random choice of variable p. At the stage in the check in which we consider a node v corresponding to event A with variable set $vbl(A) = \{p_1, \ldots, p_k\}$, the assignment considered will be precisely $\{p_i^{|S_v(p_i)|}\}$.

Hence the probability that T is a witness tree of an execution log is at most the probability that tree T "checks" via the above checking process. Note that for any two nodes u, v in a tree T

that could be a witness tree, the corresponding assignment in the checking process, say $\{p_i^{|S_v(p_i)|}\}$ and $\{q_i^{|S_u(q_i)|}\}$ must be disjoint, and hence the event that node u cheks is independent from the probability that node v checks. Thus the probability that the entire tree T checks (which upper bounds the probability that the tree is a witness tree of the execution log) is simply the product of the probabilities that each of the nodes check, as claimed. □

To finish the proof, we leverage the above lemma to bound the expected number of times the event A will be chosen (to re-randomize $vbl(A)$) during the execution of the algorithm. Let N_A denote the number of times event A is chosen in the execution log C. Note that N_A is also the number of distinct witness trees with root node labelled A that occur in C. (To see this, note that if $C(i) = A$ and A has occurred j times in $C(1), \ldots, C(i)$, then the corresponding witness tree T_i will have exactly j nodes labelled with A, and hence the witness trees that occur in C with root node A will all be distinct.)

Consider a Galton-Watson branching process for generating a witness tree with root node A, in which nodes take one of a set of different random variables (as opposed to the basic Galton-Watson process we saw in lecture 7 in which all nodes are generated according to the same random variable). That is, we start with node A at time $t = 0$, and in general at time $t = i+1$, for each node $B \in \mathcal{A}$ that exists at time $t = i$, we (independently) add a child node $D \in \gamma(B)$ with probability $x(D)$ (where these choices of $x(D)$ are as prescribed in the statement of the theorem. This branching process will continue infinitely, or until it dies out (there is some time at which no nodes have children).

The following lemma characterizes the probability that the above Galton-Watson process yields a given witness tree:

Lemma 5. *Define $x'(B) = x(B) \prod_{D \in \Gamma(B), D \neq B}(1 - x(D))$. The probability that the above Galton-Watson process produces a witness tree T with root node A is*

$$p_T = \frac{1 - x(A)}{x(A)} \prod_{\text{nodes } B \text{ of } T} x'(B).$$

Proof. For some node v of T labelled by event B, let $W_v \subset \Gamma(B)$ denote the set of neighbors of B in the dependency graph that are *not* children of v in tree T. We have

$$p_T = \frac{1}{x(A)} \prod_{\text{nodes } v \in T} \left(x(v) \prod_{u \in W_v} (1 - x(u)) \right),$$

where we admit the slight abuse of notation, and use u, v to denote both the nodes in the tree, and the events corresponding to those labels. Note the leading factor of $1/x(A)$ is there simply because A is the root, which is always present by definition of the Galton-Watson process. Simplifying this expression, we obtain:

$$p_T = \frac{1 - x(A)}{x(A)} \prod_{\text{nodes } v \in T} \frac{x(v)}{1 - x(v)} \prod_{u \in \Gamma(v)} (1 - x(u)) = \frac{1 - x(A)}{x(A)} \prod_{\text{nodes } v \in T} x'(v),$$

as claimed. □

We now finish the proof of the theorem.

Proof of Theorem 1. Let $\mathcal{T}_A$ denote the set of witness trees with root node labelled A. By Lemma 4,

$$\mathbf{E}[N_A] = \sum_{T \in \mathcal{T}_A} \Pr[T \text{ occurs in } C] \leq \sum_{T \in \mathcal{T}_A} \prod_{\text{nodes } v \in T} \Pr[v].$$

From the definition of x' and the assumptions in the theorem statement, $\Pr[v] \leq x'(v)$, hence

$$\mathbf{E}[N_A] \leq \sum_{T \in \mathcal{T}_A} \prod_{\text{nodes } v \in T} x'(v).$$

On the other hand, Lemma 5 shows that

$$\sum_{T \in \mathcal{T}_A} \prod_{\text{nodes } v \in T} x'(v) = \frac{x(A)}{1 - x(A)} \sum_{T \in \mathcal{T}_A} p_T.$$

Since each Galton-Watson process produces at most one finite-length tree per process, $\sum_{T \in \mathcal{T}_A} p_T = \Pr[\text{G.W. process goes extinct}] \leq 1$. Hence, putting together these inequalities, yields

$$\mathbf{E}[N_A] \leq \frac{x(A)}{1 - x(A)} \sum_{T \in \mathcal{T}_A} p_T \leq \frac{x(A)}{1 - x(A)},$$

and the theorem follows by summing up over all possible root events $A \in \mathcal{A}$. □

References

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

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

[3] 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.

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

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

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

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

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