The Lovász Local Lemma : A constructive proof

Andrew Li

19 May 2016

Abstract

The Lovász Local Lemma is a tool used to non-constructively prove existence of combinatorial objects meeting a certain conditions. Herein we prove a constructive variant of the lemma which improves upon work started by József Beck in 1991 by improving performance and relaxing restrictions, following the paper [2] of Moser and Tardos closely, then briefly discuss the applications of the lemma to Ramsey Theory.

Contents

1 Historical Background

Let $\mathcal{A}$ be a finite collection of mutually independent events in a probability space. The probability none of these events happen is exactly the product of the probabilities that each event does not occur $\prod_{A \in \mathcal{A}}(1-\Pr[A])$, and this probability is positive when no event in $\mathcal{A}$ has probability 1. The Lovász Local Lemma allows for limited dependence among the events, but still concludes that none of the events occur with positive probability if the independent events have bounded probability. In other words, we can relax our constraints so that if the events are mostly independent and they aren't too likely, there is still a some chance none of them occur. So first, a formal statement of the lemma:

Theorem 1.1 (Erdõs and Lovász [1975]) *Let $\mathcal{A}$ be a finite collection of events in a probability space. For $A \in \mathcal{A}$, let $\Gamma(A)$ be a subset of $\mathcal{A}$ satisfying that A is independent from the collection of events $\mathcal{A} \backslash (\{A\} \bigcup \Gamma(A))$. If there exists an assigment of reals $x : \mathcal{A} \to (0, 1)$ such that*

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

then the probability of avoiding all events in $\mathcal{A}$ is at least $\prod_{A \in \mathcal{A}}(1 - x(A))$, in particular it is positive.

The proof offered by Erdõs and Lovász was non-constructive, and did not offer a procedure to find the set with the desired property. In 1991, Beck formulated a strategy in terms of hypergraph 2-coloring, proving that if a hypergraph had k vertices in each edge and shared common vertices with no more than about $2^{\frac{k}{48}}$ other edges, then a polynomial time algorithm can 2-color the vertices without producing a monochromatic edge. The Local Lemma allows for every edge to share vertices with about $\frac{2^k}{e}$ other edges and guarantees such a coloring. To improve on this gap, several authors have made contributions: Alon [1991] improved the threshold to $2^{\frac{k}{8}}$ using a simpler and randomized variant of Beck's algorithm. Srinivasan [2008] lowered this further to $2^{\frac{k}{4}}$. Then, Moser achieved $2^{\frac{k}{2}}$ in 2008, and $\frac{2^k}{32}$ in 2009. We will focus on an improvement by Moser and Tardos to the 2009 result - closing to the tight bound so as to apply to almost all applications of the Lovasz Local Lemma known so far. We will have to impose the restriction that we consider events determined by different subsets of underlying mutually independent random variables and $\Gamma(A)$ consists of all events that depend on some of the same variables as A, but these assumptions seems to be the case in almost all known applications of the lemma.

2 Setup and Statement of Algorithm(s)

2.1 Terminology

First, we define a hypergraph - a hypergraph is a graph (collection of vertices/nodes and edges) where edges can connect/contain an arbitrary number of vertices. A hypergraph is k-uniform if each edge contains precisely k vertices. A hypergraph is n-colorable if we can assign one of n colors to each vertex in V such that no edge is monochromatic. For more information, refer to [1] page 6. Two events are probabilistically independent if the probability of A given B is the probability of A - $\Pr(A|B) = \Pr(A)$. A probability space is the set of possible outcomes from a collection of events that have certain probability assigned to them. A neighborhood around a point A in a graph is the set of nodes S adjacent A in the graph (such that an edge exists between each $S' \in S$ and A), and an inclusive neighborhood is this set of nodes including A. We will use standard notation $\binom{i}{u}$ to mean i choose u, or $\dfrac{i!}{(i-u)!u!}$.

2.2 A simple algorithm

Let $\mathcal{P}$ be a finite collection of mutually independent random vairables in a fixed probability space Ω. We consider events A which are determined by the values of some subset $S \subseteq \mathcal{P}$ of these variables. If an evaluation of the variables in S makes A happen, then S *violates* A i.e. it is a 'bad' event that violates the constraints we have placed on our problem. Now, given that some collection of variables in $\mathcal{P}$ determines A, there is some unique minimal subset $S \subseteq \mathcal{P}$ which determines A - call this $\text{vbl}(A)$. Now, let $G = G_{\mathcal{A}}$ be a dependency graph for $\mathcal{A}$ with an edge between events $A, B \in \mathcal{A}$ if $A \neq B$, but $\text{vbl}(A) \bigcap \text{vbl}(B) \neq \emptyset$. Thus, this means that event A and B depend on some of the same variables. For $A \in \mathcal{A}$ we write $\Gamma(A) = \Gamma_{\mathcal{A}}(A)$ for the neighborhood of A in G - $\Gamma(A)$ will be (as previously noted) restricted to the events of $\mathcal{A}$ which are connected to A/dependent on the some of the same variables. This also satisfies the conditions stated in Theorem 1.1 - $\mathcal{A} \backslash (\{A\} \bigcup \Gamma(A))$ are not determined by the variables which determine A by definition of our set.

Now, for the algorithm: Start with a random point in Ω and maintain an evaluation v_P of each $P \in \mathcal{P}$. We check whether some even in $\mathcal{A}$ is violated. If so, choose an arbitrary violated event A and sample another random assignment of values for the variables in $vbl(A)$ on which A depends independently on each variable and without modifying the other variables in $\mathcal{P}$. We will refer to this as *resampling* of the event A.

Algorithm 1 Simple Sequential Solver

```
1: procedure SEQUENTIAL($\mathcal{P}$, $\mathcal{A}$)
2:     for all $P \in \mathcal{P}$ do
3:         $v_P \leftarrow$ a random evaluation of $P$;
4:     while $\exists A \in \mathcal{A} : A$ is violated when $(P = v_p : \forall P \in \mathcal{P})$ do
5:         pick an arbitrary violated event $A \in \mathcal{A}$;
6:         for all $P \in \text{vbl}(A)$ do
7:             $v_P \leftarrow$ a new random evaluation of $P$;
       return $(v_P)_{P \in \mathcal{P}}$
```

2.3 A parallel algorithm

This algorithm can be parallelized to get a better bound:

Algorithm 2 Parallel Solver

```
1: procedure PARALLEL($\mathcal{P}$, $\mathcal{A}$)
2:     for all $P \in \mathcal{P}$ do in parallel
3:         $v_P \leftarrow$ a random evaluation of $P$;
4:     while $\exists A \in \mathcal{A} : A$ is violated when $(P = v_p : \forall P \in \mathcal{P})$ do
5:         $S \leftarrow$ a maximal independent set in the subgraph of $G_{\mathcal{A}}$ induced by all
6:                 events which are violated when $(P = v_p : \forall P \in \mathcal{P})$,
7:                 constructed in parallel
8:         for all $P \in \text{vbl}(A)$ do in parallel
9:             $v_P \leftarrow$ a new random evaluation of $P$;
       return $(v_P)_{P \in \mathcal{P}}$
```

In particular, we select a maximal independent set S in the subgraph of the dependency graph G spanned by the violated events and resample all the variables these events depend on in parallel. In other words, we take independent new samples of the variables in $\bigcup_{A \in S} \text{vbl}(A)$ and leave the rest of the variables as they were, and continue until we find an evaluation with no violations.

3 Analysis of Algorithms

3.1 Sequential Analysis

First, we claim that the runtime is efficient.

Theorem 3.1 *Let $\mathcal{P}$ be a finite set of mutually independent random variables in a probability space. Let $\mathcal{A}$ be a finite set of events determined by these variables. If there exists an assignment of reals $x : \mathcal{A} \rightarrow (0, 1)$ such that*

$$\forall A \in \mathcal{A} : \Pr[A] \leq x(A) \prod_{B \in \Gamma_{\mathcal{A}}(A)} (1 - x(B)),$$

there there exists an assignment of values to the variables $\mathcal{P}$ not violating any of the events in $\mathcal{A}$. Moreover, the randomized algorithm described above resamples an event $A \in \mathcal{A}$ at most an expected $\dfrac{x(A)}{1-x(A)}$ times before it finds such an evaluation. Thus, the expected total number of resampling steps is at most $\sum_{A \in \mathcal{A}} \dfrac{x(A)}{1-x(A)}$.

First, notice that the decision that violates $A \in \mathcal{A}$ that we correct in each step can be taken completely arbitrarily - choose some procedure to choose this selection. This will not matter for our analysis. Next, we want to keep a log of the execution $C : \mathbb{N} \to \mathcal{A}$, which lists the events as they have been selected for resampling in each step. If the algorithm terminates, C is partial and defined only up to the given total number of steps carried out. Our other definitions/assignments carry over from the previous part.

A *witness tree* $\tau = (T, \sigma_T)$ is a finite rooted tree T together with a labelling $\sigma_T : V(T) \to \mathcal{A}$ of all its vertices with events such that the children of a vertex $u \in V(T)$ receive labels from the inclusive neighborhood $\Gamma^+(\sigma_T(u))$. We consider a witness tree proper if distinct children of the same vertex have distinct labels. Let $[v] := \sigma_T(v)$ to shorten notation. Now, to construct a witness tree from the log C, let us define $\tau_C^{(t)}(T)$ to be an isolated root vertex labelled $C(t)$, the tth resampling step. Now, we iterate over $i = t-1, t-2, ...$ and we have two cases. If there is a vertex $v \in \tau_C^{(i+1)}(t)$ such that $C(i) \in \Gamma^+([v])$, then we choose the vertex with a maximum distance from the root and attach a new child vertex u to v that we label $C(i)$, breaking ties arbitrarily. If there is no vertex such taht $C(i) \in \Gamma^+([v])$, then we simply skip step i. Now let this tree be $\tau_C^{(i)}(t)$ and repeat. The witness tree τ occurs in the log C if there exists $T \in \mathbb{N}$ such that $\tau_{C(t)} = \tau$.

Lemma 3.2 *Let τ be a fixed witness tree and C the (random) log produced by the algorithm. We make two claims: If τ occurs in C, the τ is proper, and that the probability that τ appears in C is at most $\prod_{v \in V(\tau)} \Pr[[v]]$.*

Proof Assume τ occurs in the log C, so we have $\tau_C(t) = \tau$ for some $t \in \mathbb{N}$. For a vertex $v \in V(\tau)$ let $d(v)$ denote the depth of vertex v which is the distance from the root, and let $q(v)$ stand for the step of the algorithm constructing $\tau_C(t)$ in which v was attached, the largest value q with v contained in $\tau_C^{(q)}(t)$. First, notice that $q(u) < q(v)$ for $u, v \in V(\tau)$ and $\text{vbl}([u])$ and $\text{vbl}([v])$ are not disjoint, then $d(u) > d(v)$. When adding the vertex u to $\tau_C^{q(u)+1)}(t)$ we attach it to v or to another vertex of equal or greater depth. Thus, for two vertices $u, v \in V(\tau)$ with $d(u) = d(v)$, $[u]$ and $[v]$ do not depend on any common variables. The labels in every level of τ form an independent set in G, and τ must be proper.

Then, define the procedure τ-check: In an order of decreasing depth, visit the vertices of τ and for a vertex v take a random evaluation of the veriables in $\text{vbl}([v])$, and check if the resulting evaluation violates $[v]$. We say that the τ-check passes if all events were violated when checked.

The τ-check passes with probability $\prod_{v \in V(\tau)} \Pr[[v]]$. Now, we show that the lemma follows since τ occurs in the log and we run the τ-check on the same random source it passes. We assume that for each variable $P \in \mathcal{P}$ we randomly generate a list of independent random samples which we pop values off of when we need.

Now we want to show that the τ-check passes. Let us fix the vertex v and for $P \in \mathrm{vbl}([v])$ let $S(P)$ be the set of vertices $w \in V(\tau)$ with $d(w) > d(v)$ and $P \in \mathrm{vbl}([w])$. When the τ-check considers v it had sampled P exactly when it was considering the vertices in $S(P)$.

In step $q(v)$, our algorithm chooses the event $[v]$ for resampling, so $[v]$ must be violated before this resampling. We claim that for $P \in \mathrm{vbl}([v])$ the current value of the variable P is $P^{(|S(P)|)}$ at this time. P was sampled at the beginning of the algorithm and then at the steps $q(w) < q(v)$ for $w \in S(P)$. At the τ-check has these exact same values for the variables in $\mathrm{vbl}([v])$ when considering v it also must find that $[v]$ is violated. So, the second claim of the lemma is also true. ∎

Now, let C be the log of the execution of our algorithm. Let N_A be the random variable that counts how many times the event A is resampled during the execution of our algorithm, the number of time step t with $C(t) = A$. Let t_i denote the i-th time step, and $\mathcal{T}_a$ denote the set of all proper witness trees having the root labelled A. By the definition, $\tau_C(t_i) \in \mathcal{T}_A$ for all i, and since it occurs in $\mathcal{T}_A$, the tree $\tau_C(t_i)$ contains exactly i vertices labelled A and $t_C(t_i) \neq t_C(t_j)$ unless $i = j$. So, N_A also counts the number of distinct proper witness trees occurring in C that have their root labeled A, so

$$N_A = \sum_{\tau \in \mathcal{T}_A} 1_{\tau \text{ occurs in } C}.$$

Thus, the expectation of N_A is bounded by the sum of the bound in lemma 3.2 on the probabilites of the occurrence of these witness trees.

3.2 Parallel Analysis

Consider some arbitrary execution of the parallel algorithm, i.e. we choose an arbitrary ordering of the violated evenets which are being resampled and do these sequentially. This is an execution of the sequential algorithm. Let S_j be the segment of the log C of this execution that corresponds to resamplings done in step j of the parallel algorithm. We call this the maximal depth of a vertex in a witness tree the *depth* of the tree.

Lemma 3.3 *If $t \in S_j$, then the depth of $\tau_C(t)$ is $j - 1$.*

Proof Let t_k be the first number in the segment S_k and let $\tau_k = \tau_C^{t_k}(t)$ for $k \leq j$. As the events resampled in the jth parallel step are independent, the root is the only vertes of τ_j. For $k < j$ we obtain τ_k from τ_{k+1} by attaching some vertices corresponding to the kth parallel step of the lagorithm. As these vertices have independent labels, they can only add one to the depth. Now, we show they must add one to the depth - consider a vertex v of τ_{k+1} of maximal depth. This vertex corresponds to a resampling of the event $[v]$ some time after step k of the parallel algorithm. If τ_k has no vertex with higher depth than v, then from the parallel step k to the resampling corresponing to v no event from the inclusive neighborhood (the neighborhood and the point itself) of $[v]$ was resampled. But then, this is a contradiction since this implies that $[v]$ was already violated at parallel step k and we did not select a maximal independent set of violated events there for resampling. Finally, notice $\tau_C(t) = \tau_1$. ∎

Now, we wish to prove the following theorem regarding our parallel algorithm

Theorem 3.4 *Let $\mathcal{P}$ be a finite set of mutually independent random variables in a probability space. Let $\mathcal{A}$ be a finite set of events determined by these variables. If $\epsilon > 0$ and there exists an assignment of reals $x : \mathcal{A} \to (0,1)$ such that*

$$\forall A \in \mathcal{A} : \Pr[A] \le (1-\epsilon)x(A) \prod_{B \in \Gamma_{\mathcal{A}}(A)} (1 - x(B)),$$

then the parallel version of our algorithm takes an expected $\mathcal{O}\left(\frac{1}{\epsilon}\log \sum_{A\in\mathcal{A}} \frac{x(A)}{1-x(A)}\right)$ steps before it finds an evaluation violating no event in $\mathcal{A}$.

Proof Let $Q(k)$ denote the probability that the parallel algorithm makes at least k steps. By Lemma 3.3, some witness tree of depth $k-1$ must occur in the log in this case, and that this witness tree has at least k vertices. Let $\mathcal{T}_A(k)$ be the set of witness trees in $\mathcal{T}_A$ having at least k vertices, and let $x'(B) := x(B)\prod_{C\in\Gamma(B)}(1-x(C))$. We have

$$\begin{aligned}
Q(k) &\le \sum_{A\in\mathcal{A}} \sum_{\tau\in\mathcal{T}_A(k)} \prod_{v\in V(\tau)} \Pr[\tau \text{ appears in the log} C] \\
&\le \sum_{A\in\mathcal{A}} \sum_{\tau\in\mathcal{T}_A(k)} \prod_{v\in V(\tau)} \Pr[[v]] \\
&\le (1-\epsilon)^k \sum_{A\in\mathcal{A}} \sum_{\tau\in\mathcal{T}_A(k)} \prod_{v\in V(\tau)} x'([v]).
\end{aligned}$$

The last inequality follows from the assumption made in the theorem. So, we see

$$\begin{aligned}Q(k) &\leq (1-\epsilon)^k \sum_{A\in\mathcal{A}} \sum_{\tau\in\mathcal{T}_A(k)} \prod_{v\in V(\tau)} x'\left([v]\right)\\ &\leq (1-\epsilon)^k \sum_{A\in\mathcal{A}} \frac{x(A)}{1-x(A)} \sum_{\tau\in\mathcal{T}_A(k)} p_\tau\\ &\leq (1-\epsilon)^k \sum_{A\in\mathcal{A}} \frac{x(A)}{1-x(A)},\end{aligned}$$

which implies the bound in the theorem. ▮

4 Concluding Remarks

A particularly intriguing use of the Lovász Local Lemma is it's application to finding lower bounds for Ramsey numbers. Suppose we wished to find the Ramsey number $R(k,3)$, the minimum number of nodes needed such that all fully connected graphs contain a clique of order k or an independent set order 3. Put more visually, suppose we color all edges either red or blue. We want there to be k nodes which are connected by only red edges, or 3 nodes which are connected by only blue edges (or vice versa, the problem is symmetric). For a graph large enough, this is guaranteed to happen. Then, we can use the Lovász Local Lemma to make a probabilistic argument for a lower bound - suppose we assign colors to each edge of our graph randomly and independently, where each edge is blue with probability p. For each set of 3 vertices T, let A_T be the event that the triangle on T is blue, and for each set of K vertices S, let B_S be the event that the complete graph on S is red. We see $\Pr(A_T) = p^3$, and $\Pr(B_S) = (1-p)^{\binom{k}{2}}$. Then, we produce a dependency graph for A_T and B_S by joining two vertices with an edge iff the corresponding complete graphs share an edge. Each A_T node is adjacent to $3(n-3) < 3n$ A_T nodes and to at most $\binom{n}{k}$ B_S nodes. Each B_S node is adjacent to $\binom{k}{2}(n-k) < \frac{k^2n}{2}$ A_T nodes and to at most $\binom{n}{k}$ B_S nodes. So, we have from the Lovász Local Lemma that, given $0 < p < 1$ and $0 \leq x, y < 1$,

$$p^3 \leq x(1-x)^{3n}(1-y)^{\binom{n}{k}}$$

and

$$(1-p)^{\binom{k}{2}} \leq y(1-x)^{\frac{k^2n}{2}}(1-y)^{\binom{n}{k}}$$

would together imply that $R(k,3) > n$, which is what we want to show. So, to find the largest possible $k = k(n)$ for which we can make such a choice of p, x, y, we can eventually find a lower bound of $\frac{ck^2}{\log^2(k)}$, which was the best lower bound shown until 1995 when it was improved to $\frac{ck^2}{\log(k)}$. The same argument for $R(k,4)$ is better than any bound proven without the lemma.

References

[1] Noga Alon, Joel H. Spencer *The Probabilistic Method.* Second edition, (March 2000). Tel Aviv and New York.

[2] Robin A. Moser, Gábor Tardos *A Constructive Proof of the General Lovász Local Lemma.* J. ACM 57, 2, Article 11, (January 2010), 15 pages.