The Constructive Lovasz Local Lemma

1 Motivation

Assume there is a finite set $\mathcal{P} = \{P_i\}$ of **independent** random variables (not necessarily binary), and for each i there is an efficient procedure for sampling P_i. We are explicitly given a bipartite graph between events $\mathcal{E}_j$ and variables P_i such that $\mathcal{E}_j$ is mutually independent of all variables it does not have an edge to. Write $L(\mathcal{E}_j)$ for the P_i that $\mathcal{E}_j$ has an edge to. For the dependency graph, $(\mathcal{E}_i, \mathcal{E}_j)$ exists only if $L(\mathcal{E}_i) \cap L(\mathcal{E}_j) \neq \emptyset$.

The Moser-Tardos Algorithm is quite simple: For each $P \in \mathcal{P}$, let y_P be a random draw of P (independent of all $y_{P'}$). While there is some $\mathcal{E}_j$ that is true under $\boldsymbol{y}$, re-draw all $P \in L(\mathcal{E}_j)$.

Theorem 1. *(Moser-Tardos 2009).* Assume that there exists x_i with

$$\Pr[\mathcal{E}_i] \leq x_i \prod_{(i,j)\in E} (1 - x_j)$$

for all i. Then this algorithm finds an assignment $\boldsymbol{y}$ satisfying $\bigcap \bar{\mathcal{E}}_j$ and uses at most

$$\sum_j \frac{x_j}{1 - x_j}$$

iterations in expectation. More specifically, each $\mathcal{E}_j$ is resampled at most $\frac{x_j}{1-x_j}$ times in expectation.

2 Execution Logs and Witness Trees

2.1 Definitions

The **execution log** C has $C(t)$ the event that $\mathcal{E}_j$ is resampled in step t. Resampling was necessary because $\mathcal{E}_j$ was true under $\boldsymbol{y}_t$. In the independent case, this was because the previous time we sampled $L(\mathcal{E}_j)$, we made $\mathcal{E}_j$ true). In the general case this is more subtle because resampling one or more $\mathcal{E}_i$ with $(j, i) \in E$ may together have made $\mathcal{E}_j$ true.

We want a structure keeping track of the "blame" for resampling. We define a **witness tree** T_t for time t as follows: The root of T_t is labeled $\lambda(r) = \mathcal{E}_j$. We iterate through $t' = t-1, t-2, \ldots$ and consider the event $\mathcal{E}_i = C(t')$ that happened at t' in C.

In the first case, there is no edge $(i', i) \in E$ for any $\mathcal{E}_{i'}$ that is (so far) a label in T_t. Then, $\mathcal{E}_i$ cannot have caused any of the resampling in T_t, so we skip $\mathcal{E}_i$.

In the second case, there is a node $v \in T_t$ with label $\lambda(v) = \mathcal{E}_{i'}$ such that $(i', i) \in E$. Choose the deepest such v (breaking ties arbitrarily), create a new node u with $\lambda(u) = \mathcal{E}_i$, and make it a child of v.

2.2 Tree Check Coupling

Fact 2. If u, v are at the same level of T, then $L(\lambda(u)) \cap L(\lambda(v)) = \emptyset$.

Proof. Consider the times t_u and t_v; without loss of generality $t_u < t_v$ when $\mathcal{E}_i$ and $\mathcal{E}_{i'}$ were resampled for the nodes u, v. Then in the construction of T, it was an option to attach u to v. Because u was attached as deep as possible, it cannot be at the same level as v. □

We call a tree T a **proper** witness tree if it satisfies this condition. Sanity check: what does the witness tree look like for an empty independence graph? Well, since all variable sets are disjoint, each T_t is a chain of nodes all labeled $\mathcal{E}_j = \lambda(r) = C(t)$. The chain has length equal to the number of times $\mathcal{E}_j$ was resampled in the first t steps.

More generally, the tree for time t has exactly as many nodes labeled $C(t)$ as the number of times $C(t)$ was resampled up to time t. This implies that each proper witness tree occurs at most once in the log. That is, the set of T_t for $t = 0, 1, \ldots$ has no two identical (including labels) trees.

Lemma 3. Let T be a proper witness tree. The probability that T occurs in C is at most $\prod_{v \in T} \Pr[\lambda(v)]$.

Proof. The idea is to couple the execution of the algorithm with a "tree check": making sure that all events $\lambda(v)$ happen. The tree check for each $v \in T$ samples all $P \in L(\lambda(v))$ independently and checks if $\lambda(v)$ is true. The check is passed iff all $\lambda(v)$ are true. Clearly, the probability of passing the tree check is $\prod_{v \in T} \Pr[\lambda(v)]$.

To do a coupling, consider the tree from lowest level to the root, sampling the variables in that order. (Note there are no ties because the tree is proper). To make the coupling precise, for each variable $P \in \mathcal{P}$ we have a sequence $P^{(1)}, P^{(2)}, \ldots$ of values of P drawn independently. When the tree check needs to sample P for the i^{th} time, it uses $P^{(i)}$. When Moser-Tardos needs to sample P for the i^{th} time, it also uses $P^{(i)}$.

Consider vertex v with $\lambda(v) = \mathcal{E}_j$ occurring at time t. Then $C(t) = \mathcal{E}_j$. Let $P \in L(\mathcal{E}_j)$ be a variable of $\mathcal{E}_j$ and m be the number of timesteps $t' < t$ such that $P \in L(C(t'))$ – that is, the number of times P was resampled up to time t.

Then, Moser-Tardos at time t has $P = P^{(m+1)}$ because the first sample is "free" and we resampled m times. All of these resamplings must correspond to nodes u strictly below v in the witness tree because they occurred earlier – the witness tree had the option to attach them to v.

So, the tree check algorithm also sampled P earlier m times, and is using $P^{(m+1)}$ to evaluate v. This applies to all $P \in L(\mathcal{E}_j)$ and all vertices $v \in T$. Because all $\mathcal{E}_j$ are true in Moser-Tardos (otherwise the resampling would not have happened), the probability that T occurs in C is at most the probability that T passes the tree check. □

2.3 Galton-Watson Coupling

Write $x'_j \coloneqq x_j \prod_{(i,j) \in E} (1 - x_i)$. Let $\mathcal{T}_j$ be the set of all proper witness trees whose root is labeled $\lambda(r) = \mathcal{E}_j$. Also, let N_j be the random variable for the number of times $L(\mathcal{E}_j)$ is resampled in Moser-Tardos.

Because each resampling corresponds to a tree with root label $\mathcal{E}_j$ and the same tree cannot appear multiple times,

$$\begin{aligned}
\mathbb{E}[N_j] &= \sum_{T \in \mathcal{T}_j} \Pr[T \text{ appears in log}] \\
&\leq \sum_{T \in \mathcal{T}_j} \prod_{v \in T} \Pr[\lambda(v)] && \text{Lemma 3} \\
&\leq \sum_{T \in \mathcal{T}_j} \prod_{v \in T} x'_{\lambda(v)}. && \text{MT Assumption}
\end{aligned}$$

Our goal now is to bound $\prod_{v \in T} x'_{\lambda(v)}$ for a given tree T. We will define a probability distribution p_T over labeled trees T (finite and infinite, including some not in $\mathcal{T}_j$) such that

$$\prod_{v \in T} x'_{\lambda(v)} = \frac{x_j}{1 - x_j} p_T.$$

Then we obtain

$$\begin{aligned}
\sum_{T \in \mathcal{T}_j} \prod_{v \in T} x'_{\lambda(v)} &= \sum_{T \in \mathcal{T}_j} \frac{x_j}{1 - x_j} p_T \\
&= \frac{x_j}{1 - x_j} \sum_{T \in \mathcal{T}_j} p_T \\
&\leq \frac{x_j}{1 - x_j}
\end{aligned}$$

because the distribution generates some trees not in $\mathcal{T}_j$. Thus

$$\mathbb{E}[N_j] \leq \frac{x_j}{1 - x_j},$$

proving the theorem.

We define a generative process for trees as follows:

1. Start with a root labeled $\mathcal{E}_j$
2. For all levels $\ell = 0, 1, 2, \ldots$ and for all nodes v at level ℓ and for all events $\mathcal{E}_i$ with $\mathcal{E}_i = \lambda(v)$ or $(\mathcal{E}_i, \lambda(v)) \in E$: independently with probability x_i give v a child u with $\lambda(u) = \mathcal{E}_i$; otherwise give v no child with label $\mathcal{E}_i$.

This is called a Galton-Watson branching process. It may die out after a finite number of levels, or build infinite trees. It induces a distribution p_T over (finite and infinite) labeled trees.

Lemma 4. For every finite labeled tree,

$$p_T = \frac{1 - x_j}{x_j} \prod_{v \in T} x'_{\lambda(v)}.$$

Proof. Fix a finite labeled tree T in which no node has two children of the same label. For any vertex $v \in T$, let

$$W_v \coloneqq \{i : \lambda(v) = \mathcal{E}_i \text{ or } (\lambda(v), \mathcal{E}_i) \in E \text{ but } v \text{ has no child labeled } i\}.$$

This is the set of events that v could have had as child labels, but didn't. Then,

$$\begin{aligned}
p_T &= \prod_{v \neq r} x_{\lambda(v)} \prod_v \prod_{i \in W_v} (1 - x_i) \\
&= \frac{1}{x_j} \prod_v \big(x_{\lambda(v)} \prod_{i \in W_v} (1 - x_i) \big).
\end{aligned}$$

We multiply/divide the missing $(1 - x_i)$ terms, so

$$p_T = \frac{1}{x_j} \cdot \prod_v \Big(x_{\lambda(v)} \cdot \prod_{\mathcal{E}_i : \mathcal{E}_i = \lambda(v) \text{ or } (\mathcal{E}_i, \lambda(v)) \in E} (1 - x_i) \cdot \frac{1}{1 - x_{\lambda(v)}} \Big) \cdot (1 - x_j)$$

Essentially we added a $(1-x_i)$ for each of the children, so the final term in the inner product cancels exactly one added term. The root is not a child of anything, so the final term overall accounts for it. Thus,

$$\begin{aligned} p_T &= \frac{1-x_j}{x_j} \prod_v \Big(\frac{x_{\lambda(v)}}{1-x_{\lambda(v)}} \cdot (1-x_{\lambda(v)}) \cdot \prod_{i:(\mathcal{E}_i,\lambda(v))\in E} (1-x_i) \Big) \\ &= \frac{1-x_j}{x_j} \prod_v \Big(x_{\lambda(v)} \prod_{i:(\mathcal{E}_i,\lambda(v))\in E} (1-x_i) \Big) \\ &= \frac{1-x_j}{x_j} \prod_v x'_{\lambda(v)}. \end{aligned}$$

□