LECTURE 2:

LOVÁSZ LOCAL LEMMA

H. Xiao AUGUST 2012

2.1 Recap: pairwise independence and mutual independence

Definition 2.1 (Pairwise independence) *A collection of events $\{A_i : i \in I\}$ is pairwise independent if for every pair $i \neq j \in I$,*

$$\Pr[A_i \cap A_j] = \Pr[A_i] \cdot \Pr[A_j].$$

Note that pairwise independence is equivalent to $\Pr[A_i \mid A_j] = \Pr[A_i]$ for all $j \neq i$ assuming that $A_j \neq \emptyset$.

Definition 2.2 (Mutual independence) *A collection of events $\{A_i : i \in I\}$ is mutually independent if for all subsets $S \subset I$,*

$$\Pr[\cap_{i \in S} A_i] = \prod_{i \in S} \Pr[A_i]$$

Similar to pairwise independence, mutually independence is equivalent to

$$\forall i \in I, \forall S \subset I \backslash \{i\} : \quad \Pr[A_i \mid \cap_{j \in S} A_j] = \Pr[A_i],$$

again assuming that the event we're considering has non-zero probability.

Example: Three random variables which are pairwise independent, but not mutually independent. Let $X = \{0,1\}$ and $Y = \{0,1\}$ be (pairwise) independent variables taking 0/1 with probability 1/2. Then let $Z = X \oplus Y$ (i.e., X"xor"Y). Then, the random variables $\{X, Y, Z\}$ are pairwise independent, but not mutually independent (knowing the outcome of any two random variables determines the third one).

2.2 Lovász Local Lemma

Let $\mathcal{A} := \{A_1, A_2, \ldots, A_n\}$ be a collection of "bad" events in a probability space whose occurrences render the failure of an experiment. So we are interested in the probability $\Pr[\cap_{i=1}^n \bar{A}_i]$ that none of these "bad" events occurs. If the probability is positive, we can conclude that the experiment will succeed. Assume that $\Pr[A_i] \leq p_i < 1$. Typically, we have two ways to bound this probability. First, when events A_i are mutually independent, the probability that none of them happens is

$$\Pr[\cap_{i=1}^n \bar{A}_i] \geq \prod_{i=1}^n (1 - p_i).$$

When there is no assumption about the dependency among events, we can use the union bound,

$$\Pr[\cap_{i=1}^n \bar{A}_i] = 1 - \Pr[\cup_{i=1}^n A_i] \geq 1 - \sum_{i=1}^n \Pr[A_i] \geq 1 - \sum_{i=1}^n p_i.$$

(Thus, we can conclude that (i) If $A_1, A_2, \ldots, A_n$ are mutually independent and all $\Pr[A_i] < 1$, then $\Pr[\cap_{i=1}^n \bar{A}_i] > 0$; (ii) If $\sum_{i=1}^n \Pr[A_i] < 1$, then $\Pr[\cap_{i=1}^n \bar{A}_i] > 0$.)

Usually, however, the events under consideration are not mutually independent. But all is not lost. The Lovász local lemma generalizes the argument to the case that events are not mutually independent but the dependency is limited. Erdös and Lovósz (1975) showed that the probability $\Pr[\cap_{i=1}^n \bar{A}_i]$ will still be positive provided that the events A_i occur with low probability and are "almost independent". To this end, we make the following definition.

Definition 2.3 (Dependency Graph) *Let $\mathcal{A} := \{A_1, A_2, \ldots, A_n\}$ be a collection of events in a probability space Ω. The dependency graph of $\mathcal{A}$ is a graph $G_{\mathcal{A}}$ on vertex set $\mathcal{A}$ with edge between A_i and A_j iff A_i and A_j are dependent. Let $\Gamma(A_i)$ denote the neighbors of A_i in $G_{\mathcal{A}}$.*

(Generally, we can construct the dependency graph as follows: connect i and j iff A_i and A_j are dependent.)Note that it is not generally normal, but it works well in practice.

Note that the dependency graph is not unique.

Theorem 2.4 (Lovász Local Lemma, general version) *Let $A := \{A_1, A_2, \ldots, A_n\}$ be a collection of "bad" events in a probability space with dependency graph $G_{\mathcal{A}}$. If there exists an assignment 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

$$\Pr[\cap_{i=1}^n \bar{A}_i] \geq \prod_{i=1}^n (1 - x(A_i)) > 0.$$

$$\Pr[\cap_{i=1}^n \bar{A}_i] \geq \prod_{A \in \mathcal{A}} (1 - x(A)) > 0.$$

Remarks. When the events $A_1, A_2, \ldots, A_n$ are mutually independent, the quantities x_i can be regarded as the "ideal probabilities" of events A_i. When the events A_i are not independent, we can think of $\prod_{j \in N(i)}(1-x_j)$ as a penalty for dependencies. Each "ideal probability" x_i is reduced by a "compensation factor" $\prod_{j \in N(i)}(1-x_j)$ according to the "ideal probabilities" x_j of the events upon which A_i is dependent; the larger the dependencies among these events, the smaller the individual probabilites for $A_1, A_2, \ldots, A_n$ have to be in order for the Lovász Local Lemma to apply.

Proof. We claim that for any subset S of indices from $\{1, 2, \ldots, n\}$, $i \notin S$, we have $\Pr[A_i | \cap_{j \in S} \bar{A}_j] \leq x_i$. Assuming the claim holds, the Lovász Local Lemma follows, since

$$\Pr[\cap_{i=1}^n \bar{A}_i] = \prod_{i=1}^n \Pr[\bar{A}_i \mid \cap_{j=1}^{i-1} \bar{A}_j] = \prod_{i=1}^n (1 - \Pr[A_i \mid \cap_{j=1}^{i-1} \bar{A}_j]) \geq \prod_{i=1}^n (1 - x_i) > 0.$$

Hence it remains to prove the claim. We prove it by induction on the size of S.

For simplicity, we denote the intersection $\cap_{i \in S} A_i$ by A_S. Using this notation, an event A_i is independent of a set of events $\{A_j : j \in I\}$ if, for all subsets S of I, $\Pr[A_i \cap A_S] = \Pr[A_i] \Pr[A_S]$.

For the case $S = \emptyset$, inequality $\Pr[A_i] \leq x_i \cdot \prod_{j \in N(i)}(1 - x_j) \leq x_i$ clearly holds.

Now consider the inductive step. Given a nonempty set S and $i \notin S$, let $S_1 := S \cap N(i) = \{j_1, j_2, \ldots, j_r\}$ and $S_2 := S \backslash S_1$. By definition of conditional probabilities,

$$\begin{aligned}
\Pr[A_i \mid \bar{A}_S] &= \frac{\Pr[A_i \cap \bar{A}_S]}{\Pr[\bar{A}_S]} \\
&= \frac{\Pr[A_i \cap \bar{A}_{S_1} \mid \bar{A}_{S_2}] \cdot \Pr[\bar{A}_{S_2}]}{\Pr[\bar{A}_{S_1} \mid \bar{A}_{S_2}] \cdot \Pr[\bar{A}_{S_2}]} \\
&= \frac{\Pr[A_i \cap \bar{A}_{S_1} \mid \bar{A}_{S_2}]}{\Pr[\bar{A}_{S_1} \mid \bar{A}_{S_2}]}.
\end{aligned}$$

The numerator of the above fraction is bounded from above by

$$\Pr[A_i \cap \bar{A}_{S_1} \mid \bar{A}_{S_2}] \leq \Pr[A_i \mid \bar{A}_{S_2}] = \Pr[A_i] \leq x_i \cdot \prod_{j \in N(i)} (1 - x_j),$$

where the equality follows from the fact that A_i is mutually independent of all the events $\{A_j : j \in S_2\}$.

The denominator of the above fraction can be bounded from below by using the induction hypothesis:

$$\begin{aligned} \Pr[\bar{A}_{S_1} \mid \bar{A}_{S_2}] &= \prod_{k=1}^{r} \Pr[\bar{A}_{j_k} \mid (\cap_{l=k+1}^{r} \bar{A}_{j_l}) \cap \bar{A}_{S_2}] \\ &= \prod_{k=1}^{r} (1 - \Pr[A_{j_k} \mid (\cap_{l=k+1}^{r} \bar{A}_{j_l}) \cap \bar{A}_{S_2}]) \\ &\geq \prod_{j \in S_1} (1 - x_j) \end{aligned}$$

Therefore,

$$\Pr[A_i \mid \bar{A}_S] \leq \frac{x_i \cdot \prod_{j \in N(i)} (1 - x_j)}{\prod_{j \in S_1} (1 - x_j)} \leq x_i,$$

completing the proof. ∎

For many applications, the following simpler version of the LLL, in which the probabilities of the events A_i have a common upper bound, is sufficient. (Here e denotes the base of natural logarithms.)

Theorem 2.5 (Lovász Local Lemma, symmetric version) *Let $\mathcal{A} := \{A_1, A_2, \ldots, A_n\}$ be a collection of "bad" events in a probability space with dependency graph $G_{\mathcal{A}}$ such that the maximum degree of $G_{\mathcal{A}}$ is d. If $\Pr[A] \leq 1/[e(d+1)]$ for all $A \in \mathcal{A}$, then $\Pr[\cap_{i=1}^{n} \bar{A}_i] > 0$.*

Remarks. e is best possible.

Proof. If $d = 0$, the result follows directly, since all the events $A_1, A_2, \ldots, A_n$ are mutually independent.

If $d > 0$, then $e^{-1} \leq (1 - \frac{1}{d+1})^d$. For $1 \leq i \leq n$, set $x_i := 1/(d+1)$ and we have

$$\Pr[A_i] \leq \frac{1}{e(d+1)} \leq \frac{1}{d+1}(1 - \frac{1}{d+1})^d \leq x_i \cdot \prod_{j \in N(i)} (1 - x_j).$$

Hence, we have $\Pr[\cap_{i=1}^{n} \bar{A}_i] > 0$. ∎

2.3 Constructive version of the lemma

As noted earlier, the LLL is in general nonconstructive. However, important work initiated by Beck [B91], and leading up to a dramatic recent breakthrough by Moser and Tardos [Mos09,MT10] provides a constructive version of the Local Lemma in quite general circumstances. Furthermore, the proof of the this new version gives an efficient (poly-time) randomized algorithm for finding the assignment as a byproduct.

Suppose the collection of "bad" events $\mathcal{A}$ is defined in terms of a set of mutually independent random variables $\Xi := \{\xi_1, \xi_2, \ldots, \xi_m\}$, where each event $A \in \mathcal{A}$ is determined by some subset of variables $vbl(A) \subseteq \Xi$. In this case, the dependency graph $G_{\mathcal{A}}$ of $\mathcal{A}$ is the graph on vertex set $\mathcal{A}$ with an edge between A and B iff $vbl(A) \cap vbl(B) \neq \emptyset$. Let $\Gamma(A)$ denote the set of neighbors of A in $G_{\mathcal{A}}$(the set of all events that share at least one variable with A) and let $\Gamma^+(A) = \{A\} \cup \Gamma(A)$. They attempt to find an assignment to the variables that makes none of the "bad" events occur using an local search algorithm.

The Moser-Tardos algorithm (henceforth MT-algorithm) is incredibly simple: Start with an arbitrary assignment and repeatedly resample all variables in $vbl(A)$ of any one of the given "bad" events that holds

currently. Attempting to "fix" such an event can lead to other neighboring events happening, but an elegant argument shows that a branching process dies out quickly if the original LLL-conditions (even with the optimal constants) are fulfilled. This yields a randomized algorithm with expected linear running time in the number of "bad" events.

M-T Algorithm
Sample all variables randomly
while $\exists A \in \mathcal{A}$ happens **do**
    Resample variables in $vbl(A)$
end

(It has been proved that the local-search algorithm terminates quickly on expectation if the conditions of the Local Lemma are satisfied.)

Formally, we have the constructive version of Lovász Local Lemma as follows.

Theorem 2.6 (Lovász Local Lemma, constructive version) *Let Ξ 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 $x : \mathcal{A} \to [0, 1)$ such that*

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

then there exists an assignment of values to the variables Ξ not violating any of the events in $\mathcal{A}$ after an expected number of at most $\sum_{A \in \mathcal{A}} \frac{x(A)}{1-x(A)}$ correction steps.

The constructive version of the LLL has an intuitive understanding: if each $A \in \mathcal{A}$ has a small enough neighborhood $\Gamma(A)$ and a low enough probability, then whenever we resample A's variables, it's likely that we "fix" A and unlikely that we "break" too many B in A's neighborhood.

2.4 Proof of the constructive version

((The actual proof uses a different idea that essentially looks at this process in reverse, looking for each resampled event A at **a set of previous events whose resampling we can blame for making A happen**, and then showing that **a tree which will include every resampling operation as one of its vertices** can't be too big.))

Proof overview

For a specific execution of the algorithm, we associate with the event resampled in each step a distinct tree. The same event in different execution steps are associated with distinct trees. Upper bound the number of resamplings for each event $A \in \mathcal{A}$ by bounding the number of distinct trees associated with event A. Then consider a branching process which can run out all potential trees and estimate the possible numbers of randomly generated trees that are associated with event $A \in \mathcal{A}$ in the execution of algorithm.

((Besides, under the LLL condition, we can show that the branching process can die out quickly on average.

We do that in two steps: first, upper bound the probability that a specific witness tree occurs in a random execution log produced by the algorithm; (bound (the sum of) the probability that any particular witness tree(explanation) appears,); The running time/complexity of the algorithm can be upper bounded by the possible number of the distinct witness trees. And each witenss tree can be generated by a branching process and branching process can run out all the potential witness trees. Hence, we can bound the distinct witness tress in an execution log L by bounding the possible number of distinct witness trees associated with a specific(the same) execution log. Furthermore, process will die out(Attempting to "fix" such an event can lead to other neighboring events happening, but an elegant argument shows that a branching process) quickly if the original LLL-conditions (even with the optimal constants) are fulfilled, implying that we can calculate it easily.))

Observing that if we can show that a good assignment can be obtained after each $A \in \mathcal{A}$ is resampled at most $\frac{x(A)}{1-x(A)}$ times on average under the LLL-condition, the lemma follows directly. That's exactly the direction we are working towards in this proof.

Execution log C

To analyze the algorithm, we need to record the sequence of events that are resampled during an execution. We consider an execution log $L : \mathbb{N} \to \mathcal{A}$, where $L(i)$ keeps track of the event whose variables are resampled in the i^{th} step of the execution. If the algorithm terminates, L is partial and defined only up to the given total number of steps carried out. Note that the execution log L is a *random* variable determined by the random choices the algorithm makes.

Witness tree

In a *specific* execution log L yielded by an execution of the algorithm, we associate with each entry $L(i)$ of execution log L a *distinct* witness tree, which "explains" the sequence of events recorded in the log, i.e., how we arrived at the need to correct A at the i^{th} step of execution.

Generation of witness trees from the execution log C

For each entry $L(i)$ in a *specific* execution log L, we construct a witness tree rooted at $L(i)$ as follows. Start initially with the event $L(i)$ as the root . Go backwards through the execution log L from $L(i)$ and iteratively add event $L(i-1)$ to the tree as follows. Insert $L(i-1)$ as a child of the deepest vertex in the current tree that shares variables with $L(i-1)$(That is adjacent to $L(i-1)$ in the dependency graph $G_{\mathcal{A}}$); If there are multiple vertices in the deepest level sharing variables with $L(i-1)$, pick one arbitrarily; If there is no vertex sharing variables with $L(i-1)$, ignore $L(i)$.

Note that a witness tree rooted at $L(i)$ "explains" the execution log L by including all previous events whose resampling lead to the resampling of $L(i)$ as vertices. Besides, a event might repeatedly occur in the witness tree since it might be resampled more than once.

Now the following lemma shows that each entry of execution log L corresponds to a distinct witness tree under the construction stated above.

Lemma 2.7 *Each entry corresponds to a distinct witness tree.*

Proof. Now we will show why different entries $C(i)$ corresponds to distinct witness trees. ... (Each resampling of A generates different witness tree. Because every event we resample is the root of some witness tree, and we can argue that every event we resample is the root of a distinct witness tree. Because we only discard events A_j that have $vbl(A_j)$ disjoint from all nodes already in the tree. So the witness tree rooted at the kth occurrence of A_i in log C will include exactly k copies of A_i.) ■

Since each entry of execution log L corresponds to a distinct witness tree, we could bound the length of execution log L by the possible numbers of distinct witness trees associated with all entries of the execution log L.

For each event $A \in \mathcal{A}$, denote by N_A the random variable that counts how many times the event A is resampled during an execution of algorithm. If L is the log of a specific execution of algorithm, then N_A is the number of occurrences of A in this log L and also the number of distinct witness trees associated with all A's in the log L. Denote by T_A the set of witness trees rooted at A in the *specific* execution log L and τ_A a particular witness trees in T_A. Denote by $\Pr[\tau_A]$ the probability that τ_A occurs in the random log L.

(Now we will count how many distinct witness tree τ_A rooted at A we can possibly have.)

Since each entry of log C corresponds to a distinct witness tree rooted at it, the expected number of event A occurring in a log L can be expressed as

$$E[N_A] = \sum_{\tau_A \in T_A} \Pr[\tau_A].$$

Then we bound the size of the log L by bounding the probability that any particular witness tree (explanation) appears.

Probability of τ occurs in log C

A particular witness tree **occurs** in the execution log L if there exists a entry $L(i)$ such that the witness tree is rooted at it and the tree can be generated according to the principles stated above in the execution log L.

But since we resample all the variables in A, any subsequent assignments to these variables are independent of the ones that contributed to A; with sufficient hand-waving (or a rather detailed coupling argument as found in [MT10]) this gives that each event A occurs with independent probability $\prod_{B\in\tau_A}\Pr[B]$.

Lemma 2.8 *Let τ_A be a particular witness tree and C a random execution log produced by the algorithm. The probability that τ_A appears in C is (at most) $\prod_{B\in\tau_A}\Pr[B]$*

(Given a specific tree, it might correspond to some entry of log L (associated with some entry of C or just saying that it occurs in the log L.) or might not. It does, it's a specific valid witness tree.)

Random generation of witness trees from the assignment x in LLL-condition

(Denote by q_τ the event that specific witness τ rooted at A obtained by the random generation of witness tree using the branching method.)

For each specific witness tree, consider a Galton-Watson branching process that attempts to construct it. We start initially with a single vertex and grow a tree as follows. For every vertex A in the deepest level of the tree, give it a child B, for each $B\in\Gamma^+(A)$ with (independent) probability $x(B)$, or skip it with probability $1-x(B)$. All these choices are independent. Repeat this process until no more vertex is (available) added to the tree. Note that this process can produce infinite trees.

(The process continues until it dies out naturally because no new vertices are born in some round (depending on the probabilities used, there is, of course, the possibility that this never happens).)(This process will either die out eventually, or will continue forever (extinction depends on means as we outlined above, which in turn depend on vertex probabilities). Our question will be: what is the probability that $wtree_{GW} = wtree$?)

Probability of τ_A yielded by the random generation

For each vertex B in τ_A, let $W(B)$ be the set of events $C\in\Gamma^+(B)$ that don't occur as children of B . The probability of getting τ_A is bounded from above by the product of the probabilities of each B of getting all of its children and none of its non-children. The non-children of B collectively contributes $\prod_{C\in W(B)}(1-x(C))$ and B itself contributes $x(B)$, unless B is the root. When B is the root of τ_A, it appears in τ_A with probability 1 instead of $x(B)$. Thus, the probability p_τ that the described Galton-Watson branching process yields a prescribed witness tree is

$$p_\tau = \frac{1}{x(A)}\prod_{B\in\tau_A}\{x(B)\cdot[\prod_{C\in W(B)}(1-x(C))]\}.$$

(Now the situation is the following: the tree $wtree_{GW}$ generated by the Galton-Watson process might grow infinite. Even if the Galton-Watson halts, then it might not generate any tree in T_A when started at A. Moreover, a single run of our Galton-Watson process produces one tree $wtree_{GW}$, which can therefore be identical to at most one tree in T_A. Thus $\sum_{wtree\in T_A}\Pr[wtree_{GW}=wtree]\le 1$)

Observe that

$$\begin{aligned}
p_\tau &= \frac{1}{x(A)}\prod_{B\in\tau_A}\{x(B)\cdot[\prod_{C\in W(B)}(1-x(C))]\}\\
&= \frac{1-x(A)}{x(A)}\prod_{B\in\tau_A}\{\frac{x(B)}{1-x(B)}\cdot[\prod_{C\in\Gamma^+(B)}(1-x(C))]\}\\
&= \frac{1-x(A)}{x(A)}\prod_{B\in\tau_A}\{x(B)\cdot[\prod_{C\in\Gamma(B)}(1-x(C))]\}
\end{aligned}$$

Conclusion

Comparing the above expression with the bound of the probability that appears in the execution log, we have

$$\begin{aligned} E(N_A) &= \sum_{\tau_A \in T_A} \Pr[\tau_A \text{ occurs in execution log}] \\ &= \sum_{\tau_A \in T_A} \prod_{B \in \tau_A} \Pr[B] \\ &\leq \sum_{\tau_A \in T_A} \prod_{B \in \tau_A} x(B) \prod_{C \in \Gamma(B)} (1 - x(C)) \\ &= \frac{x(A)}{1 - x(A)} \sum_{\tau_A \in T_A} p_{\tau_A}(\tau_A \text{ is generated by branching process}) \\ &\leq \frac{x(A)}{1 - x(A)} \end{aligned}$$

Summing over all $\mathcal{A}$, we get an upper bound on the expected running time of the algorithm.

Proof review For a particular witness tree, it can be generated by branching process with probability XXX; on the other hand, it occurs in a specific execution log C with probability no more than YYY!!!

Let N_A be the variable counting the number of witness trees τ_A in a specific execution log L. Let M_A be the variable counting the number of times running branching process to abtain a particular witness tree τ_A. Each proper witness tree can be generated by a branching process, so the total number of witness tree can be bounded by the the total number of witness trees generated by branching process. Furthermore, for a set consisting of witness tree corresponding to a specific execution log, not every every tree genenrated by branching process correspond to a associated witness tree. For a generation process, it happens to generate a witness tree associated with some entry of a specific log L wiht probability $\Pr[\tau_A \text{ occurs in log } L\ ]$

In a compplete execution of braching process(it might never stop and generate a infinite witness tree), it generates a specific witness tree with probability p_{τ_A}. Thus, in an execution of branching process, we may have $E[M_A] = \frac{1}{p_{\tau_A}}$ distinct witness trees on average. And they associated the the same execution log L with probability $E[M_A] \cdot \Pr[\tau_A \text{ occurs in log } L\ ]$. That is exactly $E[N_A]$.

Counting in two ways(expectation)

In a specific execution log L of an execution of algorithm, consider the witness tree associated with last occurence of event A. Denote by M_A the number of times of running branching process to obtain it. Since branching process generates a specific witness tree with probability p_{τ_A}, then we need average run $E[M_A] = \frac{1}{p_{\tau_A}}$. On the other hand, each tree generated by branching process occurs in the specific log L with probability $\Pr[\tau_A \text{ occurs in log } L\ ]$, therefore, the average number of trees rooted at A is $E[M_A] \cdot \Pr[\tau_A \text{ occurs in log } L\ ]$,that's exactly $E[N_A]$. How to express $\frac{N_A}{M_A} = p_{\tau_A}$ or $\frac{N_A}{p_{\tau_A}} = M_A$**(on the left, is the expected number of running times of branching process to generate a tree rooted at A from some log=the right hand)** accurately???

Variable x, y and a probability p, how to express $x = py$ accurately.

Then N_A will be total number of witness tree associated with A.

for $i := 1$ to m **do**
    $v_i :=$ random assignment
while $\exists A_i$ happens **do**
    for each $v \in vbl(A_i)$
        $v :=$ new random assignment

References

[CW87] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *Proceedings of the 19th ACM Symposium on Theory of Computing*, 1987, pp. 1–6.