An Algorithm for the Lovász Local Lemma

Last time we used the Lovász local lemma to show the existence of a satisfying assignment to $k$-SAT when each variable occurs in fewer than $\frac{2^e}{e}$ clauses. While the probabilistic method is good for showing existence, it doesn’t help us to find a satisfying assignment. In this lecture, we will present an algorithm for finding the satisfying assignment.

1 Preliminaries

Let $X = X_1, \ldots, X_n$ be iid, uniform random variables taking values in $\{0, 1\}$. Additionally, let $A_1, \ldots, A_m$ be some bad events, where each $A_i$ depends on some components of $X$. Let $vbl(A_i) = \{X_j \mid A_i$ depends on $X_j\}$. The goal is to find some assignment to $X$ such that none of the bad events occur.

We also have a dependency graph $G$ where each $A_i$ is a vertex. Denote the neighborhood of $A_i$, or the set of vertices adjacent to $A_i$, by $\Gamma(A_i)$. Then $A_i \in \Gamma(A_j) \iff vbl(A_i) \cap vbl(A_j) \neq \emptyset$.

1.1 Applied to $k$-SAT

A $k$-SAT instance consists of a CNF formula where each clause is a disjunction of $k$ literals. The variables in our $k$-SAT instance will be those in $X$. Let $m$ be the number of clauses in the $k$-SAT formula. The conjunction of these $m$ clauses is the formula. The goal is to find an assignment to $X$ such that the value of the CNF formula is 1 on this assignment. We define $A_1, \ldots, A_m$ such that each $A_i$ corresponds to a specific clause and the event $A_i$ occurs if the $i$th clause is violated (all of its literals are set to 0).

Because $X$ is uniformly random,

$$\Pr[A_i] = 2^{-k} \quad \forall i.$$

Note that we are assuming $X$ is uniform, but the Lovász local lemma allows for an arbitrary probability space.

In our dependency graph $G$, our nodes will correspond to clauses of the CNF formula and a pair of nodes will be adjacent iff the clauses share a variable. Let $\Delta(G)$ be the maximum degree of $G$. Using the Lovász local lemma, we know that a satisfying assignment to $X$ exists if $\Delta(G)$ is small. More specifically, if $\Delta(G) \leq \frac{2^e}{e} - 1$, then there exists a satisfying assignment.
1.2 The Algorithm

The algorithm is straightforward. We simply start with a random assignment to $X$. While there exists some clause that is violated, we resample all of its variables to give a new assignment. If there are multiple violated clauses, we can choose one arbitrarily. This can affect other clauses, but we want to show that the effect on other clauses small enough that this algorithm terminates and finds a satisfying assignment quickly. We will do this using witness trees.

1.3 Witness Trees

A witness tree is a graph whose vertices are $A_1, \ldots, A_m$. We will use the terms clause and vertex/node interchangeably, as each vertex is representative of some clause. For some sequence of clauses we resample, the witness tree will represent the series of previous steps that led to a given $A_i$ being violated. In the above algorithm, let $A_{x_1}, \ldots, A_{x_T}$ be the sequence of violated clauses that we attempt to fix via resampling. A witness tree with root $A_{x_i}$ will show the sequence of changes that could have caused $A_{x_i}$ to be resampled. We build a separate witness tree rooted at $A_{x_i}$ for each $i \in [T]$. Below we describe the process for creating a witness tree rooted at $A_{x_j}$.

First, add a vertex for $A_{x_j}$.
For $i = j - 1, j - 2, \ldots, 1$:
- If $\exists A_{x_k}$ in the witness graph such that $vbl(A_{x_k}) \cap vbl(A_{x_i}) \neq \emptyset$:
  - Add $A_{x_i}$ to the graph and attach to one of these $A_{x_k}$ it at the maximum depth possible.

It is possible for clauses to appear multiple times in the witness graph. We must create a separate graph rooted at $A_{x_i}$ for each $i \in [T]$. We cannot simply construct the graph rooted at $A_{x_T}$ because we may end up skipping some vertices altogether. Since we are creating $T$ separate trees, some of them will probably be subtrees of each other.

The goal of this approach is to show that large witness trees are unlikely to occur and that there are not too many of them. This will allow us to upper-bound $T$.

1.4 Analysis of Witness Trees

In order to analyze the probability of a given witness tree, it will be helpful to think of the algorithm as follows. At various points of the algorithm, we resampled the assignment to the variables in a given clause by flipping coins uniformly at random. We can think of this as if we had performed all of the coin flips in advance of running the algorithm and completed a table as follows. We create a row for each variable and then fill the table with the results of several coin flips. An example of one possible table is shown in Figure 1.
Now we think of the algorithm as a deterministic algorithm that simply references this table whenever random assignments to each variable are needed. For example, to generate the initial assignment to all the variables, we will simply use the column $b_1$. For each bit, we will keep a “pointer” to the next bit to be used. Because we only use each table entry once and the entries are generated uniformly and independently, this model is equivalent to the one initially used to describe the algorithm. The only difference is that when choosing which clause to resolve next, we cannot use the table to make these decisions; this is easily resolved by fixing some arbitrary rule (such as numbering the clauses) in advance of performing the coin flips.

Now we want to calculate the probability that a given witness tree occurs. Consider the following example.

We would like to claim that the likelihood of this tree occurring is equal to that of resampling the values in each clause separately and having them fail independently. The probability that a single resampling fails is $2^{-k}$, so we would like the probability that all $T$ resamplings fail to be $2^{-kT}$, as if they were independent.

Using the randomness table constructed at the beginning of the algorithm, we can find the value of each clause at each point in time. For example, we know

<table>
<thead>
<tr>
<th></th>
<th>$b_1$</th>
<th>$b_2$</th>
<th>$b_3$</th>
<th>$b_4$</th>
<th>$b_5$</th>
<th>$b_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$x_2$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$x_3$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$x_4$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>$x_n$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 1: An example of a randomness table
the assignments to the variables of \( D \) at its first resampling. We know that it was simply the initial assignment to the variables in \( D \) because \( D \) (the deeper one) has no descendants, meaning that any previous changes to our overall assignment were in variables not in clause \( D \) by construction of the witness graph. Similarly, we know that the assignment to the variables in the second resampling of \( D \) consists of their values in \( b_2 \). Now we consider clause \( B \). Since it is an ancestor of \( D \), they share some variables. The shared variables will have values taken from column \( b_3 \), whereas the variables not in \( D \) will still have the values in column \( b_1 \). We can therefore determine the assignments for all of the variables at each time step by filling out the tree bottom-up.

If we do this, we can find the assignments both when \( B \) and \( C \) were resampled. Note that the order is not important because they are disjoint. If they shared variables, one would be a descendant of the other.

**Claim 1** The assignments to the variables in each node of the witness tree are mutually independent.

**Proof** The assignment to every shared variable is taken from a different column of the probability table for each node whose clause contains that variable. Since the entries in the table were generated independently, the assignments to the shared variables are also independent between clauses. The assignments to the disjoint variables are also clearly independent. ■

**Definition 2** Witness Tree \( \Pi \) is consistent with randomness table \( R \) if filling in the values of \( \Pi \) from the entries of \( R \) bottom-up as described above violates all clauses in \( \Pi \).

**Lemma 3** Every witness tree constructed by generating a randomness table \( R \) and running the witness tree generation algorithm described above is consistent with \( R \).

**Proof** This follows immediately from the definitions and the fact that if a clause in the witness tree is not violated, then the algorithm will not resample it. ■

**Lemma 4** For some witness tree \( \Pi \) and randomness table \( R \), \( \Pr[\Pi \text{ consistent with } R] = 2^{-k|\Pi|} \), where \( |\Pi| \) is the number of nodes in \( \Pi \).

**Proof** The quantity \( \Pr[\Pi \text{ consistent with } R] \) is the same as \( \Pr[R \text{ violates each node of } \Pi] \), where we say that \( R \) violates a node of \( \Pi \) if the assignment to the node generated by this bottom-up procedure according to \( R \) causes the clause to be unsatisfied. Since each node is independent and the probability of violating a single node is \( 2^{-k} \), the probability of violating all of them is \( (2^{-k})^{|\Pi|} \). ■

Thus far, we’ve shown that for each step of the algorithm we can generate a witness tree to bound the runtime of the algorithm. Additionally, we’ve shown that the probability of generating a large witness tree is exponentially small. Now we need a way to express the expected runtime in terms of these witness graphs.
1.5 Upper-bounding the running time

For the randomness table $R$ fixed at the beginning of the algorithm, we generate $T$ witness trees consistent with $R$, so

$$\mathbb{E}[T] \leq \mathbb{E}[\# \text{ consistent } \Pi]$$

We have a bound for the probability that a given witness tree is consistent, so we can simply take a union bound over all possible witness trees. Now we just need a good way to enumerate all of the possible witness trees. Naturally, we will first group the witness trees based on their size because our probability is based on the size of the tree.

We can also use the fact that the maximum degree of each node is $k$ because each clause has $k$ variables and all nodes at the same level of the witness tree have disjoint variables. Otherwise they would be descendants/ancestors of each other. If we wanted to extend this to a more general setting other than $k$-SAT, the maximum degree is at most $\Delta(G)$. So, taking a union bound over all witness trees, we get

$$\mathbb{E}[T] \leq \sum_{A} \sum_{S=1}^{\infty} \Pr[ \text{WT of size } S \text{ consistent with } R \text{ and rooted at } A]$$

From Lemma 4,

$$\mathbb{E}[T] \leq \sum_{A} \sum_{S=1}^{\infty} 2^{-kS}.$$ 

So we need an expression for the number of witness trees of size $S$ and max degree $\Delta(G)$. One simple way of doing this is to first impose an ordering on the nodes. Then we can create a bit string to represent the tree based on an Eulerian tour of the tree. Each node has up to $\Delta(G)$ neighbors, or $\Delta(G)$ possible edges that can exist. When we reach a node, we append a 0 to our string if the potential edge does not exist. If the edge exists, we put a 1 and follow it. At the end of this procedure, each node will have $\Delta(G)$ bits associated with it, although they will not be contiguous. So the total length of the bit string is $\Delta(G)S$. Additionally, since the witness tree has exactly $S - 1$ edges, the bitstring has $S - 1$ ones. Therefore there are $\left(\frac{\Delta(G)S}{S-1}\right)$ possible bitstrings. This is the number of witness trees, so we can substitute it to get

$$\mathbb{E}[T] \leq \sum_{A} \sum_{S=1}^{\infty} \left(\frac{\Delta(G)S}{S-1}\right) 2^{-kS}.$$
Using the inequality \( \binom{a}{b} \leq \left( \frac{e}{b} \right)^b \),

\[
\leq \sum_A \sum_{S=1}^{\infty} (\Delta(G)e)^{S-1} 2^{-kS}
\]

\[
\leq m \sum_{S=1}^{\infty} (\Delta(G)e)^{S-1} 2^{-kS},
\]

where \( m \) is the number of clauses. Further simplifying,

\[
\leq m 2^{-k} \sum_{S=1}^{\infty} (\Delta(G)e 2^{-k})^{S-1},
\]

\[
= m 2^{-k} \sum_{S=0}^{\infty} (\Delta(G)e 2^{-k})^S.
\]

From the condition of the Lovász Local Lemma, \( \Delta(G) < \frac{2k}{e} \), so the above is a convergent geometric series and

\[
\mathbb{E}[T] = O(m).
\]