CPSC 665 : An Algorithmist's Toolkit Lecture $\sum_{A\in\mathcal{A}} \frac{x(A)}{1-x(A)}$: 6 Apr 2015

Constructive Proof of Lovász Local Lemma

Instructor: Sushant Sachdeva *Presenter: Xiao Shi*

1. Introduction

Let $\mathcal{A}$ be a finite collection of mutually independent "bad" events (ones we wish to avoid) in a probability space Ω. The probability of none of these events happening is exactly $\prod_{A\in\mathcal{A}}(1-\mathbf{Pr}[A])$. In particular, this probability is positive whenever no event in $\mathcal{A}$ has probability 1.

What if the events in $\mathcal{A}$ are not mutually independent but has limited dependence? Let $\Gamma(A) \subseteq \mathcal{A}$ be the subset of events such that A is independent from the collection of events $\mathcal{A} \setminus \{A \cup \Gamma(A)\}$. Lovász Local Lemma (LLL) says if $\left|\Gamma(A)\right|$ is bounded, i.e., each event is not dependent on too many other events, the probability that none of these events happening is strictly positive. To see why this result is somewhat surprising, we show a special case of LLL involving k-SAT.

2. General Lovász Local Lemma (LLL)

Here is a general form LLL by Erdős and Lovász [EL75].

Theorem 2.1 (General LLL)**.** *If there exists an assignment of reals $x : \mathcal{A} \mapsto (0, 1)$ such that*

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

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.

3. k-SAT

The k-satisfiability ($k \geq 2$) problem is the following: given a set of boolean variables $\{x_1, x_2, \ldots, x_n\}$, and a formula φ in conjunctive normal form (CNF) with m clauses $\{C_1, C_2, \ldots, C_m\}$, where each clause is a disjunction of k literals (x_i or its negation $\neg x_i$), is there an assignment of the n variables such that φ is true?

For example, the following is a 3-SAT instance with $n = 9, m = 4$:

$$\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_4 \vee x_5) \wedge (x_2 \vee \neg x_5 \vee x_6) \wedge (\neg x_7 \vee x_8 \vee x_9).$$

We define the *support* $S(C)$ of a clause C to be the set of variables that are involved in the clause. In the above k-SAT instance, $S(C_1)$ is $\{x_1, x_2, x_3\}$, $S(C_2)$ is $\{x_1, x_4, x_5\}$.

Let $\Gamma(C_i) = \{C_j : j \neq i, S(C_j) \cap S(C_i) \neq \varnothing\}$, and $\Gamma^+(C_i) = \Gamma(C_i) \cup \{C_i\}$. Consider the dependency graph $G(V, E)$, where each vertex represents a clause, and an edge (i, j) represents that C_i and C_j share some variables, then $\Gamma(C_i)$ is exactly the neighborhood of vertex i in G, and $\Gamma^+(C_i)$ is the neighborhood including vertex i itself.

3.1. LLL as k-SAT.

Theorem 3.1 (Special Case of LLL)**.** *For each clause C_i in φ, if the support of C_i intersects with at most 2^{k-d} supports of clauses (including C_i itself), where d is a sufficiently large absolute constant, then φ is satisfiable.*

Notice the bounds here are uniform in the number of variables n. It is evident how this theorem is a special case of LLL. Let event A be a clause is not satisfied, we want to find an assignment $x : \mathcal{A} \mapsto (0, 1)$ such that the probability of the events satisfies the bound $\mathbf{Pr}[A] = 2^{-k} \leq x(A) \prod_{B \in \Gamma(A)} (1 - x(B))$. Assume symmetry among all the clauses, it is a simple calculation to show that there is an constant $x_0 \in (0, 1)$ such that $2^{-k} \leq x_0(1 - x_0)^{2^{k-d}-1}$ for a constant d. (Note: the constant d can be chosen independent of m,n,or k, e.g., $d = 3$ will work.)

3.2. **Non-Constructive vs. Constructive.** The textbook proof of LLL is generally non-constructive: it shows that the probability of φ is satisfiable is positive, but does not provide a satisfying assignment via the proof. Moser came up with a constructive proof and Tardos [MT09] generated it to the general LLL in their later paper. In particular, Moser proposed the following intuitive algorithm for producing an assignment for a k-`SAT` instance that satisfies the condition in Theorem 3.1.

3.3. **Difficulty in Producing a Satisfying Assignment.** It is tempted to just randomly sample all n boolean variables and resample until all clauses are satisfied. However the probability of success tend to be exponentially low. Consider the case where $m = n$ and $k = 1$, i.e., each clause contains one distince variable, the probability of finding a satisfying assignment is 2^{-n}.

4. Algorithm by Moser

The algorithm Moser proposed is similar. The main routine Solve(φ) keeps a global assignment $w \in \{0, 1\}^n$ which Fix(C) keeps making modifications to. When the algorithm terminates, the assignment w is the produced (satisfying) assignment.

Algorithm 1 Solve(φ)

```
w ← a random assignment to the n boolean variables involved in φ
for C_i that is not satisfied do
    Fix(C_i)
end for
```

Algorithm 2 Fix(C)

```
Randomly sample k bits and assign to the k variables in S(C)
for Clause C_j ∈ Γ⁺(C) that is violated do
    Fix(C_j)
end for
```

A few clarifications:

(1) We use a certain order of all clauses and variables, so that in the for loops, we check the clauses in increasing order of their indices.
(2) Since we keep a global assignment which changes as Fix gets called recursively, every time in the for loop, we check whether a clause is violated by the most updated assignment.
(3) Upon entry of Fix(C), we know that C is violated before the resampling step.

4.1. **Correctness.**

Lemma 4.1. If *Solve*(φ) *terminates, the assignment* w *at the point of termination satisfies* φ.

We can show this by induction on the calling tree of Fix, that if Fix(C) terminates, the resulting modification to a will satisfy Clause C; furthermore, any other clause C' which was already satisfied by w before Fix(C) is called will continue to be satisfied by w.

There is, however, a simpler argument to show the correctness provided that the algorithm terminates.

Proof. Consider the finite calling sequence of all the calls to Fix over the course of the entire run of the algorithm, e.g., $C_1, C_1, C_3, \ldots, C_4, C_2, C_1, C_2$.

Consider any clause C_j. The variables in $S(C_j)$ may be changed if and only if a clause $C_i \in \Gamma^+(C_j)$ is in the calling sequence. If no such clause C_i is in the sequence, then the assignment to the k variables in $S(C_j)$ in the end is the same as the initial assignment.

If there is such a clause C_i, since the calling sequence is finite, consider the last such C_i in the sequence. First we notice that after Step 3 of this call of Fix(C_i), the assignment to the variables in $S(C_j)$ stays the same till the algorithm terminates. Since we did not call Fix(C_j) after this point ($C_j \in \Gamma^+(C_j)$), that assignment must satisfy C_j.

Hence, if the algorithm terminates, the final assignment satisfies all clauses. □

4.2. **Termination.** If we can show that given φ and the conditions in Theorem 3.1, the probability of this algorithm terminating is positive, which means the probability of algorithm producing a satisfying assignment for the k-`SAT` instance is positive, it follows that this k-`SAT` instance is satisfiable.

5. Proof of the Theorem 3.1

5.1. **Random Bits and Algorithm Instances.** Suppose the algorithm uses a sufficiently long tape of uniformly sampled random bits as its source of randomness, i.e., whenever the algorithm samples n bits uniformly at random, it simply reads the next n bits from the tape.

We first notice that the only non-determinism in the algorithm comes from the resampling steps. In other words, if we fix a string of bits of certain length as the tape, we know exactly how the algorithm will run and whether it will terminate, which we refer to as an instance of the algorithm. Therefore, there exists a (deterministic) mapping from random bits to algorithm instances. Consider the run of the algorithm after T Fix calls, $n+Tk$ bits of the tape are consumed.

A key observation made by Moser is that we can deduce the random bits once we know the exact execution of the algorithm after T steps. Assume true, and if we could use a log string L of length l to specify the ALG instance, we can bound the probability of the algorithm not terminating in the following way.

$$
\begin{aligned}
&\mathbf{Pr}\left[\text{not terminating after } T \text{ Fix calls}\right] \\
=&\frac{\#(n+Tk)\text{-bit sequences whose corresponding ALG does not terminate in } T \text{ calls}}{2^{n+Tk}} \\
=&\frac{\#\text{ALG instances that do not terminate in } T \text{ calls}}{2^{n+Tk}} \\
\leq&\frac{2^l}{2^{n+Tk}}
\end{aligned}
$$

We now show that each ALG instance that does not terminate within T steps uniquely maps to a string of random bits it consumed.

5.2. **Deducing Random Bits.** If the current assignment w does not satisfy some clause C_i, it reveals k bits information about w, i.e., we know exactly what the k variables in $S(C_i)$ were assigned to. Therefore, each time the resampling step of $\mathrm{Fix}(C_i)$ is called, k more bits on the tape are consumed, assignment w changes to w'. Provided that we know what w' is and which clause the algorithm is being fixed (i.e., C_i), we can deduce what w iss since we know C_i was not satisfied by w, and what the k random bits read from the tape were (it is exactly the k bits assigned to $S(C)$ in w').

5.3. **Execution Log of the Fix Routines.** We specify the execution log L of Fix in the following way.

Fix routines are called in two categories. Firstly, some Fix routines are called by the Solve routine. We could use m bits to indicate whether each clause is called in Solve. Secondly, there are Fix routines called by other Fix routines. Suppose $\mathrm{Fix}(C)$ makes r calls to $\{\mathrm{Fix}(C_{j_1}), \mathrm{Fix}(C_{j_2}), \ldots, \mathrm{Fix}(C_{j_r})\}$. Since $r \leq \left|\Gamma^+(C)\right| \leq 2^{k-d}$, we can use a number between 1 and 2^{k-d} to represent the position of C_j in $\Gamma^+(C_i)$ (with respect to the fixed ordering of all m clauses). Call this number the *index* of the call $\mathrm{Fix}(C_j)$, which is representable in $\log r \leq k - d$ bits.

Now the execution log L records C each time Solve invokes a Fix call, and also records the index of any other call $\mathrm{Fix}(C)$ made during the recursive procedure. Finally, we assume that this log file records a termination symbol whenever a Fix routine terminates. By performing a stack trace, one sees that whenever a Fix routine is called, the clause C that is being fixed by that routine can be deduced from an inspection of the log file L up to that point. The log file L has size at most $m + T(k - d + O(1))$: it takes m bits to store the calls from *Solve*, and it then takes $k - d$ bits to specify each child call and a possible termination symbol.

As a consequence, at any intermediate stage of these Fix calls, all the random bits read from the tape can be deduced from the current assignment and the log file L up to this point, which takes a total number of $n + O(m) + T(k - d + O(1))$ bits.

5.4. **Bound on Termination.**

$$\begin{aligned}
&\mathbf{Pr}\left[\text{not terminating after } T \text{ Fix calls}\right] \\
\leq &\frac{2^{n+m+T(k-d+O(1))}}{2^{n+Tk}} \\
= &2^{m-T(d-O(1))}
\end{aligned}$$

Since d is larger than an absolute constant, when $T >= m$, the probability of the algorithm not terminating after T Fix calls goes down exponentially. Hence the probability of the algorithm terminating is positive, which implies that φ is satisfiable.

Given the geometric decay of probability when $T > m$, the expected number of calls it takes for the algorithm to terminate is at most $m + 2 = O(m)$.

6. Alternative Proof

Moser presented an alternative proof via the Incompressibility argument in 2009.

6.1. **Introduction to the Incompressibility Argument.**

Definition 6.1 (Shannon Entropy)**.** *Given a discrete random variable $X = \{x_1, x_2, \ldots, x_n\}$ and its probability mass function $P(X)$, the Shannon Entropy is defined as*

$$H(X) = \mathbf{E}[I(X)] = \mathbf{E}[-\log P(X)],$$

where $I(X)$ is the information content of X.

Informally, Shannon Entropy is the average amount of information contained in each event. A random coin toss conveys a full bit of information. Similarly, a random n-bit string (independently and uniformly sampled) conveys n bits of information.

A random variable cannot be compressed *losslessly* into a string of expected size smaller than the Shannon entropy of that variable. This provides a lower bound. Another way of stating this claim is the following: if you try to use $n' < n$ bits to "encode" or "recover" a uniformly random n-bit string, with probability at least $1 - 2^{-(n-n')}$ you will fail.

6.2. **Proof by Contradiction.**

Proof. Assume by way of contradiction that φ is not satisfiable. Hence the stack of Fix routines will never completely terminate. We trace this stack for T calls. Let w_T denote the assignment up to this point of the algorithm.

After these steps, the total number of random bits read is $n + Tk$. On the other hand, the log file L has size at most $m + T(k - d + O(1))$.

Hence we have a lossless compression algorithm from $n+Tk$ random bits to $n+m+T(k-d+O(1))$ bits (w_T and L). By the entropy bound, we have

$$n + Tk \leq n + m + T(k - d + O(1)).$$

When d is larger than an absolute constant, and $T > m$, this leads to a contradiction. □

7. Algorithm for General LLL

The algorithm by Moser and Tardos for the general case of LLL is very similar. Let $\mathcal{P}$ be a finite collection of mutually independent variables in a fixed probability space. Consider each event $A \in \mathcal{A}$ to be determined by a subset $S \subseteq \mathcal{P}$. There is a unique minimal such subset, we call it the *support* of A. The algorithm initially resamples all variables, and resamples $S(A)$ for events that are violated by the current assignment. The expected number of resampling steps was shown to be at most $\sum_{A \in \mathcal{A}} \frac{x(A)}{1-x(A)}$.

7.1. **Parallel Version.**

Instead of sampling $S(A)$ one at a time, in each stage, we find the maximal independent set G' in the subgraph of the dependency graph induced by all events which are violated, and resample all variables $P \in \cup_{A \in G'} S(A)$. The expected number of resampling steps are even lower. For exact results, see [MT09].

References

[EL75] Paul Erdős and László Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. 1975.

[MT09] Robin A. Moser and Gábor Tardos. A constructive proof of the general lovasz local lemma. *CoRR*, abs/0903.0544, 2009.