Fall 2014 Randomized Algorithms Oct 29, 2014

Lecture 6

Prof. Friedrich Eisenbrand *Scribes: Abbas Bazzi*

In this lecture, we introduce the symmetric version of the Lovázs Local Lemma along with some of its applications to combinatorial problems. Namely, we apply the Lemma to conclude that a CNF formula with certain properties has a satisfying assignment. Since this tool is non-constructive, i.e. it only proves the existence of such an assignment, we present an algorithm by Mossel that enables us to find this assignment.

1 Lovász Local Lemma

The Lovász Local Lemma is one of the powerful probabilistic tools that allow us to prove the existence of certain combinatorial object with certain properties. For instance, assume that we have n bad events $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$ that we trying to avoid, such that $\mathbf{Pr}[\mathcal{E}_i] \leq p$ for $1 \leq i \leq n$. If we have the guarantee that $\mathcal{E}_i$ and $\mathcal{E}_j$ are independent for all $i \neq j$, then the probability of avoiding all such events can be shown to be $(1-p)^n > 0$. However, dropping the independence assumption and using only union bound would just give us

$$\mathbf{Pr}\left[\bigwedge_i \bar{\mathcal{E}}_i\right] \geq 1 - \sum_i \mathbf{Pr}\left[\bar{\mathcal{E}}_i\right] \geq 1 - np$$

The Lovász Local Lemma, whose symmetric version is stated below, improves upon the union bound when the dependence between the bad events is restricted.

Lemma 1 *(Symmetric Lovász Local Lemma) Let $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_m$ be a set of events with $\mathbf{Pr}[\mathcal{E}_i] \leq p$ for $i = 1, \ldots, m$, and each $\mathcal{E}_i$ is dependent on at most d other $\mathcal{E}_j$. If $ep(d+1) \leq 1$, then*

$$\mathbf{Pr}\left[\bigwedge_i \bar{\mathcal{E}}_i\right] > 0$$

To appreciate this result, we study its implications on two important combinatorial problems, namely hypergraph coloring and satisfiability .

1.1 Example 1: Hypergraph Coloring

Let $H(V, E)$ be a hypergraph such that each hyperedge $e_i \in E$ contains at least k vertices, and e_i intersects at most 2^{k-3} other hyperedges. We want to apply the Lovász Local Lemma to show that H admits a non-monochromatic 2-coloring.

In order to do so, consider the following randomized coloring, where each vertex $v \in V$ is colored with either red or blue independently with probability $1/2$, and denote by $\mathcal{E}_i$ the probability that e_i is monochromatic under the resulting coloring. An edge is monochromatic if either all its vertices are red, or all of them are blue, and such an events happens with probability

$$\mathbf{Pr}\left[\mathcal{E}_i\right] \leq \left(\frac{1}{2}\right)^k + \left(\frac{1}{2}\right)^k = 2^{k-1}$$

A simple application of the Lovász Local Lemma yields that the probability that all edges are non-monochromatic ($\mathbf{Pr}\left[\bigwedge_i \bar{\mathcal{E}_i}\right]$) is greater than zero, since

$$ep(d+1) = e2^{k-1}\left(2^{k-3}+1\right) \leq 1$$

Hence we have shown that a random 2-coloring yields a non-monochromatic coloring for H with probability greater than 0, then there must exist at least one coloring that satisfies this property.

1.2 Example 2: Satisfiability

Given a CNF formula $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ over n variables $X = \{x_1, x_2, \ldots, x_n\}$, where each clause C_i is the disjunction of at most k literals, we want to derive a condition on the dependencies among the clauses, under which it is guaranteed that ϕ has a satisfying assignment. Define $\mathbf{Supp}(C_i)$ to be the support of a clause C_i, i.e.

$$\mathbf{Supp}(C_i) = \{x \in X : x \in C \text{ or } \bar{x} \in C\}$$

We claim that if for each clause C_i, $\mathbf{Supp}(C_i)$ intersects the supports of at most 2^{k-2} other clauses, then ϕ is guaranteed to have a satisfying assignment.

Consider the following randomized assignment that sets each variable x_j uniformly at random to either 0 or 1, and denote by $\mathcal{E}_i$ the event that clause C_i is unsatisfied by this randomized assignment. It is easy to see that $\mathbf{Pr}\left[\mathcal{E}_i\right] \leq 2^{-k}$ for $i = 1, \ldots, m$, and that $\mathcal{E}_i$ and $\mathcal{E}_j$ are dependent iff $\mathbf{Supp}(C_i) \cap \mathbf{Supp}(C_j) \neq \emptyset$, and hence each $\mathcal{E}_i$ depends on at most 2^{k-2} other $\mathcal{E}_j$'s. A simple application of the Lovász Local Lemma yields that the probability that all the clauses are satisfied ($\mathbf{Pr}\left[\bigwedge_i \bar{\mathcal{E}_i}\right]$) is greater than zero, since

$$ep(d+1) = e2^{k}\left(2^{k-2}+1\right) \leq 1$$

So far we have shown that there for $p = 2^{-k}$ and $d = 2^{k-2}$, there exists an assignment that satisfies all the clauses. In the remainder of this lecture, we will show how to actually find this assignment.

2 Constructive Version of the Lovász Local Lemma

We assume in what follows that for each clause C_i, $\mathbf{Supp}(C_i)$ intersects the supports of at most $2^{k-\alpha}$ other clauses, where α is a constant. Consider the following algorithm for the k-satisfiability problem.

Algorithm 1 Find_Sat_Assignment

```
1: procedure SAT($\phi$)
2:     $A \leftarrow$Random Binary Assignment for the variables of $\phi$.
3:     $T \leftarrow$Unsatisfied clauses of $\phi$ under $A$.
4:     while $T \neq \emptyset$ do
5:         $C \leftarrow$ Random Clause in T.
6:         $A \leftarrow Fix(C)$.
7:         $T \leftarrow$Unsatisfied clauses of $\phi$ under $A$.
8:     return $A$
```

Where $Fix(C)$ is the following algorithm:

Algorithm 2 Fix

```
1: procedure FIX($C$)
2:     if $C$ is satisfied by A then Do nothing.
3:     else
4:         Randomly reassign the variables in $C$ and update $A$.
5:         for $\tilde{C}$ such that $\mathbf{Supp}(\tilde{C}) \cap \mathbf{Supp}(C) \neq \emptyset$ do
6:             $A = Fix(\tilde{C})$
```

It is easy to see that if $Fix(C)$ terminates, then C and all the clauses that were satisfied before invoking $Fix(C)$ remain satisfied after it returns. Hence $Sat(\phi)$ will return a satisfying assignment; this is because each iteration of the main algorithm can only decrease the size of T, meaning that the overall number of iterations is $O(m)$. It remains to show that $Fix(C)$ terminates.

Assume that at the beginning of the algorithm, we generate a truth assignment A for the variables, and a long random string R, where the k leading bits of R are used as a random reassignment in line 4 of Algorithm 2, and then truncated from R. Let A' and R' be the assignment and the random string resulting from the execution of Line 4 in $Fix(C)$ for some clause C, then it is easy to see that knowing A', R' and C, we can recover A and R. Indeed A' and A are the same outside $\mathbf{Supp}(C)$ and $A|_{\mathbf{Supp}(C)}$ is the only unsatisfying configuration for C. On the other hand R is nothing but $A'|_{\mathbf{Supp}(C)}$ appended toR'.

In order to prove that $Fix(C)$ terminates, we will use the fact that the uniformly random string $(A + R)$ is not effectively compressible, and hence a single call of $Fix(C)$ in Algorithm 1 can induce a *bounded number* of recursive calls for Algorithm 2. Since our main argument will involve compressing $(A+R)$, and we have already established that the relation $A + R \mapsto A' + R' + C$ is reversible, we will then need some valid binary encoding of the clauses of ϕ. The naive $\lceil \log_2(m) \rceil$-encoding would not do the job, since we will get that the length of $(A' + R' + C)$ is

$$|A| + (|R| - k) + \lceil \log_2(m) \rceil$$

and we can only talk about compression when $k > \lceil \log_2(m) \rceil$.

2.1 The Binary Encoding

We differentiate between two type of clauses:

- Type I: The clauses for which $Fix(C)$ was called in Line 6 of Algorithm 1.
- Type II: The clauses C that were recursively called in Line 6 of Algorithm 2.

The catch here is that if we know for which clause C the subroutine $Fix(C)$ is called, we only need $(k-\alpha)$ bits to index any clause $\tilde{C}$ that might be recursively called at the top level of this subroutine. For a clause C, denote by $N(C)$ the set of clauses whose support intersects that of C, i.e. $N(C) = \{\tilde{C} : \mathbf{Supp}(C) \cap \mathbf{Supp}(\tilde{C}) \neq \emptyset\}$.

As a preprocessing step, we encode for each clause C its corresponding set $N(C)$ by a $(k-\alpha)$-bits code, and each $C \in T$ by a $\lceil \log_2(m) \rceil$-bits code.

2.2 The Stack of Recursive Calls

Let $C^* \in T$ be a Type I clause, and consider the stack trace S resulting from the execution of $Fix(C^*)$ constructed as follows:

- S is initialized to the $\lceil \log_2(m) \rceil$-encoding of C^* followed by a \$-symbol.
- When Fix is recursively called on some $\tilde{C} \in N(C)$ (Line 6), S is appended by the $(k-\alpha)$ encoding of $\tilde{C}$ with respect to C followed by a \$-symbol.
- When Fix terminates for some C (Line 2), S is appended by the #-symbol.

An illustration of S would like as follows:

$$S = [\mathbf{Enc}_1(C^*) \ \$ \ \mathbf{Enc}_2(C_i) \ \$ \ \mathbf{Enc}_2(C_j) \ \$ \ \mathbf{Enc}_2(C_k) \ \# \ldots]$$

where $\mathbf{Enc}_1$ and $\mathbf{Enc}_2$ denote the $\lceil \log_2(m) \rceil$ and the $(k-\alpha)$ encodings respectively. It is easy to see that at any stage i of the execution of $Fix(C^*)$, one can *backtrack* the original A and R from A' R' and S_i, where S_i is the stack trace at stage i.

Instead of showing that only $Fix(C)$ terminates, we will show that Algorithm 1 terminates, and we let S denote the stack of the main algorithm. Assume towards contradiction that it does not terminate, and denote by S_M the stack trace up to the M^{th} stage. Up to this stage, R was truncated M times, and hence shortened by Mk bits, so we assume that R was originally of length Mk and hence is *empty* at this stage.

On the other hand, S_M has size at most $O(m \log(m)) + M(k - \alpha + O(1))$, where the $O(m \log(m))$ term accounts for the $\mathbf{Enc}_1$ encoding of the clauses in T, and $M(k-\alpha+O(1))$ accounts for the $\mathbf{Enc}_2$ encoding required in each recursive call (plus the \$ and # symbols). Recall that we started from a random string $A + R$ of length $(n + Mk)$ bits, and we ended up with a string $A' + R' + S_M$ of length $n+0+O(m\log(m))+M(k-\alpha+O(1))$, from which we can losslessly recover $(A+R)$. But (A,R) is a uniformly random string and hence not effectively compressible so we get

$$\begin{aligned} n+O(m\log_2(m))+M(k-\alpha+O(1)) &\geq n+Mk \\ \implies O(m\log_2(m)) &\geq M(\alpha-O(1)) \end{aligned}$$

choosing M large enough and C greater than some absolute constant yields a contradiction.