CS 270 Combinatorial Algorithms and Data Structures, Spring 2015

Lecture 10 : Local lemma

Local lemma is a useful tool in probabilistic methods with various applications.

Today's focus is on recent developments in making this method constructive, and if time permitted we will go through an interesting application in packet routing.

Lovász Local Lemma

Let $E_1, E_2, \dots, E_n$ be a set of "bad" events.

A typical goal is to show that there exists an output with no bad events occur.

For example, in K-SAT, we want to find an assignment with no clauses violated (bad events)

That is, we want to show that $\Pr(\bigcap_{i=1}^{n} \bar{E}_i) > 0$.

There are two situations when this is easy to show:

- when the events $E_1, \dots, E_n$ are mutually independent
- when $\sum_{i=1}^{n} \Pr(E_i) < 1$, in other words, when the union bound applies.

Lovász local lemma can be seen as a clever combination of them.

We say that an event E is mutually independent of the events $E_1, E_2, \dots, E_n$ if for any subset $I \subseteq [n]$, $\Pr(E \mid \bigcap_{j \in I} E_j) = \Pr(E)$.

Definition A dependency graph for a set of events $E_1, \dots, E_n$ is a graph $G=(V,E)$ s.t. $V = \{1, \dots, n\}$ and for $1 \le i \le n$ event E_i is mutually independent of the events $\{E_j \mid (i,j) \notin E\}$.

Theorem (Lovász local lemma) Let $E_1, \dots, E_n$ be a set of events. Suppose the followings hold:

① $\Pr(E_i) \le p$

② The max degree in the dependency is at most d

③ $4dp \le 1$.

Then $\Pr(\bigcap_{i=1}^{n} \bar{E}_i) > 0$.

proof We prove by induction that $\Pr(\bigcap_{i \in S} \bar{E}_i) > 0$ on the size of S.

To prove this, there is an intermediate step showing that $\Pr(E_k \mid \bigcap \bar{E}_i) \le 2p$.

The proof structure is like this: $\Pr(\bigcap_{i\in S:|S|=1} \bar{E}_i) > 0 \Rightarrow \Pr(E_k \mid \bigcap_{i\in S:|S|=1} \bar{E}_i) \le 2p$

$\Rightarrow \Pr(\bigcap_{i\in S:|S|=2} \bar{E}_i) > 0 \Rightarrow \Pr(E_k \mid \bigcap_{i\in S:|S|=2} \bar{E}_i) \le 2p$

$\Rightarrow \dots$

$\Rightarrow \Pr(E_k \mid \bigcap_{i\in S:|S|=n-1} \bar{E}_i) \le 2p \Rightarrow \Pr(\bigcap_{i\in S:|S|=n} \bar{E}_i) > 0$

First we prove $\Pr(\bigcap_{i\in S} \bar{E}_i) > 0$ assuming the previous steps in the chain are proven.

The base case when $|S|=1$ is easy, since $\Pr(\bar{E}_i) = 1 - \Pr(E_i) = 1-p > 0$.

For the inductive step, without loss of generality assume $S = \{1, 2, \dots, s\}$.

$$\Pr\left(\bigcap_{i=1}^{s} \bar{E}_i\right) = \prod_{i=1}^{s} \Pr\left(\bar{E}_i \,\Big|\, \bigcap_{j=1}^{i-1} \bar{E}_j\right) = \prod_{i=1}^{s}\left(1 - \Pr\left(E_i \,\Big|\, \bigcap_{j=1}^{i-1} \bar{E}_j\right)\right) \ge \prod_{i=1}^{s}(1-2p) > 0.$$

Next we prove $\Pr(E_k \mid \bigcap_{i\in S} \bar{E}_i) \le 2p$ assuming the previous steps are proven.

To do this we first divide the events into two types, based on its dependency to E_k:

$S_1 = \{i \in S \mid (k,i) \in E\}$ and $S_2 = \{i \in S \mid (k,i) \notin E\}$.

If $|S| = |S_2|$, then $\Pr(E_k \mid \bigcap_{i\in S} \bar{E}_i) = \Pr(E_k) \le p$ and we're done.

Henceforth we assume $|S| > |S_2|$.

Let $F_S = \bigcap_{i\in S} \bar{E}_i$ and similarly define F_{S_1} and F_{S_2}. Note $F_S = F_{S_1} \cap F_{S_2}$.

$$\Pr(E_k \mid F_S) = \frac{\Pr(E_k \cap F_S)}{\Pr(F_S)} \underset{\uparrow}{=} \frac{\Pr(E_k \cap F_{S_1} \mid F_{S_2})\Pr(F_{S_2})}{\Pr(F_{S_1} \mid F_{S_2})\Pr(F_{S_2})} = \frac{\Pr(E_k \cap F_{S_1} \mid F_{S_2})}{\Pr(F_{S_1} \mid F_{S_2})}$$

we do this because we want to take advantage of the independence of E_k and F_{S_2}

$\downarrow$

The numerator is $\Pr(E_k \cap F_{S_1} \mid F_{S_2}) \le \Pr(E_k \mid F_{S_2}) = \Pr(E_k) \le p$.

The denominator is $\Pr(F_{S_1} \mid F_{S_2})$

$$= \Pr\left(\bigcap_{i\in S_1} \bar{E}_i \,\Big|\, \bigcap_{j\in S_2} \bar{E}_j\right) = 1 - \Pr\left(\bigcup_{i\in S_1} E_i \,\Big|\, \bigcap_{j\in S_2} \bar{E}_j\right)$$

$\ge 1 - \sum_{i\in S_1} \Pr(E_i \mid \bigcap_{j\in S_2} \bar{E}_j)$ (by union bound)

$\ge 1 - \sum_{i\in S_1} 2p$ $\leftarrow$ (induction hypothesis, because $|S_2| < |S|$)

$\ge 1 - 2dp$

$\ge 1/2$.

Plug it back, $\Pr(E_k \mid E_S) = \dfrac{\Pr(E_k \cap F_{S_1} \mid F_{S_2})}{\Pr(F_{S_1} \mid F_{S_2})} \le \dfrac{p}{1/2} = 2p$. $\square$

Applications

We show one easy and one more advanced application.

① k-SAT

A boolean formula with exactly k variables in each clause. We would like to find an assignment of T/F to each variable s.t. all the clauses are satisfied.

This problem is NP-complete.

But we can prove that if each variable appears in not ~~too~~ many clauses, then there is always a satisfying assignment.

Theorem If no variable in a k-SAT formula appear in more than $T = 2^k/4k$ clauses, then the formula has a satisfying assignment.

Proof Consider a random assignment where each variable is true with prob. $1/2$ independently.

Let E_i be the bad event that clause i is violated by the random assignment.

Since each clause has k variables, $p = \Pr(E_i) = 2^{-k}$.

The event E_i is mutually independent of all other events correponding to clauses that do not share variables with E_i.

So, $d \leq kT = 2^{k-2}$. Hence $4dp \leq 1$.

By local lemma, $\Pr(\bigcap_{i=1}^{m} \bar{E_i}) > 0$, so there is an assignment satisfying all clauses. $\square$

② packet routing

We are given an undirected graph, N pairs, each pair has a source s_i, a destination t_i and a path P_i.

In each time step, at most one packet can traverse an edge.

A packet can wait at any node at any time step.

A schedule for a set of packets specifies the timing of the movement of packets along their respective paths (i.e. when to move and when to wait).

The goal is to find a schedule to minimize the total time to route all the packets.

Let d be the maximum distance traveled by any packet. (i.e. maximum path length)

Let c be the max. number of packets that must traverse a single edge.

Then, it is clear that any schedule must require $\Omega(c+d)$ steps to finish.

A surprising result by Leighton, Maggs, and Rao shows that there is always a schedule using $O(c+d)$ steps.

This is a very strong result, as the bound is independent of n.

While usually we use Chernoff bound and union bound, the new idea here is to use Chernoff bound and local lemma.

For simplicity we only prove a weaker result, but is independent of N.

<u>Theorem</u> There is a schedule with length $O\left((c+d)(1+\alpha)^{O(\ln^*(c+d))}\right)$, where α is a constant and $\ln^*(n)$ is the number of $\ln$ it takes to bring n to ≤ 1.

($\log^*(n)$ is growing extremely slowly, e.g. $\log_2^*(2^{65536}-1)=5$.)

<u>Proof</u> Without loss of generality assume $c=d$.

For each packet, assign an initial delay from $[1, \alpha d]$ for some constant α.

We first consider a relaxed version of the problem, where the packet can go without any interruption (i.e. without waiting at a node) towards its destination.

So, the total time needed (in this relaxed version) is at most $(1+\alpha)d$.

Partition the time into periods, each period having $\ln d$ steps.

We will show that with positive probability that each edge has congestion at most $\ln d$ (equal to $\ln c$ by our assumption) in each period, i.e. $\leq \ln d$ packets using that edge in that period.

Then, for each period, we can think of it as a sub-problem with $c' = \ln d$ and $d' = \ln d$.

Then, for each subproblem, we apply the same argument <u>recursively</u>.

i.e. each subproblem can be partitioned into periods with $\log d' = \log\log d$ steps, s.t. each edge has congestion $\leq \ln d' = \ln\ln d$ in each period, and so on.

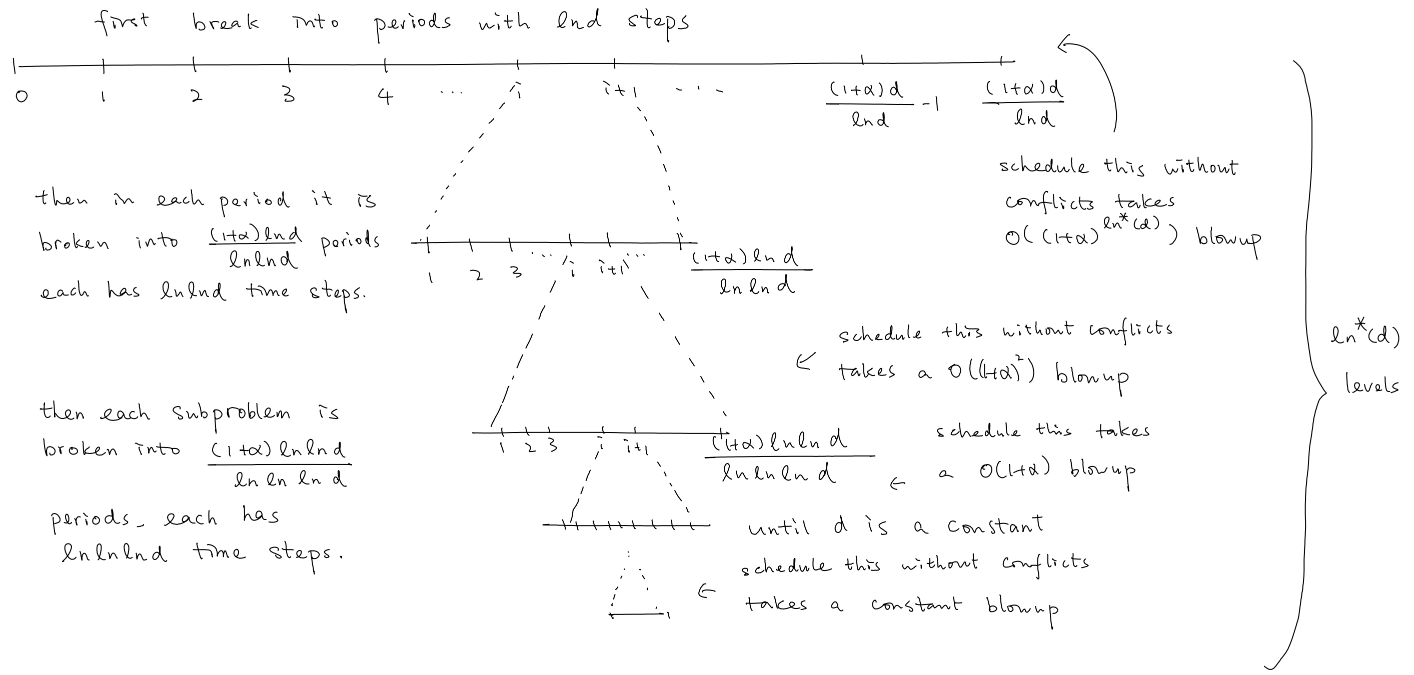
So, we recursively partition the problem until c and d become constant, and then we just find a naive $O(cd)$ solution (e.g. in arbitrary order). In these base cases this is also a $O(c+d)$ solution since c and d are constants.

To reduce the problem into constant congestion and maximum path length, we just need $O(\ln^* d) = O(\ln^*(c+d))$ recursions.

In each recursion we blow up the schedule by a factor of $(1+\alpha)$.

So the total time in the schedule is $O((c+d)(1+\alpha)^{O(\ln^*(c+d))})$ steps, and each edge is used by at most one packet in each time step.

Pictorically, the proof goes like this


first break into periods with $\ln d$ steps
0
1
2
3
4
i
$i+1$
$\frac{(1+\alpha)d}{\ln d}-1$
$\frac{(1+\alpha)d}{\ln d}$
schedule this without conflicts takes $O((1+\alpha)^{\ln^*(d)})$ blowup
then in each period it is broken into $\frac{(1+\alpha)\ln d}{\ln\ln d}$ periods each has $\ln\ln d$ time steps.
1 2 3 i $i+1$
$\frac{(1+\alpha)\ln d}{\ln\ln d}$
schedule this without conflicts takes a $O((1+\alpha)^2)$ blowup
$\ln^*(d)$ levels
then each subproblem is broken into $\frac{(1+\alpha)\ln\ln d}{\ln\ln\ln d}$ periods, each has $\ln\ln\ln d$ time steps.
1 2 3 i $i+1$
$\frac{(1+\alpha)\ln\ln d}{\ln\ln\ln d}$
schedule this takes a $O(1+\alpha)$ blowup
until d is a constant
schedule this without conflicts takes a constant blowup

Okay, it remains to prove the following lemma using local lemma.

<u>Lemma</u> When α is a large constant, each period has congestion $\leq \ln d$ with positive probability.

Let A_f be the bad event that edge f has congestion $> \ln d$ in some period.

First we bound the maximum degree in the dependency graph.

Note that whether A_e happens depends only on the at most c packets that use e.

Two events A_e and A_f are independent unless there is some packet use both e and f.

Since there are at most c packets and each such packet passes through at most d edges,

A_e is dependent on at most cd events. Hence the max. degree $\leq cd = d^2$.

Now we bound the probability that A_e happens. This is just a typical application of the Chernoff bound.

Each period is of length $\ln d$. Since there are at most $c=d$ packets that use e,

$E[\#$ of packet using e at a specific period $] \leq \frac{\ln d}{\alpha d} \cdot d = \frac{\ln d}{\alpha}$. Let this number be μ.

By Cheroff bound, $\Pr[$ congestion of e at a specific period $> \ln d]$

$$= \Pr[\text{ congestion of } e \text{ at a specific period } > (1+(\alpha-1))\mu]$$

$$\leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu} \leq \left(\frac{e}{1+\delta}\right)^{(1+\delta)\mu}$$

$$= \left(\frac{e}{\alpha}\right)^{\alpha \cdot \frac{\ln d}{\alpha}} \quad \text{since } \mu = \ln d/\alpha \text{ and } \delta = \alpha - 1$$

$$= \left(\frac{e}{\alpha}\right)^{\ln d} << \frac{d^{-4}}{\alpha} \quad \text{for say } \alpha \geq e^{11} \quad (\text{a large constant})$$

Since there are at most αd time frames, $\Pr(A_e) \leq \alpha d \cdot d^{-4}/\alpha = d^{-3}$.

So, $4 \cdot d^{-3} \cdot cd = 4d^{-1} \leq 1$ for $d \geq 4$. Hence, by local lemma, there is a choice of the initial delays such that no period has congestion more than $\ln d$.

This proves the lemma and hence the theorem. $\square$

Efficient Algorithms for Local Lemma

How to find an outcome (e.g. a satisfying assignment) whose existence is guaranteed by the local lemma?

In fact, since the probability could be very small, we don't expect that a random outcome will do, and it seems to be a very difficult algorithmic task like "finding a needle in a haystack".

There is a long history about finding efficient algorithms for local lemma, with a recent breakthough.

To illustrate the ideas, we just focus on the k-SAT problem

Original proof : It is nonconstructive, giving no idea how to find such an outcome.

Early results There is a framework developed by Beck.

Let me just try to give a very brief idea here. See MU 6.8 for details.

For this framework to work, a stronger condition is assumed: each variable appears in at most $T = 2^{\alpha k}$ clauses for some $0 < \alpha < 1$ (instead of $T = 2^k / 4k$).

The algorithm has two phases:

① Find a random "partial" assignment (each clause with at least $k/2$ variables remain unassigned). Using the local lemma itself with the stronger assumption ($T = 2^{\alpha k}$), it can be proved that the partial solution can be extended to a full solution. This step is easy if α is small enough.

② After the initial partial assignment, prove that the dependency graph is broken into small pieces, where each piece has at most $O(\log m)$ events. Since each clause has k variables, we can do exhaustive search in each piece in polynomial time to find a satisfying assignment whose existence is guaranteed by the local lemma in phase ①.

The difficult part is to show that each piece is of size $O(\log m)$, by a careful counting argument.

<u>**Recent Breakthough**</u> by Robin Moser.

The algorithm is surprisingly simple.

First fix an ordering of the clauses, say $C_1, C_2, \ldots, C_m$

Solve-SAT

```
  Find a random assignment of the variables.
  For $1 \le i \le m$
      if $C_i$ is not satisfied
          FIX($C_i$)
```

FIX(C)

```
  Replace the variables in $C_i$ by new random values.
  While there is a clause D that share variables with C and D is not satisfied
      Choose such D with the smallest index
      FIX(D)
```

Note that once we called FIX(C_i), C_i will remain satisfied after each FIX(C_j) for $j > i$, by the definition of FIX(C_j).

So, we just need to prove that FIX(C_1) terminates in a reasonable amount of time, although at first glance it seems that it could have gone into infinite loop,

Moser came up with a remarkable proof, proving that this algorithm terminates very quickly, otherwise one can obtain an algorithm to compress l random bits using fewer than l bits, which is impossible (see MU chapter 9)!

First, suppose the algorithm has run for t step but not successful yet, how many random bits it has used?

Initially, we used n random bits for the initial assignment.

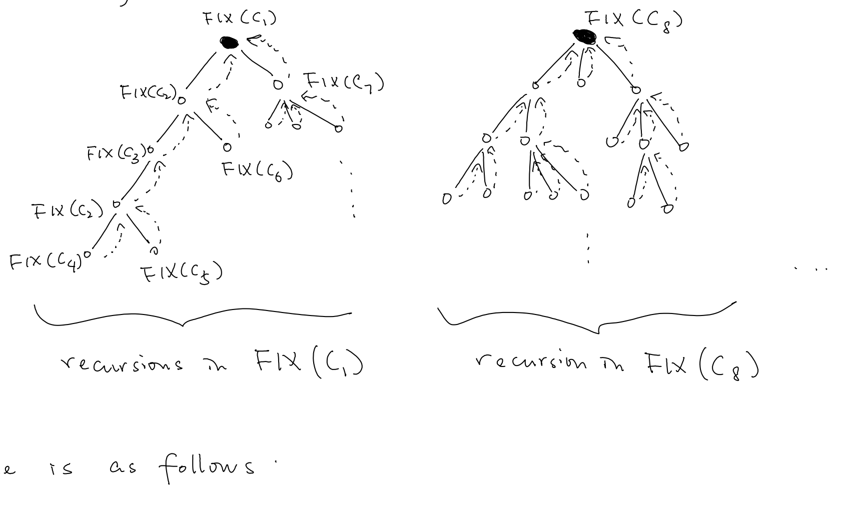
Then, for each FIX(C), we used k random bits.

So, the total number of random bits used is $n + tk$.

Now, we show how to compress the random bits if t is large.

The idea is to trace the execution of the algorithm.

(in the main loop in SOLVE)



An encoding scheme is as follows:

① use $00 + \log(m)$ bits to represent the clauses in the root nodes

② use $01 + \log(d)$ bits to represent the next clause fixed in the recursion tree. why $\log(d)$ bits are enough? Because each clause shares variables with at most d other clauses

③ use 10 to represent the end of a recursive call, that is, to represent the back arrows in the figure.

④ when the algorithm ends, remember the n bits in the variables.

How many bits we used?

- There are at most m roots, since after calling $FIX(C_i)$, C_i will remain satisfied after calling $FIX(C_j)$ for $j>i$. So at most $m(\log m+2)$ bits are used for ①.
- There are t steps in the algorithm. So at most $t(\log d+2)$ bits for ② + ③.
- Finally n bits are needed for ④.

Therefore, the total number of bits used is $m(\log m+2) + t(\log d+2) + n$.

Okay, if we can show that we can reconstruct the original random bits from the encoding, then since random bits cannot be compressed, we must have

$$m(\log m+2) + t(\log d+2) + n \geq tk + n$$

$$\Rightarrow \quad m(\log m+2) \geq t(k-\log d-2)$$

So, if $d < 2^{k-2}$, then $t = O(m\log m)$. That is, the expected value of t is $O(m\log m)$, i.e. the algorithm terminates quickly.

Finally, we just need to show that with the encoding one can reconstruct the original random bits.

Let the original random bits be

$$v_1 v_2 \ldots v_n \, r_1^{(1)} r_2^{(1)} \ldots r_k^{(1)} \, r_1^{(2)} r_2^{(2)} \ldots r_k^{(2)} \ldots\ldots r_1^{(t)} r_2^{(t)} \ldots r_k^{(t)}$$

where $v_1 \ldots v_n$ are the initial random bits on the variables, and $r_1^{(i)} \ldots r_k^{(i)}$ are the random bits used in the i-th step.

So the algorithm will maintain n variables, initially it is $v_1 \ldots v_n$.

The algorithm does not know the values of these variables, and it tries to find out.

Now the algorithm follows the execution tree to figure out the variables.

If the algorithm fixes the clause i, say clause i has variables $\{x_1, x_2 \dots x_k\}$,

then it must be the case that $\{v_1, v_2 \dots v_k\}$ must violate the clause i.

The crucial observation is that there is only ONE $\underset{\text{setting (out of } 2^k \text{ settings)}}{\wedge}$ of the k variables that make

clause i unsatisfied. So we can recover the values of the variables $v_1, v_2, \dots, v_k$.

Then we know these variables will be replaced by $r_1^{(1)}\ r_2^{(1)} \dots r_k^{(1)}$.

The compression algorithm will maintain the variables $\{r_1^{(1)}, r_2^{(1)}, \dots, r_k^{(1)}, v_{k+1}, \dots, v_n\}$

in the next step.

Then, again, we know which clause is going to be fixed next, we learn

k bits of $\{r_1^{(1)}, \dots r_k^{(1)}, v_{k+1}, \dots, v_n\}$ and can replace k variables by

$r_1^{(2)} \dots r_k^{(2)}$, and so on, until the algorithm stops. Since we have stored the

final n bits, we can recover the values of all original bits. $\square$

There are many follow-up work in this direction; see the project page.

It is very interesting that probabilistic methods can be made efficient.,

e.g. now we can use this method to find a good schedule for the packet routing problem.

In other words, local lemma has become an algorithmic tool, i.e. if we can prove that something exists by the local lemma, then we can also find it efficiently.

<u>References</u>

The original proof, the simple application and Beck's framework are from Chapter 6 of MU.

The algorithmic proof is from the talk of the paper "A constructive proof of Lovász local lemma" by Moser.

The packet routing result is from the paper "Packet routing and job-shop scheduling in O(congestion + dilution) steps by Leighton-Maggs-Rao. You can also see a simpler proof by Rothvoss in the paper "A simpler proof of O(congestion + dilution) on packet routing".

There are generalizations of the local lemma that are useful when events have different probabilities and different dependencies. Read the book "The probabilistic method" by Alon and Spencer for more.