CS 761 : Randomized Algorithms , Spring 2018 , Waterloo

Lecture 13: Local lemma

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

We will see some basic examples and an interesting application, and then a recent breakthrough by Moser in making the method constructive.

Lovász local lemma

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

A typical goal in probabilistic method is to show that there is an outcome with no bad events occured.

For example, in generating Ramsey graphs, the bad events are that some subsets are monochromatic.

Our goal is to show that $\Pr\left(\bigcap_{i=1}^{n} \overline{E_i}\right) > 0$, an outcome with no bad events.

There are some situations when this is easy to show:

- when the events $E_1, E_2, \dots, E_n$ are mutually independent;
- when $\sum_{i=1}^{n} \Pr(\overline{E_i}) < 1$, i.e. when the union bound applies.

Local lemma can be seen as a clever combination of these. We can think of it as a "local union bound".

We say that an event E is mutually independent of the events $E_1, E_2, \dots, E_n$ if for any subset $I \subseteq [n]$, we have $\Pr\left(E \mid \bigcap_{i \in I} E_i\right) = \Pr(E)$, i.e. $\Pr(E)$ doesn't change no matter what happened in $E_1, \dots, E_n$.

Definition A dependency graph for a set of events $E_1, \dots, E_n$ is a graph $G = (V, E)$ with $V = \{1, \dots, n\}$ and for $1 \le i \le n$ the 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 maximum degree in the dependency is at most d
3. $4dp \le 1$.

Then $\Pr\left(\bigcap_{i=1}^{n} \overline{E_i}\right) > 0$.

The local lemma can be interpreted as if the union bound applies locally then there exists a good outcome.

The following is the original proof by Lovász, which is non-constructive: It only proves the existence of a good outcome, without an efficient algorithm to find such a good outcome.

We will present the recent algorithmic proof later, so the original proof is optional.

Proof (optional) We prove by induction that $\Pr(\bigcap_{i\in S} \overline{E_i}) > 0$ on the size of S.

To prove this, there is an intermediate step showing that $\Pr(E_k \mid \bigcap_{i\in S} \overline{E_i}) \le 2p$.

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

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

$\Rightarrow \dots$

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

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

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

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

Then, $\Pr(\bigcap_{i=1}^{\ell} \overline{E_i}) = \prod_{i=1}^{\ell} \Pr(\overline{E_i} \mid \bigcap_{j=1}^{i-1} \overline{E_j})$ // conditional probability

$= \prod_{i=1}^{\ell} (1 - \Pr(E_i \mid \bigcap_{j=1}^{i-1} \overline{E_j}))$

$\ge \prod_{i=1}^{\ell} (1 - 2p) > 0.$ // induction hypothesis

Next, we prove $\Pr(E_k \mid \bigcap_{i\in S} \overline{E_i}) \le 2p$ assuming the previous steps in the chain are proven.

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

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

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

So, we assume $|S| > |S_2|$.

Let $F_S = \bigcap_{i\in S} \overline{E_i}$, $F_{S_1} = \bigcap_{i\in S_1} \overline{E_i}$ and $F_{S_2} = \bigcap_{i\in S_2} \overline{E_i}$, so that $F_S = F_{S_1} \cap F_{S_2}$.

Then, $\Pr(E_k \mid F_S) = \frac{\Pr(E_k \cap F_S)}{\Pr(F_S)} = \frac{\Pr(E_k \cap F_{S_1} \mid F_{S_2}) \Pr(F_{S_2})}{\Pr(F_{S_1} \mid F_{S_2}) \Pr(F_{S_2})} = \frac{\Pr(E_k \cap F_{S_1} \mid F_{S_2})}{\Pr(F_{S_1} \mid F_{S_2})}$.

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

The denominator is $\Pr(F_{S_1} \mid F_{S_2}) = \Pr(\bigcap_{i\in S_1} \overline{E_i} \mid \bigcap_{j\in S_2} \overline{E_j})$

$= 1 - \Pr(\bigcup_{i\in S_1} E_i \mid \bigcap_{j\in S_2} \overline{E_j})$

$\ge 1 - \sum_{i\in S_1} \Pr(E_i \mid \bigcap_{j\in S_2} \overline{E_j})$ // union bound

$\ge 1 - \sum_{i\in S_1} 2p$ // induction hypothesis, as $|S_2| < |S|$

$\ge 1 - 2dp$ // max degree assumption ②

$\geq \frac{1}{2}$. // assumption (3)

Plug them back, we have $\Pr(E_k \mid F_S) \leq p/(\frac{1}{2}) = 2p$. This completes the induction step. □

Basic examples

We show two classical examples of applications of local lemma.

(1) k-SAT

Given a boolean formula with exactly k variables in each clause, we would like to find a truth assignment to the variables such that every clause is satisfied, where each clause is a disjunction of k variables,

e.g. $(x_1 \vee x_2 \vee x_4) \wedge (\bar{x}_1 \vee x_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee x_4)$ and $x_1 = T, x_2 = F, x_3 = T, x_4 = T$ is a satisfying assignment.

This problem is NP-complete in general, but we can prove that if each variable appears in not too many clauses, then there is always a satisfying assignment.

Theorem If no variables 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 set to true with probability $1/2$ independently.

Let E_i be the bad event that the i-th clause is not satisfied by the random assignment.

Since each clause is a disjunction of k variables, this event happens with probability $p = \Pr(E_i) = \frac{1}{2^k}$.

Note that the event E_i is mutually independent with other events that do not share variables with E_i.

So, the maximum degree d in the dependency graph is at most $kT \leq k(2^k/4k) = 2^{k-2}$.

Since $4dp \leq 4(2^{k-2})(2^{-k}) = 1$, there is an outcome with no bad events, hence a satisfying assignment. □

(2) Edge-disjoint paths

Given a graph with k pairs $\{(s_1,t_1),(s_2,t_2),\ldots,(s_k,t_k)\}$, we would like to find a path P_i connecting s_i and t_i such that the paths $\{P_1, P_2, \ldots, P_k\}$ are edge disjoint.

This problem is also NP-complete, but we can use local lemma to show that there is always a solution if the possible paths do not share too many edges with each other.

Theorem For each $1 \leq i \leq k$, let $\mathcal{P}_i$ be a collection of L paths connecting s_i and t_i.

Suppose each path in $\mathcal{P}_i$ does not share edges with more that C paths in $\mathcal{P}_j$ for $i \neq j$ and $8kC/L \leq 1$.

Then there is a way to choose $P_i \in \mathcal{P}_i$ so that the paths $\{P_1, \ldots, P_k\}$ are edge-disjoint.

Proof Consider a random experiment that for $1 \le i \le k$, we choose a random path $P_i \in \mathcal{P}_i$ connecting s_i and t_i.

Let $E_{i,j}$ be the bad event that P_i and P_j are not edge disjoint.

Since a path in $\mathcal{P}_i$ share edges with at most C paths in $\mathcal{P}_j$ and there are L paths in $\mathcal{P}_i$,

we have $p = Pr(E_{i,j}) \le C/L$.

Since $E_{i,j}$ is mutually independent with all events $E_{i',j'}$ when $i' \notin \{i,j\}$ and $j' \notin \{i,j\}$,

we have the maximum degree d in the dependency graph is at most $2k$.

As $4dp \le 8kC/L \le 1$ by our assumption, the local lemma implies that there is an outcome

of the experiment with no bad events, hence an edge-disjoint path solution. □

Packet routing

We are given an undirected graph, and k pairs, where each pair has a source vertex s_i, a destination vertex t_i,

and a path P_i. We would like to send a packet from s_i to t_i along path P_i, for $1 \le i \le k$.

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

A packet can wait at a node at any time step. (We assume that the memory at each node is enough.)

A schedule for a set of packets specifies the timing of the movements of the packets along their

respective paths (i.e. when to move and when to wait).

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

Let d be the maximum distance travelled by a packet (i.e. maximum path length).

Let c be the maximum number of packets that must transverse a single edge (i.e. maximum congestion).

Then, it is clear that any schedule must require at least $\Omega(c+d)$ 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 it is independent of k.

Instead of using Chernoff + union bound, the proof uses Chernoff bound + local lemma (local union bound).

For simplicity, we just show a weaker result, which is still independent of k.

Theorem There is a schedule with $O\left((c+d)(1+\alpha)^{O(\log^*(c+d))}\right)$ number of steps,

where α is a constant and $\log^*(n)$ is the number of log it takes to bring n to ≤ 1.

(Note that $\log^*(n)$ grows very slowly, e.g. $\log_2^*(2^{65536} - 1) = 5$.)

Proof We assume without loss of generality that $c = d$.

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 procedure recursively, i.e. add a random delay in $[1, \alpha d']$, partition the time into periods of $\ln d' = \ln\ln d$ steps, so that each period has maximum congestion at most $\ln d' = \ln\ln d$, and so on.

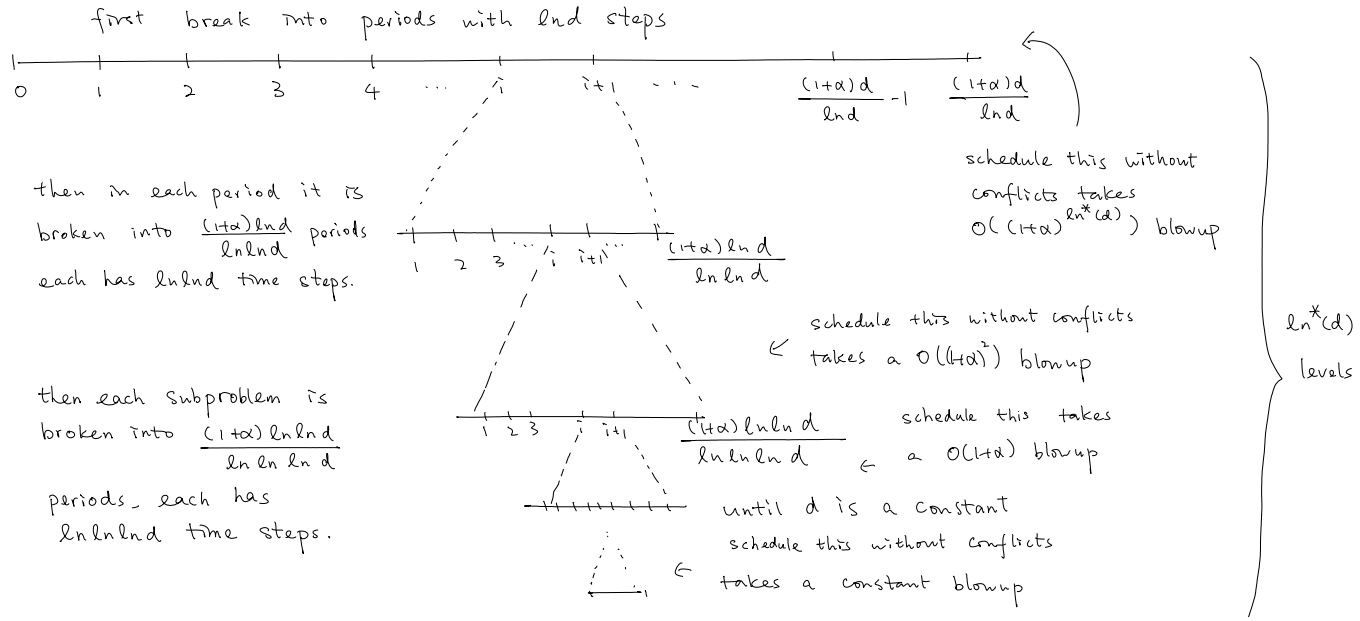
The base case is when c and d are constants, in which case the naive $O(cd)$ algorithm is also $O(c+d)$.

To reduce the problem into constant distance and constant congestion, we just need $O(\log^*(c+d))$ recursions.

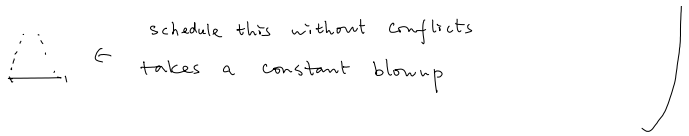
In each level, we blow up the schedule by a factor of $1+\alpha$.

So, the final schedule takes $O((c+d)(1+\alpha)^{O(\log^*(c+d))})$ steps, and each edge is used by at most one packet at a time.

Pictorically, the proof goes like this



$\ln\ln\ln d$ time steps.



So, to finish the proof, it remains to prove the following lemma using Cheroff bound + local lemma. □

Lemma When α is a large enough constant, each period has maximum congestion $\le \ln d$ with positive probability.

proof 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 $\le cd = d^2$.

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 is an initial random delay in $[1, \alpha d]$, the probability that a packet uses an edge e in a particular period is at most $\ln d / (\alpha d)$.

As there are $c = d$ packets, $E[\#\text{ packets using } e \text{ in a period}] \le \ln d / \alpha$. Call this number μ.

By Chernoff bound,

$$\begin{aligned}
&\Pr[\text{congestion of an edge at a period} > \ln d] \\
&= \Pr[\text{congestion of an edge at a period} > (1+(\alpha-1))\mu] \\
&\le \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu} \le \left(\frac{e}{1+\delta}\right)^{(1+\delta)\mu} \\
&= \left(\frac{e}{\alpha}\right)^{\alpha \cdot \frac{\ln d}{\alpha}} \quad \text{since } \mu = \ln d/\alpha \text{ and } \delta = \alpha - 1 \\
&= \left(\frac{e}{\alpha}\right)^{\ln d} << \frac{d^{-4}}{\alpha} \quad \text{for say } \alpha \ge e^{11} \text{ (a large constant)}
\end{aligned}$$

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

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

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.

Recent breakthrough by Robin Moser.

The algorithm is surprisingly simple. It was known to the experts but no one knew how to analyze it.

Algorithm

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

Solve-SAT // the main program

- Find a random assignment of the variables.
- For $1 \le i \le m$

If C_i is not satisfied

Fix(C_i)

Fix(C) // subroutine

- Substitute the variables in C with new random values
- While there is a clause D that shares variables with C and D is not satisfied

  Choose such a D with the smallest index. Fix(D).

Analysis: Note that once we called Fix(C_i) and returned to the loop in the main program, C_i will remain satisfied after each Fix(C_j) is finished in the loop for $j > i$, by the recursive fixing nature of Fix(C_j).

So, when the main loop is finished, all the clauses will be satisfied.

But it seems that the program can run into an infinite loop and never finished.

And it seems very difficult to analyze this algorithm with such a complicated dependency structure.

Idea: Moser came up with a remarkable proof. He showed that if the algorithm does not terminate in a reasonable amount of time, then we can compress random bits using fewer bits, which is impossible! (See MU chapter 9.)

Random bits: Suppose the algorithm runs for t steps but not successful yet, how many random bits used?

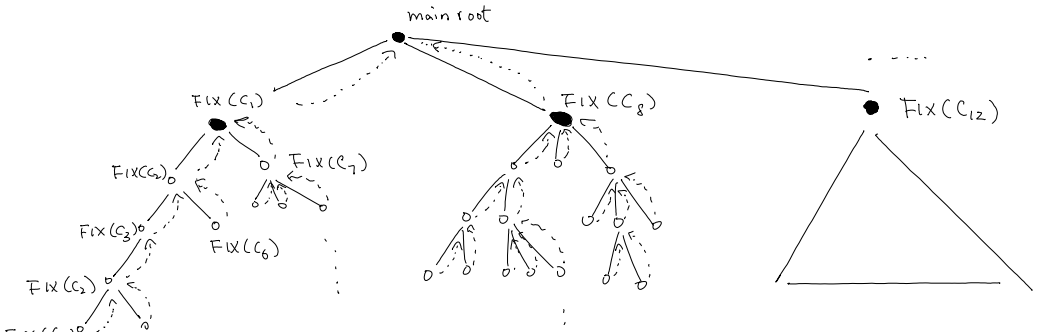
In the beginning, we used n random bits for the initial assignment.

Then, for each Fix(C), we used k random bits for the clause C.

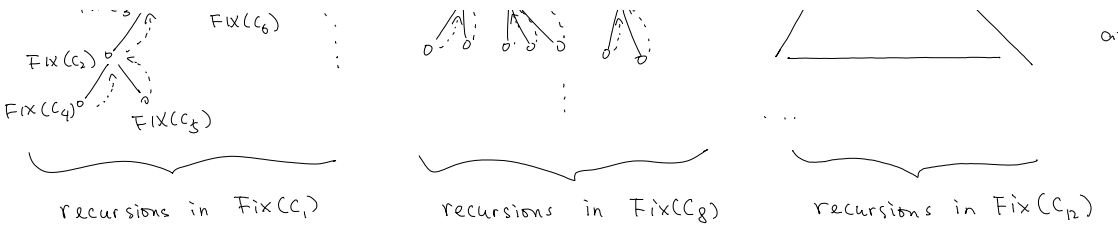
So, the total number of random bits used is $n + tk$.

Encoding: Now, we show how to compress these random bits if t is large enough.

The idea is to trace the execution of the algorithm.



and so on.



The encoding scheme is as follows:

① Use $0 + \log_2 m$ bits to represent going from the root to a clause, where the $\log_2 m$ bits is used to represent which clause to go to.

② Use $0 + \log_2 d$ bits to represent going from the current clause to another clause in the recursion tree.

Why $\log_2 d$ bits is enough to specify which clause to go to?

Because each clause shares variables with at most d other clauses, so we just need to remember which neighbor of the current clause that we go to.

③ Use 1 to represent going up, i.e. the back arrow in the recursion tree.

When we see 1 at the root, the algorithm terminates.

④ When the algorithm terminates, we remember the n bits in the variables.

Compression It should be clear from the encoding we can recover the execution of the algorithm.

How many bits have we used?

- As the main root only calls m clauses (by the previous argument that C_i remained satisfied once Fix(C_i) is called), we used at most $m(\log_2 m + 2)$ bits for ①+③.
- There are t steps in the algorithm, so at most $t(\log_2 d + 2)$ bits for ②+③.
- Finally, n bits for ④.

Therefore, the total number of bits used in the encoding scheme is at most $m(\log_2 m + 2) + t(\log_2 d + 2) + n$.

Suppose we can prove that the original bits can be recovered from the encoding scheme.

Since random bits cannot be compressed, we must have

$$m(\log_2 m + 2) + t(\log_2 d + 2) + n \geq n + tk \quad \Rightarrow \quad m(\log_2 m + 2) \geq t(k - \log_2 d - 2).$$

So, if $d < 2^{k-2}$, then $t = O(m \log_2 m)$, which implies that the algorithm terminates quickly.

Decoding: To finish the proof, we just need to show how to recover the random bits from the encoding

Let the original random bits that the algorithm used be $v_1, v_2, \ldots, v_n, r_1^{(1)}, r_2^{(1)}, \ldots, r_k^{(1)}, r_1^{(2)}, r_2^{(2)}, \ldots, r_k^{(2)}, \ldots, r_1^{(t)}, r_2^{(t)}, \ldots, r_k^{(t)}$,

where $v_1, v_2, \ldots, v_n$ are the random bits in the initial assignment and $r_1^{(i)}, r_2^{(i)}, \ldots, r_K^{(i)}$ are the K random bits in i-th step.

The decoding algorithm will maintain n variables, the current assignment, initially $v_1, v_2, \ldots, v_n$.

The algorithm does not know the values of these variables in the beginning, but it will try to find out by following the execution that the encoding algorithm has stored.

If the algorithm fixes clause i, say clause i has variables $x_{i_1}, x_{i_2}, \ldots, x_{i_K}$, then it means that the corresponding bits $v_{i_1}, v_{i_2}, \ldots, v_{i_K}$ in the current assignment must violate the clause i.

The crucial point is that there is only one possibility of $v_{i_1}, v_{i_2}, \ldots, v_{i_K}$ to violate the clause.

So, by knowing that the algorithm fixed clause i, we learn K bits of the current assignment.

Then, we know that the algorithm will replace the bits $v_{i_1}, v_{i_2}, \ldots, v_{i_K}$ by $r_1^{(i)}, r_2^{(i)}, \ldots, r_K^{(i)}$, and so we update the current assignment (that we are trying to find out).

We repeat this procedure: knowing what clause that the algorithm is trying to fix (by following the execution tree), learning K bits of the current assignment, replacing the k bits by $r_1^{(j)}, r_2^{(j)}, \ldots, r_K^{(j)}$ in the next step.

We stop when the algorithm stops, and we use the final n bits stored to recover the remaining variables.

So, we can recover all the random bits by using the encoding that we have stored.

To summarize, the key point is that each step we used K random bits, while we can keep track of by using only $\sim \log_2 d$ bits (which neighbor to fix). So, when $\log_2 d < K$, we have a contradiction.

Discussion: Moser's result sparked a lot of subsequent work. If you are interested, it is a good project topic.

It is very interesting that probabilistic methods can be turned to efficient algorithms.

So, now local local is not only a powerful probabilistic method, but also a powerful algorithmic tool.

If we have time, we can see another example of this direction - making probabilistic method constructive.

References

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.