CS 761 : Randomized Algorithms , Spring 2023, Waterloo

Lecture 7: Local Lemma

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

We will see two basic examples and an interesting recent application in "palette sparsification".

Finally, we will study the breakthrough by Moser who made this method constructive.

Lovász local lemma

Let $E_1, E_2, \ldots, E_n$ 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 cliques / independent sets.

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

There are some situations when this is easy to show:

- when the events $E_1, E_2, \ldots, E_n$ are mutually independent;
- when $\sum_{i=1}^{n} \Pr(E_i) < 1$, 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 $E_1, E_2, \ldots, E_n$ if for any subset $I \subseteq [n]$,

we have $\Pr(E \mid \bigcap_{i \in I} E_i) = \Pr(E)$, i.e. $\Pr(E)$ doesn't change conditioned on the events $\{E_i \mid i \in I\}$.

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

① $\Pr(E_i) \leq p$ for $1 \leq i \leq n$.

② Every event is mutually independent of all but at most d other events.

③ $4dp \leq 1$.

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

We sometimes call d the maximum degree in the "dependency graph".

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 $\Pr(\bigcap_{i \in S} \bar{E}_i) > 0$ on the size of S.

To prove this, there is an intermediate step showing that $\Pr(E_k \mid \bigcap_{i \in S} \bar{E}_i) \leq 2p$.

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

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

$\Rightarrow \dots$

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

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, as $\Pr(\bar{E}_i) = 1 - \Pr(E_i) = 1-p > 0$.

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

Then, $\Pr(\bigcap_{i=1}^{\ell} \bar{E}_i) = \prod_{i=1}^{\ell} \Pr(\bar{E}_i \mid \bigcap_{j=1}^{i-1} \bar{E}_j)$ // conditional probability

$= \prod_{i=1}^{\ell} (1 - \Pr(E_i \mid \bigcap_{j=1}^{i-1} \bar{E}_j))$

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

Next, we prove $\Pr(E_k \mid \bigcap_{i\in S} \bar{E}_i) \le 2p$ 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:

$S_1 := \{ i \in S \mid (i,k) \text{ dependent} \}$ and $S_2 := \{ i \in S \mid (i,k) \text{ independent} \}$.

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.

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

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

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

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 $\Pr(F_{S_1} \mid F_{S_2}) = \Pr(\bigcap_{i\in S_1} \bar{E}_i \mid \bigcap_{j\in S_2} \bar{E}_j)$

$= 1 - \Pr(\bigcup_{i\in S_1} E_i \mid \bigcap_{j\in S_2} \bar{E}_j)$

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

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

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

$\ge \frac{1}{2}$. // assumption ③

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

Applications

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.

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

<u>Proof</u> 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 is at most $kT \le k(2^k/4k) = 2^{k-2}$.

Since $4dp \le 4(2^{k-2})(2^{-k}) = 1$, there is an outcome avoiding all bad events, hence a satisfying assignment. □

② Graph Coloring

This is an easy application, when we define the right bad events.

<u>Theorem</u> Let $G = (V, E)$ be an undirected graph. Suppose each vertex $v \in V$ is associated with a set $S(v)$ of $8r$ colors where $r \ge 1$. Suppose, also, that for each vertex $v \in V$ and each color $c \in S(v)$, there are at most r neighbors u of v such that c is in $S(u)$.

Then there exists a proper coloring of G assigning to every vertex v a color from $S(v)$ so that, for any edge $uv \in E$, the colors assigned to u and v are different.

<u>Proof</u> Consider a random coloring where each vertex v is assigned a uniform random color from $S(v)$ independently.

Let $B_{e,c}$ be the bad event that both endpoints of e are assigned color c. Note that we only define such a bad event $B_{uv,c}$ when $c \in S(u)$ and $c \in S(v)$.

Then, since each vertex v is assigned a uniform random color independently, $p = \Pr(B_{e,c}) = \frac{1}{(8r)^2} = \frac{1}{64r^2}$.

Note that the event $B_{e,c}$ is mutually independent with other events $B_{e',c'}$ besides those where e and e' share a vertex and the color c' is available in both endpoints of e'.

Since each vertex v has $8r$ colors and each color $c \in S(v)$ appears in at most r neighbors of v, an event $B_{e,c}$ is mutually independent with all but at most $d = 2 \cdot 8r \cdot r = 16r^2$ events.

Therefore, $4pd = 4\left(\frac{1}{64r^2}\right) \cdot (16r^2) \le 1$, there is an outcome that avoids all bad events, hence a proper coloring. □

Palette Sparsification

There are many interesting and non-trivial applications of the local lemma.

See CS761 2018 for a surprising and famous result in packet routing.

Here we show a more recent application of "palette sparsification" that applies the above result for graph coloring in a clever way. with interesting consequences in sublinear-time / streaming / distributed graph coloring.

The idea is to do random sampling on the set of available colors so that a proper coloring still exists. using the theorem above.

A standard application of Chernoff bound will prove that $O(\log n)$ colors on each vertex suffice, but a smarter use of Chernoff bound will prove that $O(\sqrt{\log n})$ colors suffice.

Theorem Let $G=(V,E)$ be an undirected graph with maximum degree Δ. Let $K = 24\Delta$.

Suppose, for each vertex $v \in V$, we independently sample a set $L(v)$ of $30\sqrt{\log n}$ uniform random colors from the colors $\{1, 2, \ldots, K\}$.

Then, with high probability, there exists a proper coloring of G from set $L(v)$ for every $v \in V$.

Proof The idea is to show that the previous theorem for graph coloring applies after random sampling.

Let $\ell = \alpha\sqrt{\log n}$ be the size of the set $L(v)$ and $K = \beta\Delta$. We will only plug $\alpha = 30$ and $\beta = 24$ in the end.

Let $d_L(v,c) := |\{u \mid c \in L(u) \text{ and } uv \in E\}|$ be the c-degree of v after sampling.

For any $v \in V$ and $c \in L(v)$. the expected c-degree is $\mathbb{E}[d_L(c,v)] = \sum_{u: uv \in E} \Pr(c \in L(u)) \leq \Delta \cdot \frac{\ell}{K} = \frac{\ell}{\beta}$.

Since the indicator variables whether $c \in L(u)$ are independent, by Chernoff bound in L03, with $\varepsilon = 1$,

$$\Pr\left(d_L(c,v) \geq 2 \cdot \frac{\ell}{\beta}\right) \leq e^{-\frac{\ell}{3\beta}}.$$

Here, if we set $\ell \gg 3\beta \cdot \log n$, then this probability would be at most say $1/n^5$, and so $d_L(c,v) \leq 2 \cdot \frac{\ell}{\beta}$ for all $v \in V$ and all $c \in L(v)$ with high probability by union bound.

Then, choosing $\beta = 16$ would guarantee that each vertex has ℓ colors while the c-degree is at most $\frac{\ell}{8}$, and thus we can finish the proof by using the previous theorem obtained by local lemma.

To prove that $\ell \approx \sqrt{\log n}$ suffices, we use a deterministic modification trick similar to that in L01.

We say a color c is bad for v if $d_L(c,v) \geq 2 \cdot \frac{\ell}{\beta}$.

Since conditioned on a color being bad on v can only decrease the chance that other colors are bad on v,

so $\Pr(\#\text{ of bad colors on } v \geq \frac{\ell}{2}) \leq \binom{\ell}{\ell/2} \cdot \left(e^{-\frac{\ell}{3\beta}}\right)^{\frac{\ell}{2}} \leq 2^{\ell} \cdot e^{-\frac{\ell^2}{6\beta}} = 2^{\ell} e^{-\frac{\alpha^2 \log n}{6\beta}} \leq n^{-6}$ if $\alpha = 6\sqrt{\beta}$.

Therefore, with probability at least $1 - n^{-5}$, no vertices have more than $\frac{\ell}{2}$ bad colors.

Assume this happens. Now, for each vertex v, we remove all the colors bad for v.

Then, the number of colors remained in each vertex is at least $\frac{\ell}{2} = \frac{\alpha\sqrt{\log n}}{2} = 3 \cdot \sqrt{\beta}\sqrt{\log n}$,

while the c-degree of each color on $L(v)$ is at most $\frac{2\ell}{\beta} = \frac{2\alpha\sqrt{\log n}}{\beta} = \frac{12\sqrt{\log n}}{\sqrt{\beta}}$.

By setting $\beta = 24$, then the number of colors remained is at least 8 times the c-degree $\forall c \in L(v)$.

Therefore, we can apply the previous theorem by local lemma to prove the existence of a proper coloring

using the colors remained in each vertex (with $r = 12\sqrt{\log n}/\sqrt{\beta}$). □

The above theorem is to illustrate the main idea without optimizing the constants.

The proof is from [AA20], where a stronger bound $k = (1+\varepsilon)\cdot\Delta$ would still work with $|L(v)| \leq O_\varepsilon(\sqrt{\log n})$.

In [ACK19], where palette sparsification was first introduced, it was proved that $k = \Delta + 1$ would work

with list size $|L(v)| = \Theta(\log n)$.

This can be used to sparsify the graph to obtain sublinear time algorithm for graph coloring.

See [ACK19] and [AA20] for more results and applications.

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, which some researchers call "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 non-constructive, 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 chapter 6.8 of [MU] for details.

For this framework to work, a stronger condition is assumed: each variable appears in at most

$T = 2^{\alpha k}$ for some constant $0 < \alpha < 1$ (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 Breakthrough**</u> by Robin Moser (2009)

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

<u>Algorithm</u>

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

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).

<u>Analysis</u>: 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.

<u>Idea:</u> 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!

<u>Claim</u> A random string of k bits can be compressed into k-c bits with probability at most 2^{-c} for any $c \ge 1$.

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

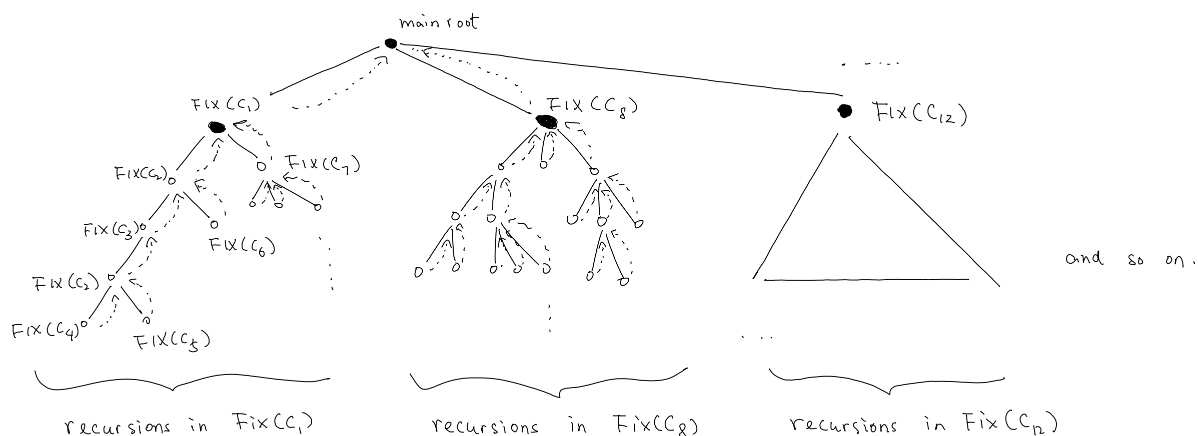
In the beginning, we used n random bits for the initial assignment.

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.



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 after 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 random bits can be recovered from the encoding scheme.

Since it is unlikely to compress random bits, we can conclude that with probability $\geq 1-2^{-c}$,

$\#$ encoding bits $\geq$ $\#$ random bits $- c$.

In our setting, $m(\log_2 m + 2) + t(\log_2 d + 2) + n \geq \#$ encoding bits $\geq \#$ random bits $- c = n + tk - c$.

This implies that $m(\log_2 m + 2) + c \geq t(k - \log_2 d - 2)$ with probability $\geq 1 - 2^{-c}$.

So, as long as $k - \log_2 d - 2 \geq \varepsilon \Leftrightarrow d \leq 2^{k-2-\varepsilon}$ for some $\varepsilon > 0$, then we have

$$t \leq \frac{m(\log_2 m + 2) + c}{\varepsilon} \quad \text{with probability} \geq 1 - 2^{-c}.$$

For example, if $d \leq 2^{k-3}$ (so $\varepsilon = 1$), then $t \leq m\log_2 m + 3m$ with probability at least $1 - 2^{-m}$.

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 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 unknown 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 in each step the algorithm injects k random bits, but we only used $\sim \log_2 d + 2$ bits in the encoding to keep track of it (i.e. which neighbor to fix).

and so when $\log_2 d + 2 < k$, we are compressing, but this should not be happening.

Discussion: Moser's result sparked a LOT of subsequent work.

It is very interesting that probabilistic methods can be turned into 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 basic examples, and the Beck 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.

Read the book "probabilistic methods" for different generalizations of local lemma, or from the project page the recent papers that made these generalizations constructive as well.

The following are two references for palette sparsification.

[ACK19] Assadi, Chen, Khanna. Sublinear algorithms for $(\Delta+1)$ vertex coloring, 2019.

[AA20] Alon, Assadi. Palette sparsification beyond $(\Delta+1)$ vertex coloring, 2020.