CS 798 - Algorithmic Spectral Graph Theory, Fall 2015, Waterloo

Lecture 23: Local lemma

We first review about what we have studied in the course.

Then we end it with an inspiring result about Lovász local lemma.

Review

In this course, we have studied some basic results and techniques in spectral graph theory, and have seen its applications in approximation algorithms, fast algorithms, and in combinatorics.

In the first half of the course, we mostly study about graph partitioning and graph expansion, and also about random walks.

Then, we took two weeks to discuss some recent developments in designing fast algorithms using related techniques.

In the last part, we study graph sparsification and the new probabilistic method using techniques about multivariate polynomials.

Problems

In terms of combinatorical problems, some highlights are graph partitioning, unique games, maximum flow, graph sparsification, traveling salesman problems.

These are important problems in theoretical computer science and look purely combinatorial.

From their definitions, it wasn't clear that there is any connection to spectral graph theory, but surprisingly spectral techniques give the best known results for these problems.

Relaxations

I don't think spectral methods are very general and universal; all these problems have nice quadratic forms and are about cut structures of graphs, for which the spectral formulations are very useful in relaxing the problem and allowing us to use linear algebra.

But looking more broadly, including convex optimization techniques like LP and SDP, then I think it is fair to say most of combinatorial problems are benefitted from the ideas and techniques in the continuous world.

This seems to be the big trend now, where analytical and combinatorial techniques are combined to obtain amazing results.

Techniques

Let's also review the useful ideas that we have studied.

In the first half, we see the connection between eigenvalues and optimization through Rayleigh quotients, and from it we understand the spectrum (not just the second eigenvalue) and its relations to graph expansion. Of course, we learn that Cheeger's rounding (and Cauchy-Schwarz) is useful in processing the eigenvectors (or random walk vectors).

We also use the spectral decomposition to study random walks and analyze mixing time, and see how these ideas are more natural for the inductive constructions of expander graphs.

Then, we study electrical networks and see how concepts like effective resistances are useful in understanding hitting times, random spanning trees, and graph sparsifications.

We see a bit of convex optimization techniques, including convergence, duality, multiplicative updates to design fast algorithms. These techniques are quite general and useful.

Then, we learnt some Chernoff bounds (scalar and matrix forms) and see its applications in random sampling.

The barrier arguments (from convex optimation) then came in, outperformed the random sampling.

Finally, we study real-stable polynomials and see its surprising applications in probabilistic method and its uses in studying probability distributions.

As in most theory courses, the most important elements are often not the theorems, but the ideas, the way of thinking, and the technical details.

I hope you will find the techniques learnt useful, and become more familar and comfortable with reading and using linear algebra in your own research.

Future directions

There are some active research areas related to this course:

- unique games / small-set expansion: using SDP to design better approximation algorithms for

finding sparse vectors, with implications in some machine learning problems.

- fast algorithms: use convex optimization to speedup combinatorial optimization.
- polynomial method: make it constructive and find new applications.

We end this course with an inspiring result (remotely) related to the last goal.

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

One can think of it as a "local union bound".

Original proof (optional)

The original inductive proof is included for reference. We won't do it in class.

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 $\Pr(E_k \mid \bigcap_{i\in S}\bar{E}_i) \le 2p$.

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 $S = \{1, 2, \dots, s\}$.

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

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 $F_S = F_{S_1} \cap F_{S_2}$.

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

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

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 $\Pr(F_{S_1} \mid F_{S_2})$

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

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

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

$$\ge 1 - 2dp$$

$$\ge 1/2.$$

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

Applications There are many applications of this lemma, but we just mention an easy one.

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 corresponding to clauses that do not share variables with E_i.

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

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

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

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 Breakthough 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 $FIX(C_j)$, 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 $FIX(C_i)$ 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!

Analysis

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.

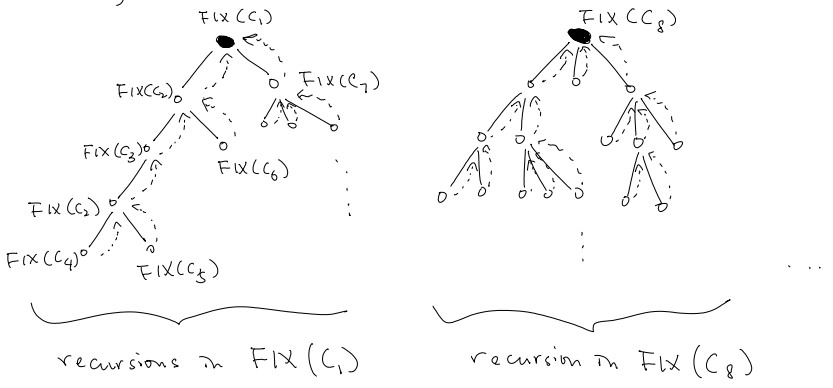
Then, for each $FIX(C)$, we used k random bits.

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 $\text{FIX}(C_j)$, C_i will remain satisfied after calling $\text{FIX}(C_j)$ 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

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

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 $v_1 \dots v_n$.

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 (setting (out of 2^k settings)) 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. □

There are many followup work and they made the general local lemma constructive.

Essentially, if one can prove that some objects exist by local lemma, there is also an

efficient (randomized) algorithm to find this object.

In other words, local lemma has become an algorithmic tool.

Discussions

First, it is an inspiring story for PhD students!

Relating to the course, in some way, the local lemma comes closest to the results obtained

by the method of interlacing families.

The result by Bilu and Linial that for every graph there is a signed matrix with

the absolute value of all eigenvalues bounded by $O(\sqrt{d \log^3 d})$ is proved by local lemma.

Related to Kadison-Singer, there is a related result by Frieze and Molloy about splitting an expander graph into two expander subgraphs, whose proof is by local lemma.

Finally, there is a paper showing a surprising connection between local lemma and roots of polynomials.

In the general setting of local lemma, we are given a dependency graph $G=(V,E)$, and we would like to ask for what vectors $p \in \mathbb{R}^V$ such that if $\Pr(E_i) \le p_i \ \forall i$, then we must have $\Pr(\bigcap_i \overline{E_i}) > 0$ (regardless of what are the events).

Call a vector p good if there is always a positive probability to avoid all bad events.

In statistical physics, the following independent set polynomial is studied.

Let $Z_G(x) = \sum_{\substack{S \subseteq V \\ S \text{ ind. set}}} \prod_{i \in S} x_i$ be the generating polynomial of independent sets of G.

Given a sequence of radii $r \in \mathbb{R}^V$, let $\overline{D_r} = \{ w \in \mathbb{C}^V : |w_i| \le r_i \ \forall i \in V \}$.

They are interested in for what vectors $r \in \mathbb{R}^V$ such that $Z_G(x)$ contains no roots in the region $\overline{D_r}$.

Scott and Sokal found a surprising connection between these two sequences.

Theorem p is good for dependency graph G if and only if $\overline{D_p}$ has no roots in the independent set polynomial $Z_G(x)$.

They also found out that the sufficient conditions proved in these two independent communities are "identical", and the inductive proofs are basically "isomorphic".

It is very interesting to study this connection further.

References

- A constructive proof of the Lovász local lemma, by Moser.
- A constructive proof of the general Lovász local lemma, by Moser and Tardos.
- On dependency graphs and the lattice gas, by Scott and Sokal.