Advanced Algorithms XI (Fall 2020)

Instructor: Chihao Zhang
Scribed by: Xinyu Mao, Ze Tang

Last modified on Nov 16, 2020

> Take what you have gathered from coincidence.
>
> *It's all over now, baby blue*
> *Bob Dylan*

Brief Review. During the lecture on Nov. 16, we first presented a proof for the asymmetric version of Lovász local lemma(theorem 3). Next, we studied the algorithmic version of Lovász Local Lemma(theorem 4), which tells us how to find a satisfying assignment for CNF formula with a simple randomized algorithm.

1 Lovász Local Lemma

For a (undirected) graph $G = (V, E)$ and $v \in V$, define

$$N(v) := \{u \in V : uv \in E\}, \quad N^+(v) := N(v) \cup \{v\}.$$

Let $\mathcal{A} := \{A_1, A_2, \ldots, A_m\}$ be a set of 'bad events'.

Definitions 1. *A graph $G = (V, E)$ is the* ***dependency graph*** *of $\mathcal{A}$ if*

1. *$V = \mathcal{A}$;*
2. *For all $A \in \mathcal{A}$, A is independent from $\mathcal{A} \setminus N^+(A)$.*

Theorem 2 (Lovász Local Lemma, symmetric version). *Let Δ be the maximum degree of the dependency graph. If the dependency graph of $\mathcal{A}$ satisfies*

$$\forall A_i \in \mathcal{A}, \ \mathbf{Pr}[A_i] \le p < 1.$$

If it holds that $4\Delta p < 1$, then

$$\mathbf{Pr}\left[\bigcap_{i=1}^{m} \overline{A_i}\right] > 0.$$

Theorem 3 (Lovász Local Lemma, asymmetric version). *Let $x : \mathcal{A} \to (0,1)$ be a function such that*

$$\mathbf{Pr}[A] \le x(A) \prod_{B \in N(A)} (1 - x(B)), \forall A \in \mathcal{A}.$$

Then

$$\mathbf{Pr}\left[\bigcap_{i=1}^{m} \overline{A_i}\right] > 0.$$

Proof. Let $S \subseteq [m], F_S = \bigcap_{i \in S} A_i$.

We start with showing that

$$\forall i \notin S, \mathbf{Pr}[A_i | F_S] \le x(A_i) \tag{1}$$

by induction on $|S|$.

Base step. In the case of $S = \emptyset$,

$$\mathbf{Pr}[A_i] \le x(A_i) \prod_{B \in N(A_i)} (1 - x(B)) \le x(A_i).$$

Induction step. Write $S_1 := N(A_i), S_2 := \mathcal{A} \backslash N^+(A_i)$. We shall give an upperbound of $\mathbf{Pr}[A_i | F_S]$. Observe that

$$\begin{aligned}
\mathbf{Pr}[A_i | F_S] &= \mathbf{Pr}\left[A_i \middle| F_{S_1} \cap F_{S_2}\right] \\
&= \frac{\mathbf{Pr}\left[A_i \cap F_{S_1} \cap F_{S_2}\right]}{\mathbf{Pr}\left[F_{S_1} \cap F_{S_2}\right]} \\
&= \frac{\mathbf{Pr}\left[A_i \cap F_{S_1} \middle| F_{S_2}\right]}{\mathbf{Pr}\left[F_{S_1} \middle| F_{S_2}\right]} \qquad \text{(by dividing out } \mathbf{Pr}\left[F_{S_2}\right]) \\
&=: \frac{X}{Y}.
\end{aligned}$$

On one hand, we try to get a upperbound of X:

$$\begin{aligned}
X &= \mathbf{Pr}\left[A_i \cap F_{S_1} \middle| F_{S_2}\right] \\
&\le \mathbf{Pr}\left[A_i \middle| F_{S_2}\right] \\
&= \mathbf{Pr}[A_i] && (A_i \text{ and } S_2 \text{ are irrelevent}) \\
&\le x(A_i) \prod_{B \in N(A_i)} (1 - x(B)) && \text{(by condition of the theorem 3).}
\end{aligned}$$

Then we will find a lowerbound of Y by induction:

$$\begin{aligned}
Y &= \mathbf{Pr}\left[F_{S_1} \middle| F_{S_2}\right] \\
&= \mathbf{Pr}\left[\bigcap_{j=1}^{r} \overline{A_j} \middle| F_{S_2}\right] && \text{(WOLG, let } S_1 = \{1, 2, \ldots, r\}\text{)} \\
&= \prod_{j=1}^{r} \mathbf{Pr}\left[\overline{A_j} \middle| \bigcap_{k<j} \overline{A_k} \cap F_{S_2}\right] \\
&= \prod_{j=1}^{r} \left(1 - \mathbf{Pr}\left[A_j \middle| \bigcap_{k<j} \overline{A_k} \cap F_{S_2}\right]\right) \\
&\geq \prod_{B \in N(A_i)} (1 - x(B)) && \text{(by induction).}
\end{aligned}$$

This establishes eq. (1).

Here comes the last strike:

$$\begin{aligned}
\mathbf{Pr}\left[\bigcap_{i=1}^{m} \overline{A_i}\right] &= \prod_{i=1}^{m} \mathbf{Pr}\left[\overline{A_i} \middle| \bigcap_{j<i} \overline{A_j}\right] \\
&= \prod_{i=1}^{m} \left(1 - \mathbf{Pr}\left[A_i \middle| F_{[i-1]}\right]\right) \\
&\geq \prod_{i=1}^{m} (1 - x(A_i)) && \text{(by eq. (1))} \\
&> 0.
\end{aligned}$$

□

Connection between two versions. If we choose x as

$$x(A_i) = \frac{1}{\Delta + 1}, \tag{2}$$

and use the condition of theorem 2, we can get a bound which is similar but a different from theorem 2. Note that

$$\begin{aligned}
x(A) \prod_{B \in N(A)} (1 - x(B)) &= \frac{1}{\Delta + 1} \prod_{B \in N(A)} \left(1 - \frac{1}{\Delta + 1}\right) \quad \text{(by eq. (2))} \\
&= \frac{1}{\Delta + 1} \left(1 - \frac{1}{\Delta + 1}\right)^{\deg(A)} \\
&\geq \frac{1}{\Delta + 1} \left(1 - \frac{1}{\Delta + 1}\right)^{\Delta} \\
&\geq \frac{1}{\Delta + 1} \cdot e^{-1}
\end{aligned}$$

If we let

$$\frac{1}{\Delta + 1} \cdot e^{-1} \geq p$$

then we will satisfy the condition of theorem 3:

$$x(A) \prod_{B \in N(A)} (1 - x(B)) \geq \frac{1}{\Delta + 1} \cdot e^{-1} \geq p \geq \mathbf{Pr}\left[A\right],$$

That is, if the Dependency Graph of $\mathcal{A}$ satisfies

$$\forall A_i \in \mathcal{A}, \mathbf{Pr}\left[A_i\right] \leq p < 1,$$

then

$$e(\Delta + 1)p < 1 \text{ implies } \mathbf{Pr}\left[\bigcap_{i=1}^{m} \overline{A_i}\right] > 0.$$

2 Algorithmic Lovász local lemma (for SAT)

Let $\phi := \bigwedge_{i=1}^{m} C_i$ ba a CNF formula with free variables $\mathcal{V} := \{x_1, x_2, \ldots, x_n\}$. An *assignment* of ϕ is a function $f : \mathcal{V} \longrightarrow \{0, 1\}$. We say assignment f *satisfies* ϕ, denoted by $f \models \phi$, if ϕ is satisfied with x_i taking the value $f(x_i)$ for every $i \in [n]$.

Let A_i be the event that the clause C_i *violates* (i.e., C_i is not satisfied). If the set of events $\mathcal{A}_\phi := \{A_i\}_{i \in [m]}$ meets the condition of theorem 3, we can assert that ϕ is satisfiable.

For $A \in \mathcal{A}_\phi$, the clause corresponding to A is denoted by $\mathtt{clause}(A)$.

2.1 The algorithm that tells how to avoid bad events

Now we go one step further: we shall devise an efficient algorithm such that if $\mathcal{A}_\phi$ satisfies the conditions in theorem 3, the algorithm outputs a satisfying assignment. As is shown in algorithm 1, the idea is simple: take a random assignment, and adjust it locally if ϕ is not satisfied.

Algorithm 1: Randomized algorithm for SAT based on local corrections

```
Input: a CNF φ := ∧_{i=1}^m C_i with V := {x_1, x_2, ..., x_n} as variables.
Output: an assignment f : V → {0,1} such that f ⊨ φ.
pick a random assignment f ;
while f ⊭ φ do
    pick an arbitrary violated clause C_j ;
    update f by resampling variables in C_j ;
end
return f;
```

As usual, let $N(A_i)$ be the neighbors of A_i in the dependency graph of $\mathcal{A}_\phi$. Then we have the following statement about algorithm 1.

Theorem 4 (Algorithmic Lovász local lemma (for SAT)). *Let $x : \mathcal{A}_\phi \longrightarrow (0, 1)$ be a function such that*

$$\mathbf{Pr}\left[A\right] \leq x(A) \prod_{B \in N(A)} (1 - x(B)), \forall A \in \mathcal{A}_\phi.$$

Then each C_i is resampled at most an expected $\frac{x(A_i)}{1 - x(A_i)}$ times in algorithm 1 before it returns a satisfying assignment of ϕ.

Thus, the expected total number of resampling steps is at most $\sum_{i=1}^{m} \frac{x(A_i)}{1-x(A_i)}$. This indicates algorithm 1 runs in expected polynomial time, that is, it is a Las Vegas algorithm.

In Moser and Tardos's original paper [1], algorithm 1 and theorem 4 are stated for general Constraint Satisfaction Problem (CSP). Here we only prove it for SAT for the sake of simplicity.

To present a proof of theorem 4 is a heavy work, and hence we break it into several parts.

2.2 Execution log and witness tree

Execution log. To analyze algorithm 1, we record which clause is resampled at each step. Formally, the *log* of execution is a function $C : \mathbb{N} \to \mathcal{A}_\phi$ where $\texttt{clause}(C(i))$ is resampled in step i. If the algorithm terminates after t iterations, then $C(i)$ is undefined for $i > t$.

Witness tree. For an arbitrary set S, an *S-labeled rooted tree* is a pair (T, σ), where T is a rooted tree with a labelling $\sigma : V(T) \to S$ of its vertices. A *witness tree* is a $\mathcal{A}_\phi$-labeled rooted tree $\tau = (T, \sigma)$ such that if v is child of u in T, then $\sigma(v) \in N^+(\sigma(u))$. For simplicity, write $[v]$ for $\sigma(v)$ and $V(\tau) := V(T)$. See fig. 1 for a simple example. Loosely speaking, $[v]$ is *the label of v*.

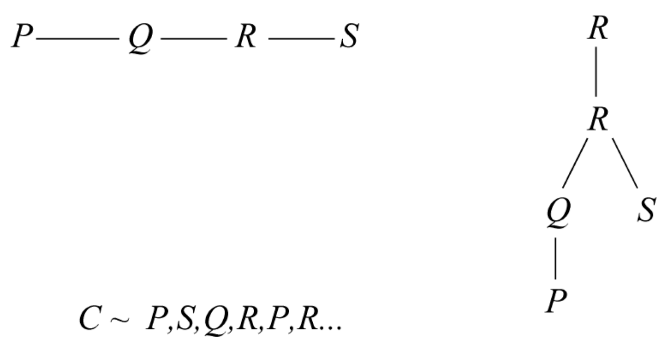

Figure 1: Simple dependency graph, a Possible Log C and the witness tree $\tau_C(6)$.

Given the log C, we now associate with each step $t \in \mathbb{N}$ a witness tree $\tau_C(t)$ constructed iteratively as follows.

1. At the beginning, $\tau_C(t)$ consists of a single vertex labelled $C(t)$.
2. For each *time* $i = t-1, t-2, \ldots, 1$:
   - if there is a vertex $v \in V(\tau_C(t))$ such that $C(i) \in N^+([v])$, choose such a v with maximum depth; if there are several such v's with the same depth, just choose one arbitrarily.
   - if there is no such a v, skip this iteration;
   - renew $\tau_C(t)$ by attaching a new child labeled $C(i)$ to v.

We say a witness tree τ *appears in* C if $\tau = \tau_C(t)$ for some $t \in \mathbb{N}$. A witness tree τ is *proper* if for every $v \in V(\tau)$, the children of v have different labels. The following observation obviously follows from the construction of $\tau_C(t)$.

Lemma 5. *For every witness tree τ, if τ appears in C, then τ is proper.*

2.3 Get an upper bound by coupling

Coupling

A coupling of two probability distributions μ and ν is a pair of random variables (X, Y) defined on a single probability space such that the marginal distribution of X is μ and the marginal distribution of Y is ν. That is, a coupling (X, Y) satisfies $\mathbf{Pr}[X = x] = \mu(x)$ and $\mathbf{Pr}[Y = y] = \nu(y)$.

The notion of coupling provides a way to compare distributions. We shall use this to obtain the following bound.

Lemma 6. *For any witness tree τ,*

$$\mathbf{Pr}[\tau \text{ appears in } C] \le \prod_{v \in V(\tau)} \mathbf{Pr}[[v]] . \tag{3}$$

Proof. Fix a witness tree τ. We consider a procedure called *τ-check*, as is shown in algorithm 2. It is easy to see the probability that τ-check returns Pass is exactly $\prod_{v \in V(\tau)} \mathbf{Pr}[[v]]$. We shall prove that

$$\mathbf{Pr}[\tau \text{ appears in } C] \le \mathbf{Pr}[\tau\text{-check returns Pass}],$$

which implies eq. (3) immediately.

Algorithm 2: τ-check

```
Input: a witness tree τ
Output: Pass or Fail
Let v_1, v_2, ..., v_s be the vertices in τ in an order of decreasing depth ;
foreach i ∈ [s] do
    assign random values for variables in clause([v]) ;
    if [v] does not happen(i.e, the random values are satisfying) then
        return Fail
return Pass
```

We will now couple the process of τ-check and the process of *independently resampling each clause in* τ. This is achieved by using the idea of resampling table.

Suppose that the algorithm uses the randomness $\mathcal{R} : [n] \times \mathbb{N} \to \{0, 1\}$. That is, when variable $x_i \in \mathcal{V}$ is resampled for the j-th time, the result of the coin toss is $\mathcal{R}(i, j)$. $\mathcal{R}$ is called the resampling table and it is assumed that the table is fixed before the coupling.

Assume that τ appears in C, say, $\tau = \tau_C(t_\star)$. We need to show that τ-check returns Pass (with randomness $\mathcal{R}$). Suppose that x_i is resampled when $v_j \in V(\tau)$ is visited by τ-check. Let $\mathtt{rec}(x_i, v_j)$ be the number of resampling for x_i before visiting v_j. Clearly,

$$\mathtt{rec}(x_i, v_j) = \{k \in [j-1] : x_i \in \mathtt{var}([v_k])\},$$

where $\mathtt{var}(A)$ is the set of variables in $\mathtt{clause}(A)$. Let $\mathtt{time}(v_j)$ be the time when v_j is added to $\tau_C(t_\star)$. We claim that $x_j = \mathcal{R}(i, |\mathtt{rec}(x_i, v_j)|)$ at step $\mathtt{time}(v_j)$ (before this resampling). Indeed, at time $t = 1, 2, \ldots, \mathtt{time}(v_j) - 1$, x_i is resampled at step t iff $t = \mathtt{time}(v_k)$ for some $k \in \mathtt{rec}(x_i, v_j)$. As the τ-check has these exact same values for the variables in $\mathtt{var}([v_j])$ when considering v_j, it finds that $\mathtt{clause}([v])$ is violated as well. This finishes the proof. □

We remark that our way of constructing witness tree is not crucial for eq. (3) to hold. The reason that we need such a construction is that witness trees encode the execution of the algorithm in a compact way. Therefore, a good upper bound to the number of distinct witness trees is a good upper bound for the length of the execution log (and hence the runtime of the algorithm). We will obtain such an upper bound in the next subsection.

2.4 Generating witness trees by Galton-Wltson Process

Fix an event $A_\star \in \mathcal{A}_\phi$ and consider the following *multitype Galton-Watson branching process* for generating a proper witness tree having its root labelled $A_\star$.

1. In the first round, we produce a singleton vertex labelled $A_\star$.
2. In ith round ($i \geq 2$), for each vertex v born in the $(i-1)$-th round and each $B \in N^+([v])$, attach a new child labelled B to v with probability $x(B)$.
3. All the choices involved are independent.

Let $x'(A) := x(A) \prod_{B \in N(A)} (1 - x(B))$.

Lemma 7. *Let τ be a fixed proper witness tree with its root vertex labeled $A_\star$. Then*

$$p_t := \mathbf{Pr}\left[\textit{the GW process yields exactly } \tau\right] = \frac{1 - x(A_\star)}{x(A_\star)} \prod_{v \in V(\tau)} x'([v]).$$

Proof. For each $v \in V(\tau)$, define

$$W_v := \{A \in N^+([v]) : \text{no child of } v \text{ is labeled } A\}.$$

Considering each vertex independently, we get

$$p_t = \frac{1}{x(A_\star)} \prod_{v \in V(\tau)} \left(x([v]) \prod_{A \in W_v} (1 - x(A)) \right),$$

where the term $\frac{1}{x(A_\star)}$ accounts for the fact the the root is always born. Next we replace W_v by $N^+([v])$:

$$p_\tau = \frac{1 - x(A_\star)}{x(A_\star)} \prod_{v \in V(\tau)} \left(\frac{x([v])}{1 - x([v])} \underline{\prod_{A \in N^+([v])} (1 - x(A))} \right). \tag{4}$$

Intuitively, in eq. (4), each node *assumes* that no child is born (see the underlined part), and each node contributes a term $\frac{x([v])}{1-x([v])}$, saying that 'oh, no, in fact I was born!'. Next, replacing $N^+([v])$ by $N([v])$ in eq. (4), we have

$$p_\tau = \frac{1 - x(A_\star)}{x(A_\star)} \prod_{v \in V(\tau)} \left(x([v]) \prod_{A \in N([v])} (1 - x(A)) \right) = \frac{1 - x(A_\star)}{x(A_\star)} \prod_{v \in V(\tau)} x'([v]),$$

completing the proof. □

Galton-Watson Process

A *Galton-Watson process* is a stochastic process (X_n) which evolves according to the recurrence formula $X_0 = 1$ and

$$X_{n+1} = \sum_{i=1}^{X_n} Z_i^{(n)},$$

where $(Z_i^{(n)} : i, n \in \mathbb{N})$ is a set of independent and I.I.D. $\mathbb{N}$-valued random variables. See the Chapter 0 of [2] for an intriguing discussion.

2.5 The coup de grace

Now everything is ready.

Proof of theorem 4. Let C be the log of execution. Let N_A be the random variable that counts how many times the `clause`(A) is resampled. Our goal is to bound $\mathbf{E}[N_A]$ from above.

Define

$$\mathcal{T}_A := \{\tau : \tau \text{ is a proper witness tree whose root is labelled } A\}.$$

The root of $\tau_C(t)$ is labelled A iff `clause`(A) is resampled at time t, and thus

$$N_A = \sum_{\tau \in \mathcal{T}_A} \mathbf{1}_{\{\tau \text{ appears in } C\}}.$$

Combining lemma 6, we have

$$\mathbf{E}[N_A] = \sum_{\tau \in \mathcal{T}_A} \mathbf{Pr}[\tau \text{ appears in } C] \le \sum_{\tau \in \mathcal{T}_A} \prod_{v \in V(\tau)} \mathbf{Pr}[[v]] \le \sum_{\tau \in \mathcal{T}_A} \prod_{v \in V(\tau)} x'([v]), \tag{5}$$

where the last inequality follows from the condition of the theorem 4.

Recall that lemma 7 says

$$\prod_{v \in V(\tau)} x'([v]) = \frac{x(A)}{1 - x(A)} \cdot p_\tau.$$

Plugging this into eq. (5) yields

$$\mathbf{E}[N_A] \le \frac{x(A)}{1 - x(A)} \sum_{\tau \in \mathcal{T}_A} p_\tau \le \frac{x(A)}{1 - x(A)}. \tag{6}$$

The last inequality of eq. (6) follows from the following simple fact:

$$\sum_{\tau \in \mathcal{T}_A} p_\tau \le \sum_{\tau \text{ is a possible result of GW process}} p_\tau = 1.$$

We are happy to see that eq. (6) is exactly what we set out to prove. ◻

References

[1] Robin A Moser and Gábor Tardos, *A constructive proof of the general lovász local lemma*, Journal of the ACM (JACM), 57 (2010), pp. 1–15. 5

[2] D. Williams, *Probability with Martingales*, Cambridge mathematical textbooks, Cambridge University Press, 1991. 8