The Algorithmic Lovasz Local Lemma

Ajay Sandhu

COMP5112

November 13, 2024

Outline

Lovasz Local Lemma (LLL)

- Discovered by Erdos and Lovasz in 1973.
- Another tool from the probabilistic method (recall COMP2804).
- Proves existence of combinatorial objects under certain constraints.

Motivation for the Lovasz Local Lemma

- We are trying to avoid certain events $\mathcal{S} = \{E_1, E_2, \ldots, E_k\}$

Motivation for the Lovasz Local Lemma

- We are trying to avoid certain events $\mathcal{S} = \{E_1, E_2, \ldots, E_k\}$

- What if each event $A \in \mathcal{S}$ is independent? Easy.

Motivation for the Lovasz Local Lemma

- We are trying to avoid certain events $\mathcal{S} = \{E_1, E_2, \ldots, E_k\}$

- What if each event $A \in \mathcal{S}$ is independent? Easy.

$$P(\text{No } A \in \mathcal{S} \text{ occurs}) = \prod_{i=1}^{k} (1 - P(E_i))$$

- Positive as long as $P(A) < 1$ for each $A \in \mathcal{S}$.

Motivation for the Lovasz Local Lemma

- We are trying to avoid certain events $\mathcal{S} = \{E_1, E_2, \ldots, E_k\}$

- What if each event $A \in \mathcal{S}$ is independent? Easy.

$$P(\text{No } A \in \mathcal{S} \text{ occurs}) = \prod_{i=1}^{k} (1 - P(E_i))$$

- Positive as long as $P(A) < 1$ for each $A \in \mathcal{S}$.

- What if the events are not independent? Union Bound?

Motivation for the Lovasz Local Lemma

- We are trying to avoid certain events $\mathcal{S} = \{E_1, E_2, \ldots, E_k\}$

- What if each event $A \in \mathcal{S}$ is independent? Easy.

$$P(\text{No } A \in \mathcal{S} \text{ occurs}) = \prod_{i=1}^{k} (1 - P(E_i))$$

- Positive as long as $P(A) < 1$ for each $A \in \mathcal{S}$.

- What if the events are not independent? Union Bound?

$$P(\text{No } A \in \mathcal{S} \text{ occurs}) \geq 1 - \sum_{i=1}^{k} P(E_i)$$

- Positive only if the probability of the events in $\mathcal{S}$ are small.

Stating the Lovasz Local Lemma

- The LLL allows us to show that it is possible for none of the events in A to occur when the events have limited dependence.

Stating the Lovasz Local Lemma

- The LLL allows us to show that it is possible for none of the events in A to occur when the events have limited dependence.
- A very general form is the following.

Theorem (Lovász Local Lemma)

Let $\mathcal{S}$ be a finite set of events in a probability space. For $A \in \mathcal{S}$, let $\Gamma(A)$ be a subset of $\mathcal{S}$ satisfying that A is independent from the collection of events $\mathcal{S} \setminus (\{A\} \cup \Gamma(A))$.
If there exists an assignment of reals $x : \mathcal{S} \to (0, 1)$ such that

$$\forall A \in \mathcal{S} : \Pr[A] \leq x(A) \prod_{B \in \Gamma(A)} (1 - x(B)),$$

then the probability of avoiding all events in $\mathcal{S}$ is at least $\prod_{A \in \mathcal{S}} (1 - x(A))$ and hence positive.

Stating the LLL

- A simple form of the lemma is the following.

Theorem (Symmetric Form Lovasz Local Lemma)

Let $\mathcal{S}$ be a finite set of events in a probability space. Let d be an integer such that each event $A \in \mathcal{S}$ is independent from all other events in A except for at most d of them. Let p be a real such that $\Pr(A) \leq p$ for all $A \in \mathcal{S}$. Then, if

$$ep(d+1) \leq 1$$

with positive probability none of the events in $\mathcal{S}$ happen.

An Application to Satisfiability

- The K-SAT problem

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee x_4)$$

An Application to Satisfiability

- The K-SAT problem

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee x_4)$$

  - A CNF boolean formula

An Application to Satisfiability

- The K-SAT problem

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee x_4)$$

  - A CNF boolean formula
  - k variables per clause

An Application to Satisfiability

- The K-SAT problem

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee x_4)$$

  - A CNF boolean formula
  - k variables per clause
  - NP-Complete problem in general

A Lower Bound On Number of Clauses

- **Warm-up:** What is minimum number of clauses needed to build an unsatisfiable K-CNF formula?

A Lower Bound On Number of Clauses

- **Warm-up:** What is minimum number of clauses needed to build an unsatisfiable K-CNF formula?
  - Pr(clause being false) $= 1/2^k$

A Lower Bound On Number of Clauses

- **Warm-up:** What is minimum number of clauses needed to build an unsatisfiable K-CNF formula?
  - $\Pr(\text{clause being false}) = 1/2^k$
  - Expected number of false clauses is $m \cdot \frac{1}{2^k}$.

A Lower Bound On Number of Clauses

- **Warm-up:** What is minimum number of clauses needed to build an unsatisfiable K-CNF formula?
  - Pr(clause being false) $= 1/2^k$
  - Expected number of false clauses is $m \cdot \frac{1}{2^k}$.
  - This is less than 1 if $m < 2^k$.

A Lower Bound On Number of Clauses

- **Warm-up:** What is minimum number of clauses needed to build an unsatisfiable K-CNF formula?
  - $\Pr(\text{clause being false}) = 1/2^k$
  - Expected number of false clauses is $m \cdot \frac{1}{2^k}$.
  - This is less than 1 if $m < 2^k$.
- We need at least 2^k clauses.

Applying the LLL

- K-SAT becomes more difficult as there is more "overlap" between the clauses.

Applying the LLL

- K-SAT becomes more difficult as there is more “overlap" between the clauses.

- If there is not too much overlap in a CNF formula, it will always be satisfiable (i.e. there exists a satisfying assignment).

Applying the LLL

- K-SAT becomes more difficult as there is more "overlap" between the clauses.

- If there is not too much overlap in a CNF formula, it will always be satisfiable (i.e. there exists a satisfying assignment).

- Let us show that if every clause shares a variable with $\leq \frac{2^k}{e} - 1$ other clauses, the formula is satisfiable.

Making it Algorithmic

- The original proof for the LLL is *non-constructive*.

Making it Algorithmic

- The original proof for the LLL is *non-constructive*.

- *Constructive* proof provided by Moser and Tardos in 2010.
  - Won them the Godel Prize!

Making it Algorithmic

- The original proof for the LLL is *non-constructive*.

- *Constructive* proof provided by Moser and Tardos in 2010.
  - Won them the Godel Prize!

- They provide simple and efficient *algorithm*
  - Efficiently *find* desired configuration rather than just show it *exists*.

Making it Algorithmic

- The original proof for the LLL is *non-constructive*.
- *Constructive* proof provided by Moser and Tardos in 2010.
  - Won them the Godel Prize!
- They provide simple and efficient *algorithm*
  - Efficiently *find* desired configuration rather than just show it *exists*.
- Closed the gap between previous constructive results and the existential version.

Making it Algorithmic

- The original proof for the LLL is *non-constructive*.
- *Constructive* proof provided by Moser and Tardos in 2010.
  - Won them the Godel Prize!
- They provide simple and efficient *algorithm*
  - Efficiently *find* desired configuration rather than just show it *exists*.
- Closed the gap between previous constructive results and the existential version.
- K-SAT: we could actually find the assignment of variables.

Algorithmic LLL Setup

- Let $\mathcal{P} = \{P_1, P_2, P_3, \ldots\}$ be a set of mutually independent random variables in the probability space Ω.

Algorithmic LLL Setup

- Let $\mathcal{P} = \{P_1, P_2, P_3, \dots\}$ be a set of mutually independent random variables in the probability space Ω.

- Let $\mathcal{S} = \{E_1, E_2, \dots\}$ be the set of events we are trying to avoid.
  - Each $A \in \mathcal{S}$ is determined by a subset of the random variables in $\mathcal{P}$.

Algorithmic LLL Setup

- Let $\mathcal{P} = \{P_1, P_2, P_3, \ldots\}$ be a set of mutually independent random variables in the probability space Ω.

- Let $\mathcal{S} = \{E_1, E_2, \ldots\}$ be the set of events we are trying to avoid.
  - Each $A \in \mathcal{S}$ is determined by a subset of the random variables in $\mathcal{P}$.

- Let $\mathrm{vbl}(A) \subseteq \mathcal{P}$ be the unique minimal subset of random variables which determines an event $A \in \mathcal{S}$.

Algorithmic LLL Setup

- Let $\mathcal{P} = \{P_1, P_2, P_3, \ldots\}$ be a set of mutually independent random variables in the probability space Ω.

- Let $\mathcal{S} = \{E_1, E_2, \ldots\}$ be the set of events we are trying to avoid.
  - Each $A \in \mathcal{S}$ is determined by a subset of the random variables in $\mathcal{P}$.

- Let $\mathrm{vbl}(A) \subseteq \mathcal{P}$ be the unique minimal subset of random variables which determines an event $A \in \mathcal{S}$.

- If an event $A \in \mathcal{S}$ happens from an evaluation of the random variables $\mathcal{P}$, we say that A was *violated*.

Algorithmic LLL Setup

- Let G be the *dependency graph* for event set $\mathcal{S}$.
  - *Vertices*: The vertex set is $\mathcal{S}$
  - *Edges*: Edge between $A, B \in \mathcal{S}$ if $A \neq B$ and $\mathrm{vbl}(A) \cap \mathrm{vbl}(B) \neq \emptyset$

Algorithmic LLL Setup

- Let G be the *dependency graph* for event set $\mathcal{S}$.
  - *Vertices*: The vertex set is $\mathcal{S}$
  - *Edges*: Edge between $A, B \in \mathcal{S}$ if $A \neq B$ and $\text{vbl}(A) \cap \text{vbl}(B) \neq \emptyset$

- Let $\Gamma(A)$ be the *neighborhood* of $A \in \mathcal{S}$ in the dependency graph G.

Algorithmic LLL Setup

- Let G be the *dependency graph* for event set $\mathcal{S}$.
  - *Vertices*: The vertex set is $\mathcal{S}$
  - *Edges*: Edge between $A, B \in \mathcal{S}$ if $A \neq B$ and $\mathrm{vbl}(A) \cap \mathrm{vbl}(B) \neq \emptyset$

- Let $\Gamma(A)$ be the *neighborhood* of $A \in \mathcal{S}$ in the dependency graph G.

- Let $\Gamma^+(A) \coloneqq \{A\} \cup \Gamma(A)$ be the *inclusive neighborhood* of $A \in \mathcal{S}$.

The Algorithm

- The algorithm is incredibly simple...
  Just keep resampling violated events until we get what we want!

The Algorithm

- The algorithm is incredibly simple...
  Just keep resampling violated events until we get what we want!

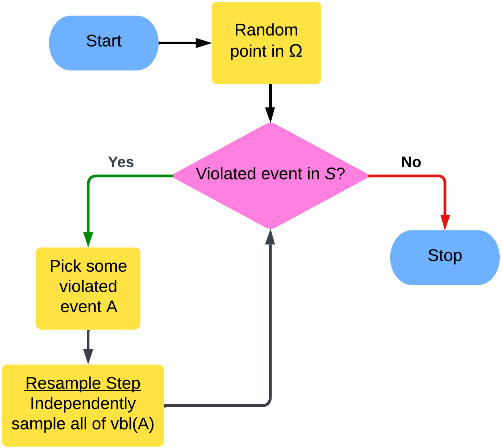

Figure 1: Algorithmic Lovasz Local Lemma Flowchart

The Algorithmic LLL Theorem

Theorem (Algorithmic Lovasc Local Lemma)

If there exists an assignment of reals $x : \mathcal{S} \to (0,1)$ *such that*

$$\forall A \in \mathcal{S} : \Pr[A] \leq x(A) \prod_{B \in \Gamma(A)} (1 - x(B)),$$

then there exists an evaluation of the variables in $\mathcal{P}$ *that does not violate any of the events in* $\mathcal{S}$.
This assignment can be found with the aforementioned randomized algorithm which resamples an event $A \in \mathcal{S}$ *at most an expected* $\frac{x(A)}{1-x(A)}$ *times.*

The Total Number of Resamples

- Let N_A be the times event $A \in \mathcal{S}$ resampled.

The Total Number of Resamples

- Let N_A be the times event $A \in \mathcal{S}$ resampled.
- Let N be the total number of resamples

$$N = \sum_{A \in \mathcal{S}} N_A$$

$$E(N) = E\left(\sum_{A \in \mathcal{S}} N_A\right)$$

$$E(N) = \sum_{A \in \mathcal{S}} E(N_A)$$

$$E(N) = \sum_{A \in \mathcal{S}} \frac{x(A)}{1 - x(A)}$$

The Total Number of Resamples

- Let N_A be the times event $A \in \mathcal{S}$ resampled.
- Let N be the total number of resamples

$$N = \sum_{A \in \mathcal{S}} N_A$$

$$E(N) = E\left(\sum_{A \in \mathcal{S}} N_A\right)$$

$$E(N) = \sum_{A \in \mathcal{S}} E(N_A)$$

$$E(N) = \sum_{A \in \mathcal{S}} \frac{x(A)}{1 - x(A)}$$

Corollary

The expected total number of resamples the algorithm performs it at most $\sum_{A \in \mathcal{S}} \frac{x(A)}{1-x(A)}$.

The Execution Log

- Let $L : \mathbb{N} \to \mathcal{S}$ be the *execution log* of the algorithm.

The Execution Log

- Let $L : \mathbb{N} \to \mathcal{S}$ be the *execution log* of the algorithm.
- Each entry corresponds to an event resampled at a certain time.

Resample Step	**Event**
1	A
2	B
3	C
4	D
5	A
6	C
7	B
8	D

Table 1: Example of an Execution Log

Witness Trees

- A *witness tree* $\tau = (T, \sigma_T)$
  - Finite rooted tree T

Witness Trees

- A *witness tree* $\tau = (T, \sigma_T)$
    - Finite rooted tree T
    - Labeling of its vertices $\sigma_T : V(T) \to \mathcal{S}$

Witness Trees

- A *witness tree* $\tau = (T, \sigma_T)$
  - Finite rooted tree T
  - Labeling of its vertices $\sigma_T : V(T) \to \mathcal{S}$
  - For any vertex $u \in V(T)$, its children can only come from $\Gamma^+(\sigma_T(u))$

Witness Trees

- A *witness tree* $\tau = (T, \sigma_T)$
  - Finite rooted tree T
  - Labeling of its vertices $\sigma_T : V(T) \to \mathcal{S}$
  - For any vertex $u \in V(T)$, its children can only come from $\Gamma^+(\sigma_T(u))$
  - For simplicity: $V(\tau) \coloneqq V(T)$, $[v] \coloneqq \sigma_T(v)$

Witness Trees

- A *witness tree* $\tau = (T, \sigma_T)$
  - Finite rooted tree T
  - Labeling of its vertices $\sigma_T : V(T) \to \mathcal{S}$
  - For any vertex $u \in V(T)$, its children can only come from $\Gamma^+(\sigma_T(u))$
  - For simplicity: $V(\tau) \coloneqq V(T)$, $[v] \coloneqq \sigma_T(v)$

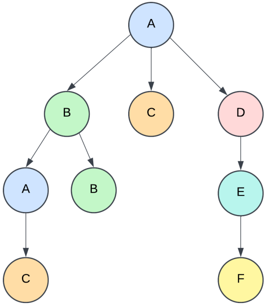

Figure 2: Example of Witness Tree with $\mathcal{S} = \{A, B, C, D, E, F\}$

Constructing Witness Trees

- Each entry in the log has a corresponding witness tree.
  - For step t in the log L (i.e. $L(t)$), the witness tree is $\tau_L(t)$.

Constructing Witness Trees

- Each entry in the log has a corresponding witness tree.
  - For step t in the log L (i.e. $L(t)$), the witness tree is $\tau_L(t)$.

- The witness tree $\tau_L(t)$ is constructed in following way.
  1. Make event $L(t)$ the root.
  2. Go backwards in L from step t (let WT at step i be $\tau_L^{(i)}(t)$)
     - If $\exists v \in \tau_L^{(i+1)}(t)$ such that $L(i) \in \Gamma^+([v])$, add $L(i)$ as child to the deepest such vertex.
     - Else skip this iteration and set $\tau_L^{(i)}(t) = \tau_L^{(i+1)}(t)$.
  3. Stop when we reach beginning of the log.

Constructing Witness Trees

- Each entry in the log has a corresponding witness tree.
  - For step t in the log L (i.e. $L(t)$), the witness tree is $\tau_L(t)$.

- The witness tree $\tau_L(t)$ is constructed in following way.
  1. Make event $L(t)$ the root.
  2. Go backwards in L from step t (let WT at step i be $\tau_L^{(i)}(t)$)
     - If $\exists v \in \tau_L^{(i+1)}(t)$ such that $L(i) \in \Gamma^+([v])$, add $L(i)$ as child to the deepest such vertex.
     - Else skip this iteration and set $\tau_L^{(i)}(t) = \tau_L^{(i+1)}(t)$.
  3. Stop when we reach beginning of the log.

- Each WT $\tau_L(t)$ is justification for having to resample at step t.

Witness Tree Definitions

- A WT τ *occurs* in an execution log L if $\tau_L(t) = \tau$ for some $t \in \mathbb{N}$.

Witness Tree Definitions

- A WT τ *occurs* in an execution log L if $\tau_L(t) = \tau$ for some $t \in \mathbb{N}$.

- A witness tree τ is *proper* if for any vertex $v \in V(\tau)$, its children all have distinct labels.

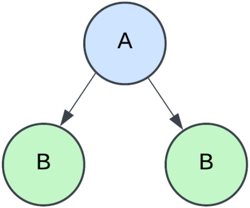

Figure 3: Example of **not** proper witness tree

First Claim about Witness Trees

Lemma (Witness Trees are Proper)

Let τ be a fixed witness tree and L be the (random) execution log produced by the algorithm.

$$\text{If } \tau \text{ occurs in the log } L \implies \tau \text{ is proper}$$

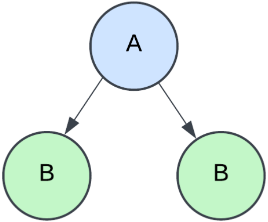

Figure 4: Example of **not** proper witness tree

Slightly Stronger Claims about Witness Trees

- Not only do siblings vertices in a witness tree (associated to a random log) always have distinct labels, but...

Slightly Stronger Claims about Witness Trees

- Not only do siblings vertices in a witness tree (associated to a random log) always have distinct labels, but...

Lemma (Witness Tree Level Uniqueness)

Let τ be a witness tree occurring in a random log.

All vertex labels for τ at equal depth are distinct.

Slightly Stronger Claims about Witness Trees

- Not only do siblings vertices in a witness tree (associated to a random log) always have distinct labels, but...

Lemma (Witness Tree Level Uniqueness)

Let τ be a witness tree occurring in a random log.

All vertex labels for τ at equal depth are distinct.

- Furthermore...

Lemma (Witness Trees Level Independence)

Let τ be a witness tree occurring in a random log.

$$\textit{For any } u, v \in V(\tau) \textit{ at equal depth, } \mathrm{vbl}([u]) \cap \mathrm{vbl}([v]) = \emptyset$$

Bounding Probability Of Occurrence

- Can we say how likely it is for a witness tree to occur in the log?

Bounding Probability Of Occurrence

- Can we say how likely it is for a witness tree to occur in the log?

Lemma (Probability of Witness Tree Occurence)

Let τ be a fixed witness tree and L be the (random) execution log produced by the algorithm.

$$\Pr(\tau \text{ occurs in } L) \leq \prod_{v \in V(\tau)} \Pr([v])$$

- The proof uses probabilistic *coupling* technique.
- We will couple two algorithms:
  - The constructive LLL algorithm
  - A procedure called τ-check.

The τ-check Procedure

- τ-check:
  1. For each vertex v in decreasing depth (i.e. reverse BFS order)
  2. Take random evaluation of $\text{vbl}([v])$
  3. Check if $[v]$ is *violated*

- The τ-check *passes* if $[v]$ is violated for every $v \in V(\tau)$.

- For a fixed witness tree τ:

$$\Pr(\tau\text{-check passes}) = \prod_{v \in V(\tau)} \Pr([v])$$

The Coupling Argument

- **The coupling argument**: A τ-check performed on a witness tree occurring in the log using the same *random source* as the constructive LLL algorithm will pass.

$$(\tau \text{ occurs in } L \implies \tau\text{-check passes})$$

The Coupling Argument

- **The coupling argument**: A τ-check performed on a witness tree occurring in the log using the same *random source* as the constructive LLL algorithm will pass.

$$
\begin{aligned}
& (\tau \text{ occurs in } L \implies \tau\text{-check passes}) \\
\implies \quad & \Pr(\tau \text{ occurs in } L) \leq \Pr(\tau\text{-check passes})
\end{aligned}
$$

The Coupling Argument

- **The coupling argument**: A τ-check performed on a witness tree occurring in the log using the same *random source* as the constructive LLL algorithm will pass.

$$
\begin{aligned}
& (\tau \text{ occurs in } L \implies \tau\text{-check passes}) \\
\implies \quad & \Pr(\tau \text{ occurs in } L) \leq \Pr(\tau\text{-check passes}) \\
\implies \quad & \Pr(\tau \text{ occurs in } L) \leq \prod_{v \in V(\tau)} \Pr\left([v]\right)
\end{aligned}
$$

Using the Same Random Source

- Imagine that each random variable $P \in \mathcal{P}$, produces an infinite stream of random samples $P^{(0)}, P^{(1)}, P^{(2)}, \ldots$

Using the Same Random Source

- Imagine that each random variable $P \in \mathcal{P}$, produces an infinite stream of random samples $P^{(0)}, P^{(1)}, P^{(2)}, \ldots$

- Each need for a new random sample, we take next unused value.
  - $P^{(i)}$ if we have already taken i samples from P.

How the τ-check uses Random Source

- Let τ be an arbitrary witness tree from our log L.
  - i.e. $\tau = \tau_L(t)$ for some $t \in \mathbb{N}$

How the τ-check uses Random Source

- Let τ be an arbitrary witness tree from our log L.
  - i.e. $\tau = \tau_L(t)$ for some $t \in \mathbb{N}$
- Let v be an arbitrary vertex in τ.

How the τ-check uses Random Source

- Let τ be an arbitrary witness tree from our log L.
  - i.e. $\tau = \tau_L(t)$ for some $t \in \mathbb{N}$

- Let v be an arbritrary vertex in τ.

- **Question:** In a τ-check, what values from the random source do we pull for $[v]$?
  - i.e. What value of $P^{(i)}$ for each $P \in \text{vbl}([v])$?

Defining the Set $Q(P)$

- Let $Q(P)$, for $v \in V(\tau)$ and $P \in \mathrm{vbl}([v])$, be the set of vertices $w \in V(\tau)$ that have a greater depth than v and $P \in \mathrm{vbl}([w])$.

Defining the Set $Q(P)$

- Let $Q(P)$, for $v \in V(\tau)$ and $P \in \text{vbl}([v])$, be the set of vertices $w \in V(\tau)$ that have a greater depth than v and $P \in \text{vbl}([w])$.

- Consider a $P \in \text{vbl}([v])$.

Defining the Set $Q(P)$

- Let $Q(P)$, for $v \in V(\tau)$ and $P \in \text{vbl}([v])$, be the set of vertices $w \in V(\tau)$ that have a greater depth than v and $P \in \text{vbl}([w])$.

- Consider a $P \in \text{vbl}([v])$.

- What happens when we visit a vertex in $Q(P)$ during a τ-check?

Defining the Set $Q(P)$

- Let $Q(P)$, for $v \in V(\tau)$ and $P \in \text{vbl}([v])$, be the set of vertices $w \in V(\tau)$ that have a greater depth than v and $P \in \text{vbl}([w])$.
- Consider a $P \in \text{vbl}([v])$.
- What happens when we visit a vertex in $Q(P)$ during a τ-check?
  - We will sample from P.

Defining the Set $Q(P)$

- Let $Q(P)$, for $v \in V(\tau)$ and $P \in \text{vbl}([v])$, be the set of vertices $w \in V(\tau)$ that have a greater depth than v and $P \in \text{vbl}([w])$.
- Consider a $P \in \text{vbl}([v])$.
- What happens when we visit a vertex in $Q(P)$ during a τ-check?
  - We will sample from P.
- Will we sample from P with any vertices w deeper than v, but where $P \notin \text{vbl}([w])$?

Defining the Set $Q(P)$

- Let $Q(P)$, for $v \in V(\tau)$ and $P \in \text{vbl}([v])$, be the set of vertices $w \in V(\tau)$ that have a greater depth than v and $P \in \text{vbl}([w])$.

- Consider a $P \in \text{vbl}([v])$.

- What happens when we visit a vertex in $Q(P)$ during a τ-check?
  - We will sample from P.

- Will we sample from P with any vertices w deeper than v, but where $P \notin \text{vbl}([w])$?
  - NO.

- Will we sample from P with any vertices w at same level as v?

Defining the Set $Q(P)$

- Let $Q(P)$, for $v \in V(\tau)$ and $P \in \text{vbl}([v])$, be the set of vertices $w \in V(\tau)$ that have a greater depth than v and $P \in \text{vbl}([w])$.

- Consider a $P \in \text{vbl}([v])$.

- What happens when we visit a vertex in $Q(P)$ during a τ-check?
  - We will sample from P.

- Will we sample from P with any vertices w deeper than v, but where $P \notin \text{vbl}([w])$?
  - NO.

- Will we sample from P with any vertices w at same level as v?
  - NO.

The value of P during a τ-check

- The only times we will sample from P during the τ-check is when we visit vertices in $Q(P)$.

The value of P during a τ-check

- The only times we will sample from P during the τ-check is when we visit vertices in $Q(P)$.

- When the variables of v are sampled for the τ-check, for $P \in \text{vbl}([v])$ we will sample:

$$P^{(|Q(P)|)}$$

The value of P during a τ-check

- The only times we will sample from P during the τ-check is when we visit vertices in $Q(P)$.

- When the variables of v are sampled for the τ-check, for $P \in \mathrm{vbl}([v])$ we will sample:

$$P^{(|Q(P)|)}$$

The value of P during the LLL algorithm

- Let us now ask a similar question, but for the constructive LLL algorithm rather than the τ-check.

The value of P during the LLL algorithm

- Let us now ask a similar question, but for the constructive LLL algorithm rather than the τ-check.

- **Question:** What is the value of P for each $P \in \text{vbl}([v])$ before the LLL algorithm resamples $[v]$?

The value of P during the LLL algorithm

- Let us now ask a similar question, but for the constructive LLL algorithm rather than the τ-check.

- **Question:** What is the value of P for each $P \in \text{vbl}([v])$ before the LLL algorithm resamples $[v]$?

- We resample P at beginning of the LLL algorithm.

The value of P during the LLL algorithm

- Let us now ask a similar question, but for the constructive LLL algorithm rather than the τ-check.

- **Question:** What is the value of P for each $P \in \mathrm{vbl}([v])$ before the LLL algorithm resamples $[v]$?

- We resample P at beginning of the LLL algorithm.

- Then we resample it for each $[w]$ for $w \in Q(P)$

The value of P during the LLL algorithm

- Let us now ask a similar question, but for the constructive LLL algorithm rather than the τ-check.

- **Question:** What is the value of P for each $P \in \text{vbl}([v])$ before the LLL algorithm resamples $[v]$?

- We resample P at beginning of the LLL algorithm.

- Then we resample it for each $[w]$ for $w \in Q(P)$
  - Since each $[w]$ shows up earlier in the log.

The value of P during the LLL algorithm

- Let us now ask a similar question, but for the constructive LLL algorithm rather than the τ-check.

- **Question:** What is the value of P for each $P \in \text{vbl}([v])$ before the LLL algorithm resamples $[v]$?

- We resample P at beginning of the LLL algorithm.

- Then we resample it for each $[w]$ for $w \in Q(P)$
  - Since each $[w]$ shows up earlier in the log.

- Thus, before we resample $[v]$, each of its $P \in \text{vbl}([v])$ holds value

$$P^{(|Q(P)|)}$$

.

Something Cool

- Each $P \in \mathrm{vbl}([v])$ has value of $P^{(|Q(P)|)}$:

Something Cool

- Each $P \in \text{vbl}([v])$ has value of $P^{(|Q(P)|)}$:
  - Before $[v]$ is resampled in the LLL algorithm.

Something Cool

- Each $P \in \text{vbl}([v])$ has value of $P^{(|Q(P)|)}$:
  - Before $[v]$ is resampled in the LLL algorithm.
  - When v has a τ-check performed.

Something Cool

- Each $P \in \text{vbl}([v])$ has value of $P^{(|Q(P)|)}$:
  - Before $[v]$ is resampled in the LLL algorithm.
  - When v has a τ-check performed.

- Recall why we have to resample $[v]$...

Something Cool

- Each $P \in \mathrm{vbl}([v])$ has value of $P^{(|Q(P)|)}$:
  - Before $[v]$ is resampled in the LLL algorithm.
  - When v has a τ-check performed.
- Recall why we have to resample $[v]$...
- $[v]$ was resampled because it was *violated*!

Something Cool

- Each $P \in \text{vbl}([v])$ has value of $P^{(|Q(P)|)}$:
  - Before $[v]$ is resampled in the LLL algorithm.
  - When v has a τ-check performed.

- Recall why we have to resample $[v]$...

- $[v]$ was resampled because it was *violated*!
  - i.e. If each $P \in \text{vbl}([v])$ has value $P^{(|Q(P)|)} \implies [v]$ violated.

Something Cool

- Each $P \in \text{vbl}([v])$ has value of $P^{(|Q(P)|)}$:
  - Before $[v]$ is resampled in the LLL algorithm.
  - When v has a τ-check performed.

- Recall why we have to resample $[v]$...

- $[v]$ was resampled because it was *violated*!
  - i.e. If each $P \in \text{vbl}([v])$ has value $P^{(|Q(P)|)} \implies [v]$ violated.

- τ-check will also find that $[v]$ is *violated*!

Something Cool

- Each $P \in \mathrm{vbl}([v])$ has value of $P^{(|Q(P)|)}$:
  - Before $[v]$ is resampled in the LLL algorithm.
  - When v has a τ-check performed.
- Recall why we have to resample $[v]$...
- $[v]$ was resampled because it was *violated*!
  - i.e. If each $P \in \mathrm{vbl}([v])$ has value $P^{(|Q(P)|)} \implies [v]$ violated.
- τ-check will also find that $[v]$ is *violated*!
- The τ-check on τ *passes*!

A simple process is defined which generates proper witness trees:

The Galton-Watson Process

A simple process is defined which generates proper witness trees:

1. Fix an event $A \in \mathcal{S}$ which becomes the root.
2. For each vertex v produced in the previous round:
   - For each $B \in \Gamma^+([v])$:
     - Add a vertex labeled B as a child of v with probability $x(B)$.
     - Otherwise (with probability $1 - x(B)$), skip the event B.
3. Continue process until it goes extinct

Known as a *multitype Galton-Watson branching process*.

Probability of Generating a Witness Tree

- Let $x'(B)$ be defined as the following

$$x'(B) := x(B) \prod_{C \in \Gamma(B)} (1 - x(C))$$

.

Probability of Generating a Witness Tree

- Let $x'(B)$ be defined as the following

$$x'(B) \coloneqq x(B) \prod_{C \in \Gamma(B)} (1 - x(C))$$

.

Lemma (Probability of Generation)

For a fixed proper witness τ with its root labeled A, the probability p_τ that the Galton-Watson process above generates exactly τ is

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

Probability of Generating a Witness Tree Cont.

- Let $D_v \subseteq \Gamma^+([v])$, for $v \in V(\tau)$, be the set of inclusive neighbors of $[v]$ that do *not* appear as a label of some child node of v.

Probability of Generating a Witness Tree Cont.

- Let $D_v \subseteq \Gamma^+([v])$, for $v \in V(\tau)$, be the set of inclusive neighbors of $[v]$ that do *not* appear as a label of some child node of v.

- i.e. the children of v can be represented as $\Gamma^+([v]) \setminus D_v$.

Probability of Generating a Witness Tree Cont.

- Let $D_v \subseteq \Gamma^+([v])$, for $v \in V(\tau)$, be the set of inclusive neighbors of $[v]$ that do *not* appear as a label of some child node of v.
- i.e. the children of v can be represented as $\Gamma^+([v]) \setminus D_v$.
- We can now derive an expression for the probability that the Galton-Watson expression generates exactly τ.

Probability of Generating a Witness Tree Cont.

$$p_\tau \quad = \frac{1}{x(A)} \prod_{v \in V(\tau)} \left(x([v]) \prod_{u \in D_v} (1 - x([u])) \right)$$

Probability of Generating a Witness Tree Cont.

$$
\begin{aligned}
p_\tau &= \frac{1}{x(A)} \prod_{v \in V(\tau)} \left(x([v]) \prod_{u \in D_v} (1 - x([u])) \right) \\
&= \frac{1 - x(A)}{x(A)} \prod_{v \in V(\tau)} \left(\frac{x([v])}{1 - x([v])} \prod_{u \in \Gamma^+([v])} (1 - x([u])) \right)
\end{aligned}
$$

Probability of Generating a Witness Tree Cont.

$$\begin{aligned} p_\tau &= \frac{1}{x(A)} \prod_{v \in V(\tau)} \left(x([v]) \prod_{u \in D_v} (1 - x([u])) \right) \\ &= \frac{1 - x(A)}{x(A)} \prod_{v \in V(\tau)} \left(\frac{x([v])}{1 - x([v])} \prod_{u \in \Gamma^+([v])} (1 - x([u])) \right) \\ &= \frac{1 - x(A)}{x(A)} \prod_{v \in V(\tau)} \left(x([v]) \prod_{u \in \Gamma([v])} (1 - x([u])) \right) \end{aligned}$$

Probability of Generating a Witness Tree Cont.

$$
\begin{aligned}
p_\tau &= \frac{1}{x(A)} \prod_{v \in V(\tau)} \left(x([v]) \prod_{u \in D_v} (1 - x([u])) \right) \\
&= \frac{1 - x(A)}{x(A)} \prod_{v \in V(\tau)} \left(\frac{x([v])}{1 - x([v])} \prod_{u \in \Gamma^+([v])} (1 - x([u])) \right) \\
&= \frac{1 - x(A)}{x(A)} \prod_{v \in V(\tau)} \left(x([v]) \prod_{u \in \Gamma([v])} (1 - x([u])) \right) \\
&= \frac{1 - x(A)}{x(A)} \prod_{v \in V(\tau)} x'([v])
\end{aligned}
$$

Equivalent Definition of N_A

- Recall N_A is number of times event A is resampled.

Equivalent Definition of N_A

- Recall N_A is number of times event A is resampled.

- **Observation:** Every witness tree that occurs in a log is distinct.

Equivalent Definition of N_A

- Recall N_A is number of times event A is resampled.
- **Observation:** Every witness tree that occurs in a log is distinct.
- Thus, equivalently...
- N_A is the number of distinct witness trees with a root labeled A which occur in the execution log.

Equivalent Definition of N_A Cont.

- Let $\mathcal{T}_A$ be the set of all proper witness trees with a root labeled A.

Equivalent Definition of N_A Cont.

- Let $\mathcal{T}_A$ be the set of all proper witness trees with a root labeled A.

- Let indicator random variable W_τ be

$$W_\tau = \begin{cases} 1 & \text{if } \tau \text{ occurs in } L \\ 0 & \text{otherwise} \end{cases}$$

Equivalent Definition of N_A Cont.

- Let $\mathcal{T}_A$ be the set of all proper witness trees with a root labeled A.

- Let indicator random variable W_τ be

$$W_\tau = \begin{cases} 1 & \text{if } \tau \text{ occurs in } L \\ 0 & \text{otherwise} \end{cases}$$

- Thus, we can represent N_A as

$$N_A = \sum_{\tau \in \mathcal{T}_A} W_\tau$$

Putting It All Together

$$E(N_A) \quad = E\left(\sum_{\tau \in \mathcal{T}_A} W_\tau\right)$$

Putting It All Together

$$
\begin{aligned}
E(N_A) &= E\left(\sum_{\tau \in \mathcal{T}_A} W_\tau\right) \\
&= \sum_{\tau \in \mathcal{T}_A} E(W_\tau)
\end{aligned}
$$

Putting It All Together

$$\begin{aligned} E(N_A) &= E\left(\sum_{\tau \in \mathcal{T}_A} W_\tau\right) \\ &= \sum_{\tau \in \mathcal{T}_A} E(W_\tau) \\ &= \sum_{\tau \in \mathcal{T}_A} \Pr(W_\tau) \end{aligned}$$

Putting It All Together

$$
\begin{aligned}
E(N_A) &= E\left(\sum_{\tau \in \mathcal{T}_A} W_\tau\right) \\
&= \sum_{\tau \in \mathcal{T}_A} E(W_\tau) \\
&= \sum_{\tau \in \mathcal{T}_A} \Pr(W_\tau) \\
&\leq \sum_{\tau \in \mathcal{T}_A} \prod_{v \in V(\tau)} \Pr\left([v]\right)
\end{aligned}
$$

Putting It All Together

$$
\begin{aligned}
E(N_A) &= E\left(\sum_{\tau \in \mathcal{T}_A} W_\tau\right) \\
&= \sum_{\tau \in \mathcal{T}_A} E(W_\tau) \\
&= \sum_{\tau \in \mathcal{T}_A} \Pr(W_\tau) \\
&\leq \sum_{\tau \in \mathcal{T}_A} \prod_{v \in V(\tau)} \Pr\left([v]\right) \\
&\leq \sum_{\tau \in \mathcal{T}_A} \prod_{v \in V(\tau)} x'([v])
\end{aligned}
$$

Putting It All Together

$$
\begin{aligned}
E(N_A) &= E\left(\sum_{\tau \in \mathcal{T}_A} W_\tau\right) \\
&= \sum_{\tau \in \mathcal{T}_A} E(W_\tau) \\
&= \sum_{\tau \in \mathcal{T}_A} \Pr(W_\tau) \\
&\leq \sum_{\tau \in \mathcal{T}_A} \prod_{v \in V(\tau)} \Pr\left([v]\right) \\
&\leq \sum_{\tau \in \mathcal{T}_A} \prod_{v \in V(\tau)} x'([v]) \\
&= \sum_{\tau \in \mathcal{T}_A} \frac{x(A)}{1 - x(A)} p_\tau
\end{aligned}
$$

Putting It All Together Cont.

$$E(N_A) \quad \leq \sum_{\tau \in \mathcal{T}_A} \frac{x(A)}{1 - x(A)} p_\tau$$

Putting It All Together Cont.

$$
\begin{aligned}
E(N_A) &\leq \sum_{\tau \in \mathcal{T}_A} \frac{x(A)}{1 - x(A)} p_\tau \\
&= \frac{x(A)}{1 - x(A)} \sum_{\tau \in \mathcal{T}_A} p_\tau
\end{aligned}
$$

Putting It All Together Cont.

$$
\begin{aligned}
E(N_A) &\leq \sum_{\tau \in \mathcal{T}_A} \frac{x(A)}{1 - x(A)} p_\tau \\
&= \frac{x(A)}{1 - x(A)} \sum_{\tau \in \mathcal{T}_A} p_\tau \\
E(N_A) &\leq \frac{x(A)}{1 - x(A)}
\end{aligned}
$$

Summary

- In a probability space where events have limited probability and dependence, we can show that it is not only possible that they all do not happen, but actually efficiently construct the desired object.

Summary

- In a probability space where events have limited probability and dependence, we can show that it is not only possible that they all do not happen, but actually efficiently construct the desired object.
- Thanks.

References

1. P. Erdos and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In Infinite and Finite Sets (Colloq., Keszthely, 1973; dedicated to P. Erdos on his 60th birthday), volume II, pages 609–627. North-Holland, 1975.
2. M. T. Goodrich and R. Tamassia. Algorithm Design and Applications. John Wiley and Sons, 2015.
3. R. A. Moser and G. Tardos. A constructive proof of the general lovász local lemma. Journal of the ACM, 57(2):11:1–11:15, 2010.