Probabilistic Proofs,
Kolmogorov Compleixity and
Laszlo Lovasz Local Lemma

Alexander Shen,
LIF CNRS & Univ. Aix – Marseille

November 2009
Probabilistic proofs of existence
Probabilistic proofs of existence

- If an event has a positive probability, it sometimes happens
Probabilistic proofs of existence

- If an event has a positive probability, it sometimes happens
- If a random variable has an expectation greater than $c$, it is sometimes greater than $c$
An example: MAX-CUT

$G = (V, E)$ is given

coloring: a mapping $V \rightarrow \{\text{black}, \text{white}\}$

multicolor edge: endpoints have different colors

theorem: every graph has a coloring with at least $\frac{|E|}{2}$ multicolor edges

proof: the expected number of multicolor edge for a random coloring is $\frac{|E|}{2}$ (every edge has probability $\frac{1}{2}$).

here derandomisation is trivial: adding a vertex choose the color to maximize the number of multicolor edges

Digression: approximating MAX-CUT (maximal number of multicolor edges) is NP-hard

Similar argument: every 3-CNF has an assignment that satisfies at least $\frac{7}{8}$ of all clauses
An example: MAX-CUT

- $G = (V, E)$ is given
An example: MAX-CUT

- $G = (V, E)$ is given
- coloring: a mapping $V \rightarrow \{black, white\}$
An example: **MAX-CUT**

- $G = (V, E)$ is given
- coloring: a mapping $V \to \{black, white\}$
- multicolor edge: endpoints have different colors

**Theorem:** Every graph has a coloring with at least $\frac{|E|}{2}$ multicolor edges.

**Proof:** The expected number of multicolor edges for a random coloring is $\frac{|E|}{2}$ (every edge has probability $\frac{1}{2}$).

Here, derandomisation is trivial: adding a vertex chooses the color to maximise the number of multicolor edges.

**Digression:** Approximating MAX-CUT (maximal number of multicolor edges) is NP-hard.

**Similar argument:** Every 3-CNF has an assignment that satisfies at least $\frac{7}{8}$ of all clauses.
An example: MAX-CUT

- $G = (V, E)$ is given
- coloring: a mapping $V \rightarrow \{black, white\}$
- multicolor edge: endpoints have different colors
- theorem: every graph has a coloring with at least $\#E/2$ multicolor edges
An example: MAX-CUT

- $G = (V, E)$ is given
- coloring: a mapping $V \rightarrow \{black, white\}$
- multicolor edge: endpoints have different colors
- theorem: every graph has a coloring with at least $\#E/2$ multicolor edges
- proof: the expected number of multicolor edge for a random coloring is $\#E/2$ (every edge has probability $1/2$).
An example: MAX-CUT

- $G = (V, E)$ is given
- coloring: a mapping $V \rightarrow \{black, white\}$
- multicolor edge: endpoints have different colors
- theorem: every graph has a coloring with at least $\#E/2$ multicolor edges
- proof: the expected number of multicolor edge for a random coloring is $\#E/2$ (every edge has probability $1/2$).
- here derandomisation is trivial: adding a vertex choose the color to maximize the number of multicolor edges

Digression: approximating MAX-CUT (maximal number of multicolor edges) is NP-hard

Similar argument: every 3-CNF has an assignment that satisfies at least $7/8$ of all clauses
An example: MAX-CUT

- $G = (V, E)$ is given
- coloring: a mapping $V \rightarrow \{black, white\}$
- multicolor edge: endpoints have different colors
- theorem: every graph has a coloring with at least $\#E/2$ multicolor edges
- proof: the expected number of multicolor edge for a random coloring is $\#E/2$ (every edge has probability $1/2$).
- here derandomisation is trivial: adding a vertex choose the color to maximize the number of multicolor edges
- Digression: approximating MAX-CUT (maximal number of multicolor edges) is NP-hard
An example: MAX-CUT

- $G = (V, E)$ is given
- coloring: a mapping $V \rightarrow \{black, white\}$
- multicolor edge: endpoints have different colors
- theorem: every graph has a coloring with at least $\#E/2$ multicolor edges
- proof: the expected number of multicolor edge for a random coloring is $\#E/2$ (every edge has probability $1/2$).
- here derandomisation is trivial: adding a vertex choose the color to maximize the number of multicolor edges
- Digression: approximating MAX-CUT (maximal number of multicolor edges) is NP-hard
- Similar argument: every 3-CNF has an assignment that satisfies at least $7/8$ of all clauses
One more example: robot in a maze

A labyrinth is drawn in a rectangle (walls go between cells, no exit, connected)

robot is placed inside the maze

instructions: up/down/left/right

if not possible (due to the wall), skip it

Theorem: for every board size there exists a sequence that guarantees that the robot visits all cells (independent of the maze and initial position).
One more example: robot in a maze

- A labyrinth is drawn in a rectangle (walls go between cells, no exit, connected)

![Maze Diagram]

Theorem: for every board size there exists a sequence that guarantees that the robot visits all cells (independent of the maze and initial position).
One more example: robot in a maze

- A labyrinth is drawn in a rectangle (walls go between cells, no exit, connected)

- robot is placed inside the maze
One more example: robot in a maze

- A labyrinth is drawn in a rectangle (walls go between cells, no exit, connected)

- robot is placed inside the maze
- instructions: up/down/left/right
One more example: robot in a maze

- A labyrinth is drawn in a rectangle (walls go between cells, no exit, connected)

- robot is placed inside the maze
- instructions: up/down/left/right
- if not possible (due to the wall), skip it
One more example: robot in a maze

- A labyrinth is drawn in a rectangle (walls go between cells, no exit, connected)

- robot is placed inside the maze
- instructions: up/down/left/right
- if not possible (due to the wall), skip it
- Theorem: for every board size there exists a sequence that guarantees that the robot visits all cells (independent of the maze and initial position).
Random program does the job

Let $N$ be the length of the maximal traversing sequence (for given board size).

A $N$-step random program works with probability at least $4^{-N}$.

A $kN$-step random program does not work with probability at most $(1 - 4^{-N})^k$.

If $k$ is large enough, this probability is less than 1 even multiplied by the number of possible mazes and initial positions (the latter does not depend on $k$).

For this $k$, a random program of length $kN$ with positive probability works for all mazes and initial positions.
Random program does the job

Let $N$ be the length of the maximal traversing sequence (for given board size)
Random program does the job

Let $N$ be the length of the maximal traversing sequence (for given board size)

$N$-step random program works with probability at least $4^{-N}$
Random program does the job

- Let $N$ be the length of the maximal traversing sequence (for given board size)
- $N$-step random program works with probability at least $4^{-N}$
- $kN$-step random program does not work with probability at most $(1 - 4^{-N})^k$
Random program does the job

- Let $N$ be the length of the maximal traversing sequence (for given board size)
- $N$-step random program works with probability at least $4^{-N}$
- $kN$-step random program does not work with probability at most $(1 - 4^{-N})^k$
- if $k$ is large enough, this probability is less than 1 even multiplied by the number of possible mazes and initial positions (the latter does not depend on $k$)
Random program does the job

- Let $N$ be the length of the maximal traversing sequence (for given board size)
- $N$-step random program works with probability at least $4^{-N}$
- $kN$-step random program does not work with probability at most $(1 - 4^{-N})^k$
- If $k$ is large enough, this probability is less than 1 even multiplied by the number of possible mazes and initial positions (the latter does not depend on $k$)
- For this $k$ a random program of length $kN$ with positive probability works for all mazes and initial positions
Box dimensions

A rectangular box is allowed if the sum of dimensions does not exceed the threshold:

\[ w + l + h < M \]

Is it possible to hide a prohibited box in a legal one?

Possible if only the maximal dimension is taken into account

Theorem: if a box \( B_1 \) is inside \( B_2 \), the sum of dimensions for \( B_1 \) does not exceed the sum of dimensions for \( B_2 \).

Proof: Look!

(Expected value of the projection to a random line is proportional to the sum of dimensions)
Box dimensions

- A rectangular box is allowed if the sum of dimensions does not exceed the threshold: $w + l + h < M$
Box dimensions

- A rectangular box is allowed if the sum of dimensions does not exceed the threshold: $w + l + h < M$
- Is it possible to hide a prohibited box in a legal one?
Box dimensions

- A rectangular box is allowed if the sum of dimensions does not exceed the threshold: \( w + l + h < M \)
- Is it possible to hide a prohibited box in a legal one?
- Possible if only the maximal dimension is taken into account
Box dimensions

- A rectangular box is allowed if the sum of dimensions does not exceed the threshold: $w + l + h < M$
- Is it possible to hide a prohibited box in a legal one?
- Possible if only the maximal dimension is taken into account
- Theorem: if a box $B_1$ is inside $B_2$, the sum of dimensions for $B_1$ does not exceed the sum of dimensions for $B_2$

(Expected value of the projection to a random line is proportional to the sum of dimensions)
Box dimensions

- A rectangular box is allowed if the sum of dimensions does not exceed the threshold: \( w + l + h < M \)
- Is it possible to hide a prohibited box in a legal one?
- Possible if only the maximal dimension is taken into account
- Theorem: if a box \( B_1 \) is inside \( B_2 \), the sum of dimensions for \( B_1 \) does not exceed the sum of dimensions for \( B_2 \)
- Proof: Look!
Box dimensions

- A rectangular box is allowed if the sum of dimensions does not exceed the threshold: \( w + l + h < M \)
- Is it possible to hide a prohibited box in a legal one?
- Possible if only the maximal dimension is taken into account
- Theorem: if a box \( B_1 \) is inside \( B_2 \), the sum of dimensions for \( B_1 \) does not exceed the sum of dimensions for \( B_2 \)
- Proof: Look!

(Expected value of the projection to a random line is proportional to the sum of dimensions)
Uniform minors: probabilistic argument

- A $k \times k$-minor in $n \times n$-matrix: select $k$ rows and $k$ columns
- A minor in a Boolean matrix is uniform if it contains only ones or only zeros
- What size of uniform minor can be guaranteed in $n \times n$ Boolean matrix?
- Theorem: if $k > 2 \log n + 1$, there exists a $n \times n$ Boolean matrix that has no uniform $k \times k$ minors.
- Probability argument: take a random $n \times n$ matrix. For a given position of a minor the probability to see an uniform minor there is $2^{-k^2} \times 2$. There are at most $n^2 k$ positions for a $k \times k$ minor. So if $n^2 k^2 - k^2 + 1 < 1$, a matrix without uniform minors exists. Taking logarithms, we get $2k \log n - k^2 + 1 < 0$ which is guaranteed if $k > 2 \log n + 1$. 
Uniform minors: probabilistic argument

- A $k \times k$-minor in $n \times n$-matrix: select $k$ rows and $k$ columns
Uniform minors: probabilistic argument

- A $k \times k$-minor in $n \times n$-matrix: select $k$ rows and $k$ columns
- A minor in a Boolean matrix is uniform if it contains only ones or only zeros
Uniform minors: probabilistic argument

- A $k \times k$-minor in $n \times n$-matrix: select $k$ rows and $k$ columns
- A minor in a Boolean matrix is uniform if it contains only ones or only zeros
- What size of uniform minor can be guaranteed in $n \times n$ Boolean matrix?

Theorem: if $k > 2 \log n + 1$, there exists a $n \times n$ Boolean matrix that has no uniform $k \times k$ minors.

Probability argument: take a random $n \times n$ matrix. For a given position of a minor the probability to see an uniform minor there is $2^{-k^2}$. There are at most $n^2$ positions for a $k \times k$ minor. So if $n^2 > k^2 - k + 1$, a matrix without uniform minors exists. Taking logarithms, we get $2^k \log n - k^2 + 1 < 0$ which is guaranteed if $k > 2 \log n + 1$. 
Uniform minors: probabilistic argument

- A $k \times k$-minor in $n \times n$-matrix: select $k$ rows and $k$ columns
- A minor in a Boolean matrix is uniform if it contains only ones or only zeros
- What size of uniform minor can be guaranteed in $n \times n$ Boolean matrix?
- Theorem: if $k > 2 \log n + 1$, there exists a $n \times n$ Boolean matrix that has no uniform $k \times k$ minors.
Uniform minors: probabilistic argument

- A $k \times k$-minor in $n \times n$-matrix: select $k$ rows and $k$ columns
- A minor in a Boolean matrix is uniform if it contains only ones or only zeros
- What size of uniform minor can be guaranteed in $n \times n$ Boolean matrix?
- Theorem: if $k > 2 \log n + 1$, there exists a $n \times n$ Boolean matrix that has no uniform $k \times k$ minors.
- Probability argument: take a random $n \times n$ matrix. For a given position of a minor the probability to see an uniform minor there is $2^{-k^2} \times 2$. There are at most $n^{2k}$ positions for a $k \times k$ minor. So if $n^{2k}2^{-k^2+1} < 1$, a matrix without uniform minors exists. Taking logarithms, we get $2k \log n - k^2 + 1 < 0$ which is guaranteed if $k > 2 \log n + 1$. 
Uniform minors: combinatorial and complexity versions

Counting version: there are \( 2^{n^2} \) matrices with an uniform minor in a given position, then we multiply this number by the number of possible positions and note that the sum is less than the number of \( n \times n \) matrices.

Complexity version: let us prove that a incompressible matrix has no uniform minors. In other words, a matrix that has uniform minor is compressible. Indeed, it can be described by specifying the position of that minor (\( 2^k \) indices in 1...\( n \) range, i.e., \( 2^k \log n \) bits), the bit in the minor (1 bit) and the remaining \( n^2 - k \) bits in the matrix, so if \( 2^k \log n + 1 + n^2 - k < n^2 \), the matrix is compressible.

This is not a rigorous proof since complexity is defined up to a constant.
Uniform minors: combinatorial and complexity versions

- Counting version: there are $2^{n^2-k^2+1}$ matrices with an uniform minor in a given position, then we multiply this number by the number of possible positions and note that the sum is less than the number of $n \times n$ matrices.

- Complexity version: let us prove that a incompressible matrix has no uniform minors. In other words, a matrix that has uniform minor is compressible. Indeed, it can be described by specifying the position of that minor ($2^k$ indices in $1...n$ range, i.e., $2^k \log n$ bits), the bit in the minor (1 bit) and the remaining $n^2-k^2$ bits in the matrix, so if $2^k \log n + 1 + n^2-k^2 < n^2$, the matrix is compressible.

This is not a rigorous proof since complexity is defined up to a constant.
Uniform minors: combinatorial and complexity versions

- Counting version: there are $2^{n^2-k^2+1}$ matrices with an uniform minor in a given position, then we multiply this number by the number of possible positions and note that the sum is less than the number of $n \times n$ matrices.

- Complexity version: let us prove that a incompressible matrix has no uniform minors. In other words, a matrix that has uniform minor is compressible. Indeed, in can be described by specifying the position of that minor ($2k$ indices in $1 \ldots n$ range, i.e., $2k \log n$ bits), the bit in the minor (1 bit) and the remaining $n^2 - k^2$ bits in the matrix, so if $2k \log n + 1 + n^2 - k^2 < n^2$, the matrix is compressible.
Uniform minors: combinatorial and complexity versions

- Counting version: there are $2^{n^2-k^2+1}$ matrices with an uniform minor in a given position, then we multiply this number by the number of possible positions and note that the sum is less than the number of $n \times n$ matrices.

- Complexity version: let us prove that a incompressible matrix has no uniform minors. In other words, a matrix that has uniform minor is compressible. Indeed, in can be described by specifying the position of that minor ($2k$ indices in $1 \ldots n$ range, i.e., $2k \log n$ bits), the bit in the minor (1 bit) and the remaining $n^2 - k^2$ bits in the matrix, so if $2k \log n + 1 + n^2 - k^2 < n^2$, the matrix is compressible

- This is not a rigorous proof since complexity is defined up to a constant.
Derandomization?

Is it possible to replace an existence proof using probabilistic arguments by an explicit construction?

(The notion of explicit construction is not formally defined)

Sometimes derandomization is easy (e.g., the first example with a graph), or an alternative proof can be easily found (e.g., the robot example)

Sometimes an open problem (e.g., the existence of Boolean functions that require circuits of exponential size)

Sometime explicit constructions exist but are rather complicated and do not achieve the best possible parameters (expanders, codes)
Derandomization?

- Is it possible to replace an existence proof using probabilistic arguments by an explicit construction?
Derandomization?

- Is it possible to replace an existence proof using probabilistic arguments by an explicit construction?
- (The notion of explicit construction is not formally defined)
Derandomization?

- Is it possible to replace an existence proof using probabilistic arguments by an explicit construction?
- (The notion of explicit construction is not formally defined)
- Sometimes derandomization is easy (e.g., the first example with a graph), or an alternative proof can be easily found (e.g., the robot example)
Derandomization?

- Is it possible to replace an existence proof using probabilistic arguments by an explicit construction?
- (The notion of explicit construction is not formally defined)
- Sometimes derandomization is easy (e.g., the first example with a graph), or an alternative proof can be easily found (e.g., the robot example)
- Sometimes an open problem (e.g., the existence of Boolean functions that require circuits of exponential size)
Derandomization?

- Is it possible to replace an existence proof using probabilistic arguments by an explicit construction?
- (The notion of explicit construction is not formally defined)
- Sometimes derandomization is easy (e.g., the first example with a graph), or an alternative proof can be easily found (e.g., the robot example)
- Sometimes an open problem (e.g., the existence of Boolean functions that require circuits of exponential size)
- Sometimes explicit constructions exist but are rather complicated and do not achieve the best possible parameters (expanders, codes)
Beyond combinatorics: independence

if $\sum_i \Pr[A_i] < 1$, then one can avoid all $A_i$ with positive probability.

if $A_i$ are independent, weaker condition $\forall i \Pr[A_i] < 1$ is enough: one can avoid all $A_i$ with positive probability.

Laslo Lovasz Local Lemma deals with partial independence.

Each node in a rectangular grid may have one of 10 colors; each edge prohibits one of 100 color combinations (different for different edges). LLLL guarantees that there exists a coloring that satisfies all restrictions.
Beyond combinatorics: independence

- if $\sum_i \Pr[A_i] < 1$, then one can avoid all $A_i$ with positive probability $1 - \sum_i \Pr[A_i]$

Laslo Lovasz Local Lemma deals with partial independence

Each node in a rectangular grid may have one of 10 colors; each edge prohibits one of 100 color combinations (different for different edges). LLLL guarantees that there exists a coloring that satisfies all restrictions.
Beyond combinatorics: independence

- if $\sum_i \Pr[A_i] < 1$, then one can avoid all $A_i$ with positive probability $1 - \sum_i \Pr[A_i]$

- if $A_i$ are independent, weaker condition $\forall i \Pr[A_i] < 1$ is enough: one can avoid all $A_i$ with positive probability $\prod_i (1 - \Pr[A_i])$
Beyond combinatorics: independence

- if $\sum_i \Pr[A_i] < 1$, then one can avoid all $A_i$ with positive probability $1 - \sum_i \Pr[A_i]$
- if $A_i$ are independent, weaker condition $\forall i \Pr[A_i] < 1$ is enough: one can avoid all $A_i$ with positive probability $\prod_i (1 - \Pr[A_i])$
- Laslo Lovasz Local Lemma deals with partial independence
Beyond combinatorics: independence

- if $\sum_i \Pr[A_i] < 1$, then one can avoid all $A_i$ with positive probability $1 - \sum_i \Pr[A_i]$

- if $A_i$ are independent, weaker condition $\forall i \ Pr[A_i] < 1$ is enough: one can avoid all $A_i$ with positive probability $\prod_i (1 - \Pr[A_i])$

- Laslo Lovasz Local Lemma deals with partial independence

- Each node in a rectangular grid may have one of 10 colors; each edge prohibits one of 100 color combinations (different for different edges). LLLL guarantees that there exists a coloring that satisfies all restrictions.
Laszlo Lovasz Local Lemma

Let $A_1, \ldots, A_n$ are events indexed by vertices of an (undirected) graph. Let $N(i)$ be the set of all neighbors of $i$ (not including $i$). Assume that for every $i$ the event $A_i$ is independent with the tuple of all events $A_j$ with $j \notin N(i)$. Assume that for every $i$ an upper bound $\varepsilon_i < 1$ for $\Pr[A_i]$ is chosen and, moreover,

$$\Pr[A_i] \leq \varepsilon_i \prod_{j \in N(i)} (1 - \varepsilon_j).$$

Then

$$\Pr[\neg A_1 \land \neg A_2 \land \ldots \land \neg A_n] \geq \prod_i (1 - \varepsilon_i).$$
Laszlo Lovasz Local Lemma

Let $A_1, \ldots, A_n$ are events indexed by vertices of an (undirected) graph. Let $N(i)$ be the set of all neighbors of $i$ (not including $i$). Assume that for every $i$ the event $A_i$ is independent with the tuple of all events $A_j$ with $j \notin N(i)$. Assume that for every $i$ an upper bound $\varepsilon_i < 1$ for $\Pr[A_i]$ is chosen and, moreover,

$$\Pr[A_i] \leq \varepsilon_i \prod_{j \in N(i)} (1 - \varepsilon_j).$$

Then

$$\Pr[\neg A_1 \land \neg A_2 \land \ldots \land \neg A_n] \geq \prod_i (1 - \varepsilon_i).$$

- If all $A_i$ are independent, $\varepsilon_i = \Pr[A_i]$
Laszlo Lovasz Local Lemma

Let $A_1, \ldots, A_n$ are events indexed by vertices of an (undirected) graph. Let $N(i)$ be the set of all neighbors of $i$ (not including $i$). Assume that for every $i$ the event $A_i$ is independent with the tuple of all events $A_j$ with $j \not\in N(i)$. Assume that for every $i$ an upper bound $\varepsilon_i < 1$ for $\Pr[A_i]$ is chosen and, moreover,

$$\Pr[A_i] \leq \varepsilon_i \prod_{j \in N(i)} (1 - \varepsilon_j).$$

Then

$$\Pr[\neg A_1 \land \neg A_2 \land \ldots \land \neg A_n] \geq \prod_i (1 - \varepsilon_i).$$

- If all $A_i$ are independent, $\varepsilon_i = \Pr[A_i]$
- If there is no information about dependence (complete graph), and $\sum \Pr[A_i] < 1/4$, one can let $\varepsilon_i = 2 \Pr[A_i]$: the product of $(1 - \varepsilon_i)$ is at least $1 - \sum \varepsilon_i > 1/2$. 
Let $A_1, \ldots, A_n$ are events indexed by vertices of an (undirected) graph. Let $N(i)$ be the set of all neighbors of $i$ (not including $i$). Assume that for every $i$ the event $A_i$ is independent with the tuple of all events $A_j$ with $j \notin N(i)$. Assume that for every $i$ an upper bound $\varepsilon_i < 1$ for $\Pr[A_i]$ is chosen and, moreover, 

$$\Pr[A_i] \leq \varepsilon_i \prod_{j \in N(i)} (1 - \varepsilon_j).$$

Then 

$$\Pr[\neg A_1 \land \neg A_2 \land \ldots \land \neg A_n] \geq \prod_{i}(1 - \varepsilon_i).$$

In our example events correspond to edges; neighbors are edges that share a vertex (6 of them). Choosing the same $\varepsilon$ for every edge, we need 

$$1/100 < \varepsilon (1 - \varepsilon)^6$$

If $\varepsilon = 1/6$, the rhs is about $1/6e \gg 1/100$. 

\[\]
Proof of the LLLL
Proof of the LLLL

- generalization:
  \[ \Pr[\neg A_i \land \neg A_j \land \ldots | \neg A_p \land \neg A_q \land \ldots] \geq (1 - \varepsilon_i)(1 - \varepsilon_j) \ldots \]
Proof of the LLLL

- generalization:
  \[ \Pr[\neg A_i \land \neg A_j \land \ldots | \neg A_p \land \neg A_q \land \ldots] \geq (1 - \varepsilon_i)(1 - \varepsilon_j) \ldots \]

- enough to show for one event (and many conditions):
  \[ \Pr[\neg A_i \land \neg A_j | \ldots] = \Pr[\neg A_j | \ldots] \Pr[\neg A_i | \neg A_j \land \ldots] \]
Proof of the LLLL

- generalization:
  \[ \Pr[\neg A_i \land \neg A_j \land \ldots | \neg A_p \land \neg A_q \land \ldots] \geq (1 - \varepsilon_i)(1 - \varepsilon_j) \ldots \]

- enough to show for one event (and many conditions):
  \[ \Pr[\neg A_i \land \neg A_j | \ldots] = \Pr[\neg A_j | \ldots] \Pr[\neg A_i | \neg A_j \land \ldots] \]

- for the complement: \[ \Pr[A_i | \neg A_j \land \neg A_k \land \ldots] \leq \varepsilon_i. \]
Proof of the LLLL

- generalization:
  \[ \Pr[\neg A_i \land \neg A_j \land \ldots | \neg A_p \land \neg A_q \land \ldots] \geq (1 - \varepsilon_i)(1 - \varepsilon_j) \ldots \]

- enough to show for one event (and many conditions):
  \[ \Pr[\neg A_i \land \neg A_j | \ldots] = \Pr[\neg A_j | \ldots] \Pr[\neg A_i | \neg A_j \land \ldots] \]

- for the complement: \( \Pr[A_i | \neg A_j \land \neg A_k \land \ldots] \leq \varepsilon_i. \)

- separating neighbors and non-neighbors \((j, k \text{ are neighbors, } l, \ldots \text{ are not})\):
  \[
  \Pr[A_i | \neg A_j \land \neg A_k \land \neg A_l \land \ldots] = \\
  = \frac{\Pr[A_i \land \neg A_j \land \neg A_k | \neg A_l \land \ldots]}{\Pr[\neg A_j \land \neg A_k | \neg A_l \land \ldots]} \leq \frac{\Pr[A_i]}{(1-\varepsilon_j)(1-\varepsilon_k)} \leq \varepsilon_i
  \]
Proof of the LLLL

- generalization:
  \[ \Pr[\neg A_i \land \neg A_j \land \ldots \land \neg A_p \land \neg A_q \land \ldots] \geq (1 - \varepsilon_i)(1 - \varepsilon_j) \ldots \]

- enough to show for one event (and many conditions):
  \[ \Pr[\neg A_i \land \neg A_j | \ldots] = \Pr[\neg A_j | \ldots] \Pr[\neg A_i | \neg A_j \land \ldots] \]

- for the complement: \[ \Pr[A_i | \neg A_j \land \neg A_k \land \ldots] \leq \varepsilon_i. \]

- separating neighbors and non-neighbors \((j, k \text{ are neighbors, } l, \ldots \text{ are not}):\)
  \[ \Pr[A_i | \neg A_j \land \neg A_k \land \neg A_l \land \ldots] = \]
  \[ = \frac{\Pr[A_i \land \neg A_j \land \neg A_k | \neg A_l \land \ldots]}{\Pr[\neg A_j \land \neg A_k | \neg A_l \land \ldots]} \leq \frac{\Pr[A_i]}{(1-\varepsilon_j)(1-\varepsilon_k)} \leq \varepsilon_i \]

- using induction (less events in the condition)
Forbidden substrings

Let $X_1, \ldots, X_n$ be some bit strings. We want to construct a sequence $\omega$ that does not contain $X_i$ as substrings (factors).

Not always possible: e.g., 00, 11, 0101.

Quantitative results: if forbidden strings are long enough and there are not too many of them, a sequence $\omega$ exists.

Let $\alpha < 1$. Assume that for every $n$ there is at most $2^{\alpha n}$ forbidden (bit) strings. Then there exists a number $c$ and a bit sequence $\omega$ that has no forbidden substrings of length $> c$.

(Kolmogorov complexity version) There exists a sequence $\omega$ such that any substring $x$ is $\omega$ has complexity at least $\alpha |x| - O(1)$.

Statements are equivalent: there is at most $2^{\alpha n}$ sequences of length $n$ and complexity $< \alpha n$; on the other hand, if $X$ is a set of forbidden strings and there is at most $2^{\alpha n}$ forbidden strings of length $n$, they are all simple (have complexity $\alpha n + o(n)$) relative to $X$. (Non-relativized version can be also used.)
Forbidden substrings

- Let $X_1, \ldots, X_n$ be some bit strings. We want to construct a sequence $\omega$ that does not contain $X_i$ as substrings (factors).
Forbidden substrings

- Let $X_1, \ldots, X_n$ be some bit strings. We want to construct a sequence $\omega$ that does not contain $X_i$ as substrings (factors).
- not always possible: e.g., 00, 11, 0101
Forbidden substrings

- Let $X_1, \ldots, X_n$ be some bit strings. We want to construct a sequence $\omega$ that does not contain $X_i$ as substrings (factors).
- Not always possible: e.g., 00, 11, 0101
- Quantitative results: if forbidden strings are long enough and there are not too many of them, a sequence $\omega$ exists.
Forbidden substrings

Let $X_1, \ldots, X_n$ be some bit strings. We want to construct a sequence $\omega$ that does not contain $X_i$ as substrings (factors).

not always possible: e.g., 00, 11, 0101

quantitative results: if forbidden strings are long enough and there are not too many of them, a sequence $\omega$ exists

Let $\alpha < 1$. Assume that for every $n$ there is at most $2^{\alpha n}$ forbidden (bit) strings. Then there exists a number $c$ and a bit sequence $\omega$ that has no forbidden substrings of length $> c$. 
Forbidden substrings

- Let $X_1, \ldots, X_n$ be some bit strings. We want to construct a sequence $\omega$ that does not contain $X_i$ as substrings (factors).
- Not always possible: e.g., 00, 11, 0101
- Quantitative results: if forbidden strings are long enough and there are not too many of them, a sequence $\omega$ exists.
- Let $\alpha < 1$. Assume that for every $n$ there is at most $2^{\alpha n}$ forbidden (bit) strings. Then there exists a number $c$ and a bit sequence $\omega$ that has no forbidden substrings of length $> c$.
- (Kolmogorov complexity version) There exists a sequence $\omega$ such that any substring $x$ is $\omega$ has complexity at least $\alpha |x| - O(1)$.

Statements are equivalent: there is at most $2^{\alpha n}$ sequences of length $n$ and complexity $< \alpha n$; on the other hand, if $X$ is a set of forbidden strings and there is at most $2^{\alpha n}$ forbidden strings of length $n$, they are all simple (have complexity $\alpha n + o(n)$) relative to $X$. (Non-relativized version can be also used.)
Combinatorial and complexity proofs

Combinatorial: use LLLL

Complexity: construct the sequence inductively adding blocks of some length $M$; each added block should increase the complexity at least by $\beta M$ for some $\beta$ in $(\alpha, 1)$.

such a block exists since we can take a block that is random relative to the prefix of the sequence; each group of $s$ consecutive blocks increases complexity at least by $\beta s M$ and therefore has complexity at least $\beta s M$. For non-aligned blocks we discard some part of them (using the difference between $\alpha$ and $\beta$ to compensate for the losses).
Combinatorial and complexity proofs

- Combinatorial: use LLLL

- Complexity: construct the sequence inductively adding blocks of some length $M$; each added block should increase the complexity at least by $\beta M$ for some $\beta$ in $(\alpha, 1)$. Such a block exists since we can take a block that is random relative to the prefix of the sequence; each group of consecutive blocks increases complexity at least by $\beta sM$ and therefore has complexity at least $\beta sM$. For non-aligned blocks we discard some part of them (using the difference between $\alpha$ and $\beta$ to compensate for the losses).
Combinatorial and complexity proofs

- Combinatorial: use LLLL
- Complexity: construct the sequence inductively adding blocks of some length $M$; each added block should increase the complexity at least by $\beta M$ for some $\beta$ in $(\alpha, 1)$.
Combinatorial and complexity proofs

- Combinatorial: use LLLL
- Complexity: construct the sequence inductively adding blocks of some length $M$; each added block should increase the complexity at least by $\beta M$ for some $\beta$ in $(\alpha, 1)$.
- such a block exists since we can take a block that is random relative to the prefix of the sequence; each group of $s$ consecutive blocks increases complexity at least by $\beta sM$ and therefore has complexity at least $\beta sM$. For non-aligned blocks we discard some part of them (using the difference between $\alpha$ and $\beta$ to compensate for the losses).
Forbidden subsequences

Let $A$ be a finite set of indices (integers) and let $\omega$ be a sequence. By $\omega(A)$ we denote the subsequence of $\omega$ with indices in $A$ (in increasing order).

A restriction "$\omega(A) \neq X$" is specified by $A$ and binary string $X$ (of length $\#A$).

In other terms, we have Boolean variables (bits of $\omega$) and clauses: e.g., $w_5 \lor \neg w_7 \lor w_{11}$ says that $\omega(\{5, 7, 11\}) \neq 010$.

Rumyantsev: for every $\alpha < 1$ there exists a sequence $\omega$ such that for every finite $A$ the complexity $K(A, \omega(A) | t)$ exceeds $\alpha \#A - O(1)$ for some $t \in A$. [Proof: use LLLL]

Corollary: if $A$ has small complexity with respect to every its element, then $K(\omega(A))$ is large.

so there is a sequence such that every substring has high complexity and a two-dimensional sequence such that every rectangle has a high complexity (close to its area)
Forbidden subsequences

Let $A$ be a finite set of indices (integers) and let $\omega$ be a sequence. By $\omega(A)$ we denote the subsequence of $\omega$ with indices in $A$ (in increasing order).
Forbidden subsequences

- Let $A$ be a finite set of indices (integers) and let $\omega$ be a sequence. By $\omega(A)$ we denote the subsequence of $\omega$ with indices in $A$ (in increasing order).
- A restriction “$\omega(A) \neq X$” is specified by $A$ and binary string $X$ (of length $\#A$).
Forbidden subsequences

Let $A$ be a finite set of indices (integers) and let $\omega$ be a sequence. By $\omega(A)$ we denote the subsequence of $\omega$ with indices in $A$ (in increasing order).

A restriction “$\omega(A) \neq X$” is specified by $A$ and binary string $X$ (of length $\#A$).

In other terms, we have Boolean variables (bits of $\omega$) and clauses: e.g., $w_5 \lor \neg w_7 \lor w_{11}$ says that $\omega(\{5, 7, 11\}) \neq 010$. 

Rumyantsev: for every $\alpha < 1$ there exists a sequence $\omega$ such that for every finite $A$ the complexity $K(A, \omega(A) | t)$ exceeds $\alpha \#A - O(1)$ for some $t \in A$. [Proof: use LLLL]

Corollary: if $A$ has small complexity with respect to every its element, then $K(\omega(A))$ is large.

so there is a sequence such that every substring has high complexity

and a two-dimensional sequence such that every rectangle has a high complexity (close to its area)
Forbidden subsequences

- Let $A$ be a finite set of indices (integers) and let $\omega$ be a sequence. By $\omega(A)$ we denote the subsequence of $\omega$ with indices in $A$ (in increasing order).
- A restriction “$\omega(A) \neq X$” is specified by $A$ and binary string $X$ (of length $\#A$).
- In other terms, we have Boolean variables (bits of $\omega$) and clauses: e.g., $w_5 \lor \neg w_7 \lor w_{11}$ says that $\omega(\{5, 7, 11\}) \neq 010$.
- Rumyantsev: for every $\alpha < 1$ there exists a sequence $\omega$ such that for every finite $A$ the complexity $K(A, \omega(A)|t)$ exceeds $\alpha \#A - O(1)$ for some $t \in A$. [Proof: use LLLL]
Forbidden subsequences

Let $A$ be a finite set of indices (integers) and let $\omega$ be a sequence. By $\omega(A)$ we denote the subsequence of $\omega$ with indices in $A$ (in increasing order).

A restriction “$\omega(A) \neq X$” is specified by $A$ and binary string $X$ (of length $\#A$).

In other terms, we have Boolean variables (bits of $\omega$) and clauses: e.g., $\omega_5 \lor \neg \omega_7 \lor \omega_{11}$ says that $\omega(\{5, 7, 11\}) \neq 010$.

Rumyantsev: for every $\alpha < 1$ there exists a sequence $\omega$ such that for every finite $A$ the complexity $K(A, \omega(A)|t)$ exceeds $\alpha \#A - O(1)$ for some $t \in A$. [Proof: use LLLL]

Corollary: if $A$ has small complexity with respect to every its element, then $K(\omega(A))$ is large.
Forbidden subsequences

- Let $A$ be a finite set of indices (integers) and let $\omega$ be a sequence. By $\omega(A)$ we denote the subsequence of $\omega$ with indices in $A$ (in increasing order).

- A restriction “$\omega(A) \neq X$” is specified by $A$ and binary string $X$ (of length $\#A$).

- In other terms, we have Boolean variables (bits of $\omega$) and clauses: e.g., $w_5 \lor \neg w_7 \lor w_{11}$ says that $\omega(\{5, 7, 11\}) \neq 010$.

- Rumyantsev: for every $\alpha < 1$ there exists a sequence $\omega$ such that for every finite $A$ the complexity $K(A, \omega(A)|t)$ exceeds $\alpha \#A - O(1)$ for some $t \in A$. [Proof: use LLL]

- Corollary: if $A$ has small complexity with respect to every its element, then $K(\omega(A))$ is large.

- so there is a sequence such that every substring has high complexity
Forbidden subsequences

- Let $A$ be a finite set of indices (integers) and let $\omega$ be a sequence. By $\omega(A)$ we denote the subsequence of $\omega$ with indices in $A$ (in increasing order).
- A restriction “$\omega(A) \neq X$” is specified by $A$ and binary string $X$ (of length $\# A$).
- In other terms, we have Boolean variables (bits of $\omega$) and clauses: e.g., $w_5 \lor \neg w_7 \lor w_{11}$ says that $\omega(\{5, 7, 11\}) \neq 010$.
- Rumyantsev: for every $\alpha < 1$ there exists a sequence $\omega$ such that for every finite $A$ the complexity $K(A, \omega(A) | t)$ exceeds $\alpha \# A - O(1)$ for some $t \in A$. [Proof: use LLLL]
- Corollary: if $A$ has small complexity with respect to every its element, then $K(\omega(A))$ is large.
- so there is a sequence such that every substring has high complexity
- and a two-dimensional sequence such that every rectangle has a high complexity (close to its area)
A constructive version of LLLL

General statement of LLLL has nothing to do with algorithms

Most applications show the existence of some constructive object (assignment, sequence, coloring etc.)

The statement itself does not provide a reasonable probabilistic algorithm (the guaranteed probability is exponentially small)

However, such an algorithm exists (Moser, 2009) it is the simple one: resample variables that appear in the violated restriction until everything is OK

The proof of the most general result (Moser and Tardos) is a bit mysterious but a very simple argument exists for special cases (as explained by Fortnow using complexity)
A constructive version of LLLL

- General statement of LLLL has nothing to do with algorithms
A constructive version of LLLL

- General statement of LLLL has nothing to do with algorithms
- Most applications show the existence of some constructive object (assignment, sequence, coloring etc.)
A constructive version of LLLL

- General statement of LLLL has nothing to do with algorithms.
- Most applications show the existence of some constructive object (assignment, sequence, coloring etc.).
- The statement itself does not provide a reasonable probabilistic algorithm (the guaranteed probability is exponentially small).

(Moser, 2009) it is the simple one: resample variables that appear in the violated restriction until everything is OK.

The proof of the most general result (Moser and Tardos) is a bit mysterious, but a very simple argument exists for special cases (as explained by Fortnow using complexity).
A constructive version of LLLL

- General statement of LLLL has nothing to do with algorithms
- Most applications show the existence of some constructive object (assignment, sequence, coloring etc.)
- The statement itself does not provide a reasonable probabilistic algorithm (the guaranteed probability is exponentially small)
- However, such an algorithm exists
A constructive version of LLLL

- General statement of LLLL has nothing to do with algorithms
- Most applications show the existence of some constructive object (assignment, sequence, coloring etc.)
- The statement itself does not provide a reasonable probabilistic algorithm (the guaranteed probability is exponentially small)
- However, such an algorithm exists
- (Moser, 2009) it is the simple one: resample variables that appear in the violated restriction until everything is OK
A constructive version of LLLL

- General statement of LLLL has nothing to do with algorithms
- Most applications show the existence of some constructive object (assignment, sequence, coloring etc.)
- The statement itself does not provide a reasonable probabilistic algorithm (the guaranteed probability is exponentially small)
- However, such an algorithm exists
- (Moser, 2009) it is the simple one: resample variables that appear in the violated restriction until everything is OK
- The proof of the most general result (Moser and Tardos) is a bit misterous
A constructive version of LLLL

- General statement of LLLL has nothing to do with algorithms
- Most applications show the existence of some constructive object (assignment, sequence, coloring etc.)
- The statement itself does not provide a reasonable probabilistic algorithm (the guaranteed probability is exponentially small)
- However, such an algorithm exists
- (Moser, 2009) it is the simple one: resample variables that appear in the violated restriction until everything is OK
- The proof of the most general result (Moser and Tardos) is a bit misterous
- but a very simple argument exists for special cases (as explained by Fortnow using complexity)
The special case: forbidden subsequences

Boolean variables $w_1, \ldots, w_N$.

Clauses: each of $M$ clauses involves $m$ variables and prohibits some combination of values: $\overline{w_i^1} \lor \cdots \lor \overline{w_i^m}$.

Looking for a satisfying assignment (that does not violate any clause).

Statement: it exists if clauses are not very small and not too dependent.

Two clauses intersect if they have common variables.

Statement: if each clause intersects at most $t$ others and $t < 2m/8$, then there exists a satisfying assignment.

... and it can be found with high probability by a polynomial probabilistic algorithm.
The special case: forbidden subsequences

- Boolean variables $w_1, \ldots, w_N$.
The special case: forbidden subsequences

- Boolean variables $w_1, \ldots, w_N$.
- Clauses: each of $M$ clauses involves $m$ variables and prohibits some combination of values: $\lnot w_{i_1} \lor \ldots \lor \lnot w_{i_m}$. 

Looking for a satisfying assignment (that does not violate any clause).

Statement: if clauses are not very small and not too dependent

Two clauses intersect if they have common variables

Statement: if each clause intersects at most $t$ others and $t < 2m/8$, then there exists a satisfying assignment
The special case: forbidden subsequences

- Boolean variables \( w_1, \ldots, w_N \).
- Clauses: each of \( M \) clauses involves \( m \) variables and prohibits some combination of values: \( \neg w_{i_1} \lor \ldots \lor \neg w_{i_m} \).
- Looking for a satisfying assignment (that does not violate any clause).
The special case: forbidden subsequences

- Boolean variables $w_1, \ldots, w_N$.
- Clauses: each of $M$ clauses involves $m$ variables and prohibits some combination of values: $\neg w_{i_1} \lor \ldots \lor \neg w_{i_m}$.
- Looking for a satisfying assignment (that does not violate any clause).
- Statement: it exists if clauses are not very small and not too dependent.
The special case: forbidden subsequences

- Boolean variables $w_1, \ldots, w_N$.
- Clauses: each of $M$ clauses involves $m$ variables and prohibits some combination of values: $\neg w_{i_1} \lor \ldots \lor \neg w_{i_m}$.
- Looking for a satisfying assignment (that does not violate any clause).
- Statement: it exists if clauses are not very small and not too dependent
- Two clauses intersect if they have common variables
The special case: forbidden subsequences

- Boolean variables $w_1, \ldots, w_N$.
- Clauses: each of $M$ clauses involves $m$ variables and prohibits some combination of values: $\neg w_{i_1} \lor \ldots \lor \neg w_{i_m}$.
- Looking for a satisfying assignment (that does not violate any clause).
- Statement: it exists if clauses are not very small and not too dependent
- Two clauses intersect if there have common variables
- Statement: if each clause intersects at most $t$ others and $t < 2^m/8$, then there exists a satisfying assignment
The special case: forbidden subsequences

- Boolean variables $w_1, \ldots, w_N$.
- Clauses: each of $M$ clauses involves $m$ variables and prohibits some combination of values: $\neg w_{i_1} \lor \ldots \lor \neg w_{i_m}$.
- Looking for a satisfying assignment (that does not violate any clause).
- Statement: it exists if clauses are not very small and not too dependent.
- Two clauses intersect if there have common variables.
- Statement: if each clause intersects at most $t$ others and $t < 2^m/8$, then there exists a satisfying assignment.
- ... and it can be found with high probability by a polynomial probabilistic algorithm.
Resampling algorithm

Main algorithm: start with any assignment for $w_i$.

FOR every clause $S$:

if $S$ is violated, Fix($S$) to satisfy $S$.

The correctness of Fix assuming it terminates is trivial (induction: if recursive calls are correct, the calling procedure is correct).

The only problem is why Fix($S$) terminates in reasonable time with high probability.
Resampling algorithm

- Main algorithm:
  start with any assignment for $w$;
  FOR every clause $S$:
    if $S$ is violated, $\text{Fix}(S)$
Resampling algorithm

- Main algorithm:
  start with any assignment for \( w \);
  FOR every clause \( S \):
    if \( S \) is violated, \( \text{Fix}(S) \)

\[
\begin{align*}
\{ & \text{ \( S \) is violated} \} \\
\text{Fix}(S) & \\
\{ & \text{ \( S \) is satisfied; no other previously satisfied clauses are violated} \}
\end{align*}
\]
Resampling algorithm

- **Main algorithm:**
  start with any assignment for \( w \);
  FOR every clause \( S \):
    if \( S \) is violated, \( \text{Fix}(S) \)
  \{ \( S \) is violated \}
  \( \text{Fix}(S) \)
  \{ \( S \) is satisfied; no other previously satisfied clauses are violated \}

- \( \text{Fix}(S) \):
  resample \( (S) \);
  FOR every neighbor clause \( S' \):
    if \( S' \) is violated, \( \text{Fix}(S') \)
Resampling algorithm

- Main algorithm:
  - start with any assignment for $w$
  - FOR every clause $S$
    - if $S$ is violated, $Fix(S)$
  

- $\{ S$ is violated $\}$
  $Fix(S)$
  $\{ S$ is satisfied; no other previously satisfied clauses are violated $\}$

- $Fix(S)$:
  - resample (S);
  - FOR every neighbor clause $S'$:
    - if $S'$ is violated, $Fix(S')$

- The correctness of $Fix$ assuming it terminates is trivial (induction: if recursive calls are correct, the calling procedure is correct)
Resampling algorithm

- Main algorithm:
  start with any assignment for $w$;
  FOR every clause $S$:
    if $S$ is violated, $Fix(S)$

  $\{ S \text{ is violated } \}
  Fix(S)
  \{ S \text{ is satisfied; no other previously satisfied clauses are violated } \}

- $Fix(S)$:
  resample $(S)$;
  FOR every neighbor clause $S'$:
    if $S'$ is violated, $Fix(S')$

- The correctness of $Fix$ assuming it terminates is trivial
  (induction: if recursive calls are correct, the calling procedure is correct)

- The only problem is why $Fix(S)$ terminates in reasonable time with high probability.
Resampling: analysis

We show only that \( \text{Fix}(S) \) terminates at some point if we use fresh bits from an incompressible string for resampling (and do not translate this argument into a probabilistic language with exact bounds).

Bit source: a long incompressible bit string split into \( m \)-bit blocks.

When resampling is needed, the next block is used.

Main observation: random bits can be reconstructed from the current values and the (chronological) list of resampled clauses.

Indeed, each clause is violated only for one combination of bits, so resampling can be “undone” and random bits can be extracted.

So if we can describe the sequence of resampled clauses using less than \( m \) bits per clause, we get a contradiction (\( N \) is fixed and for the large number of steps we get a contradiction with incompressibility).
Resampling: analysis

- We show only that $\text{Fix}(S)$ terminates at some point if we use fresh bits from an incompressible string for resampling (and do not translate this argument into a probabilistic language with exact bounds)
Resampling: analysis

- We show only that $\text{Fix}(S)$ terminates at some point if we use fresh bits from an incompressible string for resampling (and do not translate this argument into a probabilistic language with exact bounds).

- Bit source: a long incompressible bit string split into $m$-bit blocks.
Resampling: analysis

- We show only that $\text{Fix}(S)$ terminates at some point if we use fresh bits from an incompressible string for resampling (and do not translate this argument into a probabilistic language with exact bounds).
- Bit source: a long incompressible bit string split into $m$-bit blocks.
- When resampling is needed, next block is used.
Resampling: analysis

- We show only that $\text{Fix}(S)$ terminates at some point if we use fresh bits from an incompressible string for resampling (and do not translate this argument into a probabilistic language with exact bounds)
- Bit source: a long incompressible bit string split into $m$-bit blocks
- When resampling is needed, next block is used
- Main observation: random bits can be reconstructed from the current values and the (chronological) list of resampled clauses
Resampling: analysis

- We show only that $\text{Fix}(S)$ terminates at some point if we use fresh bits from an incompressible string for resampling (and do not translate this argument into a probabilistic language with exact bounds).

- Bit source: a long incompressible bit string split into $m$-bit blocks.

- When resampling is needed, next block is used.

- Main observation: random bits can be reconstructed from the current values and the (chronological) list of resampled clauses.

- Indeed, each clause is violated only for one combination of bits, so resampling can be “undone” and random bits can be extracted.
Resampling: analysis

- We show only that $\text{Fix}(S)$ terminates at some point if we use fresh bits from an incompressible string for resampling (and do not translate this argument into a probabilistic language with exact bounds).
- Bit source: a long incompressible bit string split into $m$-bit blocks.
- When resampling is needed, next block is used.
- Main observation: random bits can be reconstructed from the current values and the (chronological) list of resampled clauses.
- Indeed, each clause is violated only for one combination of bits, so resampling can be “undone” and random bits can be extracted.
- So if we can describe the sequence of resampled clauses using less than $m$ bits per clause, we get a contradiction ($N$ is fixed and for the large number of steps we get a contradiction with incompressibility).
Describing the resampled clauses

▶ Each clause is one of \( t \) neighbors of a previous one, so we need at most \( \log t \) bits to specify which one.

▶ The lists of neighbors can be fixed in advance

▶ Indeed a simplification: though in the tree of recursive calls each vertex is a neighbor of its father, this is not enough: we also go up (when exiting a recursive call)

▶ so we need an additional bit (up/down) for recursive call (down step) and one bit to describe exits (up steps)

▶ so instead of \( \log t \) per resampling we get \( (\log t + 1) + 1 \) – still less than \( m \) by assumption (\( \log t < m - 3 \))
Describing the resampled clauses

- (Simplified) Each clause is one of \( t \) neighbors of a previous one, so we need at most \( \log t \) bits to specify which one.
Describing the resampled clauses

- (Simplified) Each clause is one of $t$ neighbors of a previous one, so we need at most $\log t$ bits to specify which one.
- The lists of neighbors can be fixed in advance.
Describing the resampled clauses

- (Simplified) Each clause is one of $t$ neighbors of a previous one, so we need at most $\log t$ bits to specify which one.
- The lists of neighbors can be fixed in advance.
- Indeed a simplification: though in the tree of recursive calls each vertex is a neighbor of its father, this is not enough: we also go up (when exiting a recursive call).
Describing the resampled clauses

- (Simplified) Each clause is one of \( t \) neighbors of a previous one, so we need at most \( \log t \) bits to specify which one.
- The lists of neighbors can be fixed in advance
- Indeed a simplification: though in the tree of recursive calls each vertex is a neighbor of its father, this is not enough: we also go up (when exiting a recursive call)
- so we need an additional bit (up/down) for recursive call (down step) and one bit to describe exits (up steps)
Describing the resampled clauses

- (Simplified) Each clause is one of $t$ neighbors of a previous one, so we need at most $\log t$ bits to specify which one.
- The lists of neighbors can be fixed in advance.
- Indeed a simplification: though in the tree of recursive calls each vertex is a neighbor of its father, this is not enough: we also go up (when exiting a recursive call).
- So we need an additional bit (up/down) for recursive call (down step) and one bit to describe exits (up steps).
- So instead of $\log t$ per resampling we get $(\log t + 1) + 1$ – still less than $m$ by assumption ($\log t < m - 3$).