Randomized Algorithms

Local dependence and Moser–Tardos

Alex Gittens
Department of Computer Science, Rensselaer Polytechnic Institute

Fall 2026

PART 1

Avoiding many bad events

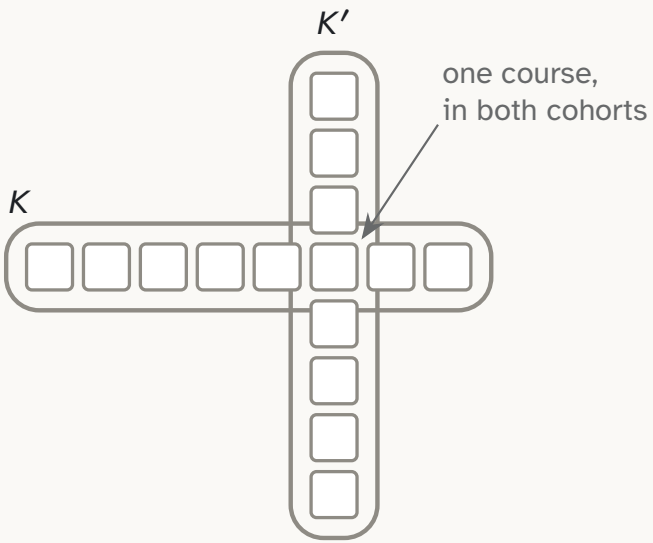
1

You are the RPI registrar

Final exams run on two days, **Monday** and **Thursday**. Each course holds one exam, on one of the two days.

A *cohort* is a group of students who all take the same 8 courses—say, the second-year students in one major. If all 8 of a cohort's exams land on the same day, those students sit 8 exams in one day.

Cohorts overlap: a course is taken by several cohorts, so one course's day affects several cohorts at once.

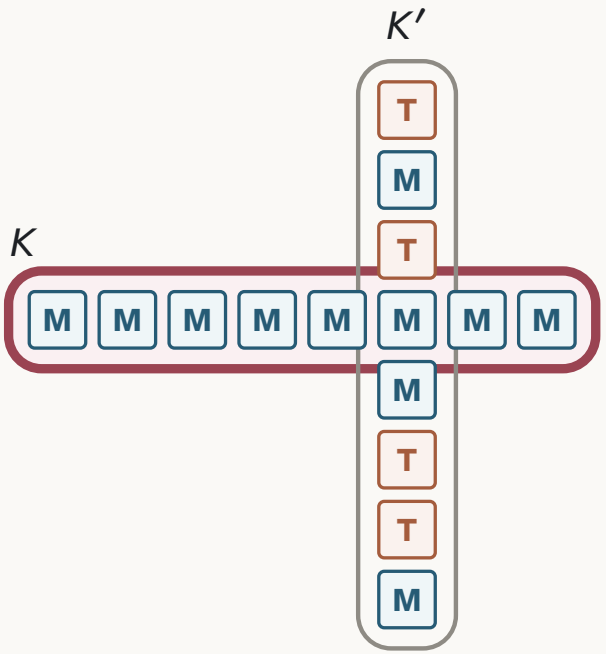


Goal: choose days so that *no cohort has all its exams on one day*.

A random exam schedule

For each course c, independently flip a fair coin to choose its exam day $X_c \in \{\mathrm{M}, \mathrm{T}\}$. For each cohort K, the undesirable event is

$$A_K = \{\text{all 8 courses of } K \text{ get the same day}\}.$$



A_K **occurs**: all of K is on Monday, so K is *stuck* (thick outline).

$A_{K'}$ does not occur: K' has both days.

Exam scheduling instance

1,600 courses and $m = 600$ cohorts. Each cohort has exactly 8 courses; each course belongs to exactly 3 cohorts.

How likely is one cohort to be stuck?

A cohort K has 8 courses, and their days are independent fair coins. A_K happens if all 8 are Monday or all 8 are Thursday:

$$\begin{aligned}\Pr(A_K) &= \Pr(\text{all M}) + \Pr(\text{all T}) \\ &= 2 \cdot 2^{-8} = 2^{-7} = \tfrac{1}{128} \approx 0.0078.\end{aligned}$$

A particular cohort has only a $1/128$ chance of being stuck. But there are 600 cohorts.

Question: is this small enough to ensure that no cohort is stuck?

Can the union bound guarantee a schedule?

For our 600 cohorts, we want to show that a random schedule has positive probability of avoiding every bad event. More generally, for events $A_1, \ldots, A_m$ we want

$$\Pr\left(\bigcap_{i=1}^{m} \overline{A_i}\right) > 0.$$

What can we say using the union bound?

Can the union bound guarantee a schedule?

For our 600 cohorts, we want to show that a random schedule has positive probability of avoiding every bad event. More generally, for events $A_1, \ldots, A_m$ we want

$$\Pr\left(\bigcap_{i=1}^{m} \overline{A_i}\right) > 0.$$

What can we say using the union bound?

$$\Pr\left(\bigcap_{i=1}^{m} \overline{A_i}\right) = 1 - \Pr\left(\bigcup_{i=1}^{m} A_i\right)$$

Can the union bound guarantee a schedule?

For our 600 cohorts, we want to show that a random schedule has positive probability of avoiding every bad event. More generally, for events $A_1, \ldots, A_m$ we want

$$\Pr\left(\bigcap_{i=1}^{m} \overline{A_i}\right) > 0.$$

What can we say using the union bound?

$$\begin{aligned}\Pr\left(\bigcap_{i=1}^{m} \overline{A_i}\right) &= 1 - \Pr\left(\bigcup_{i=1}^{m} A_i\right) \\ &\geq 1 - \sum_{i=1}^{m} \Pr(A_i).\end{aligned}$$

Can the union bound guarantee a schedule?

For our 600 cohorts, we want to show that a random schedule has positive probability of avoiding every bad event. More generally, for events $A_1, \ldots, A_m$ we want

$$\Pr\left(\bigcap_{i=1}^{m} \overline{A_i}\right) > 0.$$

What can we say using the union bound?

$$\begin{aligned}\Pr\left(\bigcap_{i=1}^{m} \overline{A_i}\right) &= 1 - \Pr\left(\bigcup_{i=1}^{m} A_i\right) \\ &\geq 1 - \sum_{i=1}^{m} \Pr(A_i).\end{aligned}$$

The union bound proves existence if $\sum_i \Pr(A_i) < 1$. Does it work for our 600 cohorts?

Applying the union bound to exam scheduling

Write $p = \Pr(A_K) = 1/128$. There are $m = 600$ cohorts.

Applying the union bound to exam scheduling

Write $p = \Pr(A_K) = 1/128$. There are $m = 600$ cohorts. The union bound gives

$$\Pr\left(\bigcap_K \overline{A_K}\right) \geq 1 - \sum_K \Pr(A_K) = 1 - \frac{600}{128} \approx -3.69.$$

A negative lower bound tells us nothing.

Applying the union bound to exam scheduling

Write $p = \Pr(A_K) = 1/128$. There are $m = 600$ cohorts. The union bound gives

$$\Pr\left(\bigcap_K \overline{A_K}\right) \geq 1 - \sum_K \Pr(A_K) = 1 - \frac{600}{128} \approx -3.69.$$

A negative lower bound tells us nothing.
Can we improve the calculation by accounting for overlapping bad events?

What does inclusion–exclusion give us?

Accounting for overlapping events

The union bound subtracts the probability of each bad event. But when two events overlap, we've subtracted their intersection twice.
For two events, inclusion–exclusion corrects this:

$$\Pr(\overline{A_1} \cap \overline{A_2}) = 1 - \Pr(A_1) - \Pr(A_2) + \Pr(A_1 \cap A_2).$$

Accounting for overlapping events

The union bound subtracts the probability of each bad event. But when two events overlap, we've subtracted their intersection twice.
For two events, inclusion–exclusion corrects this:

$$\Pr(\overline{A_1} \cap \overline{A_2}) = 1 - \Pr(A_1) - \Pr(A_2) + \Pr(A_1 \cap A_2).$$

For m events, the principle of inclusion–exclusion (PIE) gives

$$\Pr\Big(\bigcap_{i=1}^{m} \overline{A_i}\Big) = 1 - \sum_{i} \Pr(A_i) + \sum_{i<j} \Pr(A_i \cap A_j) - \sum_{i<j<k} \Pr(A_i \cap A_j \cap A_k) + \cdots.$$

Accounting for overlapping events

The union bound subtracts the probability of each bad event. But when two events overlap, we've subtracted their intersection twice.
For two events, inclusion–exclusion corrects this:

$$\Pr(\overline{A_1} \cap \overline{A_2}) = 1 - \Pr(A_1) - \Pr(A_2) + \Pr(A_1 \cap A_2).$$

For m events, the principle of inclusion–exclusion (PIE) gives

$$\Pr\Big(\bigcap_{i=1}^{m} \overline{A_i}\Big) = 1 - \sum_{i} \Pr(A_i) + \sum_{i<j} \Pr(A_i \cap A_j) - \sum_{i<j<k} \Pr(A_i \cap A_j \cap A_k) + \cdots.$$

PIE makes no independence assumptions, but requires up to $2^m - 1$ intersection probabilities. Even for a moderate number of events, this is difficult to use.

If the events are independent

Suppose all m bad events were mutually independent. What would replace the union bound?

If the events are independent

Suppose all m bad events were mutually independent. What would replace the union bound?

$$\Pr\left(\bigcap_{i=1}^{m} \overline{A_i}\right) = \prod_{i=1}^{m} \Pr(\overline{A_i})$$
$$= \prod_{i=1}^{m} (1 - \Pr(A_i)).$$

If the events are independent

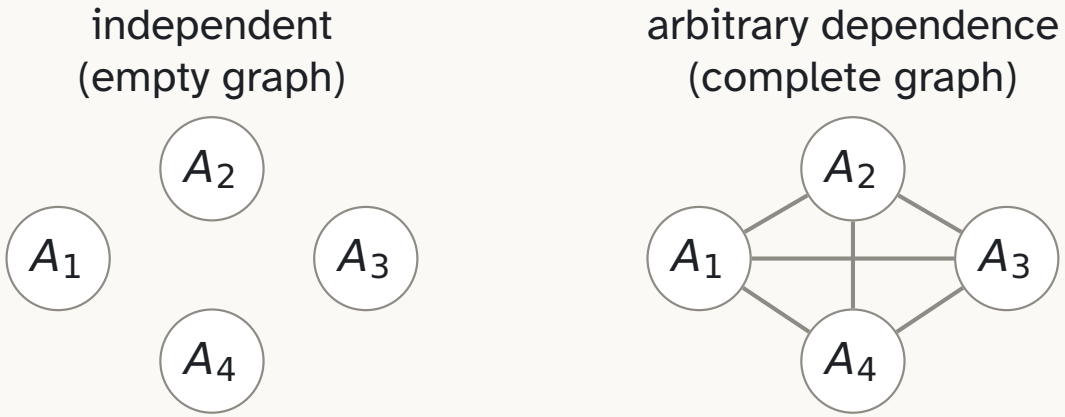
Suppose all m bad events were mutually independent. What would replace the union bound?

$$\Pr\left(\bigcap_{i=1}^{m} \overline{A_i}\right) = \prod_{i=1}^{m} \Pr(\overline{A_i})$$
$$= \prod_{i=1}^{m} (1 - \Pr(A_i)).$$

For finitely many events, this is positive whenever each $\Pr(A_i) < 1$—even if their sum is much larger than 1. For our cohorts it would give $(1 - \frac{1}{128})^{600} > 0$.

The cohort events need not be *mutually independent*: overlapping cohorts are determined by some of the same random choices.

Limited dependencies can help

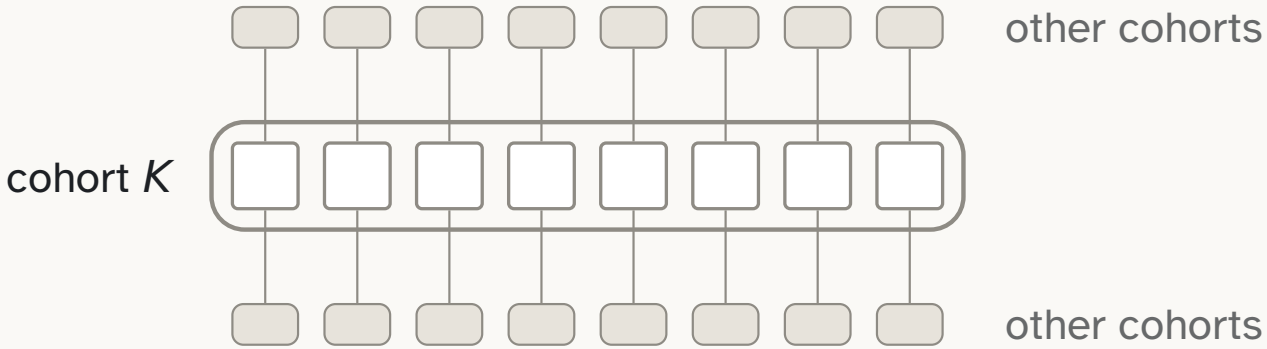


Cohort events sit between these extremes: cohorts that share a course interact, while cohorts with no course in common do not.

Let's represent these possible dependencies by a graph.

The cohort dependency graph

A_K is determined by the coins $\{X_c : c \in K\}$. Join K and K' if they share a course. If K shares no course with any cohort in a collection, A_K uses coins disjoint from all of theirs, so it is jointly independent of their events.



Each of K's 8 courses belongs to K and at most 2 other cohorts. So K shares a course with at most

$$\boxed{8 \cdot 2 = 16}$$

other cohorts. (Fewer if two courses of K lead to the same cohort.)

Dependency graph

Dependency graph

A graph G on $\{1, \ldots, m\}$ is a dependency graph for $A_1, \ldots, A_m$ if, for every i, A_i is *jointly independent* of the collection of events indexed by nonneighbors of i.

That means A_i is independent of any event determined by its nonneighbors taken together. Pairwise independence from each nonneighbor alone is insufficient.

Write $\Gamma(i)$ for the neighbors of i, and let

$$d = \max_i |\Gamma(i)|.$$

An edge allows dependence; it does not assert that two events are dependent.

Lovász Local Lemma: symmetric form

Lovász Local Lemma (Erdős–Lovász 1975; Spencer 1977)

Let $A_1, \dots, A_m$ have a dependency graph of maximum degree at most d and suppose $\Pr(A_i) \le p$ for all i. If

$$\boxed{e\,p(d+1) \le 1},$$

then

$$\Pr\left(\bigcap_{i=1}^{m} \overline{A_i}\right) > 0.$$

Notice the contrast: the union bound would need $mp < 1$. The *local* lemma only needs $p(d+1) \le e^{-1}$. **What matters is how many other events any one event can interact with, not the total number of events.**

[0] The appendix gives an induction proof.

Exam scheduling with the LLL

For the exam instance,

$$m = 600, \qquad p = \tfrac{1}{128}, \qquad d \le 16.$$

The union bound gave $600/128 \approx 4.69 > 1$. But

$$e\,p(d+1) \le \frac{17e}{128} \approx 0.36 < 1.$$

Therefore

$$\Pr(\text{no cohort has all its exams on one day}) > 0.$$

Some exam schedule leaves no cohort stuck.

A second application: SAT

A_{C_i} is the event that clause C_i is false under a uniformly random truth assignment.

$$C_1 = (x_1 \vee x_2), \qquad C_2 = (\neg x_2 \vee x_3), \qquad C_3 = (\neg x_3 \vee x_4).$$

Which pairs of clause events share a variable? Draw the dependency graph.

A second application: SAT

A_{C_i} is the event that clause C_i is false under a uniformly random truth assignment.

$$C_1 = (x_1 \vee x_2), \qquad C_2 = (\neg x_2 \vee x_3), \qquad C_3 = (\neg x_3 \vee x_4).$$

Which pairs of clause events share a variable? Draw the dependency graph.



C_1 and C_3 share no variable, so A_{C_1} and A_{C_3} are not joined. As with cohorts, the graph connects events determined by a common random variable.

SAT: what does the LLL add?

For a k-CNF formula with m clauses, a uniformly random truth assignment falsifies any fixed clause with probability 2^{-k}.
The union bound proves satisfiability if

$$m\,2^{-k} < 1.$$

Suppose instead that each variable occurs in at most L clauses. A clause shares variables with at most $k(L-1)$ others.
The LLL gives the condition

$$\boxed{e\,2^{-k}\big(k(L-1)+1\big) \le 1.}$$

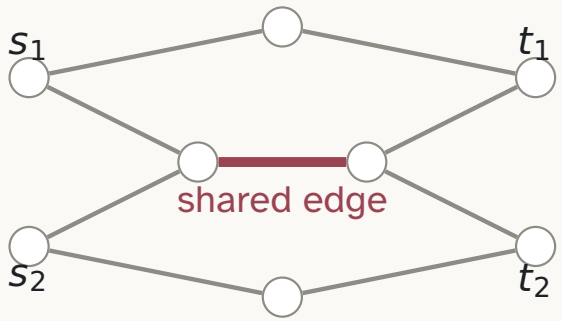
Compare: The union bound restricts the total number of clauses; the LLL restricts how often any variable appears.

Exercise: routing without conflicts

A network must carry $n \geq 3$ connections. Each connection i independently picks a random path P_i from its own list of candidate paths. Let

$$A_{ij} = \{P_i \text{ and } P_j \text{ share an edge}\}, \qquad i < j.$$

Assume every candidate path has at most ℓ edges, and no network edge is too popular: $\Pr(e \in P_j) \leq \rho$ for every edge e and connection j.



Two candidate routes may share an edge; alternatives might avoid the conflict.

1. Which random variables determine A_{ij}?
2. How many other conflict events share a variable with A_{ij}?
3. Bound $\Pr(A_{ij})$ in terms of ℓ and ρ. *Hint:* condition on P_i.
4. State the LLL condition that guarantees conflict-free routes.

Solution: Appendix C

PART 2

Local resampling

2

Bad events determined by independent variables

In the exam example, the outcome is a list of independent coin flips

$$(X_c)_{c \text{ a course}}, \qquad X_c \in \{\mathrm{M}, \mathrm{T}\}.$$

The event for cohort K,

$$A_K = \{X_c \text{ is the same for all } c \in K\},$$

depends only on the 8 coins $\{X_c : c \in K\}$.

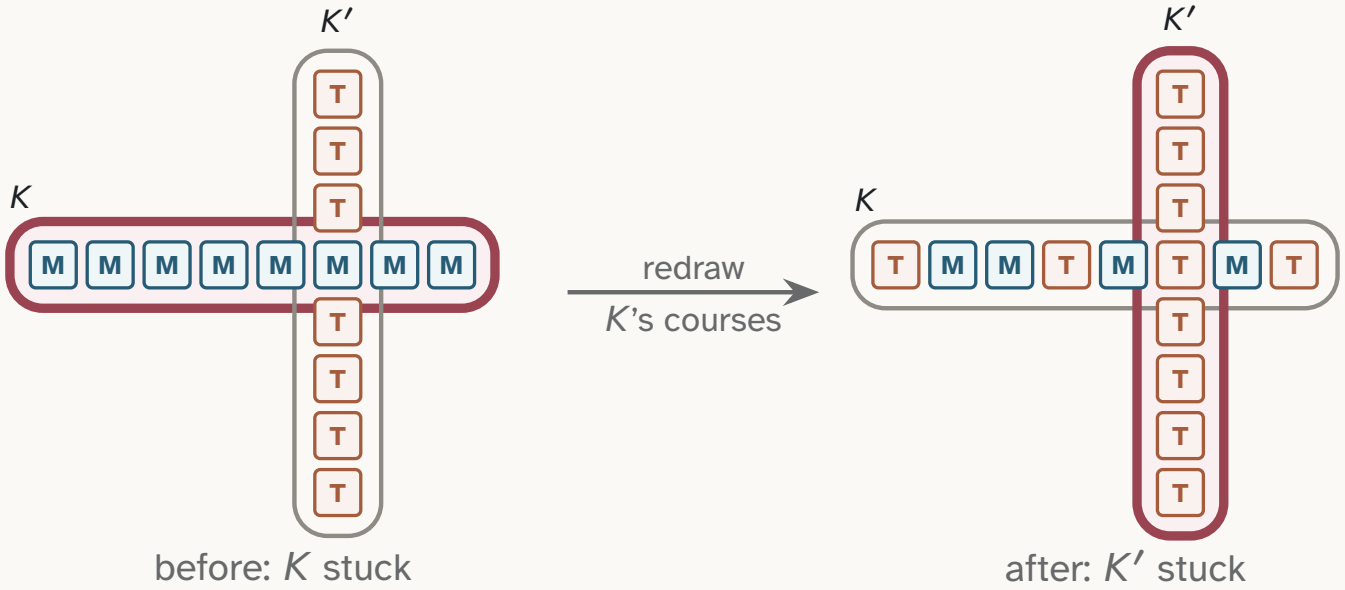
More generally, let $X_1, \ldots, X_n$ be independent variables and let each bad event A_i be determined by some subset

$$\mathrm{vbl}(A_i) \subseteq \{X_1, \ldots, X_n\}.$$

Joining two events that share variables gives a dependency graph.

Local repair

The LLL shows that a schedule with no stuck cohort exists. Can we *find* one? If cohort K is stuck, the culprit is the days of its 8 courses. Redraw those 8 coins and leave every other course alone.



The repair can backfire: the shared course moved to Thursday, and now K' is all Thursday. Recall our earlier alteration arguments: start with a random object and remove violations. Here we try something different: redraw the variables responsible for a violation. But a repair may create new violations.

Moser–Tardos resampling algorithm (2010)

For an event A_i, let $\mathrm{vbl}(A_i)$ be the independent variables that determine it.

Local resampling

1. Draw all the independent variables once.
2. While some bad event A_i occurs, select one such event and independently redraw every variable in $\mathrm{vbl}(A_i)$.
3. Stop when no bad event occurs.

Variables outside $\mathrm{vbl}(A_i)$ do not change. The selection rule among violated events can be arbitrary.

Will repeated repairs eventually stop? How many rounds will they take?

Moser–Tardos theorem (2010)

Let $A_1, \ldots, A_m$ be determined by independent random variables, each with probability at most p. Suppose each event shares variables with at most d others, where $d \geq 1$.
If

$$\boxed{ep(d+1) \leq 1},$$

the resampling algorithm terminates almost surely with an assignment avoiding all bad events.

Let N_i be the number of iterations in which A_i is selected, and let $R = \sum_i N_i$ be the number of iterations of the resampling algorithm. Then

$$\mathbb{E}[N_i] \leq \frac{1}{d}, \qquad \mathbb{E}[R] \leq \frac{m}{d}.$$

LLL applications: existence and construction

We have applied the LLL to exam scheduling, SAT, and routing. In each case, the bad events are determined by independent random choices, so we can resample violations.

Problem	What is resampled?	Expected resampling steps
Exam days	Days of the 8 courses of a stuck cohort	
SAT	Boolean variables of a false clause	
Routing	The two paths in conflict	

LLL applications: existence and construction

We have applied the LLL to exam scheduling, SAT, and routing. In each case, the bad events are determined by independent random choices, so we can resample violations.

Problem	What is resampled?	Expected resampling steps
Exam days	Days of the 8 courses of a stuck cohort	$\mathbb{E}[R] \le 600/16 = 37.5$
SAT	Boolean variables of a false clause	
Routing	The two paths in conflict	

LLL applications: existence and construction

We have applied the LLL to exam scheduling, SAT, and routing. In each case, the bad events are determined by independent random choices, so we can resample violations.

Problem	What is resampled?	Expected resampling steps
Exam days	Days of the 8 courses of a stuck cohort	$\mathbb{E}[R] \le 600/16 = 37.5$
SAT	Boolean variables of a false clause	$\mathbb{E}[R] \le m/[k(L-1)]$
Routing	The two paths in conflict	

LLL applications: existence and construction

We have applied the LLL to exam scheduling, SAT, and routing. In each case, the bad events are determined by independent random choices, so we can resample violations.

Problem	What is resampled?	Expected resampling steps
Exam days	Days of the 8 courses of a stuck cohort	$\mathbb{E}[R] \le 600/16 = 37.5$
SAT	Boolean variables of a false clause	$\mathbb{E}[R] \le m/[k(L-1)]$
Routing	The two paths in conflict	$\mathbb{E}[R] \le \binom{n}{2}/(2n-4)$

LLL applications: existence and construction

We have applied the LLL to exam scheduling, SAT, and routing. In each case, the bad events are determined by independent random choices, so we can resample violations.

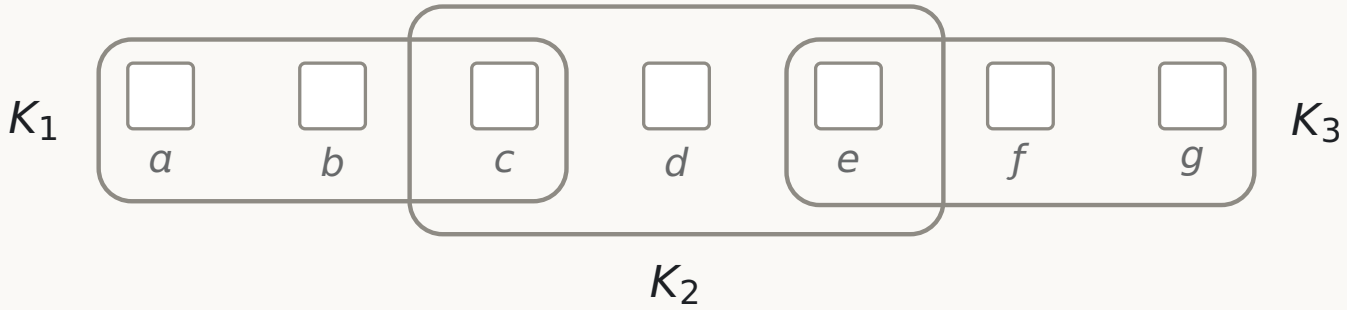
Problem	What is resampled?	Expected resampling steps
Exam days	Days of the 8 courses of a stuck cohort	$\mathbb{E}[R] \le 600/16 = 37.5$
SAT	Boolean variables of a false clause	$\mathbb{E}[R] \le m/[k(L-1)]$
Routing	The two paths in conflict	$\mathbb{E}[R] \le \binom{n}{2}/(2n-4)$

For exams, 37.5 bounds the expected number of *repairs*, including repairs of cohorts that earlier repairs broke. The expected number initially stuck is $600/128 \approx 4.7$.

A scheduling example

To watch the algorithm in action, shrink the instance: 7 courses $a, \dots, g$ and three cohorts of 3 courses each,

$$K_1 = \{a, b, c\}, \qquad K_2 = \{c, d, e\}, \qquad K_3 = \{e, f, g\}.$$

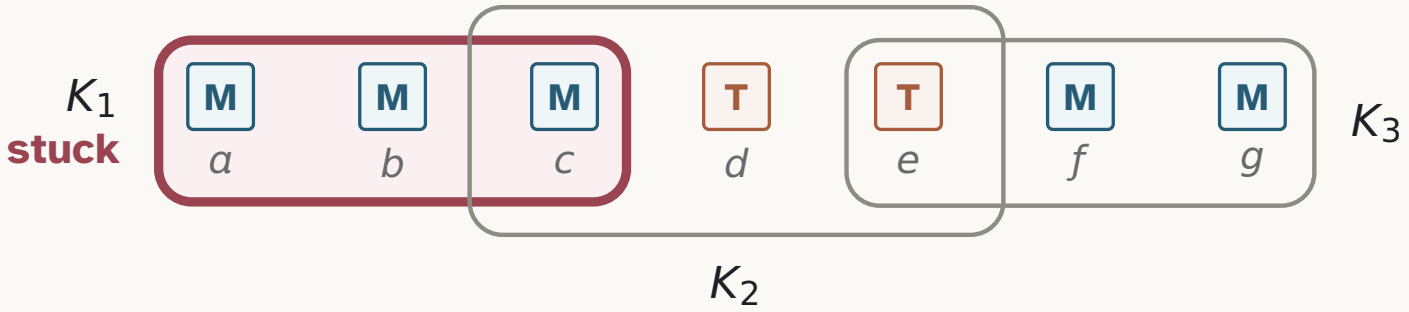


Let A_j be the event that K_j is all one day; $\Pr(A_j) = 2 \cdot 2^{-3} = \frac{1}{4}$.



Dependency graph: K_1, K_2 share c; K_2, K_3 share e. This toy is too small to meet the LLL condition; we only use it to see the mechanics.

Tracing the resampling algorithm

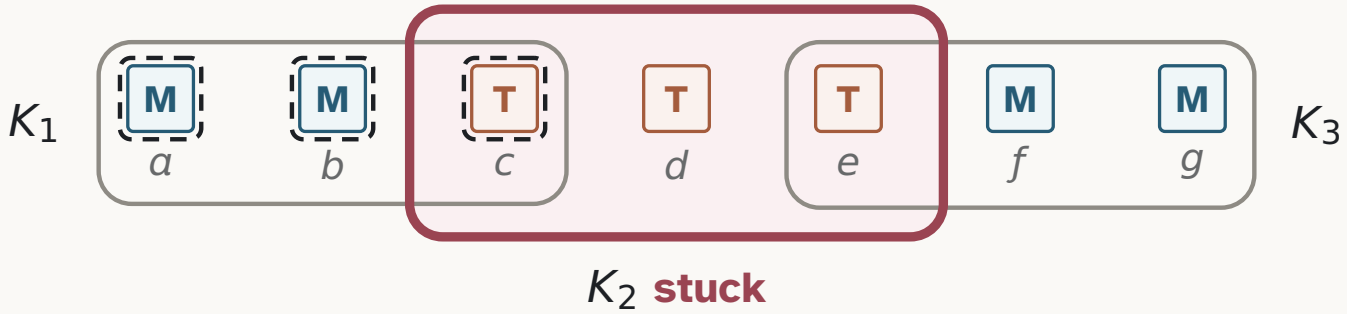


M = Monday, T = Thursday. Stuck cohorts are outlined thick. Dashed squares mark the courses just redrawn.

step	resampled	stuck afterward
initial	—	K_1

Initial draw. Only K_1 is all one day. Which coins should we redraw?

Tracing the resampling algorithm

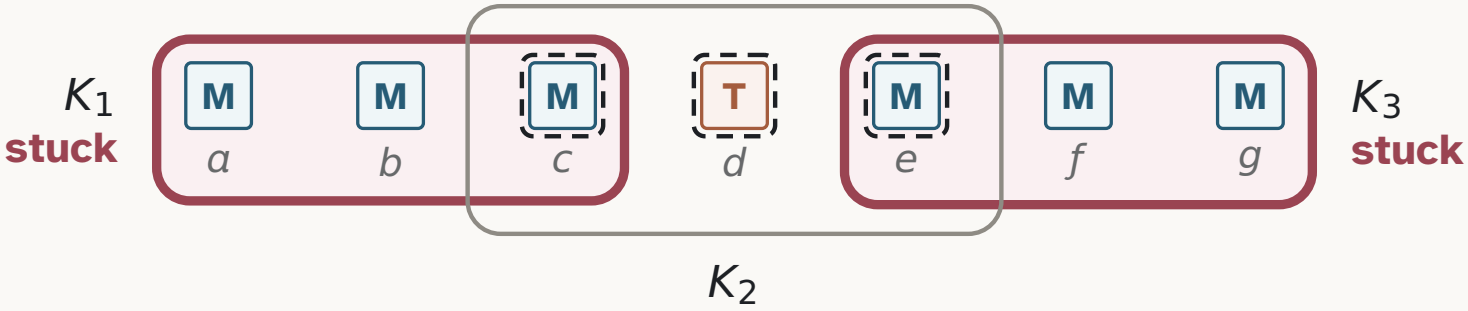


M = Monday, T = Thursday. Stuck cohorts are outlined thick. Dashed squares mark the courses just redrawn.

step	resampled	stuck afterward
initial	—	K_1
1	A_1	K_2

Step 1: redraw A_1 (courses a, b, c). K_1 is repaired, but c moved to Thursday, so now $K_2 = (\mathrm{T}, \mathrm{T}, \mathrm{T})$ is stuck.

Tracing the resampling algorithm

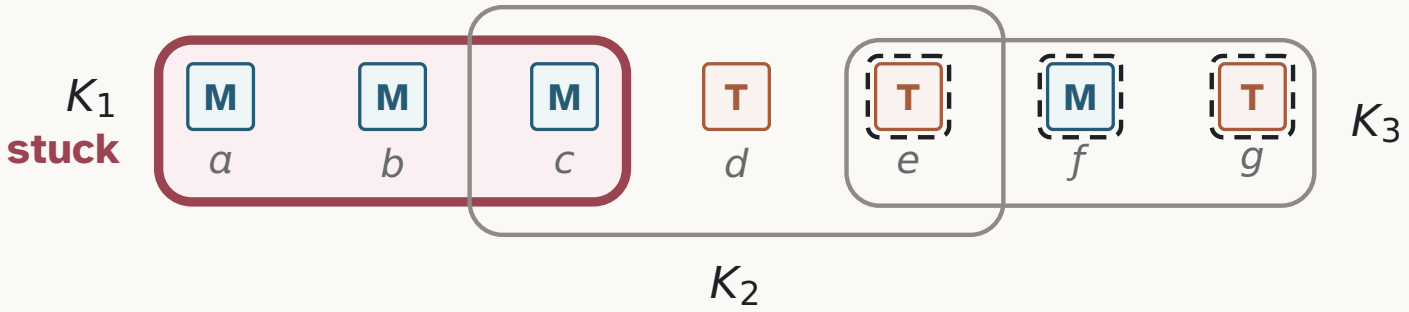


M = Monday, T = Thursday. Stuck cohorts are outlined thick. Dashed squares mark the courses just redrawn.

step	resampled	stuck afterward
initial	—	K_1
1	A_1	K_2
2	A_2	K_1, K_3

Step 2: redraw A_2 (courses c, d, e). K_2 is repaired, but changing c breaks K_1 and changing e breaks K_3. **One repair has created two violations.**

Tracing the resampling algorithm

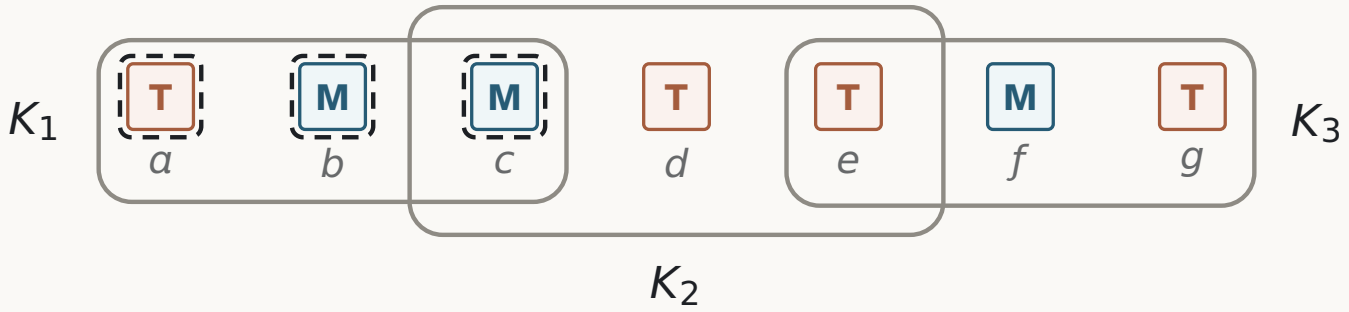


M = Monday, T = Thursday. Stuck cohorts are outlined thick. Dashed squares mark the courses just redrawn.

step	resampled	stuck afterward
initial	—	K_1
1	A_1	K_2
2	A_2	K_1, K_3
3	A_3	K_1

Step 3: redraw A_3 (courses e, f, g). K_3 is repaired; K_1 is still stuck.

Tracing the resampling algorithm



M = Monday, T = Thursday. Stuck cohorts are outlined thick. Dashed squares mark the courses just redrawn.

step	resampled	stuck afterward
initial	—	K_1
1	A_1	K_2
2	A_2	K_1, K_3
3	A_3	K_1
4	A_1	none

Step 4: redraw A_1 (courses a, b, c). Every cohort now has both days, so the algorithm stops.

Counting current violations doesn’t work

Every resampling gives the violated event a fresh chance to disappear. Could we argue that the number of violations therefore decreases?

Counting current violations doesn’t work

Every resampling gives the violated event a fresh chance to disappear. Could we argue that the number of violations therefore decreases?

- **One repair can create more violations.** In step 2 of our trace, one stuck cohort became two.
- **The algorithm adapts.** Which event it resamples next depends on the random values it has already seen.

Counting current violations doesn't work

Every resampling gives the violated event a fresh chance to disappear. Could we argue that the number of violations therefore decreases?

- **One repair can create more violations.** In step 2 of our trace, one stuck cohort became two.
- **The algorithm adapts.** Which event it resamples next depends on the random values it has already seen.

How can we account for this dependence when bounding how often the same event is resampled?

Bounding the number of resamplings

Recall that N_i counts the resamplings of A_i. We want to bound $\mathbb{E}[N_i]$.

Idea: Associate each resampling of A_i with a witness recording the earlier resamplings relevant to the violation.

What should these witnesses satisfy?

1. **Distinctness:** Different resamplings of A_i give different witnesses.
2. **Probability:** A witness with t recorded violations occurs with probability at most p^t.
3. **Counting:** We can bound the sum of these probabilities over all possible witnesses.

How can we construct witnesses with these properties?

PART 3

Local resampling terminates

3

The resampling table

Imagine drawing all the coin flips in advance, arranging successive draws for each course in a row. For course c write

$$c^{(0)},\ c^{(1)},\ c^{(2)},\ \ldots$$

The algorithm first uses $c^{(0)}$. Each time it redraws c, it moves one cell to the right. All cells, in all rows, are independent. Drawing a fresh coin when needed or reading the next unused cell gives the same algorithm.

The cells read during the first two steps of our trace:

course	(0)	(1)	(2)
a	M	M	
b	M	M	
c	M	T	M
d	T	T	
e	T	M	

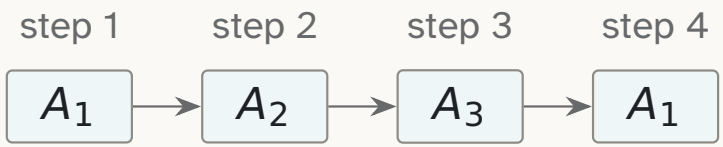
Step 1 redrew *a, b, c*; step 2 redrew *c, d, e*. So *c* has used three cells.

The resampling history

In any execution, write B_t for the bad event resampled at step t. By the algorithm, B_t must be violated immediately before that step.
In our trace, the four resampled events were

$$(B_1, B_2, B_3, B_4) = (A_1, A_2, A_3, A_1).$$

Resampled events (in time order)



At step 4, A_1 is violated again. Which random draws of a, b, c were responsible? The answer depends on the earlier resamplings that touched these courses.

Constructing the tree for step 4

To determine which draw of course c was tested at step 4, we need to know how many times c was previously redrawn. Begin with $B_4 = A_1$ and work backward:

$$(B_1, B_2, B_3, B_4) = (A_1, A_2, A_3, A_1).$$





dependency graph

Put the event resampled at step 4 at the root.

Constructing the tree for step 4

To determine which draw of course c was tested at step 4, we need to know how many times c was previously redrawn. Begin with $B_4 = A_1$ and work backward:

$$(B_1, B_2, B_3, B_4) = (A_1, A_2, A_3, A_1).$$





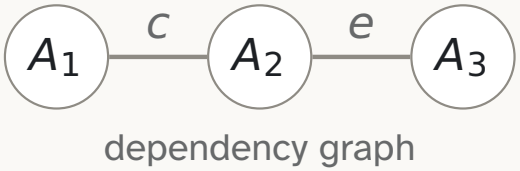
dependency graph

Step 3 resampled A_3 (courses e, f, g). It shares no course with the root's a, b, c, so omit it.

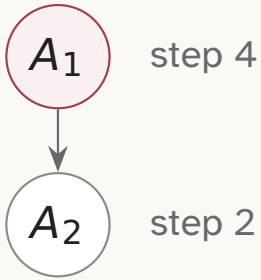
Constructing the tree for step 4

To determine which draw of course c was tested at step 4, we need to know how many times c was previously redrawn. Begin with $B_4 = A_1$ and work backward:

$$(B_1, B_2, B_3, B_4) = (A_1, A_2, A_3, A_1).$$



dependency graph

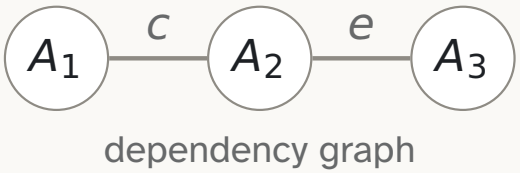


Step 2 resampled A_2, which shares course c with the root. Add A_2 as a child.

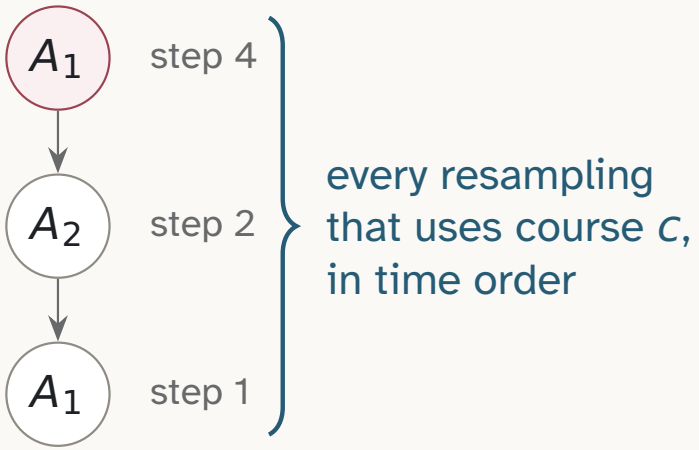
Constructing the tree for step 4

To determine which draw of course c was tested at step 4, we need to know how many times c was previously redrawn. Begin with $B_4 = A_1$ and work backward:

$$(B_1, B_2, B_3, B_4) = (A_1, A_2, A_3, A_1).$$



dependency graph



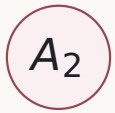
Step 1 resampled A_1, which shares courses with both retained steps. Attach it below the *deeper* one, A_2. The three resamplings that used course c now lie on one chain, in time order. We can count earlier redraws of c by looking below each node.

Witness trees are not always paths

Our four-step example gave a path. Consider instead the resampled-event sequence

$$(A_1, A_3, A_2)$$

with the same dependency graph $A_1 - A_2 - A_3$.
Build the tree for the third resampling. Which earlier events attach below A_2?

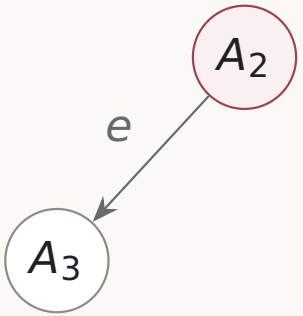


Witness trees are not always paths

Our four-step example gave a path. Consider instead the resampled-event sequence

$$(A_1, A_3, A_2)$$

with the same dependency graph $A_1 - A_2 - A_3$.
Build the tree for the third resampling. Which earlier events attach below A_2?

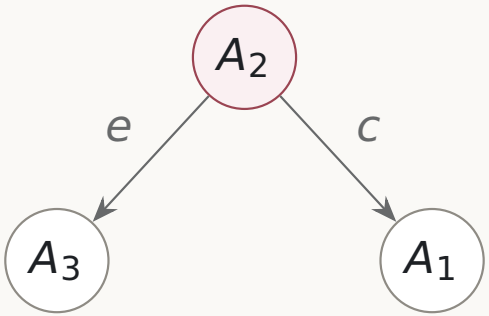


Witness trees are not always paths

Our four-step example gave a path. Consider instead the resampled-event sequence

$$(A_1, A_3, A_2)$$

with the same dependency graph $A_1 - A_2 - A_3$.
Build the tree for the third resampling. Which earlier events attach below A_2?



A_3 shares course e with A_2, and A_1 shares course c with A_2, but A_1 and A_3 share nothing. The resulting tree *branches*.

Constructing a witness tree

Fix an execution and a step t. The events resampled through step t are $B_1, \ldots, B_t$.

1. Start a tree W_t with a root labeled B_t.
2. Examine steps $t-1, t-2, \ldots, 1$ in that order.
3. Include the resampling of B_s if it uses a variable used by some resampling already represented. Attach it below a *deepest* such node; break ties consistently.

Trees produced by the construction

The events resampled at a parent and child share a variable. Moreover, any two nodes whose events use the same variable lie on one ancestor–descendant chain.

The label of a node is the event resampled there. We call the result a *witness tree*. Two children of a node cannot have the same label: when the earlier of two such resamplings is processed, it attaches below the later one already in the tree.

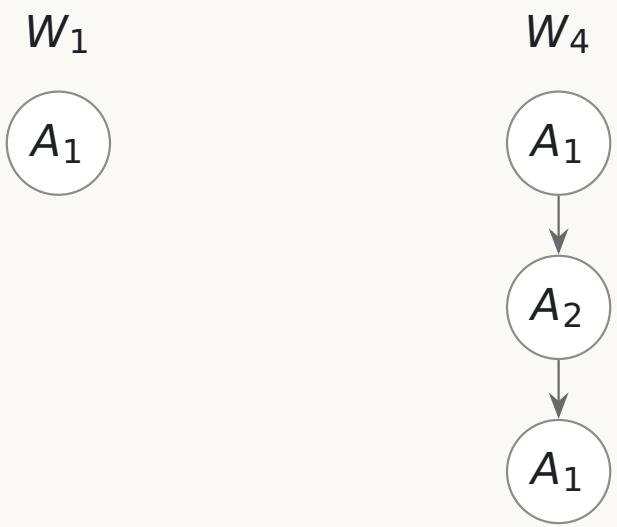
Distinct resamplings give distinct trees

Property 1: Distinctness. Different resamplings of A_i produce different witness trees.

Fix one execution. Suppose A_i is resampled at times

$$t_1 < t_2 < \cdots < t_{N_i}.$$

The root of W_{t_j} represents a resampling of A_i. *Every* earlier resampling of A_i is included, since it uses the same variables as the root.



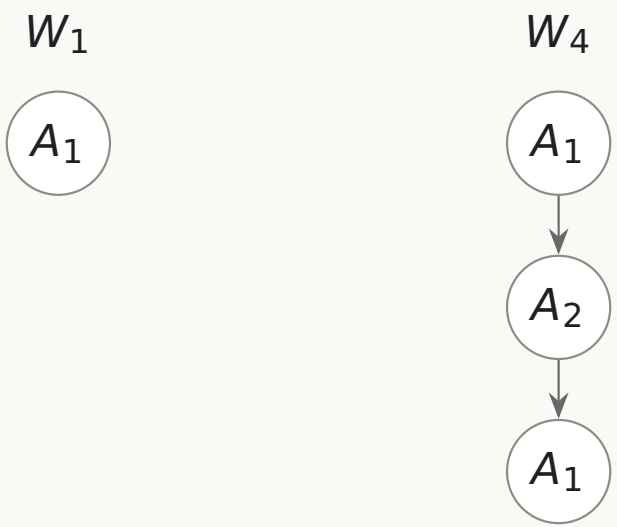
Distinct resamplings give distinct trees

Property 1: Distinctness. Different resamplings of A_i produce different witness trees.

Fix one execution. Suppose A_i is resampled at times

$$t_1 < t_2 < \cdots < t_{N_i}.$$

The root of W_{t_j} represents a resampling of A_i. *Every* earlier resampling of A_i is included, since it uses the same variables as the root.
Thus W_{t_j} has exactly j nodes representing resamplings of A_i. The trees constructed at different resamplings of A_i are distinct.



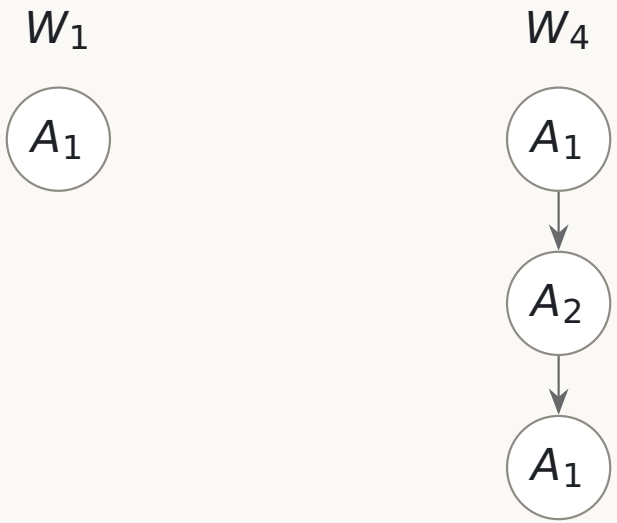
Distinct resamplings give distinct trees

Property 1: Distinctness. Different resamplings of A_i produce different witness trees.

Fix one execution. Suppose A_i is resampled at times

$$t_1 < t_2 < \cdots < t_{N_i}.$$

The root of W_{t_j} represents a resampling of A_i. *Every* earlier resampling of A_i is included, since it uses the same variables as the root.
Thus W_{t_j} has exactly j nodes representing resamplings of A_i. The trees constructed at different resamplings of A_i are distinct.
In our trace, the first resampling of A_1 gives a single-node tree; step 4 gives the three-node tree drawn earlier.



Counting resamplings by counting trees

Let $\mathcal{T}_i$ consist of all finite trees that can be produced by this backward construction from some sequence of event labels ending in A_i. The sequence need not be an actual execution, and $\mathcal{T}_i$ is generally infinite. Say that τ *appears* in an execution if $W_t = \tau$ for some step t.

Because the N_i resamplings of A_i give exactly N_i different trees rooted at A_i, for *each execution*,

$$\boxed{N_i = \sum_{\tau \in \mathcal{T}_i} \mathbf{1}\{\tau \text{ appears}\}.}$$

Counting resamplings by counting trees

Let $\mathcal{T}_i$ consist of all finite trees that can be produced by this backward construction from some sequence of event labels ending in A_i. The sequence need not be an actual execution, and $\mathcal{T}_i$ is generally infinite. Say that τ *appears* in an execution if $W_t = \tau$ for some step t.

Because the N_i resamplings of A_i give exactly N_i different trees rooted at A_i, for *each execution*,

$$\boxed{N_i = \sum_{\tau \in \mathcal{T}_i} \mathbf{1}\{\tau \text{ appears}\}.}$$

Taking expectations,

$$\boxed{\mathbb{E}[N_i] = \sum_{\tau \in \mathcal{T}_i} \Pr(\tau \text{ appears}).}$$

Counting resamplings by counting trees

Let $\mathcal{T}_i$ consist of all finite trees that can be produced by this backward construction from some sequence of event labels ending in A_i. The sequence need not be an actual execution, and $\mathcal{T}_i$ is generally infinite. Say that τ *appears* in an execution if $W_t = \tau$ for some step t.

Because the N_i resamplings of A_i give exactly N_i different trees rooted at A_i, for *each execution*,

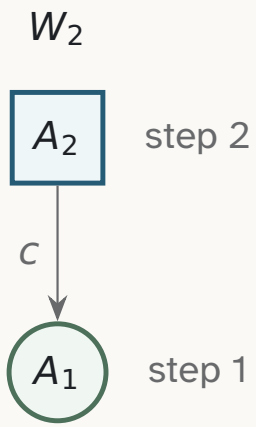
$$\boxed{N_i = \sum_{\tau \in \mathcal{T}_i} \mathbf{1}\{\tau \text{ appears}\}.}$$

Taking expectations,

$$\boxed{\mathbb{E}[N_i] = \sum_{\tau \in \mathcal{T}_i} \Pr(\tau \text{ appears}).}$$

We have reduced the problem of bounding $\mathbb{E}[N_i]$ to bounding $\Pr(\tau \text{ appears})$ for a fixed tree τ.

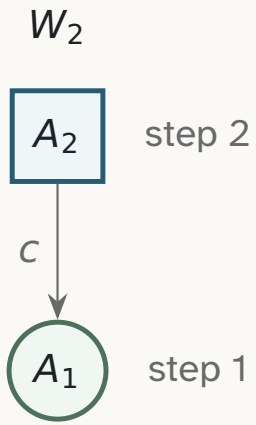
Which random values does W_2 use?



course	(0)	(1)	(2)
a	M	M	
b	M	M	
c	M	T	M
d	T	T	
e	T	M	

W_2 is the witness tree for step 2: root A_2, with the step-1 resampling of A_1 below it because both use course c. Each event was resampled because its cohort was stuck. **Which cells were used to evaluate these two events?**

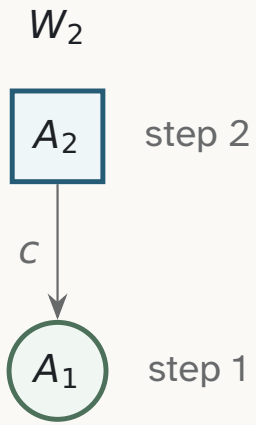
Which random values does W_2 use?



course	(0)	(1)	(2)
a	M	M	
b	M	M	
c	M	T	M
d	T	T	
e	T	M	

Step 1 (A_1, circle): nothing had been redrawn yet, so the algorithm evaluated A_1 using $a^{(0)}, b^{(0)}, c^{(0)} = (\mathrm{M}, \mathrm{M}, \mathrm{M})$.

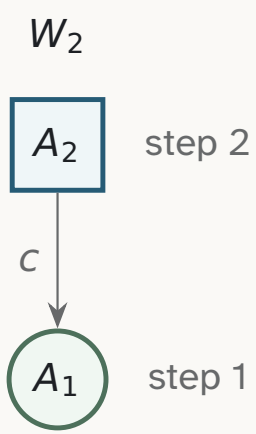
Which random values does W_2 use?



course	(0)	(1)	(2)
a	M	M	
b	M	M	
c	M	T	M
d	T	T	
e	T	M	

Step 2 (A_2, square): c had been redrawn once, by step 1. The algorithm evaluated A_2 using $c^{(1)}$, $d^{(0)}$, $e^{(0)} = (\mathsf{T}, \mathsf{T}, \mathsf{T})$—a *new* copy of c.

Which random values does W_2 use?

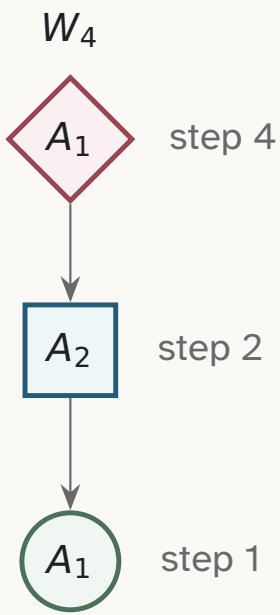


course	(0)	(1)	(2)
a	M	M	
b	M	M	
c	M	T	M
d	T	T	
e	T	M	

The two event evaluations use six distinct, independent cells. Both violations are necessary for W_2 to appear, but the algorithm must also select the corresponding events:

$$\Pr(W_2 \text{ appears}) \leq \tfrac{1}{4} \cdot \tfrac{1}{4} = p^2.$$

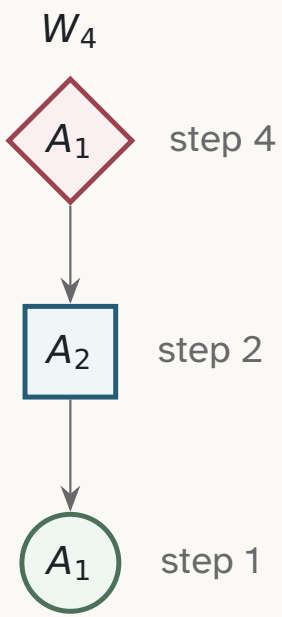
Exercise: which cells does W_4 read?



course	(0)	(1)	(2)	(3)
a	M	M	T	
b	M	M	M	
c	M	T	M	M
d	T	T		
e	T	M	T	
f	M	M		
g	M	T		

These are all the cells our four-step trace used. The bottom two nodes use the same cells as in W_2. **Which cells are used to evaluate A_1 at the root? How many times had a, b, c been redrawn?**

Exercise: which cells does W_4 read?



course	(0)	(1)	(2)	(3)
a	M	M	T	
b	M	M	M	
c	M	T	M	M
d	T	T		
e	T	M	T	
f	M	M		
g	M	T		

Root (A_1, diamond): a and b had been redrawn once and c twice, so the algorithm used $a^{(1)}, b^{(1)}, c^{(2)} = (\mathrm{M}, \mathrm{M}, \mathrm{M})$. The three event evaluations use nine distinct, independent cells:

$$\Pr(W_4 \text{ appears}) \le \left(\tfrac{1}{4}\right)^3 = p^3.$$

A witness tree determines which random draws are used

For each node, the tree tells us how many times each variable was previously resampled: count the nodes below it that use that variable. For node v and variable X_j used at v:

$$\boxed{\begin{aligned} &v \text{ reads cell } X_j^{(r(v,j))}, \\ &r(v,j) = \#\{\text{nodes below } v \text{ whose events use } X_j\}. \end{aligned}}$$

For W_4: At the root, two nodes below use c, so the root reads $c^{(2)}$; one node below uses a, so it reads $a^{(1)}$. At the middle node, one node below uses c: $c^{(1)}$.

A witness tree determines which random draws are used

For each node, the tree tells us how many times each variable was previously resampled: count the nodes below it that use that variable. For node v and variable X_j used at v:

$$\boxed{\begin{aligned} &v \text{ reads cell } X_j^{(r(v,j))}, \\ &r(v,j) = \#\{\text{nodes below } v \text{ whose events use } X_j\}. \end{aligned}}$$

For W_4: At the root, two nodes below use c, so the root reads $c^{(2)}$; one node below uses a, so it reads $a^{(1)}$. At the middle node, one node below uses c: $c^{(1)}$. If three nodes on a chain use X_j, which cells do they read, from bottom to top?

$$X_j^{(0)},\ X_j^{(1)},\ X_j^{(2)}.$$

An appearing witness tree requires every node's event to oc

Fix a witness tree τ. For each node v, let E_v be the event that the bad event labeling v occurs when evaluated using the cells prescribed by τ. Define

$$\boxed{Viol_\tau = \bigcap_{v \in \tau} E_v.}$$

Suppose $\tau = W_t$ appears in an execution.

- The cells prescribed by τ are exactly those used at the corresponding resampling steps.
- The algorithm selects an event for resampling only when it is violated.

An appearing witness tree requires every node's event to oc

Fix a witness tree τ. For each node v, let E_v be the event that the bad event labeling v occurs when evaluated using the cells prescribed by τ. Define

$$\boxed{Viol_\tau = \bigcap_{v \in \tau} E_v.}$$

Suppose $\tau = W_t$ appears in an execution.

- The cells prescribed by τ are exactly those used at the corresponding resampling steps.
- The algorithm selects an event for resampling only when it is violated.

Thus every E_v must occur, so

$$\boxed{\{\tau \text{ appears}\} \subseteq Viol_\tau.}$$

Event evaluations at different nodes are independent

Property 2: Probability. Each node requires a violation determined by cells distinct from those used at other nodes.

Two nodes whose events are determined by the same variable lie on one ancestor–descendant chain. Along that chain, $r(v, j)$ takes different values at different nodes. So *no two nodes read the same cell*.

Event evaluations at different nodes are independent

Property 2: Probability. Each node requires a violation determined by cells distinct from those used at other nodes.

Two nodes whose events are determined by the same variable lie on one ancestor–descendant chain. Along that chain, $r(v, j)$ takes different values at different nodes. So *no two nodes read the same cell*.

The events E_v are therefore determined by disjoint sets of independent cells, so they are independent, and each occurs with probability at most p:

$$\begin{aligned}\Pr(\tau \text{ appears}) \le \Pr(Viol_\tau) &= \prod_{v \in \tau} \Pr(E_v) \\ &\le p^{|\tau|}.\end{aligned}$$

Event evaluations at different nodes are independent

Property 2: Probability. Each node requires a violation determined by cells distinct from those used at other nodes.

Two nodes whose events are determined by the same variable lie on one ancestor–descendant chain. Along that chain, $r(v, j)$ takes different values at different nodes. So *no two nodes read the same cell*.

The events E_v are therefore determined by disjoint sets of independent cells, so they are independent, and each occurs with probability at most p:

$$\begin{aligned}\Pr(\tau \text{ appears}) \le \Pr(Viol_\tau) &= \prod_{v \in \tau} \Pr(E_v) \\ &\le p^{|\tau|}.\end{aligned}$$

This is what we computed by hand: p^2 for W_2 and p^3 for W_4. Combined with the counting step,

$$\mathbb{E}[N_i] \le \sum_{\tau \in \mathcal{T}_i} p^{|\tau|}.$$

Bounding the total weight of witness trees

Property 3: Counting. We now need to bound the sum of witness weights:

$$\mathbb{E}[N_i] = \sum_{\tau \in \mathcal{T}_i} \Pr(\tau \text{ appears}) \leq \sum_{\tau \in \mathcal{T}_i} p^{|\tau|}.$$

A direct count would group trees by size:

$$\sum_{\tau \in \mathcal{T}_i} p^{|\tau|} = \sum_{t \geq 1} \big(\#\{t\text{-node trees in } \mathcal{T}_i\}\big) p^t.$$

A tree may have arbitrarily many nodes. Even for a fixed number of nodes, there are many possible branching patterns and event labels.

Random generation of witness trees

We want to bound the sum $\sum_{\tau \in \mathcal{T}_i} p^{|\tau|}$ using an auxiliary random generator of labeled trees.

Let $Q(\tau)$ be the probability that the generator outputs exactly τ. Since distinct finite outputs are disjoint,

$$\sum_{\tau \in \mathcal{T}_i} Q(\tau) \leq 1.$$

Random generation of witness trees

We want to bound the sum $\sum_{\tau \in \mathcal{T}_i} p^{|\tau|}$ using an auxiliary random generator of labeled trees.

Let $Q(\tau)$ be the probability that the generator outputs exactly τ. Since distinct finite outputs are disjoint,

$$\sum_{\tau \in \mathcal{T}_i} Q(\tau) \leq 1.$$

Our goal: Design the generator so that

$$\boxed{Q(\tau) \geq d\, p^{|\tau|}} \qquad \text{for every } \tau \in \mathcal{T}_i,$$

then

$$\sum_{\tau \in \mathcal{T}_i} p^{|\tau|} \leq \frac{1}{d} \sum_{\tau \in \mathcal{T}_i} Q(\tau) \leq \frac{1}{d}.$$

Random generation of witness trees

We want to bound the sum $\sum_{\tau \in \mathcal{T}_i} p^{|\tau|}$ using an auxiliary random generator of labeled trees.

Let $Q(\tau)$ be the probability that the generator outputs exactly τ. Since distinct finite outputs are disjoint,

$$\sum_{\tau \in \mathcal{T}_i} Q(\tau) \leq 1.$$

Our goal: Design the generator so that

$$\boxed{Q(\tau) \geq d\, p^{|\tau|}} \qquad \text{for every } \tau \in \mathcal{T}_i,$$

then

$$\sum_{\tau \in \mathcal{T}_i} p^{|\tau|} \leq \frac{1}{d} \sum_{\tau \in \mathcal{T}_i} Q(\tau) \leq \frac{1}{d}.$$

How can we design such a generator? Choose which dependent events appear as children of each node.

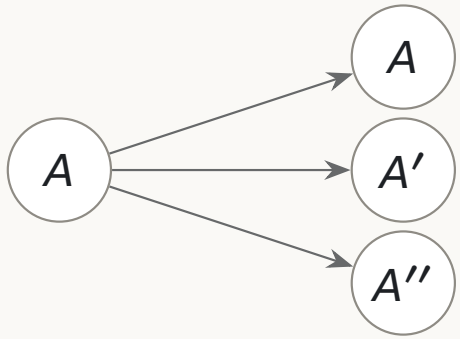
An auxiliary generator of labeled trees

Fix a root label A_i. Start with one root node labeled A_i.
For a node labeled A, the possible child labels are

$$A \quad \text{and every bad event dependent on } A.$$

These are exactly the labels allowed as children of A in a witness tree. The label A itself is allowed because the same bad event may be resampled more than once.

For each possible child label, independently include that child with probability q. Repeat from every new child.



each possible child:
included with probability q,
independently

One node of the generated tree

Suppose a node labeled A has four possible child labels:

$$A,\ A',\ A'',\ A'''.$$

If a target tree has exactly children A and A'' at this node, then the generator must make these choices:

possible child	A	A'	A''	A'''
generator choice	include	exclude	include	exclude
probability factor	q	$1-q$	q	$1-q$

This node contributes $q^2(1-q)^2$. In general, a node v with s_v possible child labels and c_v actual children contributes

$$q^{c_v}(1-q)^{s_v-c_v}, \qquad s_v \le d+1.$$

Probability of one exact finite tree

Fix a witness tree τ with $t = |\tau|$ nodes. The generator produces exactly τ when

- every child appearing in τ is included, and
- every possible child absent from τ is excluded.

These decisions are independent, so we multiply the factors from each node:

$$Q(\tau) = \prod_{v \in \tau} q^{c_v} (1-q)^{s_v - c_v}.$$

Probability of one exact finite tree

Fix a witness tree τ with $t = |\tau|$ nodes. The generator produces exactly τ when

- every child appearing in τ is included, and
- every possible child absent from τ is excluded.

These decisions are independent, so we multiply the factors from each node:

$$Q(\tau) = \prod_{v \in \tau} q^{c_v} (1-q)^{s_v - c_v}.$$

A t-node tree has $t-1$ edges, so $\sum_v c_v = t-1$:

$$\boxed{Q(\tau) = q^{t-1} (1-q)^{\sum_v (s_v - c_v)}.}$$

The exponent of $1-q$ still depends on the *shape* of τ.

Lower-bounding $Q(\tau)$ using the tree size

We want a lower bound on $Q(\tau)$ that depends only on $t = |\tau|$. Since $s_v \leq d + 1$ at every node,

$$\sum_v (s_v - c_v) \leq (d+1)t - (t-1) = dt + 1.$$

This is the worst case, in which every node has all $d + 1$ possible child labels.

Lower-bounding $Q(\tau)$ using the tree size

We want a lower bound on $Q(\tau)$ that depends only on $t = |\tau|$. Since $s_v \le d+1$ at every node,

$$\sum_v (s_v - c_v) \le (d+1)t - (t-1) = dt + 1.$$

This is the worst case, in which every node has all $d+1$ possible child labels. Because $0 < 1-q < 1$, a larger exponent makes $(1-q)^{\cdots}$ smaller, so

$$Q(\tau) \ge q^{t-1}(1-q)^{dt+1} = \frac{1-q}{q}\,[q(1-q)^d]^t.$$

The lower bound now depends only on the number of nodes.

Choosing the branching probability

The lower bound contains the t-dependent factor

$$[q(1-q)^d]^t.$$

To make the lower bound as large as possible for large trees, maximize

$$q(1-q)^d.$$

Choose

$$q = \frac{1}{d+1},$$

which maximizes this expression.
With this choice,

$$q(1-q)^d = \frac{1}{d+1}\left(\frac{d}{d+1}\right)^d > \frac{1}{e(d+1)}, \qquad \frac{1-q}{q} = d.$$

The LLL condition gives $Q(\tau) \geq dp^{|\tau|}$

The condition says

$$ep(d+1) \leq 1, \qquad \text{so} \qquad p \leq \frac{1}{e(d+1)}.$$

From the choice of q,

$$\frac{1}{e(d+1)} < q(1-q)^d, \qquad \text{so} \qquad p < q(1-q)^d.$$

For a fixed tree τ with $t = |\tau|$,

$$Q(\tau) \geq \frac{1-q}{q}[q(1-q)^d]^t > dp^t.$$

Therefore

$$\boxed{p^{|\tau|} \leq \frac{1}{d}Q(\tau)}.$$

Every event is resampled at most $1/d$ times in expectation

Fix the bad event A_i.

$$\mathbb{E}[N_i] = \sum_{\tau \in \mathcal{T}_i} \Pr(\tau \text{ appears})$$

each resampling of A_i has its own witness tree

Every event is resampled at most $1/d$ times in expectation

Fix the bad event A_i.

$$\mathbb{E}[N_i] = \sum_{\tau \in \mathcal{T}_i} \Pr(\tau \text{ appears}) \leq \sum_{\tau \in \mathcal{T}_i} p^{|\tau|}$$

probability bound for one fixed witness tree

Every event is resampled at most $1/d$ times in expectation

Fix the bad event A_i.

$$\begin{aligned}\mathbb{E}[N_i] &= \sum_{\tau \in \mathcal{T}_i} \Pr(\tau \text{ appears}) \\ &\leq \sum_{\tau \in \mathcal{T}_i} p^{|\tau|} \\ &\leq \frac{1}{d} \sum_{\tau \in \mathcal{T}_i} Q(\tau) \\ &\leq \frac{1}{d}.\end{aligned}$$

The probabilities of all finite outputs of the tree generator sum to at most 1.

Finite expected resamplings force termination

For every bad event A_i,

$$\mathbb{E}[N_i] \leq \frac{1}{d}.$$

Summing over the m bad events gives

$$\mathbb{E}[R] = \mathbb{E}\left[\sum_{i=1}^{m} N_i\right] = \sum_{i=1}^{m} \mathbb{E}[N_i] \leq \frac{m}{d} < \infty.$$

Finite expectation implies $\Pr(R = \infty) = 0$ (proof in the appendix). Thus the algorithm terminates with probability 1.
When it stops, no bad event occurs. In the variable setting this *reproves* the LLL: an algorithm that stops with probability 1 stops at an assignment avoiding every bad event, so such an assignment exists.

Questions?

Appendix contents

1. Appendix A: Classical LLL by induction
2. Appendix B: Shearer's lemma and independent-set polynomial
3. Appendix C: Edge-disjoint routing (exercise solution)
4. Appendix D: Sharpness, branching probability, and termination

Appendix A: Classical LLL by induction

Question: How can avoiding some bad events affect the probability of another?

We will bound conditional probabilities and use the chain rule to show that all bad events can be avoided.

History: Erdős and Lovász (1975) proved the lemma under $4pd \leq 1$; the $ep(d+1) \leq 1$ form is due to Spencer (1977).

Contents

Proof of the classical LLL

To prove that all bad events can be avoided, use the chain rule:

$$\Pr\left(\bigcap_{i=1}^{m} \overline{A_i}\right) = \prod_{i=1}^{m} \Pr\left(\overline{A_i} \mid \bigcap_{j<i} \overline{A_j}\right).$$

Each factor would be positive if we could show that conditioning on avoiding previous events does not make the next bad event certain.

The difficulty: avoiding events that are neighbors of A_i can increase its probability. Independence only helps us with the nonneighbors.

Can we bound $\Pr(A_i \mid$ **other bad events avoided)** **uniformly?**

Which conditional probabilities must we control?

Fix an event A_i and a set S of other bad events we have avoided. Separate its nonneighbors from its neighbors:

$$T = S \setminus \Gamma(i), \qquad U = S \cap \Gamma(i).$$

Let $F_T = \bigcap_{j \in T} \overline{A_j}$ and $F_U = \bigcap_{j \in U} \overline{A_j}$.
Because A_i is jointly independent of all its nonneighbors,

$$\Pr(A_i \mid F_T) = \Pr(A_i).$$

Conditioning on the neighboring events as well gives

$$\Pr(A_i \mid F_T \cap F_U) \le \frac{\Pr(A_i \mid F_T)}{\Pr(F_U \mid F_T)} = \frac{\Pr(A_i)}{\Pr(F_U \mid F_T)}.$$

Can we keep the denominator bounded away from zero?

A sufficient condition for the bound

Suppose we could show that, after avoiding any other events, the conditional probability of A_j is at most $w_j < 1$. The chain rule would then give

$$\Pr(F_U \mid F_T) \geq \prod_{j \in U} (1 - w_j).$$

Our preceding bound would imply

$$\Pr(A_i \mid F_T \cap F_U) \leq \frac{\Pr(A_i)}{\prod_{j \in U}(1 - w_j)}.$$

To make this at most w_i for every choice of U, it suffices that

$$\Pr(A_i) \leq w_i \prod_{j \in \Gamma(i)} (1 - w_j) \qquad \text{for every } i. \tag{$*$}$$

The induction claim

Assume we can choose $0 < w_i < 1$ satisfying $(*)$. We will prove, by induction on $|S|$, that for every $i \notin S$,

$$\boxed{\Pr\left(A_i \mid \bigcap_{j \in S} \overline{A_j}\right) \leq w_i.} \tag{$\dagger$}$$

The induction claim

Assume we can choose $0 < w_i < 1$ satisfying $(*)$. We will prove, by induction on $|S|$, that for every $i \notin S$,

$$\boxed{\Pr\left(A_i \mid \bigcap_{j \in S} \overline{A_j}\right) \leq w_i.} \qquad (\dagger)$$

For $S = \emptyset$, it follows from $(*)$ because

$$\Pr(A_i) \leq w_i \prod_{j \in \Gamma(i)} (1 - w_j) \leq w_i.$$

Bounding the conditional probability

List $U = \{j_1, \dots, j_r\}$. The chain rule and the induction hypothesis give

$$\begin{aligned}\Pr(F_U \mid F_T) &= \prod_{h=1}^{r} \Pr\left(\overline{A_{j_h}} \mid F_T, \overline{A_{j_1}}, \dots, \overline{A_{j_{h-1}}}\right) \\ &\geq \prod_{j \in U} (1 - w_j).\end{aligned}$$

Each conditioning set in this product has fewer than $|S|$ events, so induction applies.
Consequently,

$$\begin{aligned}\Pr(A_i \mid F_T \cap F_U) &\leq \frac{\Pr(A_i)}{\prod_{j \in U}(1 - w_j)} \\ &\leq \frac{w_i \prod_{j \in \Gamma(i)}(1 - w_j)}{\prod_{j \in U}(1 - w_j)} \leq w_i.\end{aligned}$$

This proves (†), and hence $\Pr(\bigcap_i \overline{A_i}) \geq \prod_i (1 - w_i) > 0$.

Recover the symmetric Local Lemma

If all event probabilities are at most p and all degrees are at most $d \geq 1$, take

$$w_i = w = \frac{1}{d+1}.$$

The preceding induction works whenever

$$p \leq w(1-w)^d = \frac{1}{d+1}\left(\frac{d}{d+1}\right)^d.$$

Since $\left(1+\frac{1}{d}\right)^d < e$,

$$\frac{1}{d+1}\left(\frac{d}{d+1}\right)^d > \frac{1}{e(d+1)}.$$

Therefore the simpler sufficient condition $ep(d+1) \leq 1$ proves the symmetric LLL.

Appendix B: Shearer's lemma

Can the full dependency graph give a better guarantee than the maximum-degree LLL?

We will use a polynomial built from the independent sets of the graph.

Contents

Using the full dependency graph

Recall PIE: it accounts for all intersections, but is hard to compute. Can we exploit the dependency graph to obtain a lower bound without calculating them all? The symmetric LLL uses only p and the maximum degree d. A criterion based on the full graph might be stronger.

First, which sets of bad events have no dependency edges among them?



Which subsets contain no adjacent pair?

Using the full dependency graph

Recall PIE: it accounts for all intersections, but is hard to compute. Can we exploit the dependency graph to obtain a lower bound without calculating them all? The symmetric LLL uses only p and the maximum degree d. A criterion based on the full graph might be stronger.

First, which sets of bad events have no dependency edges among them?



Which subsets contain no adjacent pair? These are the *independent sets*:

$$\varnothing, \quad \{1\}, \quad \{2\}, \quad \{3\}, \quad \{1,3\}.$$

Events indexed by an independent set are mutually independent.

The independent-set polynomial

For $S \subseteq V(G)$, write $\mathcal{I}(G[S])$ for the independent sets in the graph induced by S. Define

$$Z(S) = \sum_{I \in \mathcal{I}(G[S])} (-1)^{|I|} \prod_{i \in I} p_i, \qquad Z(\varnothing) = 1.$$

For the three-vertex path,

$$Z(\{1,2,3\}) = 1 - p_1 - p_2 - p_3 + p_1 p_3.$$

This resembles inclusion–exclusion, but only independent vertex sets occur.

Caution: We cannot simply discard dependent-set terms from ordinary PIE. In general $Z(S)$ is *not* the probability of avoiding all events in S. Shearer's lemma explains why it nevertheless gives a sharp lower bound.

The independent-set recurrence

Fix $v \in S$. Every independent set $I \subseteq S$ either omits v or contains it.
If $v \in I$, none of its neighbors can belong to I. The remaining elements form an independent set in $S \setminus N[v]$, where $N[v]$ contains v and its neighbors.

What recurrence does this give for $Z(S)$?

The independent-set recurrence

Fix $v \in S$. Every independent set $I \subseteq S$ either omits v or contains it.
If $v \in I$, none of its neighbors can belong to I. The remaining elements form an independent set in $S \setminus N[v]$, where $N[v]$ contains v and its neighbors.

What recurrence does this give for $Z(S)$?

$$\boxed{Z(S) = Z(S \setminus \{v\}) - p_v Z(S \setminus N[v]).}$$

The independent-set recurrence

Fix $v \in S$. Every independent set $I \subseteq S$ either omits v or contains it.
If $v \in I$, none of its neighbors can belong to I. The remaining elements form an independent set in $S \setminus N[v]$, where $N[v]$ contains v and its neighbors.

What recurrence does this give for $Z(S)$?

$$\boxed{Z(S) = Z(S \setminus \{v\}) - p_v Z(S \setminus N[v]).}$$

The actual avoidance probability $P(S) = \Pr(\bigcap_{i \in S} \overline{A_i})$ satisfies the analogous *inequality*

$$P(S) \geq P(S \setminus \{v\}) - p_v P(S \setminus N[v]).$$

The difference comes from dependencies among neighboring events.

Shearer's lemma

Shearer's lemma (1985)

Suppose G is a dependency graph for events A_i with $\Pr(A_i) \le p_i$. If

$$Z(T) > 0 \qquad \text{for every } T \subseteq V(G),$$

then for all $S \subseteq V(G)$,

$$\boxed{\Pr\left(\bigcap_{i \in S} \overline{A_i}\right) \ge Z(S) > 0.}$$

The positivity region is best possible for a universal guarantee based only on G and the individual probability bounds.

The ordinary LLL is easier to check. Shearer's criterion uses the full graph and can certify avoidance in cases where a maximum-degree LLL bound does not.

Proof of Shearer's lemma

Recall the two recurrences, where $P(S)$ is the true avoidance probability:

$$Z(S) = Z(S \setminus \{v\}) - p_v Z(S \setminus N[v]),$$
$$P(S) \geq P(S \setminus \{v\}) - p_v P(S \setminus N[v]).$$

The subtraction makes directly inducting on $P(S) \geq Z(S)$ awkward. Instead, induct on $|S|$ to show, for every $v \in S$,

$$\boxed{\frac{P(S)}{P(S \setminus \{v\})} \geq \frac{Z(S)}{Z(S \setminus \{v\})}.}$$

The denominators are positive under the induction and the assumed positivity of Z.
The left ratio is the conditional probability of avoiding A_v after avoiding the other events in S—the same kind of quantity as in the classical proof.

The induction step

For $v \in S$, delete the other vertices of $N[v] \cap S$ one by one. The induction hypothesis, applied to those smaller sets, gives

$$\frac{P(S \setminus \{v\})}{P(S \setminus N[v])} \geq \frac{Z(S \setminus \{v\})}{Z(S \setminus N[v])}.$$

Consequently, using the avoidance-probability recurrence,

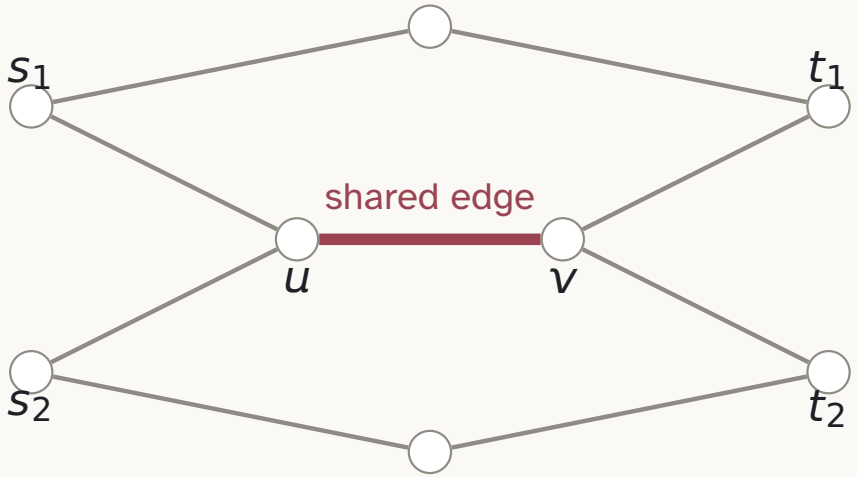
$$\begin{aligned} \frac{P(S)}{P(S \setminus \{v\})} &\geq 1 - p_v \frac{P(S \setminus N[v])}{P(S \setminus \{v\})} \\ &\geq 1 - p_v \frac{Z(S \setminus N[v])}{Z(S \setminus \{v\})} \\ &= \frac{Z(S)}{Z(S \setminus \{v\})}. \end{aligned}$$

Multiplying such ratios as we add vertices one at a time yields $P(S) \geq Z(S)$.

Shearer's sharpness direction requires an additional extremal construction; the calculation above proves the lower bound.

Appendix C: Routing exercise, solution

Suppose n source–destination pairs need routes through a network $G = (V, E)$. For each pair, we independently choose a random path from a collection of candidate paths.



The routes $s_1 \to u \to v \to t_1$ and $s_2 \to u \to v \to t_2$ conflict, but the network offers alternatives.

Contents

When are route conflicts unlikely?

Let $n \geq 3$, and let P_i be the independently selected path for connection i. For $i < j$, the bad event is

$$A_{ij} = \{P_i \text{ and } P_j \text{ share an edge}\}.$$

Assume every candidate path has at most ℓ edges and the choice distributions spread traffic:

$$\Pr(e \in P_j) \leq \rho \quad \text{for every network edge } e \text{ and every connection } j.$$

Condition on a path P_i and apply the union bound along its edges:

$$\Pr(A_{ij} \mid P_i) \leq \sum_{e \in P_i} \Pr(e \in P_j) \leq \ell\rho.$$

Hence $p \leq \ell\rho$. Each conflict involves two path variables, so it overlaps at most $d = 2n - 4$ other conflict events.

When are route conflicts unlikely?

Let $n \geq 3$, and let P_i be the independently selected path for connection i. For $i < j$, the bad event is

$$A_{ij} = \{P_i \text{ and } P_j \text{ share an edge}\}.$$

Assume every candidate path has at most ℓ edges and the choice distributions spread traffic:

$$\Pr(e \in P_j) \leq \rho \quad \text{for every network edge } e \text{ and every connection } j.$$

Condition on a path P_i and apply the union bound along its edges:

$$\Pr(A_{ij} \mid P_i) \leq \sum_{e \in P_i} \Pr(e \in P_j) \leq \ell\rho.$$

Hence $p \leq \ell\rho$. Each conflict involves two path variables, so it overlaps at most $d = 2n - 4$ other conflict events.
The LLL condition is

$$e\,\ell\rho\,(2n-3) \leq 1.$$

Moser–Tardos resamples the two paths involved in any conflict. The assumption is useful when the network has genuinely diverse route choices; it cannot hold for a compulsory bottleneck edge.

Appendix D: Proof calculations

The exact symmetric LLL condition, the branching probability in the witness-tree proof, and the last step from finite expectation to termination.

Contents

The symmetric LLL constant

The proof works under the slightly weaker condition

$$p \leq \frac{d^d}{(d+1)^{d+1}}.$$

The usual hypothesis $ep(d+1) \leq 1$ uses

$$\frac{d^d}{(d+1)^{d+1}} > \frac{1}{e(d+1)}.$$

This is a sufficient condition, not the exact threshold for the Local Lemma. More refined versions exploit event-specific probabilities and the dependency graph.

Optimizing $q(1-q)^d$

We want to maximize

$$f(q) = q(1-q)^d.$$

For $0 < q < 1$,

$$\frac{d}{dq} \log f(q) = \frac{1}{q} - \frac{d}{1-q}.$$

Setting this to zero gives

$$\frac{1}{q} = \frac{d}{1-q} \qquad \Longleftrightarrow \qquad q = \frac{1}{d+1}.$$

At this value,

$$f(q) = \frac{1}{d+1}\left(\frac{d}{d+1}\right)^d > \frac{1}{e(d+1)}.$$

Finite expectation forces termination

Let R be the total number of resampling steps, with values in $\{0, 1, 2, \dots\} \cup \{\infty\}$.
If $\Pr(R = \infty) = \alpha > 0$, then for every integer T,

$$\Pr(R \geq T) \geq \alpha.$$

Therefore

$$\mathbb{E}[R] \geq T \Pr(R \geq T) \geq T\alpha.$$

Letting $T \to \infty$ forces $\mathbb{E}[R] = \infty$. Thus $\mathbb{E}[R] < \infty$ implies $\Pr(R = \infty) = 0$.