4 Probability amplification

We have already seen in the first chapter that we can increase the success probability of randomized algorithms drastically through multiple runs. In this chapter we will present some refined methods in order to increase the success probability. In the first method, specific steps of the algorithm are repeated, instead of repeating the whole algorithm, in order to obtain a better running time. The second method is based on the so-called Lovász Local Lemma. This lemma in its original form just allows one to prove the existence of solutions to certain problems, but since several years also algorithms are known that can find these solutions efficiently.

4.1 Amplification through local repetition

We illustrate our first technique by applying it to an algorithm for the MINCUT problem. In the MINCUT problem we are given an undirected graph $G = (V, E)$ and we are looking for a subset $U \subseteq V$ such that $|(U, \bar{U})|$, i.e. the number of edges of the cut $(U, \bar{U})$, is minimal. This problem can be solved in polynomial time by using network flow algorithms. We present here a simple alternative algorithm that does not need algorithms for network flow or linear optimization.

A simple algorithm

In the following we assume that $G = (V, E)$ is a *multigraph*, i.e. there can be multiple edges between a pair of nodes. The basic idea of our algorithm is as follows:

In each step, pick a random edge $\{u, v\}$ from the current multigraph. This edge will be *contracted*, which means that the nodes u and v will be replaced by a new node w. All edges of the form $\{u, x\}$ with $x \neq v$ and $\{v, x\}$ with $x \neq u$ are replaced by the edge $\{w, x\}$. Edges between u and v are omitted. If there are only two nodes remaining, the edge set across the two sets of nodes in the original graph which are represented by the remaining nodes is returned by the algorithm. See also Figure 1.

Algorithm CONTRACT(G):
$H := G$
while H has more than two nodes **do**
    choose an edge $\{x, y\}$ uniformly at random from the edges in H and contract it
return the set of edges C of G that is represented by the remaining edges in H

Figure 1: The simple contraction algorithm.

The running time of the CONTRACT algorithm is $O(n^2)$ when using an efficient implementation (for example by storing G as an adjacency matrix). Moreover, the following theorem holds.

Theorem 4.1 *The cut computed by CONTRACT is a minimum cut with probability at least* $1/n^2$.

Proof. Let K be a minimum cut in G. If during the execution of the algorithm no edge of K is contracted, then the cut returned by the algorithm is K. We show that the probability that this happens is at least $1/n^2$. Let e_i be the edge contracted in round r of CONTRACT. It holds that:

$$\begin{aligned}
\Pr[C \text{ is a minimum cut}] &\geq \Pr[C = K] \\
&= \Pr[\bigwedge_{1 \leq i \leq n-2} (e_i \notin K)] \\
&= \prod_{1 \leq i \leq n-2} \Pr[e_i \notin K \mid \bigwedge_{1 \leq j < i} (e_j \notin K)]
\end{aligned}$$

Lemma 4.2 *For* $1 \leq i \leq n-2$ *it holds that*

$$\Pr[e_i \in K \mid \bigwedge_{1 \leq j < i} (e_j \notin K)] \leq \frac{2}{n-i+1}$$

Proof. Let H_i be the graph, that results from the contraction of the edge e_i. Let n_i be the number of nodes and m_i the number of (original edges that are still represented by) edges in H_i. By induction on i, one can show that as long as no edge in K is chosen, all edges in K are still present in H_i. This is because for any edge $e = \{v, w\} \in K$, v must be in U while w must be in $\bar{U}$ for the partition $(U, \bar{U})$ implied by K, so there cannot be two edges e and e' connecting nodes v and w with $e \in K$ and $e' \notin K$. Since every edge is chosen with the same probability, it holds that

$$\Pr[e_i \in K \mid \bigwedge_{1 \le j < i} (e_j \notin K)] = \frac{|K|}{m_{i-1}}$$

Moreover, it holds that $m_{i-1} \geq |K| \cdot n_{i-1}/2$ because otherwise there would be a node in H with degree smaller than $|K|$. In this case, however, the node set represented by it in G would be left by less than $|K|$ edges, which contradicts our assumption that K is a minimum cut. Thus, $|K|/m_{i-1} \leq 2/n_{i-1}$, and since $n_{i-1} = n - i + 1$, the lemma is proven. □

We now use Lemma 4.2 in order to complete the proof of Theorem 4.1:

$$\begin{aligned} \Pr[C \text{ is a minimum cut}] &\geq \prod_{1 \le i \le n-2} \left(1 - \frac{2}{n-i+1}\right) \\ &= \prod_{1 \le i \le n-2} \frac{n-i-1}{n-i+1} = \frac{2}{n(n-1)} \geq \frac{1}{n^2} \end{aligned}$$

□

If we repeat the CONTRACT algorithm sufficiently often, we can obtain a constant success probability. More specifically, we would need $\Theta(n^2)$ repetitions, with a total running time of $O(n^4)$, in order to achieve that. This running time can be significantly reduced by adding redundancy to the contraction of graphs.

A fast algorithm

Our improvement is based on the following observation: If we run the CONTRACT algorithm until there are t nodes left in the graph, then the probability that no edge of a minimum cut K has been contracted is at least

$$\prod_{1 \le i \le n-t} \left(1 - \frac{2}{n-i+1}\right) = \prod_{1 \le i \le n-t} \frac{n-i-1}{n-i+1} = \frac{t(t-1)}{n(n-1)}$$

Therefore, the probability that the algorithm chooses no edge of K during the first $n/2$ contractions is at least $1/4$. Hence, the high error probability results primarily from the last contractions of the algorithm. On the other hand, the algorithm runs faster on smaller graphs than in larger ones, i.e. we can also allow more trials to happen for smaller graphs. An effective method to implement this is presented in Figure 2.

Algorithm FASTCUT(G):
if $n \leq 6$ **then**
    compute the minimum cut by "brute force" and return it
else
    $t := \lceil 1 + n/\sqrt{2} \rceil$
    use the CONTRACT algorithm to compute independently of each other two graphs H_A and H_B with t nodes
    C_1 :=FASTCUT(H_A)
    C_2 :=FASTCUT(H_B)
    return the smallest of C_1 and C_2

Figure 2: The fast contraction algorithm.

The running time $T(n)$ of the FASTCUT algorithm is described through the recursion

$$T(n) = 2 \cdot T(\lceil 1 + n/\sqrt{2} \rceil) + O(n^2)$$

By using the Master Theorem we get $T(n) = O(n^2 \log n)$, which is clearly better than $O(n^4)$. Moreover it holds:

Theorem 4.3 *The FASTCUT algorithm computes a minimum cut with probability* $\Omega(1/\log n)$.

Proof. Let K be a minimum cut. As we have already seen, the probability that during the contraction of at most t nodes no edge of K is chosen, is at least

$$\frac{t(t-1)}{n(n-1)} = \frac{(\lceil 1 + n/\sqrt{2} \rceil) \cdot \lceil n/\sqrt{2} \rceil}{n(n-1)} \geq \frac{1}{2}$$

Let $E(G)$ be the event that FASTCUT(G) computes a minimum cut, $E(G, H)$ be the event that FASTCUT(G) computes a minimum cut via H, and $E(G \to H)$ be the event that the contraction from G down to H preserves a minimum cut. Then it holds that

$$\Pr[E(G, H_A)] = \Pr[E(G \to H_A)] \cdot \Pr[E(H_A)] \geq \frac{1}{2} \cdot \Pr[E(H_A)]$$

and the same holds for H_B. So

$$\begin{aligned} \Pr[\overline{E(G)}] &= \Pr[\overline{E(G, H_A)}] \cdot \Pr[\overline{E(G, H_B)}] \\ &= (1 - \Pr[E(G, H_A)]) \cdot (1 - \Pr[E(G, H_B)]) \\ &\leq (1 - \Pr[E(H_A)]/2) \cdot (1 - \Pr[E(H_B)]/2) \end{aligned}$$

and hence,

$$\Pr[E(G)] \geq 1 - (1 - \Pr[E(H_A)]/2) \cdot (1 - \Pr[E(H_B)]/2)$$

Let $p(k)$ be the probability that the FASTCUT algorithm computes a minimum cut at a recursion-height of k. From the inequality above we get

$$p(k) \geq 1 - \left(1 - \frac{p(k-1)}{2}\right)^2$$

for $k \geq 1$ and $p(0) = 1$. We now show that $p(k) \geq 1/(k+1)$ by induction. The induction basis holds due to $p(0) = 1$. Let now the assumption hold for $k - 1$. Then we have

$$\begin{aligned} p(k) &\geq 1 - \left(1 - \frac{p(k-1)}{2}\right)^2 \geq 1 - \left(1 - \frac{1}{2k}\right)^2 \\ &= \frac{1}{k} - \frac{1}{4k^2} = \frac{(4k-1)(k+1)}{4k^2} \cdot \frac{1}{k+1} \\ &\geq \frac{1}{k+1} \end{aligned}$$

Since the depth of the recursion for a graph with n nodes is $O(\log n)$, the theorem holds. □

The repetition of the FASTCUT algorithm for a logarithmic number of times results (through the choice of a cut of minimal size) into an algorithm with constant success probability.

4.2 Amplification through decomposition

In previous chapters, we have often used the following statement.

Lemma 4.4 *Let $A_1, \ldots, A_n$ be events of an arbitrary random experiment with probabilities*

$$\sum_{i=1}^{n} \Pr[A_i] < 1 \ .$$

Then there exists an outcome of the random experiment such that no event is true.

Proof. According to the inclusion exclusion principle it holds that

$$\Pr[A_1 \cup A_2 \cup \ldots \cup A_n] \leq \sum_{i=1}^{n} \Pr[A_i] < 1$$

and since $\Pr[\bar{A}_1 \cap \bar{A}_2 \cap \ldots \cap \bar{A}_n] = 1 - \Pr[A_1 \cup A_2 \cup \ldots \cup A_n]$, it follows that

$$\Pr[\bar{A}_1 \cap \bar{A}_2 \cap \ldots \cap \bar{A}_n] > 0$$

and the lemma is proven. □

This statement is the best possible we can formulate for the general case, but it can be improved significantly if more information about the dependence of the events is available. These dependencies are given through the so-called dependency graph.

Let $A_1, \ldots, A_n$ be an arbitrary collection of events. An undirected graph $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$ is called a *dependency graph* of the events $A_1, \ldots, A_n$ if for every node i and every subset $S \subseteq V \setminus \{i\}$ that does not contain a neighbor of i in G, it holds that A_i is independent of S, i.e. $\Pr[A_i \mid \bigcap_{j \in S} A_j] = \Pr[A_i]$. This condition implies following statement.

Lemma 4.5 *Let $A_1, \ldots, A_n$ be a collection of events and $G = (V, E)$ their dependency graph. Then it holds for every independent set $S \subseteq V$ (i.e. no pair of nodes in S is connected by an edge in G) that $\Pr[\bigcap_{i \in S} A_i] = \prod_{i \in S} \Pr[A_i]$.*

The proof is left as an exercise. In order to illustrate the definition of the dependency graph, we consider the MAXCUT problem for a graph $G = (V, E)$ with $|E| = m$.

Suppose we consider the random experiment where each node is assigned uniformly at random and independently of the other nodes to either U or $\bar{U}$, w.r.t. a cut $(U, \bar{U})$. Let A_e be an event that corresponds to an edge $e \in E$. A_e is true if and only if e does not belong to the cut. For these events we can define a dependency graph $G' = (\{1, \ldots, m\}, E')$ that contains a node i for each edge $e_i \in E$, and for which two nodes i and j are connected by an edge if and only if e_i and e_j have a node in G in common. Let us now consider an arbitrary node i in G and an arbitrary subset of nodes $S \subseteq \{1, \ldots, m\} \setminus \{i\}$ that are not connected with i in G'. Since in that case e_i has no node in common with all edges e_j with $j \in S$, it obviously holds that $\Pr[A_{e_i} \mid \bigcap_{j \in S} A_{e_j}] = \Pr[A_{e_i}]$, i.e., G' satisfies the conditions of the dependency graph.

As another example, we consider the MAXSAT problem. Let $\phi = \bigwedge_{i=1}^{m} C_i$ be an arbitrary instance of this problem, where each C_i is a clause. Now we consider the random experiment that each boolean variable x in ϕ is set to true or false uniformly at random and independently of the other variables. Let A_{C_i} be the event that is realized if C_i is not satisfied. Then the graph $G = (\{1, \ldots, m\}, E)$ with the property that $\{i, j\} \in E$ if and only if C_i and C_j have a common boolean variable, obviously is a dependency graph for this random experiment.

We are now ready to present the general Lovász Local Lemma [1].

Lemma 4.6 (Lovász Local Lemma) *Let $A_1, \ldots, A_n$ be "bad" events in an arbitrary probability space. Suppose that $G = (V, E)$ is a dependency graph of these events and suppose there are real numbers $x_i \in (0, 1)$ for all $1 \leq i \leq n$ with*

$$\Pr[A_i] \leq x_i \prod_{\{i,j\} \in E} (1 - x_j)$$

for all $1 \leq i \leq n$. Then,

$$\Pr[\bigcap_{i=1}^{n} \bar{A}_i] \geq \prod_{i=1}^{n} (1 - x_i) \ .$$

In particular, with positive probability no bad event A_i holds.

Proof. We first prove, by induction on s, that for any set $S \subset \{1, \ldots, n\}$, $|S| = s$, and $i \notin S$,

$$\Pr[A_i \mid \bigcap_{j \in S} \bar{A}_j] \leq x_i \ . \tag{1}$$

This is certainly true for $s = 0$. Assuming it holds for all $s' < s$, we prove it for S. Set $S_1 = \{j \in S \mid \{i,j\} \in E\}$ and $S_2 = S \setminus S_1$. Then

$$\Pr[A_i \mid \bigcap_{j\in S} \bar{A}_j] = \frac{\Pr[A_i \cap (\bigcap_{j\in S_1} \bar{A}_j) \mid \bigcap_{\ell\in S_2} \bar{A}_\ell]}{\Pr[\bigcap_{j\in S_1} \bar{A}_j \mid \bigcap_{\ell\in S_2} \bar{A}_\ell]} \ . \tag{2}$$

To bound the numerator, observe that since A_i is mutually independent of the events $\{A_\ell : \ \ell \in S_2\}$,

$$\begin{aligned} \Pr[A_i \cap (\bigcap_{j\in S_1} \bar{A}_j) \mid \bigcap_{\ell\in S_2} \bar{A}_\ell] &\leq \Pr[A_i \mid \bigcap_{\ell\in S_2} \bar{A}_\ell] \\ &= \Pr[A_i] \leq x_i \prod_{\{i,j\}\in E} (1 - x_j) \ . \end{aligned} \tag{3}$$

The denominator, on the other hand, can be bounded by the induction hypothesis. Indeed, suppose $S_1 = \{j_1, j_2, \ldots, j_r\}$. If $r = 0$, then the denominator is 1, and (1) follows. Otherwise

$$\begin{aligned} \Pr[\bigcap_{j\in S_1} \bar{A}_j \mid \bigcap_{\ell\in S_2} \bar{A}_\ell] &= \prod_{k=1}^{r} \left(1 - \Pr[A_{j_k} \mid \bar{A}_{j_1} \cap \ldots \cap \bar{A}_{j_{k-1}} \cap (\bigcap_{\ell\in S_2} \bar{A}_\ell)]\right) \\ &\geq \prod_{k=1}^{r} (1 - x_{j_k}) \geq \prod_{(i,j)\in E} (1 - x_j) \ . \end{aligned} \tag{4}$$

Substituting (3) and (4) into (2), we conclude that $\Pr[A_i \mid \bigcap_{j\in S} \bar{A}_j] \leq x_i$, completing the proof of the induction.

The assertion of Lemma 4.6 now follows easily, as

$$\begin{aligned} \Pr[\bigcap_{i=1}^{n} \bar{A}_i] &= \prod_{i=1}^{n} \left(1 - \Pr[A_i \mid \bigcap_{j=1}^{i-1} \bar{A}_j]\right) \\ &\geq \prod_{i=1}^{n} (1 - x_i) \ . \end{aligned}$$

□

There exists a simpler version of the Local Lemma that suffices for many cases. In this version it is assumed that the maximum degree of the dependency graph is d.

Lemma 4.7 (Lovász Local Lemma, symmetric case) *Let $A_1, \ldots, A_n$ be events in an arbitrary random experiment and let $G = (V, E)$ be a dependency graph of these events. If $\Pr[A_i] \leq p$ for all $1 \leq i \leq n$, G has a maximum degree of d and*

$$ep(d + 1) \leq 1 \tag{5}$$

then $\Pr[\bigcap_{i=1}^{n} \bar{A}_i] > 0$.

Proof. The proof is left as an exercise. □

Let us compare now Lemma 4.7 with Lemma 4.4. In Lemma 4.4 we would need the condition $p < 1/n$ for the n events, whereas for Lemma 4.7 it is only required that $p \leq 1/(e(d + 1))$ *independent of* n. For this reason, Lemma 4.6 is called the Lovász *Local* Lemma. The constant e in inequality (5) of Lemma 4.7 is, in general, best possible [3]. In the following we present various applications of the local lemma. For these applications no alternative proofs are known to the local lemma.

4.3 Examples

Coloring of hypergraphs

A *hypergraph* is a tuple $H = (V, E)$, that consists of a set V and a set of hyperedges E. A hyperedge can be an arbitrary subset of V. Hypergraphs can therefore be considered as a generalization of regular graphs.

A hypergraph H is called *2-colorable* if there is a way to color the nodes by using two colors so that for every hyperedge there is at least one node of each color. If this is not the case for a hyperedge e, then we call e *monochromatic*. The Lovász Local Lemma allows us to formulate the following statement about the 2-colorability of hypergraphs.

Theorem 4.8 *Let H be an arbitrary hypergraph with the property that every hyperedge contains at least k nodes and overlaps (i.e. has a node in common) with at most d other hyperedges. If $e(d+1) \leq 2^{k-1}$, then H is 2-colorable.*

Proof. We consider the random experiment where each node selects one of the two colors uniformly at random and independently from the other nodes. For each hyperedge $e \in E$, let A_e be the event that is true if and only if e is monochromatic. Obviously, it holds that

$$\Pr[A_e] = 2 \cdot \frac{1}{2^{|e|}} \leq \frac{1}{2^{k-1}}$$

Furthermore, each event A_e depends only on the events $A_{e'}$ with $e \cap e' \neq \emptyset$. I.e., we can create a dependency graph of the events with degree at most d. If $e(d+1) \leq 2^{k-1}$, then according to the Local Lemma, there exists an outcome of our random experiment so that no event A_e is true, i.e., no hyperedge is monochromatic. Hence, for $e(d+1) \leq 2^{k-1}$ the hypergraph is 2-colorable. □

Satisfiability

In the satisfiability problem *SAT* we have a boolean formula Φ in CNF, and the problem is to determine if Φ is satisfiable or not. The Local Lemma enables us to prove the following statement.

Theorem 4.9 *Let Φ be an arbitrary boolean formula in CNF where each clause contains at least k literals, and where every clause has variables in common with at most d other clauses. If $e(d+1) \leq 2^k$, then Φ is satisfiable.*

Proof. We consider the random experiment where every boolean variable x in Φ is set to true or false uniformly at random and independently of the other variables. For every clause C in Φ let A_C be the event that is true if and only if C is not satisfied. Obviously, it holds that $\Pr[A_C] = (1/2)^{|C|} \leq 1/2^k$. Furthermore, each event A_C depends only on the events $A_{C'}$ where C' has a variable in common with C. That means that we can define a dependence graph of the events A_C with degree at most d. If $e(d+1) \leq 2^k$, then according to the Local Lemma there exists an outcome of our random experiment so that no event A_C is true, i.e., all clauses are satisfied. Hence, for $e(d+1) \leq 2^k$ the formula Φ is satisfiable. □

4.4 An efficient LLL algorithm

Let $\mathcal{P}$ be a finite collection of mutually independent random variables in a fixed probability space Ω. We will consider events A that are determined by the values of some subset $S \subseteq \mathcal{P}$ of these variables. Clearly, if A is determined by $\mathcal{P}$, then there is a unique minimal subset $S \subseteq \mathcal{P}$ that determines A. We denote this set of variables by $vbl(A)$. Let $\mathcal{A}$ be a finite family of events in Ω determined by $\mathcal{P}$. We define the dependency graph $G = G_{\mathcal{A}}$ for $\mathcal{A}$ to be the graph on vertex set $\mathcal{A}$ with an edge between events $A, B \in \mathcal{A}$ if $A \neq B$ and $vbl(A) \cap vbl(B) \neq \emptyset$. For $A \in \mathcal{A}$ we write $\Gamma(A) = \Gamma_{\mathcal{A}}(A)$ for the neighborhood of A in G. Note that $\Gamma(A)$ satisfies the requirement on a dependency graph as A is determined by the variables in $vbl(A)$ and the events in $A \setminus (\{A\} \cup \Gamma(A))$ are determined by the rest of the variables in $\mathcal{P}$.

Given the family $\mathcal{A}$ of events as above our goal is to efficiently find such a solution that violates all events in $\mathcal{A}$. The algorithm we suggest (see Figure 3) is as simple and natural as it can get: We start with a random outcome in Ω and maintain a value v_P of each variable $P \in \mathcal{P}$. We check whether some event in $\mathcal{A}$ is still satisfied. If so, we arbitrarily pick a satisfied event $A \in \mathcal{A}$ and sample new random values for the variables in $vbl(A)$ that A depends on, each one independently and according to its probability distribution, while not changing the values of the variables in $\mathcal{P} \setminus vbl(A)$. We call this a *resampling* of the event A. We continue resampling true events until no such event exists anymore. We will prove that this simple algorithm quickly terminates, that is, it quickly reaches a set of values for the variables that violate all of the events in $\mathcal{A}$ if the conditions of the Local Lemma are met.

The efficiency of the method clearly depends on whether random values for each variable can be efficiently sampled and whether they can be efficiently checked against the given events. This is the case for almost all known applications of the lemma. We will analyze the efficiency of the algorithm in terms of the expected number of times an event $A \in \mathcal{A}$ is resampled.

Algorithm FastLLL($\mathcal{P}, \mathcal{A}$):

for all $P \in \mathcal{P}$ **do**
  $v_P :=$ a random value for P
while $\exists A \in \mathcal{A}$: A is satisfied for $(P = v_P)_{P \in \mathcal{P}}$ **do**
  pick an arbitrary satisfied event $A \in \mathcal{A}$
  for all $P \in vbl(A)$ **do**
    $v_P :=$ a new random value for P
return $(v_P)_{P \in \mathcal{P}}$

Figure 3: The fast LLL algorithm.

Theorem 4.10 (Moser and Tardos [2]) *Let $\mathcal{P}$ be a finite set of mutually independent random variables in a probability space. Let $\mathcal{A}$ be a finite set of events determined by these variables. If there exists a mapping $x : \mathcal{A} \to (0, 1)$ such that*

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

then there exist values for the variables in $\mathcal{P}$ so that all events in $\mathcal{A}$ are violated. Moreover, the randomized algorithm described above resamples an event $A \in \mathcal{A}$ at most an expected $x(A)/(1 - x(A))$ times before it finds such a solution. Thus, the expected total number of resampling steps is at most $\sum_{A \in \mathcal{A}} \frac{x(A)}{1-x(A)}$.

Proof. First of all, note that the decision which satisfied event $A \in \mathcal{A}$ to correct in each step can be taken completely arbitrarily. Let us fix any (deterministic or randomized) procedure for this selection to make the algorithm and the expected values we consider well defined. The selection method that the algorithm uses does not matter for our analysis.

We need to record an accurate journal of what the algorithm does. Let $L : \mathbb{N} \to \mathcal{A}$ list the events as they have been selected for resampling in each step. If the algorithm terminates, L is a partial mapping and defined only up to the given total number of steps carried out.We call L the *execution log* (or simply *log*) of the algorithm. With a fixed selection discipline as described above, L is now a random variable determined by the random choices of the algorithm.

Recall that the dependency graph G is a graph on the vertex set $\mathcal{A}$ where two distinct events $A, A' \in \mathcal{A}$ are connected if $vbl(A) \cap vbl(A') \neq \emptyset$ and that $\Gamma(A)$ denotes the neighborhood of the vertex A in G. We also use here the *inclusive neighborhood* $\Gamma^+(A) := \Gamma(A) \cup \{A\}$ of a vertex A.

A *witness tree* $\tau = (T, \sigma_T)$ is a finite rooted tree T together with a labeling $\sigma_T : V(T) \to \mathcal{A}$ of its vertices with events such that the children of a vertex $u \in V(T)$ receive labels from $\Gamma^+(\sigma_T(u))$. If distinct children of the same vertex always receive distinct labels, we call the witness tree *proper*. To shorten notation, we will write $V(\tau) := V(T)$ and for any $v \in V(\tau)$, we write $[v] := \sigma_T(v)$. Given the log L, we will now associate with each resampling step t carried out a witness tree $\tau_L(t)$ that can serve as a 'justification' for the necessity of that correction step. Let us define $\tau_L^{(t)}(t)$ to be an isolated root vertex labeled $L(t)$. Then, going backwards through the log, for each $i = t - 1, t - 2, \ldots, 1$, we distinguish two cases. If there is a vertex $v \in \tau_L^{(i+1)}(t)$ such that $L(i) \in \Gamma^+([v])$, then we choose among all such vertices the one having the maximum distance from the root and attach a new child vertex u to v that we label $L(i)$, thereby obtaining the tree $\tau_L^{(i)}(t)$. In the selection of the maximum distance vertex, we break ties arbitrarily. If there is no vertex $v \in \tau_L^{(i+1)}(t)$ such that $L(i) \in \Gamma^+([v])$, then we skip time step i and simply define $\tau_L^{(i)}(t) := \tau_L^{(i+1)}(t)$. Finally let $\tau_L(t) := \tau_L^{(1)}(t)$. See an example in Figure **??**. We say that the witness tree τ *occurs* in the log L if there exists a $t \in \mathbb{N}$ such that $\tau_L(t) = \tau$.

Lemma 4.11 *Let τ be a fixed witness tree and L be the (random) log produced by the algorithm.*

(1) If τ occurs in L, then τ is proper.

(2) The probability that τ appears in L is at most $\prod_{v \in V(\tau)} \Pr[[v]]$.

Proof. Assume τ occurs in the log L, so we have $\tau_L(t) = \tau$ for some $t \in \mathbb{N}$. For a vertex $v \in V(\tau)$ let $d(v)$ denote the *depth* of vertex v, that is, its distance from the root, and let $q(v)$ stand for the step of the algorithm constructing $\tau_L(t)$ in which v was attached, that is, $q(v)$ is the largest value q with v contained in $\tau_L^{(q)}(t)$.

First, note that if $q(u) < q(v)$ for vertices $u, v \in V(\tau)$ and $vbl([u])$ and $vbl([v])$ are not disjoint, then $d(u) > d(v)$. Indeed, when adding the vertex u to $\tau_L^{(q(u)+1)}(t)$ we attach it to v or to another vertex of equal or greater depth. As a consequence, observe that for any two vertices $u, v \in V(\tau)$ at the same depth $d(v) = d(u)$, $[u]$ and $[v]$ do not depend on any common variables, that is, the labels in every level of τ form an independent set in G. In particular τ must be proper, establishing claim (1).

Consider the following procedure that we call a *τ-check*: In order of decreasing depth (e.g., reversed breadth first search order) visit the vertices of τ and for a vertex v take a set of random values of the variables in $vbl([v])$ (according to their distribution, independent of possible earlier evaluations) and check if the resulting evaluation satisfies $[v]$. We say that the τ-check *passes* if all events were satisfied when checked.

Trivially, the τ-check passes with probability exactly $\prod_{v \in V(\tau)} \Pr[[v]]$. The lemma follows from the observation that whenever τ occurs in the log and we run the τ-check on the same random source it passes, which would imply that $\Pr[\tau$ occurs in the log$] \leq \Pr[\tau$-check passes$]$. For this coupling argument, we have to specify explicitly how the algorithms use the random source. We assume that for each variable $P \in \mathcal{P}$ the random source produces an infinite list of independent random samples $P^{(0)}, P^{(1)}, \ldots$, and whenever either algorithm calls for a new random sample of P, we pick the next unused value from this sequence.

Having assumed that $\tau = \tau_L(t)$, we need to prove that the τ-check passes, that is, when it considers a vertex $v \in V(\tau)$ and takes random samples of the variables in $vbl([v])$, the resulting values satisfy $[v]$. Let us fix the vertex v and for $P \in vbl([v])$ let $S(P)$ be the set of vertices $w \in V(\tau)$ with $d(w) > d(v)$ and $P \in vbl([w])$. When the τ-check considers the vertex v and samples P, the random source gives $P^{(|S(P)|)}$. This is because the τ-check visits the vertices in order of decreasing depth and among the vertices with depth equal to $d(v)$ only the label of v depends on P, so before the τ-check considers v, it had sampled P exactly when it was considering the vertices in $S(P)$.

In step $q(v)$, our algorithm chooses the event $[v]$ for resampling, so $[v]$ must be satisfied before this resampling. We claim that for $P \in vbl([v])$ the current value of the variable P is $P^{(|S(P)|)}$ at this time. Indeed, P was sampled at the beginning of the algorithm and then at the steps $q(w) < q(v)$ for $w \in S(P)$. As the τ-check has these exact same values for the variables in $vbl([v])$ when considering v it also must find that $[v]$ is satisfied, proving (2). □

Let L be the log of the execution of our algorithm. For any event $A \in \mathcal{A}$ let us denote by N_A the random variable that counts how many times the event A is resampled during the execution of our algorithm, that is, the number of time steps t with $L(t) = A$. Let t_i denote the i-th such time step. Let $\mathcal{T}_A$ denote the set of all proper witness trees having the root labeled A. As we have previously demonstrated, $\tau_L(t_i) \in \mathcal{T}_A$ for all i. Moreover, again for each i, the tree $\tau_L(t_i)$ contains exactly i vertices labeled A and thus $\tau_L(t_i) \neq \tau_L(t_j)$ unless $i = j$. From this we conclude that N_A not only counts the number of occurrences of A in the log, but also coincides with the number of distinct proper witness trees occurring in C that have their root labeled A. Thus,

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

where the binary random variable I_τ is 1 if and only if τ occurs in the log. Therefore, by Lemma 4.11,

$$\begin{aligned} \mathbb{E}[N_A] &= \mathbb{E}\left[\sum_{\tau \in \mathcal{T}_A} I_\tau\right] = \sum_{\tau \in \mathcal{T}_A} \mathbb{E}[I_\tau] \\ &= \sum_{\tau \in \mathcal{T}_A} \Pr[\tau \text{ occurs in the log}] \\ &\leq \sum_{\tau \in \mathcal{T}_A} \prod_{v \in V(\tau)} \Pr[[v]] \,. \end{aligned}$$

To connect this bound to the Lovász Local Lemma, we will consider an appropriate branching process.

Let us fix an event $A \in \mathcal{A}$ and consider the following multitype Galton-Watson branching process for generating a proper witness tree having its root labeled A. In the first round, we produce a singleton vertex labeled A. Then, in each subsequent round, we consider each vertex v produced in the previous round independently and, again independently, for each event $B \in \Gamma^+([v])$ identical or adjacent to $[v]$ in the dependency graph, we add to v a child node carrying the label B with probability $x(B)$ or skip that label with probability $1 - x(B)$. All these choices are independent. The process continues until it dies out naturally because no new vertices are born in some round (depending on the probabilities used, there is, of course, the possibility that this never happens).

Let $x'(B) := x(B)\prod_{C\in\Gamma(B)}(1-x(C))$. For the probability that the described Galton-Watson process yields a prescribed proper witness tree we obtain the following.

Lemma 4.12 *Let τ be a fixed proper witness tree with its root vertex labeled A. The probability p_τ that the Galton-Watson process described above yields exactly the tree τ is*

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

Proof. For a vertex $v \in V(\tau)$ we denote by $W_v \subseteq \Gamma^+([v])$ the set of inclusive neighbors of $[v]$ that do not occur as a label of some child node of v. Then clearly, the probability that the Galton-Watson process produces exactly τ is given by

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

where the leading factor accounts for the fact that the root is always born. In order to get rid of the W_v, we can rewrite this expression in an obvious way to obtain

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

where again we have to account for the root separately. Replacing inclusive by exclusive neighborhoods, this simplifies to

$$p_\tau = \frac{1-x(A)}{x(A)} \prod_{v\in V_\tau} \left(x([v]) \prod_{u\in\Gamma([v])} (1-x([u])) \right) = \frac{1-x(A)}{x(A)} \prod_{v\in V_\tau} x'([v])$$

and concludes the argument. □

Recall that

$$\mathbb{E}[N_A] \ \leq\ \sum_{\tau\in\mathcal{T}_A} \prod_{v\in V(\tau)} \Pr[[v]] \ \leq\ \sum_{\tau\in\mathcal{T}_A} \prod_{v\in V(\tau)} x'([v])$$

where the second equality follows from the assumption in Theorem 4.10. We further have

$$\mathbb{E}[N_A] \leq \sum_{\tau\in\mathcal{T}_A} \prod_{v\in V(\tau)} x'([v]) \ =\ \frac{x(A)}{1-x(A)} \sum_{\tau\in\mathcal{T}_A} p_\tau \ \leq\ \frac{x(A)}{1-x(A)}$$

where the equality comes from Lemma 4.12, while the last inequality follows from the fact that the Galton-Watson process produces exactly one tree at a time (but not necessarily one from $\mathcal{T}_A$ since it might also grow infinite). This concludes the proof of Theorem 4.10. □

References

[1] P. Erdős and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In A. Hajnal, R. Rado, and V.T. Sós, editors, *Infinite and Finite Sets (to Paul Erdős on his 60th birthday)*, pages 609–627. North-Holland, Amsterdam, 1975.

[2] R. Moser and G. Tardos. A constructive proof of the general Lovász Local Lemma. *Journal of the ACM*, 57(2), 2010.

[3] Shearer. On a problem of Spencer. *Combinatorica*, 5:241–245, 1985.