DIRECTED LOVÁSZ LOCAL LEMMA AND SHEARER'S LEMMA*

LEFTERIS KIROUSIS[1,3], JOHN LIVIERATOS[1], AND KOSTAS I. PSAROMILIGKOS[2,4]

Abstract. Moser and Tardos (2010) gave an algorithmic proof of the lopsided Lovász local lemma (LLL) in the variable framework, where each of the undesirable events is assumed to depend on a subset of a collection of independent random variables. For the proof, they define a notion of a lopsided dependency between the events suitable for this framework. In this work, we strengthen this notion, defining a novel *directed* notion of dependency and prove the LLL for the corresponding graph. We show that this graph can be strictly sparser (thus the sufficient condition for the LLL weaker) compared with graphs that correspond to other extant lopsided versions of dependency. Thus, in a sense, we address the problem "find other simple local conditions for the constraints (in the variable framework) that advantageously translate to some abstract lopsided condition" posed by Szegedy (2013). We also give an example where our notion of dependency graph gives better results than the classical Shearer lemma. Finally, we prove Shearer's lemma for the dependency graph we define. For the proofs, we perform a direct probabilistic analysis that yields an exponentially small upper bound for the probability of the algorithm that searches for the desired assignment to the variables not to return a correct answer within n steps. In contrast, the method of proof that became known as the entropic method, gives an estimate of only the expectation of the number of steps until the algorithm returns a correct answer, unless the probabilities are tinkered with.

1. Inroduction

The Lovász Local Lemma (LLL) was originally stated and proved in 1975 by Erdős and Lovász [4]. Its original *symmetric* form states that given events $E_1, \ldots, E_m$ in a common probability space, if every event depends on at most d others, and if the probabilities of all are bounded by $1/(4d)$, then

$$Pr\left[\bigwedge_{j=1}^{m} \bar{E}_j\right] > 0,$$

and therefore there exists at least one point in the space where none of the events occurs ($\bar{E}$ denotes the complement of E).

The *asymmetric* version entails an undirected *dependency graph*, i.e. a graph with vertices $j = 1, \ldots, m$ corresponding to the events $E_1, \ldots, E_m$ so that for all

[1]Department of Mathematics, National and Kapodistrian University of Athens
[2]Department of Mathematics, University of Chicago. Onassis Scholar
E-mail addresses: {lkirousis,jlivier89}@math.uoa.gr, kostaspsa@math.uchicago.edu.
[3]Research partially supported by TIN2017-86727-C2-1-R, GRAMM.
[4]Research carried out while an undergraduate student at the Department of Mathematics of the National and Kapodistrian University of Athens.
*This is a revised version of [14] with a correct statement and proof of theorem 1a.

j, E_j is mutually independent from the set of events corresponding to vertices not connected with j. The condition that in this case guarantees that $Pr[\bigwedge_{j=1}^{m} \bar{E}_j] > 0$ (and therefore that there exists at least one point where none of the events occurs) is:

(Asym) for every E_j there is a $\chi_j \in (0,1)$ such that $Pr[E_j] \leq \chi_j \prod_{i \in N_j} (1 - \chi_i)$,

where N_j is the neighborhood of vertex j in the dependency graph.

Improvements can be obtained by considering, possibly directed, sparser graphs than the dependency graph, that correspond to stronger notions of dependency. For a classic example, the *lopsided* version (LLLL) by Erdős and Spencer [5] entails a *directed* graph with vertices corresponding to the events such that for all E_j and for all $I \subseteq \{1, \ldots, m\} \setminus (\Gamma_j \cup \{j\})$ we have that

$$\Pr\left[E_j \mid \bigcap_{i \in I} \bar{E}_i\right] \leq \Pr\left[E_j\right], \tag{1}$$

where Γ_j is the set of vertices connected with j with an edge originating from j. Such graphs are known as *lopsidependency graphs*. The sufficient condition in this case that guarantees that the undesirable events can be avoided reads:

(Lop) for every E_j there is a $\chi_j \in (0,1)$ such that $Pr[E_j] \leq \chi_j \prod_{i \in \Gamma_j} (1 - \chi_i)$.

With respect to ordinary (not lopsided) dependency graphs, a sufficient *but also necessary* condition to avoid all events was given by Shearer [19]. It reads:

(Shear) For all $I \in I(G)$, $q_I(G, \bar{p}) := \sum_{J \in I(G): I \subseteq J} (-1)^{|J \setminus I|} \prod_{j \in J} p_j > 0$,

where $I(G)$ is the set of *independent sets* of G and $\bar{p} = (p_1, \ldots, p_m)$ is the vector of probabilities of the events.

By considering other graphs, and the corresponding to them Condition (Shear), variants of the Shearer lemma are obtained. These variants are in general only sufficient, however they apply to sparser dependency graphs. For example, by proving the sufficiency of Condition (Shear) when applied to the lopsidependency graph of Erdős and Spencer [5], we get Shearer's lemma for lopsidependency graphs (actually, for Shearer's lemma in this case it is the underlying undirected graph of the lopsidependency graph that is considered).

Apart from existence, research has been focused also in "efficiently" finding a point in the probability space such that no undesirable event occurs. After several partially successful attempts that expand over more than three decades, Moser [16] in 2009, initially only for the symmetric LLL, gave an extremely simple randomized algorithm that if and when it stops, it certainly produces a point where all events are avoided. Soon after Moser and Tardos [17] expanded this approach to more general versions including the lopsided one. The algorithms were given in the variable framework, where the space is assumed to be the product space of independent random variables $X_1, \ldots, X_l$, and each event is assumed to depend on a subset of them, called its *scope*. Their algorithm just samples iteratively the variables of occurring events, until all the events cease to occur. For the analysis,

they estimate the *expectation* of the number of times each event will be resampled in a given execution of the algorithm by counting "tree-like" structures they call *witness trees* and by estimating the probability that such a tree occurs in the log of the algorithm's execution. This approach became known as the *"entropy compression"* method (see [21] for a short exposition). For the proof of the lopsided LLL, Moser and Tardos [17] defined an *undirected* lopsidependency graph suitable for the variable framework.

In the variable framework, Harris [9] gave a weaker version for the Lop condition, entailing the notion of *orderability*, which takes advantage of the way events are related based on the different values of the variables they depend on. He works with the Moser-Tardos notion of lopsidependency graph and he proves that his weaker sufficient condition can yield stronger results than the classical Shearer's lemma. Also, recently He et al. [12] gave a necessary and sufficient condition for LLL in the variable framework, but for the dependency graph where two events are connected if their scopes share at least one variable. We focus, on the contrary, in working with sparser graphs.

There are numerous applications of the algorithmic versions of both LLL and its lopsided version, even for problems that do not originate from purely combinatorial issues. For example, for the non-lopsided versions let us mention the problem of *covering arrays*, a problem closely related with software and hardware interaction testing. The objective is to find the *minimum* number N, expressed as a function k, t, v, such that there exists an $N \times k$ array A, with elements taken from a set Σ of cardinality $v \geq 2$, so that every $N \times t$ sub-array of A contains as one of its rows every element $x \in \Sigma^t$. Sarkar and Colbourn [18] improve on known upper bounds for N, by using LLL. Notably, they also provide an algorithm that constructs an $N \times k$ array with the above properties, by using a variant of the Moser-Tardos algorithm [17]. As for the lopsided version let us mention, e.g., the work of Harris and Srinivasan [10] who apply it in the setting of *permutations*, where the undesirable events are defined over permutations π_k of $\{1, \ldots, n_k\}$, $k = 1, \ldots, N$.

The lopsided LLL was generalized to the framework of arbitrary probability spaces by Harvey and Vondrák [11], by means of a machinery that they called "resampling oracles". They introduced, in the framework of arbitrary probability spaces, *directed* lopsidependency graphs they called *lopsided association* graphs. They proved that a graph is a lopsided association graph if and only if it is a graph along the edges of which resampling oracles can be applied. In the generalized framework and based on their lopsided association condition, they algorithmically proved LLLL. In the same framework, they also proved Shearer's lemma (in this respect, see also the work of Kolipaka and Szegedy [15]). Finally, Achlioptas and Iliopoulos [2] have introduced a powerful abstraction for algorithmic LLL, which is inherently directed and they prove the lopsided LLL in this framework.

Let us mention here that Szegedy [20] gives a comprehensive survey of the LLL, that contains many of the algorithmic results.

Our results. We work exclusively in the variable framework. First, we define a novel relation of *directed* dependency that we call *d-dependency*, which is stronger than that of Moser and Tardos [17]. We also show that this relation may generate a strictly sparser dependency graph than other extant ones (and so it leads to weaker sufficient conditions for LLL).

We then algorithmically prove that the Lop condition suffices to avoid all events when applied to the graph defined by our notion of d-dependency. Thus, in a sense we address the problem "find other simple local conditions for the constraints (in the variable framework) that advantageously translate to some abstract lopsided condition" posed by Szegedy [20].

Our approach is based on Moser's original algorithm [16], which, upon resampling an event, checks its neighborhood for other occurring events. Like in Giotis et al. [6], we use a witness structure (forest) to depict the execution of our algorithms that, in contrast with those of the "Moser-Tardos-like" proofs, grows "forward in time", meaning that it is constructed as an execution moves on. Taking advantage of this structure, we express the probability that the algorithm executes for at least n rounds by a *recurrence relation*. We subsequently solve this recurrence by specialized analytical means, and prove that it diminishes exponentially fast in n. Specifically, we employ the result of Bender and Richmond [3] on the multivariable Lagrange inversion formula. A positive aspect of this approach is that it provides an exponentially small bound for the probability of the algorithm to last for at least n steps (including to run intermittently) before it returns the desired result, in contrast to the entropic method that estimates the expected time of the algorithm to return a correct answer. We also note that, in contrast to Harvey and Vondrák [11], our proof for the directed LLL is independent of the one for Shearer's lemma.

Finally, although we show that our notion of dependency can give stronger results than the classical Shearer's lemma, we use our forward approach to prove this lemma for the d-dependency graph. An algorithmic proof for Shearer's lemma for the ordinary dependency graph was first provided by Kolipaka and Szegedy [15], who actually gave a proof for the general case of arbitrary probability spaces. The latter result was strengthened by Harvey and Vondrák [11] for their notion of association graphs, again for general probability spaces. Our result is for the variable framework, but for the possibly sparser graph of d-dependency. Also, we give again a direct computation of an exponentially small upper bound to the probability of the algorithm to last for at least n steps. To carry out the computations in our forward approach, we employ Gelfand's formula for the spectral radius of a matrix (see [13]).

Note that, as is the case with all extant algorithmic approaches to the LLL, both the number of events m and the number of random variables l are assumed to be constants. Complexity considerations are made with respect to the number of steps the algorithms last.

2. d-Dependency and a weak version of first result

For everything that follows, we assume that $Pr[E_j] < 1$, $j = 1, ..., m$, lest there is no way to avoid all the events.

We begin by defining the following *asymmetric* relation between two events.

Definition 1. *Given events E_i, E_j, we say that E_i is* d-dependent *on E_j if:*

1. *there exists an assignment α to the random variables under which E_j occurs and E_i does not, and*
2. *the values of the variables in $sc(E_j)$, the scope of E_j, can be changed so that E_i occurs and E_j ceases occurring.*

Intuitively, E_i is d-dependent on E_j if it is possible that some *successful* attempt to avoid the occurrence of E_j may end up with E_i occurring, although initially it did not.

The binary relation of d-dependency defines a simple (no loops or multiple edges) *directed* graph $G = (V, E)$, the *d-dependency graph* of events $E_1, ..., E_m$, where $V = \{1, ..., m\}$ and $E = \{(j, i) \mid E_i \text{ is } d\text{-dependent on } E_j\}$. Trivially, this graph is *sparser* than the usual dependency graph in the variable framework, where there is an edge between events with *intersecting scopes*.

For $i = 1, ..., m$, let Γ_j be the outwards neighborhood of the event E_j in the d-dependency graph, i.e. $\Gamma_j = \{i \mid E_i \text{ is } d\text{-dependent on } E_j\}$. The notion of d-dependency was inspired by the following *symmetric* relation of Moser and Tardos [17], which will sometimes be referred to as *MT-dependency*:

Definition 2 (Moser and Tardos [17])**.** *Let E_i, E_j be events, $i, j \in \{1, ..., m\}$. We say that E_i, E_j are lopsidependent if there exist two assignments α, β, that differ only on variables in $sc(E_i) \cap sc(E_j)$, such that:*

(1) *α makes E_i occur and β makes E_j occur and*
(2) *either $\bar{E}_i$ occurs under β or $\bar{E}_j$ occurs under α.*

Moser and Tardos [17] gave an algorithmic proof that if the condition (Asym) holds for an (undirected) dependency graph with respect to the notion of Definition 2, then the undesirable events can be avoided.

The following claim is straightforward:

Claim 1. *If E_i is d-dependent on E_j, in the sense of Definition 1, then E_i and E_j are MT-dependent, in the sense of Definition 2.*

Proof. Suppose that under $\alpha = (a_1, \ldots, a_l)$, $\bar{E}_i$ and E_j occur and that we can change the values of the variables in $\mathrm{sc}(E_j)$ to get and assignment $\beta = (b_1, \ldots, b_l)$ under which E_i and $\bar{E}_j$ occur. Let now $\gamma = (c_1, \ldots, c_l)$ be such that:

$$c_i = \begin{cases} b_i, & \text{if } X_i \in \mathrm{sc}(E_i) \cap \mathrm{sc}(E_j) \\ a_i, & \text{else,} \end{cases} \tag{2}$$

for $i = 1, \ldots, l$.

Since E_i is not affected by changes in $\mathrm{sc}(E_j) \setminus \mathrm{sc}(E_i)$, E_i occurs under γ and thus E_i, E_j are lopsidependent. □

A weak version of our result (given for comparison with other extant ones) reads:

Theorem 1 (Directed Lovász local lemma)**.** *Suppose that there exist numbers $\chi_1, \chi_2, \ldots, \chi_m \in (0, 1)$, such that*

$$\Pr(E_j) \leq \chi_j \prod_{i \in \Gamma_j} (1 - \chi_i), \tag{DirLop}$$

for all $j \in \{1, \ldots, m\}$, where Γ_j denotes the neighborhood of E_j in the d-dependency graph. Then,

$$\Pr\left[\bigwedge_{j=1}^{m} \bar{E}_j\right] > 0.$$

Actually, we prove below an algorithmic version (Theorem 1a) of the existential Theorem 1, where we give exponentially small estimates of the probability of the algorithm not producing the desired results within n steps.

In the following example, we show that the d-dependency graph can be strictly sparser than other dependency graphs that have been used in the literature.

Example 1. *Suppose we have $n \geq 3$ independent* Bernoulli trials $X_1, X_2, \ldots, X_n$, *where $X_i = 1$ denotes the event that the i-th such trial is* successful, *$i = 1, \ldots, n$. Consider also the n "undesirable events":*

$$E_j = \{X_j = 1 \vee X_{j+1} = 1\},$$

where $X_{n+1} = X_1$ and assume also that each of the Bernoulli trials succeeds with probability $p \in [0, 1)$. Thus:

$$Pr[E_j] = p + (1 - p)p = 2p - p^2.$$

We begin by showing that for any two distinct *E_i, E_j, neither one of them is d-dependent on the other. Without loss of generality, let $i = 1$ and $j = 2$.*

Since $sc(E_1) \cup sc(E_2) = \{X_1, X_2, X_3\}$, both E_1 and E_2 are affected only by the first three coordinates of an assignment of values. We will thus restrict the assignmets to those coordinates.

Suppose E_1 and $\bar{E}_2$ occur under an assignment α. Then, $\alpha = (1, 0, 0)$ and there is no way to change the first two coordinates in order for $\bar{E}_1$ and E_2 to occur. Thus E_2 is not d-dependent on E_1. Furthermore, for $\bar{E}_1$ and E_2 to occur under an assignment β, $\beta = (0, 0, 1)$ and there is no way to change the last two coordinates of β in order for E_1 and $\bar{E}_2$ to occur. Thus E_1 is not d-dependent on E_2.

We can analogously prove the same things for all pairs of E_j, E_{j+1}, $j = 1, \ldots, n$, where $E_{n+1} := E_1$. Furthermore, it is easy to see that for any $i, j \in \{1, \ldots, n\}$: $i < j$ and $j \neq i + 1$, neither E_j is d-dependent on E_i nor vice versa, since E_i, E_j have no common variables they depend on. Thus, the d-dependency graph of the events has no edges and it is trivial to observe that we can avoid all the events if and only if $p < 1$.

On the other hand, consider assignments $\gamma = (1, 0, 0)$ and $\delta = (1, 1, 0)$. Under γ, $E_1, \bar{E}_2$ occur, under δ E_2 occurs and the assignments differ only on $X_2 \in sc(E_1) \cap sc(E_2)$. By Definition 2, E_1 and E_2 are lopsidependent.

Given the above, it is not difficult to see that the underlying undirected graph of the lopsidependency graph defined by the dependency relation of Definition 2, is the cycle C_n on n vertices.

Interestingly, by interpreting Harvey and Vondrák's [11] definition of resampling oracles in the variable setting as the resampling of the variables in the scope of an event, we get a directed graph, *whose underlying graph is again C_n. The same is true for the directed "potential causality graph" of Achliptas and Illiopoulos [2], where the flaws correspond to events and where we interpret an arc $f \to g$ between flaws f, g, again in the variable framework, as being able to obtain flaw g by resampling the variables in the scope of flaw f.*

In the sequel, we assume $n = 3$ in order to simplify the example. The corresponding graphs are given in Figures 1–3.

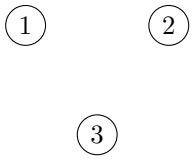

FIGURE 1. d-dependency graph, Definition 1

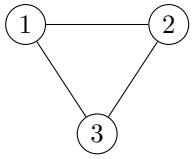

FIGURE 2. MT-dependency graph, Definition 2

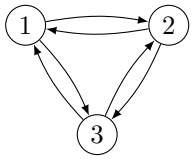

FIGURE 3. Lopsided association [11], and potential causality graph [1], when interpreted in the variable framework in the natural way.

Now, it is not difficult to see that the (Asym) *condition applied to the graph corresponding to the dependency of Definition 2 requires for* $\chi_1, \chi_2, \chi_3 \in (0,1)$ *such that:*

$$\begin{aligned} Pr[E_1] \leq & \chi_1(1-\chi_2)(1-\chi_3), \\ Pr[E_2] \leq & \chi_2(1-\chi_1)(1-\chi_3), \\ Pr[E_3] \leq & \chi_3(1-\chi_1)(1-\chi_2). \end{aligned}$$

Thus, for the Moser and Tardos lopsided LLL to apply, it must hold that:

$$2p - p^2 \leq \chi(1-\chi)^2,$$

where $\chi = \min\{\chi_1, \chi_2, \chi_3\}$. *This is maximized for*

$$\chi = \frac{2^2}{3^3} = \frac{4}{27},$$

thus p *must be* at most 0.077 *(recall that our notion only requires for* p *to be strictly less than one).*

Finally, taking the dependency graph of E_1, E_2 *and* E_3, *where two vertices are connected if their corresponding events' scopes intersect, we get as dependency graph*

the cycle C_3 and henceforth by simple calculations, classical Shearer's lemma requires that:

$$1 - 3(2p - p^2) > 0 \Leftrightarrow p < 0.184,$$

a stronger requirement than the one that suffices to show that the undesirable events can be avoided through our d-dependency notion. However, our version of Shearer's lemma (see Section 4) gives $p < 1$.

Let us note that if we consider the classical definition of a lopsidependency graph by Erdős and Spencer [5], namely a directed one satisfying inequality (1)*, then the graph turns out to be empty as well, since no event has a negative effect on any other, neither the union of any two does on the third.* ◇

We now show that the d-dependency graph is an Erdős-Spencer lopsidependency graph.

Lemma 1. *For any event E_j, $j = 1, ..., m$, let I be a set of indices of events not in $\Gamma_j \cup \{j\}$. Then, it holds that:*

$$\Pr\left[E_j \mid \bigcap_{i \in I} \overline{E_i}\right] \leq \Pr[E_j].$$

Proof. Let $E = \bigcap_{i \in I} \overline{E_i}$. Note that E is not necessarily some of the E_js (in fact if it is, then there is no assignment that avoids all the undesirable events). Now, in order to obtain a contradiction, suppose that:

$$\Pr[E_j \mid E] > \Pr[E_j]$$

or, equivalently, that:

$$\Pr[E_j \cap E] > \Pr[E_j] \cdot \Pr[E].$$

Then, it holds that:

$$\Pr[\bar{E}_j \cap E] = \Pr[E] - \Pr[E_j \cap E] < \Pr[E] - \Pr[E_j] \cdot \Pr[E] = \Pr[E](1 - \Pr[E_j]) = \Pr[\bar{E}_j] \cdot \Pr[E]. \tag{3}$$

Since $i \notin \Gamma_j$, for all $i \in I$, it holds that for any assignment α that makes E_j and E hold, there is no assignment β that differs from α only in $\mathrm{sc}(E_j)$ that makes $\bar{E}$ hold.

To obtain a contradiction, it suffices to show that:

$$\Pr[E_j \mid E] \leq \Pr[E_j] \Leftrightarrow \Pr[E_j \cap E] \leq \Pr[E_j] \cdot \Pr[E].$$

Suppose now $\alpha = (a_1, ..., a_l), \beta = (b_1, ..., b_l)$ are two assignments obtained by independently sampling the random variables twice, once to get α and once to get β. It holds that:

$$\Pr[\underbrace{(\alpha, \beta) : \text{ under } \alpha, \ E_j \cap E \text{ occurs and under } \beta, \ \bar{E}_j \text{occurs}}_{\text{event } S}] = \Pr[E_j \cap E] \cdot \Pr[\bar{E}_j].$$

Let now $\alpha' = (a'_1, ..., a'_l)$, $\beta' = (b'_1, ..., b'_l)$ be two assignments obtained by α, β by swapping values in variables in $\mathrm{sc}(E_j)$:

- $a'_i = b_i$, for all i such that $X_i \in \mathrm{sc}(E_j)$, $a'_i = a_i$ for the rest,
- $b'_i = a_i$, for all i such that $X_i \in \mathrm{sc}(E_j)$ and $b'_i = b_i$ for the rest.

Obviously α', β' are two independent samplings of all variables, since all individual variables were originally sampled independently, and we only changed the positioning of the individual variables. Also, under α', $\bar{E}_j$ occurs. Since none of the E_i's is d-dependent on E_j, E occurs under α'. Also, under b', E_j occurs. Thus, it holds that:

$$\Pr[\underbrace{\text{under } \alpha',\ \bar{E}_j \cap E \text{ occurs and under } \beta',\ E_j \text{ occurs}}_{\text{event } T}] = \Pr[\bar{E}_j \cap E] \cdot \Pr[E_j] < \Pr[\bar{E}_j] \cdot \Pr[E] \cdot \Pr[E_j],$$

where the last inequality holds by (3). Now, by the hypothesis and the construction of α', β', it also holds that S implies T. Thus:

$$\begin{aligned}\Pr[S] \leq \Pr[T] &\Leftrightarrow \Pr[E_j \cap E] \cdot \Pr[\bar{E}_j] \leq \Pr[\bar{E}_j] \cdot \Pr[E] \cdot \Pr[E_j] \\ &\Leftrightarrow \Pr[E_j \cap E] \leq \Pr[E_j] \cdot \Pr[E].\end{aligned}$$

The last inequality provides the contradiction and the proof is complete. □ □

As is the case with all (lopsi-)dependency graphs defined based on notions in the variable framework, the dependency graph defined based on mutual independence of the events can be sparser than the d-dependency graph. Consequently, the same holds with the Erdős-Spencer lopsidependency graph too. We end this section with a simple example that attests to that.

Example 2. *Let X_1 and X_2 be two independent random variables taking values, uniformly at random, in $\{0,1\}$. Let also $E_1 = \{X_1 \neq X_2\}$ and $E_2 = \{X_2 = 0\}$.*

First, observe that E_1 is d-dependent on E_2. Indeed, let $\alpha = (0,0)$. Under α, E_1 does not occur and E_2 does. By changing the value of $X_2 \in sc(E_2)$, we obtain the assignment $\beta = (0,1)$, under which E_1 occurs and E_2 doesn't. That E_2 is d-dependent on E_1 follows by taking the assignment β and changing the value of $X_2 \in sc(E_1)$ to obtain α.

On the other hand, notice that:

$$\Pr[E_1] = \frac{1}{2} = \Pr[E_1 \mid \bar{E}_2]$$

and that:

$$\Pr[E_2] = \frac{1}{2} = \Pr[E_2 \mid \bar{E}_1].$$

Thus E_1 and E_2 are independent. ◇

3. The lopsidependent case

Both approaches by Moser [16] and by Moser and Tardos [17] search for an assignment that avoids the undesirable events by consecutively resampling the variables in the scopes of currently occurring events. In the approach of Moser [16], when choosing the next event whose variables will be resampled, priority is given to the occurring events that belong to the *extended neighborhood* in the dependency graph of the last resampled event (the extended neighborhood of an event E is by definition the set of events sharing a variable with E, with the event E itself included). Thus the failure of the algorithm to return a correct answer within n steps is depicted by a structure, called the witness forest of the algorithm's execution (will be formally defined below). So, in some sense, this approach guarantees that failure to produce results, will create, step after random step, a structure out of randomness,

something that cannot last for long, lest the second principle of thermodynamics is violated. This is, very roughly, the intuition behind the entropic method. However, as we stressed, we analyze the algorithm by direct computations instead of referring to entropy. One key idea throughout this work is to give absolute priority, when searching for the next event to be resampled, to the event itself, if it still occurs, in order to be able to utilize the d-dependency graph of the events.

To be specific, see the pseudocode of M-Algorithm, which successively produces random assignments, by resampling the variables in the scopes of occurring events, until it finds one under which no undesirable event occurs. When the variables in the scope of an occurring event E_j are resampled, the algorithm checks if E_j still occurs (lines 2 and 3 of the Resample routine) and, only in case it does not, looks for occurring events in E_j's neighborhood. Finally, if and when all events in E_j's neighborhood cease occurring, the algorithm looks for still occurring events elsewhere.

Algorithm 1 M-Algorithm.

1: Sample the variables X_i, $i = 1, ..., l$ and let α be the resulting assignment.
2: **while** there exists an event that occurs under the current assignment, let E_j be the least indexed such event and **do**
3: Resample(E_j)
4: **end while**
5: Output current assignment α.

Resample(E_j)

1: Resample the variables in sc(E_j).
2: **if** E_j occurs **then**
3: Resample(E_j)
4: **else**
5: **while** some event whose index is in Γ_j occurs under the current assignment, let E_k be the least indexed such event and **do**
6: Resample(E_k)
7: **end while**
8: **end if**

Obviously, if and when M-Algorithm stops, it produces an assignment to the variables for which none of the events occurs. Our aim now is to bound the probability that this algorithm lasts for at least n steps. We count as a step an execution of the variable resampling command Resample in line 1 of the subroutine Resample.

Everywhere below the asymptotics are with respect to n, the number of steps, whereas the number l of variables and the number m of events are taken to be constants.

We first give some terminology, and then we start with a lemma that essentially guarantees that M-Algorithm makes progress.

A *round* is the duration of any Resample call during an execution of M-Algorithm. Rounds are nested. The number of nested rounds completed coincides with the number of steps the algorithm takes. A Resample call made from line 3 of the main algorithm is a *root call*, while one made from within another call is a *recursive call*.

Lemma 2. *Consider an arbitrary call of* Resample(E_j). *Let $\mathcal{X}_j$ be the set of events that do not occur at the start of this call. Then, if and when this call terminates, all events in $\mathcal{X}_j \cup \{E_j\}$ do not occur.*

Proof. Without loss of generality, say that Resample(E_j) is the root call of Resample, suppose it terminates and that α is the produced assignment of values. Furthermore, suppose that $E_k \in \mathcal{X}_j \cup \{E_j\}$ and that E_k occurs under α.

Let $E_k \in \mathcal{X}_j$. Then, under the assignment at the beginning of the main call, E_k did not occur. Thus, it must be the case that at some point during this call, a resampling of some variables caused E_k to occur. Let Resample(E_s) be the last time E_k became occurring, and thus remained occurring until the end of the main call.

Since E_k did not occur at the beginning of Resample(E_s), there is an assignment of values α such that $E_s, \bar{E}_k$ occur. Furthermore, for the main call to have terminated, Resample(E_s) must have terminated too. For this to happen Resample(E_s) must have exited lines 2 and 3 of its execution. During this time, only variables in $\mathrm{sc}(E_s)$ were resampled and at the end, E_s did not occur anymore. Thus, E_k is in the neighborhood of E_s. But then, by line 5 of the Resample routine, Resample(E_s) couldn't have terminated and thus, neither could the main call. Contradiction.

Thus $E_k = E_j$. Since under the assignment at the beginning of the main call, E_j occurred, by lines 2 and 3 of the Resample routine, it must be the case that during some resampling of the variables in $\mathrm{sc}(E_j)$, E_j became non-occurring. The main call could not have ended after this resampling, since E_j occurs under the assignment β produced at the end of this call. Then, there exists some $r \in \Gamma_j$ such that Resample(E_r) is the subsequent Resample call. Thus $E_j \in \mathcal{X}_r$ and we obtain a contradiction as in the case where $E_k \in \mathcal{X}_j$ above. □

An immediate corollary of Lemma 2, is that the events of the root calls of Resample are pairwise distinct, therefore there can be *at most* m such root calls in any execution of M-Algorithm.

Consider now *rooted forests*, i.e. forests of trees such that each tree has a special node designated as its root, whose vertices are labeled by events E_j, $j \in \{1, \ldots, m\}$. We will use such forests to depict the executions of M-Algorithm.

Definition 3. *A labeled rooted forest $\mathcal{F}$ is called* feasible *if:*

1. *the labels of its roots are* pairwise distinct*,*
2. *the labels of any two siblings (i.e. vertices with a common parent) are distinct and*
3. *an internal vertex labeled by E_j has at most $|\Gamma_j| + 1$ children, with labels whose indices are in $\Gamma_j \cup \{j\}$.*

The number of nodes of a feasible forest $\mathcal{F}$ is denoted by $|\mathcal{F}|$.

The nodes of such a labeled forest are ordered as follows: children of the same node are ordered as their labels are; nodes in the same tree are ordered by preorder (the ordering induced by running the *depth-first search* algorithm on input such a tree), respecting the ordering between siblings; finally if the label on the root of a tree T_1 precedes the label of the root of T_2, all nodes of T_1 precede all nodes of T_2.

Given an execution of M-Algorithm that lasts for *at least* n rounds, we construct, in a unique way, a feasible forest with n nodes. First, we create one node

for each RESAMPLE call and label it with its argument. Root calls correspond to the roots of the trees and a recursive call made from line 3 or 6 of a RESAMPLE(E_j) call gives rise to a child of the corresponding node of this RESAMPLE(E_j) call. We say that a feasible forest $\mathcal{F}$ constructed this way is the *n-witness* forest of M-ALGORITHM's execution and we define $W_{\mathcal{F}}$ to be the event M-ALGORITHM executes producing $\mathcal{F}$ as an n-witness forest.

Define P_n to be the probability that M-ALGORITHM lasts for *at least* n rounds. Obviously:

$$P_n = \Pr\left[\bigcup_{\mathcal{F}:|\mathcal{F}|=n} W_{\mathcal{F}}\right] = \sum_{\mathcal{F}:|\mathcal{F}|=n} \Pr\Big[W_{\mathcal{F}}\Big], \tag{4}$$

where the last equality holds because the events $W_{\mathcal{F}}$ are disjoint.

Unfortunately, M-ALGORITHM introduces various dependencies that render the probabilistic calculations essentially impossible. For example, suppose that the i-th node of a witness forest $\mathcal{F}$ is labeled by E_j and its children have labels with indices in Γ_j. Then, under the assignment produced at the end of the i-th round of this execution, E_j does not occur.

To avoid such dependencies, we introduce a *validation algorithm*, VALALG. Interestingly, VALALG produces no progress towards locating the sought after assignment. However, as we will see, it has two useful properties: (i) From round to round, the distribution of the variables does not change, a fact that makes possible a direct probabilistic analysis and (ii) the probability that it lasts for at least n steps bounds from above the respective probability of M-ALGORITHM (see Lemma 4).

Algorithm 2 VALALG.

Input: Feasible forest $\mathcal{F}$ with labels $E_{j_1}, \ldots, E_{j_n}$.

1: Sample the variables X_i, $i = 1, ..., l$.
2: **for** s=1,...,n **do**
3: **if** E_{j_s} does not occur under the current assignment **then**
4: **return** `failure` and exit.
5: **else**
6: Resample the variables in sc(E_{j_s})
7: **end if**
8: **end for**
9: **return** `success`.

A round of VALALG is the duration of any **for** loop executed at lines 2-8. If the algorithm manages to go through its input without coming upon a non-occurring event at any given round, it returns `success`. Thus, the success of VALALG has no consequence with respect to the occurrence, at the end, of the undesirable events.

The following result concerns the distribution of the random assignments at any round of VALALG.

Lemma 3 (Randomness lemma)**.** *At the beginning of any given round of* VALALG*, the distribution of the current assignment of values to the variables X_i, $i = 1, ..., l$, given that* VALALG *has not failed, is as if all variables have been sampled anew.*

Proof. This follows from the fact that at each round, the variables for which their values have been exposed, are immediately resampled. □

Now, given a feasible forest $\mathcal{F}$ with n nodes, we say that $\mathcal{F}$ is *validated* by ValAlg if the latter returns `success` on input $\mathcal{F}$. The event of this happening is denoted by $V_{\mathcal{F}}$. We also set:

$$\hat{P}_n = \sum_{\mathcal{F}:\ |\mathcal{F}|=n} \Pr[V_{\mathcal{F}}]. \tag{5}$$

Lemma 4. *For any feasible forest $\mathcal{F}$, the event $W_{\mathcal{F}}$ implies the event $V_{\mathcal{F}}$, therefore $P_n \leq \hat{P}_n$.*

Proof. Indeed, if the random choices made by an execution of M-Algorithm that produces as witness forest $\mathcal{F}$ are made by ValAlg on input $\mathcal{F}$, then clearly ValAlg will return `success`. □

From now on, we will use the following notation: $\mathbf{n} = \{n_1, \ldots, m\}$, where $n_1, \ldots, n_m \geq 0$ are such that $\sum_{i=1}^{m} n_i = n$ and $\mathbf{n} - (1)_j := (n_1, \ldots, n_j - 1, \ldots, n_m)$. We now state and prove our first result:

Theorem 1a (Algorithmic directed LLL)**.** *Suppose that there exist $\chi_1, \chi_2, \ldots, \chi_m \in (0, 1)$, such that*

$$\Pr(E_j) \leq \chi_j \prod_{i \in \Gamma_j} (1 - \chi_i),$$

for all $j \in \{1, \ldots, m\}$, where Γ_j denotes the outwards neighborhood of E_j in the d-dependency graph. Then, the probability that M-Algorithm *executes for* at least *n rounds is* inverse exponential *in n.*

Proof. We may assume, without loss of generality, that $\Pr[E_j] < \chi_j \prod_{i \in \Gamma_j}(1 - \chi_i)$ for all $j \in \{1, \ldots, m\}$, i.e. that the hypothesis is given in terms of a strict inequality. Indeed, otherwise consider an event B, such that B and $E_1, \ldots, E_m$, are mutually independent, where $\Pr[B] = 1 - \delta$, for arbitrary small $\delta > 0$. We can now perturb the events a little, by considering e.g. $E_j \cap B$, $j = 1, \ldots, m$. As a consequence, we can also assume without loss of generality that for some other small enough $\epsilon > 0$, we have that $Pr[E_j] \leq (1 - \epsilon)\chi_j \prod_{i \in \Gamma_j}(1 - \chi_i)$.

By Lemma 4, it suffices to prove that $\hat{P}_n$ is inverse exponential in n. Specifically, we show that $\hat{P}_n \leq (1-\epsilon)^n$. Let $Q_{\mathbf{n},j}$ be the probability that ValAlg is successful when started on a *tree* whose root is labeled with E_j and has $\sum_{i=1}^{m} n_i = n$ nodes labeled with $E_1, \ldots, E_m$. Observe that to obtain a bound for $\hat{P}_n$ we need to add over all possible forests with n nodes in total. Thus, it holds that:

$$\hat{P}_n \leq \sum_{\mathbf{n}} \sum_{\mathbf{n}^1 + \ldots + \mathbf{n}^m = \mathbf{n}} \left(Q_{\mathbf{n}^1,1} \cdots Q_{\mathbf{n}^m,m} \right).$$

Our aim is to show that $Q_{\mathbf{n},j}$ is exponentially small to n, for any given sequence of $\mathbf{n}$ and any $j \in \{1, \ldots, m\}$. Thus, by ignoring polynomial in n factors, the same will hold for $\hat{P}_n$ (recall that the number of variables and the number of events are considered constants, asymptotics are in terms of the number of steps n only).

Let $\Gamma_j^+ := \Gamma_j \cup \{j\}$, and assume that, for each $j \in \{1, \ldots, m\}$, $|\Gamma_j^+| = k_j$. Observe now that $Q_{\mathbf{n},j}$ is bounded from above by a function, denoted again by

$Q_{\mathbf{n},j}$ (to avoid overloading the notation), which follows the recurrence:

$$Q_{\mathbf{n},j} = \Pr[E_j] \cdot \sum_{\mathbf{n}^1+\cdots+\mathbf{n}^{k_j}=\mathbf{n}-(1)_j} \Big(Q_{\mathbf{n}^1,j_1} + \cdots Q_{\mathbf{n}^{k_j},j_{k_j}}\Big), \tag{6}$$

with initial conditions $Q_{\mathbf{n},j} = 0$ when $n_j = 0$ and there exists an $i \neq j$ such that $n_i \geq 1$; and with $Q_{\mathbf{0},j} = 1$, where $\mathbf{0}$ is a sequence of m zeroes.

To solve the above recurrence, we introduce, for $j = 1, \ldots, m$, the *multivariate generating functions*:

$$Q_j(\mathbf{t}) = \sum_{\mathbf{n}:n_j\geq 1} Q_{\mathbf{n},j}\mathbf{t}^{\mathbf{n}}, \tag{7}$$

where $\mathbf{t} = (t_1, \ldots, t_m)$, $\mathbf{t}^{\mathbf{n}} := t_1^{n_1} \cdots t_m^{n_m}$.

By multiplying both sides of (6) by $\mathbf{t}^n$ and adding all over suitable $\mathbf{n}$, we get the system of equations $\mathbf{Q}$:

$$Q_j(\mathbf{t}) = t_j f_j(\mathbf{Q}), \tag{8}$$

where, for $\mathbf{x} = (x_1, \ldots, x_m)$ and $j = 1 \ldots, m$:

$$f_j(\mathbf{x}) = (1-\epsilon) \cdot \chi_j \cdot \left(\prod_{i\in\Gamma_j}(1-\chi_i)\right) \cdot \left(\prod_{i\in\Gamma_j^+}(x_i+1)\right). \tag{9}$$

To solve the system, we will directly use the result of Bender and Richmond in [3] (Theorem 2). Let g be any m-ary projection function on some of the m coordinates. In the sequel we take $g := pr_s^m$, the (m)-ary projection on the s-th coordinate. Let also $\mathcal{B}$ be the set of trees $B = (V(B), E(B))$ whose vertex set is $\{0, 1, \ldots, m\}$ and with edges directed towards 0. By [3], we get:

$$[\mathbf{t}^{\mathbf{n}}]g((\mathbf{Q},\mathbf{R})(\mathbf{t})) = \frac{1}{\prod_{j=1}^m n_j} \sum_{B\in\mathcal{B}} [\mathbf{x}^{\mathbf{n}-\mathbf{1}}] \frac{\partial(g, f_1^{n_1}, \ldots, f_m^{n_m})}{\partial B}, \tag{10}$$

where the term for a tree $B \in \mathcal{B}$ is defined as:

$$[\mathbf{x}^{\mathbf{n}-\mathbf{1}}] \prod_{r\in V(B)} \left\{ \left(\prod_{(i,r)\in E(B)} \frac{\partial}{\partial x_i} \right) f_r^{n_r}(\mathbf{x}) \right\}, \tag{11}$$

where $r \in \{0, \ldots, m\}$ and $f_0^{n_0} := g$.

We consider a tree $B \in \mathcal{B}$ such that (11) is not equal to 0. Thus, $(i, 0) \neq E(B)$, for all $i \neq s$. On the other hand, $(s, 0) \in E(B)$, lest vertex 0 is isolated, and each vertex has out-degree *exactly* one, lest a cycle is formed or connectivity is broken. From vertex 0, we get $\frac{\partial pr_s^m(\mathbf{x})}{\partial x_s} = 1$. Since our aim is to prove that $\hat{P}_n$ is exponentially small in n, we are are interested only in factors of (11) that are exponential in n, and we can thus ignore the derivatives (except the one for vertex 0), as they introduce only polynomial (in n) factors to the product. Thus, we have that (11) is equal to the coefficient of $\mathbf{x}^{\mathbf{n}-\mathbf{1}}$ in:

$$\prod_{j=1}^m \left\{ (1-\epsilon)^{n_j} \cdot \chi_j^{n_j} \cdot \left(\prod_{i\in\Gamma_j}(1-\chi_i)^{n_j}\right) \cdot \left(\prod_{i\in\Gamma_j^+}(x_i+1)^{n_j}\right)\right\}. \tag{12}$$

We now group the factors of (12) according to the i's. We have already argued each vertex i has out-degree 1. Thus, the exponent of the term $x_i + 1$ is $n_i + \sum_{j:i\in\Gamma_j} n_j$

and the product of (12) is equal to:

$$\prod_{i=1}^{m}\left\{(1-\epsilon)^{n_i}\cdot\chi_i^{n_i}\cdot(1-\chi_i)^{\sum_{j:i\in\Gamma_j}n_j}\cdot(x_i+1)^{n_i+\sum_{j:i\in\Gamma_j}n_j}\right\}. \tag{13}$$

Using the binomial theorem and by ignoring polynomial factors, we get that the coefficient of $\mathbf{x^{n-1}}$ in (13) is:

$$\prod_{i=1}^{m}\left\{(1-\epsilon)^{n_i}\cdot\chi_i^{n_i}\cdot(1-\chi_i)^{\sum_{j:i\in\Gamma_j}n_j}\cdot\binom{n_i+\sum_{j:i\in\Gamma_j}n_j}{n_i}\right\}. \tag{14}$$

By expanding $(\chi_i+1-\chi_i)^{n_i+\sum_{j:i\in\Gamma_j}n_j}$, we get that (14) is at most:

$$\prod_{i=1}^{m}(1-\epsilon)^{n_i}=(1-\epsilon)^{\sum_{i=1}^{n}n_i}=(1-\epsilon)^n. \tag{15}$$

Thus, $\hat{P}_n$ is inverse exponential in n. □

From Theorem 1a, the existential Theorem 1 immediately follows.

4. Shearer's lemma

We now turn our attention to Shearer's lemma. The first algorithmic proof for general probability spaces was given by Kolipaka and Szegedy [15]. Harvey and Vondrák [11] proved a version of the lemma for their lopsided association graphs (again in the generalized framework).

Here, we apply it to the underlying *undirected* graph of the d-dependency graph we introduced in Section 2. Our work is situated in the variable framework and we give a forward argument that directly leads to an exponentially small bound of the probability of the algorithm lasting for at least n steps.

Let $E_1,\ldots,E_m$ be events, whose vector of probabilities is $\bar{p}=(p_1,\ldots,p_m)$, that is $Pr[E_j]=p_j\in(0,1),\ j=1,\ldots,m$. Let also $G=\langle\{1,\ldots,m\},E\rangle$ be a graph on m vertices, where we associate each event E_j with vertex j, $j=1,\ldots,m$ and where

$$E=\{\{i,j\}\mid\text{either } E_j \text{ is } d\text{-dependent on } E_i \text{ or } E_i \text{ is } d\text{-dependent on } E_j\}.$$

For each vertex j, we denote its *neighborhood* in G by Γ_j, $j=1,\ldots,m$.

A subset $I\subseteq\{1,\ldots,m\}$ of the graph's vertices is an *independent set* if there are no edges between its vertices. Abusing the notation, we will sometimes say that an independent set I contains events (instead of indices of events). Let $I(G)$ denote the set of independent sets of G. For any $I\in I(G)$, let $\Gamma(I):=\bigcup_{j\in I}\Gamma_j$ be the set of neighbors of the vertices of I. Following [15], we say that I *covers* J if $J\subseteq I\cup\Gamma(I)$.

A *multiset* is usually represented as a couple (A,f), where A is a set, called the *underlying set*, and $f:A\mapsto\mathbb{N}_{\geq 1}$ is a function, with $f(x)$ denoting the multiplicity of x, for all $x\in A$. In our case, the underlying sets of multisets are always subsets of $\{1,\ldots,m\}$. Thus, to make notation easier to follow, we use couples $(I,\bar{z})$, where $I\subseteq\{1,\ldots,m\}$ and $\bar{z}=(z_1,\ldots,z_m)$ is an m-ary vector, with $z_j\in\mathbb{N}$ denoting the multiplicity of E_j in I. Note that $z_j=0$ if and only if $j\notin I$, $j=1,\ldots,m$.

Consider our second main theorem below, which is a variation of Shearer's Lemma for d-dependency graphs.

Theorem 2 (Shearer's lemma for d-dependency graphs)**.** *If for all $I \in I(G)$:*

$$\text{(Shear)} \qquad q_I(G,\bar{p}) = \sum_{J\in I(G):\ I\subseteq J} (-1)^{|J\setminus I|} \prod_{j\in J} p_j > 0,$$

then

$$Pr\left[\bigwedge_{j=1}^{m} \bar{E}_j\right] > 0.$$

Actually, we prove below an algorithmic version (Theorem 2a) of the existential Theorem 2, where we give exponentially small estimates of the probability of the algorithm not producing the desired results.

The algorithm we use is a variation of the Maximal Set Resample algorithm, designed by Harvey and Vondrák in [11], which is a slowed down version of the algorithm in [15]. The algorithm constructs multisets whose underlying sets are independent sets of G, by selecting occurring events that it resamples until they do not occur anymore.

Algorithm 3 MaxSetRes.

1: Sample the variables X_i, $i = 1, ..., l$ and let α be the resulting assignment.
2: $t := 1$, $I_t := \emptyset$, $\bar{z}^t := (0, \ldots, 0)$.
3: **repeat**
4: **while** there exists an event $E_j \notin I_t \cup \Gamma(I_t)$ that occurs under the current assignment,
let E_j be the least indexed such event and **do**
5: $I_t := I_t \cup \{j\}$, $c := 0$, $z_j^t := 1$.
6: **while** E_j occurs **do**
7: Resample the variables in $\mathrm{sc}(E_j)$.
8: $c := c + 1$.
9: **end while**.
10: **if** there exists an occurring event that is not in $I_t \cup \Gamma(I_t)$ **then**
11: $z_j^t := c + 1$.
12: **else if** there exists an occurring event not in Γ_j **then**
13: $t := t + 1$, $I_t := \emptyset$.
14: $z_j^t := c$.
15: **else**
16: **for** $s = 1, \ldots, c$ **do**
17: $I_{t+s} := \{E_j\}$.
18: **end for**
19: $t := t + c + 1$, $I_t := \emptyset$.
20: **end if**
21: **end while**
22: **until** $I_t = \emptyset$.
23: Output current assignment α.

A *step* of MaxSetRes is a single resampling of the variables of an event in line 7, whereas a *phase* is an iteration of **repeat** at lines 3–22, except from the last iteration that starts and ends with $I_t = \emptyset$. Phases are not nested. During each phase, there are at most m repetitions of the **while**–loop of lines 6–9, where m is the

number of events (recall that the number of variables l, and the number of events m are considered to be constants). During each phase, a multiset (I_t, z^t) is created, where the underlying set I_t is an independent set of G. There are two scenarios that can happen at the end of a phase. The first is by line 13, where MaxSetRes creates a new independent set, containing c copies of the last event it resampled at lines 6–9. The second is by lines 16–19, where MaxSetRes proceeds c phases at once, creating c singleton sets, containing only the event it lastly resampled at lines 6–9. Lines 10–20 exist for technical reasons that will become apparent later.

Note that, by lines 4 and 22, if and when MaxSetRes terminates, it produces an assignment of values under which none of the events occurs.

We now proceed with the first lemma concerning the execution of MaxSetRes. Note that it refers to the underlying independent sets of the multisets created at each phase.

Lemma 5. *I_t covers I_{t+1}, for all $t \in \{1, \ldots, n-1\}$.*

Proof. Let E_j be an event in I_{t+1}. Then, at some point during phase $t+1$, E_j was occurring. We will prove below that E_j occurs also at the beginning of phase $t+1$. This will conclude the proof, since if $E_j \notin \Gamma(I_t) \cup I_t$, then at the moment when phase I_{t+1} was to start, the algorithm instead of starting I_{t+1} would opt to add E_j to I_t, a contradiction.

Assume that $I_{t+1} \neq \{E_j\}$, lest we have nothing to prove. To prove that E_j occurs at the beginning of phase $t+1$, assume towards a contradiction that it does not. Then it must have become occurring during the repeated resamplings of an event E_r introduced into I_{t+1}.

Therefore, under the assignment when E_r was selected, E_r occurred and E_j did not. Furthermore, during the repeated resamplings of E_r, only variables in $\mathrm{sc}(E_r)$ had their values changed, and under the assignment at the end of these resamplings, E_r ceases occurring and E_j occurs. By Definition 1, E_j is d-dependent on E_r and thus $E_j \in \Gamma(I_{t+1})$. By lines 4, 10 and 12, E_j could not have been selected at any point during round $t+1$. This concludes the proof. □

Consider the following definition:

Definition 4 (Kolipaka and Szegedy [15])**.** *A* stable *sequence of events is a sequence of non-empty independent sets $\mathcal{I} = I_1, \ldots, I_n$ such that I_t covers I_{t+1}, for all $t \in \{1, \ldots, n-1\}$.*

Stable sequences play the role of witness structures in the present framework. By Lemma 5, the underlying sets of the sequence of multisets MaxSetRes produces in an execution that lasts for at least n phases, is a stable sequence of length n.

We now prove:

Theorem 2a (Algorithmic Shearer's lemma for d-dependency graph)**.** *If for all $I \in I(G)$:*

$$q_I(G, \bar{p}) = \sum_{J \in I(G):\ I \subseteq J} (-1)^{|J \setminus I|} \prod_{j \in J} p_j > 0,$$

then the probability $\mathrm{P_n}$ that MaxSetRes *lasts for at least n phases is exponentially small, i.e. for some constant $c < 1$, $\mathrm{P_n}$ is at most c^n, ignoring polynomial factors.*

Again, Theorem 2 follows immediately from Theorem 2a.

Proof. Let $\mathbf{z} = (\bar{z}^1, \ldots, \bar{z}^n)$ be an n-ary vector, whose elements $\bar{z}^t = (z_1^t, \ldots, z_m^t)$ are m-ary vectors of non-negative integers, $t = 1, \ldots, n$. Let also

$$(\mathcal{I}, \mathbf{z}) = (I_1, \bar{z}^1), \ldots, (I_n, \bar{z}^n)$$

be a sequence of multisets, whose underlying sequence $\mathcal{I}$ is a stable sequence. We denote by $|(\mathcal{I}, \mathbf{z})|$ its length, i.e. the number of pairs $(\mathcal{I}_t, \bar{z}^t)$ it contains. If $\mathrm{P}(\mathcal{I}, \mathbf{z})$ is the probability that an execution of MaxSetRes produced $(\mathcal{I}, \mathbf{z})$ (which can be zero), it is easy to see that:

$$\mathrm{P_n} = \sum_{(\mathcal{I},\mathbf{z}):|(\mathcal{I},\mathbf{z})|=\mathrm{n}} \mathrm{P}(\mathcal{I}, \mathbf{z}), \tag{16}$$

where the sum is over all possible pairs of stable sequences $\mathcal{I}$ of length n and vectors $\mathbf{z}$.

To bound the rhs of (16), consider the validation algorithm MaxSetVal below. MaxSetVal, on input a stable sequence $\mathcal{I} = I_1, \ldots, I_n$, proceeds to check each event contained in each independent set. If this event does not occur, it fails; else it resamples the variables in its scope. Note that the success or failure of this algorithm has nothing to do with finding an assignment such that none of the events occur.

Algorithm 4 MaxSetVal.

Input: Stable sequence $\mathcal{I} = I_1, \ldots, I_n$.

```
1: Sample the variables X_i, i = 1, ..., l.
2: for t=1,...,n do
3:     for each event E_j of I_t do
4:         if E_j does not occur under the current assignment then
5:             return failure and exit.
6:         else
7:             Resample the variables in sc(E_j)
8:         end if
9:     end for
10: end for
11: return success.
```

A phase of MaxSetVal is any repetition of lines 2–10. Let $\hat{\mathrm{P}}(\mathcal{I})$ be the probability that MaxSetVal is successful on input $\mathcal{I}$ and:

$$\hat{\mathrm{P}}_n := \sum_{\mathcal{I}:|\mathcal{I}|=n} \hat{\mathrm{P}}(\mathcal{I}). \tag{17}$$

To obtain our result, we now proceed to show: (i) that $\mathrm{P_n} \leq \hat{\mathrm{P}}_\mathrm{n}$ and (ii) that $\hat{\mathrm{P}}_n$ is inverse exponential to n. For the former, consider the validation algorithm MultiSetVal, algorithm 5, below.

MultiSetVal takes as input a sequence $(\mathcal{I}, \mathbf{z})$ of multisets whose underlying sequence is stable. It then proceeds, for each multiset, to check if its events occur under the current assignment it produces. If not it fails, else it proceeds. When the last copy of an event inside a multiset, apart from the last event, is resampled, it checks if that event still occurs (line 11). If it does, the algorithm fails. If it manages to go through the whole sequence without failing, it succeeds. Note again

Algorithm 5 MULTISETVAL.

Input: $(\mathcal{I}, \mathbf{z}) = (I_1, z^1), \ldots, (I_n, z^n)$, $I_t = \{E_{t_1}, \ldots, E_{t_{k_t}}\}$, $t = 1, \ldots, n$.

1: Sample the variables X_i, $i = 1, ..., l$.
2: **for** $t = 1, \ldots, n$ **do**
3: **for** $s = 1, \ldots, k_t - 1$ **do**
4: **for** $r = 1, \ldots, z^t_{t_s}$ **do**
5: **if** E_{t_s} does not occur under the current assignment **then**
6: **return** `failure` and exit.
7: **else**
8: Resample the variables in $\mathrm{sc}(E_{t_s})$
9: **end if**
10: **end for**
11: **if** E_{t_s} occurs under the current assignment **then**
12: **return** `failure` and exit.
13: **end if**
14: **end for**
15: **if** $E_{t_{k_t}}$ does not occur under the current assignment **then**
16: **return** `failure` and exit.
17: **else**
18: Resample the variables in $\mathrm{sc}(E_{t_{k_t}})$
19: **end if**
20: **end for**
21: **return** `success`.

that the success or failure of MULTISETVAL has nothing to do with obtaining an assignment such that none of the events holds.

We call a *phase* of MULTISETVAL each repetition of lines 2–20. Let also $\tilde{\mathrm{P}}(\mathcal{I}, \mathbf{z})$ be the probability that MULTISETVAL succeeds on input $(\mathcal{I}, \mathbf{z})$. We prove two lemmas concerning MULTISETVAL.

Lemma 6. *For each sequence* $(\mathcal{I}, \mathbf{z})$, $\mathrm{P}(\mathcal{I}, \mathbf{z}) \leq \tilde{\mathrm{P}}(\mathcal{I}, \mathbf{z})$. *Thus:*

$$\mathrm{P_n} \leq \sum_{(\mathcal{I},\mathbf{z}):|(\mathcal{I},\mathbf{z})|=\mathrm{n}} \tilde{\mathrm{P}}(\mathcal{I}, \mathbf{z}) \tag{18}$$

Proof. It suffices to prove the first inequality, as the result is then derived by (16). Note that the last event in every multiset that MAXSETRES produces always has multiplicity 1 and that, furthermore, it is not required to be non-occurring after resamping it, in contrast with all the other events in the multiset. It is now straightforward to notice that if MULTISETVAL makes the same random choices as MAXSETRES did when it created any sequence $(\mathcal{I}, \mathbf{z})$, MULTISETVAL will succeed on input $(\mathcal{I}, \mathbf{z})$. □

Lemma 7. *For any* $(\mathcal{I}, \mathbf{z})$, *it holds that:*

$$\sum_{(\mathcal{I},\mathbf{z}):|(\mathcal{I},\mathbf{z})|=n} \tilde{\mathrm{P}}(\mathcal{I}, \mathbf{z}) = \sum_{\mathcal{I}:|\mathcal{I}|=n} \hat{\mathrm{P}}(\mathcal{I}). \tag{19}$$

Proof. We will rearrange the sum in the lhs of (19). Assume that the stable sequences of length n in G are arbitrarily ordered as $\mathcal{I}_1, \ldots, \mathcal{I}_s$. Then, it holds that:

$$\sum_{(\mathcal{I},\mathbf{z}):|(\mathcal{I},\mathbf{z})|=n} \tilde{\mathrm{P}}(\mathcal{I},\mathbf{z}) = \sum_{\mathbf{z}=(\bar{z}^1,\ldots,\bar{z}^n)} \tilde{\mathrm{P}}(\mathcal{I}_1,\mathbf{z}) + \cdots + \sum_{\mathbf{z}=(\bar{z}^1,\ldots,\bar{z}^n)} \tilde{\mathrm{P}}(\mathcal{I}_s,\mathbf{z}). \tag{20}$$

Let $\mathcal{I} = (I_1, \ldots, I_n)$ be a stable sequence and consider the term

$$\sum_{\mathbf{z}=(\bar{z}^1,\ldots,\bar{z}^n)} \tilde{\mathrm{P}}(\mathcal{I},\mathbf{z})$$

of (20) corresponding to $\mathcal{I}$. It suffices to show that is is equal to $\hat{\mathrm{P}}(\mathcal{I})$.

Assume again that $I_t = \{E_{t_1}, \ldots, E_{t_{k_t}}\}$ and that $\bar{z}^t = (z_1^t, \ldots, z_m^t)$, where $z_j^t \geq 0$, $j = 1, \ldots, m$, $t = 1, \ldots, n$. Finally, set:

$$\tilde{\mathrm{P}}(I_t, \bar{z}^t) := \Pr[E_{t_1}]^{z_1^t} \Pr[\bar{E}_{t_1} \cap E_{t_2}] \Pr[E_2]^{z_2^t - 1} \cdots \Pr[E_{t_{k_t}-1}]^{z_{k_t}^t - 1} \Pr[\bar{E}_{t_{k_t}-1} \cap E_{t_{k_t}}].$$

Then, it holds that:

$$\sum_{\mathbf{z}=(\bar{z}^1,\ldots,\bar{z}^n)} \tilde{\mathrm{P}}(\mathcal{I},\mathbf{z}) = \sum_{\mathbf{z}=(z^1,\ldots,z^n)} \prod_{t=1}^{n} \tilde{\mathrm{P}}(I_t, z^t). \tag{21}$$

By Lemma 1, it holds that all the factors $\Pr[\bar{E} \cap E']$ that appear in (21) are less or equal than $\Pr[\bar{E}] \cdot \Pr[E']$. Now, by factoring out:

$$\hat{\mathrm{P}}(\mathcal{I}) = \prod_{t=1}^{n} \left(\Pr[E_{t_1}] \cdots \Pr[E_{t_{k_t}}] \right)$$

from the rhs of (21) and by rearranging the terms according to the sets $\mathcal{I}_t$, we get:

$$\sum_{\mathbf{z}=(\bar{z}^1,\ldots,\bar{z}^n)} \tilde{\mathrm{P}}(\mathcal{I},\mathbf{z}) = \hat{\mathrm{P}}(\mathcal{I}) \cdot \prod_{t=1}^{n} \left(\sum_{z^t=(z_1^t,\ldots,z_{k_t}^t)} \Pr[E_{t_1}]^{z_1^t - 1}(1 - \Pr[E_{t_1}]) \cdots \right. \\ \left. \Pr[E_{t_{k_t}-1}]^{z_{k_t-1}^t - 1}(1 - \Pr[E_{t_{k_t-1}}]) \right). \tag{22}$$

The proof is now complete, by noticing that all the factors, except from $\hat{\mathrm{P}}(\mathcal{I})$ in the rhs of (22) are equal to 1. □

Thus, by (18), (19) and (17), we get:

$$\mathrm{P_n} \leq \hat{\mathrm{P}}_\mathrm{n}. \tag{23}$$

Thus, what remains is to show that $\hat{\mathrm{P}}_n$ is inverse exponential to n. Towards this, for $n \geq 1$ let

$$\hat{\mathrm{P}}_{n,I} = \sum_{\substack{\mathcal{I}:|\mathcal{I}|=n \\ \mathcal{I}_1 = I}} \hat{\mathrm{P}}(\mathcal{I}), \tag{24}$$

where $\mathcal{I}_1$ is its first term of $\mathcal{I}$.

Observe now that, for any independent set I, $\hat{\mathrm{P}}_{1,I} = \prod_{j \in I} p_j$. Thus we obtain the following recursion:

$$\hat{\mathrm{P}}_{n+1,I} = \begin{cases} \prod_{j \in I} p_j \left(\sum_{J:I \text{ covers } J} \hat{\mathrm{P}}_{n,J} \right) & \text{if } n \geq 1, \\ \prod_{j \in I} p_j & \text{if } n = 0. \end{cases} \tag{25}$$

If the class of all non-empty independent sets is $\{I_1, \ldots, I_s\}$, following again the terminology of [15], we define the *stable set matrix* M, as an $s \times s$ matrix, whose element in the i-th row and j-th column is $\prod_{j \in I} p_j$ if I covers J and 0 otherwise. Furthermore, let $q_n = (\hat{\mathrm{P}}_{n,I_1}, \ldots, \hat{\mathrm{P}}_{n,I_s})$. Easily, (25) is equivalent to:

$$q_n = M q_{n-1},$$

thus

$$q_n = M^{n-1} q_1. \tag{26}$$

Let $\| \cdot \|_1$ be the 1-*norm* defined on $\mathbb{R}^s$. It is known that any vector norm, and thus 1-norm too, yields a norm for square matrices called the *induced norm* [13] as follows:

$$\|M\|_1 := \sup_{x \neq 0} \frac{\|Mx\|_1}{\|x\|_1} \geq \frac{\|Mq_1\|_1}{\|q_1\|_1}. \tag{27}$$

By (26) and (27), we have that:

$$\|q_n\|_1 = \|M^{n-1} q_1\|_1 \leq \|M^{n-1}\|_1 \cdot \|q_1\|_1. \tag{28}$$

Note now that:

$$\hat{\mathrm{P}}_n \leq \sum_{i=1}^{s} \hat{\mathrm{P}}_{n,I_i} = \|q_n\|_1 = \|M^{n-1}\|_1 \|q_1\|_1. \tag{29}$$

Since $\|q_1\|_1$ is a constant, it suffices to show that $\|M^{n-1}\|_1$ is exponentially small in n. Let $\rho(M)$ be the *spectral radius* of M [13], that is:

$$\rho(A) := \max\{|\lambda| \mid \lambda \text{ is an eigenvalue of } A\}.$$

By Gelfand's formula (see again [13]) used for the induced matrix norm $\| \cdot \|_1$, we have that:

$$\rho(M) = \lim_{n \to \infty} \|M^n\|_1^{1/n}. \tag{30}$$

Furthermore, in [15] (Theorem 14), it is proved that the following are *equivalent*:

(1) For all $I \in I(G)$: $q_I(G, \bar{p}) > 0$.
(2) $\rho(M) < 1$.

Using $(1 \Rightarrow 2)$ we can select an $\epsilon > 0$ such that $\rho(M) + \epsilon < 1$. Then, by (30), we have that there exists a n_0 (depending only on ϵ, M) such that, for $n \geq n_0$: $\|M^{n-1}\|_1 \leq (\rho(M) + \epsilon)^{n-1}$, which, together with (29), gives us that $\hat{\mathrm{P}}_n$ is exponentially small in n.

Thus, by the analysis above, we get that there is a constant $c < 1$ (depending on $\|q_1\|, p$ and $\rho(M) + \epsilon$) such that $\mathrm{P_n} \leq \mathrm{c^n}$, for $n \geq n_0$ and by ignoring polynomial factors. This concludes the proof.

□

Acknowledgment

We are truly grateful to Ioannis Giotis and Dimitrios Thilikos for their substantial contribution to earlier versions of this work (see [7] and [8]).

References

[1] Dimitris Achlioptas and Fotis Iliopoulos. Random walks that find perfect objects and the Lovász local lemma. In *Proceedings 55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 494–503. IEEE, 2014.

[2] Dimitris Achlioptas and Fotis Iliopoulos. Random walks that find perfect objects and the lovász local lemma. *Journal of the ACM (JACM)*, 63(3):22, 2016.

[3] Edward A Bender and L Bruce Richmond. A multivariate Lagrange inversion formula for asymptotic calculations. *The Electronic Journal of Combinatorics*, 5(1):33, 1998.

[4] Paul Erdős and László Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. *Infinite and finite sets*, 10:609–627, 1975.

[5] Paul Erdős and Joel Spencer. Lopsided Lovász local lemma and Latin transversals. *Discrete Applied Mathematics*, 30(2-3):151–154, 1991.

[6] Ioannis Giotis, Lefteris Kirousis, Kostas I. Psaromiligkos, and Dimitrios M. Thilikos. On the algorithmic Lovász local lemma and acyclic edge coloring. In *Proceedings of the twelfth workshop on analytic algorithmics and combinatorics*. Society for Industrial and Applied Mathematics, 2015. Available: `http://epubs.siam.org/doi/pdf/10.1137/1.9781611973761.2`.

[7] Ioannis Giotis, Lefteris M. Kirousis, John Livieratos, Kostas I. Psaromiligkos, and Dimitrios M. Thilikos. Alternative proofs of the asymmetric Lovász local lemma and Shearer's lemma. In *Proceedings of the 11th International Conference on Random and Exhaustive Generation of Combinatorial Structures, GASCom*, 2018. Available: `http://ceur-ws.org/Vol-2113/paper15.pdf`.

[8] Ioannis Giotis, Lefteris M. Kirousis, Kostas I. Psaromiligkos, and Dimitrios M. Thilikos. An alternative proof for the constructive asymmetric Lovász local lemma. In *13th Cologne Twente Workshop on Graphs and Combinatorial Optimization*, 2015.

[9] David G Harris. Lopsidependency in the Moser-Tardos framework: beyond the lopsided Lovász local lemma. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1792–1808. SIAM, 2015.

[10] David G Harris and Aravind Srinivasan. A constructive algorithm for the Lovász local lemma on permutations. In *Proceedings 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 907–925. SIAM, 2014.

[11] Nicholas JA Harvey and Jan Vondrák. An algorithmic proof of the Lovász local lemma via resampling oracles. In *Proceedings 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1327–1346. IEEE, 2015.

[12] Kun He, Liang Li, Xingwu Liu, Yuyi Wang, and Mingji Xia. Variable-version Lovász local lemma: Beyond shearer's bound. In *58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 451–462. IEEE, 2017.

[13] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 1990.

[14] Lefteris Kirousis, John Livieratos, and Kostas I Psaromiligkos. Directed Lovász local lemma and shearer's lemma. *Annals of Mathematics and Artificial Intelligence*, 88(1-3):133–155, 2020.

[15] Kashyap Kolipaka, Babu Rao, and Mario Szegedy. Moser and Tardos meet Lovász. In *Proceedings 43rd Annual ACM symposium on Theory of Computing (STOC)*, pages 235–244. ACM, 2011.

[16] Robin A. Moser. A constructive proof of the Lovász local lemma. In *Proceedings 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 343–350. ACM, 2009.

[17] Robin A. Moser and Gábor Tardos. A constructive proof of the general Lovász local lemma. *Journal of the ACM (JACM)*, 57(2):11, 2010.

[18] Kaushik Sarkar and Charles J Colbourn. Upper bounds on the size of covering arrays. *SIAM Journal on Discrete Mathematics*, 31(2):1277–1293, 2017.

[19] James B. Shearer. On a problem of Spencer. *Combinatorica*, 5(3):241–245, 1985.

[20] Mario Szegedy. The Lovász local lemma–a survey. In *International Computer Science Symposium in Russia*, pages 1–11. Springer, 2013.

[21] Terence Tao. Moser's entropy compression argument. 2009. Available: `https://terrytao.wordpress.com/2009/08/05/mosers-entropy-compression-argument/`.