ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ
ΥΠΟΛΟΓΙΣΤΩΝ

Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Λογικής και Επιστήμης Υπολογιστών CoReLab

Algorithmic aspects of the Lovasz Local Lemma

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Θεμιστοκλή Γουλεάκη

Επιβλέπων: Ευστάθιος Ζάχος
Καθηγητής Ε.Μ.Π.

Συνεπιβλέπων: Δημήτρης Αχλιόπτας
Καθηγητής Ε.Κ.Π.Α.

Αθήνα, Νοέμβριος 2011

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Algorithmic aspects of the Lovasz Local Lemma

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Θεμιστοκλή Γουλεάκη

Επιβλέπων: Ευστάθιος Ζάχος
Καθηγητής Ε.Μ.Π.

Συνεπιβλέπων: Δημήτρης Αχλιόπτας
Καθηγητής Ε.Κ.Π.Α.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 14η Νοεμβρίου 2011.

....	....	....
Ευστάθιος Ζάχος	Δημήτρης Αχλιόπτας	Δημήτρης Φωτάκης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Κ.Π.Α	Λέκτορας Ε.Μ.Π.

....

Θεμιστοκλής Γ. Γουλεάκης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Ο σκοπός της διπλωματικής αυτής εργασίας είναι η μελέτη και ανάλυση πιθανοτικών αλγορίθμων που οδηγούν σε κατασκευαστική απόδειξη του "Lovasz Local Lemma". Το αποτέλεσμα αυτό είναι γνωστό στη θεωρία πιθανοτήτων από το 1975 και έχει χρησιμοποιηθεί σε πολλές εφαρμογές σε συνδυασμό με την πιθανοτική μέθοδο (probabilistic method). Η πιθανοτική μέθοδος είναι ένας τρόπος να αποδυκνύει κανείς την ύπαρξη ενός συνδυαστικού αντικειμένου με χρήση πιθανοτήτων. Συγκεκριμμένα, αν θεωρήσουμε το ενδεχόμενο ένα τυχαίο αντικείμενο (σε κάποιο δειγματικό χώρο) να έχει κάποια επιθημητή ιδιότητα και αποδείξουμε ότι η πιθανότητα του ενδεχομένου αυτού είναι μη μηδενική, τότε η ύπαρξή του αντικειμένου είναι εξασφαλισμένη. Το "Lovasz Local Lemma" αντιμετωπίζει προβλήματα στα οποία υπάρχει ένα σύνολο μη επιθυμητών ιδιοτήτων. Σαν παράδειγμα μπορούμε να αναφέρουμε το πρόβλημα της ικανοποιησιμότητας λογικών προτάσεων (SAT) σε κανονική συζευκτική μορφή (CNF form). Στο πρόβλημα αυτό, το τυχαίο πείραμα είναι η απονομή τιμών αληθείας στις λογικές μεταβλητές και το ζητούμενο είναι να βρεθεί αν υπάρχει κάποια απονομή αλήθειας έτσι ώστε όλες οι "clauses" να είναι αληθείς. Μη επιθυμητή ιδιότητα είναι ότι κάποια "clause" είναι ψευδής. Το "Lovasz Local Lemma" αντιμετωπίζει περιπτώσεις όπου δεν έχουμε πλήρη αλλά μερική ανεξαρτησία αυτών των ενδεχομένων και η απόδειξη με διαφορετικό τρόπο θα ήταν δύσκολη. Η αρχική απόδειξη του "Lovasz Local Lemma" ήταν υπαρξιακή. Δηλαδή απλώς εξασφάλιζε την ύπαρξη του αντικειμένου χωρίς να δείχνει κάποιο τρόπο για να κατασκευαστεί. Όμως σε πολλές εφαρμογές, συμπεριλαμβανομένης της ικανοποιησιμότητας (SAT), είναι επιθυμητό να κατασκευάσουμε - υπολογίσουμε το αντικείμενο. Στη διπλωματική, θα μελετήσουμε την πρόσφατη (2010) απόδειξη των R.Moser , G.Tardos ότι κάτι τέτοιο είναι εφικτό, που έγινε με χρήση πιθανοτικού αλγορίθμου. Επίσης, θα δούμε μια βελτίωση - γενίκευση του "Lovasz Local Lemma" που αποδείχτηκε αρχικά υπαρξιακά με μεθόδους της Στατιστικής Φυσικής, και θα δούμε στη συνέχεια και μια κατασκευαστική απόδειξη με τη χρήση του αλγορίθμου των R.Moser , G. Tardos. Να σημειώσουμε εδώ ότι μια από τις αποδείξεις που θα παρουσιαστούν και οφείλεται κυρίως στον R.Moser περιέχει μια ενδιαφέρουσα καινούργια γενική μέθοδο απόδειξης τερματισμού για πιθανοτικούς αλγορίθμους, η οποία μας δίνει δυναμική για περεταίρω έρευνα.

Abstract

The purpose of this thesis is the study and analysis of randomized algorithms which lead to a constructive proof of the ”Lovasz Local Lemma”. This is a well-known result in probability theory since 1975, which is also quite useful in applications of the probabilistic method. The probabilistic method is a way to prove the existence of a combinatorial object using probability. Specifically, consider the event that a random object (in some sample space) has a desired property. If we prove that the probability of this event to occur is non-zero, then it’s existence is guaranteed. The Lovasz Local Lemma deals with problems in which there is a set of undesired properties. We can give as an example, the boolean CNF-SAT problem. In this setting, the random experiment is to assign random values to the boolean variables. The problem is to determine whether or not, there is a truth assignment that makes all clauses TRUE. So, for each clause, we define the possibility for it to be FALSE as the undesirable property. The Lovasz Local Lemma deals with instaces where there the events are not mutually independent, so an alternative proof would possibly be more complicated and harder. The initial proof of the Lovasz Local Lemma was existential. That is, we could prove the existence of the object but we had no clue how to construct it. However, in many applications, including SAT, it is desirable to construct or compute the combinatorial object. In this thesis, we will study the recent (2010) proof of R.Moser and G.Tardos who proved that this is actually possible using a randomized algorithm. Moreover, we will see an improvement-generalization of the Lovasz Local Lemma initially proved existentially using methods of statistical physics, and it’s constructive counterpart using the algorithm of R.Moser , G. Tardos. We should note here that one of the proof presented (attributed to R.Moser) contains an interesting new general method to prove the termination of a randomized algorithm under certain conditions. This gives us potential for further research.

Keywords

probabilistic method , Lovasz Local Lemma, constructive proof, randomized algorithms, mutual independence, entropy, sourse coding, compression

Ευχαριστίες

Με την ολοκλήρωση της διπλωματικής αυτής εργασίας και των προπτυχιακών μου σπουδών στο Εθνικό Μετσόβιο Πολυτεχνείο, θα ήθελα να ευχαριστήσω όλους τους καθηγητές μου και τους ανθρώπους του ιδρύματος που με βοήθησαν όλα αυτά τα χρόνια. Ιδιαίτερα θα ήθελα να ευχαριστήσω τους δύο επιβλέποντες καθηγητές της διπλωματικής μου. Τον καθηγητή μου κ. Στάθη Ζάχο, που με ενέπνευσε από το πρώτο εξάμηνο των σπουδών μου και τον κ. Αχλιόπτα για την πολύτιμη βοήθεια και το χρόνο που μου αφιέρωσε. Επίσης, θα ήθελα να ευχαριστήσω θερμά τα μέλη του Εργαστηρίου Λογικής και Επιστήμης Υπολογισμών (Corelab) και ιδιαίτερα τους καθηγητές κ. Φωτάκη και κ. Παγουρτζή, για το πολύ καλό κλίμα συνεργασίας που έχουμε.

Ευχαριστώ επίσης, τους Μανώλη Παπαδάκη, Χάρη Αγγελιδάκη και Ελένη Μπακάλη για την πολύτιμη βοήθειά τους με το LaTeX.

Πάνω απ΄ όλα θα ήθελα να ευχαριστήσω την οικογένεια μου. Τους γονείς μου και τον αδερφό μου, για την αμέριστη συμπαράστασή τους όλα αυτά τα χρόνια.

Θέμης Γουλεάκης

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation and statement of the lemma

The Lovász local lemma is a very powerful tool that we can use in several interesting applications of the probabilistic method. In particular, we often encounter the following setting: There is a set of "bad" events $\{A_1, A_2, ..., A_n\}$ in a probability space Ω which correspond to specific properties of some combinatorial object. We would like to prove the existence of such an object which does not have any of the aforementioned "bad" properties. Using the probabilistic method, it is sufficient to prove that $Pr[\wedge_{i=1}^{n} \neg A_i] > 0$. Clearly, our ability to prove that will depend on their probabilities as well as the dependencies between the events $\{A_i\}$. Of course, the only interesting case is when $\forall i \quad Pr[A_i] < 1$ (otherwise that probability trivially equals 0).

Now, consider the following two special cases:

1. The events $\{A_1, A_2, ..., A_n\}$ are mutually independent.

   OR

2. $$Pr[A_1] + Pr[A_2] + ... + Pr[A_n] < 1 \tag{1.1}$$

It is easy to see that in both the above cases we get a positive answer, and the proof is as follows:

1. $$Pr[\wedge_{i=1}^{n} \neg A_i] = \prod_{i=1}^{n} (1 - Pr[A_i]) > 0 \tag{1.2}$$

2. By a union bound we get:

   $$Pr[\vee_{i=1}^{n} A_i] \leq \sum_{i=1}^{n} Pr[A_i] < 1 \Rightarrow Pr[\wedge_{i=1}^{n} \neg A_i] = 1 - Pr[\vee_{i=1}^{n} A_i] > 0 \tag{1.3}$$

The Lovász local lemma allows us to go even further and make the proof even under less restrictive conditions. The general form of the LLL is most commonly stated as follows:

Lemma 1.1.1 *The Local Lemma (General case) [7]*

Let $\{A_1, A_2, ..., A_n\}$ be events in an arbitrary probability space. A directed graph D=(V,E) on the set of vertices $V = \{1, 2, ..., n\}$ is called a dependency digraph for the events $\{A_i\}$ for each i, $1 \le i \le n$, the event A_i is mutually independent of all the events $\{A_i : (i, j) \notin E\}$. Suppose that D=(V,E) is a dependency digraph for the above events and suppose there are real numbers $x_1, x_2, ..., x_n$ such that $0 \le x_i < 1$ and

$$Pr[A_i] \le x_i \prod_{(i,j)\in E} (1 - x_j) \tag{1.4}$$

for all $1 \le i \le n$. Then $Pr[\wedge_{i=1}^{n} \neg A_i] \ge \prod_{i=1}^{n}(1-x_i)) > 0$. In particular, with positive probability no event A_i holds.

As a corollary of the above general case we get the following more convenient form of the LLL:

Corollary 1.1.2 *[The Local lemma (Symmetric case)]*
Let $\{A_1, A_2, ..., A_n\}$ be events in an arbitrary probability space. Suppose that each event A_i is mutually independent of the set of all the other events A_j but at most d, and that $Pr[A_i] \le p$ for all $1 \le i \le n$. If

$$ep(d + 1) \le 1 \tag{1.5}$$

then $Pr[\wedge_{i=1}^{n} \neg A_i] > 0$.

1.2 Definitions

1.2.1 Some definitions and notation

- $\mathcal{P}$: a finite collection of mutually independent random variables:$P_1, P_2, ..., P_n$ in a fixed probability space Ω
- $\mathcal{A} = \{A_i\}$: a family of events in Ω determined by $\mathcal{P}$.
- $H(A_i|P_{j_1}P_{j_2}...P_{j_k}) = 0$ for some subset $S_i = \{P_{j_1}, P_{j_2}, ..., P_{j_k}\} \subset \mathcal{P}$. Where H(|) denotes the conditional entropy. So, S_i determines A_i.
- An evaluation of the variables in some $S \subseteq \mathcal{P}$ violates A_i if it makes A_i happen.
- *vbl*(A_i) $\triangleq$ the minimal S_i.
- Dependency graph: $G_{\mathcal{A}}$ is defined as the graph on vertex set $\mathcal{A}$ with an edge between events $A, B \in \mathcal{A}$ if $A \neq B$ but $vbl(A) \cap vbl(B) \neq \emptyset$.
- $\Gamma(A_i) = \Gamma_G(A_i)$: The neighbourhood of A_i in G.
- Inclusive neighbourhood of a vertex A_i: $\Gamma^+(A_i) \triangleq \Gamma(A_i) \cup \{A_i\}$.
- $[u]$: The event $A_i \in \mathcal{A}$ represented be node u in G.

1.2.2 A note on the definitions

Definition 1.2.1 *We call an event $A \in \mathcal{A}$ **elementary**, if there is a single evaluation of the variables in $vbl(A)$ violating A.*

Note that, for any elementary event, we can write the probability of it being violated as follows:

$$Pr[A] = \prod_{P \in vbl(A)} Pr[P = p_i]$$

Also, the elementary events A and B are independent if and only if their variable sets are disjoint. Consider the case where the variable sets are not disjoint. If they have a common variable for which they "require" a different value in order to be violated, then they are mutually exclusive and not independent.

$$Pr[A|B] = 0, \;\; Pr[B|A] = 0$$

On the other hand, if they "require" the same value for each of their common variables, they are positively dependent:

$$Pr[A|B] = \prod_{P \in vbl(A) \setminus vbl(B)} Pr[P = p_i] > Pr[A]$$

Note that, the definition of the dependency graph in the previous paragraph uniquely determines a dependency graph, whereas the definition given in lemma 1.1.1 does not. Moreover, it is obvious that if $G \subseteq G'$ (where $\subseteq$ denotes the spanning subgraph relation) are two graphs on n vertices and lemma 1.1.1 holds for G, then it holds for G' as well. So, since the spanning subgraph relation is a partial ordering over the set of all graphs with n vertices, we should only be interested in applying the lemma to a minimal dependency graph w.r.t this relation.

In this sence, the following lemmas show the equivalence of the two definitions mentioned:

Lemma 1.2.2 *For any set $\mathcal{A}$ of elementary events, there is a unique minimal dependency graph (w.r.t spanning subgraph relation).*

- Clearly, each event A_i is mutually independent from any set of events $\mathcal{B} \subset P(\mathcal{A})$ such that

$$\left(\bigcup_{B \in \mathcal{B}} vbl(B) \right) \cap vbl(A_i) = \emptyset \tag{1.6}$$

- Conversely, if A_i is mutually independent from a set $S \subset P(A)$, then 1.6 should hold for $\mathcal{B} = S$. Otherwise, there must be an event $A_j \in S$ such that: $vbl(A_i) \cap vbl(A_j) \neq \emptyset$, so (as mentioned before)

$$Pr[A_i|A_j] \neq Pr[A_i]$$

  and A_i cannot be mutually independent from S.

- Let $\mathcal{B}, \mathcal{B}' \subset P(\mathcal{A})$ be two maximal (w.r.t set inclusion) sets of events, mutually independent from $\mathcal{A}$. Then 1.6 holds also for the set $\mathcal{B} \cup \mathcal{B}'$. Indeed,

$$\left.\begin{array}{l} \mathcal{B} \;\; ind\,from\, \mathcal{A} \Rightarrow (\bigcup_{B \in \mathcal{B}} vbl(B)\ ) \cap vbl(A_i) = \emptyset \\ \mathcal{B}' \;\; ind\,from\, \mathcal{A} \Rightarrow (\bigcup_{B \in \mathcal{B}'} vbl(B)\ ) \cap vbl(A_i) = \emptyset \end{array}\right\} \Rightarrow \left(\bigcup_{B \in \mathcal{B}' \cup \mathcal{B}} vbl(B)\ \right) \cap vbl(A_i) = \emptyset \Rightarrow \tag{1.7}$$

$$\Rightarrow \mathcal{B} \cup \mathcal{B}' \ ind\ from\ \mathcal{A}$$

However, $\mathcal{B}, \mathcal{B}' \subseteq \mathcal{B} \cup \mathcal{B}'$ and using their maximality we get:

$$\mathcal{B} = \mathcal{B} \cup \mathcal{B}' = \mathcal{B}'$$

So, the maximal set is unique.

Also, let for each i, $\mathcal{B}_i$ be this maximal set which corresponds to A_i.

$$A_j \in \mathcal{B}_i \Rightarrow vbl(A_i) \cap vbl(A_j) \neq \emptyset \Rightarrow A_i \in \mathcal{B}_j$$

The second implication comes from equation 1.7 and the fact $\mathcal{B}_j$ is maximal. Due to symmetry, the opposite direction also holds.
So,

$$A_j \in \mathcal{B}_i \Leftrightarrow A_i \in \mathcal{B}_j$$

This enables us to define an undirected graph $G = (V, E)$ as follows:

$$V = \{A_i\} \ \ and \ \ \forall i : \Gamma^+(A_i) = V \setminus \mathcal{B}_i$$

This graph is a spanning subgraph of every other dependency graph by the maximality of $\mathcal{B}_i$'s.
So, every set of elementary events can be mapped to a poset of dependency graphs (w.r.t spanning subgraph relation) having graph G as the unique minimal element.

Lemma 1.2.3 *Let $\mathcal{P}$ be a finite set of mutually independent random variables in a probability space. Let $\mathcal{A}$ be a finite set of elementary events determined by these variables. The (unique) variable dependency graph is isomorphic to the (unique) minimal dependency graph on $\mathcal{A}$ (as defined in lemma 1.1.1).*

Proof:
Since the two graphs are on the same set of vertices, it suffices to prove that:

1. If the events $A_i, A_j \in \mathcal{A}$ do not have disjoint variable sets (i.e they are connected by an edge in the variable dependency graph), then they are also connected by an edge in every possible dependency graph on $\mathcal{A}$.
2. If the events A_i, A_j are connected by an edge in the minimal dependency graph, then their variable sets are not disjoint.

We argued before that two elementary events are independent if and only if the their variable sets are disjoint.

So, the conditional probability that A_i is violated given that A_j is violated, can be written as follows:

$$Pr[A_i|A_j] = \frac{Pr[A_i \cap A_j]}{Pr[A_j]} \neq Pr[A_i] \tag{1.8}$$

So, 1 follows easily, as no set $S \ni A_j$ can be mutually independent from A_i.

In order to prove 2, let the two events A_i, A_j be connected by an edge in the minimal dependency graph $G = (V, E)$, while $vbl(A_i) \cap vbl(A_j) = \emptyset$. So, the events A_i, A_j are independent and (using the proof of lemma 1.2.2) $G' = (V, E - \{A_i, A_j\}) \subseteq G$ is also a dependency graph ($V \setminus \Gamma_{G'}(A_i) = V \setminus (\Gamma_G(A_i) \cup \{A_j\})$ is a set of events, mutually independent from A_i) implying that G is not minimal. So, 2 is proved by contradiction.

1.3 A recent improvement of the lemma

In a recent paper of Bissacot, et al [5] it was proved that the local lemma holds with even weaker conditions. Before discussing their result, we have to rephrase the conditions of the local lemma (general case) as follows:

Let $\mu(A_i) = \dfrac{x(A_i)}{1 - x(A_i)}$. We can easily see that:

$$x(A_i) \in [0, 1) \Rightarrow \mu(A_i) \in (0, +\infty)$$

Moreover, $x(A_i) = \dfrac{\mu(A_i)}{1 + \mu(A_i)}$ Using this, from (1.4) we get:

$$Pr[A_i] \leq \frac{\mu(A_i)}{1 + \mu(A_i)} \prod_{(i,j) \in E} (1 - \frac{\mu(A_j)}{1 + \mu(A_j)}) = \frac{\mu(A_i)}{\varphi_i(\mu)} \tag{1.9}$$

where

$$\varphi_i(\mu) = \prod_{v \in \Gamma_G^+(u_i)} (1 + \mu([v])) = \sum_{R \subseteq \Gamma_G^+(u_i)} \prod_{u \in R} \mu([u]) \tag{1.10}$$

The improvement is shown in the next theorem:

Theorem 1.3.1 ***[Improved Lovász Local Lemma]***

Suppose that G is a dependence graph for the family of events $\{A_i\}$ each one with probability $P(A_i) = p_i$ and there exist $\mu(A_i)$ real numbers in $[0, +\infty)$ such that, for each event A_i,

$$p_i \leq R_i^* = \frac{\mu(A_i)}{\varphi_i^*(\mu)} \tag{1.11}$$

where

$$\varphi_i^*(\mu) = \sum_{R \subseteq \Gamma_G^+(A_i), R \quad indep} \prod_{u \in R} \mu([u]) \tag{1.12}$$

Then $Pr[\wedge_{i=1}^n \neg A_i] > 0$

Remark:

Compare equations (1.10) and (1.12). The improvement brought by the above theorem is heavily depending on the density of the dependency graph. More specifically, for a fixed event A_i, the upper bound on $Pr[A_i]$ is maximally (minimally) increased if the vertices in $\Gamma_G(A_i)$ induce a clique (independent set).

1.4 Previous constructive results

The first attempt to prove the LLL constructively, was made by Beck [4]. Considering the specific problem of hypergraph 2-coloring, he constructed a polynomial-time algorithm which, given a hypergraph in which every edge contains at least k vertices and shares a common vertex with roughly at most $2^{k/48}$ other edges, outputs a 2-coloring of the vertices without producing a monochromatic edge. Note that, the existential symmetric version of the LLL allows for every edge to share vertices with $2^k/e$ other edges.

After that, there are many results that improve the bound in the number of dependencies. Namely, it was improved by the works of: Alon [1] ($2^{k/8}$), Srinivasan [20] ($2^{k/4}$), Moser [15] ($2^{k/2}$ and later [16] roughly 2^{k-5}). Finally, Moser and Tardos [17] made a constructive proof of the asymetric (more general) version of the LLL making only minor restrictions to the original lemma (see below), thus achieving (in the symmetric case) essentially the same bound in the number of dependencies as in the existential version. The only difference from the existential version is the assumption that there is a finite set of mutually independent random variables: $\mathcal{P} = \{P_i\}$ and each event A_i is determined by a subset of $\mathcal{P}$. Very recently, Wesley Pegden[18] improved the conditions on which the MT algorithm works, based on the recent work of Bissacot, et al.[5] who gave an extension of the existential LLL through statistical mechanics.

Chapter 2

Constructive proof for SAT: incompressibility method

2.1 Algorithm

2.1.1 Statement of the recursive algorithm

Given a k-SAT formula $\phi \equiv C_1 \wedge C_2 \wedge ... \wedge C_m$, we will will try to investigate under which conditions the following algorithm terminates after having found a satisfying assignment. We need to consider that there is a total ordering ”$\preceq$” over the set of clauses.

ALgorithm 1:

```
    Local fix
Procedure: Fix($C$)
resample vbl($C$)
repeat
    $D \leftarrow$ the smallest unsatisfied clause in the inclusive neighbourhood of $C$
    Fix($D$)
until the inclusive neighbourhood of $C$ is satisfied.
end procedure
    Global fix
Take a uniformly random truth assignment
repeat
    $C \leftarrow$ smallest unsatisfied clause.
    Fix(C)
until the current assignment satisfies $\phi$.
return the current assignment
```

2.1.2 Intuition

The idea is to define a procedure that generates a string, we will call: the ”LOG of execution”. This can be subsequently used to recover all the random bits used by the algorithm. Such reconstruction is possible if and only if the set of ”possible” random bit strings used by the algorithm can be mapped injectively into the set of ”LOGs of execution” or equivalently we can reconstruct all the random bits used by the algorithm by just reading the LOG. By the term ”possible”, we mean that they do not contain a prefix that leads to termination by itself. Given that the random bits are mutually independent and uniform (i.e the algorithm uses a

Kolmogorov random string), the mapping cannot be injective if the length of he "LOG" is smaller than the input random string. So, a sufficient condition for the algorithm to terminate is the "compressibility of the LOG" (i.e if the average increase in the length of the LOG per resampling is strictly less than the number of extra bits we are able to recover).

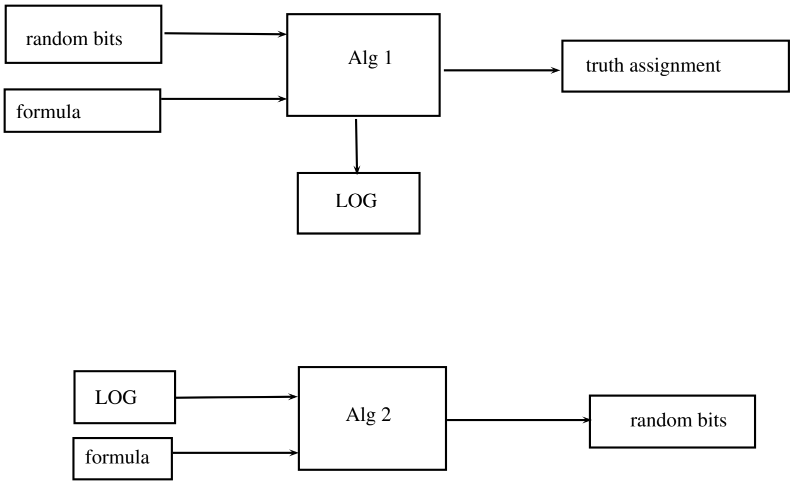

Figure 2.1: The first algorithm takes an input formula and a random bit string and produces a LOG of execution and a (hopefully satisfying) truth assignment. The second algorithm takes the LOG of the first and the same formula as input, and reconstructs the random bits. Simple counting arguments imply that for almost all possible output strings, the length of the LOG cannot be significantly smaller than the output of algorithm 2.

2.1.3 Random bits used by the algorithm

We can partition the random bits used by the algorithm 1, in two categories:

1. The n random bits used for the initial truth assignment.
2. The random bits assigned to the resampled variables: k bits for each resampling.

2.2 Proof

2.2.1 LOG of execution

During the run of the algorithm, we form a LOG in which we encode all the information about which clauses (and with what order) were resampled. We also include the n bits of the final (satisfying) assignment at the end of it. If the LOG does not lead to a satisfying assignment, it will be called a "partial LOG" and will encode all the execution up to some point. The final (not satisfying) assignment is also explicitly written at the end of the "partial LOG" in this case. In order to achieve better compression, we will use two possible encodings for each clause (entry in the LOG). So, the first part of the LOG is a string of symbols from the set $\{C_1, C_2, ..., C_m, C'_1, C'_2, ..., C'_d, T\}$. The first m symbols represent the clauses in an explicit manner. The next d symbols represent the rank of the $i-th$ clause of the LOG in the lexicographical ordering of some previous clause. Finally, the symbol "T" is a terminating symbol that signs the end of a recursive call. Each recursive call corresponds to one symbol from the second category and one symbol "T" in the LOG (in case of partial LOG, we put at he end of LOG_1, as many symbols "T" as the number of recursive calls still running) , whereas each of the other calls corresponds to only one symbol from the first category. The second part of the LOG is a string of n bits which represents a truth assignment. This is a satisfying assignment if and only if the LOG is "complete" (not partial).

More formally:

- $LOG = [LOG_1 LOG_2]$
- $LOG_1 \in \{C_1, C_2, ..., C_m, C'_1, C'_2, ..., C'_d, T\}^*$
- $LOG_2 \in \{0,1\}^n$

2.2.2 Encoding and reconstruction

As mentioned earlier, it is desirable to show that we can always use the encoding of the LOG we have produced, to reconstruct all the random bits, the algorithm has used. The proof will be made in two steps:

1. encoding of the LOG $\Longrightarrow$ LOG (decoding algorithm)
2. LOG $\Longrightarrow$ random bit string (reconstruction)

For the **first step**, we design a uniquely decodable code which is sufficient for our purpose. **Encoding:**

- $C'_j : 1 \leq j \leq d \longleftrightarrow$ "0"+binary representation of j ($\lceil log(d+1) \rceil + 1$ bits)
- $C_i : 1 \leq i \leq m \longleftrightarrow \lceil log(d+1) \rceil + 1$ zeros + binary representation of $i-1$. ($\lceil logm \rceil + \lceil log(d+1) \rceil + 1$ bits)
- $T \longleftrightarrow$ "1" (1 bit)

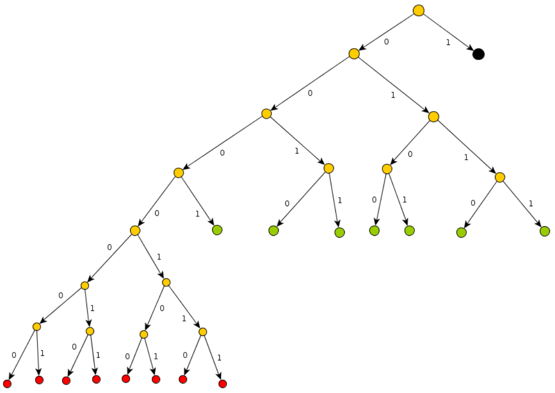

Figure 2.2: Tree of the encoding

Decoding Algorithm:

```
repeat
    repeat
        $s0 \leftarrow$ next bit
        if $s0 ==$ "1" then
```

```
            print "$T$"
        end if
    until $s0 ==$ "0" OR the remaining string is empty.
    if the remaining string is not empty then
        $s0 \leftarrow$ next $\lceil log(d+1) \rceil$ bits
        if $s0$="all zeros" then
            $s1 \leftarrow$ next $\lceil logm \rceil$ bits
            Find the index $i : 1 \leq i \leq m$ such that $i-1$ has binary representation $s1$
            print "$C_i$"
        else
            Find the index $j : 1 \leq j \leq d$ whose binary representation is $s0$
            print "$C'_j$"
        end if
    end if
until the remaining string is empty
```

For the **second step**, we will show that using information about the exact order in which the resamplings were performed (provided by the LOG$\in \{C_1, C_2, ..., C_m, C'_1, C'_2, ..., C'_d, T\}^* \times \{0,1\}^n$), we can reconstruct all the random bits.

Bit reconstruction algorithm:

function reconstruct(ϕ,LOG)

{The argument ϕ includes a $2d$ matrix with entries $vbl(i,j)$: the j-th variable of the i-th clause and another $2d$ matrix with enries $sign(i,j)$: the sign of the j-th variable in the i-th clause (e.g 0 if the variable appears negated and 1 otherwise)}

```
for $i = 1$ to $n$ do
    $pointer(i) \leftarrow i$ {the index of the next value of variable $i$ in the random string }
end for
$s \leftarrow \emptyset$
$c \leftarrow 0$
repeat
    $c \leftarrow c + 1$
    $s \leftarrow$ the next symbol from the LOG
    if $s == C_i$ then
        {not a recursive call}
        Push $i$ {in the recursion stack}
        for $l = 1$ to $k$ do
            $rbits(pointer(vbl(i,l))) \leftarrow 1 - sign(i,l)$    {reconstruct}
            $pointer(vbl(i,l)) \leftarrow n + (c-1) * k + l$    {update pointer}
        end for
    else if $s == C'_j : 1 \leq j \leq d$ then
        {recursive call}
        $s\_parent \leftarrow$ first element of the stack
        Find the index $i$ (in $\phi$) of the $j-th$ neighbour of $s\_parent$.
        Push $i$
        for $l = 1$ to $k$ do
            $rbits(pointer(vbl(i,l))) \leftarrow 1 - sign(i,l)$
            $pointer(vbl(i,l)) \leftarrow n + (c-1) * k + l$
        end for
    else if $s ==$ "$T$" then
        Pop
    end if
until there are no remaining clauses in the LOG
for $i = 1$ to $n$ do
    $rbits(pointer(i)) \leftarrow$ next bit of the LOG
end for
return rbits
```

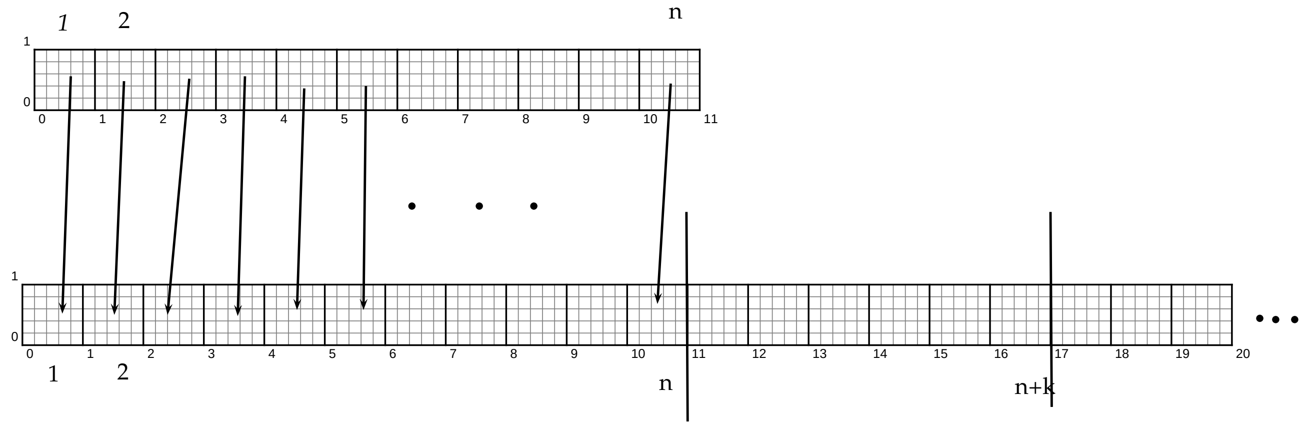

Figure 2.3: Up: array of pointers , Down: Random bit string

2.2.3 Incompressibility method and length of the encoding

Our proof relies on a simple fact: a typical uniformly random bit string is incompressible. In other words, a simple counting argument shows that almost all strings of length l cannot be "represented" by strings of length significantly (e.g asymptotically) less than l. This can be considered as an application of the general incompresibility method. In a typical proof using the incompressibility method, one first chooses a random object from the class under discussion. This object is incompressible. Then one proves that the desired property holds for this object. The argument invariably says that if the property does not hold, then the object can be compressed. This yields the required contradiction.

In our setting, the object is the random bits used by algorithm 1. We would like to prove that algorithm 1 terminates deterministically if uses a specific "typical" (incompressible) string instead of a random bit string. We are going to prove it's contrapositive version: "If algorithm 1 has not terminated after T steps, then it has used a non-typical random string ". The probability of this happening will be exponentially small in $T - T_0$ for some value T_0. So the expected "running time" is shown to be $T_0 + c$, for some constant c.

In order to prove this exponential dependence on T, let $D(T) = (n + k * T) - |enc(LOG)|$. We need to find the conditions under which the restriction of $D(T)$ in an interval of the form $[t, +\infty]$ is a strictly increasing function of T.

Not suprisingly, the symmetric form of the Lovász local lemma gives us good evidence supporting this property. More specifically, we have argued that the encoding of a recursive call of procedure FIX (by another instance of FIX) needs only $\lceil log(d+1) \rceil + 1$ bits and 1 more bit to sign it's termination. Implying that if we restrict the maximum degree to be at most $2^{k-3} - 1$, the desired condition for $D(T)$ could be met. To see this, note that if we add one more recursive call of Fix in the LOG, we use $k-1$ bits to write it, but the algorithm has used k extra random bits. However, it remains to prove that: "As the number of resamplings increases, almost all calls of FIX are recursive calls". This is established by the following lemma:

Lemma 2.2.1 *Given that a call of Fix terminates, then the number of unsatisfied clauses after termination is strictly less that is was before that call.*

Proof: By looking at the algorithm again, we notice that $\forall i : Fix(C_i)$ terminates if and only if all the clauses adjacent to C_i, and C_i itself are satisfied. Let $G = (V, E)$ be the dependency graph for the formula ϕ, and let $V' \subseteq V$ be the set of vertices-clauses which were "explored" by that call of Fix. Obviously, all the decendants of this particular call in the recursion tree must also terminate. So, by induction all clauses whose distance from V' is at most 1 (e.g $\bigcup_{u \in V'} \Gamma^+(u)$) will be satisfied after the termination of Fix. All the clauses in the set $V \setminus \bigcup_{u \in V'} \Gamma^+(u)$ are not altered by this part of the execution. Also, at least one clause in $\bigcup_{u \in V'} \Gamma^+(u)$ (namely C_i) was unsatisfied before. This implies the lemma.

■

Using the above lemma, we can easily prove the following corollary:

Corollary 2.2.2 *At any time during execution, the number of non-recursive calls that have already been made is at most m.*

Proof: Let $U \leq m$ be the number of clauses unsatisfied by the initial random assignment. By lemma 2.2.1, if at least $U \leq m$ non-recursive calls have terminated, the algorithm has found a satisfying assignment, and terminates.

■

We will now try to give some intuition about why the probability that the algorithm does not terminate is zero. We will show that if it does run forever, the the following proposition holds:

$$\exists n, k, T_0 : \forall T > T_0 \; a \; random \; string \; of \; length \; n{+}k{*}T \; has \; a \; description \; of \; length \; at \; most \; n{+}(k{-}1){*}T$$

At this point, we have to view the LOG as a string of two symbols: N, R where N stands for "non-recursive call" and R stands for "recursive call". Each symbol N, R corresponds to a "block" of $\lceil logm \rceil + \lceil log(d+1) \rceil + 1$ *or* $\lceil log(d+1) \rceil + 1$ bits respectively in the original LOG (suppose that a termination symbol "T" is included in R as well). As we have seen in corollary 2.2.2, there cannot be more than m symbols "N". Also, the condition:

$$d \leq 2^{k-3} - 1$$

implies that: $|R| = \lceil log(d+1) \rceil + 1 \leq k-2$. Obviously, $|N| = \lceil logm \rceil + \lceil log(d+1) \rceil + 1 > \lceil logm \rceil > k$ in general.

Now, fix a (probably partial) LOG: $L = C_{i_1} C_{i_2} ... C_{i_l}$ and consider it's length (in bits) as a function of the resampling step t.
Namely,

$$f(t) = |C_{i_1} C_{i_2} ... C_{i_t}|$$

A qualitative graph of this function is demonstrated below:

Figure 2.4: **Horizontal axis:** number of resamplings , **Vertical axis:** number of bits

- The "blue line" is the number of random bits the algorithm has used up to step (i.e resampling) t. It's value is clearly: $n + k \cdot t$
- The "red line" is the graph of the function $f(t)$ for the fixed LOG L. Notice that this is a piecewise linear function whose slope changes between two values: $\lceil logm \rceil + k - 1$ and $k - 1$.
- The "green line" is an upper bound for $f(t)$ for all steps t and for any fixed LOG. The existence of an upper bound of this form is quaranteed by corollary 2.2.2. It follows from that, that the number of "jumps" in the red line are at most $U \leq m$.
- The red line coinsided with the green line if it happens that it makes U non-recusrive calls in the beginning ($slope = \lceil logm \rceil + k - 3 + 1 = \lceil logm \rceil + k - 2$) and continues with recursive calls only ($slope = k - 2 + 1 = k - 1$).
- The starting point for all the three lines is $(0, n)$ and corresponds to

Let $(T_0, n + k \cdot T_0)$ be the coordinates of the intersection of the blue and green lines.

Claim 2.2.3 *If algorithm* 1 *is given as input a formula with maximum clause degree:* $d \leq 2^{k-3} - 1$ *and hasn't terminated after* $T > T_0$ *for some* $T_0 \leq m \cdot \lceil logm \rceil$ *steps, then the random bit string it has used can be "represented" by another bit string which is at least* $T - T_0$ *bits shorter.*

Proof: By definition, the green curve is the graph of the function

$$g(t) = (k - 1) \cdot t + C$$

for the last part (which is where the intersection takes place). Also, we have $f(t) = n + k \cdot t$. At the intersection, it holds that

$$\begin{aligned} f(T_0) &= g(T_0) \\ n + k \cdot T_0 &= (k - 1) \cdot T_0 + C \\ T_0 &= C - n \end{aligned}$$

For the value of the constant C consider the following: The first green line segment has equation:

$$g_0(t) = n + (\lceil logm \rceil + \lceil log(d + 1) \rceil + 1) \cdot t$$

Also,

$$\begin{aligned} g_0(U) &= g(U) \\ n + (\lceil logm \rceil + \lceil log(d + 1) \rceil + 1) \cdot U &= (k - 1) \cdot U + C \\ n + (\lceil logm \rceil + k - 2) \cdot U &= (k - 1) \cdot U + C \\ C &= n + U \cdot (\lceil logm \rceil - 1) \end{aligned}$$

So,

$$T_0 = U \cdot (\lceil logm \rceil - 1) \leq m \cdot \lceil logm \rceil$$

For $T > T_0$ the difference betwwen the blue and red curve yields:

$$\begin{aligned} f(t) - g(t) &= n + k \cdot t - (k-1) \cdot t - C && (2.1)\\ &= n + t - n - U \cdot (\lceil logm \rceil - 1) && (2.2)\\ &= t - U \cdot (\lceil logm \rceil - 1) = t - T_0 && (2.3) \end{aligned}$$

This proves the claim as for every t, $g(t)$ is an upper bound for the value of the red line at step t. Recall that, the red line represents the binary length of the LOG, and as we have seen before there is an algorithm which uses this LOG of length at most $g(t)$ and some extra constant size (with respect to t) information (i.e the formula) and recostructs the random string of length $f(t)$.

■

Theorem 2.2.4 *Let ϕ be a k-CNF formula, in which every clause has a common variable with at most $2^{k-3} - 2$ other clauses (i.e $d \leq 2^{k-3} - 1$). Then ϕ is satisfiable. Moreover, algorithm 1 on input ϕ terminates after an expected number of steps at most $m \cdot \lceil logm \rceil + 1$. More specifically, the probability that it has not terminated after $m \cdot \lceil logm \rceil + c$ steps is at most 2^{-c}.*

Proof: Suppose that there is an unsatisfiable k-CNF formula ϕ in which every clause has a common variable with at most $d = 2^{k-3} - 1$ clauses (including itself). Now, we are able to "compress" a random string. More specifically, if we give ϕ as an input to algorithm 1, it will never terminate as ϕ is unsatisfiable. However, we can produce the LOG of ececution and encode it as we described above. Let $|enc(LOG(t))|$ be the length of this encoding for the first t steps, and $RB(t)$ be the number of random bits used. The following holds:

$$\lim_{t\to\infty} \frac{|enc(LOG(t))|}{RB(t)} \leq \lim_{t\to\infty} \frac{C + (k-1) \cdot t}{n + k \cdot t} = \frac{k-1}{k}$$

The random bit string is an incompressibe object. We just showed that in can be recovered by a bit string which is shorter by a constant factor. So this is a contradiction as the incompressibility method indicates. So, ϕ is satisfiable.

Now, fix a number t_0 and consider the set S_{t_0} of all bit strings of length $n + k \cdot t_0$:

$$S_{t_0} = \{0,1\}^{n+k\cdot t_0}$$

Also let L be set of all possible LOGs that can be produced by at most $n + k \cdot t_0$ bits. Notice that, the cardinality of L is, in general, less that the cardinality of S_{t_0}. The reason for this is that if a string $w \in S_{t_0}$ has a prefix that makes the algorithm on input ϕ terminate, then all strings having this prefix, will correspond to the same LOG. We say that these strings form an equivalence class of S_{t_0}. So, the number of equivalence classes will be equal to the cardinality of the set L. We are interested in the set of strings that do not contain a prefix that leads to termination. Namely, the random bit strings that either terminate at step t_0 or haven't terminated yet. We call that set S'_{t_0}. The fact that we can use the LOG to reconstruct the random string, implies that there is no pair of distinct strings: $w_1, w_2 \in S'_{t_0}$ that produce the same LOG. Otherwise, the reconstruction would not be possible.

$$\forall w_1, w_2 \in S'_{t_0} : w_1 \neq w_2 \Rightarrow LOG(w_1) \neq LOG(w_2) \tag{2.4}$$

Also, let T_1 be the step at which the algorithm terminates. The probability of termination can now be expressed as follows:

$$Pr[T_1 \geq t_0] = \frac{|S'_{t_0}|}{|S_{t_0}|}$$

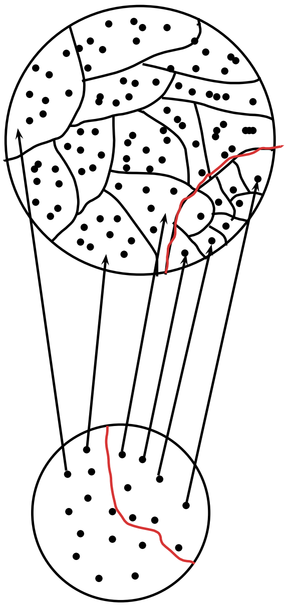

Figure 2.5: Up: Set of all possible strings of length $n + k \cdot t_0$, Down: Set of all possible LOGS produced by at most $n + k \cdot t_0$ random bits. The set of strings is partitioned into equivalence classes of strings producing the same LOG (only if they have a common terminating prefix). Each LOG corresponds to exactly one equivalence class. The LOGS with t_0 clauses (which either terminate at step t_0 or not) are mapped to a single string

However, equation 2.4 implies that $|S'_{t_0}| \leq |L|$. Also, if $t_0 = T_0 + c$ then for every LOG $l \in L$, $|l| \leq n + k \cdot t_0 - c$ (see claim 2.2.3).

So, an upper bound on the number of possible LOGs is:

$$|L| \leq 2^{n+k \cdot t_0 - c}$$

So,

$$Pr[T_1 \geq T_0 + c] = \frac{|S'_{t_0}|}{|S_{t_0}|} \leq \frac{|L|}{|S_{t_0}|} \leq \frac{2^{n+k\cdot t_0 - c}}{2^{n+k\cdot t_0}} = 2^{-c} \Leftrightarrow$$

$$E[T_1 - T_0] = \sum_{i=1}^{\infty} Pr[T_1 - T_0 \geq i] \leq \sum_{i=1}^{\infty} 2^{-i} = 1 \Leftrightarrow$$

So,

$$E[T_1] \leq T_0 + 1 = U \cdot \lceil logm \rceil + 1 \leq m \cdot \lceil logm \rceil + 1$$

■

Chapter 3

The Moser-Tardos Algorithm and witness trees

3.1 Sequential algorithm

3.1.1 Preliminaries

As it was mentioned before, we will work under the assumption that there is a finite set of mutually independent random variables: $\mathcal{P} = \{P_i\}$ and each event A_i is determined by a subset of $\mathcal{P}$ as we can see in the figure below along with the corresponding dependency graph. Note that: for a given vertex u of the dependency graph, the set of vertices non-adjacent to u represent a set of events which are mutually independent of [u]. However this set is not guaranteed to be maximal (e.g two events A_i, A_j could be independent even though $vbl(A_i) \cap vbl(A_i) \neq \emptyset$).

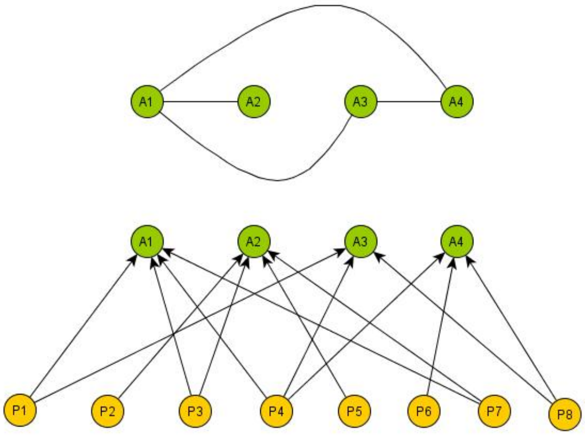

Figure 3.1: Up: Dependency graph , Down: Events and variables

The following non-recursive algorithm outputs an evaluation of the random variables $\{P_i\}$ which does not violate any of the events $\{A_i\}$ if the sufficient conditions of the local lemma are satisfied.

Theorem 3.1.1 *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 with a dependency graph G determined by these variables.*

```
funcion sequential_lll($\mathcal{P}, \mathcal{A}$)

for all $P \in \mathcal{P}$ do
    $u_p \leftarrow$ a random evaluation of $P$
end for
while $\exists A \in \mathcal{A} : A$ is violated do
    pick an arbitrary violated event $A \in \mathcal{A}$.
    for all $P \in vbl(A)$ do
        $u_p \leftarrow$ a new random evaluation of $P$
    end for
end while
return $(u_P)_{P \in \mathcal{P}}$
```

Figure 3.2: The sequential solver

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

$$\forall A_i \in \mathcal{A} : Pr[A_i] \leq x(A_i) \prod_{B \in \Gamma_G(A_i)} (1 - x(B)) \tag{3.1}$$

then there exists an assignment of values to the variables P_i not violating any of the events in $\mathcal{A}$. Moreover the randomized algorithm described above resamples an event $A_i \in \mathcal{A}$ at most an expected $x(A_i)/(1 - x(A_i))$ times before it finds such an evaluation. Thus, the expected total number of resampling steps is at most $\sum_{A_i \in \mathcal{A}} \dfrac{x(A_i)}{1 - x(A_i)}$

We will need some definitions and a couple of lemmas before we can prove the theorem.

Definitions:

- Log of execution: Let $C : N \rightarrow \mathcal{A}$ list the events as they have been selected for resampling in each step. We call C the log of the execution.
- Witness tree $\tau = (T, \sigma_T)$: a finite rooted tree T together with a labelling $\sigma_T) : V(T) \rightarrow \mathcal{A}$ of its vertices (with events) such that the children of a vertex $u \in V(T)$ receive labels from $\Gamma^+(\sigma_T(u))$.
- A witness tree is called proper if distinct children of the same vertex always receive distinct labels.
- Notation: $V(\tau) \triangleq V(T)$, $[u] \triangleq \sigma_T(u)$.
- We say that the witness tree τ occurs in the log C if there exists $t \in N$ such that $\tau_C(t) = \tau$.

Before doing that, we will introduce the witness trees which by definition are constructed by the following iterative procedure:

1. $\tau_C^{(t)}(t) \triangleq$ an isolated root vertex labelled C(t) (the event that was resampled at step t).

2. For each i=t-1,t-2,...,1 we find the "deepest" vertex v such that $C(i) \in \Gamma^{+}_{G}([v])$ and attach to it a new vertex u labelled C(i) to construct $\tau^{(i-1)}_{C}(t)$ from $\tau^{(i)}_{C}(t)$. If no such vertex exists we define: $\tau^{(i-1)}_{C}(t) = \tau^{(i)}_{C}(t)$.

3. Finally, $\tau_C(t) = \tau^{(1)}_{C}(t)$

Note that, given the dependency graph, the family of the witness trees has 2 parameters (the 3rd in only used in the construction process). Namely, there is a injection from the tuples (C,t) to the set of witness trees, where C denotes the execution log and t is the resampling step.

Here is an example of a witness tree construction:

Execution log: $C = [A_3, A_2, A_4, A_3, A_2, A_1, A_2, A_3]$

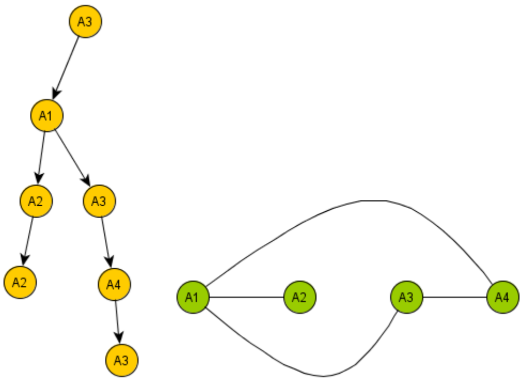

Figure 3.3: Left: Example of witness tree , Right: Dependency graph

Clearly, given the execution log and the dependency graph (green nodes) we can construct a unique witness tree.

Lemma 3.1.2 *Let τ be a fixed witness tree and C the (random) log produced by the algorithm.*
(i) If τ occurs in C, then τ is proper.
(ii) The probability that τ occurs in C is at most $\prod_{v \in V(\tau)} Pr[[v]]$.

Proof: (i): The procedure cannot output a tree in which 2 vertices v,u with the same label have a common parent w. If w.l.o.g vertex u was created after vertex v, then u could be placed as a child of v (which has greater depth than w). So, (i) is proved by contradiction.
(ii): Consider the following procedure that we call τ -check: In an order of decreasing depth (e.g., reversed breadth first search order) visit the vertices of τ and for a vertex v take a random evaluation of the variables in $vbl([v])$ (according to their distribution, independent of possible earlier evaluations) and check if the resulting evaluation violates $[v]$. We say that the τ -check passes if all events were violated when checked.

Clearly, Pr[τ-check passes]=$\prod_{v \in V(\tau)} Pr[[v]]$. We are going to prove that:

$$\tau \textit{ occurs in } C \;\Rightarrow\; \textit{the } \;\tau - \textit{check (on the same random source) passes}$$

We assume that for each variable $P_i \in P$ the random source produces an infinite list of independent random samples $P_i^{(0)}, P_i^{(1)}, ...$, and whenever the algorithm calls for a new random sample of P_i we pick the next unused value from this sequence.

Let $S_v(P_i) = \{w \in V(\tau) : d(w) > d(v) \wedge P_i \in vbl(w) \cap vbl(v)\}$ where d() denotes the depth of a node. From the construction of the witness tree, we have that all vertices of the same depth correspond to mutually independent events. So, in our decreasing depth order every event P_i is resampled at most once per depth value taking the value $P_i^{(|S_v(P)|)}$ no matter in which order we choose to resample events of the same depth. From that we can also deduce that when P_i is resampled for the k-th time in the algorithm, it is resampled for the k-th time in the τ-check as well because we take decreasing depth order. Finally, the fact that a vertex v is evaluated by the τ-check means that:

$$\begin{aligned} The\ event\ [v]\ is\ in\ the\ execution\ log\ C \Rightarrow\ & it\ was\ violated\ the\ time\ it\ was\ written\ in\ C \\ \Rightarrow\ & the\ \tau - check\ will\ find\ that\ [v]\ is\ violated \end{aligned}$$

□

We are now ready to calculate the expected running time of the algorithm.

Note that the number (N_{A_i}) of times an event A_i is resampled equals the number of occurrences of A_i in C (the execution log) as well as the number of distinct proper witness trees, with A_i as the root, occurring in C.

To prove this, recall that if t_j is the number of the j-th time step in which the event A_i is resampled then $\tau_C(t_j)$ contains exactly j nodes with label A_i. So, $\tau_C(t_j) = \tau_C(t_k) \Rightarrow j = k$.

3.1.2 Random generation of witness trees

Let us fix an event $A_i \in \mathcal{A}$ and consider the following Galton-Watson branching process for generating a proper witness tree having it's root labelled A_i:

1. Produce a singleton vertex labelled A_i.
2. For each new vertex v, and for each event $B \in \Gamma^+([v])$ add to v a child node with the label B independently with probability $x(B)$.
3. Repeat step 2 until no new nodes are added.

Note: This process may continue forever with positive probability.

The following lemma will finally help us count the number of resamplings:

Lemma 3.1.3 ***Branching Lemma [17]:*** *Let τ be a fixed proper witness tree with it's root vertex labelled A_i. The probability p_τ that the Galton-Watson process described above yields exactly the tree τ is*

$$p_\tau = \frac{1 - x(A_i)}{x(A_i)} \prod_{v \in V(t)} x'([v]) \tag{3.2}$$

where

$$x'(B) \triangleq x(B) \prod_{C \in \Gamma(B)} (1 - x(C)) \tag{3.3}$$

Proof: Let $W_v \subset \Gamma^+([v])$ be the set of inclusive neighbours of $[v]$ that do not occur as a label of some child node of v. Since the root node A_i is always produced we have that:

$$p_\mathcal{T} = \frac{1}{x(A_i)} \prod_{v \in V(\tau)} (x([v]) \prod_{u \in W_v} (1 - x([u]))) \tag{3.4}$$

Note that, in the 2nd product we don't consider the children of v in order to avoid duplicates. Equivalently, we have:

$$p_\mathcal{T} = \frac{1 - x(A_i)}{x(A_i)} \prod_{v \in V(\tau)} (\frac{x([v])}{1 - x([v])} \prod_{u \in \Gamma^+([v])} (1 - x([u]))) \tag{3.5}$$

We divided by $(1 - x([v]))$ in the first product in order to change the subscript in the second to be the (more convenient) inclusive neighbourhood of v.
if we now replace inclusive with exclusive neighbourhood, we get:

$$\Leftrightarrow p_\mathcal{T} = \frac{1 - x(A_i)}{x(A_i)} \prod_{v \in V(\tau)} (x([v]) \prod_{u \in \Gamma([v])} (1 - x([u]))) \tag{3.6}$$

$$= \frac{1 - x(A_i)}{x(A_i)} \prod_{v \in V(\tau)} x'([v]) \tag{3.7}$$

□

3.1.3 Counting the number of resamplings

Let T_{A_i} be the set of all proper witness trees having their root labelled A_i.
By linearity of expectation:

$$E[N_{A_i}] = \sum_{\tau \in T_{A_i}} Pr[\tau \ \ occurs \ \ in \ \ C] \tag{3.8}$$

$$\leq \sum_{\tau \in T_{A_i}} \prod_{v \in V(\tau)} Pr[[v]] \tag{3.9}$$

$$\leq \sum_{\tau \in T_{A_i}} \prod_{v \in V(\tau)} x'([v]) \tag{3.10}$$

By lemma 5.3 we have:

$$E[N_{A_i}] \leq \sum_{\tau \in T_{A_i}} \prod_{v \in V(\tau)} x'([v]) \tag{3.11}$$

$$= \frac{x(A_i)}{1 - x(A_i)} \sum_{\tau \in T_{A_i}} p_\mathcal{T} \tag{3.12}$$

$$\leq \frac{x(A_i)}{1 - x(A_i)} \tag{3.13}$$

This bounds the expected number of resamplings for each event and proves theorem 3.1.1.

Remark:

The reason we have chosen that particular Galton - Watson process is the following.

We want to construct non-zero probabilities for every witness tree possible to occur in the log, which are also big enough, so that if write each term of equation (3.11) in the form: $cp_\mathcal{T}$

(where τ is the witness tree that corresponds to that term), c will have a small value. The value of c is critical, since the expected number of resamplings finally depends on that.

As we will see later, we can get better results using a modified Galton-Watson process which satisfies both the above properties and guarrantees even greater probabilities for each witness tree occuring in the log.

3.2 Parallel version of the algorithm

It turns out that there is a very similar parallel version of the MT algorithm discussed above, which has a significantly greater performance (as it was also proved in [17]).

The algorithm should now be written as follows:

```
funcion parallel_lll($\mathcal{P}, \mathcal{A}$)

for all $P \in \mathcal{P}$ do
    $u_p \leftarrow$ a random evaluation of $P$
end for
while $\exists A \in \mathcal{A} : A$ is violated do
    $S \leftarrow$ a maximal independent set in the subgraph of $G_A$ induced by all events which are
    violated, constructed in parallel.
    for all $P \in \bigcup_{A \in S} vbl(A)$ do
        $u_p \leftarrow$ a new random evaluation of $P$ {in parallel}
    end for
end while
return $(u_P)_{P \in \mathcal{P}}$
```

Figure 3.4: The parallel solver

It's performance is quarranted by the following theorem:

Theorem 3.2.1 *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 $\varepsilon > 0$ and there exists an assignment of reals $x : \mathcal{A} \rightarrow (0,1)$ such that*

$$\forall A_i \in \mathcal{A} : Pr[A_i] \leq (1-\varepsilon)x(A_i) \prod_{B \in \Gamma_{\mathcal{A}}(A_i)} (1 - x(B)) \tag{3.14}$$

then the parallel version of our algorithm takes an expected $O(\frac{1}{\varepsilon} log \sum_{A_i \in \mathcal{A}} \frac{x(A_i)}{1-x(A_i)})$ steps before it finds an evaluation violating no event in $\mathcal{A}$.

Note that, if x_{A_i} is bounded away from 1 for every i, the the $\sum_{A_i \in \mathcal{A}} \frac{x(A_i)}{1-x(A_i)}$ is proportional to the number of events. So, the paralell algorithm allows an exponentially greater number of events for the same number of resamplings needed.

Proof:

In the parallel case we form the execution log C sorting the events in increasing step number. We also call $\underline{s_j}$ the segment of C that corresponds to the events resampled at step j (whose order is chosen arbitrarily).

The main idea of the proof is to show that in every step of the parallel algorithm, an extra node is added to any witness tree having a root from that step(segment). Namely,the size of every witness tree whose root belongs to s_j is strictly greater that the size of any other having a root in s_{j-1}. Then, we can use this to get a better bound on the running time. Keep in mind that larger witness trees are less likely to occur in the log, so the probability that the many resamplings will be needed is low as well.

The following lemma implies something even stronger: "Every new step strictly increases the depth of the witness trees".

Lemma 3.2.2 *If* $t \in s_j$ *, then the depth of* $\tau_C(t)$ *is j - 1.*

Proof:
Let:

- $\underline{t_k \triangleq \sum_{i=1}^{k-1} |s_i| + 1}$.
- $\tau_k = \tau_C^{(t_k)}(t)$ for $k \leq j$ and for some fixed $t \in s_j$.

So, t_k is the first number in the segment s_k and τ_k is the "partial" witness tree which describes the resamplings done at all steps between k and j. Also, the events resampled in each parallel step are independent.

So by definition we have: $\tau_j = \tau_C^{(t_j)}(t)$ which is a tree with only one vertex labelled : C(t).

For $k < j$, τ_{k+1} is a subgraph of τ_k , which has some extra vertices corresponding to the k-th parallel step of the algorithm. As these vertices have independent labels, they add at most one to the depth (none can be a child of another).

We will prove that they also add at least one to the depth: Consider a vertex v of τ_{k+1} of maximal depth=d. If τ_k has no vertex with depth d+1, then from the parallel step k to the resampling corresponding to v no event from $\Gamma^+([v])$ was resampled.

So, $[v]$ was violated since step k and the algorithm did not chose it for resampling so it did not chose a maximal independent set at step k, which is a contradiction.

To finish the proof, notice that $\tau_C(t) = \tau_1$.

□

Lemma 3.2.3 *Let X be an integer-valued random variable. If there is a constant* $\varepsilon \in (0, 1]$ *and another constant* $C > 1$*, such that:*

$$\forall k \in \mathbb{N}^* : \quad Pr[X \geq k] \leq (1 - \varepsilon)^k \cdot C \tag{3.15}$$

then

$$E[X] \leq \frac{1}{\varepsilon} log(C) + \frac{1 + \varepsilon}{\varepsilon} \tag{3.16}$$

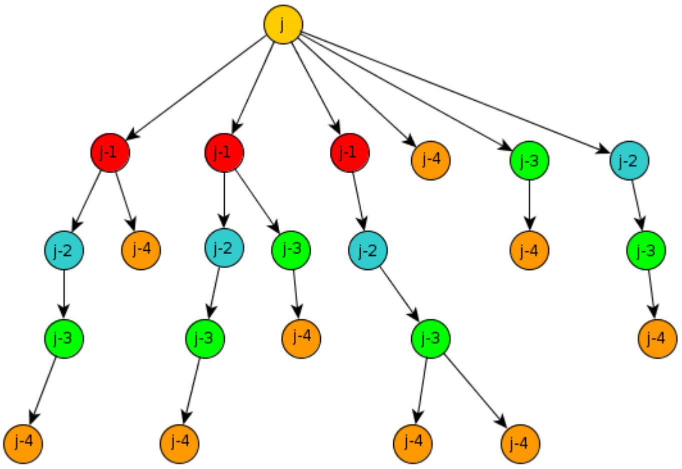

Figure 3.5: Different colors correspond to different resampling steps. There should always be a path containing every color.

Proof:

Let k_0 be the smallest integer such that: $C \cdot (1-\varepsilon)^{k_0} < 1$. By taking logarithms, we get: $k_0 > \dfrac{log(C)}{log(\dfrac{1}{1-\varepsilon})}$.

Since k_0 is the smallest integer we can write:

$$\frac{log(C)}{log(\dfrac{1}{1-\varepsilon})} + 1 > k_0 > \frac{log(C)}{log(\dfrac{1}{1-\varepsilon})} \tag{3.17}$$

Using the inequality: $logx \leq x - 1$ we have:

1.

$$log(\frac{1}{1-\varepsilon}) \leq \frac{1}{1-\varepsilon} - 1 = \frac{\varepsilon}{1-\varepsilon} \Leftrightarrow \frac{1}{log[(1-\varepsilon)^{-1}]} \geq \frac{1-\varepsilon}{\varepsilon} = \frac{1}{\varepsilon} - 1$$

2.

$$\begin{aligned} log(1-\varepsilon) \leq 1 - \varepsilon - 1 = -\varepsilon \Leftrightarrow -log(1-\varepsilon) \geq \varepsilon \\ \Leftrightarrow \frac{1}{log[(1-\varepsilon)^{-1}]} \leq \frac{1}{\varepsilon} \end{aligned}$$

Thus, from (3.17) we get:

$$\frac{1}{\varepsilon}log(C) + 1 > k_0 > (\frac{1}{\varepsilon} - 1)log(C) \tag{3.18}$$

Let $X' = X - k_0$. Then we have:

$$Pr[X' \geq k] = Pr[X \geq k + k_0] \leq C \cdot (1-\varepsilon)^{k+k_0} < (1-\varepsilon)^k \tag{3.19}$$

The random variable X', takes only integer values greater or equal to $-k_0$.
So,

$$\begin{aligned} E[X'] &= \sum_{i=-k_0}^{\infty} i \cdot Pr[X' = i] \\ &\leq \sum_{i=1}^{\infty} i \cdot Pr[X' = i] = \sum_{i=1}^{\infty} Pr[X' \geq i] \\ &< \sum_{i=1}^{\infty} (1-\varepsilon)^i = \frac{1}{1-(1-\varepsilon)} = \frac{1}{\varepsilon} \qquad (3.20) \end{aligned}$$

Thus,

$$E[X] \leq k_0 + \frac{1}{\varepsilon} \leq \frac{1}{\varepsilon} log(C) + \frac{1+\varepsilon}{\varepsilon} \tag{3.21}$$

□

Let: $\underline{Q(k) \triangleq Pr[\sharp steps \geq k]}$.

By Lemma 3.2.2, some witness tree of depth k - 1 (which has at least k vertices) must occur in the log. Let $\underline{T_{A_i}(k)}$ be the set of witness trees in T_{A_i} having at least k vertices.

$$\begin{aligned} Q(k) &\leq \sum_{A_i \in \mathcal{A}} \sum_{\tau \in T_{A_i}(k)} Pr[\tau \ \ occurs \ \ in \ \ the \ \ log \ \ C] && (3.22) \\ &\leq \sum_{A_i \in \mathcal{A}} \sum_{\tau \in T_{A_i}(k)} \prod_{v \in V(\tau)} Pr[[v]] && (3.23) \\ &\leq (1-\varepsilon)^k \sum_{A_i \in \mathcal{A}} \sum_{\tau \in T_{A_i}(k)} \prod_{v \in V(\tau)} x'([v]) && (3.24) \end{aligned}$$

where the last inequality follows from the assumption in Theorem 3.2.1.

$$\begin{aligned} Q(k) &\leq (1-\varepsilon)^k \sum_{A_i \in \mathcal{A}} \sum_{\tau \in T_{A_i}(k)} \prod_{v \in V(\tau)} x'([v]) && (3.25) \\ &= (1-\varepsilon)^k \sum_{A_i \in \mathcal{A}} \frac{x(A_i)}{1-x(A_i)} \sum_{\tau \in T_{A_i}(k)} p_\tau && (3.26) \\ &\leq (1-\varepsilon)^k \sum_{A_i \in \mathcal{A}} \frac{x(A_i)}{1-x(A_i)}. && (3.27) \end{aligned}$$

if we set: $\underline{M = \sum_{A_i \in \mathcal{A}} \frac{x(A_i)}{1-x(A_i)}}$, and T be the number of steps, we have:

$$Pr[T \geq k] \leq M \cdot (1-\varepsilon)^k \tag{3.28}$$

So, by lemma 3.2.3 ,

$$E[T] = O(\frac{1}{\varepsilon} log(M)) \tag{3.29}$$

■

3.3 Dealing with superpolynomially many bad events

The following theorem gives sufficient conditions so that the number of resamplings of the MT algorithm remains polynomial in the number of variables $n = |\mathcal{P}|$, even if the number of events is exponential in n.

Theorem 3.3.1 *Suppose there is an $\varepsilon \in [0, 1)$ and an assignment of reals $x : A \to (0, 1)$ such that:*

$$\forall A_i \in \mathcal{A} : Pr[A_i] \leq (1 - \varepsilon)x(A_i) \prod_{B \in \Gamma(A_i)} (1 - x(B)) \tag{3.30}$$

With δ denoting $\min_{A_i \in \mathcal{A}} x_{A_i} \prod_{B \in \Gamma(A_i)} (1 - x(B))$, we have

$$T := \sum_{A_i \in \mathcal{A}} x_{A_i} \leq nlog(1/\delta) \tag{3.31}$$

Furthermore:

1. *If $\varepsilon = 0$, then the expected number of resamplings done by the MT algorithm is at most $v_1 = T \max_{A_i \in \mathcal{A}} \frac{1}{1 - x(A)}$, and for any parameter $\lambda \geq 1$, the MT algorithm terminates within λv_1 resamplings with probability at least $1 - 1/\lambda$.*

2. *If $\varepsilon > 0$, then the expected number of resamplings done by the MT algorithm is at most $v_2 = O(\frac{n}{\varepsilon} log \frac{T}{\varepsilon})$, and for any parameter $\lambda \geq 1$, the MT algorithm terminates within λv_2 resamplings with probability $1 - e^{-\lambda}$.*

Proof:

The first part of the theorem follows immediately from theorem 3.1.1, because we have proved that if $\varepsilon = 0$, then the expected number of resamplings cannot be more than $\sum_{A_i \in \mathcal{A}} \frac{x_{A_i}}{1 - x_{A_i}} \leq T \cdot \max_{A_i \in \mathcal{A}} \frac{1}{1 - x_{A_i}}$. Also, the rest of the first part is proven by a simple application of Markov's inequality.

Now, we will prove the bound on T.
Let,

$$A_{P_i} = \{A_i \in \mathcal{A} | P_i \in vbl(A_i)\} \tag{3.32}$$

Now, it is sufficient to prove that

$$\forall P_i : \sum_{B \in A_{P_i}} x_B \leq log(1/\delta) \;\; since \;\; A = \cup_{P_i \in \mathcal{P}} A_{P_i} \;\; and \;\; |\mathcal{P}| = n$$

Let $A_i \in A_P$, $P \in \mathcal{P}$ be the event with the smallest x-value.
By definition, we have:

$$\delta \leq x_{A_i} \prod_{B \in A_P - \{A_i\}} (1 - x_B) = \frac{x_{A_i}}{1 - x_{A_i}} \prod_{B \in A_P} (1 - x_B) \tag{3.33}$$

- If $x_{A_i} \leq \frac{1}{2}$, then $\frac{x_{A_i}}{1-x_{A_i}} \leq 1 \Rightarrow \delta \leq \prod_{B \in A_P}(1-x_B) \leq e^{-\sum_{B\in A_{P_i}} x_B}$. Then by taking the logarithm of both sides, we get:

$$\sum_{B \in A_{P_i}} x_B \leq ln(1/\delta) < log(1/\delta)$$

- If $x_{A_i} > \frac{1}{2}$, let $B_1 \in A_P - \{A_i\}$.
  Then,

$$\delta \leq x_{A_i} \prod_{B \in A_P - \{A_i\}} (1-x_B) = x_{A_i}(1-x_{B_1}) \prod_{B \in A_P - \{A_i, B_1\}} (1-x_B) \tag{3.34}$$

$$\leq x_{A_i}(1-x_{B_1}) e^{-\sum_{B \in A_P - \{A_i, B_1\}} x_B} \tag{3.35}$$

Since x_{A_i} is the smallest x-value, $\frac{1}{2} \leq x_{A_i} \leq x_{B_1} \leq 1$.
Then, by using some calculus (see [10]) we can show that

$$x_{A_i}(1-x_{B_1}) \leq e^{-(x_{A_i}+x_{B_1})} \tag{3.36}$$

So, we get the same upper bound for δ as before:

$$\delta \leq e^{-\sum_{B \in A_{P_i}} x_B} \tag{3.37}$$

So, $\sum_{B \in A_{P_i}} x_B \leq ln(1/\delta) < log(1/\delta)$ as required.

We are now ready to prove the 2nd part of the theorem. In the proof of theorem 3.2.1, we have proved the following:

$$\sum_{A_i \in \mathcal{A}} \sum_{\tau \in T_{A_i}(k)} Pr[\tau \ \ occurs \ \ in \ \ the \ \ log \ \ C] \leq (1-\varepsilon)^k \sum_{A_i \in \mathcal{A}} \frac{x_{A_i}}{1-x_{A_i}} = w \tag{3.38}$$

So, the expected number of witness trees with at least k vertices is at most w. So, if we let k_l be the smallest integer such that $w < \frac{1}{l}$, then $k_l \leq \frac{1}{\varepsilon} log(l \cdot \sum_{A_i \in \mathcal{A}} \frac{x_{A_i}}{1-x_{A_i}}) + 1$ (see the proof of lemma 3.2.3).

By applying Markov's inequality, the probability that there is at least one witness tree of size at least k, is not greater than the expected number of w.t of size at least k. More formally,

$$Pr[\exists \tau : |\tau| \geq k \wedge \tau \ \ occurs \ \ in \ \ the \ \ log] \leq E[\sharp w.t : |\tau| \geq k] \tag{3.39}$$

Also,

$$Pr[\max_{\tau \ \ occuring \ \ in \ \ C} |\tau| \geq k] \leq Pr[\exists \tau : |\tau| \geq k \wedge (\tau \ \ occurs \ \ in \ \ the \ \ log)] \tag{3.40}$$

So, if the expected number of witness trees larger than k decreases exponentially in k, then the probability that the size of the biggest witness tree exceeds k also decreases exponentially in k.

In particular,

$$Pr[\max_{\tau \ \ occuring \ \ in \ \ C} |\tau| \geq k] \leq (1-\varepsilon)^k \sum_{A_i \in \mathcal{A}} \frac{x_{A_i}}{1-x_{A_i}} \tag{3.41}$$

Thus, by lemma 3.2.3 the expected maximum size of $|\tau|$ is $O(\frac{1}{\varepsilon} log(\sum_{A_i \in \mathcal{A}} \frac{x_{A_i}}{1-x_{A_i}}))$. It is also argued in [17] that for every assignment of x-values satisfying (3.30) there is an equivalent assignment, still satisfying all the requirements, which has all x-values bounded away from 1.

In particular, if

$$\forall A_i \in \mathcal{A} : Pr[A_i] \leq (1-\varepsilon)x(A_i) \prod_{B \in \Gamma(A_i)} (1 - x(B)) \tag{3.42}$$

then if we set $x'(A_i) = (1 - \frac{\varepsilon}{2})x(A_i)$, we will get:

$$\forall j : x'(A_j) \leq 1 - \varepsilon/2 \ \wedge \ \forall A_i \in \mathcal{A} : Pr[A_i] \leq (1-\varepsilon/2)x(A_i) \prod_{B \in \Gamma(A_i)} (1 - x(B)) \tag{3.43}$$

this is easily proven by the identity: $(1 - \frac{\varepsilon}{2})^2 > 1 - \varepsilon$.

So, $x_{A_j} \leq 1 - \varepsilon/2 \Rightarrow \frac{1}{1-x_{A_j}} \leq \frac{2}{\varepsilon}$. Also, the expected maximum size of $|\tau|$, which is $O(\frac{1}{\varepsilon}log(\sum_{A_i \in \mathcal{A}} \frac{x_{A_i}}{1-x_{A_i}}))$, can be rewritten as $O(\dfrac{1}{\varepsilon}log(T \cdot \max\limits_i \dfrac{1}{1-x_{A_i}}))$ and simplified to $M = O(\frac{1}{\varepsilon}log(\frac{2T}{\varepsilon}))$.

Now, the construction of the witness trees implies that none of the n variables P_i can be resampled more than M times (otherwise not only the size, but the depth would also be grater than M). So, the expected running time of the MT algorithm is at most $O(\frac{n}{\varepsilon}log(\frac{T}{\varepsilon}))$.

■

The above theorem shows that the number of resamplings needed by the MT algorithm is polynomial in n (the number of variables in $\mathcal{P}$) even if the number of events m is exponential in n. However, the running time is not yet guarranteed to be polynomial in n, because before we resample we have to find a violated event. Of course, we cannot do this ,in general , in polynomial time if the number of events is exponential. The definitions and theorems following, will help us overcome this obstacle.

Definition 3.3.2 ***(Efficient verifiability):*** *A set A of events that are determined by variables in $\mathcal{P}$ is efficiently verifiable if, given an arbitrary assignment to $\mathcal{P}$, we can efficiently (in polynomial time) find an event $A_i \in \mathcal{A}$ that holds or detect that there is no such event.*

In cases where the set $\mathcal{A}$ of events is not efficiently verifiable itself, we will need the following definition:

Definition 3.3.3 *A set $\mathcal{A}' \subseteq \mathcal{A}$ will be called a core subset of $\mathcal{A}$ if it has the additional property of being efficiently verifiable.*

At this point, we will need the following theorem from [10]:

Theorem 3.3.4 *Suppose there is an assignment of reals $x : \mathcal{A} \to (0,1)$ such that (3.1) holds. Let B be any event that is determined by $\mathcal{P}$. Then, the probability that B was true at least once during the execution of the MT algorithm on the events in $\mathcal{A}$, is at most $Pr[B] \prod\limits_{C \in \Gamma(B)} (1-x_C)^{-1}$. In particular, the probability of B being true in the output distribution of MT obeys this upper-bound.*

Proof:

As shown in [17], if B be was true at least once during the execution, then there must be a witness tree with root node B and non-root nodes $V'(\tau) \subseteq \mathcal{A} - \{B\}$. So, we can bound the expected number of such trees and then apply Markov's inequality.

Let τ be a fixed proper witness tree with its root vertex labeled B. Also for $A \in \Gamma(B)$, let τ_A be the subtree starting from A if it is chosen as a child of B during the branching process. Following the proof of lemma 3.1.3 and keeping in mind that the products are only over non-root nodes, we get that the probability that this particular tree is produced by the Galton-Watson branching process defined in section 3.1.2, is exactly :

$$\begin{aligned} Pr[\tau|B \ \ is \ \ the \ \ root] &= \prod_{A\in\Gamma_T(B)} x(A)\cdot Pr[\tau_A] \prod_{A\in\Gamma_G(B)\setminus\Gamma_T(B)} (1-x_A) && (3.44)\\ &= \prod_{A\in\Gamma_G(B)} (1-x(A)) \prod_{u\in V'(\tau)} x'[u] && (3.45)\end{aligned}$$

In a similar way as in equations (3.8) - (3.13), we can deduce that

$$\begin{aligned} E[\sharp witness\ trees\ with\ root\ B] &\le Pr[B]\sum_{\tau\in T_B}\prod_{v\in V(\tau)\setminus\{B\}} Pr[[v]] && (3.46)\\ &\le Pr[B]\sum_{\tau\in T_B}\prod_{v\in V(\tau)\setminus\{B\}} x'([u]) && (3.47)\\ &= Pr[B]\prod_{A_i\in\Gamma(B)}(1-x(A_i))^{-1}\sum_{\tau\in T_B} p_\tau && (3.48)\\ &\le Pr[B]\prod_{A_i\in\Gamma(B)}(1-x(A_i))^{-1} && (3.49)\end{aligned}$$

So, by Markov's inequality:

$$Pr[\sharp witness\ trees\ with\ root\ B > 1] \le E[\sharp witness\ trees\ with\ root\ B]$$

we get the desired upper bound.

■

The following theorem examines the performance of a modified Monte Carlo version of the MT algoritm (which is to run the MT algorithm only on the core subset) and gives bounds on the failure probability.

Theorem 3.3.5 *Let $\mathcal{A}' \subseteq \mathcal{A}$ be an efficiently verifiable core subset of $\mathcal{A}$. If there is an $\varepsilon \in [0,1)$ and an assignment of reals $x : \mathcal{A} \to (0,1)$ such that:*

$$\forall A_i \in \mathcal{A} : Pr[A_i] \le (1-\varepsilon)x(A_i)\prod_{B\in\Gamma(A_i)}(1-x(B)) \tag{3.50}$$

Then the modified MT-algorithm can be efficiently implemented with an expected number of resamplings according to theorem 3.3.1. The algorithm furthermore outputs a good assignment with probability at least $1 - \sum_{A_i\in\mathcal{A}-\mathcal{A}'} x_{A_i}$.

Proof:
The expected number of resamplings is given by theorem 3.3.1 , if we apply it on $\mathcal{A}'$. So, the only thing we have to care about, is the failure probability of the modified MT algorithm. The inequality (3.50) gives:

$$x(A_i) \ge Pr[A_i]\prod_{B\in\Gamma(A_i)}(1-x(B))^{-1} \tag{3.51}$$

Using the above inequality together with theorem 3.3.4 the probability that a non-core event A_i is violated by the output assignment of MT algorithm, is at most $x(A_i)$ (the core events are surely not violated). So,by a simple union bound, the algorithm fails with probability at most $\sum_{A_i \in \mathcal{A}-\mathcal{A}'} x(A_i)$ as desired.

■

In order for the above theorem to be applicable, there are a couple of more things to consider. Firstly, we must be able to find the "core subset", and secondly, the sum of the x-values in the non-core events should be strictly less than 1. Otherwise, the theorem is obviously useless. The following theorem deals with the aforementioned issues, given some slightly stricter conditions.

Theorem 3.3.6 *Suppose $log(1/\delta) \leq poly(n)$. Suppose further that there is a fixed constant $\varepsilon \in (0,1)$ and an assignment of reals $x : \mathcal{A} \to (0, 1-\varepsilon)$ such that:*

$$\forall A_i \in \mathcal{A} : Pr[A_i]^{1-\varepsilon} \leq x(A_i) \prod_{B \in \Gamma(A_i)} (1 - x(B)) \tag{3.52}$$

Then for every $p \geq \frac{1}{poly(n)}$ the set $\mathcal{A}' = \{A_i \in \mathcal{A} : Pr[A_i] \geq p\}$ has size at most poly(n), and is thus essentially always an efficiently verifiable core subset of $\mathcal{A}$. If this is the case, then there is a Monte Carlo algorithm that terminates after $O(n \cdot logn)$ resamplings and returns a good assignment with probability at least $1 - n^{-c}$, where $c > 0$ is any desired constant.

Proof:

Since by definition : $A_i \in \mathcal{A}' \Rightarrow x(A_i) \geq p$ and

$$\sum_{B \in \mathcal{A}'} x(B) \leq \sum_{B \in \mathcal{A}} x(B) \leq O(nlog(1/\delta))$$

we deduce that : $|\mathcal{A}'| \leq O(\frac{nlog(1/\delta)}{p}) = poly(n)$, so $\mathcal{A}'$ is efficiently verifiable.

For $\varepsilon > 0$ sufficiently small, as we have agued before, we have :

$$\forall A_i \in \mathcal{A} : Pr[A_i] \leq x(A_i) \leq 1 - \varepsilon$$

So, equation (3.52) now becomes:

$$Pr[A_i] \leq Pr[A_i]^{\varepsilon} x(A_i) \prod_{B \in \Gamma(A_i)} (1 - x(B)) \tag{3.53}$$

$$< (1-\varepsilon)^{\varepsilon} x(A_i) \prod_{B \in \Gamma(A_i)} (1 - x(B)) \tag{3.54}$$

One can easily verify (by applying de L'Hospital rule twice) that :

$$\lim_{\varepsilon \to 0} \frac{1 - (1-\varepsilon)^{\varepsilon}}{\varepsilon^2} = 1$$

So, we can rewrite (3.54) as follows:

$$Pr[A_i] < (1 - \Theta(\varepsilon^2)) x(A_i) \prod_{B \in \Gamma(A_i)} (1 - x(B)) \tag{3.55}$$

and the apply theorem 3.3.1. So, the algorithm terminates with high probability after $O(nlogn)$ resamplings (assuming ε is a small positive constant).

It remains to prove the bound on the failure probability:

$$\sum_{A_i \in \mathcal{A}-\mathcal{A}'} x(A_i) \leq \sum_{A_i \in \mathcal{A}} x(A_i) = poly(n)$$

Particularly for the non-core events the following inequality holds:

$$\forall A_i \in \mathcal{A} - \mathcal{A}' : Pr[A_i] \leq (1-\varepsilon) x^*(A_i) \prod_{B \in \Gamma(A_i)} (1 - x(B)) \tag{3.56}$$

where $x^*(A_i) = p^\varepsilon \cdot x(A_i)$. So, theorem 3.3.1 can be applied using the x^*-values so

$$\sum_{A_i \in \mathcal{A}-\mathcal{A}'} x^*(A_i) = O(p^\varepsilon \cdot poly(n)) = O(p^\varepsilon \cdot n^k)$$

and if we choose $p = n^{-\frac{1}{\varepsilon}(k+c)} = \frac{1}{poly(n)}$, then by theorem 3.3.5 the failure probability is at most n^{-c} on non-core events, while the core is always avoided.

■

3.4 The lopsided version of LLL

3.4.1 Existential version

An improved version of the existential LLL was introduced by Erdös and Spencer in [6]. The improvement comes from the fact that for every possible dependency graph G, we can contruct a "sparser" dependency graph G' on the same set of events such that: $(i,j) \in G' \Rightarrow (i,j) \in G$ but the converse is not nessesarily true. So, this makes the conditions in equation (1.4) weaker. We will call G' the *lopsidependency graph* of the set of events $\{A_i\}$

Definition 3.4.1 *Lopsidependency graph*
Let $A_1, A_2, ..., A_n$, be events in a probability space, G a graph on the indices We say G is a lopsidependency graph (for the events) if

$$Pr[A_i | \bigwedge_S \overline{A_j}] \leq Pr[A_i] \tag{3.57}$$

for all i, S with $i \notin S$ and no $j \in S$ adjacent to i.

Remark: In the original version, the inequality (3.57) becomes an equality due to mutual independence. So, the lopsidependency graph is clearly "sparser" than the dependency graph.

Using the above definition of the dependency graph, we can prove the following:

Theorem 3.4.2 *Lopsided General Local Lemma:*
Let $\{A_C\}_{C \in I}$ be a finite set of events in some probability space. Let $\Gamma(C)$ be a subset of I for each $C \in I$ such that for every subset $J \subseteq I \setminus (\Gamma(C) \cup \{C\})$ we have

$$Pr[A_C | \bigwedge_{D \in J} \neg A_D] \leq Pr[A_C]$$

Suppose there are real numbers $0 < x_C < 1$ *for* $C \in I$ *such that for every* $C \in I$ *we have*

$$Pr[A_C] \leq x_C \prod_{D \in \Gamma(C)} (1 - x_D)$$

Then

$$Pr[\bigwedge_{C \in I} A_C] > 0$$

The corresponding symmetric version can be stated as follows:

Theorem 3.4.3 *Lopsided Symmetric Lovász Local Lemma*
Let $A_1, A_2, ..., A_n$*, be events with lopsidependency graph* G *and suppose all the events have probability at most* p *and that each* $i \in G$ *has degree at most* d*. Assume* $4dp \leq 1$*. Then*

$$Pr[\bigwedge_{i=1}^{n} \overline{A_i}] > 0 \tag{3.58}$$

3.4.2 Constructive version

Moser and Tardos [17] tried to adapt this notion to their setting (the case where each event is determined by a subset of a set $\mathcal{P}$ of mutually independent events).

Definition 3.4.4 *We say that two events* $A, B \in \mathcal{A}$ *are* ***lopsidependent*** *if there exist two evaluations* f *and* g *of the variables in* $\mathcal{P}$ *that differ only on variables in* $vbl(A) \cap vbl(B)$ *such that* f *violates* A *and* g *violates* B *but either* f *does not violate* B *or* g *does not violate* A*.*

Definition 3.4.5 *The* ***variable lopsidependency graph*** *is the graph on the vertex set* A*, where lopsidependent events are connected by an edge. We write* $\Gamma(A) = \Gamma'_{\mathcal{A}}(A)$ *for the neighborhood of an event* A *in this graph.*

Clearly, if $vbl(A)$ is disjoint from $vbl(B)$, then A and B cannot be lopsidependent, so we have $\Gamma'(A) \subseteq \Gamma(A)$, which makes the conditions (3.1) weaker.

Definition 3.4.6 *We call an event* $A \in \mathcal{A}$ ***elementary****, if there is a single evaluation of the variables in* $vbl(A)$ *violating* A*.*

Note that, the elementary events A and B are lopsidependent if and only if they are mutually exclusive. Indeed, if the unique assignment in the set $vbl(A)$ that violates A, also ensures B is not violated, then the unique assignment in $vbl(B)$ that violates B has to assign different values to the variables in $vbl(A) \cap vbl(B)$. The converse is true by the definition of elementary events and lopsided dependence.

Example: Given a CNF formula with m clauses, we can define the set of events $A_1, A_2, ..., A_m$ as follows:

$$A_i : clause\ i\ is\ not\ satisfied$$

The above events are clearly *elementary*, since they are determined by the set of literals they contain, and they are true if and only if each of them is false.

Additionaly, two (elementary) events A_i, A_j are mutually exclusive (or equivalently lopsidependent) if and only if the corresponding clauses have at least one common variable appearing in complementary literals. Let x_l be one of those variables. Indeed, if for example x_l appears in the clause A_i complemented, then A_i is violated only if x_l is True, which satisfies A_j. Conversely, if the events are mutually exclusive, the unique assignment in $vbl(A_i)$ and the unique assignment in $vbl(A_j)$ differ in $vbl(A) \cap vbl(B)$. So, since the events are elementary, the two clauses have at least one common variable appearing in complementary literals.

3.4.3 Comparing definition 3.4.1 and definition 3.4.5

By looking more carefully into the definitions of lopsidependency graph (definition 3.4.1) and variable lopsidependency graph (definition 3.4.5), we notice that latter uniquely determines a graph G, whereas definition 3.4.1 doesn't.

Example:
Suppose X and Y are two independent tosses of a fair coin, where we designate 1 for heads and 0 for tails. Let the third random variable Z be equal to 1 if both coin tosses resulted in "heads" or both resulted in "tails". Then jointly the triple (X, Y, Z) has the following probability distribution:

$$(X,Y,Z) = \begin{cases} (0,0,1) & \text{with probability } 1/4 \\ (0,1,0) & \text{with probability } 1/4 \\ (1,0,0) & \text{with probability } 1/4 \\ (1,1,1) & \text{with probability } 1/4 \end{cases} \tag{3.59}$$

It is easy to verify that:

1. $Pr[Z|X] = Pr[Z] \leq Pr[Z] \Leftrightarrow$ X and Z are independent
2. $Pr[Z|Y] = Pr[Z] \leq Pr[Z] \Leftrightarrow$ Y and Z are independent
3. $Pr[Z|X, Y] = 1 > Pr[Z]$
   Similarly,
4. $Pr[X|Z] = Pr[X] \leq Pr[X]$
5. $Pr[X|Y] = Pr[X] \leq Pr[X] \Leftrightarrow$ X and Y are independent
6. $Pr[X|Z, Y] = 1 > Pr[X]$
   And similarly,
7. $Pr[Y|X] = Pr[Y] \leq Pr[Y]$
8. $Pr[Y|Z] = Pr[Y] \leq Pr[Y]$
9. $Pr[Y|X, Z] = 1 > Pr[Y]$

So, we can easily verify that there are three minimal dependency graphs shown below (whereas according to definition 3.4.5 we get the middle graph):

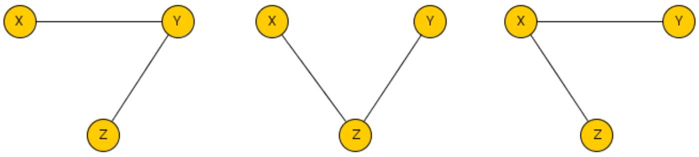

Figure 3.6: The three possible dependency graphs

However, the following theorem holds:

Theorem 3.4.7 *Let $\mathcal{P}$ be a finite set of mutually independent random variables in a probability space. Let $\mathcal{A}$ be a finite set of elementary events determined by these variables. The (unique) variable lopsidependency graph (definition 3.4.5) is a subgraph of every possible lopsidependency graph (definition 3.4.1) on $\mathcal{A}$.*

Proof:

Since the two graphs are on the same set of vertices, it suffices to prove that: If the events $A_i, A_j \in \mathcal{A}$ are lopsidependent, they are also connected by an edge in every possible lopsidependency graph on $\mathcal{A}$.

We argued before that two elementary events are lopsidependent if and only if the are mutually exclusive.

So, we get:

$$A_i \Rightarrow \neg A_j$$

$$A_j \Rightarrow \neg A_i$$

The conditional probability that A_i is violated given that A_j isn't, can be written as follows:

$$Pr[A_i|\neg A_j] = \frac{Pr[A_i \cap \neg A_j]}{Pr[\neg A_j]} = \frac{Pr[A_i]}{Pr[\neg A_j]} > Pr[A_i] \tag{3.60}$$

So, if they were not adjacent in some lopsidependency graph, then by (3.57) we get a contradiction.

Example:

Let ϕ be a k-SAT formula, in which every variable is TRUE of FALSE with probability 1/2. We define the events:

$$A_i : the\ i-th\ clause\ of\ \phi\ is\ not\ satisfied\ , 1 \leq i \leq m$$

As mentioned before, all events are elementary and also every pair of them which has at least one common variable appearing in complementary literals, is lopsidependent.

So, by (3.60) we get:

$$Pr[A_i|\neg A_j] = \frac{Pr[A_i]}{Pr[\neg A_j]} = \frac{2^{-k}}{1-2^{-k}} > 2^{-k} \tag{3.61}$$

In case, A_i, A_j have no variables that appear in complementary literals, which makes them independent in the lopsided sence (not adjacent in the variable lopsidependency graph), let $0 \leq s \leq k$ be the number of their common literals.

Let's see how these two events depend on each other with regard to s. If A_i holds, then s out of k literals of the $i-th$ clause are already FALSE.

So,

$$Pr[A_i|A_j] = 2^{-(k-s)} \tag{3.62}$$

which denotes the probability that the remaining $k-s$ literals are also FALSE.

Using the law of total probability, we have:

$$Pr[A_i|A_j]Pr[A_j] + Pr[A_i|\neg A_j]Pr[\neg A_j] = Pr[A_i] \tag{3.63}$$

or equivalently,

$$Pr[A_i|\neg A_j] = \frac{Pr[A_i] - Pr[A_i|A_j]Pr[A_j]}{Pr[\neg A_j]} = \frac{2^{-k} - 2^{-(k-s)}2^{-k}}{1-2^{-k}} = \frac{2^{-k}(1-2^{-(k-s)})}{1-2^{-k}} \tag{3.64}$$

We can verify that if s=0 (the clauses consist of completely different variables, so the events are independent) $Pr[A_i|\neg A_j] = 2^{-k} = Pr[A_i]$ as expected. Also, if s=k (the clauses are identical) $Pr[A_i|\neg A_j] = 0$ as expected. Note also that, the conditional probability in equation (3.64) is a strictly decreasing function of s.

Theorem 3.4.8 *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 an assignment of reals $x : \mathcal{A} \rightarrow (0,1)$ such that:*

$$\forall A_i \in \mathcal{A} : Pr[A_i] \leq x(A) \prod_{B \in {\Gamma'}_{\mathcal{A}}(A_i)} (1 - x(B)) \tag{3.65}$$

then there exists an assignment of values to the variables $\mathcal{P}$ not violating any of the events in $\mathcal{A}$. Moreover,the randomized algorithm resamples an event $A_i \in \mathcal{A}$ at most an expected $\frac{x(A_i)}{1-x(A_i)}$ times before it finds such an evaluation. Thus the expected total number of resampling steps is at most $\sum_{A_i \in \mathcal{A}} \frac{x(A_i)}{1-x(A_i)}$.

Chapter 4

Improvements on the analysis of MT algorithm

4.1 Improved analysis of the sequential algorithm

In a very recent work, Wesley Pegden [18] managed to fit the improvement of Bissacot,et al.[5] into the algorithmic framework of Moser and Tardos[17] and thus give a constructive proof of the "Improved Lovász local lemma" (section 3).

The main idea is the observation that the set of witness trees that can actually occur in the random log of the MT algorithm is much smaller than the set of possible outputs of the Galton - Watson process. In particular, we can modify the Galton - Watson process to (randomly) choose witness trees from a proper subset of "proper" witness trees, which we will call "strongly proper" witness trees.

Definition 4.1.1 *If a witness tree has the property that any two children of a common vertex have labels which are nonadjacent in the dependency graph, we will call it a strongly proper witness tree.*

Notice that, every witness tree that occurs in the log is necessarily strongly proper. So, our random branching process does not need to produce witness trees outside of this set.

We are going to prove the following theorem:

Theorem 4.1.2 *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 with a dependency graph G determined by these variables. If there exists an assignment of reals $\mu : \mathcal{A} \to (0, +\infty)$ such that*

$$p_i \leq R_i^* = \frac{\mu(A_i)}{\varphi_i^*(\mu)} \tag{4.1}$$

where

$$\varphi_i^*(\mu) = \sum_{R \subseteq \Gamma_G^+(A_i), R \ \ indep} \prod_{u \in R} \mu(u) \tag{4.2}$$

then there exists an assignment of values to the variables P_i not violating any of the events in $\mathcal{A}$. Moreover the MT algorithm resamples an event $A_i \in \mathcal{A}$ at most an expected $\mu(A_i)$ times before it finds such an evaluation. Thus, the expected total number of resampling steps is at most $\sum_{A_i \in \mathcal{A}} \mu(A_i)$.

Modifying the branching process:
To prove the theorem, we will need an improved branching lemma, and to get to that we need to redefine the branching process, so that it only produces strongly proper witness trees. So, the modified process will be exactly the same as the original (we will use $x(A_i) = \dfrac{\mu(A_i)}{1+\mu(A_i)}$ instead of $x(A_i)$), but when we finish the choice of one node's children we check if they form an independent set,otherwise we choose another random set of children until they form an independent set. By doing this, we ensure that no non-strongly proper witness tree will be produced, and also any set of children of the same node will correspond to an independent set in the dependency graph.

Notice that the modification does not act in favour of any independent set, which means that the likelihood of the choice of each independent set I_u is weighted according to:

$$w(I_u) = \prod_{v \in I_u} x_{A_v} \prod_{v \in \Gamma_G^+(u) - I_u} (1 - x_{A_u}) \;\; (1) \tag{4.3}$$

which is exactly the probability I_u is chosen from the set of vertices:$\Gamma_G^+(u)$ by the unmodified process.

So,

$$Pr[I_u \;\; is \;\; chosen] = \frac{w(I_u)}{\sum\limits_{I \subset \Gamma_G^+(u), I \;\; indep} w(I)} \tag{4.4}$$

Lemma 4.1.3 ***[Improved Branching lemma]*** *For any strongly proper witness tree T with root labelled A_r, the probability p'_T that the modified branching process described above produces exactly the tree T is*

$$\mu_{A_r}^{-1} \prod_{u \in T} \frac{\mu([u])}{\sum\limits_{I \subset \Gamma_G^+(u), I \;\; indep} \prod\limits_{A \in I} \mu_A} \tag{4.5}$$

Proof:
For each node $u \in T$, let I_u be the set of it's children in T (which is an independent set in G). According to the process, we choose the children of every node independently.
So,

$$p'_T = \prod_{u \in T} Pr[I_u \;\; is \;\; chosen \;\; from \;\; \Gamma_G^+(u)] \tag{4.6}$$

$$p'_T = \prod_{u \in T} \frac{Pr[I_u \;\; is \;\; chosen]}{Pr[I_u \;\; is \;\; indep]} \tag{4.7}$$

$$= \prod_{u \in T} \frac{\prod\limits_{v \in I_u} x_{A_v} \prod\limits_{B \in \Gamma_G^+(u) - I_u} (1 - x_B)}{\sum\limits_{I \subset \Gamma_G^+(u), I \;\; indep} \prod\limits_{A \in I_u} x_A \prod\limits_{B \in \Gamma_G^+(u) - I_u} (1 - x_B)} \tag{4.8}$$

by dividing the numerator and denominator by $\prod_{B\in\Gamma_G^+(u)}(1-x_B)$, we get:

$$p'_T = \prod_{u\in T} \frac{\displaystyle\prod_{v\in I_u} \frac{x_{A_v}}{1-x_{A_v}}}{\displaystyle\sum_{I\subset\Gamma_G^+(u),I\ \ indep} \prod_{A\in I_u} \frac{x_A}{1-x_A}} \tag{4.9}$$

$$= \frac{\displaystyle\prod_{u\in T-\{r\}} \mu(u)}{\displaystyle\prod_{u\in T} \sum_{I\subset\Gamma_G^+(u),I\ \ indep} \prod_{A\in I_u} \mu(A)} \tag{4.10}$$

where r is the root of T and $\mu(v) = \dfrac{x_{A_v}}{1-x_{A_v}}$, or equivalently:

$$p'_T = \frac{1}{\mu(r)} \prod_{u\in T} \frac{\mu(u)}{\displaystyle\sum_{I\subset\Gamma_G^+(u),I \text{ indep}} \prod_{A\in I_u} \mu(A)} \tag{4.11}$$

□

Now, we would like to calculate the expected number of resamplings for each event. Let $T^S_{A_i}$ be the set of strongly proper witness trees with root A_i.
By lemma 3.1.2 we have:

$$E[N_{A_i}] = \sum_{\tau\in T^S_{A_i}} Pr[\tau\ \ occurs\ \ in\ \ C] \tag{4.12}$$

$$\leq \sum_{\tau\in T^S_{A_i}} \prod_{v\in V(\tau)} Pr[[v]] \tag{4.13}$$

$$\leq \sum_{\tau\in T^S_{A_i}} \prod_{v\in V(\tau)} \frac{\mu(A_i)}{\displaystyle\sum_{R\subseteq\Gamma_G^+(A_i)\text{R indep}} \prod_{u\in R} \mu(u)} \tag{4.14}$$

$$= \mu_{A_i} \sum_{\tau\in T^S_{A_i}} p'_T \leq \mu_{A_i} \tag{4.15}$$

since at most one strongly proper witness tree from $T^S_{A_i}$ is generated by the branching process each time (or none if it doesn't terminate).

So, the expected number of resamplings made by the MT algorithm is again $\sum_{A\in\mathcal{A}} \mu_A$.

■

4.2 Improved analysis of the parallel algorithm

We are now going to examine the parallel case:

Theorem 4.2.1 *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 with a dependency graph G determined by these variables. If there exists an assignment of reals $\mu : \mathcal{A} \to (0, +\infty)$ such that*

$$p_i \leq (1-\varepsilon)\frac{\mu(A_i)}{\varphi_i^*(\mu)} \tag{4.16}$$

where

$$\varphi_i^*(\mu) = \sum_{R \subseteq \Gamma_G^+(A_i), R \ \ indep} \prod_{u \in R} \mu(u) \tag{4.17}$$

then the parallel version of our algorithm takes an expected $O(\frac{1}{\varepsilon} log \sum_{A_i \in \mathcal{A}} \mu(A_i))$ steps before it finds an evaluation violating no event in $\mathcal{A}$.

Proof:

It is easy to see that lemma 3.2.2 still holds. So, the equations (3.22)-(3.24) become:

$$\begin{aligned} Q(k) &\leq \sum_{A_i \in \mathcal{A}} \sum_{\tau \in T_{A_i}^S(k)} Pr[\tau \ \ occurs \ \ in \ \ the \ \ log \ \ C] && (4.18)\\ &\leq \sum_{A_i \in \mathcal{A}} \sum_{\tau \in T_{A_i}^S(k)} \prod_{v \in V(\tau)} Pr[[v]] && (4.19)\\ &\leq (1-\varepsilon)^k \sum_{A_i \in \mathcal{A}} \sum_{\tau \in T_{A_i}^S(k)} \prod_{v \in V(\tau)} \frac{\mu(A_i)}{\sum_{R \subseteq \Gamma_G^+(A_i), R \ \ indep} \prod_{u \in R} \mu(u)} && (4.20) \end{aligned}$$

where the last inequality follows from the assumption in Theorem 4.2.1.
So,

$$\begin{aligned} Q(k) &\leq (1-\varepsilon)^k \sum_{A_i \in \mathcal{A}} \sum_{\tau \in T_{A_i}^S(k)} \prod_{v \in V(\tau)} \frac{\mu(A_i)}{\sum_{R \subseteq \Gamma_G^+(A_i), R \ \ indep} \prod_{u \in R} \mu(u)} && (4.21)\\ &= (1-\varepsilon)^k \sum_{A_i \in \mathcal{A}} \mu(A_i) \sum_{\tau \in T_{A_i}^S(k)} p_\tau && (4.22)\\ &\leq (1-\varepsilon)^k \sum_{A_i \in \mathcal{A}} \mu(A_i) && (4.23) \end{aligned}$$

If we set: $\underline{M = \sum_{A_i \in \mathcal{A}} \mu(A_i)}$, and T be the number of steps, we have:

$$Pr[T \geq k] \leq M \cdot (1-\varepsilon)^k \tag{4.24}$$

So, by lemma 3.2.3 ,

$$E[T] = O(\frac{1}{\varepsilon} log(M)) \tag{4.25}$$

■

4.3 Superpolynomial number of events

The next theorem is an analogue of theorem 3.3.1 which also exploits the improvement first introduced in [5].

Theorem 4.3.1 *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 with a dependency graph G determined by these variables. If there exists an assignment of reals $\mu : \mathcal{A} \to (0, +\infty)$ such that*

$$p_i \leq (1-\varepsilon)\frac{\mu(A_i)}{\varphi_i^*(\mu)} \tag{4.26}$$

where

$$\varphi_i^*(\mu) = \sum_{R \subseteq \Gamma_G^+(A_i), R \ \ indep} \prod_{u \in R} \mu(u) \tag{4.27}$$

then with δ denoting $\min\limits_{A_i \in \mathcal{A}} \dfrac{\mu(A_i)}{1+\mu(A_i)} \prod\limits_{B \in \Gamma(A_i)} \dfrac{1}{1+\mu(A_i)}$, we have

$$T := \sum_{A_i \in \mathcal{A}} \frac{\mu(A_i)}{1+\mu(A_i)} \leq nlog(1/\delta) \tag{4.28}$$

Furthermore:

1. *If $\varepsilon = 0$, then the expected number of resamplings done by the MT algorithm is at most $v_1 = T(1 + \max\limits_{A_i \in \mathcal{A}} \mu(A_i))$, and for any parameter $\lambda \geq 1$, the MT algorithm terminates within λv_1 resamplings with probability at least $1 - 1/\lambda$.*

2. *If $\varepsilon > 0$, then the expected number of resamplings done by the MT algorithm is at most $v_2 = O(\frac{n}{\varepsilon} log \frac{T}{\varepsilon})$, and for any parameter $\lambda \geq 1$, the MT algorithm terminates within λv_2 resamplings with probability $1 - e^{-\lambda}$.*

Proof:

The first part of the theorem follows immediately from theorem 4.1.2, because we have proved that if $\varepsilon = 0$, then the expected number of resamplings cannot be more than $\sum_{A_i \in \mathcal{A}} \mu(A_i) \leq T \cdot (1 + \max_{A_i \in \mathcal{A}} \mu(A_i))$. Also, the rest of the first part is proven by a simple application of Markov's inequality.

The bound on T is easily proven by making the substitution: $\mu(A_i) = \frac{x(A_i)}{1-x(A_i)}$ and then following the proof of theorem 3.3.1.

Finally, the proof of the 2nd part can be done in exactly the same way as in theorem 3.3.1 , because from equations (4.18)-(4.20) we have

$$\sum_{A_i \in \mathcal{A}} \sum_{\tau \in T_{A_i}^S(k)} Pr[\tau \ \ occurs \ \ in \ \ C] \leq (1-\varepsilon)^k \sum_{A_i \in \mathcal{A}} \sum_{\tau \in T_{A_i}^S(k)} \prod_{v \in V(\tau)} \frac{\mu(A_i)}{\sum\limits_{R \subseteq \Gamma_G^+(A_i), R \ \ indep} \prod\limits_{u \in R} \mu(u)} \tag{4.29}$$

istead of (3.38) , so we finally get

$$Pr[\max_{\tau\ occuring\ in\ C}|\tau| \geq k] \leq (1-\varepsilon)^k \sum_{A_i\in\mathcal{A}} \sum_{\tau\in T^S_{A_i}(k)} \prod_{v\in V(\tau)} \frac{\mu(A_i)}{\sum\limits_{R\subseteq\Gamma^+_G(A_i),R\ \ indep} \prod\limits_{u\in R}\mu(u)} \tag{4.30}$$

instead of (3.41). By applying lemma 3.2.3 and using the argument that the x-values can be bounded away from 1, we can show that the expected number of steps that the algorithm performs is $O(\frac{n}{\varepsilon}log(\frac{T}{\varepsilon}))$.

■

In order to exploit "effcient verifiability" the way we did in section 3.3, we will need the following theorem, an analogue of theorem 3.3.4.

Theorem 4.3.2 *Suppose there is an assignment of reals $\mu : \mathcal{A} \to (0, +\infty)$ such that (4.1) and (4.2) hold. Let B be any event that is determined by $\mathcal{P}$. Then, the probability that B was true at least once during the execution of the MT algorithm on the events in $\mathcal{A}$, is at most $Pr[B] \sum\limits_{R\subseteq\Gamma(B),R\ \ indep} \prod\limits_{u\in R}\mu([u])$. In particular, the probability of B being true in the output distribution of MT obeys this upper-bound.*

Proof:
Similarly to the proof of theorem 3.3.4, we calculate the probability that the witness tree τ is produced by the modified branching process conditional that B is the root.

Consider the neighbours of node B in G. Then the desired probability is (using lemma 4.1.3):

$$Pr[\tau|B\ \ is\ \ the\ \ root] =$$

$$= Pr[\Gamma_T(B)\ \ is\ \ chosen] \cdot \prod_{A_i\in\Gamma_T(B)} \mu_{A_i}^{-1} \prod_{u\in T} \frac{\mu(u)}{\sum\limits_{I\subset\Gamma^+_G(u),I\ \ indep} \prod\limits_{A\in I}\mu_A} \tag{4.31}$$

Note that, each child of B is a root of a subtree generated by an independent random process. By comparing equations (4.6) and (4.9) we can see that :

$$Pr[\Gamma_T(B)\ \ is\ \ chosen] = \frac{\prod\limits_{v\in\Gamma_T(B)} \frac{x_{A_v}}{1-x_{A_v}}}{\sum\limits_{I\subset\Gamma^+_G(B),I\ \ indep} \prod\limits_{A\in\Gamma_T(B)} \frac{x_A}{1-x_A}} \tag{4.32}$$

$$= \frac{\prod\limits_{v\in\Gamma_T(B)} \mu([v])}{\sum\limits_{I\subset\Gamma^+_G(B),I\ \ indep} \prod\limits_{A\in\Gamma_T(B)} \mu(A)} \tag{4.33}$$

So,

$$Pr[\tau|B\ \ is\ \ the\ \ root] =$$

$$= \frac{\prod\limits_{v \in \Gamma_T(B)} \mu([v])}{\sum\limits_{I \subset \Gamma_G^+(B), I \ indep} \prod\limits_{A \in \Gamma_T(B)} \mu(A)} \cdot \prod_{A_i \in \Gamma_T(B)} \mu_{A_i}^{-1} \prod_{u \in T} \frac{\mu(u)}{\sum\limits_{I \subset \Gamma_G^+(u), I \ indep} \prod\limits_{A \in I} \mu_A} \tag{4.34}$$

$$= \frac{1}{\sum\limits_{I \subset \Gamma_G^+(B), I \ indep} \prod\limits_{A \in \Gamma_T(B)} \mu(A)} \cdot \prod_{u \in T} \frac{\mu(u)}{\sum\limits_{I \subset \Gamma_G^+(u), I \ indep} \prod\limits_{A \in I} \mu_A} \tag{4.35}$$

If we work in a similar way as in equations (4.12)-(4.15), we get that the expected number of such witness trees is upper bounded by:

$$Pr[B] \sum_{R \subseteq \Gamma(B), R \ indep} \prod_{u \in R} \mu([u]) \tag{4.36}$$

So, by simply applying Markov's inequality, we get the desired upper bound.

■

In the following two theorems, we show that theorems 3.3.5 and 3.3.6 can be extended to apply to our new setting as well.

Theorem 4.3.3 *Let $\mathcal{A}' \subseteq \mathcal{A}$ be an efficiently verifiable core subset of $\mathcal{A}$. If there is an $\varepsilon \in [0, 1)$ and an assignment of reals $x : \mathcal{A} \to (0, 1)$ such that:*

$$\forall A_i \in \mathcal{A}: \quad Pr[A_i] \leq (1 - \varepsilon) \frac{\mu(A_i)}{1 + \mu(A_i)} \frac{1}{\sum\limits_{R \subseteq \Gamma_G^+(A_i), R \ indep} \prod\limits_{u \in R} \mu(u)} \tag{4.37}$$

Then the modified MT-algorithm can be efficiently implemented with an expected number of resamplings according to theorem 4.3.1. The algorithm furthermore outputs a good assignment with probability at least $1 - \sum\limits_{A_i \in \mathcal{A} - \mathcal{A}'} \dfrac{\mu(A_i)}{1 - \mu(A_i)}$.

Proof:
The expected number of resamplings is given by theorem 4.3.1 , if we apply it on $\mathcal{A}'$. So, the only thing we have to care about, is the failure probability of the modified MT algorithm. The inequality (4.37) gives:

$$\forall A_i \in \mathcal{A}: \quad Pr[A_i] \leq \frac{\mu(A_i)}{1 + \mu(A_i)} \frac{1}{\sum\limits_{R \subseteq \Gamma_G^+(A_i), R \ indep} \prod\limits_{u \in R} \mu(u)} \tag{4.38}$$

$$\forall A_i \in \mathcal{A}: \quad \frac{\mu(A_i)}{1 + \mu(A_i)} \geq Pr[A_i] \sum_{R \subseteq \Gamma_G^+(A_i), R \ indep} \prod_{u \in R} \mu(u) \tag{4.39}$$

Using the above inequality together with theorem 4.3.2 the probability that a non-core event A_i is violated by the output assignment of MT algorithm, is at most $\frac{\mu(A_i)}{1+\mu(A_i)}$ (the core events are surely not violated). So,by a simple union bound, the algorithm fails with probability at most $\sum_{A_i \in \mathcal{A} - \mathcal{A}'} \frac{\mu(A_i)}{1+\mu(A_i)}$ as desired.

■

Theorem 4.3.4 *Suppose $log(1/\delta) \leq poly(n)$. Suppose further that there is a fixed constant $\varepsilon \in (0,1)$ and an assignment of reals $\mu : \mathcal{A} \to (0, +\infty)$ such that:*

$$\forall A_i \in \mathcal{A} : Pr[A_i]^{1-\varepsilon} \leq \frac{\mu(A_i)}{1+\mu(A_i)} \frac{1}{\sum\limits_{R \subseteq \Gamma(A_i), R \ \ indep} \prod\limits_{B \in R} \mu(A_i)} \tag{4.40}$$

Then for every $p \geq \frac{1}{poly(n)}$ the set $\mathcal{A}' = \{A_i \in \mathcal{A} : PrA_i \geq p\}$ has size at most poly(n), and is thus essentially always an efficiently verifiable core subset of $\mathcal{A}$. If this is the case, then there is a Monte Carlo algorithm that terminates after O(nlogn) resamplings and returns a good assignment with probability at least $1 - n^{-c}$, where $c > 0$ is any desired constant.

Proof:

By carefully following the proof of theorem 3.3.6, we can prove that $|\mathcal{A}'| = poly(n)$ and instead of equation (3.55) we now have:

$$Pr[A_i] < (1 - \Theta(\varepsilon^2)) \frac{\mu(A_i)}{1+\mu(A_i)} \frac{1}{\sum\limits_{R \subseteq \Gamma(A_i), R \ \ indep} \prod\limits_{B \in R} \mu(A_i)} \tag{4.41}$$

Then, theorem 4.3.1 is applicable. From that we get the desired O(nlogn) bound in the number of resamplings.

Also, theorem 4.3.3 is applicable. So, the error probability is upper bounded by:

$$\sum_{A_i \in \mathcal{A} - \mathcal{A}'} \frac{\mu(A_i)}{1+\mu(A_i)} \leq \sum_{A_i \in \mathcal{A}} \frac{\mu(A_i)}{1+\mu(A_i)} = poly(n)$$

Particularly for the non-core events the following inequality holds:

$$\forall A_i \in \mathcal{A} - \mathcal{A}' : Pr[A_i] \leq (1 - \varepsilon) x^*(A_i) \frac{1}{\sum\limits_{R \subseteq \Gamma(A_i), R \ \ indep} \prod\limits_{B \in R} \mu(A_i)} \tag{4.42}$$

where $x^*(A_i) = p^\varepsilon \cdot \frac{\mu(A_i)}{1+\mu(A_i)}$. So, theorem 4.3.1 can be applied using the x^*-values so

$$\sum_{A_i \in \mathcal{A} - \mathcal{A}'} x^*(A_i) = O(p^\varepsilon \cdot poly(n)) = O(p^\varepsilon \cdot n^k)$$

and if we choose $p = n^{-\frac{1}{\varepsilon}(k+c)} = \frac{1}{poly(n)}$, then by theorem 4.3.3 the failure probability is at most n^{-c} on non-core events, while the core is always avoided.

■

Chapter 5

Tightness results for the Lovász Local Lemma

5.1 Introduction

5.1.1 Definitions

As we have already seen, the Lovász local lemma, provides sufficient conditions for a combinatorial object to exist. It is an interesting question whether these conditions are also necessary.

In a recent work, H. Gebauer et'al [10] have partially answered this question, by showing that the local lemma is asymptotically tight for SAT. They construct unsatisfiable k-CNF formulas with distinct literals in each clause, where every variable appears in at most $(\frac{2}{e}+o(1))\frac{2^k}{k}$ clauses. They also use the lopsided local lemma (theorem 3.4.3) to prove that every k-CNF formula where every variable appears in at most $\frac{2}{e}\cdot\frac{2^k}{k+1}-1$ clauses is satisfiable.

More interestingly, it has been proven that the k-SAT problem with bounded number of occurences undergoes a very sharp "complexity phase transition". In order to explain that more formally, we will use the following definitions:

Definition 5.1.1 *A k-CNF formula is called a (k,s)-CNF formula if every variable appears in at most s clauses.*

Definition 5.1.2 *Let $f(k)$ be the largest integer s such that every (k,s)-CNF formula is satisfiable.*

Theorem 5.1.3 *For every $k \geq 3$ the $(k, f(k)+1) - SAT$ is already NP-complete.*

This means that if s=**f(k)** k-SAT is trivial, but if we allow just one more occurence in each variable, it becomes NP-complete. This explains the term: "complexity phase transition".

In order to use the Lovász Local Lemma to give a lower bound for $f(k)$, it is more convenient to introduce a different number as follows:

Definition 5.1.4 *Let **l(k)** be the largest integer number satisfying that whenever all clauses of a k-CNF formula intersect at most $l(k)$ other clauses the formula is satisfiable.*

5.1.2 Lower bounds for number of occurences and intersections

So, we are going to prove the following corollary of the LLL for k-SAT.

Corollary 5.1.5 *Let ϕ be a k-CNF formula, for $k \geq 2$.*
If $l(k) \leq \frac{2^k}{e}-1$, then ϕ is satisfiable.

Proof:

Using the symmetric version of the LLL (equation 1.5), we get:

$$e \cdot p \cdot (l(k)+1) \leq 1 \Rightarrow \phi \; is \; satisfiable$$

Since $p = 2^{-k}$ and using the hypothesis, we get:

$$e \cdot p \cdot (l(k)+1) \leq e \cdot 2^{-k} \cdot \frac{2^k}{e} = 1$$

So, the corollary holds.

We can easily observe that this bound lower bound is asymptotically optimal. The formula all possible k-clauses on k variables is clearly unsatisfiable and contains only 2^k clauses, which are pairwise intersecting. This means that $l(k) < 2^k - 1$.

However, H.Gebauer et'al [10] proved that the factor of $\frac{1}{e}$ cannot be improved, as we will see next.

We can use corollary 5.1.5 to give a lower bound for $f(k)$ as well. Indeed, let ϕ be an unsatisfiable instance where the maximum degree of some clause is $l(k)+1$. We fix a clause C_i in ϕ which intersects with $l(k)+1$ other clauses. Each of it's k variables may cause C_i intersect with some other clause, and using the pidgeonhole principle, it is clear that there must be a variable appearing in at least $\lfloor \frac{l(k)+1}{k} \rfloor + 1$ clauses, which implies that:

$$f(k) \geq \lfloor \frac{l(k)+1}{k} \rfloor \geq \lfloor \frac{2^k}{e \cdot k} \rfloor \tag{5.1}$$

Now, we are going to use the (stronger) lopsided version of the LLL to obtain a factor 2 improvement and then show that the bound is tight up to an asymptotically smaller additive term.

Theorem 5.1.6 *Let ϕ be a k-CNF formula and $f(k)$ be the largest integer s such that every (k,s)-CNF formula is satisfiable. The following lower bound on $f(k)$ holds:*

$$f(k) \geq \lfloor \frac{2^{k+1}}{e(k+1)} \rfloor \tag{5.2}$$

Proof:

Let ϕ be a (k,s)-CNF formula with $s = \lfloor \frac{2^{k+1}}{e(k+1)} \rfloor$. Also, let n_l be the number of occurences of the literal l in ϕ.

Consider the following random experiment:

Each variable x_i is set to:

- *TRUE with probability: $P_{x_i} = \frac{1}{2} + \frac{2n_{\neg x_i} - s}{2sk}$*
- *FALSE with probability $P_{\neg x_i} = \frac{1}{2} - \frac{2n_{\neg x_i} - s}{2sk}$*

Note that,

$$P_{\neg x_i} = \frac{1}{2} - \frac{2n_{\neg x_i} - s}{2sk} \geq \frac{1}{2} + \frac{2n_{\neg x_i} - s}{2sk} \Leftrightarrow \tag{5.3}$$

$$n_{x_i} + n_{\neg x_i} - s \leq 0 \tag{5.4}$$

$$\tag{5.5}$$

which holds by definition.

So, every literal l is satisfied by this assigment with probability at least $\frac{1}{2} + \frac{2n_{\neg l}-s}{2sk}$

For every clause C_i of ϕ, we define it's ***neighbourhood*** $\Gamma(C)$ *in the lopsidependency graph to be:*

$$\Gamma(C) : \{D|\exists l \in D \ s.t : \ \neg l \in C\}$$

From this definition, it is clear that every clause $D \notin \Gamma(C)$ has either no common variables with C or their common variables appear in identical literals. As this is the case, we can use an argument similar to what we have used in section 3.4, and claim that for every $J \subseteq V(G) \setminus (\Gamma(C) \cup \{C_i\})$, knowing that the event: $\bigwedge_{D\in J} \neg A_D$ occurs cannot increase the probability of A_{C_i}, so the conditions for the lopsidependency graph

$$\forall C : Pr[A_C| \bigwedge_{D\in J} \neg A_D] \leq Pr[A_C]$$

are met.

It remains to prove the validity of equation **??** *to be able to use the lemma.*

A clause C of ϕ is violated if and only if all the literals it contains $(l_1, l_2, ..., l_k)$ are FALSE.

So, we get:

$$Pr[A_C] = \prod_{i=1}^{k}(1 - P_{l_i}) \tag{5.6}$$

$$\leq \prod_{i=1}^{k}(\frac{1}{2} - \frac{2n_{\neg l_i} - s}{2sk}) \tag{5.7}$$

$$\leq \frac{1}{2^k}\prod_{i=1}^{k}(1 - \frac{2n_{\neg l_i} - s}{sk}) = \frac{1}{2^k}\prod_{i=1}^{k}(1 + \frac{1}{k} - \frac{2n_{\neg l_i}}{sk}) \tag{5.8}$$

$$\leq \frac{1}{2^k}\prod_{i=1}^{k}(1 + \frac{1}{k} - \frac{2n_{\neg l_i} \cdot e(k+1)}{2^{k+1} \cdot k}) \tag{5.9}$$

$$= \frac{1}{2^k}\prod_{i=1}^{k}(1 + \frac{1}{k} - \frac{n_{\neg l_i} \cdot e}{2^k} - \frac{n_{\neg l_i} \cdot e}{2^k \cdot k}) = \frac{1}{2^k}\prod_{i=1}^{k}(1 + \frac{1}{k})(1 - \frac{n_{\neg l_i} \cdot e}{2^k}) \tag{5.10}$$

By setting:

$$\forall i : x_i = x = \frac{e}{2^k}$$

and using the identity:

$$(1 - x)^\alpha \geq 1 + \alpha \cdot x, \ \ for\ all\ x \in [0, 1)\ and\ \alpha \geq 1$$

we get:

$$Pr[A_C] \leq \frac{(1 + \frac{1}{k})^k}{2^k}\prod_{i=1}^{k}(1 - x_i)^{n_{\neg l_i}} \tag{5.11}$$

$$< \frac{e}{2^k}(1 - x)^{|\Gamma(C)|} \tag{5.12}$$

$$= x_C \prod_{D\in\Gamma(C)}(1 - x_D) \tag{5.13}$$

We have just shown that all the conditions of the lopsided local lemma hold. So, if we sample the variables of ϕ independently and according to the proposed distribution, ϕ is satisfied with strictly positive probability. This means that ϕ is satisfiable for $s = \lfloor \frac{2^{k+1}}{e(k+1)} \rfloor$, meaning that $f(k) \geq \lfloor \frac{2^{k+1}}{e(k+1)} \rfloor$.

Remark:
Notice that, in order to gain the most benefit from the stronger lopsided LLL, we needed to assign values to the variables x_i in a biased way, which is reasonable as $n_{x_i}, n_{\neg x_i}$ can be arbitrary. However, the way we it is done may seem counter-intuitive at first glance. It turns out that when sampling the most frequent variables (which appear to be the bottleneck), we favor their less frequent literal satisfying on average less clauses directly from this literal. The reason behind this, is that the clauses that contain the less frequent literal intersect with more clauses in general, which makes it harder to satisfy them. Even though this bias is sometimes reversed for less frequent variables (e.g less than s/2 occurences), no literal with more than $s/2$ occurences has a probability to be satisfied which is greater than 1/2.

5.1.3 Unsatisfiable CNF formulas from binary trees

In this section, we will show a way to construct unsatisfiable k-CNF formulas using binary trees. The same procedure was used in [10]. All non-leaf nodes of the trees we consider, will have exactly two children.

Given such a tree T and a positive integer k, we define the k-CNF formula $F_k(T)$ as follows:

1. For every non-leaf node, we generate a new variable x_i and label it's left and right children with literals x_i and $\neg x_i$ respectively.

2. For every leaf node w, we build a k-clause C_w by picking as literals the first k labels of nodes we encounter if we follow the unique path towards the root (which is not labelled).

Lemma 5.1.7 *The formula $F_k(T)$ is unsatisfiable.*

Proof: Consider an arbitrary truth assignment α for the constructed variables. Now, form a path starting from the root r in which every label represents a literal where α has assigned the value FALSE. This path will eventuallly stop at a leaf node u. It is now obvious that the clause C_u is not satisfied. These is at least one such clause for each truth assignment. So, F is unsatisfiable.

If there is an unsatisfiable formula in which every variable appears in at most s clauses, then $f(k) < s$. So, using the above construction, it suffices to prove the existence of a binary tree with every node having a bounded number of *close* leaf decendants in order to derive an upper bound for $f(k)$.

Definition 5.1.8 *A leaf w is said to be l-close to a node v, if v is an ansestor of w and their distance is at most l.*

Definition 5.1.9 *A binary tree T will be called a $\underline{(k,d)\text{-tree}}$ if*

1. *Every leaf has depth at least $k+1$.*

2. *For every node v of T, there are at most d leaves which are k-close to v.*

The following lemma explicitely states the connection between the existence of (k,d)-trees the values of $f(k)$ and $l(k)$. It particular, we can prove the upper bounds for these values by proving the existence of specific witness trees.

Lemma 5.1.10 *Let T be a (k,d)-tree. With regard to the unsatisfiable formulas: $F_k(T)$ and $F_{k+1}(T)$, the following hold:*

1. *The formula $F_{k+1}(T)$ is a $(k+1,2d)$-CNF formula where every clause intersects $(k+1)d$ other clauses. So, $l(k+1) \leq (k+1)(d-1)-1$.*
2. *The formula $F_k(T)$ is a (k,d)-CNF formula. So, $f(k) \leq d-1$.*

Proof:

1. By the definition of the (k,d)-tree, for every node v there are at most d leaves k-close to it. So, the corresponding literal can appear in at most d of the clauses constructed. Consider that in order v to be included in a clause C_w of size $k+1$, the leaf w must be k-close to v. Since there are exactly two nodes in T representing the two complementary literals of te same variable, there are at most $2d$ clauses where a particular variable may appear. So, $F_{k+1}(T)$ is a $(k+1,2d)$-CNF formula. Also, if we fix a clause $C=(l_1 \wedge ... \wedge l_{k+1})$ of $F_{k+1}(T)$, each of the $k+1$ literals it contains, appear in at most $d-1$ other clauses (as many as the rest of the leaves k-close to some l_i). So, every clause intersects $(k+1)(d-1)$ other clauses. This implies that:
$$l(k+1) \leq (k+1)(d-1)-1 \Leftrightarrow l(k) \leq k(d-1)-1$$
2. Consider two nodes u,v which have a common parent node p. Clearly, their labels will be $x_i, \neg x_i$ respectively for some i. By definition, there are at most d-leaves k-close to p, which (by induction) implies that there are at most d-leaves $(k-1)$-close to u or $(k-1)$-close to v. So, there are at most d leaves w such that C_w contains either x_i or $\neg x_i$. This implies that $F_k(T)$ is a (k,d)-CNF formula and $f(k) \leq d-1$.

5.2 The bound on $f(k)$ is asymptotically tight for powers of 2.

The asymptotical tightness of the bound in equation **??** can be reduced to the existence of $(k, \Theta(\frac{2^k}{k}))$-trees via lemma 5.1.10.

In particular, we are going to prove the following:

Lemma 5.2.1 *For $k=2^m, m \in \mathbb{N}$, $(k, \frac{2^{k+2}}{k})$-trees exist.*

Proof:

Let T_0 be a full binary tree of height k (i.e T_0 it has 2^k leaves at depth k). We enumerate those leaves starting from the leftmost to the rightmost. So we label them as follows:
$$l_0, l_1, ..., l_{2^k}$$

Now, at each leaf l_i, we attach a full binary tree of height $h = i \; mod \; (\frac{k}{2})$. The resulting tree will be denoted T. Also, let $r(v)$ denote the number of leaves k-close to node v. We need to upper bound the value of $r(v)$. We make the proof by induction. By the construction, the root node (v_0) is k-close to the leaf l_i if and only if $i \equiv 0 \; mod \; \frac{k}{2}$.

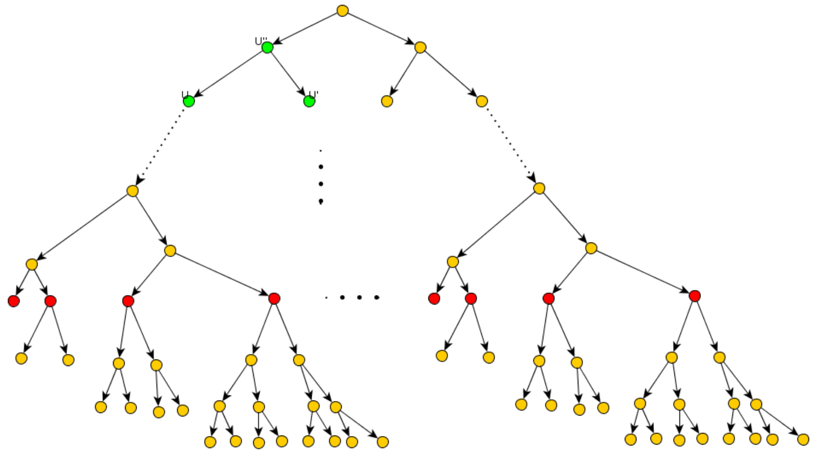

Figure 5.1: The depth of the red nodes is k

The construction for $k = 8$ can be seen in the following figure:

So,

$$r(v_0) = \frac{2^k}{k/2} = \frac{1}{2} \cdot \frac{2^{k+2}}{k}$$

Now, suppose that the result doesn't hold and that u is a node of least depth, for which $r(u) > \frac{2^{k+2}}{k}$. Let, u' be the parent of u. We consider the following two cases:

1. $$Depth(u) = i \leq \frac{k}{2}$$

   Let u' be the sibling of u and u'' be their common parent. Also, let $r_d(u)$ be the number of leaves at distance exactly d from u. Due to the symmetry of the construction and since u and u' cannot be leaves, it holds that exactly half of the leaves k-close to u'' are also $(k-1)$-close to u (the other half are $(k-1)$-close to u').
   Formally,

   $$\sum_{d=0}^{k-1} r_d(u) = \frac{r(u'')}{2} \Rightarrow r(u) = \frac{r(u'')}{2} + r_k(u)$$

   The minimality of the depth of u implies that $r(u'') \leq \frac{2^{k+2}}{k}$. Also, the value of $r_k(u)$ can be computed as follows:
   Firstly, there are exactly 2^{k-i} leaf decendants $\{l_j\}$ of node u in or first tree (T_0). Out of these nodes, there are exactly $\frac{2^{k-i}}{k/2}$ of rank $r \equiv i \ mod \ \frac{k}{2}$. Each of these nodes has 2^i decendants at the level $k + i$ of T (which we are interested in) and no other l_j does.
   So,

   $$r_k(u) = \frac{2^{k-i}}{k/2} \cdot 2^i = \frac{2^{k+1}}{k}$$

   Finally,

   $$r(u) = \frac{r(u'')}{2} + r_k(u) \leq \frac{2^{k+2}}{k}$$

   So, this holds for every node u.

2.

$$Depth(u) > \frac{k}{2}$$

In this case, it is clear that :

$$r(u) = r(u'') \leq \frac{2^{k+2}}{k}$$

since there is no node of T with depth at least $k + \frac{k}{2} - 1$.

5.3 Complexity phase transition

We consider CNF formulas with bounded number of variable occurences.

Definition 5.3.1 *A k-CNF formula is called a (k, s)-CNF formula if every variable appears in at most s clauses.*

Now, we can define the following decision problem:

Definition 5.3.2 *(k, s)-SAT:*
Let ϕ be a (k, s)-CNF formula. Is ϕ satisfiable?

By the definition of the value $f(k)$, it is clear that the problem: (k, s)-SAT is trivial for every $s \leq f(k)$. Of course, (k, ∞)-SAT $\equiv$ k-SAT, which is NP-Complete. So, it is reasonable to ask if there is a finite s such that (k, s)-SAT is NP-complete. If there is such s, where does this compexity hardness jump occur and how sharp it is? As it turns out, this "complexity phase transition" is the sharpest it could be. The following theorem [8, 12] shows that it takes place between two consecutive values of s.

Theorem 5.3.3 *Let $k \geq 3$. The $(k, f(k) + 1)$-SAT problem is already NP-complete.*

Proof: We first introduce a useful gadget:
Given a set of $j \geq 2$ variables $U = \{x_0, x_1, ..., x_{j1}\}$, the 2-CNF formula

$$(x_0 \vee \neg x_1) \wedge (x_1 \vee \neg x_2) \wedge ... \wedge (x_{j-1} \vee x_0)$$

is called an equaliser of U. The equaliser of a singleton set U is the empty formula.

Now let F be a k-CNF formula, $k \geq 3$. For each variable $x \in vbl(F)$, we replace every occurrence (as x or $\neg x$) by a new variable inheriting the sign of x in this occurrence. For each variable $x \in vbl(F)$, we add an equaliser for the set of variables that have replaced occurrences of x. It can be easily proved that:

$$F' \; is \; satisfiable \; \Leftrightarrow \; F \; is \; satisfiable$$

and every variable occurs at most 3 times in F'.

To see this, consider that the formula we constructed is the same as the originalif we disregard that we now put new variables in the different occurences of the same variable. However, we compensate for that by adding the "equalizers".

Example:

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

$$F' = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_4 \vee \neg x_5 \vee x_6) \wedge (x_2 \vee \neg x_4) \wedge (x_4 \vee \neg x_2) \wedge (x_3 \vee \neg x_5) \wedge (x_5 \vee \neg x_3)$$

We are now going to introduce another gadget defined as follows:

1. Fix some minimal unsatisfiable $(k, f(k)+1)$-CNF formula G.
2. Choose some clause C in G and replace one of its literals by $\neg x$ for a new variable x.

This new formula, which we denote by $G(x)$, has the following properties:

1. It is satisfiable (otherwise G would not be minimal).
2. Every satisfying assignment has to set x to 0 (since otherwise G would be satisfiable).
3. All variables have degree at most $f(k)+1$.

The reduction:
We are given a formula F (instance of k-SAT).

1. We generate F' from F as described above.
2. We augment each 2-clause in F by $k-2$ positive literals of new variables so that it becomes a k-clause.
3. For each of the new variables x we add a copy of $G(x)$ to our formula using new variables each time.

The new formula F'' is k-CNF with at most $k\cdot|F|+k^2\cdot|F|+k(k-2)|G|$ literals and is satisfiable iff F is satisfiable.
Moreover, the maximum variable degree is $max\{3, f(k)+1\}$ which is $f(k)+1$, since we assumed $k \geq 3$.

The equivalence can be proved as follows:
It suffices to prove that

$$F''\ is\ satisfiable\ \Leftrightarrow\ F'\ is\ satisfiable$$

If F'' is satisfiable, all the literals added in step 2 of the reduction should be FALSE in every satisfying assignment. Then clearly F' should also be satisfiable. Conversely, if F' is satisfiable, then merging one of it's satisfying assignments with the satisfying assignments for each $G(x)$ (their variable sets are disjoint) will give us a satisfying assignment for F''. Since F' is satisfiable, we can satisfy all clauses from step 2 by the first or the second literal in each clause (which come from F'). These are the only clauses that can contain literals from F' and some $G(x)$ at the same time. So the "merging" is feasible.

Bibliography

[1] Noga Alon. A parallel algorithmic version of the Local Lemma. FOCS 1991, pages 586-593

[2] Alon, N. and Spencer, The probabilistic method, Wiley-Interscience series in discrete mathematics and optimization, 2008

[3] Alon, N. and Spencer, The probabilistic method, Wiley-Interscience series in discrete mathematics and optimization, 2008

[4] Beck Jozsef, An algorithmic approach to the Lovsz local lemma. Volume 2, number 4, Wiley Subscription Services, Inc., A Wiley Company, pages 343-365, 1991

[5] J.H.Rodrigo Bissacot and Roberto Fernández and Aldo Procacci and Benedetto Scoppola. An Improvement of the Lovász Local Lemma via Cluster Expansion, Combinatorics, Probability & Computing, volume 20, number 5, 2011, pages 709-719.

[6] Paul Erdös and Joel Spencer. Lopsided Lovász Local Lemma and Latin transversals, Discrete Applied Mathematics, volume 30, number 2-3, 1991, pages 151-154.

[7] Paul Erdos and Laslo Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In Infinite and Finite Sets, Colloq. Math. Soc. J. Bolyai, volume 11,number 5, 1975, pages 609-627

[8] Heidi Gebauer and Robin A. Moser and Dominik Scheder and Emo Welzl. The Lovász Local Lemma and Satisfiability, Efficient Algorithms 2009, pages 30-54

[9] Heidi Gebauer and Tibor Szabó and Gábor Tardos. The Local Lemma is Tight for SAT, SODA 2011, pages 664-674

[10] Bernhard Haeupler and Barna Saha and Aravind Srinivasan. New Constructive Aspects of the Lovasz Local Lemma, FOCS 2010, pages 397-406

[11] Kashyap Babu Rao Kolipaka and Mario Szegedy. Moser and tardos meet Lovász, STOC 2011, pages 235-244

[12] Jan Kratochvíl and Petr Savický and Zsolt Tuza. One More Occurrence of Variables Makes Satisfiability Jump From Trivial to NP-Complete, SIAM J. Comput., volume 22 , number 1, 1993, pages 203-210

[13] Jochen Messner and Thomas Thierauf. A Kolmogorov Complexity Proof of the Lovász Local Lemma for Satisfiability, COCOON, 2011, pages 168-179

[14] Michael Molloy and Bruce A. Reed. Further Algorithmic Aspects of the Local Lemma, STOC 1998, pages 524-529

[15] Robin A. Moser. A constructive proof of the Lovász local lemma, STOC 2009, pages 343-350

[16] Robin A. Moser. Derandomizing the Lovasz Local Lemma more effectively, CoRR, volume abs/0807.2120, 2008, http://arxiv.org/abs/0807.2120

[17] Robin A. Moser and Gábor Tardos. A constructive proof of the general lovász local lemma. J. ACM , Volume 57, number 2, 2010

[18] Wesley Pegden. An improvement of the Moser-Tardos algorithmic local lemma, CoRR, volume abs/1102.2853, 2011, http://a rxiv.org/abs/1102.2853

[19] Pascal Schweitzer. Using the incompressibility method to obtain local lemma results for Ramsey-type problems, Inf. Process. Lett., volume 109, number 4, 2009 , pages 229-232

[20] Aravind Srinivasan. Improved algorithmic versions of the Lovász Local Lemma. SODA 2008, pages 611-620.