ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΛΟΓΙΚΗΣ ΚΑΙ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ CORELAB

Algorithms inspired by the Lovász Local Lemma

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του **Φώτιου Ηλιόπουλου**

Επιβλέπων: Ευστάθιος Ζάχος, Καθηγητής Ε.Μ.Π

Αθήνα, Φεβρουάριος 2014

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΛΟΓΙΚΗΣ ΚΑΙ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ CORELAB

Algorithms inspired by the Lovász Local Lemma

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του **Φώτιου Ηλιόπουλου**

Επιβλέπων: Ευστάθιος Ζάχος, Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 10/02/2014.

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

Αθήνα, Φεβρουάριος 2014

....
Φώτιος Δ. Ηλιόπουλος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Σκοπός της διπλωματικής αυτής εργασίας είναι η μελέτη, η ανάλυση και η σχεδίαση πιθανοτικών αλγορίθμων τοπικής αναζήτησης για διακριτά προβλήματα περιορισμών με μεθόδους και τεχνικές που χρησιμοποιήθηκαν για την αλγοριθμική απόδειξη του "Lovász Local Lemma". Συγκεκριμένα, θεωρούμε αλγορίθμους της εξής μορφής:

- Άρχισε από μία τυχαία ανάθεση τιμών στις μεταβλητές του προβλήματος περιορισμών.
- Όσο υπάρχουν διαψευδούμενοι περιορισμοί, χρησιμοποίησε μια τυχαιοκρατική διαδικασία για να διαλέξεις έναν διαψευδούμενο περιορισμό c καθώς και νέες τιμές για τις μεταβλητές του

Τόσο για την επιλογή του περιορισμού c όσο και για την επιλογή των τιμών στις μεταβλητές του, υπάρχει πληθώρα ευρεστικών, αλλά όχι πολλά για να τις διαφοροποιήσουν πέρα από πειράματα. Στο paper του R. Moser για την κατασκευαστική απόδειξη του Lovász Local Lemma στο STOC '09 η επιλογή του c είναι αυθαίρετη ενώ η επιλογή της ανάθεσης τιμών στις μεταβλητές γίνεται τυχαία και ομοιόρφα. Ο τερματισμός του αλγορίθμου αποδεικνύεται δείχνοντας ότι μη τερματισμός θα σήμαινε καθολική κωδικοποίηση (universal compression). Τόσο στη δουλειά του Moser όσο και στις μετέπειτα δουλειές, μια βασική απαίτηση είναι ότι θα πρέπει να υπάρχει ένα πιθανοτικό μέτρο - γινόμενο (product measure) επί των μεταβλητών του προβλήματος έτσι ώστε κάθε φορά που επαναδειγματοληπτούμε τις μεταβλητές ενός περιορισμού, οι τιμές που θα τις αναθέσουμε θα πρέπει να προκύπτουν μέσω (της προβολής του) πιθανοτικού μέτρου (στις μεταβλητές του περιορισμού). Αυτό έρχεται σε έντονη αντίθεση με τους αλγορίθμους της πράξης στους οποίους οι τιμές που δίνουμε στις μεταβλητές εξαρτώνται από το πώς ο περιορισμός για τις μεταβλητές του οποίου διαλέγουμε τιμές συσχετίζεται με τους γείτονές του, τόσο στατικά (από τις μεταβλητές που μοιράζονται), όσο και " δυναμικά", δηλαδή από την τωρινή "κατάσταση" αυτών των περιορισμών.

Στην παρούσα διπλωματική δείχνουμε μία μέθοδο για να απαλλαχθεί κανείς από αυτή την απαίτηση. Συγκεκριμένα, δείχνουμε ότι η μέθοδος του Moser, δηλαδή το να φράξουμε τον αριθμό των πιθανών άκαρπων τροχιών του αλγορίθμου χρησιμοποιώντας ιδιότητες της εισόδου, για παράδειγμα το μέγιστο βαθμό του γράφου εξάρτησης, παραμένει πλήρως λειτουργική ακόμα και με την απουσία ενός πιθανοτικού μέτρου, δηλαδή όταν κάθε περιορισμός επιτρέπεται να αλλάξει τις μεταβλητές του με έναν " ιδιωτικό " τρόπο και στην πραγματικότητα, λαμβάνοντας υπόψιν τις τρέχουσες τιμές των μεταβλητών των γειτονικών περιορισμών. Επιπρόσθετα, παρατείθενται ορισμένα βιβλιογραφικά στοιχεία αναφορικά με τις αλγοριθμικές επεκτάσεις του Lovász Local Lemma και μερικές αποδείξεις/ βελτιώσεις υπάρχοντων αποτελεσμάτων χρησιμποιώντας την "εντροπική μέθοδο " του R. Moser.

Abstract

The purpose of this thesis is the study, the analysis and the design of randomized local search algorithms for discrete constraint satisfaction problems, using methods and techniques employed in the constructive proof of the Lovász Local Lemma. Specifically, we consider algorithms that operate as follows:

- Start at a random assignment to the variables of the CSP.
- While violated constraints exist, employ a randomized process to select a violated constraint c and new values for its variables.

For both the choice of c and the choice for its values, there is a cornucopia of heuristics, but not much to distinguish between them besides experiments. In Moser's STOC'09 argument the choice of c is adversarial, while the choice of value assignment is uniformly random. Termination is proven by showing that non-termination would amount to universal compression. Both in Moser's work and in all subsequent related works a key requirement is that there exists a *product probability measure* on the variables such that every time a constraint is resampled, the assigned values must be chosen via (the projection of) that measure (on the constraint's variables). This is in sharp contrast with practical algorithms whose choice of value assignment depends heavily on how the constraint being resampled relates to its neighbors, both statically (shared conflicting variables), but also dynamically, i.e., based on the current "state" (number of satisfied literals) of these constraints.

In our work *we dispense with this requirement.* We show that Moser's method, i.e., bounding the number of possible unfruitful trajectories of the algorithm in terms of structural properties of the input, e.g., the maximum degree of the constraint-conflict/dependency graph, remains fully potent even in the absence of a background product measure, i.e., when each constraint is allowed to resample its variables in a manner specific to itself and, in fact, that takes into account the *current* values of its neighboring constraints. Furthermore, we present work related to the algorithmic aspects of the Lovász Local Lemma and proofs/improvements of already existing results using R. Moser's "entropic method".

Keywords

probabilistic method, entropic method, incompressibility method, Lovász Local Lemma, constructive proof, entropic potential, randomized algorithms, compression, entropy, source coding

Ευχαριστίες

Ολοκληρώνοντας τη διπλωματική αυτή εργασία καθώς και τις προπτυχιακές σπουδές μου θα ήθελα να ευχαριστήσω όλους τους καθηγήτες μου και τους ανθρώπους του ιδρύματος που με βοήθησαν όλα αυτά τα χρόνια. Θα ήθελα να ευχαριστήσω ιδιαιτέρως τον κ. Αχλιόπτα για την πολύ καλή συνεργασία μας καθώς και τον επιβλέποντα καθηγητή κ. Ζάχο. Επίσης θα ήθελα να ευχαριστήσω όλα τα μέλη του Εργαστηρίου Λογικής και Επιστήμης Υπολογιστών (Corelab) και ιδαίτερα τον κ. Φωτάκη που με ενέπνευσε από τα πρώτα εξάμηνα των σπουδών μου. Τέλος, ευχαριστώ την οικογένεια μου για την αμέριστη συμπαράστασή της όλα αυτά τα χρόνια.

Contents

Chapter 1

The Algorithmic Lovász Local Lemma

1.1 Introduction

The Lovász Local Lemma is a powerful tool to prove the existence of combinatorial objects meeting a prescribed collection of criteria. Its' original proof was inherently non-constructive. In 2008 Robin A. Moser [14], and in 2009 Moser and Gábor Tardos [15] gave a constructive proof having as basis an astonishing simple algorithm. In this chapter we state the lemma and present its constructive proof by Moser and Tardos.

1.2 Motivation and Statement of the lemma

The probabilistic method is a method, primarily used in combinatorics, for proving the existence of a prescribed kind of mathematical object. It works by showing that if one randomly chooses objects from a specified class, the probability that the result is of the prescribed kind is more than zero.

In a typical probabilistic proof of a combinatorial result, one usually has to show that the probability of a certain event is positive. However, many of these proofs actually give more and show that the probability of the event considered is not only positive but is large. In fact, most probabilistic proofs deal with events that hold with high probability, i.e, as the dimensions of the problem grow.

On the other hand, there is a trivial case in which one can show that a certain event holds with positive, though very small, probability. Imagine that we have m mutually independent events $A_1, \ldots, A_m$, and each of them holds with probability at least $p > 0$. Then, the probability that all events hold simultaneously is at least $\prod_{i=1}^{m} \Pr[A_i] \geq p^m$, which is positive, although it maybe exponentially small in m.

The Lovász Local Lemma can be thought as a generalization of the above case showing that in case that the events A_i are "not too dependent" the probability of the event of interest still remains positive.

Theorem 1.1 (The Local Lemma; General Case). *Let $\mathcal{A}$ be a finite set of events in a probability space. For $A \in \mathcal{A}$ let $\Gamma(A)$ be a subset of $\mathcal{A}$ satisfying that A is independent from the collection of events $\mathcal{A} \setminus (\{\mathcal{A}\} \cup \Gamma(A))$. If there exists an assignment of reals $x : \mathcal{A} \to (0,1)$ such that*

$$\forall A \in \mathcal{A} : \Pr[A] \leq x(A) \prod_{B \in \Gamma(A)} (1 - x(B)), \tag{1.1}$$

then the probability of avoiding all events in $\mathcal{A}$ is at least $\prod_{A \in \mathcal{A}} (1 - x(A))$, in particular it is positive.

The following two propositions are corollaries of the General Case of the Local Lemma and are widely used in applications.

Corollary 1.1 (The Local Lemma; Symmetric Case)**.** *Let $A_1, \dots, A_m$ be events in an arbitrary probability space. Suppose that each event A_i is mutually independent of a set of all other events A_j but at most d, i.e, $|\Gamma(A_i)| \leq d$, and that $Pr[A_i] \leq p$ for all $1 \leq i \leq m$. If*

$$ep(d+1) \leq 1 \tag{1.2}$$

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

Corollary 1.2 (The Local Lemma; Asymmetric Case)**.** *Let $A_1, \dots, A_m$ be events in an arbitrary probability space. If*

$$\sum_{j \in \Gamma(A_i)} \Pr[A_j] \leq \frac{1}{4} \tag{1.3}$$

for all i then $\Pr[\bigcap_{i=1}^{n} \bar{A}_i] \geq \prod_{i=1}^{m} (1 - 2\Pr[A_i]) > 0$

1.3 Two Applications

A hypergraph $H(V, E)$ is 2-colorable if there is a coloring of V by two colors so that no edge $f \in E$ is monochromatic.

Theorem 1.2. *Let $H(V, E)$ be a hypergraph in which every edge has at least k elements, and suppose that each edge of H intersects at most d other edges. If $e(d+1) \leq 2^{k-1}$ then H is two colorable.*

Proof. We start by coloring each vertex v of H either blue or red uniformly at random and independently. For each edge $c \in E$ let A_c be the event that c is monochromatic. Clearly we have that $\Pr[A_c] \leq \frac{2}{2^{|c|}} = \frac{1}{2^{|c|-1}}$. Moreover, each event A_c is clearly mutually independent of all other events $A_{c'}$ for all edges c' that do not intersect c. The result now follows from corollary 1.1. □

Theorem 1.3. *Let F be a CNF formula such that each clause has length (i.e number of variables) at least 3 and shares variables with at most a_i other clauses of length i, where*

$$\sum_i a_i 2^{-i} \leq \frac{1}{4} \tag{1.4}$$

, then F is satisfiable.

Proof. We start by assigning a uniformly random value (either zero or one) to each variable of the formula independently. For each clause c let A_c be be the event that c is violated. Obviously, $\Pr[A_c] = \frac{1}{2^{|c|}}$. Furthermore, each event A_c is independent from any event $A_{c'}$ such that clauses c and c' do not share variables. Therefore, the result follows from corollary 1.2.

□

1.4 Beyond the General Version

Given a set of events $\{A_1, \dots, A_m\}$, their dependency graph, and their corresponding probabilities $p_1, \dots, p_m$, it is natural to ask what is the optimal condition for the probability to avoiding all these events to be strictly positive. It turns out that Theorem 1.1 is not the optimal condition for a fixed graph since it only uses "local information".

Shearer in [21] found an exact characterization of probabilities for which $\Pr[\bar{A}_1 \wedge \dots \wedge \bar{A}_m] > 0$ whenever $\Pr[A_i] = p_i$ and $\{A_i\}_{i=1}^m$ has dependency graph G. To describe this characterization, let $\mathrm{Indep}(G)$ denote the set of all independent sets of G, including the empty set, and define the quantities

$$q_I = q_I(G,p) = \sum_{J \in \mathrm{Indep}(G), I \subset J} (-1)^{|J|-|I|} \prod_{i \in J} p_i \tag{1.5}$$

for all $I \in \mathrm{Indep}(G)$

Theorem 1.4. *Let G be a dependency graph on $[m]$. Then for a vector $p = (p_1, \dots, p_m)$ of non-zero the following are equivalent:*

1. *For every system $\{A_i\}_{i=1}^m$ of events with $\Pr[A_i] = p_i$ $(1 \le i \le m)$ with dependency graph G it holds that $\Pr[\bar{A}_1 \wedge \dots \wedge \bar{A}_m] > 0$*

2. *$q_I(G,p) > 0$ for all $I \in \mathrm{Indep}(G)$*

   Furthermore, when the above holds, there exists $\{B_i\}_{i=1}^m$ such that $\Pr[B_i] = p_i (1 \le i \le m)$ with dependency graph G such that for every independent set I of G :

$$\Pr[\bigwedge_{i \in I} B_i \wedge \bigwedge_{i \notin I} \bar{B}_I] = q_I(G,p) \tag{1.6}$$

   This is the unique instance that minimizes $\Pr[\bar{A}_1 \wedge \dots \wedge \bar{A}_m]$, and it also has the property that all neighbouring events are disjoint. We call this the extreme instance.

Kolipaka and Szegedy showed in [10] that the algorithm proposed by Moser and Tardos for the General LLL also applies in the Shearer's conditions under the same mild restrictions (see variable version bellow). However, we will not present this proof in this thesis since we think it is out of its purpose.

It is not hard to see that while Theorem 1.4 is optimal the original condition of Lovász is simpler and more practical. The next question that comes to mind is if there are "intermediate" lemmas, i.e lemmas that are stronger than the original LLL but more practical than Shearer's condition. The answer is actually positive as it can be seen in a series of papers by Bissacot [25] , Pegden [24] , Kolipaka Szegedy [11] etc in which such "intermediate" theorems are provided. The two most popular of them are presented below:

Theorem 1.5 (Improved Lovász Local Lemma). *Suppose that G is a dependence graph for the family of events $\{A_i\}$ each one with probability* $\Pr[A_i] = p_i$ *and there exist $\mu(A_i)$ real numbers in* $[0, +\infty)$ *such that, for each event A_i,*

$$p_i \leq R_i^* = \frac{\mu(A_i)}{\phi_i^*(\mu)} \tag{1.7}$$

where

$$\phi_i^*(\mu) = \sum_{R \subseteq \Gamma_G^+(A_i), R \ indep} \prod_{u \in R} \mu([u]) \tag{1.8}$$

Then: $\Pr[\bigwedge_i \bar{A}_i > 0)$

Observation 1.1. *1. Numbers $\mu(A_i)$ of Theorem 1.5 and $x(A_i)$ of Theorem 1.1 are connected via the following relation:*

$$\mu(A_i) = \frac{x(A_i)}{1 - x(A_i)} \tag{1.9}$$

2. The sum in the relation for $\phi_i(\mu)^*$, defined above, is over the independent sets in the neighbourhood of A_i. Therefore, the improvement (comparing to the original LLL) is significant if the subgraphs induced by the vertex neighbourhoods in G are dense. On the other hand, the improvement is vanishing if the dependency graph is triangle-free.

Theorem 1.6 (Clique Lovász Local Lemma). *Let $\{A_1, \dots, A_m\}$ be a set of events with dependency graph G and let $\{K_1, \dots, K_n\}$ be a set of cliques in G covering all the edges (not necessarily disjointly). If there exists a set of vectors $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ from $(0, 1)^m$ such that the following conditions are satisfied,* *1.* $\forall v \in [n] : \sum_{i \in K_v} x_{i,v} < 1$

2. $\forall i \in [m]$, $\forall$ *v such that* $i \in K_v$:

$$\Pr[A_i] \leq x_{i,v} \prod_{u \neq v : K_u \ni i} (1 - \sum_{j \in K_u \setminus \{i\}} x_{j,u}) \tag{1.10}$$

then: $\Pr[\cap_{i \in [m]} \bar{A}_i] \geq \prod_{v \in [n]} (1 - \sum_{i \in K_v} x_{i,v} > 0]$

Observation 1.2. *Although Theorem 1.6 and Theorem 1.5 cannot be compared directly, Clique Lovász Local Lemma gives better results for all examples it has been studied.*

1.5 The Variable Version- MT Algorithm

Moser and Tardos presented in [15] a constructive proof of Theorem 1.1 by imposing one small restriction upon the general setting. That is, they considered events determined by different subsets of underlying mutually independent random variables. This setting is sometimes referred to as the *variable version* of the LLL and, while it appears as the only way in order to get any efficient algorithmic access to the problem, it seems as well to be the case in almost all known applications. We give the exact formulation below.

Let $\mathcal{P} = \{X_1, \ldots, X_n\}$ be mutually independent random variables in a fixed probability space Ω. In this space we define a set $\mathcal{A}$ that consists of m events, viewed as *constraints*, and we denote by $A_i (1 \leq i \leq m)$ the event that the i-th constraint does not hold. We denote by $\mathrm{vbl}(A_i)$ the set of variables on which A_i depends. In particular, if $\mathrm{vbl}(A_i) \cap \mathrm{vbl}(A_j) = \emptyset$ then A_i and A_j are independent. Finally, we define the *dependency graph* $G = G_{\mathcal{A}}$ for $\mathcal{A}$ to be the graph on vertex set $\mathcal{A}$ with an edge between dependent events. For each $A_i \in \mathcal{A}$ we write $\Gamma(A_i) = \Gamma_{\mathcal{A}}(A_i)$ for the neighbourhood of A_i in G. Finally, we define the *inclusive neighborhood* of an event A to be $\Gamma^+(A) := \Gamma(A) \cup \{A\}$.

Consider the following algorithm:

Algorithm 1 MT-Algorithm

```
1: procedure RESAMPLE
2:     for  $i = 1 \to n$ do
3:         $v_i \leftarrow$ a random evaluation of $X_i$
4:     while  $\exists A_i \in \mathcal{A} : A_i$ is violated  do
5:         pick an arbitrary violated event $A_i \in \mathcal{A}$
6:         for  all $X \in \mathrm{vbl}(A_i)$ do
7:             $v_X \leftarrow$ a new random evaluation of $X$
```

Theorem 1.7 (The Lovász Local Lemma; Variable Version). *Let $\mathcal{P}$ be a finite set of mutually independent random variables in a probability space. Let $\mathcal{A} = \{A_1, \ldots, A_m\}$ be a set of m events determined by these variables. If there exists an assignment of reals $x : \mathcal{A} \to (0, 1)$ such that:*

$$\Pr[A_i] \leq x(A_i) \prod_{A_j \in \Gamma(A_i)} (1 - x(A_j)) \tag{1.11}$$

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

1.6 Analysing the MT Algorithm

1.6.1 Execution logs and witness trees

Note that the decision which violated event $A \in \mathcal{A}$ to correct in each step of Algorithm 1 can be taken completely arbitrarily. Fixing any (deterministic or randomized) procedure for this selection makes the algorithm and the expected values we consider well defined. The selection method does not matter for the analysis.

The analysis is based on keeping an accurate journal of what the algorithm does. Let $C : \mathbb{N} \to \mathcal{A}$ list the events as they have been selected for resampling in each step. We call C "the LOG" of the execution. Notice that C is a well defined random variable determined by the random choices the algorithm makes.

A *witness tree* $\tau = (T, \sigma_T)$ is a finite rooted tree T together with a labelling $\sigma_T : V(T) \to \mathcal{A}$ of its vertices with events such that the children of a vertex $u \in V(T)$ receive labels from $\Gamma^+(\sigma_T(u))$. A *proper* witness tree is a witness tree in which the children of the same vertex always receive distinct labels. To simplify the notation, we will write $V(\tau) := V(T)$ and for any $v \in V(\tau)$, we write $[v] := \sigma_T(v)$.

Given the value of C, we will now associate with each resampling t carried out a witness tree $\tau_C(t)$ the following way:

1. Define $\tau_C^{(t)}(t)$ to be an isolated root vertex labelled $C(t)$.
2. For each $i = t - 1, t - 2, \ldots, 1$ go backwards through the log and distinguish two cases:
    (a) If there is a vertex $v \in \tau_C^{i+1}(t)$ such that $C(i) \in \Gamma^+([v])$, then we choose among all such vertices the one having the maximum

distance from the root and attach a new child vertex u to v that we label $C(i)$, thereby obtaining the tree $t_C^{(i)}(t)$. In the selection of the maximum distance vertex we break ties arbitrarily.

(b) If there is no vertex $v \in \tau_C^{(i+1)}$ such that $C(i) \in \Gamma^+([v])$, then we skip the time step i and simply define $\tau_C^{(i)} := t_C^{(i+1)}(t)$.

3. Let $\tau_C(t) := \tau_C^{(1)}(t)$

Lemma 1.1. *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 appears in C is at most $\prod_{v \in V(\tau)} \Pr[[v]]$

Proof. Notice that the procedure cannot output a tree in which 2 vertices u, v with the same label have a common parent w. If *w.l.o.g* vertex u was created after a vertex v, then u could be placed as a child of v (which has greater depth than w). So, (i) is proved by contradiction.

To prove (ii) consider the following procedure that we call $\tau - check$: In an order of decreasing depth (e.g, reversed breadth first order) visit the vertices of τ and for a vertex v take a random evaluation of the variables in $\mathrm{vbl}(v)$ (according to their distribution, independent of possible earlier evaluations) and check if the resulting evaluation violates $[v]$. We say that the $\tau-$ check passes if all events were violated when checked.

Clearly, $\Pr[\tau - \text{check passes}] = \prod_{v \in V(\tau)} \Pr[[v]]$. We are going to prove that: τ occurs in $C \implies$ the $\tau-$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 $P_i^{(0)}, P_i^{(1)}, \ldots$, and whenever either algorithm calls for a new random sample of P_i we pick the next unused value from the sequence.

Let $S_v(P_i) = \{w \in V(\tau) : d(w) > d(v) \wedge P_i \in \mathrm{vbl}(w) \cap \mathrm{vbl}(v)\}$ 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 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 $\tau-$check as well because we take decreasing depth order. Finally, the fact that a vertex v is evaluated by the τ-check means that the event $[v]$ is in the execution log $C \implies$ it was violated the time was written in $C \implies$ that the τ-check will find that $[v]$ is violated (using the above argument). □

For any event $A \in \mathcal{A}$, let us denote by N_A the random variable that counts how many times the event A is resampled during the execution of the algorithm. If C is the log of the execution of our algorithm then N_A is the number of occurrences of A in this log and also the number of distinct proper witness trees occuring in C that have their root labeled A. The latter statement holds because if t_i is the i-th time step with $C(t_i) = A$, then obviously the tree $t_C(t_i)$ contains exactly i vertices labelled A, thus $\tau_C(t_i) \neq \tau_C(t_i)$ unless $i = j$. Therefore one can bound the expectation of N_A simply by summing the bounds in Lemma 1.1 on the probabilities of the occurrences of the different proper witness trees.

1.6.2 Random generation of witness trees

Let us fix an event $A_i \in \mathcal{A}$ and consider the following Galton-Watson branching process for generation 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 $B \in \Gamma^+([u])$ add to v a child node with label B independently with probability $x(B)$.
3. Repeat step 2 until new nodes are added.

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

Lemma 1.2. *Let τ a fixed proper witness tree with its root vertex labelled A. The probability p_τ that the Galton-Watson process described above yields exactly the tree τ is*

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

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

$$p_\tau = \frac{1}{x(A)} \prod_{v \in V(\tau)} \left(x([v]) \prod_{u \in W_v} (1 - x([u])) \right), \tag{1.12}$$

□

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

$$p_{\mathcal{T}} = \frac{1-x(A)}{x(A)} \prod_{v\in V(\tau)} \left(\frac{x([v])}{1-x([v])} \prod_{u\in\Gamma^+([u])} (1-x([u])) \right), \tag{1.13}$$

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

$$p_{\mathcal{T}} = \frac{1-x(A)}{x(A)} \prod_{v\in V(\tau)} \left(x([v]) \prod_{u\in\Gamma([u])} (1-x([u])) \right) = \frac{1-x(A)}{x(A)} \prod_{v\in V(\tau)} x'([u]). \tag{1.14}$$

Let $\mathcal{T}_A$ denote the set of all proper witness trees having the root labelled A. We have:

$$\mathbb{E}[N_A] = \sum_{\tau\in\mathcal{T}_A} \Pr[\tau \text{ appears in the log } C] \le \sum_{\tau\in\mathcal{T}_A} \prod_{v\in V(\tau)} \Pr[[v]] \le \sum_{\tau\in\mathcal{T}_A} \prod_{v\in V(\tau)} x'([v]), \tag{1.15}$$

where the first inequality follow from Lemma 1.1, while the last inequality follows from the fact that the Galton-Watson process produces exactly one tree at a time (not necessarily one fro $\mathcal{T}_A$ since it might also grow infinite). This concludes the proof of Theorem 1.7.

1.7 The Lopsided Local Lemma

We can compute $\Pr[\bigcap_{i=1}^m A_i]$ from the inclusion-exclusion formula as long as we know the probability of the other $2^m - 1$ atomic events created by the A_is. Obviously this is not very helpful because of the large number of atoms and the alterating signs in the formula. The LLL is valuable exactly because we can deduce the positiveness of the above probability from simple information on the dependency graph and the probabilities of the events. It may occur however, that simple information other that the latter can lead us to conclusion we want to get. There is at least one well known condition, that relies on other than $G_{\mathcal{A}}$ and $(\Pr[A_1], \Pr[A_2], \ldots, \Pr[A_m])$ alone. This is known as the *lopsided* LLL. Here instead of the dependency graph we can have aBranching Lemmany arbitrary graph, which call the *lopsidependency graph.* Of course this does not come without a cost. Specifically:

Definition 1.1 (Lopsidependency Graph). *Let $A_1, \ldots, A_m$ be events in a probability space, G a graph on the indices. We say G is a lopsidependency graph (for the events) if*

$$\Pr[A_i | \bigwedge_S \bar{A}_j] \leq \Pr[A_i] \tag{1.16}$$

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

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

Theorem 1.8 (Lopsided LLL). *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*

$$\Pr[A_C | \bigwedge_{D \in J} \bar{A}_D] \leq \Pr[A_C] \tag{1.17}$$

Suppose there are real numbers $0 < x(C) < 1$ for $C \in I$ such that for every $C \in I$ we have

$$\Pr[A_C] \leq x(C) \prod_{D \in \Gamma(C)} (1 - x(D)) \tag{1.18}$$

Then

$$\Pr[\bigwedge_{C \in I} A_C] > 0 \tag{1.19}$$

Moser and Tardos adapted this framework in to their setting by defining two events A_i and A_j to be *lopsidependent*, if there exist two evaluations f and g on the variables in $\mathcal{P}$ that differ only on variables in $\mathrm{vbl}(A) \cap \mathrm{vbl}(B)$ such that f violates A and g violates B but either f does not violate B or g does not violate A. Instead of the usual dependency graph they consider the one on the vertex set $\mathcal{A}$, where lopsidependent events are connected by an edge, and prove the analogous lopsidependent theorem of Theorem 1.7. We present here their theorem:

We write $\Gamma'(A) = \Gamma'_{\mathcal{A}}(A)$ for the neighborhood of an event A in this graph. Clearly, if $\mathrm{vbl}(A)$ is disjoint from $\mathrm{vbl}(B)$, then A and B cannot be lopsidependent, so we have $\Gamma'(A) \subseteq \Gamma(A)$. Substituting $\Gamma'(A)$ for $\Gamma(A)$ in the statement of Theorem 1.7 makes the assumption weaker and therefore the theorem itself stronger. Specifically we get:

Theorem 1.9 (Lopsided Local Lemma Variable Version)**.** *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 $x : \mathcal{A} \to (0,1)$ such that:*

$$\forall A \in \mathcal{A} : \Pr[A] \leq x(A) \prod_{B \in \Gamma'_{\mathcal{A}}(A)} (1 - x(B)) \tag{1.20}$$

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

1.8 Dealing with superpolynomially many bad events

In this section we study further algorithmic aspects of the Lovász Local Lemma. Specifically we a present a theorem due to [17] that 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 1.10. *Suppose there exists $\epsilon \in [0,1)$ and an assignment of reals $x : A \to (0,1)$ such that:*

$$\forall A_i \in \mathcal{A} : \Pr[A_i] \leq (1-\epsilon) x(A_i) \prod_{B \in \Gamma(A_i)} (1 - x(B)) \tag{1.21}$$

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

$$T := \sum_{A_i \in \mathcal{A}} x_{A_i} \leq n \log(\frac{1}{\delta}) \tag{1.22}$$

Furthermore:

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

Proof. The main idea is of relating T to n and δ is to use the high density of the dependency graph G (see below) and the nature of the LLL-conditions which force highly connected events to have small probabilities and x-values.

The first part of the theorem follows immediately from theorem 1.7, because we have proved that if $\epsilon = 0$, 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.

We proceed with proving the bound on T. Let,

$$A_{P_i} = \{A_i \in \mathcal{A} | P_i \in \mathrm{vbl}(A_i)\} \tag{1.23}$$

i.e A_{P_i} is the clique of the dependency graph G that is formed by the events that include the variable P_i. Observe that that the m vertices of G can be partitioned into n such cliques with potentially further edges between them, and therefore has at least $n \cdot \binom{m/n}{2} = m^2/(2n) - m/2$ edges, which is high density for $m >> n$. To prove the bound on T it is sufficient to prove that :

$$\forall P_i : \sum_{B \in A_{P_i}} x_B \leq \log(1/\delta) \tag{1.24}$$

, 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:

$$\delta \leq x_{A_i} \prod_{B \in A_P \setminus \{A_i\}} (1 - x_B) = \frac{x_{A_i}}{1 - x_{A_i}} \prod_{B \in A_P} (1 - x_B) \tag{1.25}$$

There are two cases depending on the value x_{A_i}.

(a) If $x_{A_i} \leq \frac{1}{2}$, then $\frac{x_{A_i}}{1-x_{A_i}} \leq 1$ implies that $\delta \leq \prod_{B \in A_P} (1 - x_B) \leq e^{-\sum_{B \in A_{P_i}}} x_B$ and therefore:

$$\sum_{B \in A_{P_i}} x_B < \log_2(\frac{1}{\delta}) \tag{1.26}$$

(b) If $x_{A_i} > \frac{1}{2}$, let $B_1 \in A_P \setminus \{A_i\}$. Then,

$$\begin{aligned} \delta &\leq x_{A_i} \textstyle\prod_{B \in A_P \setminus A_i} (1 - x_B) \\ &= x_{A_i}(1 - x_{B_1}) \textstyle\prod_{B \in A_P \setminus \{A_i, B_1\}} (1 - x_B) \\ &= x_{A_i}(1 - x_{B_1}) e^{-\sum_{B \in A_P \setminus (A_i cup B_i)}} \end{aligned} \tag{1.27}$$

Since x_{A_i} is the smallest x-value, $\frac{1}{2} \leq x_{A_i} \leq x_{B_1} \leq 1$. Therefore, using some calculus we get that $x_{A_i}(1 - x_{B_1}) \leq e^{-(x_{A_i}+x_{B_1})}$ and therefore:

$$\delta \leq e^{-\sum_{B \in A_{P_i}} x_B} \tag{1.28}$$

Thus, once again: $\sum_{B \in A_{P_i}} < \log_2(\frac{1}{\delta})$

The second part of the theorem now also follows directly from the main theorem of Moser and Tardos and by a simple application of Markov's inequality. Specifically, in section 5 of their paper, Moser and Tardos show that saving a $(1-\epsilon)$ factor in the probability of every resampling step implies that with high probability, no witness tree of size $\Omega(\frac{1}{\epsilon} \log \sum_{A \in \mathcal{A}} \frac{x_A}{1-x_A})$ occurs. This easily implies that none of the n variables can be resampled more often. It is furthermore shown that without loss of generality all x-values can be assumed to be bounded away from 1 by at least $O(\epsilon)$. This simplifies the upper bound on the expected running time to $n \cdot O(\frac{1}{\epsilon} \log \frac{T}{\epsilon})$
□

Theorem 1.10 implies that the number of resampling MT algorithm needs in order to find a satisfying assignment is polynomial in n and δ. Note that, without loss of generality, $\delta \leq \frac{1}{4}$ because otherwise all $A \in \mathcal{A}$ are independent, that is, defined on disjoint set of variables. Indeed if $\delta > \frac{1}{4}$ and there is an edge in G between $A \in \mathcal{A}$ and $B \in \mathcal{A}$, then we have $\frac{1}{4} > x(A)(1 - x(B))$ and $\frac{1}{4} > x(B)(1 - x(A))$, and thus $\frac{1}{4} \cdot \frac{1}{4} > x(A)(1 - x(A)) \cdot x(B)(1 - x(B))$ which is a contradiction because $x(1-x) \leq \frac{1}{4}$ for all x.

The fact that the number of resampling is polynomial in n does not immediately implies that the MT algorithm is polynomial in n if if the number of events m is exponential in n. This is because at each step of the algorithm we have to find a violated event (before we resample it).

Definition 1.2. *A set $\mathcal{A}$ of events that are determined by variables in $\mathcal{P}$ is* efficiently verifiable *if given an arbitrary assignment to $\mathcal{P}$, we can efficiently find an event $A \in \mathcal{A}$ that holds or detect that there is no such event.*

To overcome this obstacle, i.e the sets of events that are not efficienlty verifiable, Haupler, Saha and Srinivasan introduced the notion of *core subset*:

Definition 1.3. *A set $\mathcal{A}' \subseteq \mathcal{A}$ will be called a* core subset *of $\mathcal{A}$ if it has the additionally property of being efficiently verifiable.*

The main idea is to find a core subset of $\mathcal{A}' \subseteq \mathcal{A}$ and to apply the MT-algorithm on it. In other words, we only resample events from $\mathcal{A}'$ and terminate when we cannot find one such violated event. The non-core events will have small probabilities and will be sparsely connected to core events and as such their probabilities in the output distribution of the algorithm does not blow up much. Specifically, we prove the following theorem:

Theorem 1.11. *Assume there is an assignment of reals $x : \mathcal{A} \to (0, 1)$ such that the general LLL conditions 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 $\Pr[B]\prod_{C\in\Gamma(B)}(1-x_C)^{-1}$. In particular, the probability of B being true in the output distribution of MT obeys this upperbound.*

Proof. As we saw in section 1.6, if B became true at least once during the execution of the algorithm, then there must be witness tree with root node B and non-root nodes $V'(\tau) \subseteq \mathcal{A} \setminus \{B\}$. Using a very similar way 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 labelled B. Also for $A \in \Gamma(B)$, let τ_A be the subtree starting from A. Following the proof of lemma 1.2 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 process defined in the section 1.6.2, is exactly:

$$\begin{aligned} \Pr[\tau] &= \prod_{A\in\Gamma_G(B)} x(A)\cdot\Pr[\tau_A] \prod_{A\in\Gamma_G(B)\setminus\Gamma_G(B)} (1-x_A) && (1.29)\\ &= \prod_{A\in\Gamma_G(B)} (1-x(A)) \prod_{u\in V'(\tau)} x'[u] && (1.30)\end{aligned}$$

Furthermore, if $W(B)$ is the random variable equal to the number of witness trees with root B that appear in an execution of the algorithm then:

$$\begin{aligned} \mathbb{E}[W(B)] &\le \Pr[B]\sum_{\tau\in\mathcal{T}_B}\prod_{v\in V(\tau)\setminus\{B\}} \Pr[[v]] && (1.31)\\ &\le \Pr[B]\sum_{\tau\in\mathcal{T}_B}\prod_{v\in V(\tau)\setminus\{B\}} x'([u]) && (1.32)\\ &= \Pr[B]\prod_{A_i\in\Gamma(B)}(1-x(A_i))^{-1}\sum_{\tau\in\mathcal{T}_B} p_\tau && (1.33)\\ &\le \Pr[B]\prod_{A_i\in\Gamma(B)}(1-x(A_i))^{-1} && (1.34)\end{aligned}$$

Finally, an application of the Markov's inequality yields the theorem. □

Using theorem 1.11 we can now prove the following theorem that makes formal the ideas presented above regarding the core subset.

Theorem 1.12. *Let $\mathcal{A}' \subseteq \mathcal{A}$ be an efficiently verifiable core subset of $\mathcal{A}$. If there is an $\epsilon \in [0,1)$ and an assignment of reals $x : \mathcal{A} \to (0,1)$ such that:*

$$\forall A_i \in \mathcal{A} : \Pr[A_i] \leq (1-\epsilon)x(A_i) \prod_{B \in \Gamma(A_i) \cap \mathcal{A}'} (1 - x(B)), \tag{1.35}$$

Then the modified MT-algorithm can be efficiently implemented with an expected number of resamplings according to theorem 1.11. The algorithm furthermore outputs a satisfying assignment with probability at least $1 - \sum_{A_i \in \mathcal{A} \setminus \mathcal{A}'} x_{A_i}$

Proof. The expected number of resampling is given by the standard proof of the MT algorithm if we apply it on $\mathcal{A}'$. So all it remains is to bound the failure probability of the modified MT-algorithm. According to hypothesis we have that:

$$x(A_i) \geq \Pr[A_i] \prod_{B \in \Gamma(A_i)} (1 - x(B))^{-1} \tag{1.36}$$

Combining the above inequality with theorem 1.11 we get that the probability that a non-core event A_i is violated by the output assignment of the MT algorithm is at most $x(A_i)$. A simple union bound yields the theorem.

□

We proceed now by presenting a theorem that indicates when theorem 1.12 is applicable. Notice that this is not trivial since in order for one to use theorem 1.12, one has to be able to find an efficient verifiable core $\mathcal{A}'$ and further to prove that the events in $A \subseteq \mathcal{A}'$ have small probabilities.

Theorem 1.13. *Assume $\log \frac{1}{\delta} \leq \text{poly}(n)$. Assume further that there is a fixed constant $\epsilon \in (0,1)$ and an assignment of reals $x : \mathcal{A} \to (0, 1-\epsilon)$ such that:*

$$\forall A_i \in \mathcal{A} : \Pr[A_i]^{1-\epsilon} \leq x(A_i) \prod_{B \in \Gamma(A_i)} (1 - x(B)) \tag{1.37}$$

Then for every $p \geq \frac{1}{\text{poly}(n)}$ the set $\mathcal{A}' = \{A_i \in \mathcal{A} : \Pr[A_i] \geq p\}$ has size at most $\text{poly}(n)$, 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 $O(n \log m)$ resamplings and returns a good assignment with probability at least $1 - n^c$, where $c > 0$ is any desired constant.

Proof. To prove that $\mathcal{A}'$ is efficiently verifiable it suffices to prove that its size is polynomial in n. Indeed, since by definition $A_i \in \mathcal{A}'$ implies that

$x(A_i) \geq p$ and:

$$\sum_{B \in \mathcal{A}'} x(B) \leq \sum_{B \in \mathcal{A}} x(B) \leq O(n \log(1/\delta) \tag{1.38}$$

we have that $|\mathcal{A}'| \leq O(\frac{n \log(1/\delta)}{p}) = \text{poly(n)}$, so $\mathcal{A}'$ is efficiently verifiable. For $\epsilon > 0$ sufficiently small we have:

$$\forall A_i \in \mathcal{A} : \Pr[A_i] \leq x(A_i) \leq 1 - \epsilon \tag{1.39}$$

Therefore, using the hypothesis we have that:

$$\begin{aligned} \Pr[A_i] &\leq \Pr[A_i]^{\epsilon} x(A_i) \prod_{B \in \Gamma(A_i)} (1 - x(B)) \\ &< (1-\epsilon)^{\epsilon} x(A_i) \prod_{B \in \Gamma(A_i)} (1 - x(B)) \\ &< (1 - \Theta(\epsilon^2)) x(A_i) \prod_{B \in \Gamma(A_i)} (1 - x(B)) \end{aligned} \tag{1.40}$$

and now we can apply theorem 1.10 to prove that the algorithm terminates after $O(n \log n)$ resamplings.

All it remains is bound the failure probability of the algorithm. Note that for every non-core event $A \in \mathcal{A} \setminus \mathcal{A}'$ we have that:

$$\Pr[A_i] \leq p^{\epsilon} x(A_i) \prod_{B \in \Gamma(A_i)} (1 - x(B)) \tag{1.41}$$

Applying theorem 1.10 with $x^*(A_i) = p^{\epsilon} x(A_i)$ gives:

$$\sum_{A_i \in \mathcal{A} \setminus \mathcal{A}'} x^*(A_i) = O(p^{\epsilon} \text{poly}(n)) = O(p^{\epsilon} n^k) \tag{1.42}$$

Choosing $p = n^{-\frac{1}{\epsilon}(k+c)} = \frac{1}{\text{poly}(n)}$, then by theorem 1.12 we get that the failure probability is at most n^{-c} on non-core events, while the core is always avoided.

$\square$

Chapter 2

Constructive Proofs for CNF Formulas: Incompressibility Method

2.1 Introduction

The Lovász Local Lemma applied to satisfiability problem states that a k-CNF formula is satisfiable if each clause has common variables with at most $\frac{2^k}{e} - 1$ other clauses. In this chapter we present R.Moser's [14] constructive proof (slightly altered using the ideas of [12]) of the Lovász Local Lemma applied to such formulas, which is based on an ingeniously simple technique, usually referred to as the "Incompressibility Method" (also knows as "The Entropic Method"). Furthermore, we apply this method to show a constructive result for CNF formulas in which the are clauses with different number of variables as well as the Lefthanded Local Lemma [23] for the case of k-CNF Formulas with dependency graphs that are chordal. Finally, note an extension of this argument will also be the main tool upon which we develop our ideas in chapter 3.

2.2 Constructive Proof of the LLL for the case of k-CNF Formulas

2.2.1 The Recursive Algorithm

Let F be an arbitrary $k-$CNF formula with n variables and m clauses where we have fixed an arbitrary total order over the set of clauses. The *variables* of a clause are the variables that occur in a clause, positive or negative. A *literal* is a variable or the negation of a variable.

The *dependency graph* of formula F is the graph with vertex set the set of clauses, where two clauses are connected if they share variables. For a clause c we denote by $\Gamma(c)$ the set of neighbours of c in the dependency graph, and by $\Gamma^+(c) = \Gamma(c) \cup \{c\}$ the *inclusive* neighbourhood of c. The Lovász Local Lemma implies the following:

Theorem 2.1. *Let d be the maximum degree in the dependency graph of the $k - CNF$ formula F. If*

$$\frac{(d+1)^{d+1}}{d^d} < 2^k \tag{2.1}$$

then F is satisfiable.

Moser presented a simple algorithmic proof for this Theorem by considering the following algorithm:

Algorithm 2 Recursive Algorithm

```
1: procedure FIX($c$)
2:     resample vbl($c$)
3:     repeat
4:         $u \leftarrow$ lowest indexed unsatisfied clause in $\Gamma^+(c)$.
5:         Fix($u$)
6:     until the inclusive neighbourhood of $c$ is satisfied
7:     return
8: procedure GLOBAL FIX
9:     Pick a random truth assignment $\sigma_0$ for the variables of $F$.
10:    while there current assignment does not satisfy $F$ do
11:        $c \leftarrow$ lowest indexed violated clause
12:        Fix ($c$)
13:    return the current assignment
```

It is not hard to see that if the above algorithm terminates then we have found a satisfying assignment for formula F. Clearly however, there is a chance that it runs forever. To show that it terminates under the conditions of Theorem 2.1 we use the incompressibility method, an argument presented by Moser in its breakthrough paper [14].

2.2.2 The Incompressibility Method

In order to execute the above algorithm we imagine an infinite, predefined, uniformly random binary string R. Each time a clause c resamples its variables, our algorithm reads k new random bits from R and it assigns them to the variables of c. It also reads n bits at the beginning in order to define the initial truth assignment. The key of the analysis is to keep a (compact) record of each step of the algorithm, in such a way that at any step T, the record until step T and the final truth assignment, denoted by σ_T, are enough to deduce the prefix of the random string that has been "consumed" up to this step. In particular, the number of prefixes that lead to a violating truth assignment σ_T is smaller than the set of all possible records of all steps and final assignments σ_T. The total number of random string prefixes are 2^{kT+n}. Therefore, if we prove that the number of possible records is $2^{C+T(k-\epsilon)}$ for positive constants C, ϵ, this shows that the algorithm terminates with high probability after a finite number of steps.

2.2.3 Preliminaries

- **Binomials and Entropy** By h we denote the binary entropy, $h(p) = -p\log_2(p) - (1-p)\log_2(1-p)$ for $0 < p < 1$. We use the following well known upper for for $\binom{Td}{T}$. For $d \leq 2$, and $T \geq 1$

$$\binom{Td}{d} < 2^{d \cdot h(1/d)T} \tag{2.2}$$

- **The number of d-ary labelled forests with at most m trees and T nodes**

  An important tool in our analysis is an estimate of the number of d-ary labelled forests with at most m trees and T nodes, denoted by $F_{d,m}(T)$. Recall that a d-ary tree is a tree where each node has *at most* d children. We have that (see [19]):

$$F_{d,m}(T) = \frac{m}{d \cdot T + m}\binom{d \cdot T + m}{m} \tag{2.3}$$

  Using the aforementioned inequality for the binomial coeffecient we have that for $d \geq 2$ and $T, m \geq 1$

$$\begin{aligned} F_{d,m}(T) &< 2^{(d \cdot T + m)\cdot h(\frac{1}{d+\frac{m}{s}})} \\ &< 2^{(d \cdot T + m)\cdot h(\frac{1}{d})} \end{aligned} \tag{2.4}$$

2.2.4 The LOG

We now precise what we meant by compact record of each step of the algorithm. During the execution of the algorithm, we form a LOG in which we encode the sequence of resampled clauses along with 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 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.

We start by showing that we can reconstruct the trajectory of the algorithm from the LOG.

Lemma 2.1. *Let $C_T = c_1, c_2 \ldots, c_T$ be the sequence of resampled clauses and let $P_T = \sigma_0, \sigma_1, \ldots, \sigma_T$ be the corresponding sequence of truth assignments, i.e, σ_i is the truth assignment after resampling c_i.*

(a) C_T and σ_T determine P_T

(b) P_T determines the random bits used in the first T resamplings.

Proof. (a) For each $1 \leq i \leq T$, the resampling of c_i implies that c_i was unsatisfied under σ_{i-1}. Moreover σ_i and σ_{i-1} differ only on the variables in c_i. Since we are given σ_T we can thus determine σ_{T-1} as the truth assignment that results by changing σ_T so that c_T is unsatisfied. Similarly for c_{T-1}, σ_{T-1} and σ_{T-2}, etc.

(b) Notice that σ_0 determines the first n random bits used for the choice of σ_0, as well as c_1. The knowledge of σ_1, c_1 and σ_0 determine the next k random bits used. The trajectory up to this point (i.e σ_0, σ_1) implies the next clause that was resampled, i.e c_2. Repeating the process in the same fashion implies the lemma.

□

2.2.5 Encoding-Decoding the LOG

From the LOG we will next define a *recursion forest* that represents the sequence of resampled clauses. Our final algorithm to reconstruct the random bits used by the Recursive Algorithm will have a coding of this forests as input.

We construct the recursion forest of a LOG $(c_1, \dots, c_T, \sigma_T)$ as follows. Suppose c_i is the i-th clause to be resampled. Then:

(a) If c_i was picked by procedure GLOBAL FIX (at line 11 of Algorithm 2) create a new tree (with one node) with c_i as the label of its root.

(b) If c_i was picked by procedure FIX(c_j) (at line 5 of Algorithm 1 ($j < i$)) then we know that c_j is already added in the forest. Add a node labelled c_i as a child of c_j, to the right of the rightmost child of c_j.

It is not hard to see that traversing the recursion forest in preorder gives the sequence of resampled clauses. All it remains is to bound the number of possible recursion forests and thus to bound the number of possible LOGs. In order to do so will need the following lemma.

Lemma 2.2. *Given that a call of Fix terminates, then the number of unsatisfied clauses after termination is strictly less than it was before that call*

Proof. 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 F, and let $V' \subseteq V$ be the set of vertices-clauses which were "explored" by that call of Fix. Obviously, every descendant 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 $V \setminus_{u \in V'} \Gamma^+(u)$ are not altered by this part of 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 deduce the following corollary:

Corollary 2.1. *The number of trees in the recursion forest is at most m.*

Proof. Each time the algorithm starts a new tree, by the time the tree is finished, the total number of satisfied clauses have increased by at least one, i.e, by the root-clause of the tree. Therefore, the total number of trees in the forest cannot be more than m.

□

To count the recursion forests we simply have to encode them. To do so the next observation is crucial: we don't need to store the labels of the nodes, because we can compute the labels from the precise structure of the forest. Namely, we can use the order of the clauses as they appear in F as an order on the clauses. This induces an order on al the sets $\Gamma^+(c)$. Hence, in the recursion forest, we can think of every node as having $d+1$ slots reserved in a certain order, one for each child to come. That is, we can for example distinguish the tree where the child of c_1 has label c_3. Similar, for the potential roots of the trees, we have m slots reserved. When c_i becomes the root of a tree, we put it at slot j , if j is the rank of c_i in the order of all clauses. Therefore, if we know the precise structure of the forest, we can reconstruct the labels of the nodes. Since we can enumerate all $(d+1, m)$-forests with T nodes we are able to bound the number of the recursion forests.

2.2.6 Putting things together

Now we have all the tools for the incompressibility method argument. Specifically we prove the following Theorem that implies Theorem 2.1.

Theorem 2.2. *Let F be a k-CNF formula and d be the maximum degree of its dependency graph. If*

$$\frac{(d+1)^{d+1}}{d^d} < 2^k \tag{2.5}$$

then the Recursive Algorithm finds a satisfying assignment for F after $O(m)$ steps with high probability

Proof. We will say that an input (random) string R is T-good if the algorithm terminates after T steps, by consuming some prefix R' of R, otherwise we will say that R is T-bad. Observe that:

1. The number of random strings that can be an input for a T - step execution of the algorithm are exactly 2^{n+kT}.
2. There is a bijection between the T-bad random strings and the number of LOGs for T-step execution for the algorithm.

Therefore, the number of T-bad random strings are at most:

$$2^n \cdot F_{d,m}(T) < 2^{n+m\cdot h(\frac{1}{d+1})+d\cdot h(\frac{1}{d+1})\cdot T} \tag{2.6}$$

An easy calculation shows that the assumption $\frac{(d+1)^{d+1}}{d^d} < 2^k$ is equivalent to $k-(d+1)\cdot h(\frac{1}{d+1}) = g > 0$ for integral $d \geq 2$.

Let $\mathrm{Bad}(T)$ denote the number of T-bad random strings. Since the random string R is chosen uniformly at random, if S is the random variable that equals the number of steps of the recursive algorithm then:

$$\begin{aligned}\Pr[S=T] &= \tfrac{\mathrm{Bad}(T)}{2^{n+kT}} \\ &\leq \tfrac{2^{n+m\cdot h(\frac{1}{d+1})+d\cdot h(\frac{1}{d+1})\cdot T}}{2^{n+kT}} \\ &\leq 2^{m\cdot h(\frac{1}{d+1})}\cdot 2^{-Tg}\end{aligned} \tag{2.7}$$

Let $T_0 = \frac{m\cdot h(\frac{1}{d+1})}{g}$. It is easy to see that the probability that the recursive algorithm runs more than $T_0+\lambda$ steps is less than $2^{-\lambda\cdot g}$. Since $T_0 = O(m)$ the theorem is implied.

□

2.3 An asymmetric result for CNF Formulas

In this section we prove the following theorem for CNF Formulas where there are clauses with different number of variables. It should be noted that, as far as we know, this is a new result.

Theorem 2.3. *Let F be a CNF formula with m clauses and n variables and let $k = \max_{c \in F} |c|$. For each clause $c \in F$ define $d_c = \sum_{\gamma \in \Gamma^+(c)} 2^{k-|\gamma|}$ and let $d = \max_{c \in F} d_c$. If:*

$$\frac{(d+1)^{d+1}}{d^d} < 2^k \tag{2.8}$$

then F is satisfiable and Algorithm 2 finds a satisfying assignment for F after $O(m)$ steps

Proof. For each clause c define the real number $a(c) = k - |c|$ and say that a clause c is *needy* if $a(c) > 0$. We will form a formula F' by replacing the needy clauses of F one-by-one as follows.

To replace a needy clause c, introduce $a(c)$ new variables $x_{c_1}, x_{c_2}, \dots, x_{a(c)}$ and replace c by the $2^{a(c)}$ distinct clauses each of which contains the literals of c and one signed combination of the $a(c)$ new variables. We will denote the set of the new clauses as children(c).

Observe now that F is satisfiable if and only if F' is satisfiable. Namely, if we find a satisfying assignment for F' we have also found a satisfying assignment for F. Furthermore, observe that F' is a k-CNF formula that satisfies the conditions of the Local Lemma, and therefore the proof of the previous section can be used to prove that F' is satisfiable. To see that the recursive algorithm terminates after $O(m+n)$ steps (and not after $O(|F'|+n)$ steps) notice that the number of trees in the recursion forest is at most m (and not $|F'|$) since at each state σ and for each clause c only one of the clauses in children(c) can be violated.

□

2.4 A constructive Proof for the Left Handed LLL for the case of k-CNF Formulas

2.4.1 Statement of the Lefthanded Lovász Local Lemma

An interesting variant of the Local Lemma is the Lefthanded Lovász Local introduced by Pegden in [22] and [23]. The lemma holds for a specific family of dependency graphs, but for those graphs that it hold it gives better results than the General LLL. In this section we state the lefthanded lemma for the case of chordal graphs, where it is optimal. As Pegden showed, in this case the left handed local lemma is optimal, meaning it is equivalent to Shearer's condition (Theorem 1.4). However, as we will see, its statement is closer to the General LLL, which means that it is not hard to apply. First we need some definitions:

Definition 2.1. *A* tree order *is a partial order, denoted by $\leq$, in which $w < u, v$ implies that u and v are comparable .*

Notice that every partial ordered set whose partial order is a tree order, has a unique maximal element. In particular, any set with a tree order can be represented as tree whose vertices are labelled with elements of the set, with the root being the maximal element.

Definition 2.2. *A graph is a* lefthanded graph *with respect to a tree-order $\leq$ if*

1. *$u \sim v$ implies that $u \leq v$ or $v \leq u$, and*
2. *$(w < u < u)$ and $(v \sim w)$ together imply $(v \sim u)$*

A *subtree graph* is a graph whose vertex-set is a set of subtrees of some fixed tree, where adjacency corresponds to intersection.

Lemma 2.3. *A graph is lefthanded with respect to some tree-order if and only if it is isomorphic to a subtree graph. Furthermore, a graph is isomorphic to a subtree graph if and only if it is chordal.*

Lemma 2.4. *A graph is lefthanded if and only if is chordal. Furthermore, given a chordal graph G, one can find a tree order $\leq$, so that G is lefthanded with respect to this tree order, in polynomial time.*

Proofs for lemmata 2.3 and 2.4 can be found in [23]. We are now ready to state the theorem.

Theorem 2.4 (The Lefthanded Lovász Local Lemma)**.** *Consider a family of events $\{A_v\}_{(v\in V)}$ with a lefthanded dependency graph $(G, \leq)$ on V. If there is an assignment of numbers $0 \leq x_v \leq 1 \quad (v \in V)$ such that for any $v \in G$*

$$\Pr[A_v] \leq x_v \prod_{u\sim v, u\leq v} (1 - x_u) \tag{2.9}$$

Then we have

$$\Pr[\bigcap_{A\in\mathcal{A}} \bar{A}] \geq \prod_{v\in G} (1 - x_v) \tag{2.10}$$

The proof given by Pegden for Theorem 2.4 was not constructive. However, due to the work of Kolipaka and Szegedy in [10] we know that the MT algorithm terminates when applied to (variable) settings where the Left-handed Lovász Local Lemma holds. An immediate corollary of the above discussion is the following.

Corollary 2.2 (Symmetric LLLL on k-CNF Formulas)**.** *Let F be a k-CNF Formula with m clauses and n variables whose dependency graph is left-handed with respect to a tree order $\leq$. For each clause $c \in F$ let $\Gamma_{\leq}(c) = \{u \in \Gamma(c) : u \leq c\}$ and $d = \max_{c\in F} \Gamma_{\leq}(c)$. If*

$$\frac{(d+1)^{d+1}}{d^d} < 2^k \tag{2.11}$$

then F is satisfiable and there exists an algorithm that finds a satisfying assignment for F after $O(m)$ steps

In the next subsection we give a constructive proof for corollary 2.2 using the incompressibility method.

2.4.2 A Focused Recursive Algorithm

Recall that each tree order on a specific set can be represented as a tree whose vertices are labelled with elements of the set. Let τ be the labelled (with clauses of the formula) tree that corresponds to $\leq$. Consider the following algorithm :

Algorithm 3 Focused Recursive Algorithm

```
1: procedure FIX($c$)
2:     resample vbl($c$)
3:     repeat
4:         $u \leftarrow$ lowest indexed unsatisfied clause in $\Gamma_{\leq}(c)$.
5:         Fix($u$)
6:     until the inclusive neighbourhood of $c$ is satisfied
7:     return
8: procedure LEFTHANDED GLOBAL FIX
9:     Pick a uniformly random assignment
10:    $\tau' \leftarrow \tau$
11:    while $\tau' \neq \emptyset$ do
12:        Let Leaf($\tau'$) be the set of leaves of $\tau'$
13:        while There exist violated clause in Leaf($\tau'$) do
14:            $c \leftarrow$ The lowest indexed violated clause in Leaf($\tau'$)
15:            Fix($c$)
16:        $\tau' \leftarrow \tau' \setminus Leaf(\tau')$
```

In words, Algorithm 2.4.2 traverses τ in backwards BFS fashion starting from the lowest indexed leaf and applies the FIX procedure to every violated clause it meets. Notice that unlike the case of the standard recursive algorithm it is not yet obvious that the termination of Algorithm 2.4.2 implies satisfiability. To prove this we need the following lemma.

Lemma 2.5. *Let s be a subtree of τ and let c be the label of its root. Let σ be a state such that every clause in s, except for the root, is satisfied. Call* Fix(c) *and let σ' be the output, assuming it terminates. Then, every clause of s, including the root, is satisfied in σ'.*

Proof. Let $u_1, \ldots, u_l$ be the l clauses in $\Gamma_{\leq}(c)$ that correspond to the labels of the recursion tree that corresponds to FIX(c). Notice that by definition, this clauses belong in the subtree of τ with root c. By induction, we may assume that every clause that belonged in the subtrees of τ with roots $u_1, \ldots, u_l$ remained satisfied after the termination of FIX(c). Therefore, it remains to prove that every member of $\Gamma_{\leq}(c)$ that does not belong in $\{u_1, \ldots, u_l\}$ or in one of the subtrees of τ with roots $\{u_1, \ldots, u_l\}$ remained satisfied after the termination of FIX(c). Observe however that (1) and (2) in Definition 2.2 imply that such clauses do not exist.

$\square$

Lemma 2.5 asserts that if Algorithm 2.4.2 terminates, then it finds a satisfying assignment for F. Furthermore, lemma 2.5 that the sequence of resampled clauses can be represented as a $d+1$-ary forest with at most m roots. Therefore, following the exact similar proof of the previous section Corollary 2.2 yields.

Chapter 3

Probabilistic Hill Climbing of Entropic Potentials

3.1 Motivation

For a CNF formula F with n variables, let $E : \{0,1\}^n \to \mathbf{N}$ be the (energy) function counting the number of violated clauses at each value assignment. A number of practically useful satisfiability algorithms start at a random $\sigma \in \{0,1\}^n$ and, while violated clauses exist, employ a randomized process to select a violated clause c and new values for its variables, leading to a new state τ. For both the choice of c and the choice of values for its variables there exists a cornucopia of heuristics, but not much to distinguish between them besides experiments, the systematic conduct of which forms an entire subarea of satisfiability research. One property that is shared by the vast majority of these heuristics is that they are primarily guided by the new number of violated clauses, $E(\tau)$, i.e., they largely focus on the energetic potential near the current truth assignment σ.

Inspired by Moser's [14] groundbreaking algorithmic argument for the Lovász Local Lemma we show that there is an alternative *entropic* potential to consider that lends itself to rigorous mathematical analysis. Focusing on this potential suggests a systematic methodology for designing algorithms for any discrete-valued CSP, while also elucidating why certain existing heuristics perform as well as they do. For example, our work suggests an explanation for the following, rather surprising, experimental finding of Balint and Schöning [2]: in choosing new values for c one should focus on minimizing the number of clauses that will become unsatisfied, ignoring the clauses that will become satisfied. As a matter of fact, trying to avoid satisfying too many clauses can enhance the performance of the algorithm.

3.2 Introduction

In a discrete Constraint Satisfaction Problem (CSP) we are given a set of variables $V = v_1, \dots, v_n$ with discrete domains and collection of functions $\mathcal{F}$, called constraints, where each $f \in \mathcal{F}$ takes as input a subset of the variables and outputs an element of $\{0,1\}$. Whenever a value assignment is such that a constraint evaluates to 1 we say that a constraint is satisfied, otherwise that it is violated. Given a CSP we would like to know if there exists a value assignment to the variables which simultaneously satisfies all constraints, i.e., a satisfying assignment, and, if the answer is positive, to find such an assignment efficiently.

A powerful tool for determining the *existence* of a satisfying assignment, developed in the context of the Probabilistic Method, is the Lovász Local Lemma (LLL) [6]. The LLL *presumes* the existence of a probability space within which we are interested in the probability that none of a collection $\mathcal{A}$ of "bad" events occurs. Clearly, this probability is always positive if the events in $\mathcal{A}$ are independent (and none of them has probability 1). The LLL asserts that the probability is, in fact, positive under a condition of limited dependence among the events in $\mathcal{A}$.

LLL (General). *Let $\mathcal{A}$ be a set of m events such that each $A_i \in \mathcal{A}$ is mutually independent of $\mathcal{A} - (\mathcal{D}_i \cup A_i)$, for some $\mathcal{D}_i \subseteq \mathcal{A}$. If there exist real numbers $x_1, \ldots, x_m \in [0,1)$ such that $\Pr(A_i) \le x_i \prod_{A_j \in \mathcal{D}_i}(1 - x_j)$ for every $i \in [m]$, then the probability that none of the events in $\mathcal{A}$ occur is at least $\prod_{i=1}^{m}(1 - x_i) > 0$.*

Even though the general LLL is stated as an implication, it is in fact a characterization. To see this, consider any k-CNF formula F with n variables and m clauses and let A_i denote the event that clause c_i is violated. By the correctness of the theorem, if F is unsatisfiable, then no matter what μ is, there are no $x_i \in [0,1)$ satisfying the theorem's conditions. On the other hand, if $\sigma \in \{0,1\}^n$ is a satisfying assignment of F, then under the trivial measure that places all its mass on σ, taking $x_i = 0$ for all i satisfies the conditions. In other words, if a formula F is satisfiable, there always exist a probability measure μ and numbers $\{x_i\}$ that satisfy the LLL condition for F, thus witnessing its satisfiability.

The value of the LLL lies in that in many settings even the most uneducated choice of measure, unlike in the paragraph above, combined with a straightforward choice for the x_i, yields highly non-trivial results. For example, if every clause in F shares variables with at most Δ other clauses, then taking μ to be the uniform measure on $\{0,1\}^n$ and letting $x_i = 1/(\Delta + 1)$, implies that F is satisfiable if $\Delta \le 2^k/e - 1$. How good is this bound on Δ? It is *tight*. Gebauer, Szabó, and Tardos [7] proved that there exist unsatisfiable k-CNF formulas where each clause shares variables with at most $(1 + o_k(1))2^k/e$ other clauses.

As one can imagine, whenever the LLL establishes that "good configurations exist", the question of whether these can be found efficiently soon follows. Finding the answer to this question has been a long quest, starting with the work of Beck [3], with subsequent works of Alon [1], Molloy and Reed [13], Czumaj and Scheideler [4], Srinivasan [17] and others, establishing increasingly weaker conditions under which good configurations can be found efficiently, yet in all cases still requiring significantly more than the LLL. The breakthrough was made by Moser [14] a few years ago, who showed that a shockingly simple algorithm nearly matches the LLL condition for k-CNF formulas. Very shortly afterwards, Moser and Tardos [15] closed the gap completely, also establishing a general setting within which the LLL is constructive.

Specifically, the so-called *variable setting* of Moser and Tardos [MT] requires that the measure μ is a measure over n explicitly presented variables, the values of which fully determine the events of interest. Crucially, the measure μ must be a *product* measure over the n variables so that sampling from μ amounts to setting each variable v independently of all others using its own measure $\mu(v)$. In the variable setting we can associate with each event A_i a subset of variables $\mathrm{vbl}(A_i)$ that determine A_i, and we can express dependencies by letting $\Gamma(A_i)$ be the set of events A_j for which $\mathrm{vbl}(A_i) \cap \mathrm{vbl}(A_j) \neq \emptyset$, writing $j \sim i$. In this variable setting Moser and Tardos introduced and analyzed the algorithm Resample, given below.

Algorithm 4 RESAMPLE

```
1: for  $i \in [n]$ do
2:     Sample variable $v_i$ according to $\mu(v_i)$
3: while at least one of $A_1, \ldots, A_m$ occurs do
4:     Pick an arbitrary occurring event $A_j$
5:     Sample each variable $v \in \mathrm{vlb}(A_j)$ according to $\mu(v)$
6: return the current assignment
```

Theorem 3.1 ([15])**.** *If there exist real numbers $x_1, \ldots, x_m \in (0,1)$ such that $\Pr(A_i) \leq x_i \prod_{j \sim i}(1 - x_j)$ for every $i \in [m]$, then the average number of resamplings until* Resample *terminates is $\sum_{i \in [m]} x_i(1 - x_i)^{-1}$.*

For example, for k-CNF formulas, if we set each variable to 0 or 1 with equal probability, i.e., $\mu(v) = 1/2$ for every v, and for every $i \in [m]$ we let $x_i = 1/(\Delta + 1)$ where $\Delta = \max_c |\Gamma(c)|$, Theorem 3.1 implies that if $\Delta \leq 2^k/e - 1$, exactly as in the LLL, a satisfying assignment will be found in $O(m)$ time.

As mentioned earlier, the value of the LLL lies in the fact that we get useful results even if we chose the measure μ in an oblivious manner [18]. In particular, since typically one does have access to explicitly presented variables, e.g., the logical variables of a Boolean formula, or the colors to be assigned to the vertices of a graph, in the vast majority of applications of the LLL μ is simply taken to be the product measure in which each $\mu(v)$ is uniform over the the domain of variable v. Doing this makes the search for suitable $\{x_i\}$ relatively straightforward and, in practice, one typically does not even need to use the general LLL directly, but can use instead one of the following two corollaries (offering "pre-packaged" $\{x_i\}$'s).

Corollary 3.1 (Symmetric LLL)**.** *If each event A_i is mutually independent of all but at most d other events and $\Pr[A_i] \leq p$ for all $i \in [m]$, then if $p(d+1) \leq 1/e$, the probability that none of the A_i occurs is positive.*

Corollary 3.2 (Asymmetric LLL)**.** *If $\sum_{j \sim i} \Pr[A_j] \leq 1/4$ for every event A_i, then the probability that none of the A_i occurs is positive.*

In this frame of mind, Resample seems like a perfectly natural algorithmization of the LLL. Yet, it is hard to miss its striking similarity to randomized local search algorithms for CSPs. Given that such algorithms perform well in practice, way beyond what the LLL and Resample can guarantee, it is tempting to ask if the approach of Moser and Tardos [15] for analyzing Resample can be applied to them.

Unfortunately, there is a huge obstacle in this direction: the requirement of product measure. Local search algorithms typically resample the variables of each constraint in a way that depends both on which constraint is being resampled and, more importantly, on the current value of neighboring variables. But for the approach of [15], the requirement of a product measure is, in the words of Joel Spencer, "critical for the entire result" [16]. Similarly for all works following [15], including [12, 8, 10, 11], and up to the most recent work of Harris and Srinivasan [9], where the *partial* resampling approach is developed: to resample an event A_j, resample an appropriate random *subset* of $\mathrm{vbl}(A_j)$ using the product measure.

Our work is inspired instead by Moser's groundbreaking entropic argument of the symmetric LLL for k-CNF formulas [14]. Specifically, our contribution begins with the realization that the proof technique of Moser remains fully potent even without the backdrop of a probability measure on the variables, i.e., outside the realm of "The Probabilistic Method". Specifically:

> Rather than requiring each variable to carry its own measure and insisting that each and every value assignment to each and every variable is independent of all other value assignments (and obeys the variable's utterly uniformed measure), we allow each and every resampling of each and every constraint to have its own resampling distribution that *depends on the current state* of the system.

We formulate our analysis in the context of satisfiability for CNF formulas but our ideas easily carry over to arbitrary discrete-valued CSPs. Our approach reduces the problem of choosing good resampling distributions for each clause c to a *constrained source coding* problem, the size of which depends only on $\Gamma(c)$, i.e is independent of the size of the formula. We develop both a *symmetric* and *asymmetric* version of our conditions such that, in the case of uniform resampling, they reduce to Corollaries 3.1 and 3.2 (applied to CNF formulas). Unlike the asymmetric version of the LLL which is typically used to deal with constraints of different arities, in our setting, the freedom to choose arbitrary probability distribution per clause (and state) brings about a whole new role for asymmetry. Specifically, when the source coding problem associated with one or more clauses is infeasible (in a sense we make precise), the degree of (in)feasibility of each problem highlights "where the pain is" vs. which clauses are "easy to satisfy". The asymmetric conditions then, make it possible to use this information to modify the resampling distributions so that the easy to satisfy clauses help the hard to satisfy clauses, potentially making all associated source coding problems feasible.

Our conditions take as input the CSP instance itself, not just its dependency graph (as the LLL/MT conditions do). As a result, our analysis is sensitive not only to "how many" constraints share variables with each constraint, but also to "how" these neighboring constraint are connected to it, e.g., do the clauses share one or more variables, are there triplets $\{i, j, k\}$ such that $i \sim j$, $j \sim k$, and $k \sim i$, etc. For example, they readily encompass the (and often significantly subsume) the improvements that result by considering the *lopsided* dependency graph in the variable as formulated by Moser and Tardos [15].

Finally, we note that from a strict worst-case point of view, our analysis only improves upon the LLL/MT conditions in specialized settings which are hard (and perhaps too ugly) to abstract away. Rather, we consider our main contribution the demonstration that entropic arguments, such as the one pioneered by Moser [15], offer a new tool for run-time analysis that may have practical implications in algorithm design.

3.3 Setting

Let F be an arbitrary CNF formula with clauses $\mathcal{A} = \{c_1, \ldots, c_m\}$. For $\sigma \in \{0,1\}^n$, let $U(\sigma) = U_F(\sigma)$ denote the set of unsatisfied clauses of F under σ. We will analyze algorithms which as long as $U(\sigma) \neq \emptyset$ select a clause $c \in U(\sigma)$ and assign to its variables an element of $\{0,1\}^{|c|}$ selected from a probability distribution $R_c(\sigma)$, i.e., one that depends on the current state. For now, these resampling distributions will be arbitrary. Let $\beta > 0$ be the largest real number such that in every distribution every value assignment receives probability either 0 or at least $2^{-\beta}$. For the analysis, it will be helpful to consider algorithms that continue resampling clauses even after a satisfying assignment has been found. With this in mind, let $\pi_1, \pi_2 \ldots$ be an infinite sequence of permutations of $[m]$. We will analyze the following ceaseless algorithm.

```
• Let $i = 1$ and set $\sigma_i = \mathbf{0}$.
• Repeat forever:
    – If $U(\sigma_i) \neq \emptyset$ then let $c$ be the lowest indexed clause in $U(\sigma_i)$
      according to $\pi_i$
                else let $c$ be the lowest indexed clause according to
      $\pi_i$
    – Let $\sigma_{i+1}$ be the state that results by resampling $c$ according to
      $R_c(\sigma_i)$.
    – $i \leftarrow i + 1$
```

Remark 3.2. *If $\pi_1 = \pi_2 = \cdots$, then the algorithm fixes an ordering of the clauses and always resamples the lowest violated clause per the order-*

ing. If the π_i are i.i.d. uniformly random permutations, then the algorithm resamples a random (violated) clause in each step.

To simplify exposition, we will consider the sequence $\pi_1, \pi_2, \ldots$ to be arbitrary but fixed, i.e., we will consider it part of the algorithm's description, similarly to $\sigma_1 = \mathbf{0}$.

Definition 3.1. *We refer to the (random) sequence $\sigma_1, \ldots, \sigma_{t+1}$, entailing the first t resamplings, as the algorithm's* trajectory *of length t, or t-*trajectory*. A t-trajectory is* bad *iff $\sigma_1, \ldots, \sigma_{t+1}$ are all unsatisfying. For a bad trajectory Σ, the sequence of t clauses that were resampled along Σ is called its* witness *$W(\Sigma)$.*

There are two key ideas involved in bounding the probability that the algorithm's trajectory is bad, both introduced in Moser's groundbreaking proof [14].

- The map from a bad t-trajectory Σ to $\langle W(\Sigma), \sigma_{t+1} \rangle$ is 1-1. Specifically, if $W(\Sigma) = r_1, \ldots, r_t$, then σ_t is the truth assignment that results by setting in σ_{t+1} the variables of r_t to the unique value assignment violating r_t (the uniqueness stemming from the nature of satisfiability constraints). And so on.
- The growth of the set of witnesses of a formula as a function of t can be bounded in a non-trivial way by *static* considerations, i.e., without unfolding the algorithm's probabilistic dynamics.

To refine the second point above in our work, we model the possible executions of the algorithm as an infinite tree whose root is labelled by the value of σ_1, i.e., $\mathbf{0}$, per π_1. If c is the clause resampled by the algorithm at σ_1, then the root has $2^{|c|}$ children corresponding to (and labelled by) each possible value of σ_2. More generally, for every $i \geq 1$, a vertex labeled by a possible value σ of σ_i has $2^{|c|}$ children, each child labeled by a distinct possible value of σ_{i+1}, where c is the clause resampled by the algorithm when $\sigma_i = \sigma$ (recall that the clause selected for the i-th resampling depends only on σ_i and π_i). We thus get an infinite vertex-labelled tree. We also label every edge of the tree between a parent vertex labelled σ and a child vertex labelled τ with the probability that the algorithm makes the transition $\sigma \to \tau$ conditional on having reached the vertex labelled σ.

We call the above labelled infinite tree the *computation tree* and note that it is nothing but the unfolding of the Markov chain corresponding to the state-evolution of the algorithm. In particular, for every vertex v of the tree the probability, p_v, that an infinite trajectory will go through v equals the product of the edge-labels on the root-to-v path. In visualizing the computation tree it will be helpful to draw each vertex v at (Euclidean) distance $\log_2 p_v$ from the root. This way all finite trajectories whose last vertex is at the same distance from the root are equiprobable, even though they may entail wildly different numbers of resamplings (this also means that sibling vertices are not necessarily equidistant from the root). Finally, we color the vertices of the computation tree as follows. For every infinite path that starts at the root determine its maximal prefix forming a bad trajectory. Color the vertices of the prefix red and the remaining vertices blue. So, for example, whenever F is unsatisfiable every vertex of the computation tree is red.

In terms of the above picture, our goal will be to prove that there exists a critical radius x_0 and $\delta > 0$, such that the proportion of red states at distance x_0 from the root is at most $1 - \delta$. Therefore, if we repeatedly "run the algorithm until it consumes x_0 bits of randomness", the probability we only encounter red states in s/δ runs is at most $\exp(-s)$. Crucially, x_0 will be polynomial, in fact linear, in $m = |F|$.

To prove that red vertices thin out beyond a critical radius we stratify the computation tree as follows. Fix any real number $x > 0$ and on each infinite path starting from the root mark the first vertex of probability 2^{-x} or less. Truncate the computation tree so that the marked vertices become leaves (of a finite tree). Let $L(x)$ be the set of root-to-leaf paths in this finite tree (trajectories) and let $B(x) \subseteq L(x)$ consist of the bad trajectories. Now, let I be the random variable equal to an infinite trajectory of the algorithm and let $\Sigma = \Sigma(x)$ be the prefix of I in $L(x)$. By definition, $\sum_{\Sigma \in L(x)} \Pr[\Sigma] = 1$, and $\Pr[\Sigma] \in (2^{-x-\beta}, 2^{-x}]$. Let $P = P(\Sigma)$ be the maximal red prefix of Σ and observe that if $\Sigma \in B(x)$ then $P = \Sigma$. Therefore,

$$H[P] \geq \sum_{\Sigma \in B(x)} \Pr[\Sigma](-\log_2 \Pr[\Sigma]) \geq x \sum_{\Sigma \in B(x)} \Pr[\Sigma] = x \Pr[\Sigma \in B(x)] \ . \tag{3.1}$$

Assume now that there exist $M, \epsilon > 0$, such that $H[P] \leq (1-\epsilon)x + M$, for every $x > 0$. Then (3.1) implies that for every $x \geq \theta M/\epsilon$,

$$\Pr[\Sigma \in B(x)] \leq \frac{H[P]}{x} \leq \frac{(1-\epsilon)x + M}{x} \leq 1 - \epsilon + \epsilon/\theta \ ,$$

and, thus, for $\theta > 1$, the probability of finding a satisfying assignment is strictly positive. Moreover, observe that by adding the numbers along the edges of the computation tree traversed so far, i.e., by keeping track of "how much randomness it has consumed so far", the algorithm can keep track of the probability p_v of the current vertex. Therefore, for any fixed $x > 0$, the algorithm can test whether its trajectory is in $L(x)$ or not. With this in mind, consider the following meta-algorithm, with parameters T and x_0:

- $t \leftarrow 1$.
- While $t \leq T$ do:
  - Run the algorithm until the trajectory is in $L(x_0)$

– $t \leftarrow t+1$

Setting $x_0 = M(2/\epsilon)$ and $T = s(2/\epsilon)$, we see that the probability that the meta-algorithm never encounters a blue state is at most $(1-\epsilon/2)^{s(2/\epsilon)} < \exp(-s)$. In other words, consuming $(4M/\epsilon^2)s$ random bits suffices to drive the probability of failure to find a satisfying assignment down to $\exp(-s)$.

3.4 Local Distributions

For each clause $c \in F$, let $\Gamma(c)$ denote the set of clauses with which c shares at least one variable and let $\Gamma^+(c) = \Gamma(c) \cup \{c\}$. For a value assignment σ violating c, the set of *c-critical* clauses consists of the satisfied clauses in $\Gamma(c)$ whose every satisfied literal belongs to a variable shared with c. We denote the characteristic vector of the set of c-critical clauses as $v(\sigma) = v_c(\sigma) \in \{0,1\}^{\Gamma(c)}$ and refer to it as the *critical vector* of c in σ. We denote by $\mathcal{V}(c)$ the union of c-critical vectors over all $\sigma \in \{0,1\}^n$ and note that it is quite possible that $\mathcal{V}(c) \subset 2^{\Gamma(c)}$, i.e., that not all c-critical vectors may be realizable. Finally, for each critical vector $v \in \mathcal{V}(c)$, if $D(v) \subseteq \Gamma^+(c)$ denotes the union of c and the c-critical clauses in v, we define $\mathrm{Br}(v) = 2^{D(v)}$, i.e., the set of all possible subsets of $\Gamma^+(c)$ that may become (or in the case of c remain) unsatisfiable when resampling c at v. Again, it is important to note that even if $D(v) = \Gamma^+(c)$, the set $\mathrm{Br}(v)$ may be much smaller than $2^{\Gamma^+(c)}$ due to our choice of $R_c(v)$.

We will focus on probability distributions for which $R_c(\sigma) = R_c(v_c(\sigma))$, i.e., which out of the entire current value assignment (state) σ, only consider [the projection of σ amounting to] the critical vector $v_c(\sigma)$. In that sense, the resampling probability distributions are "local", since variables in clauses outside $\Gamma(c)$ are ignored, and "coarse", since non-c-critical clauses are not distinguished by, say, how many satisfied literals they have (and, therefore, by the impact of c's resampling to their criticality with respect to other clauses).

Definition 3.2. *For each clause c, the* potential *of c, is*

$$\mathrm{Pot}(c) = \min_{v \in \mathcal{V}(c)} H[R_c(v)] \ . \tag{3.2}$$

In words, $\mathrm{Pot}(c)$ is a lower bound on the *average* amount of randomness consumed by any resampling of clause c. It will be convenient to also define the potential of sets of clauses, i.e., $\mathrm{Pot}(S) = \sum_{c \in S} \mathrm{Pot}(c)$.

Definition 3.3. *For each clause $c \in \mathcal{A}$ and $v \in \mathcal{V}(c)$, let $\mathcal{B}_v^c : \mathrm{Br}(v) \to [0,1]$ be the probability distribution on the elements of $\mathrm{Br}(v)$ that results by resampling c at critical vector v according to $R_c(v)$.*

3.5 Amenability

For reasons that will become clear later, the key quantity we need to control is determined by the following experiment, which we refer to as the unit-experiment. Fix a clause $c \in \mathcal{A}$ and a probability distribution $\mathcal{P}$ on its set of critical vectors $\mathcal{V}(c)$. Fix also a function f that takes as input a critical vector v and a subset B of the set of clauses that are critical in v, i.e., $\langle v \in \mathcal{V}(c), B \in \mathrm{Br}(v)\rangle$, and outputs a probability distribution f_v^B on the subsets of B.

Finally, for each clause $c \in F$ let $\mathrm{Self}(c) \leq \mathrm{Pot}(c)$ be an arbitrary real number and define $\mathrm{Parent}(c) = \mathrm{Pot}(c) - \mathrm{Self}(c)$. For a set of clauses S we define $\mathrm{Self}(S)$ and $\mathrm{Parent}(S)$ similarly as $\mathrm{Pot}(S)$. Let the random variable B^* be defined by the following experiment:

1. Select $v \in \mathcal{V}(c)$ according to $\mathcal{P}$.
2. Select $B \in \mathrm{Br}(v)$, according to $\mathcal{B}_v^c$.
3. Select $B^* \subseteq B$, according to f_v^B.

Definition 3.4. *A clause c is ϵ-amenable under resampling distributions R_c if there exists $\epsilon > 0$ such that for every distribution $\mathcal{P}$ and every function f,*

$$H[B^* \mid |B^*|] + 2\mathbb{E}|B^*| \leq (1-\epsilon)\left(\mathrm{Self}(c) + \mathbb{E}[\mathrm{Parent}(B^*)]\right) \ . \tag{3.3}$$

Theorem 3.3. *If every $c \in F$ is ϵ-amenable, then for every $x > 0$,*

$$H[P] \leq (1-\epsilon)x + n + 2m + \beta \ . \tag{3.4}$$

3.6 A first application

Theorem 3.4. *Let F be a CNF formula with $\max_{c \in F} |c| = k$. If for every $c \in F$, $\sum_{\gamma \in \Gamma^+(c)} 2^{k-|\gamma|} < 2^{k-2}$, then there exists $\epsilon > 0$ such that every clause in F is ϵ-amenable.*

The hypothesis of Theorem 3.4 has long been known to imply the satisfiability of F via the LLL. Moreover, the existence of an efficient algorithm under the same hypothesis follows readily by considering the uniform measure on $\{0,1\}^n$ and applying Theorem 1 of Moser and Tardos [15]. Observe now that our main result, Theorem 3.3, allows one to establish algorithmic efficiency from a *local* condition on the distributions, amenability, without any global requirement that the resampling distributions in their totality form some coherent probabilistic object (or, worse, that they must be projections of some product measure). Our purpose in proving Theorem 3.4 is to demonstrate that this flexibility does not come at a significant cost, indeed none in this case (we discuss this point further later). In particular, when F is a k-CNF formula, Theorem 3.4 applies whenever $|\Gamma^+(c)| < 2^{k-2}$ for every c in F, matching Moser's [14] original result. (Note that in the proof below we do use a product measure, but this fact is not considered by the proof.)

Proof of Theorem 3.4. For every clause c and every value assignment σ let $R_c(\sigma)$ be the uniform distribution over all $2^{|c|}$ assignments to the variables of c and $\mathrm{Self}(c) = 0$. Thus, $\mathrm{Pot}(c) = \mathrm{Parent}(c) = |c|$ for every $c \in F$. Let $\Gamma^{+,i}(c)$ denote the set of all i-subsets of $\Gamma^+(c)$. For each $c \in F$, let $a(c) = k - |c|$, and for a set of clauses S let $a(S) = \sum_{\gamma \in S} a(\gamma)$.

Fix any clause c and consider the unit-experiment for c. To lighten notation, let $p_j = p^c_j = \Pr[|B^*| = j]$ and $p_j(S) = p^c_j(S) = \Pr[B^* = S \mid |B^*| = j]$. Then, $\mathbb{E}\mathrm{Pot}(B^*) - H[B^* \mid |B^*|] - 2\mathbb{E}|B^*|$ equals

$$\begin{aligned}
& \sum_j p_j \sum_{S\in\Gamma^{+,j}(c)} p_j(S)\left(\mathrm{Pot}(S) + \log_2 p_j(S) - 2j\right) \\
= & \sum_j p_j \sum_{S\in\Gamma^{+,j}(c)} p_j(S)\left(jk - a(S) + \log_2 p_j(S) - 2j\right) \\
= & \sum_j p_j\, j(k-2) - \sum_j p_j \sum_{S\in\Gamma^{+,j}(c)} p_j(S)(-\log_2 p_j(S) + a(S)) \\
= & \sum_j p_j\, j(k-2) - \sum_j p_j \sum_{S\in\Gamma^{+,j}(c)} \sum_{i=1}^{2^{a(S)}} \frac{p_j(S)}{2^{a(S)}} \log_2 \frac{2^{a(S)}}{p_j(S)} \qquad (3.5) \\
\geq & \sum_j p_j\, j(k-2) - \sum_j p_j \log_2 \left(\sum_{S\in\Gamma^{+,j}(c)} 2^{a(S)} \right) , \qquad (3.6)
\end{aligned}$$

where (3.6) follows from from the fact that the rightmost sum in (3.5) can be seen as the entropy of a distribution over a set with $\sum_{S\in\Gamma^{+,j}(c)} 2^{a(S)}$ elements. To conclude the proof we will prove by induction on j that under the hypothesis of the lemma, for every integer j,

$$\sum_{S\in\Gamma^{+,j}(c)} 2^{a(S)} < 2^{j\cdot(k-2)} , \qquad (3.7)$$

implying that the quantity in (3.6) is strictly positive and, therefore, that c is ϵ-amenable for some $\epsilon > 0$. Observe that the hypothesis of the lemma is (3.7) with $j = 1$. Moreover, if the lemma holds for $j \geq 1$, then

$$\sum_{S\in\Gamma^{+,j+1}(c)} 2^{a(S)} \leq \sum_{S'\in\Gamma^{+,j}(c)} 2^{a(S')} \sum_{\gamma\in\Gamma^+(c)} 2^{a(\gamma)} < \sum_{S'\in\Gamma^{+,j}(c)} 2^{a(S')}\cdot 2^{k-2} \leq 2^{(k-2)(j+1)} .$$

□

3.7 Representing Trajectories by Break Sequences

To prove Theorem 3.3, i.e., to control bad trajectories and thus $H[P]$, we introduce the following notation.

Definition 3.5. *Let $B_0 = U(\sigma_1)$. For $i \geq 1$, let*

$$B_i = \begin{cases} U(\sigma_{i+1}) \setminus U(\sigma_i) & \textit{if } \ \sigma_{i+1} \neq \sigma_i \\ \{r_i\} & \textit{if } \ \sigma_{i+1} = \sigma_i \end{cases}$$

That is, B_i is the set of clauses "broken" by the i-th resampling, where a clause "breaks itself" if it remains violated after its own resampling (something that implies $\sigma_{i+1} = \sigma_i$).

Let Σ be a bad t-trajectory and let $W(\Sigma) = r_1, \ldots, r_t$ be its witness. As we saw, we can recover Σ from $\langle W(\Sigma), \sigma_{t+1}\rangle$. We will see that there is sequence $B_0^*, B_1^*, \ldots, B_{t-1}^*$ of subsets of B_i that can be used as an alternative to $W(\Sigma)$ to recover Σ. Informally, each B_i^* is formed by removing from B_i those clauses that were never resampled, either because they became satisfied by the resampling of some other clause later in the execution, or because they remained violated (without being resampled) throughout the rest of the execution. In other words, the sequence $B_0^*, B_1^*, \ldots, B_{t-1}^*$, which we will refer to as *the break sequence* of Σ, is a grouping of $W(\Sigma)$ in t sets. The benefit of this grouping is that it allows us to distinguish between a clause $c' \in \Gamma^+(c)$ *being* unsatisfied after a resampling of c vs. the resampling of c having *caused* c' to become unsatisfied. More formally:

Definition 3.6. *For $i \geq 0$, let*

$$\begin{aligned} D_i &= \{c \in B_i \mid \forall k \in [i+1, t+1] : c \in U(\sigma_k) \textit{ and } c \neq r_k\} \\ E_i &= \{c \in B_i \mid \exists j > i \textit{ such that } c \notin U(\sigma_{j+1}) \textit{ and } \forall k \in [i+1, j] : c \neq r_k \wedge c \in U(\sigma_k)\} \\ B_i^* &= B_i \setminus \{D_i \cup E_i\} \ . \end{aligned}$$

If B^* is the *multi*set comprising $B_0^*, \ldots, B_{t-1}^*$ and R is the *multi*set comprising the clauses of the sequence $W(\Sigma)$, then $R = B^*$ and we have a 1-1 correspondence between the elements of $W(\Sigma)$ and the elements of $B_0^*, B_1^*, \ldots, B_{t-1}^*$. In particular, for every $i \in [t]$, the i-th resampled clause r_i is the lowest indexed clause according to π_i in

$$\bigcup_{j=0}^{i-1} B_j^* - \bigcup_{j=1}^{i-1} r_j \ . \tag{3.8}$$

Observe that (3.8) implies that given $B_0^*, B_1^*, \ldots, B_{i-1}^*$ we can inductively determine $r_1, \ldots, r_i$.

3.7.1 Moser's Algorithm via Break Sequences

Assume that F is a k-CNF formula and that $\Gamma(c) \leq \Delta$ for every $c \in F$. Consider the algorithm which in every resampling assigns equal probability to the $2^k - 1$ satisfying assignments of the clause c being resampled (and note that such resampling distributions can *not* be had as projections of a product measure over the variables). Since the algorithm "consumes" precisely $\kappa = \log_2(2^k - 1)$ random bits in each and every resamlping we see that if we set $x = \kappa t$ for some integer t, all trajectories in $B(\kappa t)$ have exactly t resamlpings and probability $2^{\kappa t}$. Therefore, the probability that we fail to find a satisfying assignment after t resamplings is $2^{\kappa t}|B(\kappa t)| \leq 2^{n-\kappa t}|\mathrm{BS}(\kappa t)|$, where $\mathrm{BS}(\kappa t)$ is the set of Break sequences that correspond to trajectories in $B(\kappa t)$ (recall that there are at most 2^n possible states σ_{t+1} and that we recover each trajectory in $B(\kappa t)$ from its Break Sequence and its final state).

To bound $|B(\kappa t)|$ we will perform one last 1-1 mapping, organizing each Break sequence $B_0^*, B_1^*, \ldots, B_t^*$ into a structure we call a *Break Forest*. The Break Forest will have precisely one vertex per resampling, labelled by the resampled clause. The set B_0^* will provide the roots of the trees of the forest, while for each $i \geq 1$, the children of the vertex corresponding to the i-th resampling will be labeled with the clauses in B_i^*. For example, if $B_i^* = \emptyset$, then the vertex corresponding to the i-th resampling will be a leaf in the forest. Thus, the Break Forest $\phi = (\Phi, l_\Phi)$ of a bad t-trajectory is a finite rooted forest Φ with exactly t vertices, organized into no more than $m = |F|$ trees, together with a function (labeling) $l_\Phi : V(\Phi) \to \{c_1, c_2, \ldots, c_m\}$. If Σ is a bad trajectory with $W = W(\Sigma) = r_1, r_2, \ldots, r_t$ and Break sequence $B_0^*, B_1^*, \ldots, B_t^*$, then:

- Let $E = B_0^*$. The Break Forest $\phi = \phi(\Sigma)$ consists of $|E|$ trees, the roots of which are labelled by distinct elements of E. (Neither the trees nor the children of each vertex are ordered.)
- If v is the vertex of ϕ corresponding to r_i, then v has $|B_i^*|$ children labelled by the elements of B_i^*.

To bound $|B(\kappa t)|$ we observe that every element of $B(\kappa t)$ has exactly t vertices, at most $|F| = m$ trees, and every vertex has at most Δ children, since the resampling of a clause can not do more damage than break all its neighbors, i.e., trivially $|B_i^*| \leq \Delta$ for all i. Finally, and crucially, if we fix any ordering of the clauses of F, then this ordering induces an ordering of the clauses in each neighborhood $\Gamma(c)$. Thus, if we know the label $c = l_\Phi(v)$ of a vertex v in the Break Forest, to recover the labels of v's children it suffices to know for each child its ordinal in $\Gamma(c)$, i.e., an integer in $\{1, \ldots, \Delta\}$. Specifically, to see that $\phi(\Sigma)$ is indeed a representation of $W(\Sigma)$ observe that we can recover $W(\Sigma)$ from $\phi(\Sigma)$ by the following simple procedure:

- Let E be the union of the labels of the roots of ϕ and let $i = 1$
- While E is not empty repeat:
  - Let r_i be the lowest indexed clause in E according to π_i
  - Let B be the set of labels of the children of the node that corresponds to r_i in ϕ.
  - Let $E \leftarrow (E \setminus \{r_i\}) \cup B$
  - $i \leftarrow i + 1$

Therefore, we are left to count the number of forests satisfying the above constraints. Let $\mathcal{F}(m,t)$ be the set of all rooted forests with at most m trees and exactly t vertices, where neither the trees, nor the children of each vertex are ordered. Let $\mathcal{L}(m,t)$ be the set of all vertex-labeled rooted forests that result by labeling the vertices of each forest in $\mathcal{F}(m,t)$ in every possible way subject to the requirement that each root vertex carries a distinct label from $\{1,\ldots,m\}$ and that the children of each vertex carry distinct labels from $\{1,2,\ldots,\Delta\}$. It is known (but non-trivial), see for example [12], that

$$|\mathcal{L}(m,t)| = \frac{m}{m+\Delta t}\binom{m+\Delta t}{m} \ . \tag{3.9}$$

From (3.9), Stirling's approximation yields $\log_2 |\mathcal{L}(m,t)| < (m+\Delta t)h(1/\Delta)$, where h is the binary entropy, i.e., $h(x) = -x\log_2 x - (1-x)\log_2(1-x)$. Thus, the binary logarithm of the probability that the algorithm has not encountered any satisfying assignment after t resamplings, is at most

$$-kt + n + \log_2 |B(kt)| < -kt + (m+\Delta t)h(1/\Delta) = mh(1/\Delta) - (k - \Delta h(1/\Delta))t \ . \tag{3.10}$$

From this we see that if $\lambda := k - \Delta h(1/\Delta) > 0$, then we can make the probability of failure arbitrarily small by making t sufficiently large, i.e., without needing to perform multiple runs of the algorithm.

For all $\Delta \geq 2$, requiring $\lambda > 0$ is equivalent to $\frac{\Delta^\Delta}{(\Delta-1)^{\Delta-1}} < 2^k$, matching the condition in [12]. In particular, $\lambda > 0$ for all $\Delta < 2^k/e$. At the same time, $h(1/\Delta) \leq 1$ implying that if $t = M + s/\lambda$, where $M = \lambda^{-1}(h(1/\Delta)m + n)$, then the probability of failure is at most 2^{-s}. Thus, we see that after the critical amount of randomness M is consumed, the probability that the algorithm fails to find a satisfying assignment drops geometrically *per resampling*, as in a cutoff phenomenon[5]. This is in contrast to Theorem 3.3 where the drop was geometric *per run*, i.e., per consumption of $\Omega(m+n)$ bits).

3.7.2 Local Algorithm Design

It is important to note that the analysis up to this point is *tight*.

This is because the rate $x^{-1}\log_2|H[P]|$ is always bounded by 1, even if F is unsatisfiable. Therefore, the existence of $\lambda = \lambda(n,m) < 1$ such that $H[P] \leq \lambda x + M$, *characterizes* satisfiability. This suggests that finding optimal resampling probability distributions, i.e., minimizing $H[P]$, is a task we can not hope to resolve fully. On the other hand, even just trying to minimize $H[P]$ yields a quantitative tool in the search for resampling probability distributions. This is because minimizing $H[P]$ reduces to minimizing the *entropy* of the probability distribution on Break Sequences, enabling a telescoping comparison of x and $H[P]$ over the resamplings.

Concretely, to minimize the entropy of the distribution on Break Forests, we note that in any Break Sequence the children of a vertex labelled by clause c can only carry labels from $\mathrm{Br}(c)$, i.e the clauses that can be broke by c . Clearly, a necessary condition for $c' \in \mathrm{Br}(c)$ is that c and c' contain a complementary pair of literals, i.e., the condition of the lopsided LLL [18]. But this is hardly sufficient. It must also be that:

1. There is at least one state σ^* in which c is violated and c' is satisfied.
2. The clause selected for resampling in state σ^* is c.
3. The resampling distribution $R_c(\sigma^*)$ assigns positive probability to a value assignment that breaks c'.

It is in the three conditions above, and more importantly their interactions, that we will seek advantage.

In fact, we will not have much to offer in the way of controlling the occurrence of 2 above. For example, if we select a uniformly random violated clause in each step, as many successful heuristics do, it is bound to happen (but see the discussion in Section 3.11 as to why choosing a random clause choice is a good idea). This weakness in clause selection is actually a key point of the entropic method[1], as discussed in Section 3.12.

Regarding condition 1 we note that while it is rather trivial in and of itself, i.e., when considering only one clause $c' \in \Gamma(c)$, it becomes far more interesting when one considers the *set* of clauses that may be broken when resampling a clause c. In particular, in the presence of dense local neighborhoods, one can show that not all possible breaking patterns are possible, thus reducing the entropy of the random sets B_i^* (and thus the rate of growth of $|H[P]|$).

Most of our gain will come from condition 3. Rather than merely considering whether the probability of c breaking c' is positive or not, we will chose the resampling probability distribution by weighing the increase in

[1]In famous words: It's not a bug. It's a feature.

the distribution's entropy afforded by allowing the breaking of c' vs. the increase that this breaking causes to $H[P]$. Specifically, we will craft the resampling probability distributions to minimize the (per resampling) rate of change of $x^{-1}H[P]$. In that sense, the algorithm that results is simply performing steepest descent, except that rather than operating on the obvious (energetic) potential, i.e., the number of violated clauses, it operates on an entropic potential that takes into account some of the interactions between the unsatisfied clauses (not just their number). This notion of an entropic potential is at the heart of the entropic method. One can see Moser's original argument as implicitly constructing a first such potential, based on bounds on $|\Gamma(c)|$, and Break Forests as a refinement that accounts for the capacity to adapt the resampling distributions to the "local state", i.e., to the values of the variables in $\Gamma(c)$. We believe that significantly more refined potentials may be developed and we hope that our work serves as the starting point for such an exploration.

The above approach reduces the problem of choosing good resampling distributions for each clause c to a *constrained source coding* problem, the size of which depends only on $|\Gamma(c)|$, i.e., independent of the size of the formula. If the coding problem associated with every clause in the formula is feasible, then running the algorithm with the probability distributions that correspond to feasible solutions will yield a satisfying assignment after $O(n+m)$ resamplings with positive probability. If, on the other hand, the source coding problems associated with one or more clauses are infeasible (in a sense we make precise), the degree of (in)feasibility of each problem may be used to modify the resampling distributions, potentially making all clauses feasible.

Note that the entire discussion above is *per instance*. That is, given a formula F, we propose that instead of "diving right in", one first studies F for a while to find a good collection of resampling distributions, and only then attacks F by unfolding the algorithm's dynamics. Besides such "pointwise-algorithm-design", the entropic method also allows the derivation of structural conditions, in particular the existence of dense neighborhoods and the sharing of multiple variables between clauses, that imply the existence of satisfying assignments for classes of formulas not satisfying the LLL conditions under uniform resampling.

3.8 Motivating Amenability

For the special case of k-CNF formulas and t uniform among the satisfying assignments resamplings we saw in the previous section that we can map Break sequences to forests and use a non-trivial result to bound the number of possible Break sequences by the exact number of possible forests. Consider, though, the following much more direct way to encode (and thus count) Break sequences. Again we fix any ordering of the clauses of F so that if we know that the i-th resampled clause was r_i, then to recover B_i^* it suffices to know for each element of B_i^* its ordinal in $\Gamma(r_i)$, i.e., an integer in $\{1, \ldots, \Delta\}$. With this in mind, we define the following.

- Since $B_0^* \subseteq \mathcal{A}$, encode B_0^* as a string R in $\{0,1\}^m$.
- Let $S = 1^{|B_1^*|}01^{|B_2^*|}0\ldots$
- Let $E = \gamma_1\gamma_2\ldots$, where each γ_i consists of the $|B_i^*|$ integers in $\{1,\ldots,\Delta\}$ corresponding to the ordinals of the elements in B_i^* in $\Gamma(r_i)$ per the global ordering of the clauses.

To decode R, S, E we start by reading R to get B_0^* and initialize a set $U = B_0^*$. We then consult π_1 and determine r_1 as the lowest indexed element in U. We then read the first block of S to determine $|B_i^*|$ and armed with this information we read the first $|B_i^*|$ characters of E which, along with r_1, allow us to determine B_1. We remove r_1 from U and add to it the elements of B_1^*. Consulting π_2 gives us r_2, etc.

Observe now that we can convert E to a binary string at a cost of $\lceil \log_2 \Delta \rceil$ bits per character, allowing us to bound the total number of bits to represent S and E as follows. If $b = |B_1^*| + |B_2^*| + \cdots$, then $|S| = b + t$, while $|E| = b\lceil \log_2 \Delta \rceil$. Since $b \le t$, we get $|S| + |E| \le (2 + \lceil \log_2 \Delta \rceil)t$. Therefore, if $\lambda := k - \lceil \log_2 \Delta \rceil - 2 > 0$, i.e., $\lceil \Delta \rceil < 2^{k-2}$, the binary logarithm of the probability that the algorithm does not encounter any satisfying assignment after t resamplings, is at most

$$-kt + n + m + (2 + \lceil \log_2 \Delta \rceil)t = n + m - \lambda t \ . \tag{3.11}$$

Thus, for any $s > 0$, the probability of failure after $(n+m)/k + s/\lambda$ resamplings is at most 2^{-s}.

The (rather small) losses relative to the sharp result of the previous section stem from the fact that we encoded the sequence of the numbers $|B_i^*|$ separately from the content of the sets B_i^*. In particular, observe that we count all possible permutations of the elements of each block in E (rather than, say, the unique permutation corresponding to the increasing order). Moreover, we get $\lceil \log_2 \Delta \rceil$ because we use a "block" code to represent the elements of each set B_i^*. At the same time, though, this separation will be key in allowing us to deal with state-dependent resampling distributions as we show next.

3.9 Proof of Theorem 3.3

Let I be the random variable equal to an infinite trajectory of the algorithm. For any fixed $x > 0$, we define the following random variables.

- Let Σ be the prefix of I in $L(x)$.
- Let $P = P(\Sigma)$ be the maximal bad prefix of Σ.
- Let σ be the final state of P.
- Let Z be the number of resamplings in P.
- For $i \in [Z]$, let r_i be the clause that was resampled in the i-th resampling. For $i > Z$, let $r_i = \emptyset$.
- Let $Y = Y(P) = B_0^*, B_1^*, \ldots,$ be the Break Sequence of P, where $B_j^* = \emptyset$, for $j \geq Z$.
- Let $U = |B_1^*|, |B_2^*|, \ldots$

Recall that there is a bijection between P and the pair Y, σ implying

$$H[P] = H[Y, \sigma] \leq H[Y] + H[\sigma] \leq H[Y] + n \ . \tag{3.12}$$

To bound $H[Y]$ we decompose it as follows.

$$\begin{aligned} H[Y] &= H[Y, B_0^*, U] \\ &= H[B_0^*, U] + H[Y \mid B_0^*, U] \\ &= H[B_0^*, U] + H[B_1^*, B_2^*, \ldots \mid B_0^*, U] \\ &= H[B_0^*, U] + \sum_i H[B_i^* \mid B_0^*, B_1^*, B_2^*, \ldots, B_{i-1}^*, U] \ . \end{aligned} \tag{3.13}$$

In the rest of this section, all subscript indices i are to be read as $i \geq 1$. Theorem 3.3 follows readily from the following lemmata.

Lemma 3.1. $H[B_0^*, U] \leq m + 2\mathbb{E}Z - \mathbb{E}|B_0^*|$.

Lemma 3.2. *If every clause is ϵ-amenable, then for every i,*

$$H[B_i^* \mid B_0^*, B_1^*, \ldots, B_{i-1}^*, U] \leq (1-\epsilon)\left(\mathbb{E}[\mathrm{Self}(r_i) + \mathrm{Parent}(B_i^*)]\right) - 2\mathbb{E}|B_i^*| \ .$$

Lemma 3.3. $\sum_i \mathbb{E}[\mathrm{Self}(r_i) + \mathrm{Parent}(B_i^*)] \leq (x + \beta)$ *and* $\sum_i \mathbb{E}|B_i^*| = \mathbb{E}Z - \mathbb{E}|B_0^*|$.

Proof of Theorem 3.3. Combining (3.12) and (3.13) with the bounds from Lemmata 3.1, 3.2, and 3.3 and using the fact $\mathbb{E}|B_0^*| \leq m$ proves the theorem. □

We prove Lemmata 3.1 and 3.3 here. The proof of Lemma 3.2 constitutes Section 3.10.

Proof of Lemma 3.1. We will represent B_0^*, U as a binary string s of length $m+2Z-|B_0^*|$. Since $B_0^* \subseteq \mathcal{A}$ the first m bits of s are the characteristic vector of B_0^*. We encode U immediately afterwards, representing the i-th element of U, for each $i \in [Z]$, as $1^{|B_i^*|}0$. Decoding, other than termination, is trivial: after reading the first m bits of s, the rest of the string is interpreted in blocks of the form 1^*0. To determine termination we note that, by construction, $|B_0^*| + \sum_{i=1}^{j} |B_i^*| - j \geq 0$ for every $j \in [Z]$ with equality holding only for $j = Z$. Therefore, decoding stops as soon as equality holds for the first time. The representation of U in this manner consists of $\sum_i |B_i^*|$ ones and Z zeroes, i.e., of $2Z - B_0^*$ bits, since $\sum_i |B_i^*| = Z - |B_0^*|$. □

Proof of Lemma 3.3. For $\sum_i \mathbb{E}|B_i^*|$ the claim follows readily from the fact $\sum_i |B_i^*| = Z - |B_0^*|$.

Recall that r_i is the i-th resampled clause and let $\mathrm{Pot}(P) = \sum_i \mathrm{Pot}(r_i)$, where $\mathrm{Pot}(\emptyset) = 0$. Since there is 1-to-1 correspondence between the resampled clauses in P and the clauses in the sets $B_0^*, B_1^*, \ldots$, we get $\mathrm{Pot}(P) = \mathrm{Pot}(B_0^*) + \sum_i \mathrm{Pot}(B_i^*)$ and, therefore,

$$\begin{aligned}
\sum_i \mathbb{E}\mathrm{Pot}(r_i) &= \mathbb{E}\mathrm{Pot}(B_0^*) + \sum_i \mathbb{E}\mathrm{Pot}(B_i^*) \\
&\geq \sum_i \mathbb{E}\mathrm{Self}(B_i^*) + \sum_i \mathbb{E}[\mathrm{Pot}(B_i^*) - \mathrm{Self}(B_i^*)] \\
&= \sum_i \mathbb{E}[\mathrm{Self}(r_i) + \mathrm{Pot}(B_i^*) - \mathrm{Self}(B_i^*)] \\
&= \sum_i \mathbb{E}[\mathrm{Self}(r_i) + \mathrm{Parent}(B_i^*)] \qquad (3.14)
\end{aligned}$$

Recall now that for each i, state σ_{i+1} is reached from state σ_i by selecting a clause c deterministically as a function of π_i and σ_i and resampling c according to a probability distribution that depends only on σ_i. Therefore, if c_i is the random variable that equals the clause selected for resampling at state σ_i, then

$$\begin{aligned}
H[\Sigma] &= \sum_i H[\sigma_{i+1} \mid \sigma_i] \\
&= \sum_i \sum_{\sigma\in\{0,1\}^n} \Pr[\sigma_i = \sigma] \cdot H[\sigma_{i+1} \mid \sigma_i = \sigma] \\
&= \sum_i \sum_{c\in F} \Pr[c_i = c] \sum_{v\in\mathcal{V}(c)} \Pr[v(c) = v \mid c_i = c] \cdot H[R_c(v)] \\
&\geq \sum_i \sum_{c\in F} \Pr[c_i = c] \cdot \min_{v\in\mathcal{V}(c)} H[R(c,v)] \\
&= \sum_i \sum_{c\in F} \Pr[c_i = c] \cdot \mathrm{Pot}(c) \\
&\geq \sum_i \sum_{c\in F} \Pr[r_i = c] \cdot \mathrm{Pot}(c) \\
&= \sum_i \mathbb{E}\mathrm{Pot}(r_i) \ . \qquad (3.15)
\end{aligned}$$

Combining (3.14) and (3.15) with the fact $H[\Sigma] \leq x + \beta$ proves the lemma. $\square$

3.10 Proof of Lemma 3.2

Recall that the random variables $B_0^*, B_1^*, \ldots, B_{i-1}^*$ determine the random variables r_i, while U determines $|B_i^*|$. Therefore,

$$H[B_i^* \mid B_0^*, B_1^*, \ldots, B_{i-1}^*, U] \leq H[B_i^* \mid r_i, |B_i^*|] \ .$$

To bound $H[B_i^* \mid r_i, |B_i^*|]$ we define the following probability distributions.

- For each $i \geq 1$, let $\mathcal{S}_i : \mathcal{A} \cup \emptyset \to [0,1]$ be the probability distribution of r_i.
- For each $i \geq 1$ and clause $c \in \mathcal{A}$, let $\mathcal{P}_i^c : \mathcal{V}(c) \to [0,1]$ be the probability distribution of c-critical vectors conditional $r_i = c$.
- For each clause $c \in \mathcal{A}$ and $v \in \mathcal{V}(c)$, let $\mathcal{B}_v^c : \mathrm{Br}(v) \to [0,1]$ be the probability distribution of B_i conditional on $r_i = c$ and the c-critical vector being v.
- For each $i \geq 1$, clause $c \in \mathcal{A}$, critical vector $v \in \mathcal{V}(c)$, and set $B \in \mathrm{Br}(v)$, let $\mathcal{L}_{i,v}^{c,B} : \{0, \ldots, |B|\} \to [0,1]$ be the probability distribution of $|B_i^*|$ conditional on $r_i = c$, the c-critical vector being v, and $B_i = B$.

Let $s_i = |B_i^*|$ to lighten notation. Then,

$$\begin{aligned}
H[B_i^* \mid r_i, |B_i^*|] &= H[B_i^* \mid r_i, s_i] \\
&= \sum_{c \in F} \sum_j \Pr[r_i = c \wedge s_i = j] \cdot H[B_i^* \mid r_i = c \wedge s_i = j] \\
&= \sum_{c \in F} \Pr[r_i = c] \sum_j \Pr[s_i = j \mid r_i = c] \cdot H[B_i^* \mid r_i = c \wedge s_i = j] \\
&= \sum_{c \in F} \mathcal{S}_i(c) \sum_{v \in \mathcal{V}(c)} \mathcal{P}_i^c(v) \sum_{B \in \mathrm{Br}(v)} \mathcal{B}_v^c(B) \sum_j \mathcal{L}_{i,v}^{c,B}(j) \cdot H[B_i^* \mid r_i = c \wedge s_i = j] \ .
\end{aligned}$$

Recall now that the unit-experiment defined in Section 3.5 for a fixed clause c has outcome B^* and let $\mathcal{L}_v^B(j) : \mathbb{N} \to [0,1]$ be the probability distribution of $|B^*|$ conditional on the selections in Steps 1, 2 being v, B, respectively. Define also

$$\begin{aligned}
Q(c,\mathcal{P},f) &= \sum_{v \in \mathcal{V}(c)} \mathcal{P}(v) \sum_{B \in \mathrm{Br}(v)} \mathcal{B}_v^c(B) \sum_j \mathcal{L}_v^B(j) \cdot H[B^* \mid |B^*| = j] \\
W(c,\mathcal{P},f) &= \sum_{v \in \mathcal{V}(c)} \mathcal{P}(v) \sum_{B \in \mathrm{Br}(v)} \mathcal{B}_v^c(B) \cdot \sum_{S \subseteq B} f_v^B(S) \cdot [(1-\epsilon)(\mathrm{Self}(c) + \mathbb{E}[\mathrm{Parent}(B^*)] - 2\mathbb{E}[|B^*|]) \ .
\end{aligned}$$

Assume now that we select a clause c according to $\mathcal{S}_i(c)$ and perform the unit-experiment of clause c with $\mathcal{P} = \mathcal{P}_i^c$. Clearly, we can couple the unit-experiment with the algorithm so that $B = B_i$ always. Moreover, if we define the function f of the unit-experiment by setting f_v^B to the probability distribution of B_i^* conditional on $r_i = c$, the c-critical vector being v, and $B_i = B$, then we can further couple the experiment with the algorithm so that $B^* = B_i^*$ always. In such a case,

$$H[B_i^* \mid r_i, |B_i^*|] = \sum_{c \in F} \mathcal{S}_i(c) \sum_{v \in \mathcal{V}(c)} \mathcal{P}_i^c(v) \sum_{B \in \mathrm{Br}(v)} \mathcal{B}_v^c(B) \sum_j \mathcal{L}_v^B(j) \cdot H[B^* \mid |B^*| = j] \ .$$

Observe that for any fixed $c, \mathcal{P}, f$, if B^* is the output of the unit-experiment, then

$$\begin{aligned}
Q(c,\mathcal{P},f) &= H[B^* \mid |B^*|] && (3.16) \\
W(c,\mathcal{P},f) &= (1-\epsilon)(\mathrm{Self}(c) + \mathbb{E}[\mathrm{Parent}(B^*)]) - 2\mathbb{E}[|B^*|] \ . && (3.17)
\end{aligned}$$

Since every clause is ϵ-amenable, $Q(c,\mathcal{P},f) \le W(c,\mathcal{P},f)$ for every $c, \mathcal{P}, f$ and, therefore, in particular for $\mathcal{P} = \mathcal{P}_i^c$ and the function f designed above to couple B^* with B_i^*. As a result,

$$\begin{aligned}
H[B_i^* \mid r_i, |B_i^*|] &= \sum_{c \in F} \mathcal{S}_i(c)\, Q(c, \mathcal{P}_i^c, f) \\
&\le \sum_{c \in F} \mathcal{S}_i(c)\, W(c, \mathcal{P}, f) \\
&= (1-\epsilon)(\mathbb{E}[\mathrm{Self}(r_i)] + \mathbb{E}[\mathrm{Parent}(B_i^*)]) - 2\mathbb{E}|B_i^*| && (3.18)
\end{aligned}$$

3.11 Discussion

As mentioned, a key point of the entropic method is that since we don't have control over the state evolution we are limited in our choice of which clause to resample in each step, as discussed in Section 3.12. This meant that our analysis had to ensure that the clause resampled in *every* step is "entropically gainful", i.e., that it consumes more randomness than the increase in the entropy of the Break Forest. Meaning that if not all clauses are amenable, if we reach a state in which there is a single non-amenable violated clause the analysis fails. Note though that if we select a random violated clause to resample in each step, a choice allowed in our setting, then it is actually enough that there is entropic gain *on average*, i.e., with respect to the random clause choice. In that sense, selecting a random violated clause appears like a reasonable choice, a notion compatible with the fact that such a choice works very well in practice.

3.12 Clause Choice: Bug or Feature?

Our capacity to compress witness into Break sequences relied crucially on the nature of the method we used for selecting which clause to resample in each step. Specifically, the method (fixed permutations) was such that it was possible to reconstruct which clause was selected for resampling in a step without knowledge of the exact underlying state σ, or even of the set $U(\sigma)$ in its entirety. Some such "insensitivity" of the clause-selection function with respect to the underlying state seems an essential element of the entropic approach. Recall that Moser's fundamental observation was that every bad trajectory $\Sigma = \sigma_1, \ldots, \sigma_t$ can be recovered from $\langle W(\Sigma), \sigma_t \rangle$. We showed that one can bound the number of witnesses sequences in terms of a quantity more refined than $\Delta = \max_c |\Gamma^+(c)|$ by mapping them injectively into Break Sequences while, like Moser, we trivially bounded the entropy of σ_t by n.

The aggregation of multiple bad trajectories (paths of the computation tree) into a single fiber (either based on having the same Witness sequence, or on having the same Break Sequence) amounts to giving up all control over state evolution within each fiber. Yet, for the argument to work the transformation

$$\text{Witness sequence} \rightarrow \text{Break sequence}$$

must remain losslesseral (injective). This was only true because of the nature of the clause-selection method employed, i.e., its relative insensitivity to the underlying state. Overcoming this restriction, so that the choice of which clause to resample in each step depends on the state in a meaningful way, appears to us to require genuinely new ideas.

Bibliography

[1] Noga Alon. A parallel algorithmic version of the local lemma. In *32nd Annual Symposium on Foundations of Computer Science (San Juan, PR, 1991)*, pages 586–593. IEEE Comput. Soc. Press, Los Alamitos, CA, 1991.

[2] Adrian Balint and Uwe Schöning. Choosing probability distributions for stochastic local search and the role of make versus break. In Alessandro Cimatti and Roberto Sebastiani, editors, *SAT*, volume 7317 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2012.

[3] József Beck. An algorithmic approach to the Lovász local lemma. I. *Random Structures Algorithms*, 2(4):343–365, 1991.

[4] Artur Czumaj and Christian Scheideler. Coloring non-uniform hypergraphs: a new algorithmic approach to the general Lovász local lemma. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (San Francisco, CA, 2000)*, pages 30–39, New York, 2000. ACM.

[5] P Diaconis. The cutoff phenomenon in finite markov chains. *Proceedings of the National Academy of Sciences*, 93(4):1659–1664, 1996.

[6] P. Erdős and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In *Infinite and finite sets (Colloq., Keszthely, 1973; dedicated to P. Erdős on his 60th birthday), Vol. II*, pages 609–627. Colloq. Math. Soc. János Bolyai, Vol. 10. North-Holland, Amsterdam, 1975.

[7] Heidi Gebauer, Tibor Szabó, and Gábor Tardos. The local lemma is tight for sat. In Dana Randall, editor, *SODA*, pages 664–674. SIAM, 2011.

[8] Bernhard Haeupler, Barna Saha, and Aravind Srinivasan. New constructive aspects of the Lovász local lemma. *J. ACM*, 58(6):Art. 28, 28, 2011.

[9] David G. Harris and Aravind Srinivasan. Constraint satisfaction, packet routing, and the lovasz local lemma. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *STOC*, pages 685–694. ACM, 2013.

[10] Kashyap Babu Rao Kolipaka and Mario Szegedy. Moser and tardos meet lovász. In Lance Fortnow and Salil P. Vadhan, editors, *STOC*, pages 235–244. ACM, 2011.

[11] Kashyap Babu Rao Kolipaka, Mario Szegedy, and Yixin Xu. A sharper local lemma with improved applications. In Anupam Gupta, Klaus Jansen, José D. P. Rolim, and Rocco A. Servedio, editors, *APPROX-RANDOM*, volume 7408 of *Lecture Notes in Computer Science*, pages 603–614. Springer, 2012.

[12] Jochen Messner and Thomas Thierauf. A kolmogorov complexity proof of the lovász local lemma for satisfiability. *Theor. Comput. Sci.*, 461:55–64, 2012.

[13] Michael Molloy and Bruce Reed. Further algorithmic aspects of the local lemma. In *STOC '98 (Dallas, TX)*, pages 524–529. ACM, New York, 1999.

[14] Robin A. Moser. A constructive proof of the Lovász local lemma. In *STOC'09—Proceedings of the 2009 ACM International Symposium on Theory of Computing*, pages 343–350. ACM, New York, 2009.

[15] Robin A. Moser and Gábor Tardos. A constructive proof of the general Lovász local lemma. *J. ACM*, 57(2):Art. 11, 15, 2010.

[16] Joel Spencer. Needles in exponential haystacks ii.

[17] Aravind Srinivasan. Improved algorithmic versions of the Lovász local lemma. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 611–620, New York, 2008. ACM.

[18] Mario Szegedy. The lovász local lemma - a survey. In Andrei A. Bulatov and Arseny M. Shur, editors, *CSR*, volume 7913 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2013.

[19] R.Graham, D.Knuth, and O.Patashnik. *Concrete Mathematics.* Addison-Wesley, 2nd edition, 1994

[20] N. Alon, J.Spencer, and P. Erd́os. *The probabilistic method.* Wiley, 1992

[21] J.B.Shearer. On a problem of Spencer. *Combinatorica*, 5(3):241-245, 1985

[22] Wesley Pegden. Highly nonrepetitive sequences: Winning strategies from the local lemma. Random Struct. Algorithms, 38(1-2):140–161, 2011.

[23] Wesley Pegden. The lefthanded local lemma characterizes chordal depen- dency graphs. Random Struct. Algorithms, 41(4):546–556, 2012.

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

[25] Rodrigo Bissacot, Roberto Fernandez, Aldo Procacci, and Benedetto Scop- pola. Combinatorics Probability and Computing, 20(5):709–719, 2011.

[26] Themistoklis Gouleakis, Algorithmic aspects of the Lovász Local Lemma, Unpublished undergraduate thesis for the National Technical University of Athens

[27] Acyclic edge-coloring using entropy compression. European Journal of Combinatorics 34-6 pp. 1019-1027 (2013).