Entropy compression: algorithmic Lovász Local Lemma and graph coloring

Anqi Li

1 Introduction and overview

In this article, we introduce the method of entropic compression, which first arose in Moser's [Mos09] constructive proof of the Lovász Local Lemma (LLL). The entropic compression method has found many applications elsewhere, of which there are too many to enumerate. We focus on the constructive algorithmic (LLL) and graph list coloring application in this article.

In both of our applications, we will describe a recursive "modified rejection sampling" algorithm which upon termination produces a solution to our problem. The hard part is demonstrating that the algorithm terminates in a short amount of time and this is where the entropic compression argument comes in. Vaguely speaking, the entropic compression argument (a term coined by Terry Tao in his blogpost [Tao10] on Moser's algorithmic LLL) is a sort of information theoretic monotonicity property to guarantee that a recursive procedure terminates in a finite amount of time. I cannot summarize the gist of the argument better than Terry Tao, so here is a quote from his blog that summarizes how the entropic compression argument works.

> [The entropic compression argument] applies to probabilistic algorithms which require a certain collection R of random "bits" as part of the input, thus each loop of the algorithm takes an object A(which may also have been generated randomly) and some portion of the random string R to (deterministically) create a better object A' (and a shorter random string R', formed by throwing away those bits of R that were used in the loop). The key point is to design the algorithm to be partially reversible, in the sense that given A' and R' and some additional data H' that logs the cumulative history of the algorithm up to this point, one can reconstruct A together with the remaining portion R not already contained in R'. Thus, each stage of the argument compresses the information-theoretic content of the string $A+R$ into the string $A+R'+H'$ in a lossless fashion. However, a random variable such as $A+R$ cannot be compressed losslessly into a string of expected size smaller than the Shannon entropy of that variable. Thus, if one has a good lower bound on the entropy of $A+R$, and if the length of $A'+R'+H'$ is significantly less than that of $A+R$ (i.e. we need the marginal growth in the length of the history file H' per iteration to be less than the marginal amount of randomness used per iteration), then there is a limit as to how many times the algorithm can be run, much as there is a limit as to how many times a random data file can be compressed before no further length reduction occurs.

This entropic compression argument has also come up in other combinatorial applications such as graph list coloring. In the problem of list coloring, each vertex of a graph G is assigned a list of allowable colors. We want to choose a color for each vertex out of the assigned list so that no two adjacent vertices are assigned the same color.

Definition 1. We say that a graph G is *k-choosable* if it has a proper coloring no matter how one assigns a list of K colors to each vertex.

Molloy [Mol19] used the method entropic compression to establish the following theorem.

Theorem 1 ([Mol19])**.** If G is a triangle-free graph of maximum degree Δ, then G is $(1 + o(1))\frac{\Delta}{\log \Delta}$-choosable.

We will use the a simplified version of the entropic compression in [Mol19] to establish the following theorem.

Theorem 2. There exists k such that the following is true for all $\Delta \geq k$: that if G is a triangle-free graph with maximum degree Δ, then G is $\left\lceil \frac{3\Delta}{\log \Delta} \right\rceil$-choosable.

Outline. Section 2 is about the algorithmic Lovász Local Lemma. In Subsection 2.1 we motivate the Lovász Local Lemma and introduce the k-SAT problem. We specialize to the k-SAT problem in Subsection 2.2, motivate Moser's algorithmic approach to the problem and introduce the compression/decoder framework (Lemma 2) which forms the crux of the analysis of termination of Moser's algorithm. In Section 3, we pivot topics and discuss how to port the framework from Subsection 2.2 to the context of graph list coloring problems.

Acknowledgements. This paper was written as part of MIT's 18.424 (Seminar in Information Theory). We would like to express gratitude towards Prof. Jon Kelner for his suggestion of the topic and for his helpful feedback. We would also like to thank Prof. Kuikui Liu for his valuable comments and suggestions on an earlier draft of this paper.

2 Algorithmic Lovász Local Lemma

In this section, we study one application of the entropy compression technique to give an algorithmic version of the Lovász Local Lemma (LLL). The LLL is an important probabilistic method pioneered by Lovász and Erdős [EL75] to identify certain combinatorial structures (one of its first applications was in the context of Ramsey theory; for a plethora of applications of the LLL we refer the interested reader to [AS16]). In the next section, we will first give a overview to the statement of LLL, before delving into Moser's [Mos09] groundbreaking algorithmic approach to the LLL.

2.1 What is the Lovász Local Lemma?

The Lovász Local Lemma is an important technique in the probabilistic method toolbox. In general, we are trying to find an object in a search space Ω that avoids some bad (combinatorial) properties. To leverage the probabilistic method, we will sample an element of Ω according to some probability measure and define a number of bad events that correspond to sampling elements with those bad properties. Suppose each of these bad events happens with probability at most 1. Observe that if we can show that with positive probability we can avoid all of the bad events then it must follows that there exists an element in Ω that avoids all the bad properties. Now, if these m bad events $\{\mathcal{B}_i\}_{i=1}^m$ were all independent then such a scheme works fine; indeed, suppose $\mathbb{P}[\mathcal{B}_i] = 1 - b_i$ for $1 \leq i \leq m$. Then we can conclude that

$$\mathbb{P}\left[\bigwedge_{i=1}^m \neg\mathcal{B}_i\right] = \prod_{i=1}^m \mathbb{P}[\neg\mathcal{B}_i] = \prod_{i=1}^m (1 - b_i) > 0. \tag{1}$$

However, in reality, the condition of wanting $\mathcal{B}_i$ to all be independent is often too strong a condition to ask for; it is usually the case that they are somewhat dependent on each other. At this juncture we introduce an important running example – namely the *k-SAT* problem – for this section of this expository article. We will then define the bad events in this example which we will see exhibits some mild dependence on each other, which is illustrative of the typical kind of set-up that we will encounter when applying the LLL.

Example 2.1. In the *k-SAT problem* we have a formula φ of m clauses on n variables where each clause is the disjunction of k literals, each of which is a variable or its negation and we want to find an assignment of `TRUE/FALSE` to the variables that satisfies the formula i.e. so that φ evaluates as `TRUE`. For example, $\neg x_1 \vee x_2 \vee \neg x_3$ is a potential 3-SAT clause. A 3-SAT instance could be

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$$

and $x_1 = x_2 =$ `FALSE`, $x_3 =$ `TRUE` is one (of many) satisfying assignment.

We begin by making some preliminary observations about the k-SAT problem. First, note that there are 2^k possible variable assignments to each of the distinct k variables constituting a clause. Only one of them fails to satisfy the clause, because every variable has to be assigned incorrectly. And so it is very easy to satisfy a clause: if we randomly assign its variables we succeed with probability $1 - 2^{-k}$. This leads to the following easy observation.

Claim 1. *If a k-SAT formula has $m \leq 2^k$ clauses then it is satisfiable.*

Proof. This follows by the union bound. Specifically, for a randomly chosen assignment the probability that it satisfies all the clauses is at least $1 - m \cdot 2^{-k} > 0$. □

Putting this problem in the framework that we introduced at the start of the section, we can define Ω to be the set of all possible 2^n assignments to the n variables. We define m bad events $\mathcal{B}_i$ that correspond to the event that the ith clause is not satisfied. As we have seen above, over a uniformly random choice of variable assignment we have $\mathbb{P}(\mathcal{B}_i) = 2^{-k}$. However, the argument that we gave at the very beginning of this subsection would not work in this case because the $\mathcal{B}_i$ are not independent! In fact, $\mathcal{B}_i$ and $\mathcal{B}_j$ are not independent when the ith and jth clause share variables. Now, heuristically, if we constrain any two of the clauses in our k-SAT formula to not share many variables, then they would be "largely" independent.

In the proof of Claim 1 we used the union bound which was significantly lossier than (1); indeed using the union bound, we can show that in the general set-up of bad events $\mathcal{B}_i$, we have

$$\mathbb{P}\left[\bigwedge_{i=1}^{n} \neg \mathcal{B}_i\right] \geq 1 - \sum_{i=1}^{n} \mathbb{P}[\mathcal{B}_i] = 1 - \sum_{i=1}^{n} b_i. \tag{2}$$

However, while the RHS of (2) is completely vacuous beyond a certain point, (1) is meaningful in all possible regimes of the b_i. So if we were to impose the constraint that no two clauses in our k-SAT formula shares many variables we could conceivably conclude that the clauses are "close to independent". In this case, we would more likely be "closer" to the regime of applying (1) than (2). In particular, we would hope to show that our formula is satisfiable, even when m is quite large. And this is what the LLL would give us.

We first state a version of Claim 1 in this set-up where we constrain clauses to not be too dependent on each other.

Theorem 1. *If a k-SAT formula has the property that every clause in the formula shares variables with less than 2^{k-2} other clauses then it is satisfiable.*

In particular, note that there is no constraint on m in Theorem 1, and so we have proven satisfiability for aribtrarily long formulas (though subject to the dependency constraint)! The proof of this theorem follows immediately from the LLL, which we state here.

Theorem 2 (Lovász Local Lemma)**.** *Let $\mathcal{B}_1, \ldots, \mathcal{B}_k$ be a sequence of events such that each event occurs with probability at most p and such that each event is independent of all the other events except for at most d of them. If $ep(d+1) < 1$ then then there is a nonzero probability that none of the events occurs.*

Proof of Theorem 1. It follows from Theorem 2 and the inequality $e2^{-k}((2^{k-2}-1)+1)<1$. □

We give an alternate proof that does not blackbox Theorem 2. In particular, the inductive argument in this proof is basically what goes into the proof of Theorem 2.

"Alternate" proof of Theorem 1. In the following proof, the probabilities are all taken with respect to a uniformly random sample of assignments to the variables. Let F be a formula with the local constraints on the variables as in the theorem. Let $G \subset F$ be a union of some clauses, and let C be a clause in $F \setminus G$. If C does not share variables with any clause in G then we know that

$$\mathbb{P}_\sigma[\sigma \text{ does not satisfy } C | \sigma \text{ satisfies } G] = 2^{-k}. \tag{3}$$

Of course, we cannot hope for this to be true in general for all choices of C and G in F. However, we claim that a slightly weaker form of the above actually holds under the conditions of the theorem, namely we claim that

$$\mathbb{P}_\sigma[\sigma \text{ does not satisfy } C | \sigma \text{ satisfies } G] \leq 2^{-k+1}. \tag{4}$$

This is sufficient for our purposes because we can write

$$\begin{aligned}\mathbb{P}_\sigma[\sigma \text{ satisfies } F] &= \mathbb{P}_\sigma[\sigma \text{ satisfies } C_1] \cdot \mathbb{P}[\sigma \text{ satisfies } C_2 | \sigma \text{ satisfies } C_1] \cdot \ldots \cdot \mathbb{P}[\sigma \text{ satisfies } C_m | \sigma \text{ satisfies } C_1, \ldots, C_{m-1}] \\ &\geq \left(1-2^{-k+1}\right)^m > 0.\end{aligned}$$

In the remainder of the proof we establish (4). To that end, we induct on the number of clauses in G. Let G' be the set of clauses in G that share variables with C. Let $\widetilde{G} = G \setminus G'$. We may assume that $\widetilde{G}$ has less clauses than G because of (3). By the induction hypothesis, for any $D \in G'$, we have that

$$\frac{\mathbb{P}_\sigma[\sigma \text{ does not satisfy } D, \sigma \text{ satisfies } \widetilde{G}]}{\mathbb{P}_\sigma[\sigma \text{ satisfies } \widetilde{G}]} \leq 2^{-k+1}. \tag{5}$$

This implies that

$$\begin{aligned}\mathbb{P}_\sigma[\sigma \text{ satisfies } G] &\geq \mathbb{P}_\sigma[\sigma \text{ satisfies } G] - \mathbb{P}_\sigma[\sigma \text{ does not satisfy } G', \sigma \text{ satisfies } \widetilde{G}] \\ &\geq_{(5)} \mathbb{P}_\sigma[\sigma \text{ satisfies } \widetilde{G}] - \frac{|G'|}{2^{k-1}} \mathbb{P}_\sigma[\sigma \text{ satisfies } \widetilde{G}] \\ &\geq \frac{\mathbb{P}_\sigma[\sigma \text{ satisfies } \widetilde{G}]}{2}\end{aligned}$$

where we used the union bound in the penultimate inequality and the locality condition in the last inequality. This combined with (3) allows us to write

$$\frac{\mathbb{P}_\sigma[\sigma \text{ does not satisfy } C, \sigma \text{ satisfies } G]}{\mathbb{P}_\sigma[\sigma \text{ satisfies } G]} \leq \frac{\mathbb{P}_\sigma[\sigma \text{ does not satisfy } C, \sigma \text{ satisfies } \widetilde{G}]}{\mathbb{P}_\sigma[\sigma \text{ satisfies } \widetilde{G}]/2} \leq 2^{-k+1}$$

as desired. □

While Theorem 1 has the very appealing consequence that demonstrates the *existence* of a satisfiable formula, the proofs we have seen above does not give an efficient algorithm for finding a satisfying assignment under the premises of the theorem. The goal of the next subsection is to demonstrate such an algorithmic procedure.

2.2 Algorithmic LLL in the context of k-SAT

The goal of this section is to prove the following theorem, due to Moser [Mos09].

Theorem 3. *If a k-SAT formulas has the property that every clause in the formula shares variables with less than 2^{k-3} other clauses then a satisfying assignment can be found in expected polynomial time via a randomized algorithm.*

Remark 1. *There exists a derandomization of this algorithm; we refer the interested reader to [CGH10].*

The most naïve algorithm would be to just repeatedly sample uniformly random assignments to the literals. We can easily check if an assignment satisfies the formula. So we repeat this sampling until we find one that actually satisfies the formula. The following proposition shows that such a naïve rejection sampling algorithm will not terminate in finite time; effectively this kind of naïve algorithm amounts to finding a needle in a haystack.

Proposition 1. *There exists a family of arbitrarily large k-SAT formulas F with the property that every clause in the formula shares variables with less than 2^{k-3} other clauses and such that these formulas are satisfied with exponentially small probability in the size of F when sampling variable assignments uniformly at random.*

Proof. Consider clauses of the form $(\underbrace{x \vee x \vee \cdots \vee x}_{k \text{ times}})$. We select m distinct clauses of this form from the variables set $\{x_1, \ldots, x_n\}$. Then the probability that a uniformly random assignment to the variables satisfies the formula thus constructed is 2^{-m}. □

The next most natural next step to try is to somehow try to "fix" our initial starting random assignment somehow instead of just doing rejection sampling; this is especially since it appears the kind of clauses that make Proposition 1 fail are "semi-random" and have a lot of structure that rejection sampling is not accounting for. Specifically, we may attempt something like this: try a random assignment and then so long as there is a clause that is not satisfied, we can try a new random assignment for the variables of that clause. In the case when there are few dependencies among the clauses, it turns out that such a scheme actually works.

To describe the algorithm, we assume that the clauses of F are ordered. The recursive procedure `fix`(C) attempts to fix a clause that is not satisfied and then invoke itself recursively.

```
1  Moser's algorithmic LLL ;
   Input  : a k-SAT formula F obeying constraints of Theorem 3
   Output: a satisfying assignment
2  Initialize by choosing a random assignment to the variables;
3  while there is an unsatisfied clause C, choose the lexicographically first do
4  |  Fix(C);
5  end
6  Function Fix(C):
7  |  Resample the variables of C uniformly at random;
8  |  while there is an unsatisfied clause C' sharing variables with C, choose the lexicographically first do
9  |  |  Fix(C');
10 |  end
```

It may be somewhat surprising at first glance that such a scheme terminates, and in fact we will show that it terminates in time polynomial in the size of the formula.

Lemma 1. *Each clause of the k-SAT formula appears at most once as an unsatisfied clause in the outer loop of the above algorithm.*

Proof. For any variable assignment, after calling `Fix`(C), a clause that was satisfied before this function call stays satisfied after the call. In other words, upon each implementation of the outer loop, we make a genuine improvement since we reduce the number of violated clauses by 1: after terminating `Fix`(C), C would be satisfied as well. □

Suppose F has m clauses. The lemma shows that the outer loop (line 3) is executed at most m times. It remains to prove that the total number of recursive calls to `Fix` is not too large.

The key idea for this step is the following slogan that we have seen information theory:

> 💡 Uniformly random data cannot be efficiently compressed.

We formalize this intuition as follows. Let $r < s$ be two integers. A *compression function* is a function $\mathcal{C}\colon \{0,1\}^s \to \{0,1\}^r \cup \{\perp\}$ that outputs a shorter binary string or failure symbol $\perp$. A *decoder function* $\mathcal{D}\colon \{0,1\}^r \to \{0,1\}^s$ takes a compressed binary string and decodes the longer string.

Definition 2. We say that a pair of compression and decoder functions satisfy *soundness* if for all $\mathbf{b} \in \{0,1\}^s$ such that $\mathcal{C}(\mathbf{b}) \neq \perp$, we have that

$$\mathcal{D}(\mathcal{C}(\mathbf{b})) = \mathbf{b}.$$

Remark 2. *In practice, we would want properties like $\mathcal{C}$ and $\mathcal{D}$ to run in polynomial time, but we omit these considerations for now.*

We can formalize our key intuition above (💡) using the language of compression functions and decoder functions. Colloquially, the following lemma states that we cannot have sound compression on uniformly random data for a compression function that does not fail.

Lemma 2. *Let $r < s$ be two integers, then if $\mathcal{C}\colon \{0,1\}^s \to \{0,1\}^r \cup \{\perp\}$ and $\mathcal{D}\colon \{0,1\}^r \to \{0,1\}^s$ satisfy soundness, we must have that*

$$\mathbb{P}[\mathcal{C}(\mathbf{b}) = \perp] \geq 1 - 2^{r-s}$$

where the probability is over uniformly random chosen strings in $\{0,1\}^s$.

Proof. This is not a deep theorem and is basically counting. Note that there are no distinct strings $\mathbf{b}, \mathbf{b}' \in \{0,1\}^s$ such that $\mathcal{C}(\mathbf{b}) = \mathcal{C}(\mathbf{b}') \neq \perp$. So there can be at most 2^r of the 2^s input strings that are mapped to an input that is not $\perp$ which then gives the desired conclusion. □

In order to use this lemma to bound the running time of our algorithm, we will use a binary string to keep track of the randomness that we are utilizing in Moser's algorithmic LLL(1). Then we show that either the algorithm would find a satisfiable assignment in fast, or it would then give a sound compression for all possible inputs of the binary string, which would then contradict Lemma 2.

As the algorithm runs, we want to record a log of the clause that is being fixed. One way to do so is to run down a binary string on $\log_2 m$ bits corresponding to the index of the clause for which we have run `fix` on. But there is a specific structure from our local constraint: if we ran `fix`(C) and then recursed on a clause C' which shares with variables with C then because there are only 2^{k-3} candidates for C' we should be able to do so much more compactly. We use this idea to extend the above algorithm to include a log in the following fashion.

Definition 3. For a clause C in a k-SAT formula F, we denote $N(C)$ to be the collection of all clauses of F that share a variable with C.

In the Extended Moser's algorithmic LLL 2, for a clause C in a collection of clauses F, we use `bin`(C, F) to be the function that returns the binary string corresponding to the index of C in F. We keep track of an additional *log* which we periodicially add more binary strings to as we run the algorithm.

The way to parse what we are writing to the log is at follows: whenever we make a recursive call is made to fix a violated clause $C' \in N(C)$ in the inner loop, we append a bit '1' to the log to indicate that we have entered one deeper layer of the loop. And once we have corrected all the violated clauses

```
1  Extended Moser's algorithmic LLL ;
   Input  : a k-SAT formula F obeying constraints of Theorem 3
   Output: a satisfying assignment
2  Initialize by choosing a random assignment to the variables;
3  while there is an unsatisfied clause C, choose the lexicographically first do
4  |  add_to_Log(Bin(C,F));
5  |  Fix(C);
6  end
7  Function Fix(C):
8  |  Resample the variables of C uniformly at random;
9  |  while there is an unsatisfied clause C' sharing variables with C, choose the lexicographically first do
10 |  |  add_to_Log('1' ∘ Bin(C,N(C)));
11 |  |  Fix(C');
12 |  end
13 |  add_to_Log('0')
```

$C' \in N(C)$ we append a bit '0' to the log to indicate that we have returned to an outer loop. For the outer loop, we record the index of the clause for which we are initiating the fixing process on. Note that we are saving a lot of bits for encoding the index of the formula in the inner loop.

In summation, each outer level adds a total of $\log m + 1$ (the '+1' comes from the fact that upon termination of the outer layer we will also append a '0' to the log) bits to the log and each inner loop adds an additional $\log_2(2^{k-3}) + 2 = k - 1$ bits to the log (the '+2' comes from the fact that we add a '1' to the log to go into a deeper level of recursion and upon termination of this level of the recursion we add a '0' to the log).

Suppose we make t recursive calls of the inner loop, then by the above as well as Lemma 1, the number of bits stored on the log is

$$m(\log_2 m + 1) + (k-1)t. \tag{6}$$

Given the log, we can reconstruct the entire procedure of clause corrections that the algorithm made. In particular, we claim that we can actually "reconstruct all the randomness when we resampled the variables" if we are given the log. Indeed, let C be the first corrected clause, and we know the initial values that were assigned before corrections. Suppose C_1 is the next corrected clause, which are made up of variables *not* shared with C_1 which we know from the initial values and for the other ones that are shared with C because C_1 is violated we now know the values of these variables. In other words, each entry in one depth lower (so after the '1' we wrote on the log0 allows us to reconstruct the respective next values of the k variables in the clause. So after reading the whole log we can describe all the values eahc variable has received during the procedure with the expect for the final assignment of values that it receives.

Lemma 3. *The probability that the Extended Moser's algorithmic LLL(2) does not terminate before doing the inner loop t times is at most*

$$2^{m(\log_2 m - 1) - t}.$$

This lemma would immediately give us the conclusion that we desire – the expected time that it will take for the algorithm to terminate is therefore $O(m \log_2 m)$. To prove this lemma, we want to invoke Lemma 2. To that end, we describe this whole procedure of reconstructing the randomness from the log into the language of compressing functions and decoder functions.

Proof of Lemma 3. Consider a compressing function

$$\mathcal{C}\colon \{0,1\}^{n+tk} \to \{\{0,1\}^{n+(m\log_2 m+1)+t(k-1)}\} \cup \{\bot\}$$

where for a string $\mathbf{b} \in \{0,1\}^{n+tk}$ we run the Extended Moser's algorithmic LLL(2) using $\mathbf{b}$ as the randomness: the first n bits are used in the initialization of the variables (line 2), then we batch the remaining $\mathbf{b}$ into clusters of k each and use them for the random reassignment each time `Fix` is called. Let the algorithm run for t time and if it has not produced a satisfying assignment, then let $\mathcal{C}$ output a string given by the concatenation of the log and also the n bits representing the current assignment of the variables upon termination. By (6) we know that the output has at most the output length we desire; pad zeroes if necessary. Else, if the algorithm found a satisfying assignment in less than t steps then output $\perp$.

Then the decoder function

$$\mathcal{D}\colon \{0,1\}^{n+(m\log_2 m+1)+t(k-1)}\} \to \{0,1\}^{n+tk}$$

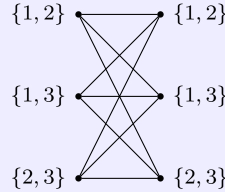
basically follows the procedure we have described in the preamble to the proof. Recall that we can figure out the history of all the values the variables has taken besides the final value, but we have now given the decoder these values by the compression scheme. In other words, we have guaranteed that we have that $(\mathcal{C}, \mathcal{D})$ as constructed satisfies soundness. In other words, by applying Lemma 2, it follows that the probability we get an output of $\perp$ from $\mathcal{C}$ has to be at least $1 - 2^{m(\log_2 m+1)-t}$ which is exactly our desired conclusion. $\square$

3 List coloring

In this section, we study a combinatorial application of the entropy compression argument in Sub-section 2.2 to *list coloring*. Recall the following definition of k-choosability.

Definition 4. We say that a graph G is k*-choosable* if it has a proper coloring no matter how one assigns a list of K colors to each vertex.

Example 3.1. The bipartite graph $K_{3,3}$ is not 2-choosable. It is an easy exercise to check that the following of assignments of colors does not yield a valid coloring.



The goal of this section is to prove Theorem 2, which we recall here for reference.

Theorem 2. There exists k such that the following is true for all $\Delta \geq k$: that if G is a triangle-free graph with maximum degree Δ, then G is $\left\lceil \frac{3\Delta}{\log \Delta} \right\rceil$-choosable.

In fact, we will even produce an explicit algorithm to produce such a coloring; as in Sub-section 2.2 we will give a recursive algorithm for finding such a valid coloring. Before we describe the algorithm, we make some general remarks about coloring problems. In what follows, we will use the following notation: for a vertex v, let $\ell(v)$ denote the list of colors that is assigned to it. For a vertex v, we will also write $N(v)$ to denote the number of neighbors of v in the graph.

A partial coloring of a graph is one where only small of its vertices are assigned a color and others are uncolored which we think of as assigning the color 'blank'. A *valid partial coloring* is one where

adjacent non-blank vertices are not assigned the same color. It is often easier to find a valid partial coloring since we are reducing the number of constraints on the system. One of the key techniques in coloring problems is to first identify a partial coloring in which colors appear many times in vertex neighborhoods so that we can extend the partial coloring greedily to a full coloring. Let t be a parameter to be determined. For a vertex v, and a partial coloring ϕ of the vertices, let us define the events:

- S_v: the event that $|\ell(v) \setminus \bigcup_{w \in N(v)} \phi(w)| \leq t$, i.e. the event where fixing the colors that ϕ assigns to the vertices in $N(v)$, the number of possible colors we can assign to v so that the coloring remains valid is at most t
- R_v: the event that at least t vertices of $N(v)$ are uncolored.

The important observation is that:

> If we find a partial coloring ϕ such that for every vertex v the events S_v and R_v do not hold, then we can extend the partial coloring ϕ to a valid coloring of the entire graph by a greedy coloring process.

Henceforth we will work solely with partial colorings. We can think of S_v and R_v as the bad events corresponding to the unsatisfiability of a clause that we considered in Subsection 2.2. To that end, consider the following analogue to Extended Moser's algorithmic LLL(2). First, fix an ordering on the vertices and the events S_v and R_v; in other words we have an ordering $<$ on vertices and $\prec$ on events with the consistency relationship that if $u < v$ then $S_u \prec S_v$, $R_u \prec R_v$ and furthermore *for all* u, v we have $S_u \prec R_v$. Let $B_v = R_v \vee S_v$. Given a current partial coloring ϕ, when we say recolor a neighborhood of v, we mean for each vertex $u \in N(v)$, choose a uniformly random color in $(\ell(u) \setminus_{w \in N(u)} \phi(w)) \cup \{\text{blank}\}$.

---

1 Algorithmic list coloring;
Input : a triangle free graph with maximum degree Δ and a list of $\lceil \frac{3\Delta}{\log \Delta} \rceil$ colors for each vertex
Output: a valid coloring
2 Initialize by choosing a uniformly random coloring for each vertex;
3 **while** *there is a v for which B_v is true, choose the lexicographically first* **do**
4 | `Fix(`v`);`
5 **end**
6 **Function** `Fix(`v`)`**:**
7 | `add_to_Log(Bin(`$\bigcup_{w \in N(v)} \phi(w)$`));`
8 | Recolor the neighborhood of v;
9 | **while** S_u *holds for a vertex u such that* $\text{dist}(u, v) \leq 3$ *or* R_u *holds for a vertex u such that* $\text{dist}(u, v) \leq 2$ **do**
10 | | `add_to_Log(`*'1'* $\circ$ *('1' if R_u and '0' if S_u)* $\circ$ `Bin(`*shortest path from u to v*`));`
11 | | `Fix(`u`);`
12 | **end**
13 | `add_to_Log(`*'0'*`);`

Here we used `Bin` to denote binary encodings of the objects in question. Note that it suffices to demonstrate that the algorithm terminates in finite time, because upon termination we will find a partial coloring such that S_v and R_v do not hold for every vertex, as desired. First, we note that an analogue of Lemma 1 holds which shows that the outer loop makes a genuine improvement to the partial coloring each time it is run.

Lemma 4. *Each vertex of a graph G (that satisfies the input constraints of the Algorithmic list coloring) appears at most once in the outer loop of the algorithm.*

Proof. For any assignment of colors, after calling $\texttt{Fix}(v)$, a vertex u for which B_u is false remains false. This is because while new events may appear from the recoloring of the neighborhood of v, they are contained in the neighbood of distance at most 2 from v and they would then be fixed in the inner loop. Consequently, the number of vertices for which B_v is true reduces by 1, and this proves the lemma. □

It remains to show that the number of times we run the inner loop is not too large. As before, we want to use the *entropy compression* technique via Lemma 2. The key details for implementing this in Sub-section 2.2 was the construction of an execution log that had the following two properties:

(1) In each iteration of the inner loop, the number of bits β that we wrote to the log is much smaller than than the total bits of randomness γ that we used.

(2) Given the final assignment and the execution log we could "recover all the randomness" that went into running the algorithm; precisely, we could figure out all the values that the variables took at every stage of the algorithm.

Once we have checked that each of these properties hold, then we can run the same proof as Lemma 3 to conclude that we do not run the inner loop for too many steps.

Proof sketch of Theorem 2, assuming (1) and (2) above. Consider a compressing function

$$\mathcal{C}\colon \{0,1\}^{n\log(3\Delta/\log\Delta)+t\gamma} \to \{0,1\}^{n\log(3\Delta/\log\Delta)+t\beta} \cup \{\perp\}$$

where for a string $\mathbf{b} \in \{0,1\}^{(3\Delta/\log\Delta)}$, we run Algorithm 3 using $\mathbf{b}$ as the randomness: the first $n\log(3\Delta/\log\Delta)$ bits are used in the initilization of the coloring (line 2), and then we batch the remainder of $\mathbf{b}$ into clusters of γ bits each and use them for the random recoloring of neighbors each time $\texttt{Fix}$ is called. Let the algorithm run for t time and output $\perp$ if it has not produced a valid coloring. Let $\mathcal{C}$ output a string given by the concatenation of the log and the final coloring upon termination.

By (2), we can write a decoder function $\mathcal{D}\colon \{0,1\}^{n\log(3\Delta/\log\Delta)+t\beta} \cup \{\perp\} \to \{0,1\}^{n\log(3\Delta/\log\Delta)+t\gamma}$ because we can figure out the history of all the colors that we have assigned to each vertex. In other words, $(\mathcal{C},\mathcal{D})$ satisfies soundness, and by Lemma 2, the probability that we output $\perp$ and do not terminate is $2^{3n\log(\Delta/\log\Delta)-t}$. In particular, for a fixed Δ we can find a valid coloring for the graph in linear time! □

For the remainder of the section, we check that there is a way to write few bits to the log to encode the random resampling steps when $\texttt{Fix}$ is called so that properties (1) and (2) hold.

First, to check that (1) holds, we should formally define the binary encodings in the algorithm. The first time we record to the log in (line 7), we are recording the colors of the neighborhood of a vertex v. Apriori we know that there are at least

$$\alpha = \prod_{u\in N(v)} \left(\left| \ell(u) \setminus \bigcup_{w\in N(u)} \phi(w) \right| + 1 \right) \tag{7}$$

possible colorings, and this would mean that encoding the colors of the neighborhood of a vertex v. However, it turns out that actually we can narrow down the possibilities a lot by using concentration bounds.

The key point lies in utilizing the property that G is triangle-free. This would imply that for any vertex v, its neighborhood is an independent set. Consequently if we just restrict to studying events on vertices in $N(v)$ we can think of the events on each vertex as being independent and apply Chernoff/bounded difference inequalities to obtain concentration bounds on these events that would then give us an estimate on the number of viable colorings that we have. We will see that in order for the probabilistic calculations to go through we should set $t = \sqrt{14\Delta\log\Delta}$.

Recall that when we say recolor a neighborhood of v, we mean for each vertex $u \in N(v)$, choose a uniformly random color in $(\ell(u) \setminus_{w\in N(u)} \phi(w)) \cup \{\text{blank}\}$.

Lemma 5. *There exists k such that if $\Delta \geq k$, for a graph satisfying the constraints in Algorithmic list coloring and a vertex v of the graph, if we recolor the neighbors of v according to the process described earlier then*

$$\mathbb{P}[S_v] \leq \Delta^{-4}.$$

In (line 9) of the inner loop, we required S_u to hold for vertices u such that $\operatorname{dist}(u, v) \leq 3$ and furthermore in the lexicographic ordering we also wanted $S_u \prec R_v$ to hold *for all* u, v. Those may seem like arbitrary choices, but actually they were chosen so that whenever the inner loop was executed on R_v then we know that S_v did not hold for any of the neighbors of v.

Lemma 6. *There exists k such that if $\Delta \geq k$, for a graph satisfying the constraints in Algorithmic list coloring and a vertex v of the graph such that S_v is not true for all neighbors of v, if we recolor the neighbors of v according to the process described earlier then*

$$\mathbb{P}[R_v] \leq \Delta^{-4}.$$

We will give proof sketches for each of these lemmas at the end of this section. These lemmas imply only a 2α fraction of the total possible re-colorings of the neighborhood of vertex v allows for either R_v or S_v to hold, this means if we called `Fix(v)` then there are at most $\Delta^{-4}\alpha$ candidates for its neighborhood. In other words, we need at most $\log(2\Delta^{-4}\alpha) = \log_2 \alpha - 4\log_2 \Delta + 1$ bits to do $\operatorname{Bin}\left(\bigcup_{w \in N(v)} \phi(w)\right)$.

The second time we record to the log in (line 10) is about the shortest distance to a violating vertex in the neighborhood of v. Note that there are at most Δ^3 paths from a vertex v to its radius 3 neighborhood (i.e. to vertices that are distance at most 3 from v). Consequently, to encode the shortest path in the algorithm we would need at most Δ^3 bits. In total each call to the inner loop writes about $3\log_2 \Delta + 2$ bits to the log.

To summarize, in each execution of the inner loop we use $\log_2(\alpha)$ number of bits of randomness to sample a uniformly random recoloring of the neighborhod of v as in 7. However, we only write $3\log_2 \Delta + \log_2 \alpha - 4\log_2 \Delta + O(1) < \log_2 \alpha$ bits to the log, which shows that (1) is true.

To show that (2) is true, note that what we record to the log in line (9) allows us to recover exactly which bad event S_u or R_u is being processed at the current step of the algorithm. Now suppose we are processing the call `Fix`(u). Then we know that the coloring at this step matches those of the last step on $N(N(u))$ and in particular we are able to recover $\ell(v) \setminus \bigcup_{w \in N(v)} \phi(w)$ for all $v \in N(u)$ and therefore we would be able to make decode from (line 10) that we described above.

3.1 Deferred proof sketches

We will need to use the following two concentration inequalities.

Theorem 4 (Bounded differences inequality). *Let $X_1 \in \Omega_1, \ldots, X_n \in \Omega_n$ be independent random variables. Suppose $f\colon \Omega_1 \times \cdots \times \Omega_n \to \mathbb{R}$ satisfies*

$$|f(x_1, \ldots, x_n) - f(x_1', \ldots, x_n')| \leq c$$

whenever $(x_1, \ldots, x_n)$ and $(x_1', \ldots, x_n')$ differ on exactly on coordinate. Then the random variable $Z = f(X_1, \ldots, X_n)$ satisfies for any $t \geq 0$

$$\mathbb{P}[|Z - \mathbb{E}[Z]| > t] \leq 2^{-\frac{t^2}{2n}}.$$

Theorem 5 (Chernoff inequality). *If X is a random variable such that $\mu_t = \frac{X_1 + \cdots + X_t}{t}$ where X_i are independent Bernoulli random variables, then*

$$\mathbb{P}[X \geq (1 + a)\mathbb{E}[X]] \leq 2\exp(-2\mathbb{E}[X]/3).$$

Proof sketch of Lemma 5. For notational simplicity, write $k(v) = \left|\ell(v) \setminus \bigcup_{w\in N(v)} \phi(w)\right|$. First note that, taking expectation over the random recoloring of neighbors of v procedure, we have that

$$\begin{aligned}\mathbb{E}\left[k(v)\right] &= \sum_{c\in\ell(v)} \prod_{\substack{u\in N(v)\\ c\in k(u)}} \left(1 - \frac{1}{|k(u)|+1}\right) \\ &> \sum_{c\in\ell(v)} e^{\sum_{\substack{u\in N(v)\\ c\in k(u)}} \frac{1}{|F_u|}} \\ &\geq \left\lceil \frac{3\Delta}{\log\Delta} \right\rceil e^{-d/\lceil \frac{3\Delta}{\log\Delta}\rceil} \geq 2t,\end{aligned}$$

where the second last inequality follows from Jensen's inequality. Now we can apply Theorem 4 since $k(v)$ depends on Δ independent trials in each of the neighborhood and changing one of the neighbors affects $k(v)$ by at most 1 to show that we have good concentration around the average which gives us the desired bounds:

$$\mathbb{P}[|k(v)| \leq t] \leq 2e^{-\frac{t^2}{2\Delta}} < \Delta^{-4},$$

as desired □

Proof sketch of Lemma 6. Let N be the number of neighbors of v that are labeled 'blank'. Because A_u does not hold for all $u \in N(v)$ we have that $\mathbb{E}[N] \leq \Delta/t$. N is also the sum of Δ independent random variables so we can just use Theorem 5 to conclude that

$$\mathbb{P}[N \geq t] \leq \mathbb{P}[N \geq t^2/\Delta \cdot \mathbb{E}[X]] \leq e^{-(14\log\Delta - 1)/3\mathbb{E}[X]} \leq e^{-4\log\Delta} = \Delta^{-4},$$

as desired. □

References

[AS16] Noga Alon and Joel H. Spencer. *The probabilistic method*. Wiley Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc., Hoboken, NJ, fourth edition, 2016.

[CGH10] Karthekeyan Chandrasekaran, Navin Goyal, and Bernhard Haeupler. Deterministic algorithms for the Lovász local lemma. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 992–1004. SIAM, Philadelphia, PA, 2010.

[EL75] 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), Vols. I, II, III*, Colloq. Math. Soc. János Bolyai, Vol. 10, pages 609–627. North-Holland, Amsterdam, 1975.

[Mol19] Michael Molloy. The list chromatic number of graphs with small clique number. *J. Combin. Theory Ser. B*, 134:264–284, 2019.

[Mos09] 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.

[Tao10] Terence Tao. *An epsilon of room, II.* American Mathematical Society, Providence, RI, 2010. Pages from year three of a mathematical blog.