Entropy Compression

Evin Mulligan
Supervisor: Tobias Muller
Second Assessor: Pieter Trapman

July 15, 2022

Abstract

Entropy Compression is a technique to show that a random process or algorithm terminates in finite time. It can be used to prove the existence of certain objects, such as graphs, samples, strings etc. and to find them using a randomised algorithm. By defining an algorithm that losslessly compresses some random inputs into a string which records the history of the steps taken by ur algorithm we can guarantee termination if the bits of entropy read from our random inputs increases faster than the bits of information stored in our record. In this paper we examine the strategy behind entropy compression, discuss the mathematical background and give some examples of it's use.

Contents

1 Introduction

Oftentimes in combinatorics we might try to find an object by taking some other initial object and defining some procedure to incrementally correct it until we've found what we're looking for. This usually involves defining an algorithm which iterates over a given object, for example a set, string or graph, and monotonically improves it until the algorithm terminates, and the desired property is attained. To guarantee that our procedure terminates we try to exploit some bounded but monotonically increasing quantity, so that the algorithm terminates when the bound is reached.[10]

Depending on what is to be proven, there are several options for which quantity we wish to exploit. Some well known choices include mass increment arguments, where the object is made "heavier", or rank reduction arguments, where the rank, dimension or order of our object is reduced to some minimum. In 2010, Moser and Tardos [6] introduced a new monotonicity argument, which came to be known as *entropy compression.* The argument applies to probabilistic algorithms which, taking a set of random inputs F, uses elements of F to modify a randomly chosen object from a distribution A, replacing it at each step t with an improved A_t and a shorter F_t made up of whichever elements of F are not satisfactory. The argument relies on keeping a record of each step and storing it in a string R, such that at each step of the algorithm the original object A and input F can be reconstructed by the states A_t, F_t, R_t, in other words, our algorithm *compresses* the information content of the original object and input $A + F$ into $A_t + F_t + R_t$, which encodes all of the courses the algorithm can take in t steps.

The notion of entropy captures the amount of information we expect to gain by sampling from a random variable. Since our inputs $A + F$ is randomly chosen, and each correction we make is a random sample from $A + F$, at each step of the algorithm we are "reading" a certain amount of entropy, and recording it in R. A random variable like $A + F$ can't be losslessly compressed into a string whose expected size is smaller than it's Shannon entropy (see preliminaries), so if the amount of information we store in R at each step t is less than the amount we read from $A + F$, that is, if we can find a sufficient lower bound on the entropy of $A + F$, and if the size of $A_t + F_t + R_t$ is strictly smaller than this bound for large enough t, then there must be a limit on the amount of times the algorithm can iterate, because eventually we run out of space to store the information we gain from each iteration. Note that this limit can be randomly distributed and is not necessarily fixed. Once there is no more room to compress, the algorithm must halt.

Theorem: Entropy Compression Consider an algorithm Φ and an input F of IID uniformly sampled bits. Let Φ be such that for each step $t = 0, 1, 2, ...$ the following holds

- Φ maintains a string R_t recording it's history after each step t, such that the random bits of F read so far can be recovered from R_t
- R_t has length at most $r_0 + t\Delta r$ at time t
- $f_0 + t\Delta f$ bits have been read after time t

If $\Delta_f > \Delta_r$, then the step t at which Φ terminates is at most

$$\frac{r_0 - f_0}{\Delta_f - \Delta_r}$$

This statement gives a formal bound on the runtime in terms of the entropy read per iteration, however since this is almost never explicitly computed we will decline to prove it, and simply note that it must halt in principle. Interested readers may refer to the original blog post [8].

Since we in general consider randomised Las Vegas algorithms, there is a point where the probability of halting becomes greater than zero, and since our algorithm can only terminate if the desired object is found, there is a positive probability of finding it which can only be the case if it exists. Our strategy is thus a type of probabilistic method, common in combinatorics and popularised by Erdős. These problems are quite easy when the corrections made to our initial object are independent, however it can become quite complicated when some degree of dependency is allowed. In this case a correction in one area can undo a previous correction, or otherwise violate some area of our desired object. Then we can have violations stack up faster than they can be corrected, or get stuck in a loop where our desired property is continuously violated and corrected. Much of the work then lies in determining how much dependency can be tolerated, and in mapping these dependencies to find the likelihood of falling into one of these traps.

Since it's introduction in 2010, the method has seen uses in varied areas of mathematics. Moser's original argument gives the first constructive proof of the Lovasz Local Lemma, an important tool for probabilistic methods. Moser first deals with a special case involving a k-satisfiability problem[7], showing relevance to logic, and later proves the theorem in it's full generality, a probability problem [6]. Perhaps the most fertile area of research has been combinatorics, with many applications to graph-colouring problems, an example of which will be dealt with below[4]. In general I wish to demonstrate the ease and broad applicability of the method, to perhaps inspire the reader to make use of it in future work. I also hope to give an impression of common problems in combinatorics and the interesting constructive methods with which they are handled.

Entropy compression is an example of mathematics at it's best and most beautiful. Moser's argument begins with the simple intuition behind the Lovasz Local Lemma; if failure can not account for all sufficiently independent possibilities, a successful solution must exist. From there, through careful consideration we can discover complex truths about a variety of concepts and constructs.

2 Preliminaries

This article aims to be as accessible as possible to those without a strong background in combinatorics. Knowledge of basic concepts in probability theory and graph theory are assumed, but since many of the arguments draw on various areas of mathematics we give some of the necessary background below.

2.1 Graph Colouring

A graph colouring is an assignment of labels conventionally called colours to the elements of a graph G, subject to certain constraints. The case where colours are assigned to the edges of a graph is called an edge colouring, and when colours are assigned to vertices a vertex colouring. Usually we consider *proper* colourings, where no two edges (vertices) which share a vertex (edge) are assigned the same colour. A proper colouring with at most k colours is called a *k-colouring.* The chromatic number $\chi(G)$ of a graph G is the smallest number of colours required for a proper colouring. (include section on girth)

2.2 Acyclic Colouring

An acyclic colouring is a colouring in which every cycle contains at least three colours; alternatively, a colouring for which any subgraph of edges/vertices of only two colours is acyclic.

2.3 Random Algorithms

In graph colouring problems it is common to use an algorithm to construct a colouring with the desired property. For our purposes we are interested in random or randomised algorithms, in particular so-called Las Vegas algorithms, wherein the algorithm is guaranteed to complete with the correct answer, but the runtime is randomly distributed. Compare to Monte Carlo algorithms, where the run time is fixed, but the algorithm can fail with some probability.

2.4 Dyck Words

To help with a counting problem we will use the concept of Dyck words. A formal language L over an alphabet $\mathcal{A}$ is a subset of *words* made up of elements belonging to $\mathcal{A}$. An alphabet can be any set, and a word is a string or sequence of elements from an alphabet. A *prefix* of a string w is a substring beginning with the first element of w. A partial Dyck word on $\{0, 1\}$ is a word such that every prefix contains at least as many zeroes as ones, and a Dyck word of length $2t$ is a partial Dyck word with exactly t zeroes and t ones.[4]

2.5 Rooted Plane Tree

Like Dyck words, these will be used for counting. A tree is a connected graph with no cycles. A rooted plane tree is a tree, wherein a vertex is designated as the root, and an ordering is assigned to the children of each vertex. Since the vertices of a rooted plane tree are ordered we can label them $v_1, v_2, ...$ and define the generating function as f (add def. of generation)

2.6 Shannon Entropy

Shannon entropy is a measure of how much information we expect to gain by sampling a random variable and having some event occur. It is equal to the negative log probability of the event.

3 Lovasz Local Lemma

For a sequence of independent events $A_1, A_2, ...A_n$, there is always a chance that no events occur, provided each event occurs with probability less than one. However if we allow dependence between events, it is possible that some event A is fully determined by a collection of others, such that if these events occur, A occurs with probability one. We can take as a simple example a binary random variable X_1 that takes values 1 with probability p, and 0 with probability $(1 - p)$, and another X_2 that equals $1 - X_1$. Then the events $X_1 = 1$ and $X_2 = 1$ can not both occur.
The Lovasz Local Lemma allows us to relax our independence assumption, meaning as long as the A_is are sufficiently independent, that is if events are only dependent "locally", we can guarantee some possibility of no events occurring. The first proof of this statement was given by Lovász and Erdős in 1975 [3], the statement below is an improved result from Joel Spencer in 1977 [9].

Lovász Local Lemma: *Let $\mathcal{A} = \{A_1, A_2, ...A_n\}$ be a sequence of random events where each event has probability at most p, and each event is independent from all others except at most k of them. If p is small enough that*

$$ep(k + 1) \leq 1$$

then the probability that none of the events in $\mathcal{A}$ occurs is nonzero.

There also exists an asymmetric version, where p, k are not fixed for all events, but are allowed to vary. In this case the Lemma becomes

Asymmetric Lovász Local Lemma: *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 $\mathcal{A} \setminus (A \cup \Gamma(A))$. If there exists an assignment of reals $\psi : \mathcal{A} \to (0, 1)$ such that*

$$\forall A \in \mathcal{A} : \mathbb{P}(A) \leq \psi(A) \prod_{B \in \Gamma(A)} (1 - \psi(B))$$

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

This shows that we can allow more dependencies for an event A, but must accordingly reduce the probability (or vice versa) in order to preserve the result of the symmetric version.

Moser and Tardos discovered a constructive proof for a special case of the Lemma in 2009 [7], where they deal with the k-satisfiability problem. The proof in full generality was developed the following year [6], a feat for which they were awarded the Gödel prize in 2020. Prior to Moser's result only nonconstructive proofs existed, the Lemma could only guarantee the existence of an object and offered no way to explicitly construct one. Moser's result, sometimes known as the Algorithmic Lovász Local Lemma, improves the result by providing an algorithm for which desired object can itself be found. The only extra restrictions for this algorithmic version of the Lemma are that we consider events determined by a subset of a set of random variables $\mathcal{P}$, and let $\Gamma(A)$ be the set of events which depend on one or more of the same random variables as A

Algorithmic Lovász Local Lemma: [6] *Let $\mathcal{P}=\{P_1,P_2,...P_m\}$ be a set of independent random variables, and let $\mathcal{A}=\{A_1,A_2,...A_n\}$ be a sequence of random events determined by these variables. If there exists an assignment $\psi:\mathcal{A}\to[0,1)$ such that*

$$\forall A\in\mathcal{A},\mathbb{P}(A)\leq\psi(A)\prod_{B\in\Gamma(A)}(1-\psi(B))$$

then there exists a sampling from the variables in $\mathcal{P}$ such that none of the events in $\mathcal{A}$ occur. Moreover, the expected number of steps taken by the algorithm before halting is

$$\sum_{A\in\mathcal{A}}\frac{\psi(A)}{1-\psi(A)}$$

3.1 Moser and Tardos

We first let $\mathcal{P}$ be a finite collection of mutually independent random variables in a probability space Ω. We consider events A which are determined by $\mathcal{P}$. We call S the unique minimal subset of random variables $P\in\mathcal{P}$ which determine A. Since we are looking for the case where some collection of events do *not* happen, we say that an evaluation of the variables in S *violates* A if A occurs in the evaluation. For each A in a desired set of events $\mathcal{A}$ we denote the set of variables which determine A as $\mathrm{vbl}(A)$.

Dependency between events can be modelled by a dependency graph, where each node represents an event and the edges represent dependency between them. We define the dependency graph $G_{\mathcal{A}}$ as the graph with nodes in $\mathcal{A}$, and where edges exist between events A,B if intersection of the sets of variables which determine A,B $\mathrm{vbl}(A)\cap\mathrm{vbl}(B)=\emptyset$, in other words if there is at least one random variable in $\mathcal{P}$ which determines both A,B. We then denote by $\Gamma(A)$ the set of neighbours of A in $G_{\mathcal{A}}$, and by $\Gamma^+(A)=\Gamma(A)\cup\{A\}$ we denote the *inclusive neighbourhood*, which is the set consisting of A together with it's neighbours. Then A is dependent on the events in $\Gamma(A)$, and independent from $\mathcal{A}\setminus(\Gamma(A)\cup\{A\})$. Note that this means $\Gamma(A)$ satisfies the constraints from the theorem statement.

Moser then defines the algorithm at the heart of the entropy compression method, which will guarantee the existence of an evaluation which does not violate any $A\in\mathcal{A}$, and more importantly finds such an evaluation. The algorithm works as follows:

1. For each $P\in\mathcal{P}$ take a random evaluation v_P from Ω

2. For each $A \in \mathcal{A}$ which is violated, we "resample", or take a new random evaluation v_P for each $P \in vbl(A)$. A can be chosen arbitrarily.

3. When we have reached an evaluation $v = (v_P)_{P \in \mathcal{P}}$ such that no event A is violated, we are done.

By construction the algorithm creates an evaluation for which no A occurs. What remains to be seen is that the algorithm terminates in finite time. It could be the case that each resampling violates some other clause, and the violations stack up faster than they can be resolved. Moser shows that the algorithm terminates in finite times, in fact for each A the expected number of times A is resampled is

$$\frac{\psi(A)}{1-\psi(A)}$$

meaning the total runtime is

$$\sum_{A \in \mathcal{A}} \frac{\psi(A)}{1-\psi(A)}$$

3.1.1 Execution Logs and Witness Trees

First we fix a procedure for selecting which A to correct at each step. This can be chosen arbitrarily, deterministic or random, and the specific procedure does not matter to the result. Our argument relies on keeping a log of each step of the algorithm. We therefore let $C : \mathbb{N} \to \mathcal{A}$ record which event has been chosen for resampling at each step, and call this the *log.* If event A is resampled at time t then $C(t) = A$. Note that if the sampling is random, C is a random variable determined by the random sampling of $\mathcal{P}$

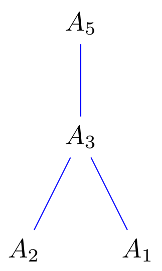
We can use C to define a *witness tree* $\tau = (T, \sigma_T)$ which will help us limit the runtime of our algorithm. This consists of a finite, rooted tree T along with a labelling $\sigma_T : V(T) \to \mathcal{A}$ assigning each vertex of the tree to an event A. The map is chosen such that for each u, the children of u is labelled with events in the inclusive neighbourhood of u. We obtain τ from C as follows: At step t of the algorithm let $\tau_C^{(t)}$ having label $C(t)$ be the root of our tree. We then work backwards through $C(t)$. For $i = t-1, t-2, ...1$ we build our tree as follows. If there is a vertex $v \in \tau_C T^{(i+1)}$, such that $C(i) \in \Gamma^+(v)$, we choose the farthest such v from the root and attach a vertex labelled $C(i)$ to it. If there are more than one v at equal distances we can choose arbitrarily. Finally let $\tau_C^(t) = \tau_C^{(1)}(t)$.

As an example, let:

- $C = (A_1, A_2, A_3, A_4, A_5)$
- $\Gamma^+(A_1) = \{A_1, A_3, A_4\}$
- $\Gamma^+(A_2) = \{A_2, A_3, A_5\}$
- $\Gamma^+(A_3) = \{A_1, A_2, A_3, A_5\}$
- $\Gamma^+(A_4) = \{A_1, A_4\}$
- $\Gamma^+(A_5) = \{A_2, A_3, A_5\}$

Then the witness tree for the 5th step of our algorithm will be

$\tau_C^{(1)}$



The witness tree $\tau_C(t)$ gives all the reasons $C(t)$ might have been resampled, that is all of the evaluations that may have violated $C(t)$. If the witness tree contains only the root, it can only have been violated by the initial sampling, all other nodes represent resamplings between $1, t$ which may have violated $C(t)$. For some tree τ, we say τ occurs in our record C if there is some t with $\tau_C(t) = \tau$. Most importantly for our argument, each resampling of an event A corresponds to a distinct witness tree τ, in particular, the number of times an event A is sampled is equal to the number of witness trees with root labelled A. Moreover the probability of A being resampled is is equal to the probability of some witness tree occurring in C. We can therefore bound the expected number of resamplings of A, which we call N_A, by determining the probability of a witness tree occurring in C.

To do this we will need to analyse these witness trees further, then come up with a way of probabilistically generating them. We call a tree proper if each distinct child of a vertex receives a distinct label. In the following Lemma we show that witness trees as defined above are proper, and bound the probability of a given tree occuring in C.

Lemma 2.1 *Let τ be a fixed witness tree and C be the (random) log produced by the algorithm.*

1. *If τ occurs in C, then τ is proper*
2. *The probability that τ occurs in C is at most $\prod_{v \in V(\tau)} \mathbb{P}(v)$*

Proof: Suppose the τ occurs in C. Then $\tau = \tau_C(t)$ for some t. For any vertex $v \in \tau$ we call the distance from the root the *depth* and denote it by $d(v)$. We then denote by $q(v)$ the step of the algorithm at which v was added. Since our procedure for constructing witness trees counts backwards from t, this is the largest tree $\tau_C^{(q)}(t)$ containing v. Note that if u, v are dependent, and if u was added later than v $(q(u) < q(v))$, then u must be further from the root, $(d(u) > d(v))$, because u is either attached to v, or attached to another vertex of equal or greater depth, since our procedure specifies we attach it to the furthest desired vertex from the root. This means any two vertices of the same depth must be mutually independent, so any two children of a given vertex must have different labels. We conclude that τ is proper.

To prove our second claim we need to define a so-called $\tau-$check. This is a breadth first search, which starts from the last generation and visits each vertex at this depth, before moving back a generation and visiting the vertices, and so on until the root. At each vertex we sample the event with which it is labelled, and check if it is violated. If all events are violated we say the $\tau-$check passes. Of course, the probability that the $t-$check passes is equal to the product of the probability of each event being violated, that is $\prod_v \in V(\tau)\mathbb{P}(v)$. To bound the probability of a witness tree occurring in C we must show that any witness tree passes a $\tau-$check. Intuitively this is simple. For τ to be a witness tree it must record vertexes which are resampled, and vertices are only resampled if they are violated.

Let v be a vertex in $V(\tau)$ for some witness tree τ. Since v is resampled at step $q(v)$, meaning v was violated before this sampling, so when the $\tau-$check arrives at v it must find it to be violated. This holds for all v, meaning any witness tree passes a $\tau-$check. This means the set of trees occuring in C is contained in the set of trees which pass a $\tau-$check. Furthermore, the probability of a tree τ occuring in C is less than or equal to the probability that τ passes a $\tau-$check. This proves our second claim. For a more rigorous treatment refer to [6].

We have now obtained a bound on the probability of a witness tree occuring in our algorithm. We can now use this to bound the expected number of times an arbitrary event is sampled. To do this we will define a stochastic process to randomly generate witness trees. In particular we will be using a Galton-Watson process. For those unfamiliar with stochastic processes, this is a collection of random variables which can be used to characterise a tree. Usually we consider the random variables Z_n which denotes the number of vertices in the nth generation, and variable $X_{n,i}$ which determines the number of children of the ith vertex in the nth generation. The process works by progressing through the vertices and adding children to them according to some random distribution. In order to generate witness trees having som event $A \in \mathcal{A}$ as a root,.we define a Galton-Watson branching process where chidren are added to each vertex as follows. In the first round we consider the root labelled A, then for each dependent event $B \in \Gamma^+(A)$ we add a vertex labelled B with probability $\psi(B)$, and skip it with probability $1 - \psi(B)$. These choices are taken independently. Depending on the probabilities in question the process can go extinct because no new children are born.

We are now ready to find the probability that a given witness tree occurs is generated by our Galton Watson process. Let $\psi'(B) = \psi(B) \prod_{C \in \Gamma(B)}(1 - \psi(C))$. This is the probability that a vertex labelled with event B is added to vertex A, and that none of the other events dependent on A are added, that is to say it is the probability that B in particular is chosen. Note this is in the assumption from the original theorem statement.

Lemma 3.1 *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*

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

Proof: For our process to arrive at τ, we must have that every vertex $v \in V(\tau)$ is added and every other potential vertex is avoided. We take an arbitrary $v \in V(\tau)$ with label A and denote by $W_v \subseteq \Gamma^+(v)$ be the set of events dependent on v which do not occur in τ, that is to say, the vertices we wish to avoid. Then to obtain the probability of τ occuring p_τ we multiply the probability of each $v \in V(\tau)$ being added, and the probability of each dependent event being avoided. This gives us

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

This can be rewritten by multiplying by $1 = \frac{1-\psi(v)}{1-\psi(v)}$ for each v to obtain

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

We recognise $\psi'(B)$ as defined above, and the equation becomes

$$p_\tau = \frac{1}{\psi(A)} \prod_{v \in V(\tau)} \psi'(v)$$

Proving the Lemma.

We now approach the end of our proof. Recall that we can count the number of times an event A is resampled by counting the number of witness trees with root A in the log C of steps taken in the entropy compression algorithm. We denote by $\mathcal{T}_A$ the set of all such witness trees. The event is resampled if and only if there is a corresponding witness tree occurring in the log C. This means the number of resamplings of A is equal to the number of trees in $\mathcal{T}_A$ in C and the expectation becomes

$$\mathbb{E}(N_A) = \sum_{\tau \in \mathcal{T}(A)} \mathbb{E}\ \mathbf{1}_{\{\tau \text{ occurs in } C\}} = \sum_{\tau \in \mathcal{T}(A)} p_{\{\tau \text{ occurs in } C\}}$$

From Lemma 2.1 we obtain

$$\mathbb{E}(N_A) = \sum_{\tau \in \mathcal{T}(A)} p_{\{\tau \text{ occurs in } C\}} \leq \sum_{\tau \in \mathcal{T}(A)} \prod_{v \in V(\tau)} \mathbb{P}(v)$$

By assumption from our theorem this becomes

$$\mathbb{E}(N_A) \leq \sum_{\tau \in \mathcal{T}(A)} \prod_{v \in V(\tau)} \mathbb{P}(v) \leq \sum_{\tau \in \mathcal{T}(A)} \prod_{v \in V(\tau)} \psi'(v)$$

Now we can use the result from lemma 3.1 to obtain

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

and finally since the galton-watson process produces a single tree, and each tree in $\mathcal{T}(A)$ is only one possibility, their probabilities sum to less than or equal to one (inequality is strict when there is a possibility the tree is infinite, in this case the process produces a tree that is not in $\mathcal{T}_A$).

$$\mathbb{E}(N_A) \leq \frac{\psi(A)}{1 - \psi(A)} \sum_{\tau \in \mathcal{T}(A)} p_\tau \leq \frac{\psi(A)}{1 - \psi(A)}$$

So for each A the expected number of resamplings is finite, and the expected runtime of the algorithm is the sum of each N_A

$$\sum_{A \in \mathcal{A}} \frac{\psi(A)}{1 - \psi(A)}$$

which is finite. In particular this guarantees the algorithm terminates with positive probability. Since termination of the algorithm means we have found a sample which does not violate our set of events, the algorithm has a positive probability of finding such a solution, which guarantees it's existence. We have therefore proven the desired result.
You might note that we never compute or even discuss entropy in the proof. Indeed we don't need to, although it's clear that as our algorithm is generating a witness tree at each step and as the probability of one of these trees being generated gets smaller and smaller, the entropy accordingly gets larger. Recall that the entropy of an event is equal to the negative log probability of the event and is monotonically increasing as the probability decreases. While the amount of entropy read by our algorithm is therefore increasing, we are storing the steps of the algorithm in our record according to a fixed procedure, and the information stored per step doesn't increase accordingly,

and so the algorithm halts. Usually when using the method we will not need to explicitly compute the entropy, and the idea of entropy compression mostly shows what's going on "under the hood", a more theoretical motivation than whichever explicit argument is used in a particular proof.

Moser's paper goes into further detail about the Local Lemma, defines a parallel algorithm to the same effect and also offers a non-deterministic alternative to the random procedure hereabove. Rather than repeat the process for the non-deterministic case we will instead demonstrate by giving an example of the non-deterministic algorithm applied to a completely different problem; the task of finding an acyclic graph colouring for an arbitrary graph.

4 Acyclic Edge Colouring Using Entropy Compression

In this section we relate and compare the findings of Esperet and Parreau(2013)[4], wherein the deterministic variant of Moser's entropy compression argument is used to establish a bounds on the *acyclic chromatic index,* or the smallest possible number of colours in an acyclic edge colouring, and to guarantee the existence of such a colouring.

4.1 Esperet and Parreau

As early as 1991 [1], the Lovasz Local Lemma has been used to impose bounds on the acyclic chromatic index. Here however, the aim is to apply Moser's entropy compression strategy directly to the problem rather than using the lemma itself, in order to show the strength and versatility of this argument, how it works, and how it can be adapted to a wide range of graph colouring problems.

In this instance, the broad strategy is as follows; We define a randomised (deterministic but with random inputs) algorithm which terminates when an acyclic edge colouring is attained. The algorithm is equipped with a set of records which log each step i of the algorithm and which, together with the state of the colouring step i, is enough to uniquely determine the input. By showing that, for some stopping time t sufficiently large, the number of possible combinations of records and partial graph colourings is strictly larger than the number of possible inputs such that the algorithm *does not* terminate, we conclude that there exists some input for which the algorithm *does* terminate. Since our inputs are randomly chosen, this implies the algorithm terminates by time t with positive probability. Finally, since the chances of selecting an input that guarantees an acyclic colouring is positive, such a colouring must exist. Our algorithm includes a parameter γ which determines the number of colours in our colouring, and by bounding this parameter we can bound the acyclic chromatic index.

The bulk of the work in this argument consists essentially of counting problems, as we need to bound the set of inputs and records. This is tricky to do directly but fortunately, for large enough t the bounds are quite forgiving and by showing equivalencies between our set of records and other objects which are easier to count, (namely Dyck words and rooted plane trees) we can at last attain our desired bounds. Now that we have an overview of the argument we are ready to get into the details.

We'll begin with some notation. Let $a'(G)$ denote the acyclic chromatic index, i.e. the smallest number required for an acyclic colouring of an arbitrary graph G. Let Δ be the maximum degree of the vertices in G. Recall that a proper edge colouring requires at least $\Delta - 1$ colours, so we take a parameter γ greater than 1, and we take $K = \lceil(\gamma + 2)(\Delta - 1)\rceil$. We claim that for γ, therefore K large enough, there exists an acyclic edge colouring on G with at most K colours.

4.1.1 Entropy Compression Algorithm

We define a randomised (Las Vegas) algorithm on the edge set E of G. The basic idea is to order the edges of E as $e_1, e_2, ...e_m$, where $m = |G|$, and starting with the e_1 assign each successive edge

e_j with some “colour” from $\{1, 2, ...K\}$ that is not already assigned to an edge adjacent to e_j, and if a 2-coloured cycle is created we recolour it and all other edges (except two, as we will explain later). This guarantees a proper colouring, since edges are chosen excluding adjacent colours, and an acyclic colouring, since 2-cycles are recoloured. What remains to be seen is that this algorithm terminates in finite time, it could be the case that cycles are recoloured faster than the graph than be coloured. To show that the algorithm terminates we will need to analyse the algorithm fully. The analysis uses a similar entropy compression argument to that developed by Moser, and relies on keeping a record of each step.

We are interested in a randomised algorithm running on a deterministic instance. To achieve this we define a deterministic algorithm where the input is a vector randomly chosen from our randomly chosen entries. We take for our set of colours $\{1, 2, 3, ...K\}$.

Let t be large and consider the vector of randomly chosen entries

$$F = (f_1, f_2, ...f_t)$$

where, at step i of our algorithm, f_i is used to colour an edge e_j as follows. Let $e_j = uv$. and let S be the set of free colours, that is $S = \{1, 2, ...K\} \setminus S'$, where S' is the set of colours assigned to edges $xy \neq uv$ such that

- x = u or x = v
- edges ux and vy exist and have the same colour.

This first condition prevents adjacent colours and the second prevents 4-cycles.

The set of “taken” colours S' has cardinality $|S'| \leq 2(\Delta - 1)$ elements, because each edge e_j has two vertices which have at most $\Delta - 1$ edges excluding e_j. So the set of “free” colours S will be at least

$$\begin{aligned} K - 2(\Delta - 1) &= \lceil(\gamma + 2)(\Delta - 1)\rceil - 2(\Delta - 1) \\ &= \lceil\gamma(\Delta - 1)\rceil \end{aligned}$$

This means that entries of F will be in $\{1, 2, ...\lceil\gamma(\Delta - 1)\rceil\}$. We use F to colour G by taking at each step i the f_ith smallest element of S and applying it to our next uncoloured edge e_j. As mentioned, choosing our colours in this manner guarantees our colouring is proper and no 2-coloured $4-$cycle (cycle of length 4) is created. Cycles of odd length are of no concern, since they can not be properly coloured with only 2 colours and will never be cyclic. Now, in case our colouring produces a $2-$coloured cycle of length 6 or greater, say $e_{i_1}, e_{i_2}, ...e_{i_{2k}}$, with $e_{i_1} = e_j$ we uncolour all edges except 2 edges, say e_{i_2}, e_{i_3}, which will be used for our record. Since e_j is uncoloured, and all earlier coloured edges contain no cycles, our colouring remains acyclic.

To show that the algorithm terminates, we must keep a record such that at each step i, the record until step i together with the partial colouring ϕ_i is enough to uniquely determine the first i entries $f_1, f_2, ..f_i$ of our randomly chosen input vector F. We define for our record a vector R having t entries, one added at each step of our algorithm as follows. If, at step i, an edge e_j was coloured without issue and no cycle was created, the record R_i is left empty. In the case that a cycle was created we record the cycle as follows. Suppose a cycle of length $2k$ was created, $C = e_{i_1}, e_{i-2}, ...e_{i_{2k}}, e_{i_1}$, with $e_{i_1} = e_j$. Since there is a finite number of cycles of length $2k$ containing e_j, (at most $(\Delta - 1)^{2k-2})$), we can fix an order on these cycles $C_1, C_2, ...C_s$. If our coloured cycle is, say, C_l for $1 \leq l \leq s$ we enter it in our record vector as the pair (k, l): the l^{th} cycle of length $2k$ according to our ordering.

Our record R will be used to determine our input F. Below we show that this is valid, i.e. that F can be determined using R

4.2 Determining the Input from R

We will now show that the pair (R, Φ_t) consisting of the record vector and the partial colouring at time t are enough to uniquely determine the input vector F.

Lemma 1: *At each step i, the set of uncoloured edges is uniquely determined by the $R_1, R_2, ...R_i$*

Proof: Let X_i be the set of uncoloured edges after step i. We prove the lemma by induction on i

1. **Base Case:** X_1 is simply the set of all edges except e_1, so X_1 can be determined from R_1
2. **Induction Hypothesis:** We suppose X_{i-1} is uniquely determined by $R_1, R_2, ...R_{i-1}$
3. **Induction Step** By induction, X_{i-1} is determined, and since edges are coloured in order of their index, $X_{i-1} = \{e_j, e_{j+1}, ...e_m\}$ for some j. If our record for this step, R_i is empty, then e_j is coloured and $X_i = X_{i-1} \setminus e_j$. Otherwise $R_i = (k, l)$, and we know the lth cycle C_l of length $2k$ containing e_j was uncoloured (except for two edges which are determined by our algorithm). So $X_i = X_{i-1} \bigcup (C_l \setminus \{e_{i_2}, e_{i_3}\})$. In either case the set X is uniquely determined.

Lemma 2: *At each step i the application defined by our algorithm, i.e. the application that assigns to each input $(F_j)_{j \leq i}$ to the outputs $((R_j)_{j \leq i}, \Phi_i)$* is injective. Alternatively, our input F is uniquely determined by the pair (ΦR).

Proof: We again use induction on i. Let $\Phi_i, (R_j)_{j \leq i}$

1. **Base Case:** For $i = 1$, only one edge is coloured, so the colour on this edge is the first entry f_1.
2. **Induction Hypothesis:** Assume $(F_j)_{j \leq i}$ uniquely determined by $((R_j)_{j \leq i}, \Phi_i)$.
3. **Induction step:** By Lemma 1 we can determine sets of uncoloured edges X_i and X_{i-1}, from which we obtain the particular edge e_j which is coloured at step i. First assume R_i is empty. Then from Φ_i we simply uncolour e_j to obtain Φ_{i-1}. Then since we know $(R_j)_{j \leq i-1}$ and Φ_i, by our induction hypothesis we can determine $(f_j)_{j \leq i-1}$, and we need only find f_i. Let $c \in 1, 2, ...K$ be the colour assigned to e_j in Φ_i, and a be the number of colours disallowed by our algorithm at step i which are smaller than c, i.e. the number of elements $\#\{s \in S' | s < c\}$. We obtain our colour c by taking the $f_i th$ free colour, that is starting at i and counting f_i steps, skipping all taken colours, we have $c = f_1 + a$. Since c and a are known, we can find $f_1 = c - a$. So $(f_j)_{j \leq i}$ is determined. Now assume R_i is (k, l). Then from our record know the exact cycle uncoloured at step i, $C_l = (e_{i_1}, e_{i_2}, ...e_{i_{2k}})$, with $e_j = e_{i_1}$. Since C_l is two coloured, we know every odd indexed edge $e_{i_5}, e_{i_7}, ...e_{i_{2k-1}}$, was previously coloured with the same colour as e_{i_3} which we recall was left coloured, and every even indexed edge $e_{i_4}, e_{i_6}, ...e_{i_2 k}$ was previously coloured with the same colour as e_{i_2}. So, to obtain ϕ_{i-1} we simply recolour these edges. By our induction hypothesis we can determine $(f_j)_{j \leq i}$. We also know that e_j received the same colour as the odd edges, as in our cycle $e_j = e_{i_1}$, so we can obtain $f_i = c - a$ as above. In both cases we have determine $(f_j)_{j \leq i}$ from $(\Phi_i, (R_j)_{j \leq i})$, and we can conclude our assignment is injective.

We now define the set $\mathcal{F}$ as the set of all possible inputs. As noted before, the elements of f are in $\{1, 2, ...\lceil \gamma(\Delta - 1) \rceil\}$, so the number of all choices for F is $|\mathcal{F} = |\lceil \gamma(\Delta - 1) \rceil^t$. We denote by $\mathcal{F}_t$ the set of all inputs such that our algorithm fails to terminate by time t, that is the set of inputs such that the graph is not coloured by time t. By definition, we have $\mathcal{F}_t \leq \mathcal{F}$, meaning

$$\mathcal{F}_t \leq \lceil \gamma(\Delta - 1) \rceil^t$$

If we can prove that the number of records is $o(\lceil \gamma(\Delta-1)\rceil^t)$, that is if our set of all possible records $\mathcal{R}$ is such that

$$\lim_t \to \infty \frac{|\mathcal{F}_t|}{\lceil \gamma(\Delta-1)\rceil^t} \to 0 \qquad \text{as } t \to \infty$$

Then there must initial vector F such that the algorithm does terminate, for t large enough, in other words, that an acyclic colouring exists.

4.3 Bounding $|\mathcal{F}_t|$

Now we let $\mathcal{R}_t$ be the set of all possible records up to time t. As we have just proven, any F can be uniquely determined from Φ and R_t. This means that $|\mathcal{F}_t|$ is less than the number of possible colourings times the number of possible records by time t. An obvious upper bound for the number of possible partial colourings using K colours is $(K+1)^m$: m edges with $(K+1)$ choices for each edge, K colours and an option for uncoloured. This proves our next lemma.

Lemma 3: $|\mathcal{F}_t| \leq (K+1)^m |\mathcal{R}_t|$

Since $(K+1)^m$ does not depend on t, it is enough to find a sufficient bound for $|\mathcal{R}_t|$ to show that $|\mathcal{F}_t| \leq |\mathcal{F}|$. To do this we will need a few tricks, namely to translate our record vectors into Dyck words, and then rooted trees, for which a bound is available.

Our non-empty record entries (k, l) refer to a unique cycle of length $2k$ containing e_j, which is the uncoloured edge of smallest index at step i. Since Δ is the maximum vertex degree, so for each consecutive edge in the cycle there will be at most $\Delta - 1$ choices for the next edge, and since the first edge is determined as e_j, and the last edge must be whichever leads from the second-last edge to e_j, we have at most $2k-2$ such choices. This gives us an upper bound on the number of cycles of length $2k$ containing e_j, and our cycle is C_l for $l \leq (\Delta-1)^{2k-2}$. We will now attempt to find an injection between $\mathcal{R}_t$ and Dyck words of length $2t$.

We consider our cycles as a word $w = w_1, w_2, ...w_{2k-2}$ of length $2k-2$ on the alphabet $\mathcal{A} = \{1, 2, ...\Delta - 1\}$, where each w_j represents the j_th edge in a cycle, and define the function

$$\theta_k(w) = 1 + \sum_{i=1}^{2k-2} (w_i - 1)(\Delta - 1)$$

This function has range in $\{1, 2, ...(\Delta-1)^{2k-2}\}$ and establishes a bijection between these integers and the words of length $2k-2$. Note any bijection will do, this one is chosen for simplicity. In particular, we can take a number l in $1, 2, ...2k-2$ and by taking the inverse $\theta^{-1}(l)$ we can find the word that produces it, so using θ^{-1} we can use the index of our cycle C_l to find the the deleted edges.

Now let's take a record $R \in \mathcal{R}_t$. Define the following sequence of words $R^* = (R^*_i)_{i \leq t}$ on the alphabet $\mathcal{A}^*\{0, 1, 2, ...\Delta - 1\}$. If R_i is empty, $R^*_i = 0$,. If $R_i - (k, l)$ then R^*_i is zero followed by $\theta^{-1}(l)$. A zero represents an edge coloured, and a positive integer represents an edge being uncoloured, so every word in the sequence begins with a zero, since an edge must be coloured to complete a cycle before the cycle is uncoloured. We then concatenate the sequence of words R^* into a single word $R^\bullet$, and finally convert this to a word R° on $\{0, 1\}$ by setting $R^\circ_i = 0$ if $R^\bullet_i = 0$ and $R^\circ_i = 1$ if $R^\bullet_i$ is a positive integer. Taking for example $\Delta = 4$ and $t = 10$, this should look like the following

$$\begin{aligned} R &= (\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,(3,4),\emptyset,\emptyset,\emptyset,(3,15)) \\ R^* &= (0,0,0,0,0,01211,0,0,0,03221) \\ R^\bullet &= (000000121100003221) \\ R^\circ &= (000000111100001111) \end{aligned}$$

Now we have that $R \to R^*$ is a bijection since θ^{-1} is bijective, and $R^* \to R^\bullet$ is injective, because each word in R^* begins with a zero and has no other zeroes. Then $R \to R^\bullet$ must also be an injection.

Recall that a partial Dyck word is a word w on $\{0,1\}$ such that any prefix, that is a subword $w' = w_1, ..w_k$ consisting of the first k letters in w,, has at least as many zeroes as ones. A Dyck word of length $2t$ is a partial Dyck word with exactly t zeroes and ones, and a *descent* in a Dyck word is a maximal sequence of consecutive ones. Our next lemma shows that R° is a partial Dyck word.

Lemma 4: *For any $R \in \mathcal{R}_t$, the word R° as defined above is a partial Dyck word with t zeroes and $t-r$ ones, where r is the number of coloured edges after step t. Moreover, all descents in R° and if every cycle in G has length at least $2l+1$ for some $l \geq 1$ then all descents in R° have length at least* $\max\{4, 2l\}$

Proof: Reading from left to right, every zero in R° represents an edge being coloured and every one represents an edge being uncoloured. In fact, since we uncolour all edges of a cycle except 2, each uncolouring event uncolours $2k-2$, meaning each word in R_i^* is either zero or becomes a subword zero followed by $2k-2$ ones, meaning it can be written 01^{2k-2} for $k \geq 1$, where $k=1$ means the edge was coloured without issue and $k > 1$ means $2k-2$ edges where uncoloured. Since of course we can not uncolour more edges than were originally coloured, any prefix must contain more zeroes than ones, meaning R° is a Dyck word. Assume now that all cycles in G have length at least $2l+1$. Then all even cycles have length at least $2l+2$. Since our algorithm forces even cycles to be even and have length at least 6, it follows that every descent in our Dyck word has length at least $\max(4, 2l)$.
Recall that $R \to R^\bullet$ as defined above is injective and each element in $R^\bullet$ takes a value $R_i^\bullet \in \{1, 2, ...\Delta - 1\}$. This means any R° with $t-r$ ones is the image of at most $(\Delta-1)^{t-r}$ record vectors R, in other words the preimage of R° under $R \to R^\circ$ has at most $(\Delta-1)^{t-r}$ vectors R. Let $\mathcal{R}_t^\circ = \{R^\circ | R \in \mathcal{R}_t\}$ be the set of partial Dyck words attainable from records $R \in \mathcal{R}_t$ under $R \to R^\circ$. So each R° has at most $(\Delta-1)^{t-r}$ choices for R, and using this and the fact that R° is a partial Dyck word with no more ones than zeroes, we know that

$$|\mathcal{R}_t| \leq (\Delta-1)^t |\mathcal{R}_t^\circ|$$

Subbing this into our bound in lemma 3 then gives us our next lemma

Lemma 5: $|\mathcal{F}_t| \leq (K+1)^m (\Delta-1)^t |\mathcal{R}_t^\circ|$

This means bounding $|\mathcal{F}_t|$ becomes the more manageable problem of bounding the number of partial *Dyck* words $|\mathcal{R}_t^\circ|$. In fact we can make this easier, since we can show that Dyck words and partial Dyck words are almost equivalent as long as the difference r between ones and zeroes in the partial Dyck words is not too large (Note the similarity to Lovasz). As will be shown later, the difference r can be at most $m-1$ where m is the number of edges in G, since only $m-1$ edges can be coloured without terminating the algorithm.

Since "true" Dyck words are easier to count, we use the following lemma to bound the number of partial Dyck words by the number of Dyck words of a slight longer length.

Lemma 6: *Let t and $r \leq t$ be integers, and let $E \neq \{1\}$ be a nonempty set of positive integers. Denote by $C_{t,r,E}$ be the number of partial Dyck words having t zeroes, $t-r$ ones and descents having lengths in E. Similarly denote by $C_{t,E}$ the number of Dyck words having t zeroes and ones and lengths in E. Then $C_{t,r,E} \leq C_{t+r(s-1)}$, where $s = \min\{E \setminus \{1\}\}$.*

Proof: We can define an injective function from the set $\mathcal{D}_{t,r,E}$ of partial Dyck words with t zeroes and $t-r$ ones to $\mathcal{D}_{t+r(s-1),E}$ of Dyck words with $t+r(s-1)$ ones and zeroes. Define $\psi : \mathcal{D}_{t,r,E} \to \mathcal{D}_{t+r(s-1),E}$ as follows; for each word in $\mathcal{D}_{t,r,E}$ we add $r(s-1)$ zeroes followed by rs ones. Then we are left with a Dyck word of length $2(t+r(s-1))$. Since ψ is an injection, we have $|\mathcal{D}_{t,r,E}| \leq |\mathcal{D}_{t+r(s-1),E}|$, or in other words $C_{t,r,E} \leq C_{t+r(s-1)}$.
Now we are left with the much easier problem of determining asymptotics for $C_{t,E}$, since we are interested in how the sequence $(C_{t,E})_t$ grows in t. We can do this by finding a bijection between Dyck words and better-known structures. Our next lemma gives a bijection between Dyck words and Rooted Plane Trees.

Lemma 7: *The number $C_{t,E}$ of Dyck words of legnth $2t$ and all descents in E is equal to the number of rooted plane trees on $t+1$ vertices such that the degree of each vertex is in E.*

Proof: We show that there exists a bijection between the following three objects

1. Rooted plane trees on $t+1$ vertices such that the degree of each vertex is in $E \cup \{0\}$;
2. Dyck words of length $2t$, in which the length of any maximal consecutive sequence is in E;
3. Dyck words of length $2t$ such that the length of each descent is in E.

First we show there is a bijection between 1 and 2. By definition of a rooted plane tree, there is one vertex designated as the root, and each vertex is given an index (i,j) where i is the depth, denoting the "generation" or distance from the root, and j is the jth vertex in the given generation, according to some ordering. A depth-first search is an algorithm that runs over the vertices by following each path to its end before moving on to another. That is, it begins at the root $(0,0)$, progresses through $(i,0)$ until the end, say k, then moves on to $(k-1,1)$ etc. moving backwards up the subtrees of each branch beforing moving on to the next, as shown below (show diagram). We take a tree T as in 1. and through a Depth First search and define a word w as follows: For every vertex in T with i children we encounter we add i zeroes followed by a 1, ignoring the very last vertex in our search. This algorithm assigns a one and a zero to each vertex except the last so w has exactly t zeroes and t ones, and since it is a depth first search each vertex that is not at the end of a path is followed by one of its children, and because zero is assigned to a vertex when it's parent is encountered the zeroes are assigned first, meaning every prefix will have more zeroes than ones. We conlude w is a Dyck word and since the algorithm runs on all $t+1$ vertices save the last, it is a Dyck word of length $2t$. Moreover, since the degrees of T are in E, each maximal sequence of consecutive zeroes is in E.

Now we show that there is a bijection between 2. and 3.. We simply take the mirror of the word, that is reorder the elements from last to first, and then switch all the zeroes to ones. What remains is a Dyck word with all sequences of consecutive ones in E, i.e. with all descents in E.

Now that we have shown the above correspondence, we estimate $C_{t,E}$ by counting on rooted trees. In Esperet and Parreau's paper, this is done using a method from [2]. For more detail on this method we recommend this paper along with [5]. First, we denote by $X_E(z)$ the ordinary generating function associated with trees on $t+1$ vertices such that the degree of each vertex is in $E \cup \{0\}$. As defined in 4 the ordinary generating function for the number set $\mathcal{P}$ is

$$P(x) = \sum_{n=0}^{\infty} p_n x^n$$

Where p_n denotes the number of such trees on n nodes. By our previous lemma the number of trees with vertex degrees in $E \cup \{0\}$ on $t+1$ nodes is $C_{t,E}$ for each t . So using our generating function as defined above this becomes

$$X_E(z) = \sum_{t=0}^{\infty} C_{t,E} z^{t+1} = z \sum_{t=0}^{\infty} C_{t,E} z^t$$

However, a rooted tree as described above must be either the root, or the root together with a sequence of i smaller rooted plane trees, each with vertex degrees in E. The generating function above can then be written as a sum of generating functions of these smaller trees as follows

$$X_E(z) = \sum_{t=0}^{\infty} C_{t,E} z^{t+1} = z \sum_{t=0}^{\infty} C_{t,E} z^t = z(1 + \sum_{i \in E} X_E(z)^i)$$

If we let $\phi_E(x) = 1 + \sum_{i \in E} x^i$ this becomes

$$X_E(z) = z\phi_E(X_E(z))$$

We now use a theorem 5 from [2]:

Theorem 5, Drmota: *Let R denote the radius of convergence of $\phi(t)$ and suppose there exists τ with $0 < \tau < R$ that satisfies $\tau\phi'(\tau) = \phi(\tau)$. Set $d = gcd\{j > 0 | \phi_j > 0\}$. Then*

$$y_n = d\sqrt{\frac{\phi(\tau)}{2\phi''(\tau)}} \frac{\phi'(\tau)^n}{n^{\frac{3}{2}}}(1 + \mathcal{O}(n_{-1})) \qquad \textit{if } (n \equiv 1 \mod d)$$

and $y_n = 0$ if $n \not\equiv 1 \mod d$.

Here y_n denotes a weighted number of trees on n vertices. We can use this result to bound $C_{t,E}$. First we observe that for any nonempty set $E \neq \{1\}$, all the coefficients of ϕ_E are nonnegative and $\phi_E(x)$ is not linear in x. Our next lemma is a corollary of the theorem given above.

Lemma 8: *Let $E \neq 1$ be a nonempty set of nonnegative integers such that the equation $\phi_E(x) = x\phi'(x)$ has a solution $x = \tau$ with $0 < \tau < R$ where R is the radius of convergence of ϕ_E. Then τ is the unique solution of the equation in the open interval $(0, R)$. Moreover there is a constant c_E such that*

$$C_{t,E} \leq c_E \gamma^t t^{-\frac{3}{2}}$$

where $\gamma = \phi'_E(\tau) = \frac{\phi_E(\tau)}{\tau}$

We have at last obtained a bound on $C_{t,E}$ and therefore on $|\mathcal{F}_t|$, and we can now show the main result.

Theorem 1: *Let $l \geq 1$ be a fixed integer and let $k = \max(2, l)$. Then the polynomial $P(x) = (2k-3)x^{2k+2} + (1-2k)x^{2k} + x^4 - 2x^2 + 1$ has a unique root τ in the open interval $(0, 1)$. Moreover every graph with maximum degree Δ and girth at least $2l + 1$ has an acyclic edge colouring with at most $\lceil (2+\gamma)(\Delta-1) \rceil$ colours, where $\gamma = \frac{(\tau^k - \tau^2 + 1)}{\tau - \tau^3}$*

Proof: Recall E represents the length of descents, therefore the lengths of uncoloured cycles, and these must be even and have length at least $\max(4, 2l)$ (see Lemma 4), so we first let let $E = 2\mathbb{N} + 2k$. Then for ϕ defined as above we use the formula for the geometric series,

$$\begin{aligned}\phi_E(x) &= 1 + \sum_{i \in E} x^i \\ &= 1 + \sum_{i=k}^{\infty} x^{2i} \\ &= 1 + \frac{x^{2k}}{1 - x^2}\end{aligned}$$

noting the radius of convergence is 1
The derivative is then

$$\phi_E'(x) = \frac{2kx^{2k-1} - (2k-2)x^{2k+1}}{(1-x^2)^2}$$

and the characteristic equation is

$$\begin{aligned}\phi_E(x) - x\phi_E'(x) &= 0 \\ 1 + \frac{x^{2k}}{1-x^2} - x\frac{2kx^{2k-1} - (2k-2)x^{2k+1}}{(1-x^2)^2} &= 0\end{aligned}$$

which reduces to

$$(2k-3)x^{2k+2} + (1-2k)x^{2k} + x^4 - 2x^2 + 1 = 0$$

We recognise this is $P(x) = 0$. ϕ_E is $(0, 1)$ because we used the geometric series, and since $P(0) = 1$ and $P(1) = -2$ the polynomial has a root τ in the open interval $(0, 1)$. This is the unique root in $(0, 1)$ as outlined in Lemma 8.

Lemma 8 also gives us a constant c_E which we can use to bound the number of rooted plane trees $C_{t,E}$, and gives us an expression for γ in terms of τ

$$C_{t,E} \leq c_E \gamma^t t^{-\frac{3}{2}}$$

where

$$\gamma = \phi_E'()t = \frac{\phi_E'(\tau)}{\tau} = \frac{(\tau^{2k} - \tau^2 + 1)}{\tau - \tau^3}$$

Remember to prove the theorem we are trying to show that there exists an input vector $F \in \{1, 2, ...\gamma(\Delta-1)^t\}$ such that the algorithm terminates by time t, in which case an acyclic colouring has been found. This can be proven by showing that for t large enough the number of inputs which fail to terminate is strictly less than the number of inputs in total,

$$|\mathcal{F}_t| < |\mathcal{F}|$$

Now, for a graph on m edges, Lemma 5 states that the number of inputs is bounded by the number of possible colourings on m edges, times the number of partial Dyck words obtainable from our records as described in lemma 4.

$$|\mathcal{F}_t| \leq (\lceil (\gamma+2)(\Delta-1) \rceil + 1)^m (\Delta-1)^t |\mathcal{R}_t^\circ|$$

Now we note that for any records $R_1, R_2 \in \mathcal{R}_t$, the number of zeroes and ones in each prefix of R_1°, R_2° differ by at most $m-1$, because there are m edges in the graph so at most $m-1$ edges can be coloured at each step of the algorithm. Now by Lemma 4, any record R is transformed by $\mathcal{R}_t \to \mathcal{R}_t^\circ$ into a partial Dyck word with t zeroes and $t-r$ ones. Lemma 6 states that the number of these can be bounded by the number of dyck words as follows

$$C_{t,r,E} \leq C_{t+r(s-1)}, \qquad \text{where } s = \min\{E \setminus \{1\}\}$$

Now, recall that $E = 2\mathbb{N} + 2k$, so $s = \min E \setminus 1 = 2k$. moreover since r can take values only up to $m-1$, we can sum over the above bound for each r to conclude

$$|\mathcal{R}_t^\circ| \leq \sum_{r=1}^{m-1} C_{t+r(2k-1),E}$$

Using Lemma 8 this becomes

$$|\mathcal{R}_t^\circ| \leq \sum_{r=1}^{m-1} c_E \gamma^{t+r(2k-1)} (t + r(2k-1))^{-\frac{3}{2}}$$

If we take $c'_E = c_E/(\gamma^{2k-1} - 1)$, after some work this reduces to

$$|\mathcal{R}_t^\circ| \leq c'_E \gamma^{t+m(2k-1)} (t)^{-\frac{3}{2}}$$

Subbing this into our bound for $\mathcal{F}_t$ we have

$$|\mathcal{F}_t| \leq (\lceil (\gamma+2)(\Delta-1) \rceil + 1)^m (\Delta-1)^t c'_E \gamma^{t+m(2k-1)} t^{-\frac{3}{2}}$$

We can now consolidate terms that are constant in t into a single constant M

$$|\mathcal{F}_t| \leq M (\Delta-1)^t \gamma^t t^{-\frac{3}{2}}$$

and note that the ceiling function gives us $\lceil \gamma(\Delta-1) \rceil \geq \gamma(\Delta-1)$, so we have

$$\frac{|\mathcal{F}_t|}{\lceil \gamma(\Delta-1) \rceil^t} \leq \frac{M \gamma^t (\Delta-1)^t \gamma^t t^{-\frac{3}{2}}}{\lceil \gamma(\Delta-1) \rceil^t}$$

Observe that the expression on the right is decreasing in t this means that asymptotically we have

$$\lim_{t \to \infty} \frac{|\mathcal{F}_t|}{\lceil \gamma(\Delta-1) \rceil^t} = 0$$

So there is some t' such that for all $t > t'$ we have

$$|\mathcal{F}_t| < \lceil \gamma(\Delta-1) \rceil^t$$

Now recall that our input F is in $\{1, 2, ... \lceil \gamma(\Delta-1) \rceil\}^t$, so there are $\lceil \gamma(\Delta-1) \rceil^t$ choices for F, and

$$|\mathcal{F}_t| < |\mathcal{F}|$$

We conclude that the set of input vectors such that our algorithm fails to find an acyclic colouring less than the set of all possible inputs, which directly implies there is an input which achieves a colouring in $K = \lceil (\gamma+2)(\Delta-1) \rceil$ colours.

4.4 Bounds on $a'(G)$

As a final corollary we show that the acyclic chromatic number $a'(G)$ can be bounded above by $4(\Delta - 1)$. Note that the minimum girth of a graph of interest is three, since this is the shortest possible cycle length. For a graph of this type $k = \max(2, l) = 2$, so E becomes $2\mathbb{N} + 4$. Subbing $k = 2$ characteristic polynomial

$$\begin{aligned} P(x) &= (2k-3)x^{2k+2} + (1-2k)x^{2k} + x^4 - 2x^2 + 1 \\ &= x^6 - 2x^4 - 2x^2 + 1 \end{aligned}$$

The unique root of $P(x) = 0$ in the radius of convergence $(0, 1)$ is then

$$\tau = \frac{\sqrt{5} - 1}{2}$$

and using the formula for γ we find

$$\begin{aligned} \gamma &= \frac{(\tau^4 - \tau^2 + 1)}{\tau - \tau^3} \\ &= 2 \end{aligned}$$

subbing this into K we find

$$\begin{aligned} K &= \lceil (\gamma + 2)(\Delta - 1) \rceil \\ &= 4(\Delta - 1) \end{aligned}$$

and since any graph has girth at least three, this holds for all G. Note that if the girth is known this bound can be improved.

5 Discussion and Conclusion

The method is versatile, being applicable to any case where the local lemma applies, as well as offering a strategy for Las Vegas algorithms in general. As we hope our examples have shown, this versatility extends not only as far as applications but in the strategies themselves. Computing the entropy itself is rarely necessary (or possible!) and the heart of the method lies in the objects which are randomly encountered as, at each step, our algorithm makes a new choice. The method encourages us to think abut the true nature of the dependencies we are mapping without being bound to whichever representation we use at first, and to nimbly switch between different structures which encode the same information. For example, in the proof of the local lemma we used randomly generated witness trees to limit the expected runtime, in our treatment of acyclic colouring we defined Dyck words using the record to bound the size of non-successful histories, and other papers have even more varied tricks to find how much entropy we read from our algorithm. Indeed, each application requires a unique solution and requires us to think about the problem in terms of the steps our algorithm can take. Using objects in this context helps to gain a stronger intuition about the relationships they represent.

This technical breadth is matched by a theoretical depth afforded by the idea of entropy compression itself. All of these varied processes produce a flow of information as a byproduct, and even when it is not explicitly read it can be a determinant of the process itself. As in many particularly elegant results in mathematics, the immediate is determined by the abstract.

6 Acknowledgements

I would like to especially thank my supervisor Tobias Muller for his guidance, his patience and his interest. Many thanks also to my second assessor Pieter Trapman for his time and help, and thanks to Juan Peypouquet for answering my many questions.

References

[1] Noga Alon, Colin McDiarmid, and Bruce Reed. "Acyclic coloring of graphs". In: *Random Structures & Algorithms* 2.3 (1991), pp. 277–288.

[2] Michael Drmota. "Combinatorics and asymptotics on trees". In: *Cubo Journal*. Citeseer. 2004.

[3] Paul Erdős and László Lovász. "Problems and results on 3-chromatic hypergraphs and some related questions". In: *Colloquia Mathematica Societatis Janos Bolyai 10. Infinite and Finite Sets, Keszthely (Hungary)*. Citeseer. 1973.

[4] Louis Esperet and Aline Parreau. "Acyclic edge-coloring using entropy compression". In: *European Journal of Combinatorics* 34.6 (2013), pp. 1019–1027.

[5] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009. DOI: 10.1017/CBO9780511801655.

[6] Robin A Moser and Gábor Tardos. "A constructive proof of the general Lovász local lemma". In: *Journal of the ACM (JACM)* 57.2 (2010), pp. 1–15.

[7] Robin A. Moser. "A constructive proof of the Lovász local lemma". In: *In STOC '09: Proceedings of the 41st annual ACM Symposium on Theory of Computing*. 2009, pp. 343–350.

[8] Sloan Nietert. *Moser's Algorithm and the Lovász Local Lemma*. URL: https://www.cs.cornell.edu/~nietert/blog/2019/12/26/mosers-algorithm-and-the-lov%5C%C3%5C%A1sz-local-lemma/.

[9] Joel Spencer. "Asymptotic lower bounds for Ramsey functions". In: *Discrete Mathematics* 20 (1977), pp. 69–76. ISSN: 0012-365X. DOI: https://doi.org/10.1016/0012-365X(77)90044-9. URL: https://www.sciencedirect.com/science/article/pii/0012365X77900449.

[10] Terence Tao. *Moser's entropy compression argument*. Aug. 2009. URL: https://terrytao.wordpress.com/2009/08/05/mosers-entropy-compression-argument/.