Electronic Colloquium on Computational Complexity, Report No. 164 (2019)

Improved bounds for perfect sampling of k-colorings in graphs

Siddharth Bhandari* Sayantan Chakraborty*

November 6, 2019

Abstract

We present a randomized algorithm that takes as input an undirected n-vertex graph G with maximum degree Δ and an integer $k > 3\Delta$, and returns a random proper k-coloring of G. The distribution of the coloring is *perfectly* uniform over the set of all proper k-colorings; the expected running time of the algorithm is $\text{poly}(k,n) = \tilde{O}(n\Delta^2 \cdot \log(k))$. This improves upon a result of Huber (STOC 1998) who obtained polynomial time perfect sampling algorithm for $k > \Delta^2 + 2\Delta$. Prior to our work, no algorithm with expected running time $\text{poly}(k,n)$ was known to guarantee perfectly sampling for $\Delta = \omega(1)$ and for any $k \leq \Delta^2 + 2\Delta$.

Our algorithm (like several other perfect sampling algorithms including Huber's) is based on the Coupling from the Past method. Inspired by the *bounding chain* approach pioneered independently by Häggström & Nelander (Scand. J. Statist., 1999) and Huber (STOC 1998), our algorithm is based on a novel bounding chain for the coloring problem.

1 Introduction

We consider the problem of randomly sampling a proper coloring in graphs. The input to the problem is a graph G and an integer k. Our goal is to generate a proper k-coloring (using colors from the set $[k] = \{1, 2, \ldots, k\}$ to color the vertices of G), uniformly at random from the set of all proper colorings.[1] The problem of sampling k-colorings has several implications in theoretical computer science and statistical mechanics. For example, Jerrum, Valiant and Vazirani [JVV86] show that from an almost uniform sampler for proper k-colorings of G, one can obtain a Fully Polynomial Randomized Approximation Scheme (FPRAS) for counting the number of such colorings. In statistical mechanics, sampling proper colorings is central to simulation based studies of phase transitions and correlation decay (see, e.g., the paper of Martinelli and Olivieri [MO94]).

The problem is computationally tractable if we are allowed significantly more colors than the maximum degree Δ of the graph. The more colors we are allowed, the easier it appears to be to produce a random k-coloring. Indeed, if k is much smaller than Δ, it is NP-hard to even determine whether a valid k-coloring exists [GJS76]. Sampling algorithms, therefore, typically require a lower

*Tata Institute of Fundamental Research, Mumbai. email: {siddharth.bhandari,sayantan.chakraborty}@tifr.res.in

A preliminary version of this paper (arXiv:1909.10323v1) proved a weaker result that achieves expected polynomial time when $k > 2e\Delta^2/\ln(\Delta)$.

[1] Throughout the paper by k-colorings we refer to proper k-colorings.

ISSN 1433-8092

bound on k in terms of Δ in order to guarantee efficiency. There has been a steady stream of works that have progressively reduced the lower bound on k in terms of Δ.

Most works in this line of research focus on producing *approximately* uniform samples, for approximate solutions often suffice in applications. In this setting, the input to the problem consists of an undirected graph G with n vertices and maximum degree Δ, a number k and a parameter $\varepsilon \in (0,1)$. The goal is to generate a proper k-coloring whose distribution is within ε (in total variational distance) of the uniform distribution on the set of all proper k-colorings of G. Let $k_+(\Delta)$ be the smallest integer k^* such that for all integers $k > k^*$ there is such a sampling algorithm running in expected time $\text{poly}(k, n, \log(1/\varepsilon))$ for all $\varepsilon > 0$. By showing that the Markov chain based on Glauber Dynamics mixes fast whenever $k > 2\Delta$, Jerrum [Jer95] established that $k_+(\Delta) \leq 2\Delta$. Similar results appeared in the statistical physics literature (see Salas and Sokal [SS97]); also, the path coupling approach developed by Bubley and Dyer [BD97] can be used to provide a very appealing alternate justification for Jerrum's result. Subsequent works obtained better upper bounds on $k_+(\Delta)$. Vigoda [Vig00] provided a better analysis of Glauber dynamics (by relating it to a different Markov chain based on *flip dynamics*) and concluded that $k_+(\Delta) \leq \frac{11}{6}\Delta$; recently, Chen, Delcourt, Moitra, Perarnau and Postle [CDM+19] showed that $k_+(\Delta) \leq (\frac{11}{6} - \delta)\Delta$ (for a positive $\delta \sim 10^{-4}$). Even better upper bounds are known for certain special classes of graphs. For graphs with girth at least 9, Hayes and Vigoda [HV03] showed that for all $\delta > 0$, we have $k_+(\Delta) \leq (1+\delta)\Delta$ provided $\Delta \geq c_\delta \ln n$ (where c_δ is a constant depending on δ); for graphs of girth at least 6 and large enough Δ, Dyer, Frieze, Hayes and Vigoda [DFHV04] showed that $k_+(\Delta) \leq 1.49\Delta$.

1.1 Perfect sampling

The algorithms described above produce samples that are only approximately uniform. The variation from uniformity can be reduced by allowing the algorithm to run longer, but it cannot be made zero; that is, these methods do not yield perfectly uniform samples. Apart from its independent theoretical appeal, perfect sampling has some advantages over approximate sampling. It potentially yields FPRASs with smaller expected running time [Hub98, Theorem 7], because unlike with approximate sampling algorithms there is no need to ensure that the output distribution of the algorithm is sufficiently close to the target distribution in total variation distance. Moreover, perfect sampling algorithms are typically designed in such a way that the output produced when the algorithm stops is guaranteed to be uniform. One might be unable to formally guarantee that the expected running time is small; yet the quality of the output is never in question. In contrast, for efficient algorithms for approximate sampling, the running time may be bounded, but in the absence of guarantees (on the mixing time, say) the output distribution may be far away from the target distribution, thereby rendering the output unreliable for statistical applications.

The intriguing fact that perfect sampling is in general possible using Markov chains was established by Propp and Wilson [PW96], in a seminal paper which introduced the technique of *coupling from the past (CFTP)* to generate perfectly uniform samples; Levin, Peres and Wilmer [LPW06, Section 22.1] point out that ideas that underlie CFTP can be traced back to the 1960s. This paradigm has been applied to the problem of perfectly sampling k-colorings. However, in contrast to the the best bounds on $k_+(\Delta)$, which grow only linearly in Δ, the upper bounds on k for perfect sampling are less impressive. To better describe and compare these results, let us define $k_0(\Delta)$ to be the minimum integer k^* such that there is a randomized algorithm that, given an n-vertex graph of maximum degree Δ and integer $k > k^*$, produces a perfectly uniform proper k-coloring of G

in expected time $\text{poly}(k, n)$. By applying CFTP, Huber [Hub98, Hub04], showed that $k_0(\Delta) \leq \min\left(\Delta^2 + 2\Delta + 1, \frac{\Delta \ln n}{\ln \ln n}\right)$. Recently, Feng, Guo and Yin [FGY19, Condition 2.4] showed that Huber's result can be improved to obtain an expected polynomial time perfect sampling algorithm when $k > \Delta^2 - \Delta + 3$ for the setting when $\Delta = O(1)$ (Their algorithm requires time $n^{\text{poly}(\Delta)}$ and is thus not polynomial when $\Delta = \omega(1)$). Note that the upper bounds on $k_0(\Delta)$ obtained in these works is quadratic in Δ (in contrast to the linear upper bounds for approximate sampling).

Another paradigm for perfect sampling, related to the Moser-Tardos framework for algorithmic versions of the Lovász local lemma, and also to the celebrated cycle-popping algorithm of Wilson for sampling uniformly random spanning trees, has recently been proposed by Guo, Jerrum and Liu [GJL17]. However, it turns out that that when this framework is applied to the problem of sampling k-colorings, it degenerates into usual rejection sampling: one samples a uniformly randomly coloring (including improper colorings), accepts if the coloring is proper, and rejects the current coloring and repeats otherwise. The expected running time of such a procedure is proportional to the inverse of the fraction of proper colorings among all colorings, and hence cannot in general be bounded by a polynomial in the size of the graph.

Hence the question remains: can k-colorings be efficiently and perfectly sampled when k is a constant times Δ? It was observed that such an improvement can be obtained if one relaxes the (expected) running time to be polynomial in only n (and not in Δ and k). To state these results, let us define the relaxed version $\widetilde{k_0}(\Delta)$ as follows: $\widetilde{k_0}(\Delta)$ is the minimum integer k^* such that there is a randomized algorithm that, given an n-vertex graph of maximum degree Δ and integer $k > k^*$, produces a perfectly uniform proper k-coloring of G in expected time $\text{poly}_{\Delta,k}(n)$ (i.e., the dependence on n is polynomial, but the dependence on Δ and k can be arbitrary). A general method for perfect sampling based on approximate counting was suggested by Jerrum, Valiant and Vazirani [JVV86, Thm 3.3]; that is, using an efficient algorithm for deterministically approximately counting the number of k-colorings, one can sample perfectly. This approach when used together with the deterministic approximate counting algorithm of Gamarnik and Katz [GK06], yields $\widetilde{k_0}(\Delta) \leq 2.78\Delta$ for triangle free graphs; approximate counting algorithms in subsequent works due to Lu & Yin [LY13], and Liu, Sinclair & Srivastava [LSS19], yield $\widetilde{k_0}(\Delta) \leq 2.58\Delta$ and $\widetilde{k_0}(\Delta) \leq 2\Delta$, respectively. The running time of these algorithms has the form $O(n^{f(k,\Delta)})$ (for instance in [LSS19] the exponent contains an $\exp(\Delta)$ term). Another point to be noted is that these deterministic approximate counters are based on decay of correlations or the so-called 'polynomial interpolation' method of Barvinok [Bar16], which are not as simple as the Markov chain based algorithms (e.g., the CFTP based algorithms of Huber and in this paper). For instance, the MC based algorithms offer an appealing combinatorial explanation of the number of colors required for 'mixing' to take place unlike the other methods. None of these algorithms yield truly linear bounds on $k_0(\Delta)$.

In particular, it was open if there exist efficient randomized algorithms that can be guaranteed to efficiently sample perfectly when $k < \Delta^2 + 2\Delta$ (and remains polynomial even when $\Delta = \omega(1)$). Our work answers this question in the affirmative.

1.2 Our contribution

Theorem 1.1 (Main result)**.** $k_0(\Delta) \leq 3\Delta$. *In particular, there is a randomized algorithm which we call* PERFECTSAMPLER *(Algorithm 1) based on CFTP that given an n-vertex graph $G = (V, E)$ of maximum degree Δ and $k > 3\Delta$, returns a uniformly random proper k-coloring of G. The algorithm uses fair unbiased*

coin tosses (bias $1/2$*), and stops in expected time* $O(n \log^2(n) \cdot \Delta^2 \cdot \log k)$.

Our result is based on Coupling From the Past (CFTP). In the rest of this section, we briefly review CFTP as it is applied to the problem of k-coloring, and describe its efficient implementation using the Bounding Chain method roughly along the lines of Huber [Hub98, Hub04]. We then describe the key idea that allows us to improve the upper bound on $k_0(\Delta)$ to 3Δ.

Consider the standard Markov chain for k-coloring that evolves based on the Glauber Dynamics: in each step a random vertex v is chosen and its color is replaced by a uniformly chosen color not currently used by any of its neighbors. The standard CFTP algorithm based on this Markov chain, assumes that we generate a sequence of random variables, $(v_{-1}, \sigma_{-1}), (v_{-2}, \sigma_{-2}), \ldots$, where v_i is a random vertex in $V(G)$ and σ_i is a random permutation of the set of colors $[k]$, chosen uniformly and independently. For $i = -1, -2, \ldots$, let U_i be the operation on k-colorings that performs the following update based on the pair (v_i, σ_i). Given a proper k-coloring $\chi : V \to [k]$ and the pair (v_i, σ_i), let $U_i(\chi)$ be the coloring χ' defined as follows: $\chi'(v_i)$ is the first color in σ_i that is not in $\chi(N(v_i))$ and for vertices $w \neq v_i$, $\chi'(w) = \chi(w)$. Note that U_i maps proper k-colorings to proper k-colorings. The CFTP algorithm is based on the following principle. Let t be an integer such that $U_{-1} \circ U_{-2} \circ \cdots U_t$ is a constant function, that is, this sequence of updates applied to every proper k-coloring results in the same coloring, say χ_f. The algorithm outputs χ_f. Note that this output does not depend on the choice of t. For example, we could run through $i = -1, -2, \ldots$ until the first index t when $U_{-1} \circ U_{-2} \circ \cdots U_t$ becomes a constant function, and output the unique k-coloring in its image. It is well known that if $k > \Delta + 1$, then with probability 1 such a $t < \infty$ exists, and χ_f is uniformly distributed in the set of all colorings. We may, alternatively, think of the update operations defined above as a method for coupling several Markov chains, each starting at a different k-coloring and evolving according to the Glauber dynamics.

The randomized algorithm as stated above is not efficient. The number of starting states for the chains grows exponentially with n, and the time taken to keep track of the updates on each will be prohibitively large. To keep the computation tractable, Huber employs the Bounding Chain (BC) Method (pioneered by him in the context of coloring and also independently by Häggström & Nelander [HN99]). In the Bounding Chain method, instead of keeping track of the various states that are reached after each update operation precisely, we maintain an upper bound: a list of states, which contains all the states that could potentially be reached. In fact, this upper bound for the state reached after update U_{j-1} has been applied will have the special form: $\prod_{v \in G} L_j(v)$, where $L_j(v) \subseteq [k]$. That is, when simulating the actions of successive updates $U_{j-1} \circ \cdots \circ U_t$, we do not explicitly maintain the correlations across vertices, but rather just a list $L_j(v)$ that includes all colors that vertex v can take in any k-coloring reached by performing these updates starting from any initial k-coloring. Note in particular, that our choice of t will be good if we can ensure that $|L_0(v)| = 1$ for all vertices v; for, then we know that $U_{-1} \circ U_{-2} \circ \cdots \circ U_t$ is a constant function (on the space of k-colorings), and χ_f is the unique coloring in $\prod_v L_0(v)$.

We are now in a position to give a high-level description of Huber's BC method [Hub98, Hub04]. After a short initial phase of updates which act as warm-up, Huber maintains the invariant that $|L_j(v)| \leq \Delta + 1$ for all vertices v. To measure progress towards the goal that $|L_0(v)| = 1$ for all vertices v, let us define W_j to be the number of vertices v such that $|L_j(v)| = 1$. Hence, we want $W_0 = n$. Now, suppose that at time j we have the update operation U_j given by (v, σ). Consider L_j such that $|L_j(v)| \leq \Delta + 1$ and let $S_{L_j}(v)$ be the union of colors present in the lists of neighbors of v. Then, Huber sets $L_{j+1}(v) = \{\sigma(1)\}$ if $\sigma(1) \notin S_{L_j}(v)$; otherwise $L_{j+1}(v) = \{\sigma(1), \ldots, \sigma(\Delta + 1)\}$. For all $w \neq v$, $L_{j+1}(w) = L_j(w)$. Notice that if $\chi \in \prod_{w \in G} L_j(w)$ then $U_j(\chi) \in \prod_{w \in G} L_{j+1}(w)$ as v defi-

nitely finds an available color in $\{\sigma(1), \ldots, \sigma(\Delta+1)\}$ and hence $(U_j(\chi))(v) \in \{\sigma(1), \ldots, \sigma(\Delta+1)\}$. Hence, we make progress (towards our goal of $W_0 = n$) whenever $\sigma(1) \notin S_{L_j}(v)$ and suffer a loss otherwise. To be able to have a non-trivial probability of making progress we need that $k > |S_{L_j}(v)|$ and this is ensured by having $k > \Delta^2 + \Delta$. However, W_j evolves as a random walk on $\{0, \ldots, n\}$ (with n as absorbing state) and to have sufficient drift to the right we require an extra margin of Δ in k and hence Huber assumes $k > \Delta^2 + 2\Delta$. It then follows that in expected time $\mathrm{poly}(n, k)$ one can find the starting time t so that $|L_0(v)| = 1$ for all vertices v and hence $U_{-1} \circ U_{-2} \circ \cdots \circ U_t$ is a constant function. For now, we only note, roughly, that for this method to succeed, k must be larger than the product of the maximum degree (Δ) and the upper bound on the size of $L_j(v)$ that we can ensure plus an extra margin of Δ.

We improve upon this using a better implementation of the Bounding Chain Method. After a short initial warm-up phase of updates, the set of colors $L_i(v)$ in our implementation will be of size at most two; this will allow us to ensure perfect sampling as long as $k > 3\Delta$. The correctness of our algorithm will still rely on the Markov chain based on Glauber Dynamics described earlier. However, we depart substantially from earlier works in designing our coupling. Recall that a sequence of random update operations $U_{-1}, U_{-2}, \ldots$, need to be designed to specify this coupling. In Huber's coupling the U_i's were independently and identically distributed (based on independent choices of the pairs (v_i, σ_i)). Our new update operations will not be chosen independently but will have a rather special distribution. This distribution is designed keeping in view our goal of restricting the bounding lists $L_t(v)$ to size at most two, and is best understood in the context of the evolution of these lists in our implementation of the Bounding Chain Method, which we describe in the subsequent sections. For now, we outline the main properties of this distribution. For $0 > i > j$, let $U[i, j] := (U_i, U_{i-1}, \ldots, U_j)$ and let $U(i, j) := U_i \circ U_{i-1} \circ \cdots \circ U_j$. (Note $U[i, j]$ refers to the set of random choices that describe the $(i - j)$ update functions while $U(i, j)$ refers to the composed update function.)

Lemma 1.2. *Let G be an n-vertex graph with maximum degree Δ. Let $k > 3\Delta$. Then, there is a positive integer T which is $\mathrm{poly}(k, n)$, a joint distribution $\mathcal{D}$ for T updates, $U[-1, -T]$, and a predicate Φ on the support of $\mathcal{D}$, satisfying the following conditions.*

(a) A sample from $\mathcal{D}$ can be generated and the predicate Φ can be computed in time $\mathrm{poly}(n, k)$; further, each update instruction U_i is efficient, i.e., given a k-coloring χ, $U_i(\chi)$ can be computed in time $\mathrm{poly}(k, n)$;

(b) If Z is a uniformly generated proper k-coloring of G and $U(-1, -T)(Z)$ is picked according to $\mathcal{D}$ independently of Z, then $U(-1, -T)(Z)$ is a uniformly distributed proper k-coloring;

(c) If $\Phi(U[-1, -T]) = \mathsf{true}$, then $U(-1, -T)$ is a constant function (on the set of proper k-colorings), that is, $|\{U(-1, -T)(\chi) : \chi \text{ is a proper } k\text{-coloring}\}| = 1$;

(d) $\Pr_{\mathcal{D}}[\Phi(U[-1, -T]) = \mathsf{true}] \geq \frac{1}{2}$.

Remark: The update operations (which act on an exponentially large set) need to be represented succinctly for our algorithm to be efficient. Each operation will be encoded succinctly by tuples. (For example, in the discussion above the tuple (v_i, σ_i) can be thought of as the encoding of the update operation U_i.) The encoding we use is described below. Thus, in part Lemma 1.2 (a), when we need to generate a sample from $\mathcal{D}$, we actually generate the sequence of T tuples corresponding

to the update operations. Similarly, the predicate Φ is expected to take as argument a sequence of tuples and efficiently evaluate to true or false; further when Φ = true we can efficiently compute the (unique) image of $U(-1,-T)$ from the tuples. We will ensure that the decoding is efficient: given a tuple that represents an update operation U and a proper k-coloring χ, the coloring $U(\chi)$ can be computed efficiently.

We then have the following natural randomized algorithm for perfectly sampling k-colorings.

Algorithm 1: PERFECTSAMPLER

```
for i = 0, 1, 2, ..., do
    Generate U[-iT - 1, -(i+1)T] according to D ;
    if Φ(U[-iT - 1, -(i+1)T]) = true then
        Output the unique k-coloring in the image of U(-1, -(i+1)T) and STOP;
    end if
end for
```

In other words, let i be the first index in Algorithm 1 such that we find $\Phi(U(-iT-1,-(i+1)T))$ = true. So, we know that $|U(-iT-1,-(i+1)T)(v)| = 1$ for all $v \in V$. Now, we update this unique coloring with $U(-iT,-1)$ and output the updated coloring. The correctness of the algorithm and that it runs in expected time poly(n,k) follow immediately from Lemma 1.2; in particular, this justifies Theorem 1.1 barring the expected running time. In the rest of this introduction, describe the distribution $\mathcal{D}$ and outline our proof of the lemma

Representation of update operations: Our definitions will be inspired by the Bounding Chain method, which maintains the list of colors that a vertex can potentially take. To make this precise, we need a definition. By a *bounding list* we mean a list of the form $L = (L(v) : v \in L)$, where each $L(v)$ is a set of colors. We say that a k-coloring χ is compatible with L, and write $\chi \sim L$, if $\chi(v) \in L(v)$ for all v, that is, if $\chi \in \prod_v L(v)$. We are now in a position to describe the representation we use. Each update operation will be associated with a 5-tuple of the form $\alpha = (v, \tau, L, L', M)$, where v is a vertex, $\tau \in [0,1]$, and L and L' are bounding lists, and M is a sequence of at most $\Delta+1$ distinct colors. We refer to the update operation associated with α as U_α. Thus, the distribution of $U[-1,-T]$ will be specified by providing a randomized algorithm for generating the corresponding sequence of tuples $\alpha[-1,-T]$ and letting $U_t = U_{\alpha_t}$. We now describe some of the essential properties of this sequence.

Fix $t \in \{-T,\ldots,-1\}$. Suppose $\alpha[t-1,-T]$ have been generated. Now, consider $\alpha_t = (v_t, \tau_t, L_t, L'_t, M_t)$. We will ensure that the following conditions hold.

[C1] The random vertex v_t is independent of $\alpha[t-1,-T]$. In Huber's chain, v_t was actually uniformly distributed; we will not be able to ensure that; in fact, some our vertices will be determined by the index t (the current time step); for example, v_{-T} will be a fixed vertex of the graph, not a random vertex.

[C2] The distribution of α_t must implement the Glauber Dynamics at vertex v in the following sense. Condition on $\alpha[t-1,-T]$ and v_t (the first component of α_t). Fix a coloring χ in the image of $U(-T,t-1)$ (note that this operator is determined completely by $\alpha[t-1,-T]$, which we have conditioned on). Now, we require that $\chi' = U_t(\chi)$ has the following distribution: $\chi'(w) = \chi(w)$, for all $w \neq v_t$ and $\chi'(v_t)$ is uniformly distributed in the set of colors $[k] \setminus$

$\chi(N(v_t))$. If this condition is satisfied, then we say that α_t satisfies GLAUBER DYNAMICS(χ, v_t). Note that this will ensure Lemma 1.2 (b).

[C3] The lists L_t imposes a certain restriction on the domain of U_t: U_t will be defined only on colorings $\chi \sim L_t$. Thus L_t represents a *precodition* for U_t to be applicable. Similarly, L'_t represents a *postcondition*: if $\chi \sim L_t$, then $U_t(\chi) \sim L'_t$. We will, therefore, take care that in our sequence $L'_t \subseteq L_{t+1}$. In particular, L_{-T} will be $([k])^V$. If the above discipline concerning precondition and post-condition is maintained, then for the image of $U(-T,-1)$ to be a singleton, it is enough that $|L'_{-1}(v)| = |L_0(v)| = 1$ for all $v \in V$. Indeed, our predicate Φ will verify this by examining α_{-1}, and to show Lemma 1.2(d), we will show that this condition holds with probability at least $\frac{1}{2}$.

The key ideas: The above discussion describes some of the conditions that our random sequence of tuples $\alpha[-1,-T]$ must satisfy, but we have not yet formally described the translation (decoding) of α_t to obtain U_t, nor the distribution of $\alpha[-1,-T]$. We will now informally describe these. However, our description will not exactly match the more formal one present in Section 2; but it will let us motivate our definitions, and also throw light on how the new method makes do with fewer colors than Huber's method.

We recall that W_t denotes the number of vertices whose list sizes have become one at time t. Hence, our goal is to ensure that $W_0 = n$ with probability at least 1/2. W_t will evolve as a random walk on $0, 1, \ldots, n$ (with n as absorbing state). As pointed out above in the discussion of Huber's method, to have sufficient drift to the right, k must be larger than the product of the maximum degree (Δ) and the upper bound on the size of $L_t(v)$ for all $v \in V$ that we can ensure, plus an extra margin of Δ. Huber's method first ensures that for all vertices the lists are of size at most $\Delta + 1$ colors. Then since a vertex has only Δ neighbors, Huber's algorithm is able to make progress provided $k > \Delta^2 + 2\Delta$. Hence, if we are to have sufficient drift to the right with $k > 3\Delta$ we need an upper bound of 2 on the size of $L_t(v)$. To achieve this we will design a *Warm-up* phase whence for all v, $|L_t(v)| \leq 2$. Then, we will have another phase where will maintain $|L_t(v)| \leq 2$ while having a non-trivial probability of $|L_t(v)| = 1$.

The method of bounding chains actually allows us to make progress if we can find enough colors for a vertex that will not conflict with the colors in the lists of its neighbors. That is, somehow we need to ensure that $\bigcup_{w \in N(v)} L(w)$ is small. What might allow us to bound this size by $O(\Delta)$? There are two possibilities. First, we might be able to somehow ensure that the $L(w)$'s are actually much smaller than $\Delta + 1$, maybe even a constant. Second, even if we fail to ensure that $L(w)$ is individually small, can we somehow ensure that their union is small, say $O(\Delta)$. We, in fact, use both these ideas; the second idea will be used first!

We will start with an L where $L(v) = [k]$ for all $v \in V$. Let us describe a sequence of update operations that ensures that the union of the lists of the neighbors of every vertex is small. In fact, we will ensure that all $L(v)$ will be of the form, $[\Delta] \cup \{c_v\}$. Note that this ensures that the union of any set of r lists has at most $\Delta + r$ colors. The update operations that achieve this are as follows. The first update operation is generated by picking a random color in $c_1 \in [k] \setminus [\Delta]$ and an ordering σ of the colors in $[\Delta]$. Fix a coloring χ and a vertex v whose color we wish to update. Now, we have a choice of either accepting c_1 or the first vertex in the ordering that is not in $\chi(N(v))$. We would like to update v's color to a uniform color outside $\chi(N(v))$ (for we have promised to be faithful to Glauber dynamics). One verifies that by accepting c_1 or c_2 with appropriate probabilities (depending on χ), we can ensure that v's color is updated using a

uniformly chosen color. The ordering σ will actually be recorded in the component M of α. The choice between c_1 and c_2 will be made using the random real number τ, which is also part of α. Thus α completely determines the update operation, as required by our formalism. Once this operation is performed, we can be certain that v's new color is confined to $[\Delta] \cup \{c_1\}$; the actual color will depend on the original coloring. That is, for the corresponding $\alpha = (v, \tau, L, L', M)$, we will have $L(v) = [k]$ and $L'(v) = [\Delta] \cup \{c_1\}$. After a sequence of n such updates, one for each $v \in V$, all the lists $L(v)$ will have the form $[\Delta] \cup \{c_w\}$, and the union of the lists of colors of the neighbors of v will have most 2Δ colors. The actual identity of the lists depends on what random update operation was performed, but they are determined unambiguously, based on the first n choices of α_t's. The type of updates used above will be called *warm-up* updates.

Next, we see how ensuring that the union of the list of v's neighbors is at most 2Δ, allows us to ensure that the sizes of the lists rapidly fall to at most two. Indeed, assume that we have generated $\alpha[-T, -(T-n)+1]$ and the list $L_f := L'_{-(T-n)+1}$ has the above property. Now, let v be a vertex whose color we wish to update. The update operations we now describe will be called *bounded list updates*. They work as follows. We pick two colors: c_1 uniformly from the set $B = [k] \setminus \bigcup_{w \in N(v)} L_f(w)$, and c_2 uniformly from the set $\bigcup_{w \in N(v)} L_f(w)$ itself. Note that c_1 is always a valid new color for v, but depending on the current coloring χ, its importance varies. A probabilistic choice between c_1 and c_2 can be arranged (mediated by the parameter τ, as earlier) to ensure that the new color of v is uniformly chosen. And what can we say about the $L'(v)$ at the end of this operation? It is $\{c_1, c_2\}$, a list of size at most two! This can then be repeated for other vertices systematically, ensuring in the end that all lists have size at most two. This informal outline will be rigorously justified in the following section. We note that for it to be implemented, we require the number of colors k to be more than 4Δ; additional care in the design of the update operations will be needed to ensure that the some conclusion can be reached as long as $k > 3\Delta$. The combination of warm-up and bounded list update operations described so far constitute the *Warm-up* phase of our algorithm.

Our goal now is to extend the above sequence with some additional updates so that we can ensure that the the final list sizes with high probability all become one. This is achieved using the bounded-list update operations we used already. The difference is only in the analysis. An update operation on a vertex whose list size is already one will with some probability result in a list size of 2. However, vertices with list-size 2 can hope to see a reduction in their list size. If we track the number W_t of vertices whose list sizes have become one at time t, this quantity performs a random walk on the number line (between 0 and n with n as absorbing) with a non-negligible bias towards n. We observe that if k is large enough ($k > 3\Delta$), then with high probability this walk will hit n within $\text{poly}(n, k)$ steps, and helps us justify Lemma 1.2 (d).

Organisation of this paper

In the following sections, we elaborate on ideas outlined above, and justify Lemma 1.2. In Section 2.1-Section 2.3, we formally define T, the distribution $\mathcal{D}$, the primitives that we use to generate the αs at different stages of the algorithm, and the precise correspondence between the strings of type α and the corresponding update operations of type U_α. In Section 2.4 we provide a precise computation of T. Finally, in Section 2.5 we formally define the predicate Φ and collate all our results from the previous sections to establish parts $(a), (b), (c)$ and (d) of Lemma 1.2. Section 3 contains the running time analysis of our algorithm.

2 The distribution $\mathcal{D}$ and the predicate Φ

In this section, we will prove Lemma 1.2. Recall that we have a graph $G = (V, E)$ on n vertices and the number of colors $k > 3\Delta$. The update sequence $(\alpha_{-T}, \ldots, \alpha_{-T'-1})$ will correspond to the warm-up phase of our algorithm and the sequence $(\alpha_{-T'}, \ldots, \alpha_{-1})$ will correspond to phase where we shrink the list sizes. In particular, we set $T' = 2\frac{k-\Delta}{k-3\Delta} n \ln n$ and $T = T' + |E(G)| + n$ where $|E(G)|$ is the number of edges in G.

2.1 The update α and its relation to U_α

Recall that an update operation is represented by a tuple α of the form (v, τ, L, L', M). In the previous section, we informally indicated the role played by each of the components of this 5-tuple. In this section, we specify exactly how these components are generated and how they determine the update operation U_α. The update operation associated with $\alpha = (v, \tau, L, L', M)$ will act on colorings $\chi \sim L$; that is, whenever we use α in our sequence, it will be guaranteed that the previous update operations result in a coloring $\chi \sim L$. However, if each $L(v) = [k]$ for all v, then U_α acts on all colorings.

As stated in the introduction, we have two types of update operations, the warm-up update operation and the bounded-list update operation. The generation and decoding methods are different for the two. We describe, for each type, how the corresponding α is generated and how, given such an α, the corresponding U_α is applied to a coloring χ. Our final sequence of updates will be obtained by generating the updates one after another according to a strategy that we will prescribe.

Fix a coloring χ. The operation U_α will attempt to recolor the vertex v (leaving the colors of the other vertices untouched) by picking a color from the list M. The α we generated will have $L'(v) = M$; this will ensure that $U_\alpha(\chi) \sim L'$. In order to ensure that the coloring is proper, the color chosen for v must avoid the colors used by v's neighbors. In particular, if $|L(w)| = 1$ for a neighbor w of v, then the unique color in $L(w)$ will never be a candidate color for v. Thus, the following two sets will play a central role in our definition of U_α:

$$S_L(v) = \bigcup_{w \in N(v)} L(w) \qquad \text{and} \qquad Q_L(v) = \bigcup_{\substack{w \in N(v) \\ |L(w)|=1}} L(w).$$

The following are the two types of αs, which correspond to the warm-up and bounded list update operations.

Case $|M| = \Delta + 1$: Such an α will be used to generate the initial part of our sequence of updates, the warm-up phase, whose goal is to ensure that all sets in our list of colors have size at most 2. We say that such an α is of type *Warm-Up*.

Case $|M| \leq 2$: In this case we say the α is of the type *BoundedList* and such an α will be used to reduce the list sizes to ≤ 2 and with non-trivial probability make the list size 1.

Accordingly, we distinguish the following two subsections based on the types of αs :*Warm-Up* and *BoundedList*. In each subsection, we describe how each type of α is generated and how we decode it to U_α.

2.1.1 Warm-up updates

To specify the warm-up updates we will present two procedures: WARMUP.GEN and WARMUP.DECODE. The procedure WARMUP.GEN takes a tuple $\alpha_{\mathsf{in}} = (v_{\mathsf{in}}, \tau_{\mathsf{in}}, L_{\mathsf{in}}, L'_{\mathsf{in}}, M_{\mathsf{in}})$, a vertex v and a list A consisting of Δ colors, and returns another tuple. This procedure is randomized: its output α_{f} is a random tuple of the type *Warm-Up*. The update operation corresponding to such a tuple is obtained using procedure WARMUP.DECODE, which takes a tuple α_{f} (produced by WARMUP.GEN) and a coloring $\chi \sim L'_{\mathsf{in}}$, and produces another coloring, say χ'. Thus, the update operation $U_{\alpha_{\mathsf{f}}}$ is the map $\chi \mapsto$ WARMUP.DECODE$[\alpha_{\mathsf{f}}, \chi]$. The following lemma describes the relationship between the two procedures, and their important properties.

Lemma 2.1. *Let $\alpha_{\mathsf{in}} = (v_{\mathsf{in}}, \tau_{\mathsf{in}}, L_{\mathsf{in}}, L'_{\mathsf{in}}, M_{\mathsf{in}})$ be an arbitrary 5-tuple, $v \in B$ and A be a subset of Δ colors. Then,*

(a) If $\alpha_{\mathsf{f}} = (v_{\mathsf{f}}, \tau_{\mathsf{f}}, L_{\mathsf{f}}, L'_{\mathsf{f}}, M_{\mathsf{f}})$ is a tuple produced by WARMUP.GEN$[\alpha_{\mathsf{in}}, v, A]$*, then $L_{\mathsf{f}} = L'_{\mathsf{in}}$, $L'_{\mathsf{f}}(u) = L_{\mathsf{in}}(u)$ for all $u \neq v$, and $L'_{\mathsf{f}}(v)$ has the form $A \cup \{c\}$ for some color c outside A.*

(b) For all $\chi \sim L_{\mathsf{f}}$, we have $\chi' =$ WARMUP.DECODE$[\alpha_{\mathsf{f}}, \chi] \sim L'_{\mathsf{f}}$ *(with probability 1).*

(c) For all $\chi \sim L_{\mathsf{f}}$, the coloring χ' has the same distribution as GLAUBER DYNAMICS(χ, v) [2] *(recall that the output of* WARMUP.DECODE$[\alpha_{\mathsf{f}}, \chi]$ *is random).*

To prove this lemma, we need to specify WARMUP.GEN and WARMUP.DECODE. Before presenting the code and the proof of the lemma, we present the idea behind them. Given α_{in}, v and A, we somehow want to update the color of vertex v. The precise color to assign to v will need to depend on the current coloring χ, in particular, on $\chi(N(v))$. If all we wanted was to restrict the size $L'_{\mathsf{f}}(v)$, we could just insist that v's color be confined to a random subset of size $\Delta + 1$; that is, WARMUP.GEN would specify a random sequence of $\Delta + 1$ distinct colors and once χ is known, we would replace $\chi(v)$ by the first color in this list not currently used by any neighbor of v. However, as explained in the introduction, we wish to ensure that the lists of different vertices have an overlap of size Δ. So we actually generate a random permutation of the input set A and append to it at the end a random color c_1 chosen from $[k] \setminus A$. This simple procedure is our WARMUP.GEN. Now, once such an α_{f} has been specified, to update $\chi(v)$, we have a choice: either we pick c_1 or one of the colors from $[\Delta]$. If c_1 is an invalid option (it is being used by a neighbor of v), then we have no choice but to pick a color from $[\Delta]$ (there must be one available!). Now, c_1 will be a valid option with probability $(k - |\chi(N(v)) \cup A|)/(k - \Delta)$, whereas such a color should actually be used to replace $\chi(v)$ with probability $(k - |\chi(N(v)) \cup A|)/(k - |\chi(N(v))|)$. So we accept the c_1 with probability $(k - \Delta)/(k - |\chi(N(v))|)$ and with the remaining probability we use the first valid vertex from σ. To implement this acceptance sampling we pick a random number $\tau \in [0, 1]$ and accept c_1 if it is at least the threshold $1 - (k - \Delta)/(k - |\chi(N(v))|)$. This is all that WARMUP.DECODE does. We

[2] GLAUBER DYNAMICS(χ, v) requires that χ' = WARMUP.DECODE$[\alpha_{\mathsf{f}}, \chi]$ has the following distribution: $\chi'(w) = \chi(w)$, for all $w \neq v$ and $\chi'(v)$ is uniformly distributed in the set of colors $[k] \setminus \chi(N(v))$.

now present the code, and formally prove the Lemma 2.1

Algorithm 2: WARMUP: generation and decoding

1 **Function** `gen()`:
  Input : $\alpha_{\text{in}} = (v_{\text{in}}, \tau_{\text{in}}, L_{\text{in}}, L'_{\text{in}}, M_{\text{in}})$, $v \in V$ and $A \subseteq [k]$ with $|A| = \Delta$
  Output: $\alpha_{\text{f}} = (v_{\text{f}}, \tau_{\text{f}}, L_{\text{f}}, L'_{\text{f}}, M_{\text{f}})$
2 $\tau_{\text{f}} \xleftarrow{R} [0,1]$; $\sigma \xleftarrow{R} S_A$; $c_1 \xleftarrow{R} [k] \setminus A$;
3 $L_{\text{f}} \leftarrow L'_{\text{in}}$; $L'_{\text{f}} \leftarrow L'_{\text{in}}$;
4 $L'_{\text{f}}(v) \leftarrow A \cup \{c_1\}$;
5 $M_{\text{f}} \leftarrow (\sigma, c_1)$; $v_{\text{f}} \leftarrow v$;
6 **return** $\alpha_{\text{f}} = (v_{\text{f}}, \tau_{\text{f}}, L_{\text{f}}, L'_{\text{f}}, M_{\text{f}})$
7 **Function** `decode()`:
  Input : $\alpha = (v, \tau, L, L', M)$ and a coloring $\chi \sim L$
  Output: $\chi' \sim L'$
8 $\chi' \leftarrow \chi$;
9 $p_\chi(v) \leftarrow 1 - \frac{k-\Delta}{k-|\chi(N(v))|}$;
10 **if** $c_1 \notin \chi(N(v))$ *and* $\tau \geq p_\chi(v)$ **then**
11 $\chi'(v) \leftarrow M[\Delta + 1]$;
12 **else**
13 $\chi'(v) \leftarrow$ first color in the list $M[1, \Delta]$ that is not in $\chi(N(v))$; `// If` $c_1 \in \chi(N(v))$ `then such a color is always available as` $|\chi N(v)| \leq \Delta$`!`
14 **end if**
15 **return** χ'

Proof of Lemma 2.1. Part (a) is clear from the procedure of WARMUP.GEN. We update L_{f} as L'_{in} and L'_{f} differs from L'_{in} only at the vertex v where $L'_{\text{f}}(v) = A \cup \{c_1\}$.

For part (b) consider any $\chi \sim L'_{\text{in}} = L_{\text{f}}$. Note that during WARMUP.GEN we set $L'_{\text{f}}(v) = A \cup \{c_1\}$ and M_{f} as $(\sigma(A), c_1)$ and during WARMUP.DECODE we update the color of $\chi'(v)$ from within M_{f} and for all $w \neq v$ we copy the color of χ. This proves part (b).

For part (c) we remind ourselves that the process GLAUBER DYNAMICS(χ, v) requires $\chi'(v)$ to be uniformly distributed on the set $[k] \setminus \chi(N(v))$. Notice that $c_1 = M[\Delta + 1]$ is a uniformly random choice of color over $[k] \setminus A$ and hence whenever c_1 is chosen for $\chi'(v)$ by WARMUP.DECODE we know its distribution will be uniform over $k \setminus (A \cup \chi(N(v)))$. Also, whenever we choose a color from $M[1, \Delta] = \sigma$ in WARMUP.DECODE, where σ is a uniformly random permutation of A, to update at $\chi'(v)$ we know that its distribution is uniform over $A \setminus \chi(N(v))$. Hence, to prove part (c), it suffices to show that c_1 is chosen with probability $\frac{k-|A \cup \chi(N(v))|}{k-\chi(N(v))}$. From Line 10 in WARMUP.DECODE we have:

$$\Pr[\chi'(v) = c_1] = \Pr[c_1 \notin \chi(N(v))] \times \Pr[\tau \geq p_\chi(v)]$$
$$= \frac{(k - |\chi(N(v)) \cup A|)}{(k-\Delta)} \times \frac{k-\Delta}{k - |\chi(N(v))|} = \frac{(k - |\chi(N(v)) \cup A|)}{(k - |\chi(N(v))|)}.$$

□

2.1.2 Bounded list Updates

In this section our main goal will be to describe αs of the type *BoundedList* which reduce the list size at the vertex at which they act, to ≤ 2 and with non-trivial probability make the list size 1. These type of updates will be applied after the *Warm-up* phase whence we will maintain that for all vertices the list sizes are at most 2. As in section Section 2.1.1 we will present two procedures: BOUNDEDLIST.GEN and BOUNDEDLIST.DECODE. The procedure BOUNDEDLIST.GEN takes a tuple $\alpha_{\mathsf{in}} = (v_{\mathsf{in}}, \tau_{\mathsf{in}}, L_{\mathsf{in}}, L'_{\mathsf{in}}, M_{\mathsf{in}})$ with the promise that $|S_{L'_{\mathsf{in}}}| < k - \Delta$ and a vertex v, and returns a random tuple α_{f} of the type *BoundedList*. The update operation corresponding to such a tuple is obtained using procedure BOUNDEDLIST.DECODE, which takes a tuple α_{f} (produced by BOUNDEDLIST.GEN) and a coloring $\chi \sim L'_{\mathsf{in}}$, and produces another coloring, say χ'. Thus, the update operation $U_{\alpha_{\mathsf{f}}}$ is the map $\chi \mapsto$ BOUNDEDLIST.DECODE$[\alpha_{\mathsf{f}}, \chi]$. The following lemma describes the relationship between the two procedures, and their important properties.

Lemma 2.2. *Let $\alpha_{\mathsf{in}} = (v_{\mathsf{in}}, \tau_{\mathsf{in}}, L_{\mathsf{in}}, L'_{\mathsf{in}}, M_{\mathsf{in}})$ be an arbitrary 5-tuple and $v \in V$. Suppose $|S_{L_{\mathsf{in}}}(v)| < k - \Delta$ and that $\alpha_{\mathsf{f}} = (v_{\mathsf{f}}, \tau_{\mathsf{f}}, L_{\mathsf{f}}, L'_{\mathsf{f}}, M_{\mathsf{f}})$ is a tuple produced by* BOUNDEDLIST.GEN$[\alpha_{\mathsf{in}}, v]$. *Then :*

(a) Let $L = L'_{\mathsf{in}}$ $L_{\mathsf{f}} = L$, $L'_{\mathsf{f}}(u) = L(u)$ for all $u \neq v$, and with probability $p_L = 1 - \frac{|S_L(v)| - |Q_L(v)|}{k-\Delta}$ we have $|L'_{\mathsf{f}}(v)| = 1$. With the remaining probability we have $|L'_{\mathsf{f}}(v)| = 2$.

(b) For all $\chi \sim L_{\mathsf{f}}$, we have $\chi' =$ BOUNDEDLIST.DECODE$(\alpha_{\mathsf{f}}, \chi) \sim L'_{\mathsf{f}}$ *(with probability 1).*

(c) For all $\chi \sim L_{\mathsf{f}}$, the coloring χ' has the same distribution as GLAUBER DYNAMICS(χ, v).

We now describe the ideas behind BOUNDEDLIST.GEN and BOUNDEDLIST.DECODE. Consider $\alpha_{\mathsf{in}} = (v_{\mathsf{in}}, \tau_{\mathsf{in}}, L_{\mathsf{in}}, L'_{\mathsf{in}}, M_{\mathsf{in}})$ and let $L = L'_{\mathsf{in}}$. Also consider a vertex $v \in V$ and a coloring $\chi \sim L$. We wish to produce $\alpha_{\mathsf{f}} = (v_{\mathsf{f}}, \tau_{\mathsf{f}}, L_{\mathsf{f}}, L'_{\mathsf{f}}, M_{\mathsf{f}})$ with L'_{f} of size at most 2 (with some probability of being of size 1) and a coloring χ' such that $\chi' \sim L'_{\mathsf{f}}$, and χ' should be distributed according to GLAUBER DYNAMICS(χ, v). For now let us focus on producing L'_{f} of size 2. Hence, without knowing $R(v) \coloneqq \chi(N(v))$ we need to produce two colors $L'_{\mathsf{f}} = \{c_1, c_2\}$ such that by choosing one of the colors we may ensure that $\chi'(v)$ is distributed uniformly over $[k] \setminus R(v)$. Notice that since $\chi \sim L$ we have $Q_L(v) \subseteq R(v) \subseteq S_L(v)$. An initial attempt is to sample a color $c_1 \notin S_L(v)$ uar and set $\chi'(v) = c_1$. While this is a valid choice of color at v it is not distributed uniformly over $[k] \setminus R(v)$; in particular, no mass is put on colors in $S_L(v) \setminus R(v)$. To remedy this situation we sample another color c_2 from $S_L(v)$ uar and make a probabilistic choice between c_1 and c_2 to be used as $\chi'(v)$. In particular, we prescribe the update at v as follows. Let τ be chosen from $[0,1]$ uar and let p_χ be a threshold in $[0,1]$: if $\tau \leq p_\chi$ or $c_2 \in R(v)$ then $\chi'(v) = c_1$; else $\chi'(v) = c_2$. Now, it is a matter of calculation to arrange for an appropriate value of p_χ such that $\chi'(v)$ is uniform over $[k] \setminus R(v)$. A direct calculation (see proof of Lemma 2.2) shows that $p_\chi = 1 - \frac{|S_L(v)| - |Q_L(v)|}{k - |R(v)|}$. For the remaining part of having a non-trivial probability of L'_{f} being of size 1, we provide a threshold p_L such that

always $p_L \le p_\chi$. Thus, whenever $\tau \le p_L$ we set $L'_{\mathsf{f}} = \{c_1\}$. We set p_L as $1 - \frac{|S_L(v)|-|Q_L(v)|}{k-|\Delta|}$. To ensure $p_\chi \ge p_L > 0$ we use that assumption that $|S_L(v)| < k - \Delta$.

Algorithm 3: BOUNDEDLIST: generation and decoding

1 **Function** `gen()`:
  Input : $\alpha_{\mathsf{in}} = (v_{\mathsf{in}}, \tau_{\mathsf{in}}, L_{\mathsf{in}}, L'_{\mathsf{in}}, M_{\mathsf{in}})$, $v \in V$ with $|S_L(v)| < k - \Delta$
  Output: $\alpha_{\mathsf{f}} = (v_{\mathsf{f}}, \tau_{\mathsf{f}}, L_{\mathsf{f}}, L'_{\mathsf{f}}, M_{\mathsf{f}})$
2 $L \leftarrow L'_{\mathsf{in}}$ $L_{\mathsf{f}} \leftarrow L$; $L'_{\mathsf{f}} \leftarrow L$; $\tau_{\mathsf{f}} \xleftarrow{R} [0,1]$; $v_{\mathsf{f}} \leftarrow v$;
3 $S \leftarrow S_L(v)$; $Q \leftarrow Q_L(v)$;
4 $c_1 \xleftarrow{R} [k] \setminus S$; $c_2 \xleftarrow{R} S \setminus Q$;
5 $p_L \leftarrow 1 - \frac{|S|-|Q|}{k-\Delta}$; // $|S| < k - \Delta$ ensures $p_L \in [0,1]$
6 **if** $\tau \le p_L$ **then**
7 $L'_{\mathsf{f}}(v) \leftarrow \{c_1\}$; $M_{\mathsf{f}} \leftarrow (c_1)$
8 **else**
9 $L'_{\mathsf{f}}(v) \leftarrow \{c_1, c_2\}$; $M_{\mathsf{f}} \leftarrow (c_1, c_2)$
10 **end if**
11 **return** $\alpha_{\mathsf{f}} = (v_{out}, \tau_{\mathsf{f}}, L_{\mathsf{f}}, L'_{\mathsf{f}}, M_{\mathsf{f}})$
12 **Function** `decode()`:
  Input : $\alpha = (v, \tau, L, L', M)$, $\chi \sim L'$ with $|S_L(w)| < k - \Delta$, $M[1] \notin S_L(v)$ and $M[2] \in S_L(v)$ or $M(2) = \varnothing$
  Output: $\chi' \sim L'$
13 $\chi' \leftarrow \chi$;
14 $R \leftarrow \chi(N(v))$; $S \leftarrow S_L(v)$; $Q \leftarrow Q_L(v)$;
15 $c_1 \leftarrow M(1)$; $c_2 \leftarrow M(2)$; $p_\chi \leftarrow 1 - \frac{|S|-|Q|}{k-|R|}$; // $|S| < k - \Delta$ ensures $p_\chi \in [0,1]$
16 **if** $\tau \le p_\chi$ *or* $c_2 \in R$ **then**
17 $\chi'_{\mathsf{f}}(v) \leftarrow c_1$;
18 **else**
19 $\chi'_{\mathsf{f}}(v) \leftarrow c_2$;
20 **end if**
21 **return** χ'

Proof. We let $L = L'_{\mathsf{in}}$ and $\chi(N(v)) = R(v)$. We first note that $Q_L(v) \subseteq R(v) \subseteq S_L(v)$ as $\chi \sim L$. Also, p_L at Line 5 in Algorithm 3 is at most p_χ at Line 15 in Algorithm 3.

To prove part (a), first observe that line 2 directly implies $L_{\mathsf{f}} = L$. It is also evident that throughout the execution of the algorithm, $L'_{\mathsf{f}}(u)$ is never updated, for all $u \ne v$, after the execution of line 2. This proves that $L'_{\mathsf{f}}(u) = L(u)$ for all $u \ne v$. Finally, we note that :

- τ_{f} is distributed uniformly at random in $[0,1]$.
- $L'_{\mathsf{f}}(v)$ is singleton iff $\tau_{\mathsf{f}} \le p_L$.

Thus the probability of $L'_f(v)$ being singleton is exactly p_L. Referring to the value of p_L in line 5 of Algorithm 3 proves the claim.

To prove part (b), note from BOUNDEDLIST.GEN that the set M_f (disregarding the ordering) is actually the same as the set $L'_f(v)$ (Line 6-10). Also, we see from BOUNDEDLIST.DECODE that $\chi'_f(v)$ is always contained within M_f. For all $w \neq v$, BOUNDEDLIST.DECODE sets $\chi'_f(w)$ to $\chi(w)$. By the hypothesis of the Lemma, $\forall w \in V$, $\chi(w) \in L_f(w)$. Finally, since BOUNDEDLIST.GEN sets $L'_f(w)$ to $L(w)$ for all $w \neq v$, we conclude that the first claim is true.

To prove part (c), we first note that the random process GLAUBER DYNAMICS(χ, v) recolors the vertex v with a color chosen uar from the set $[k] \setminus \chi(N(v))$, while retaining the color of every other vertex. Observe that:

a. The distribution of $\chi'(v)$ (induced by Algorithm 3), is a convex combination of two uniform distributions : one supported on the set $[k] \setminus S_L(v)$ (when $\chi'(v) = c_1$) and the other supported on the set $S_L(v) \setminus R(v)$ (when $\chi' = c_2$).

b. $\chi'(v) = c_1$ iff either $\tau \leq p_\chi$ or $c_2 \in \chi(N(v))$. Hence,

$$\Pr[\chi'(v) = c_1] = p_\chi + (1 - p_\chi) \cdot \Big(\frac{|R(v)| - |Q_L(v)|}{|S(v)| - |Q_{L(v)}|} \Big).$$

Observation [a] implies that if $\Pr[\chi'(v) = c_1]$ turns out to be of the form $\frac{k-|S_L(v)|}{k-|R(v)|}$, it would imply that the distribution of $\chi'(v)$ is indeed uniform on the set $[k] \setminus R(v)$. Referring to BOUNDEDLIST.DECODE and solving for $\Pr[\chi'(v) = c_1]$ by substituting p_χ in observation [b], we verify that this is indeed true. This proves part 3. □

2.2 Warm-up Phase

The warm-up phase will run for $T - T'$ steps from time $t = -T$ to $t = -T'$. The goal of this phase is to bring the list size at every vertex to at most 2. During the warm-up phase we will generate a sequence $(\alpha_{-T}, \ldots, \alpha_{-T'-1})$. The main idea of the warm-up phase is, given any vertex v, to bring the union of the colors in the lists of the neighbors of v down to 2Δ. This will allow us to use the primitive BOUNDEDLIST.GEN to reduce the list size of v down to 2.

Fix an ordering of the vertices say $\{v_1, \ldots, v_n\}$ and let A be an arbitrary set of Δ colors. A first attempt is to repeatedly use WARMUP.GEN with v_i and A as input and cycle through the vertices in order. Let the output of this operation be $\alpha' = (v_n, ., ., L, .)$. At the end of this since A is present in $L(v)$ for all $v \in V$ we know that $S_L(v) \leq 2\Delta$ for all $v \in V$. Hence, we may now use BOUNDEDLIST.GEN$[\alpha', v_1]$. This will result in the list size at v_1 dropping to at most 2. May we now use BOUNDEDLIST.GEN$[\alpha', v_1]$? No! In particular if $v_2 \in N(v)$ the union of colors in the lists of neighbors of v_2 may now be as large as $2\Delta + 1$ due to the freshly acquired color/s in the list of v_1. To remedy this situation we repeatedly run WARMUP.GEN with v_i and A' (defined shortly) as input and cycle through the vertices $\{v_2, \ldots, v_n\}$ in order. Here A' is a set of size Δ which include the color/s in the list of v_1. This does not disturb the list size of v_1 and now what may we conclude about the union of colors in the lists of neighbors of v_1? It is at most 2Δ as the colors in the list of v_1 are already accounted for in A'! We may continue in this manner until all the vertices have list size at most 2.

Now, we present the above strategy formally and a bit more parsimoniously (if we want to use BOUNDEDLIST.GEN with vertex v_i we need to worry only about the lists of its neighbors). We first

present a sub-routine Algorithm 4 which is input a state where the lists of all vertices up till vertex v_{i-1} are of size 2. It outputs a state where the lists of vertices up till v_{i-1} are unchanged and further the list at vertex v_i is of size at most 2. We then repeatedly invoke Algorithm 4 in Algorithm 5 to ensure that the lists at all vertices are of size at most 2.

For an ordering on the vertices $V = \{v_1, \ldots, v_n\}$ let $N_>(v_i) \coloneqq \{v_j \in N(v) \mid j > i\}$ and $N_<(v_i) \coloneqq \{v_j \in N(v) \mid j < i\}$

Algorithm 4: WARMUPHELPER

Input : $\alpha_{\mathsf{in}} = (v_{\mathsf{in}}, \tau_{\mathsf{in}}, L_{\mathsf{in}}, L'_{\mathsf{in}}, M_{\mathsf{in}})$, $i \in [n]$

1 **Promise:** for all $j < i$: $|L'_{\mathsf{in}}(v_j)| \leq 2$

Output: $\alpha[-1, -(1 + |N_>(v_i)|)]$

2 $t \leftarrow -(1 + |N_>(v_i)|)$; $L \leftarrow L'_{\mathsf{in}}$;

3 Let A be any subset of $[k]$ of size Δ which maximally intersects the set $\bigcup_{w \in N_<(v_i)} L(w)$;

4 **for** $w \in N_>(v_i)$ **do**

5 $\quad \alpha_t \leftarrow$ WARMUP.GEN$[\alpha_{t-1}, w, A]$;

6 $\quad t \leftarrow t + 1$;

7 **end for**

8 $\alpha_t \leftarrow$ BOUNDEDLIST.GEN$[\alpha_{t-1}, v_i]$; `// here` $t = -1$ `and` $|S_{\tilde{L}}(v_i)| < k - \Delta$ `where` $\tilde{L} = L_{(-1)}$ `(to be able to use` BOUNDEDLIST.GEN`) is shown in Lemma 2.3`

9 **return** $\alpha[-1, -(1 + |N_>(v_i)|)]$

Lemma 2.3. *Consider L'_{in} in α_{in} and $i \in [n]$ such that for all $j < i$: $|L'_{\mathsf{in}}(v_j)| \leq 2$. Let $\alpha[-1, -(1 + |N_>(v_i)|)]$ be the output of* WARMUPHELPER$[\alpha_{\mathsf{in}}, i]$ *and let $\alpha_{-1} = (v, \tau, L', L'', M)$. Then, for all $j \leq i$: $|L''(j)| \leq 2$.*

Proof. Firstly, note that Algorithm 4 does not update any vertex with an index $< i$. This is ensured in the For loop (line 4-7), since only those neighbors w of v_i are updated which are after it in the ordering, by the definition of $N_>(v_i)$. Hence, we observe that for all $j < i$, $L''(j) = L'_{\mathsf{in}}(j)$. Thus the lemma is true for all $j < i$. For the vertex v_i, we observe that :

- $\bigcup_{w \in N_<(v_i)} L'(w)$ may be of size at most $2|N_<(v_i)|$.
- $\bigcup_{w \in N_>(v_i)} L'(w)$ may be of size at most $\Delta + |N_>(v_i)|$ since for all $w \in N_>(v_i) : A \subseteq L'(w)$ by Lemma 2.1[a] where A is the set of size Δ at Line 3.

Let a be the size of $\bigcup_{w \in N_<(v_i)} L'(w)$, let b be the size of $\bigcup_{w \in N_>(v_i)} L'(w)$ and c be the size of the intersection of $\bigcup_{w \in N_<(v_i)} L'(w)$ and $\bigcup_{w \in N_>(v_i)} L'(w)$. Note that the choice of A ensures that the intersection of A and $\bigcup_{w \in N_<(v_i)} L'(w)$ is of size at least $\min\{\Delta, a\}$. Hence, $a \leq 2|N_<(v_i)|$ and $b \leq \Delta + |N_>(v_i)|$ and $c \geq \min\{\Delta, a\}$. Taking all these arguments together, we see that :

$$|\bigcup_{w \in N(v_i)} L'(w)| = a + b - c \leq a + b - \min(\Delta, a) \leq 2\Delta.$$

Since $k > 3\Delta$, we have thus ensured the condition that $k - |S_{L'}(v_i)| > \Delta$, which implies that we can legitimately use Algorithm 3 on v_i. Invoking Lemma 2.2[a], we conclude the proof. □

Before we begin the description of the WARMUP algorithm, we setup the symbol JUNK as a filler for those variables whose contents are not accessed during execution.

Algorithm 5: WARMUP

Output: $\alpha[-1, -(T - T')]$

$t \leftarrow -(T - T')$;

$v, \tau, L, M \leftarrow$ JUNK ;

$L' \leftarrow [k]^V$;

$\alpha_t = (v, \tau, L, L', M)$;

for $i = 1, 2, \ldots, n$ **do**

  $\alpha[t + |N_>(v_i)| + 1, t + 1] \leftarrow$ WARMUPHELPER$[\alpha_t, i]$;

  $t \leftarrow t + |N_>(v_i)|$; // T, T'(Section 2.4) are such that loop exits with $t = -1$

end for

return $\alpha[-1, -(T - T')]$

Lemma 2.4. *Let $\alpha[-1, -(T - T')]$ be the output of* WARMUP*[] and let $\alpha_{-1} = (v, \tau, L', L'', M)$. Then, for all $w \in V$ we have $|L''(w)| \leq 2$.*

Proof. Let us refer to the last time when $L''(v_i)$ was updated as t_i. Appealing to Lemma 2.3, one can see that, after being updated for the last time, $|L_{t_i+1}(v_i)| \leq 2$. Collating all the arguments together, it is clear that at the end of execution :

- For all $v_i \in V$, $L''(v_i) = L_{t_i+1}(v_i)$
- Since for all $i \in [n]$, $|L_{t_i+1}(v_i)| \leq 2$, the same holds true for $L''(v_i)$ as well.

This concludes the proof. □

2.3 Coalescence Phase

In this section we describe the sequence $(\alpha_{-1}, \ldots, \alpha_{-T'})$ which is geared towards obtaining $|L_0(v)| = 1$ for all $v \in V$. This phase starts with a state where all the list sizes are at most 2, which is the condition after the warm-up phase. We then repeatedly apply BOUNDEDLIST.GEN$[\alpha_{t-1}, v]$ to obtain α_t where v is chosen uar from V, and exploit the fact during the update by BOUNDEDLIST.GEN there is a significant probability of $|L_{t+1}(v)| = 1$. In case, $|L_t(v)| = 1$ this is progress. However, it may also be the case that $|L_t(v)| = 1$ and $|L_{t+1}(v)| = 2$. Hence, if we define $W_t \coloneqq \{v \mid |L_t(v)| = 1\}$ (earlier we had defined W_t to be the number of vertices of list size 1) then $|W_t|$ perform a random walk on $[0, n]$. We would like $|W_0| = n$. We show in Lemma 2.5 that there is a drift to the right and

that this walk absorbs with probability at least 1/2.

Algorithm 6: COALESCENCE

Input : $\alpha_{in} = (v_{\mathsf{in}}, \tau_{\mathsf{in}}, L_{\mathsf{in}}, L'_{\mathsf{in}}, M_{\mathsf{in}})$

Promise: for all $v \in V$: $|L'_{\mathsf{in}}(v)| \le 2$

Output: $\alpha[-1, -T']$

$\alpha_{-T'-1} \leftarrow \alpha_{\mathsf{in}}$;

for $t = -T', \ldots, -1$ **do**

  $v \overset{R}{\leftarrow} V$;

  $\alpha_t \leftarrow$ BOUNDEDLIST.GEN$[\alpha_{t-1}, v]$; // the condition $|S_L(v)| < k - \Delta$ is

  // shown in Lemma 2.5

end for

return $\alpha[-1, -T']$

Lemma 2.5. *Consider $L'_{\mathsf{in}} \in \alpha_{\mathsf{in}}$ such that for all $v \in V$: $|L'_{\mathsf{in}}(v)| \le 2$. Let $\alpha[-1, -T']$ be the output of* COALESCENCE*$[\alpha_{\mathsf{in}}]$. Also, for all $t \in \{-T', -1\}$ let $\alpha_t = (., ., L_t, L_{t+1}, .)$. Then,*

(a) For all $v \in V$ and t we have $|L_t(v)| \le 2$ and hence $|S_{L_t}(v)| < k - \Delta$. In particular, the promise for BOUNDEDLIST.GEN*[.,.] is always maintained.*

(b) With probability at least 1/2 we have for all $v \in V$: $|L_0(v)| = 1$.

Proof. To prove part (a), suppose that up till time t we have for all $v \in V$ $|L_t(v)| \le 2$. The update α_t is obtained using the primitive BOUNDEDLIST.GEN$[\alpha_{t-1}, v]$. By Lemma 2.2[a] we have that $|L_{t+1}(v)| \le 2$ for all $v \in V$ Since the initial L'_{in} has this property, the proof is completed by induction.

To prove part (b) we require the following definitions. We recall $W_t \coloneqq \{v \mid |L_t(v)| = 1\}$ and set $D_t \coloneqq V \setminus W_t$. In one time step $|W_t|$ can change by at most 1 :

- $|W_{t+1}|$ increases by one over $|W_t|$ when $v \in D_t$ is chosen as the vertex to update, and $L_{t+1}(v)$ becomes a singleton. This happens with probability at least $p_L(v) = 1 - \frac{|S_L(v)| - |Q_L(v)|}{k - \Delta}$ where $L = L_t$ by Lemma 2.2[a].
- $|W_{t+1}|$ decreases by one over $|W_t|$ when $v \in W_t$ is chosen as the vertex to update, and $L_{t+1}(v)$ is updated to have size 2. This happens with probability at most $p_L(v) = 1 - \frac{|S_L(v)| - |Q_L(v)|}{k - \Delta}$ where $L = L_t$.

We also define: $\mathrm{bad}_t(v) \coloneqq |\{u \mid u \in N(v), u \in D_t\}|$ and $\delta_t \coloneqq |W_{t+1}| - |W_t|$.

$$\begin{aligned}
\Pr[\delta_t = 1] &= \frac{1}{n} \sum_{v \in D_t} p_L(v) \\
&\overset{(a)}{\ge} \frac{1}{n} \sum_{v \in D_t} \left(1 - \frac{2\,\mathrm{bad}_t(v)}{k - \Delta}\right) \\
&= \frac{|D_t|}{n} - \frac{1}{n} \sum_{v \in D_t} \frac{2\,\mathrm{bad}_t(v)}{k - \Delta}.
\end{aligned}$$

Again,

$$\Pr[\delta_t = -1] = \frac{1}{n}\sum_{v\in W_t}\big(1 - p_L(v)\big) \overset{(b)}{\leq} \frac{1}{n}\sum_{v\in W_t}\Big(\frac{2\,\mathrm{bad}_t(v)}{k-\Delta}\Big)$$

Inequality (a) and (b) are valid since :

- For all $v \in V(G)$, $p_L(v) = 1 - \frac{|S_L(v)|-|Q_L(v)|}{k-\Delta}$
- $|S_L(v)| \leq 2\mathrm{bad}_t(v) + |Q_L(v)|$

Therefore,

$$\begin{aligned}\mathbb{E}[\delta_t] = \Pr[\delta_t = 1] - \Pr[\delta_t = -1] &\geq \frac{|D_t|}{n} - \frac{1}{n}\sum_{v\in V}\frac{2\,\mathrm{bad}_t(v)}{k-\Delta} \\ &\geq \frac{|D_t|}{n}\left(1 - \frac{2\,\Delta}{k-\Delta}\right) \quad \left(\because \sum_{v\in V}\mathrm{bad}_t(v) \leq \Delta|D_t|\right).\end{aligned}$$

Notice that if for any time $t \in \{-T', -1\}$ if we have that $|W_t| = n$ then $|W_0| = n$ since for all $v \in V$ $p_L(v) = 1$ where $L = L_t$. Hence the evolution of $|W_t|$ corresponds to that of a random walk on $[0, n]$ with unit steps and positive drift: at i the drift is $\frac{(n-i)}{n}(1 - \frac{2\Delta}{k-\Delta})$. Here, 0 is the reflecting state and n is the absorbing state. The expected time to be absorbed is $\frac{k-\Delta}{k-3\Delta}n\ln n$, which follows from Lemma A.1.

Recall that $|W_0| = n$ implies that $|L_0(v)| = 1$ for all $v \in V$. As we have set T' to be $\big(2\frac{k-\Delta}{k-3\Delta}n\ln n\big)$. Then, by Markov's inequality, the proof is complete.

□

2.4 Setting the values of T and T'

Recall from the proof of Lemma 2.5, that $T' = 2\frac{k-\Delta}{k-3\Delta}n\ln n$. To set the value of T, notice that the WarmUp phase lasts for $|E(G)| + n$ steps, where $E(G)$ is the edge set of the graph G. This is easy to see, as for any ordering of the vertices the update rule WarmUpHelper ensures that if the vertex v is chosen for update, only v and those of its neighbors which succeed it in the ordering are updated. Hence, the edges joining v to these neighbors are counted only this one time with the update. Thus, we set $T = T' + |E(G)| + n$. As the maximum degree of G is Δ we have $T \leq T' + \frac{n\Delta}{2} + n$.

2.5 Justification of Lemma 1.2 and Φ

Now we describe the algorithm which generates $U[-1, -T]$ by generating $\alpha[-1, -T]$

Algorithm 7: Generating $\alpha[-1, -T]$

Output: $\alpha[-1, -T]$

1 $\alpha[-T, -T' - 1] \leftarrow$ WarmUp[];

2 $\alpha[-T', -1] \leftarrow$ Coalescence[$\alpha_{-T'-1}$];

The predicate Φ outputs true iff $|L_0(v)| = 1$. Let us now justify the various parts of Lemma 1.2 in order.

(a) That a sample from $\mathcal{D}$ can be computed efficiently is clear from the fact all sub-routines in Algorithm 7 are efficient. That each update instruction U_i can also be computed efficiently is clear from Algorithm 2 and Algorithm 3. Also, Φ is clearly efficiently computable justifying part (a).

(b) Notice that the vertex v_t (either random or fixed) we choose to update in Algorithm 7 is independent of the evolution up till time $t-1$. Further, at time t whichever sub-routine is used, faithfully follows GLAUBER DYNAMICS at the vertex v_t. Hence, $U(-1,-T)$ takes uniform distributions to uniform distribution justifying part (b).

(c) As mentioned above the predicate Φ outputs true iff $|L_0(v)| = 1$. We start with $L_T = [k]^V$ and maintain that if at time t there is a coloring $\chi \sim L_t$ then $U_t(\chi) \sim L_{t+1}$. This justifies part (c).

(d) Part (d) follows immediately from Lemma 2.5[b].

3 Running Time Analysis of Algorithm 1

In this section we justify Theorem 1.1 by analyzing the expected running time of Algorithm 1 detailing the various steps performed during its execution.

To analyze the expected running time of Algorithm 1 we need to analyze the expected time needed for generating $U(-1,-T)$. For this let us first investigate the expected running time of the two sub-routines WARMUPSAMPLER and BOUNDEDLISTSAMPLER. We assume that read/write and compare operations in these primitives have $O(\log n + \Delta \log k)$ cost (this can be achieved by storing the address of the vertex where the update took place and the updated colors in its list)

To that end, notice that in BOUNDEDLISTSAMPLER, the only operations with non-constant running time consist of :

- Pick τ uniformly from $[0,1]$ to compare with p_L which is a fraction whose denominator may be represented with at most $O(\log k)$ bits.
- Pick c_1 uniformly from $[k] \setminus S_L(v)$
- Pick c_2 uniformly from $S_L(v) \setminus Q_L(v)$.

With access to fair coins, each of the above operations require expected time $O(\log k + \log n)$. Hence the expected running time of BOUNDEDLISTSAMPLER is $O(\Delta \log k + \log n)$.

Next, we move on to WARMUPSAMPLER and note that the operations with non-trivial running time are :

- Pick τ uniformly from $[0,1]$ to compare with p_χ which is a fraction whose denominator may be represented with at most $O(\log k)$ bits.
- Pick σ uar from $\mathcal{S}_\Delta$.
- Pick c_1 uar from $[k] \setminus A$.

Again, the first and the third operations require expected time $O(\log k + \log n)$. The second operation requires expected time $O(\Delta \log \Delta)$. Hence the expected running time of WARMUPSAMPLER is $O(\Delta \log k + \log n)$

Note that, WARMUPSAMPLERis called exactly $\frac{n\Delta}{2}$ times during the WARMUP phase, BOUNDEDLISTSAMPLER is called n times during WARMUP and $2\frac{k-\Delta}{k-3\Delta} n \log n$ times during COALESCENCE. We also mention that it takes $O(n)$ time for initializing lists to $[k]$ every vertex at time $-T$.

Collating all the above arguments together, we see that the running time of Algorithm 7 is $O(n \log^2(n) \cdot \Delta^2 \log k)$.

Finally, we calculate the expected running time of Algorithm 1 Let i be the first index in Algorithm 1 such that we find $\Phi(U(-iT-1, -(i+1)T) = \text{true}$. So, we know that $|L_{-it}(v)| = 1$ for all $v \in V$. Now, we need to update this unique coloring $\chi \in \prod_{v \in V} L_{-it}(v)$ with $U(-iT, -1)$. By Lemma 1.2[d] on expectation $i = 2$. Hence on expectation we need to generate $U(-2T, -T+1)$ and $U(-T, 0)$ and simulate from $U(-2T, 0)$ (simulating a single update again takes $O(\Delta(\log n + \log k))$ time). Hence, the overall running time is $O(n \log^2(n) \cdot \Delta^2 \log k)$.

4 Bottleneck for achieving $k > 2\Delta$

In this current framework we have two primitives namely, WARMUP and BOUNDEDLISTSAMPLER, which are the workhorses of our algorithm for $k > 3\Delta$. Now, suppose we shoot for a better bound and work with $k \leq 3\Delta$ (even $k = 2\Delta + 1$). In this case we face two hurdles.

Firstly, after the warm-up phase of the algorithm say we end up with the bounding list L. Then, we are able to guarantee that $|S_L(v)| \leq 2\Delta$ for all $v \in V$. However, if $k \leq 3\Delta$ this is not enough to meet the input requirement for BOUNDEDLISTSAMPLER, i.e., $|S_L(v)| < k - \Delta$.

Secondly, even if we somehow manage to apply BOUNDEDLISTSAMPLER we may still produce lists of size 2. Recall, that the drift analysis of $|W_t|$ (where W_t is the number of vertices with list size 1) requires an extra margin of Δ, over the product of the maximum degree (Δ) and the bound on the list sizes we can guarantee, in k. If $k \leq 3\Delta$ then we do not have this margin.

We note that for the specific case of $k = 3\Delta$ the current analysis obtains that $|W_t|$ is a random walk on $[0, n]$ with non-negative drift. It may then be possible to analyze the variance of this walk and conclude that it absorbs to n in time $O(n^2)$.

Acknowledgements

We are grateful to Piyush Srivastava for pointing us to this problem and for the numerous detailed discussions which led to this work. We also thank him for proof-reading the write-up and pointing us to the relevant references. Finally, we are indebted to Prahladh Harsha and Jaikumar Radhakrishnan without whom the write-up would not have appeared in its current form. They greatly helped us in organizing our ideas and putting them in a presentable format.

References

[Bar16] Alexander I. Barvinok. *Combinatorics and Complexity of Partition Functions*, volume 30 of *Algorithms and combinatorics*. Springer, 2016. 3

[BD97] R. Bubley and M. Dyer. Path coupling: A technique for proving rapid mixing in Markov chains. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 223–231, October 1997. 2

[CDM+19] S. Chen, M. Delcourt, A. Moitra, G. Perarnau, and L. Postle. Improved Bounds for Randomly Sampling Colorings via Linear Programming. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2216–2234. Society for Industrial and Applied Mathematics, January 2019. 2

[DFHV04] M. Dyer, A. Frieze, T. P. Hayes, and E. Vigoda. Randomly coloring constant degree graphs. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 582–589, Oct 2004. 2

[FGY19] Weiming Feng, Heng Guo, and Yitong Yin. Perfect sampling from spatial mixing. *arXiv e-prints*, page arXiv:1907.06033, Jul 2019. 3

[GJL17] Heng Guo, Mark Jerrum, and Jingcheng Liu. Uniform Sampling Through the Lovász Local Lemma. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 342–355, New York, NY, USA, 2017. ACM. 3

[GJS76] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1(3):237 – 267, 1976. 1

[GK06] David Gamarnik and Dmitriy Katz. Correlation decay and deterministic fptas for counting colorings of a graph. *J Discr Algor*, 12, 07 2006. 3

[HN99] Olle Häggström and Karin Nelander. On exact simulation of markov random fields using coupling from the past. *Scandinavian Journal of Statistics*, 26(3):395–411, 1999. 4

[Hub98] Mark Huber. Exact sampling and approximate counting techniques. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 31–40, 1998. 2, 3, 4, 22

[Hub04] Mark Huber. Perfect sampling using bounding chains. *Ann. Appl. Probab.*, 14(2):734–753, 05 2004. 3, 4

[HV03] Thomas P. Hayes and Eric Vigoda. A non-markovian coupling for randomly sampling colorings. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '03, Washington, DC, USA, 2003. IEEE Computer Society. 2

[Jer95] Mark Jerrum. A very simple algorithm for estimating the number of k-colorings of a low-degree graph. *Random Structures & Algorithms*, 7(2):157–165, 1995. 2

[JVV86] Mark R. Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169 – 188, 1986. 1, 3

[LPW06] David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov chains and mixing times*. American Mathematical Society, 2006. 2

[LSS19] Jingcheng Liu, Alistair Sinclair, and Piyush Srivastava. A deterministic algorithm for counting colorings with 2Δ colors *(To appear in proceedings of IEEE FOCS 2019)*. *arXiv e-prints*, page arXiv:1906.01228, Jun 2019. 3

[LY13] Pinyan Lu and Yitong Yin. Improved fptas for multi-spin systems. In *APPROX-RANDOM*, 2013. 3

[MO94] F. Martinelli and E. Olivieri. Approach to equilibrium of glauber dynamics in the one phase region. i. the attractive case. *Comm. Math. Phys.*, 161(3):447–486, 1994. 1

[PW96] James Gary Propp and David Bruce Wilson. Exact sampling with coupled markov chains and applications to statistical mechanics. *Random Structures & Algorithms*, 9(1-2):223–252, 1996. 2

[SS97] Jesús Salas and Alan D. Sokal. Absence of phase transition for antiferromagnetic potts models via the dobrushin uniqueness theorem. *Journal of Statistical Physics*, 86(3):551–579, Feb 1997. 2

[Vig00] Eric Vigoda. Improved bounds for sampling colorings. *Journal of Mathematical Physics*, 41(3):1555–1569, 2000. 2

A Appendix

We state a useful lemma, which allows us to prove bounds on the number of steps required for the final phase of the algorithm to coalesce.

Lemma A.1. *Suppose that M_t is a random walk on $\{0, 1, \ldots, n\}$ where 0 is a reflecting state and n is an absorbing state. Suppose:*

- *e_i is the expected number of times the walk hits the state i.*
- $|M_{t+1} - M_t| \leq 1$
- $\Pr[M_{t+1} \neq M_t] > 0$
- $\mathbb{E}[M_{t+1} - M_t \mid M_t = i] \geq \kappa_i > 0$ *for all $M_t < n$.*

Then

$$\sum_{i=0}^{n} e_i \leq \sum_{i=0}^{n} \frac{1}{\kappa_i}$$

For a proof of this lemma we refer the reader to [Hub98, Theorem 4].

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il