Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 164 (2019)

Improved Bounds for Perfect Sampling of k-Colorings in Graphs

Siddharth Bhandari[*] Sayantan Chakraborty[*]

May 21, 2020

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 $\mathrm{poly}(k,n) = \widetilde{O}(n\Delta^2 \cdot \log(k))$. This improves upon a result of Huber (STOC 1998) who obtained a polynomial time perfect sampling algorithm for $k > \Delta^2 + 2\Delta$. Prior to our work, no algorithm with expected running time $\mathrm{poly}(k,n)$ was known to guarantee perfectly sampling with sub-quadratic number of colors in general.

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 Huber (STOC 1998) and Häggström & Nelander (Scand. J. Statist., 1999), we employ a novel bounding chain to derive our result for the graph coloring problem.

1 Introduction

A k-coloring of a graph is an assignment of colors from the set $[k] = \{1, 2, \ldots, k\}$ to the vertices so that adjacent vertices are assigned different colors. We consider the problem of randomly sampling colorings of a given graph. The input is a graph G and an integer k: our goal is to generate a k-coloring uniformly at random from the set of all k-colorings of G. 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

[*]Tata Institute of Fundamental Research, Mumbai. email: `{siddharth.bhandari,sayantan.chakraborty}@tifr.res.in`. Research of the authors supported by the Department of Atomic Energy, Government of India, under project no. 12-R&D-TFR-5.01-0500.

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

ISSN 1433-8092

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 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 k-coloring whose distribution is within ε (in total variation distance) of the uniform distribution on the set of all 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 $\operatorname{poly}(k, n, \log(1/\varepsilon))$ for all $\varepsilon > 0$ (the subscript $+$ in k_+ indicates that we allow some error). By showing that the Markov chain based on Glauber Dynamics mixes fast whenever $k > 2\Delta$, Jerrum [Jer95] established that $k_+(\Delta) \leq 2\Delta$[1]. 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 an alternative 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 [DFHV13] showed that $k_+(\Delta) \leq 1.49\Delta$; for planar graphs Hayes, Vera and Vigoda [HVV15] obtained the sub-linear bound $k_+(\Delta) \leq O(\Delta/\ln(\Delta))$.

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; 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 work which introduced the technique of *coupling from the past (CFTP)* to generate perfectly uniform samples; Levin, Peres and Wilmer [LPW17, Sec-

[1]Jerrum in [Jer95] mentions that the Glauber Dynamics mixes in polynomial time even when $k = 2\Delta$ and credits Frieze for this observation: hence, we have $k_+(\Delta) \leq 2\Delta - 1$.

tion 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 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 k-coloring of G in expected time $\mathrm{poly}(k, n)$. By applying CFTP with the bounding chain approach Huber [Hub98, Hub04], showed that $k_0(\Delta) \le \Delta^2 + 2\Delta$[2].

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 [GJL19]. 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. Recently, Feng, Guo and Yin [FGY19] extending the ideas of [GJL19] showed that Huber's result can be improved to obtain an expected polynomial time perfect sampling algorithm when $k \ge \Delta^2 - \Delta + 3$; their algorithm requires time $O(n \exp(\exp(\mathrm{poly}(k))))$. 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.

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 $\mathrm{poly}_{\Delta,k}(n)$ (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 efficiently sample perfectly. This approach when used together with the deterministic approximate counting algorithm of Gamarnik and Katz [GK12], yields $\widetilde{k_0}(\Delta) \le 2.78\Delta$ for triangle free graphs; approximate counting algorithms in subsequent works due to Lu & Yin [LY13], and Liu, Sinclair & Srivastava [LSS19], yield $\widetilde{k_0}(\Delta) \le 2.58\Delta$ and $\widetilde{k_0}(\Delta) \le 2\Delta$, 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

[2]Huber [Hub98] presents two different algorithms for sampling colorings which together imply that $k_0(\Delta) \le \min\left(\Delta^2 + 2\Delta, \frac{\Delta \ln n}{\ln \ln n}\right)$; however, the analysis of the algorithm which gives $k_0(\Delta) \le \frac{\Delta \ln n}{\ln \ln n}$ seems incomplete (the algorithm actually shows that $k_0(\Delta) \le r\Delta$, where r is the smallest natural number such that $r^r > n$; the journal version of the paper [Hub04] does not include this algorithm); an alternative algorithm based on similar ideas and achieving the same bound is described in detail in the preliminary version of this work: see arXiv:1909.10323v1.

for 'mixing' to take place unlike the other methods. In any case, none of these algorithms yield truly linear bounds on $k_0(\Delta)$.

1.2 Our contribution

Theorem 1.1 (Main result). $k_0(\Delta) \le 3\Delta$. *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 k-coloring of G. The algorithm uses fair independent unbiased coin tosses (with probability $1/2$ for head and tail), and stops in expected time $O((n \log^2 n) \cdot (\Delta^2 \log \Delta \log k))$.*

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 ideas that allow 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 [PW96] 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 $U_{-1} \circ U_{-2} \circ \cdots \circ U_t$ 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 $U_{-1} \circ U_{-2} \circ \cdots \circ U_t$ 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.

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 precisely keeping track of the various states that are reached after each update operation, 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 $U_{j-1}, \ldots, U_t$, we do not explicitly maintain the colors 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_j, σ_j). 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 definitely 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 $k > |S_{L_j}(v)|$ ($|S_{L_j}(v)|$ can potentially be as large as $\sum_{w \in N(v_j)} |L_j(w)|$ which in turn can be $\Delta \times (\Delta+1)$) 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. We omit the detailed analysis of Huber's method, but note 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; this implementation, therefore, yields only a quadratic bound on $k_0(\Delta)$.

We improve upon this using a better implementation of the Bounding Chain Method. After a short initial warm-up phase of updates (which we formally call the collapsing phase), 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 updates that implement the Glauber Dynamics. Recall that a sequence of random update operations $U_{-1}, U_{-2}, \ldots,$ need to be designed in the CFTP algorithm. In Huber's approach 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 sequence (or array) of random choices that describe the $i - j + 1$ 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. ($T = 2n \ln n(k-\Delta)/(k-3\Delta) + |E(G)| + n$ where $|E(G)|$ is the number of edges in G.)*

(a) A sample with distribution $\mathcal{D}$ 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 $U[-1,-T]$ is picked according to $\mathcal{D}$ independently of Z, then $U(-1,-T)(Z)$ is a uniformly distributed k-coloring;

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

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

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 $\Phi = \text{TRUE}$ 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

```
1 for $i = 0, 1, 2, \ldots,$ do
2     Generate $U[-iT-1, -(i+1)T]$ according to $\mathcal{D}$ ;
3     if $\Phi(U[-iT-1, -(i+1)T]) = \text{TRUE}$ then
4         Output the unique $k$-coloring in the image of $U(-1, -(i+1)T)$ and STOP;
5     end if
6 end for
```

Proof of Theorem 1.1. We wish to show that the output of the above algorithm is uniformly distributed in the set of all k-colorings. Let $U[-1,-T], U[-T-1,-2T], \ldots, U[-(i-1)T,-iT], \ldots$ be the random sequences that arise when the algorithm samples independently from the distribution $\mathcal{D}$. It may be that some of the later sequences are not used by the algorithm if the predicate Φ evaluates to true on an earlier sequence, but we define all of them anyway for our argument. Let χ be uniformly chosen k-coloring. Fix $i \geq 1$. Then, by Lemma 1.2 (b), $\chi_i = U(-1,-iT)(\chi)$ is uniformly distributed. Let χ^* be the output of the above algorithm. By Lemma 1.2 (c), χ^* and χ_i are identical whenever Φ evaluates to true on one of $U[-1,-T], U[-T-1,-2T], \ldots, U[-(i-1)T-1,-iT]$, which happens with probability at least $1-2^{-i}$ by Lemma 1.2 (d). From the duality of total variation distance and coupling, it follows that the distribution of χ^* is within 2^{-i} of the uniform distribution (the distribution of χ_i). Since, i was arbitrary, we see that the distribution of χ^* is uniform.

The algorithm is efficient[3] because it performs at most two iterations of the *for* loop in expectation, and sampling from $\mathcal{D}$ and the computation of Φ are efficient by part Lemma 1.2 (a). For a detailed analysis of the running time refer to Section 2.4. □

[3]Line 4 of the algorithm can be performed by taking the trivial coloring $\chi = 1^V$ and then outputting $U(-1,-(i+1)T)(\chi)$; however, in our implementation of the updates the condition $\Phi(U[-iT-1,-(i+1)T]) = \text{TRUE}$ will be validated by producing the unique coloring χ in the image of $U(-iT-1,-(i+1)T)$; so for Line 4 we output $U(-1,-iT)(\chi)$.

In other words, let i be the first index in Algorithm 1 such that we find $\Phi(U(-iT-1,-(i+1)T)) = \textsc{true}$. So, we know that $|\{U(-1,-T)(\chi) : \chi \text{ is a } k\text{-coloring}\}| = 1$. 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 $\mathrm{poly}(n,k)$ follow immediately from Lemma 1.2; in particular, this justifies Theorem 1.1 barring the expected running time. In the rest of this introduction, we describe the distribution $\mathcal{D}$ and outline our proof of the lemma

Representation of update operations: Our approach is inspired by the Bounding Chain method. 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 refer to $L(v)$ as v's list of colors; thus L is a list of lists. 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 important features 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 of 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 will implement the Glauber Dynamics at vertex v_t 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 $\textsc{Glauber Dynamics}(\chi, v_t)$. Note that this will ensure Lemma 1.2 (b).

[C3] The lists L_t impose 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 *precondition* 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, have in our sequence that $L'_t = L_{t+1}$. Also, L_{-T} will be $([k])^V$. If the above discipline concerning preconditions and post-conditions 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}; to establish Lemma 1.2(d), we will show that this condition holds with probability at least $\frac{1}{2}$.

The key ideas: We discussed above some of the conditions that our random sequence of tuples $\alpha[-1,-T]$ will satisfy. We now informally describe how α_t is translated or decoded to obtain U_t and how $\alpha[-1,-T]$ is generated. This informal description will differ slightly from the more formal one we 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.

Initially, at time $-T$, each vertex's list is $[k]$: that is, $L_{-T}(v) = [k]$ for all all v. We wish to ensure that in the end all lists have size 1: that is, $|L'_{-1}(v)| = 1$ for all v. We will achieve this in two phases. At the end of the first phase, we will ensure that all vertices have lists of size at most 2 with probability 1. We refer to this phase as the *collapse* phase. The second phase, the *coalesce* phase, will ensure that with probability at least 1/2, the lists of all vertices have size one. The total number of updates in the first and second phases put together will be T. We now briefly describe the ideas involved in the two phases.

The updates in these two phases will be generated by two primitives. (i) The first primitive takes vertex w and a set A of at most Δ colors and produces an update called *compress* update; after this update, the list at w will have at most one element outside A. (ii) The second primitive takes a vertex v and generates a random update called *contract* update; for this primitive to be used, we must ensure that the previous updates have spruced up the neighborhood of v which is said to have occurred when the union of colors in the lists of neighbors of v has size less than $k - \Delta$. But whenever such an update is performed, the list of v immediately contracts to size at most two; in fact, with significant probability it contracts to size one. We describe these primitives in detail in the following sections. For now, let us see roughly see how they are deployed to achieve the goals of the two phases.

Collapsing phase: The reduction in list size all the way to just two will be achieved by using contracting updates. However, for such an update to be applied at a vertex, the total number of colors in the union of the lists of its neighbors must be small (for us it will need to be less than $k - \Delta$; in fact, we will ensure that it is at most 2Δ). Note that our initial bounding list L_{-T} does not satisfy this condition; all lists have size k. We, therefore, need to first *spruce up* the neighborhood. Fix an ordering of the vertices, say $v_1, v_2, \ldots, v_n$[4]. Conceptually, the contracting phase will perform the actions in the following sequence:

$$\textsc{spruceup}(v_1), \textsc{contract}(v_1), \textsc{spruceup}(v_2), \textsc{contract}(v_2),$$

$$\ldots, \textsc{spruceup}(v_n), \textsc{contract}(v_n).$$

Here $\textsc{spruceup}(v_i)$ is a composite update operation. It consists of several updates that compress the lists at the neighbors of v_i using a common set A_i of Δ colors. For example, if v_1 has d_1 neighbors, then $\textsc{spruceup}(v_1)$ will consist of d_1 compress update operations, one for each of its neighbors. It is easy to see that then the union of the lists at v_1's neighbors will have at most 2Δ colors (each of the at most Δ neighbors will contribute at most one new color outside A_1)— the neighborhood of v_1 is thus spruced up. In general, for v_i the operation $\textsc{spruceup}(v_i)$ will perform the compress operation on those neighbors of v_i which are after v_i in the ordering. Once the neighborhood of v_i has been spruced up in this fashion, a single contract update ensures that the list size of v_i contracts to two. There is one subtlety, however. After contracting the lists of $v_1, \ldots, v_i$, when we proceed to spruce up the neighborhood of v_{i+1}, we only perturb the lists of vertices after v_{i+1} (in particular, the lists of vertices before v_{i+1} remain unperturbed): yet we need to ensure that the union of the lists at v_{i+1}'s neighbors (both preceding and succeeding) will have

[4]There is a notational overload here: earlier we had used v_i to denote the vertex chosen at time step i for the update operation, but now we mean it to be the i^{th} vertex in the ordering. This will be clear from the context.

at most 2Δ colors. So we choose the set A_{i+1} so that it includes at least one color from the lists of the neighbors where a contraction has already been achieved. In Section 2.2, we describe the collapsing phase in detail.

Coalescing phase: Suppose the collapsing phase has successfully contracted all lists to size at most two. Our goal now is to extend the above sequence with some more randomly generated updates so that with probability at least 1/2 the final list sizes all become one. We again use the contract update operation described above, this time exploiting the feature that it contracts lists to size just one with significant probability. However, while vertices with list size two can hope to see a reduction in their list size, a vertex whose list size is already one will, with some probability, acquire a list size of two. In this phase, we randomly pick vertices and perform a contract update on them. Note that a contract update never results in a list of size more than two; so, all neighborhoods stay spruced up at every point in the coalescing phase. If we track the number W_t, which is the number of vertices with list size 1 at time t, this quantity performs a random walk on the number line (between 0 and n, with n as absorbing)[5] 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 $\mathrm{poly}(n,k)$ steps, and helps us justify Lemma 1.2 (d). In Section 2.3, we describe the coalescing phase in detail.

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. Finally, in Section 2.4 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. The running time analysis of our algorithm is also presented in Section 2.4.

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 collapsing phase of our algorithm and $(\alpha_{-T'}, \ldots, \alpha_{-1})$ to the coalescing phase. 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. The reasons for these values will be clear in the following subsections.

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. As stated in the introduction, we have two types of updates, the compress update and the contract update. The generation and decoding methods are different

[5]Whenever for a vertex v all its neighbors have list size 1 then the contract update applied to v produces a list of size 1 at v.

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 describe later.

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. Fix a coloring χ. The operation U_α will attempt to recolor the vertex v (leaving the colors of the other vertices unchanged) by picking a color from the sequence M. The α we generated will have $L'(v) = M$ barring the order; 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:

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

In the following subsections we will consider αs of two types, depending on the size of M.

Type <u>compress</u> ($|M| = \Delta + 1$): Such an α will be used to spruce up the neighborhoods.

Type <u>contract</u> ($|M| \le 2$): Such an α will be used in the collapsing phase to contract the list sizes to size at most two, and again in the coalescing phase to make make all list sizes 1.

2.1.1 Compress updates

To specify the compress updates we will present two procedures: COMPRESS.GEN and COMPRESS.DECODE. The procedure COMPRESS.GEN takes a tuple $\alpha_{\mathrm{IN}} = (v_{\mathrm{IN}}, \tau_{\mathrm{IN}}, L_{\mathrm{IN}}, L'_{\mathrm{IN}}, M_{\mathrm{IN}})$, 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 type compress, and is the immediate successor of α_{IN} in our sequence of updates. The update operation corresponding to such a tuple is obtained using procedure COMPRESS.DECODE, which takes a tuple α_{f} (produced by COMPRESS.GEN) and a coloring $\chi \sim L'_{\mathrm{IN}}$, and produces another coloring, say χ'. Thus, the update operation $U_{\alpha_{\mathsf{f}}}$ is the map $\chi \mapsto$ COMPRESS.DECODE$[\alpha_{\mathsf{f}}, \chi]$. The following lemma describes the relationship between the two procedures, and their important properties.

Lemma 2.1. *Let $\alpha_{\mathrm{IN}} = (v_{\mathrm{IN}}, \tau_{\mathrm{IN}}, L_{\mathrm{IN}}, L'_{\mathrm{IN}}, M_{\mathrm{IN}})$ be an arbitrary 5-tuple, $v \in V$ 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 random tuple produced by COMPRESS.GEN$[\alpha_{\mathrm{IN}}, v, A]$, *then $L_{\mathsf{f}} = L'_{\mathrm{IN}}$, $L'_{\mathsf{f}}(u) = L_{\mathrm{IN}}(u)$ 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 $\chi' :=$ COMPRESS.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), *that is, $\chi'(w) = \chi(w)$, for all $w \neq v$, and $\chi'(v)$ is uniformly distributed[6] in the set of colors $[k] \smallsetminus \chi(N(v))$.*

[6]Note that the randomness in $\chi'(v)$ arises from the random choices made in generating α_{f} using COMPRESS.GEN$[\alpha_{\mathrm{IN}}, v, A]$.

(d) *Except for copying of the list L'_{IN}, the expected running time of* COMPRESS.GEN *is* $O(\Delta \log k + \log n)$. *The time needed to update a k-coloring* χ *using* COMPRESS.DECODE *is* $O(\Delta(\log \Delta \log k + \log n))$.

To prove this lemma, we need to specify COMPRESS.GEN and COMPRESS.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, COMPRESS.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 overlap with A. So we actually generate a random permutation of the input set A, say σ and append to it at the end a random color c_1 chosen from $[k] \setminus A$; thus α_{f} has the form $(v_{\mathsf{f}}, \tau_{\mathsf{f}}, L_{\mathsf{f}}, L'_{\mathsf{f}}, (\sigma, c_1))$; here τ_{f} will be chosen uniformly from $[0, 1]$; its role will become clear soon. This simple procedure is our COMPRESS.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 A. 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 A (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 whenever c_1 is a valid option, we replace $\chi(v)$ by c_1 with probability $(k - \Delta)/(k - |\chi(N(v))|)$ and with the remaining probability we use the first valid color from σ. To implement this acceptance sampling we pick a random number $\tau_{\mathsf{f}} \in [0, 1]$ and accept c_1 if it is at least the threshold $1 - (k - \Delta)/(k - |\chi(N(v))|)$. This is all that COMPRESS.DECODE does. We now present the code (which may be skipped) that implements what we discussed above and formally prove Lemma 2.1.

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

For part (b) consider any $\chi \sim L'_{\text{IN}} = L_{\mathsf{f}}$. Note that during COMPRESS.GEN we set $L'_{\mathsf{f}}(v) = A \cup \{c_1\}$ and M_{f} as (σ, c_1) and during COMPRESS.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 COMPRESS.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 COMPRESS.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 8 in COMPRESS.DECODE we have:

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

Algorithm 2: COMPRESS: generation and decoding

```
Function gen():
    Input  : $\alpha_{\mathrm{IN}} = (v_{\mathrm{IN}}, \tau_{\mathrm{IN}}, L_{\mathrm{IN}}, L'_{\mathrm{IN}}, M_{\mathrm{IN}})$, $v \in V$ and $A \subseteq [k]$ with $|A| = \Delta$
    Output: $\alpha_{\mathsf{f}} = (v_{\mathsf{f}}, \tau_{\mathsf{f}}, L_{\mathsf{f}}, L'_{\mathsf{f}}, M_{\mathsf{f}})$
    $\tau_{\mathsf{f}} \overset{R}{\leftarrow} [0,1]$ ; $\sigma \overset{R}{\leftarrow} S_A$; $c_1 \overset{R}{\leftarrow} [k] \setminus A$ ;
    $L'_{\mathsf{f}} \leftarrow L'_{\mathrm{IN}}$ ;
    $L'_{\mathsf{f}}(v) \leftarrow A \cup \{c_1\}$; $M_{\mathsf{f}} \leftarrow (\sigma, c_1)$ ;          // Appending $c_1$ at the end of $\sigma$
    return $\alpha_{\mathsf{f}} = (v, \tau_{\mathsf{f}}, L'_{\mathrm{IN}}, L'_{\mathsf{f}}, M_{\mathsf{f}})$
Function decode():
    Input  : $\alpha = (v, \tau, L, L', M)$ and a coloring $\chi \sim L$
    Output: $\chi' \sim L'$
    $\chi' \leftarrow \chi$;
    $p_\chi(v) \leftarrow 1 - \frac{k-\Delta}{k-|\chi(N(v))|}$;
    if $c_1 \notin \chi(N(v))$ and $\tau \geq p_\chi(v)$ then
        $\chi'(v) \leftarrow M[\Delta+1]$ ;  // $M$ has the form $(\sigma, c_1)$ where $\sigma$ is list of $\Delta$ colors.
    else
        $\chi'(v) \leftarrow$ 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 $|\chi(N(v))| \leq \Delta$.
    end if
    return $\chi'$
```

To prove part (d) note that for Compress.gen the operations with non-trivial running time are:

- Pick τ_{f} 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 $\mathcal{S}_A$.
- Pick c_1 uar from $[k] \setminus A$.
- Updating $L'_{\mathsf{f}}(v)$ and M_{f}.

With access to fair coins the first, second and the third operations require expected time $O(\Delta \log k)$. The fourth operation requires expected time $O(\Delta \log k + \log n)$. Hence the expected running time of Compress.gen is $O(\Delta \log k + \log n)$. For Compress.decode, notice that the *If* clause in Line 9-Line 14 takes time $O(\Delta(\log k + \log n))$. The *Else* clause finds the first color in $M[1:\Delta]$ which is not in $\chi(N(v))$: we implement this by first sorting the colors in $\chi(N(v))$ and then doing a binary search in the sorted list, sequentially for every color in $M[1:\Delta]$. Thus we conclude that the running time of Compress.decode is $O(\Delta(\log \Delta \log k + \log n))$. □

2.1.2 Contract updates

In this section we describe the tuples α of the type contract which reduce the list size at some vertex to ≤ 2, and with significant probability, make the list size 1. This type of updates will be applied both in the collapsing and the coalescing phase. As in Section 2.1.1, we will present two procedures: Contract.gen and Contract.decode. The procedure Contract.gen takes as input a tuple $\alpha_{\mathrm{IN}} = (v_{\mathrm{IN}}, \tau_{\mathrm{IN}}, L_{\mathrm{IN}}, L'_{\mathrm{IN}},$
$M_{\mathrm{IN}})$ and a vertex v with the promise that $|S_{L'_{\mathrm{IN}}}(v)| < k - \Delta$, and returns a random tuple α_{f} of type contract. The update operation corresponding to such a tuple is obtained using procedure Contract.decode, which takes a tuple α_{f} (produced by Contract.gen) and a coloring $\chi \sim L'_{\mathrm{IN}}$, and produces another coloring, say χ'. Thus, the update operation $U_{\alpha_{\mathsf{f}}}$ is the map $\chi \mapsto$ Contract.decode$[\alpha_{\mathsf{f}}, \chi]$. The following lemma describes the relationship between the two procedures, and their important properties.

Lemma 2.2. *Let $\alpha_{\mathrm{IN}} = (v_{\mathrm{IN}}, \tau_{\mathrm{IN}}, L_{\mathrm{IN}}, L'_{\mathrm{IN}}, M_{\mathrm{IN}})$ be an arbitrary 5-tuple and $v \in V$. Suppose $|S_{L_{\mathrm{IN}}}(v)| < k - \Delta$ 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* Contract.gen$[\alpha_{\mathrm{IN}}, v]$. *Then*

(a) Let $L = L'_{\mathrm{IN}}$. Then $L_{\mathsf{f}} = L$, $L'_{\mathsf{f}}(u) = L(u)$ 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 $\chi' \coloneqq$ Contract.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).

(d) Except for copying of the list L'_{IN} the expected running time of Contract.gen *is $O(\Delta(\log k + \log n))$. The time needed to update a k-coloring χ using* Contract.decode *is $O(\Delta(\log n + \log k))$.*

Algorithm 3: CONTRACT: generation and decoding

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

We now describe the ideas behind CONTRACT.GEN and CONTRACT.DECODE. Consider $\alpha_{\mathrm{IN}} = (v_{\mathrm{IN}}, \tau_{\mathrm{IN}}, L_{\mathrm{IN}}, L'_{\mathrm{IN}}, M_{\mathrm{IN}})$ and let $L = L'_{\mathrm{IN}}$. Also consider a vertex $v \in V$ and a coloring $\chi \sim L$. We wish to produce $\alpha_{\mathrm{f}} = (v_{\mathrm{f}}, \tau_{\mathrm{f}}, L_{\mathrm{f}}, L'_{\mathrm{f}}, M_{\mathrm{f}})$ with $L'_{\mathrm{f}}(v)$ of size at most 2 (with some probability of it being of size 1) and a coloring χ' such that $\chi' \sim L'_{\mathrm{f}}$, and χ' should be distributed according to GLAUBER DYNAMICS(χ, v). For now let us focus on producing $L'_{\mathrm{f}}(v)$ of size 2. Hence, without knowing $\chi(N(v)) \coloneqq \chi(N(v))$ we need to produce two colors $L'_{\mathrm{f}}(v) = \{c_1, c_2\}$ such that by choosing one of them (based on the coloring χ), we may ensure that $\chi'(v)$ is distributed uniformly over $[k] \setminus \chi(N(v))$. Notice that since $\chi \sim L$ we have $Q_L(v) \subseteq \chi(N(v)) \subseteq S_L(v)$. An initial attempt is to sample a color $c_1 \notin S_L(v)$ uar and insist that the update operation set $\chi'(v) = c_1$ (no matter what χ is). While this is a valid choice of color at v it is not necessarily distributed uniformly over $[k] \setminus \chi(N(v))$, because such an update places no mass on colors in $S_L(v) \setminus \chi(N(v))$. To remedy this situation we sample another color c_2 from $S_L(v) \setminus Q_L(v)$ uar and allow the update to choose between c_1 and c_2 depending on χ. In particular, we prescribe the update at v as follows. Let τ_{f} be chosen from $[0,1]$ uar and let p_χ be a threshold in $[0,1]$: if $\tau_{\mathrm{f}} \le p_\chi$ or $c_2 \in \chi(N(v))$ 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 $[k] \setminus \chi(N(v))$. A direct calculation (see proof of Lemma 2.2) shows that $p_\chi = 1 - \frac{|S_L(v)| - |Q_L(v)|}{k - |\chi(N(v))|}$. To ensure that with significant probability L'_{f} has size 1, we provide a threshold p_L such that always $p_L \le p_\chi$. Thus, whenever $\tau_{\mathrm{f}} \le p_L$ we set $L'_{\mathrm{f}} = \{c_1\}$. We let $p_L \coloneqq 1 - \frac{|S_L(v)| - |Q_L(v)|}{k - |\Delta|}$. The assumption $|S_L(v)| < k - \Delta$ implies that $p_\chi \ge p_L > 0$. Note that the threshold p_χ is computed after the actual coloring χ is available for update. Algorithm 3 (which may be skipped) is the code implementing the above ideas along with a proof of Lemma 2.2.

Proof of Lemma 2.2. Let $L = L'_{\mathrm{IN}}$. Note that $Q_L(v) \subseteq$
$\chi(N(v)) \subseteq S_L(v)$ as $\chi \sim L$. Also, p_L at Line 4 is at most p_χ at Line 13 in Algorithm 3. To prove part (a), observe that line 2 directly implies $L_{\mathrm{f}} = L$. It is also evident that throughout the execution of the algorithm, $L'_{\mathrm{f}}(u)$ is never updated, for all $u \neq v$, after the execution of line 2. This proves that $L'_{\mathrm{f}}(u) = L(u)$ for all $u \neq v$. Finally, we note that τ_{f} is distributed uniformly at random in $[0,1]$ and $L'_{\mathrm{f}}(v)$ is singleton iff $\tau_{\mathrm{f}} \le p_L$. Thus $L'_{\mathrm{f}}(v)$ is a singleton with probability exactly p_L. The claim follows by noting the value of p_L in line 4.

To prove part (b), note from CONTRACT.GEN that the set M_{f} (disregarding the ordering) is actually the same as the set $L'_{\mathrm{f}}(v)$ (Line 5-9). Also, we see from CONTRACT.DECODE that $\chi'_{\mathrm{f}}(v)$ is always contained within M_{f}. For all $w \neq v$, CONTRACT.DECODE sets $\chi'_{\mathrm{f}}(w)$ to $\chi(w)$. By the hypothesis of the Lemma, $\forall w \in V, \chi(w) \in L_{\mathrm{f}}(w)$. Finally, since CONTRACT.GEN sets $L'_{\mathrm{f}}(w)$ to $L(w)$ for all $w \neq v$, we conclude that the part (b) of the 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:

(i) 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 $S_L(v) \setminus \chi(N(v))$ (when $\chi' = c_2$).

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

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

Observation (i) implies that if $\Pr[\chi'(v) = c_1]$ turns out to be of the form $\frac{k-|S_L(v)|}{k-|\chi(N(v))|}$, it would imply that the distribution of $\chi'(v)$ is indeed uniform on the set $[k] \setminus \chi(N(v))$. Referring to CONTRACT.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 (c).

To prove part (d), notice that in CONTRACT.GEN, the only operations with non-trivial running time consist of :

- Pick τ_{f} 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 $S_L(v) \setminus Q_L(v)$
- Updating the list $L'_{\mathsf{f}}(v)$ and M_{f}.

With access to fair coins the expected running time of CONTRACT.GEN is $O(\Delta(\log k + \log n))$. For the running time of CONTRACT.DECODE, recall that we only update the color at the vertex v: the checking and update together take time $O(\Delta(\log k + \log n))$ which concludes the proof. □

2.2 Collapsing phase

The collapsing 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 collapsing phase we will generate a sequence $(\alpha_{-T}, \dots, \alpha_{-T'-1})$. Once the corresponding updates are applied, the list sizes of all the vertices will be brought down to at most two. As mentioned in the introduction, this reduction in list size will be achieved by updates of type contract; each such update will be preceded by a sequence of updates that spruce up the neighbourhood of the vertex whose list we wish to contract. To spruce up the neighborhood (recall that this happens when the union of the lists of the neighbors has size less than $k - \Delta$), we will repeatedly use compression; recall that the COMPRESS primitive accepts a set of colors A of size Δ and ensures that, after the update is applied, the list of the updated vertex has at most one color outside A. While doing so, we must ensure that the lists of vertices that have already been collapsed are not disturbed. Let $V = \{v_1, \dots, v_n\}$, $N_>(v_i) := \{v_j \in N(v) \mid j > i\}$ and $N_<(v_i) := \{v_j \in N(v) \mid j < i\}$. To spruce up the neighborhood of v_i, we will compress the lists of vertices in $N_>(v_i)$ and not in $N_<(v_i)$. Yet we need to ensure that the entire neighborhood, i.e., $N_<(v_i) \cup N_>(v_i)$ is spruced up; so the set for sprucing up the neighborhood of v_i, A will be chosen such that it includes at least one element from the list of each $w \in N_<(v_i)$. The following code implements this.

Lemma 2.3. *Let $\alpha[-1, -|N_>(v_i)|]$ be the output of the algorithm* SPRUCEUP$[\alpha_{\mathrm{IN}}, i]$ *and let $\alpha[-1] = (v, \tau, L', L'', M)$. Then, (a) for $w \notin N_>(v_i)$: $L''(w) = L'(w)$; (b) $\bigcup_{w \in N(v_i)} |L(w)| \le 2\Delta$.*

Proof. Part (a) is true because the list of no vertex outside $\{v_i\} \cup N_>(v_i)$ is perturbed by the algorithm. Part (b) follows as the at most Δ neighbors of v_i can each contribute to the union at most one color outside the set A . □

By successively sprucing up the neighborhood and contracting the lists of all vertices in V, we complete the collapsing phase. The following code implements this formally; here we adopt the notation, that if α is a sequence of update tuples, then $\alpha[\text{LAST}]$ is the tuple in this list corresponding to the latest update.

Algorithm 4: SPRUCEUP

```
Input  : $\alpha_{\mathrm{IN}} = (v_{\mathrm{IN}}, \tau_{\mathrm{IN}}, L_{\mathrm{IN}}, L'_{\mathrm{IN}}, M_{\mathrm{IN}}), i \in [n]$
1 Promise: for all $j < i$: $|L'_{\mathrm{IN}}(v_j)| \leq 2$
  Output: $\alpha[-1, -|N_>(v_i)|]$
2 $t \leftarrow -|N_>(v_i)|$; $L \leftarrow L'_{\mathrm{IN}}$ ;
3 Pick a $\Delta$-element subset $A$ of $[k]$ that intersects every set in $\{L(w) : w \in N_<(v_i)\}$;
4 for $w \in N_>(v_i)$ do
5     $\alpha_t \leftarrow$ COMPRESS.GEN$[\alpha_{t-1}, w, A]$;
6     $t \leftarrow t + 1$;
7 end for
8 return $\alpha[-1, -|N_>(v_i)|]$
```

Algorithm 5: COLLAPSE

```
  Output: $\alpha[-1, -(T - T')]$
1 $\alpha_{\mathrm{LAST}} \leftarrow (v_1, 0, [k]^V, [k]^V, ())$ ;
2 $\alpha \leftarrow$ empty ;
3 for $i = 1, 2, \ldots, n$ do
4     $\alpha \leftarrow$ SPRUCEUP$[\alpha_{\mathrm{LAST}}, i] \circ \alpha$;
5     $\alpha_{\mathrm{LAST}} \leftarrow$ CONTRACT.GEN$[\alpha[\mathrm{LAST}], v_i]$;
6     $\alpha \leftarrow \alpha_{\mathrm{LAST}} \circ \alpha$;
7 end for
8 return $\alpha$
```

Lemma 2.4. *The collapsing phase lasts for $T - T' = |E(G)| + n$ steps. Let $\alpha[-1, -(T - T')]$ be the output of* COMPRESS[] *and let $\alpha_{-1} = (v, \tau, L', L'', M)$. Then, for all $w \in V$ we have $|L''(w)| \leq 2$.*

Proof. When v_i is chosen for update, only v_i 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 $T - T' = |E(G)| + n$, where $E(G)$ is the edge set of the graph G. The remaining part of the claim is obvious based on previous lemmas. □

2.3 Coalescence phase

The collapsing phase produces a random sequence of updates, say α, at the end of which the lists of all vertices have size at most two. We now propose to follow this up by a another sequence $(\beta_{-1}, \ldots, \beta_{-T'})$ and ensure that $|L_0(v)| = 1$ for all $v \in V$, with probability at least 1/2. As stated in the introduction, this is achieved by applying contracting updates T' times at vertices chosen uniformly at randomly. More precisely, let $w[-1, -T'] = (w_{-T'}, w_{-T'+1}, \ldots, w_{-1})$ be chosen uniformly from $V^{T'}$: the random sequence $\beta[-1, -T']$ is obtained using the random process $\beta[-T'] \leftarrow$ CONTRACT.GEN$[\alpha[\text{LAST}], w_{-T'}]$, and $\beta[-i+1] \leftarrow$ CONTRACT.GEN$[\beta[-i], w_{i+1}]$, for $i = -T', -T'+1, \ldots, -2$. Note that after the collapsing phase all the neighborhoods are spruced up (since each list is of size at most 2), and thus further application of the contract updates leaves all the neighborhoods spruced up, which is the case for the entirety of the coalescence phase.

Algorithm 6: COALESCENCE

```
  Input  : $\alpha_{in} = (v_{\text{IN}}, \tau_{\text{IN}}, L_{\text{IN}}, L'_{\text{IN}}, M_{\text{IN}})$
1 Promise: for all $v \in V$: $|L'_{\text{IN}}(v)| \leq 2$
  Output: $\beta[-1, -T']$
2 $\alpha_{-T'-1} \leftarrow \alpha_{\text{IN}}$;
3 for $t = -T', \ldots, -1$ do
4     $v \overset{R}{\leftarrow} V$;
5     $\beta_t \leftarrow$ CONTRACT.GEN$[\alpha_{t-1}, v]$
6 end for
7 return $\beta[-1, -T']$
```

Recall that after the collapsing updates, the list sizes have a significant probability reducing to 1 from 2; this is progress. However, it can also be the case that when an update is performed at a vertex with list size 1, its list size become 2.
Lemma 2.2 shows that if the vertex has many neighbors with singleton lists, then it has a greater chance of acquiring a singleton list; in particular, if all its neighbors have list size 1, then it definitely acquires a singleton list after the update. To track our progress, we define $W_t := \{v \mid |L_t(v)| = 1\}$ (earlier we had defined W_t to be the number of vertices of list size 1). Then, $|W_t|$ performs a random walk on $[0, n]$. Lemma 2.5 establishes that this walk has a drift towards n, and that this walk reaches the absorbing state n with probability at least 1/2.

Lemma 2.5. *Assume $k > 3\Delta$ and let $T' = 2\frac{k-\Delta}{k-3\Delta} n \ln n$. Suppose the last update of the collapse phase has the form $\alpha[\text{LAST}] = (v_n, \tau, L, L', M)$ such that $|L'(v)| \le 2$, for all $v \in V$. Let $\beta[-1, -T']$ be the random sequence of updates generated by the above process for the coalescence phase, starting from $\alpha[\text{LAST}]$. Suppose $\beta[-1]$ has the form $(.,.,.,L_0,.)$. Then, with probability at least $1/2$, we have for all $v \in V : |L_0(v)| = 1$.*

To prove this we will require the following claim.

Claim 2.6 ([Hub98, Theorem 4]). *Suppose that X_t is a random walk on $\{0, 1, \dots, n\}$ where 0 is a reflecting state and n is an absorbing state. Further $|X_{t+1} - X_t| \le 1$, and $\mathbb{E}[X_{t+1} - X_t \mid X_t = i] \ge \kappa_i > 0$ for all $X_t < n$. Let e_i is the expected number of times the walk hits the state i. Then*

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

Proof of Lemma 2.5. For $t = -T', \dots, -1, 0$, let $W_t := \{v : |L_t(v)| = 1\}$, let $X_t = |W_t|$ and $\delta_t = X_{t+1} - X_t$. Note that X_t is a random variable based on the random choice of β. We will use Lemma 2.2 to establish

$$\mathbb{E}[X_{t+1} - X_t \mid X_t] \ge \frac{n - X_t}{n}\left(1 - \frac{2\Delta}{k - \Delta}\right). \tag{1}$$

Note that the drift is positive if $2\Delta < k - \Delta$, that is, $k > 3\Delta$. Then, our lemma follows immediately from Eq. (1), Claim 2.6 and Markov's inequality.

It remains to establish Eq. (1). Let L be the lists at time t. We have the following.

$X_{t+1} - X_t = 1$ iff w_t (the vertex updated in step t) has list size 2 and then its list size becomes 1 after the update. By Lemma 2.2, the last event happens with probability $(1 - (|S_L(w)| - |Q_L(w)|)/(k - \Delta))$.

$X_{t+1} - X_t = -1$ iff w_t has list size 1 and then its list size becomes 2 after the update. By Lemma 2.2, this happens with probability $(|S_L(w)| - |Q_L(w)|)/(k - \Delta)$.

Note that $|S_L(v)| - |Q_L(v)| \le 2|N(v) \cap \overline{W}_t|$, so

$$\sum_{v} |S_L(v)| - |Q_L(v)| \le 2|\overline{W}_t|\Delta. \tag{2}$$

Thus,

$$\begin{aligned}
&\mathbb{E}[X_{t+1} - X_t \mid W_t] \\
&= \frac{1}{n}\left[\sum_{v \notin W_t}\left(1 - \frac{|S_L(v)| - |Q_L(w)|}{k - \Delta}\right) - \sum_{v \in W_t} \frac{|S_L(v)| - |Q_L(w)|}{k - \Delta}\right] \\
&= \frac{1}{n}\left[|\overline{W}_t| - \sum_{v \in V} \frac{|S_L(v)| - |Q_L(w)|}{k - \Delta}\right] \\
&\ge \frac{1}{n}\left[|\overline{W}_t| - \frac{2|\overline{W}_t|\Delta}{k - \Delta}\right] \\
&= \frac{n - X_t}{n}\left[1 - \frac{2\Delta}{k - \Delta}\right].
\end{aligned}$$

The claim follows from this.

□

2.4 Proof of Lemma 1.2 and running time analysis of Algorithm 1

Let $\alpha[-T'-1,-T]$ be the random sequence of update tuples of length $T-T' = |E(G)|+n$ produced in the collapse phase; let $\beta[-1,-T']$ be the random sequence of update tuples of length T' produced in the coalescence phase. Our final sequence of tuples will be $\alpha[-1,T] := \beta[-1,-T'] \circ \alpha[-T'-1,-T]$, where $T := |E(G)|+n+T'$. Let $U[-1,-T]$ be the sequence of updates corresponding to $\alpha[-1,-T]$, obtained by applying the appropriate *decode* procedure to each tuple in $\alpha[-1,-T]$. 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 used in the various algorithms 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 U_t 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.

Finally, we justify Theorem 1.1 by analyzing the expected running time of Algorithm 1. Let i be the first index where $\Phi(U[-iT-1,-(i+1)T]) = \text{TRUE}$) and let $\chi = L'_{-iT-1}$ be the unique coloring in the image of $U(-iT-1,-(i+1)T)$. Notice that a particular block of updates $U[-jT-1,-(j+1)T]$ (where $j < i$) is processed twice by Algorithm 1: once during generation of $U[-jT-1,-(j+1)T]$ and once while computing $U(-1,-iT)(\chi)$. For applying the function $U(-jT-1,-j(T+1))$, we need to invoke both COMPRESS.DECODE and CONTRACT.DECODE which take as input a tuple $\alpha = (v,\tau,L,L',$
$M)$ and a coloring χ. Notice that both these procedures actually never require the lists L and L'. Hence, during the generation of $U[-1,-(i+1)T]$ which corresponds to generating the sequence $\alpha[-1,-(i+1)T]$, we implement the changes performed to the lists by COMPRESS.GEN and CONTRACT.GEN in-place without creating new lists. Thus, during the execution of COMPRESS.GEN and CONTRACT.GEN we skip the step of copying the lists.

To calculate the expected time needed to generate $U[-jT-1,-(j+1)T]$ we analyze the expected time needed to generate the updates corresponding to the two phases. The collapse phase involves generating $|E(G)|+n$ updates during which we call COMPRESS.GEN $|E(G)|$ times and CONTRACT.GEN n times. Recall from part (d)s of Lemma 2.1 and Lemma 2.2 that the expected running times for both COMPRESS.GEN and CONTRACT.GEN are $O(\Delta(\log k+\log n))$. Hence, the expected time needed for the collapse phase is $O(n\Delta^2(\log k+\log n))$. The coalescence phase involves calling CONTRACT.GEN $2\frac{k-\Delta}{k-3\Delta}n\ln n$ times. Hence, the expected time needed for the coalescence phase is $O((n\log n)\Delta^2(\log k+\log n))$ and the overall expected time for generating $U[-jT-1,-(j+1)T]$ is $O((n\log n)\Delta^2(\log k+\log n))$.

To calculate the time needed to decode $U(-jT-1,-(j+1)T)$ observe that we need to call COMPRESS.DECODE $|E(G)|$ times and CONTRACT.DECODE $n+2\frac{k-\Delta}{k-3\Delta}n\ln n$ times. Recall from part (d)s of Lemma 2.1 and Lemma 2.2 that the expected running times for both COMPRESS.DECODE and CONTRACT.DECODE are $O(\Delta(\log\Delta\log k+\log n))$. Hence, the running time of decoding $U(-jT-1,-(j+1)T)$ is $O((n\log n)\Delta^2(\log\Delta\log k+\log n))$.

By part (d) of Lemma 1.2 on expectation the value of j is 2 the overall expected running time of Algorithm 1 is $O((n\log^2 n)\cdot(\Delta^2\log\Delta\log k))$.

3 Bottleneck for achieving $k > 2\Delta$

As the coupling proofs for efficient approximate sampling of colorings work all the way to the bound of $k > 2\Delta$, it seems natural to ask if we can obtain an efficient perfect sampler which works with $k > 2\Delta$. In this current framework we have two primitives namely, COMPRESS and CONTRACT, 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 application of compress updates to spruce up the neighborhood of a vertex v we are able to guarantee that $|S_L(v)| \leq 2\Delta$: however, if $k \leq 3\Delta$ this is not enough to meet the input requirement for CONTRACT, i.e., $|S_L(v)| < k-\Delta$.

Secondly, even if we somehow manage to apply CONTRACT 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.

Acknowledgements

We are grateful to Piyush Srivastava for introducing us to this problem and for the numerous detailed discussions which led to this work. We also thank him for pointing us to the relevant references. Finally, we are indebted Prahladh Harsha and Jaikumar Radhakrishnan with whom we had many discussions that helped us organize our ideas and put them in a presentable form that is the current write-up.

References

[Bar16] ALEXANDER I. BARVINOK. *Combinatorics and Complexity of Partition Functions*, volume 30 of *Algorithms and Combinatorics*. Springer, 2016. 3

[BD97] RUSS BUBLEY and MARTIN E. DYER. *Path coupling: A technique for proving rapid mixing in markov chains*. In *Proc. 38rd IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 223–231. 1997. 2

[CDM+19] SITAN CHEN, MICHELLE DELCOURT, ANKUR MOITRA, GUILLEM PERARNAU, and LUKE POSTLE. *Improved bounds for randomly sampling colorings via linear programming*. In *Proc. 30th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 2216–2234. 2019. arXiv:1810.12980. 2

[DFHV13] MARTIN E. DYER, ALAN M. FRIEZE, THOMAS P. HAYES, and ERIC VIGODA. *Randomly coloring constant degree graphs*. Random Structures Algorithms, 43(2):181–200, 2013. (Preliminary version in *45th FOCS*, 2004). `eccc:2004/TR04-009`. 2

[FGY19] WEIMING FENG, HENG GUO, and YITONG YIN. *Perfect sampling from spatial mixing*, 2019. (manuscript). `arXiv:1907.06033`. 3

[GJL19] HENG GUO, MARK JERRUM, and JINGCHENG LIU. *Uniform sampling through the Lovász Local Lemma*. J. ACM, 66(3):18:1–18:31, 2019. (Preliminary version in *49th STOC*, 2017). `arXiv:1611.01647`. 3

[GJS76] MICHAEL R. GAREY, DAVID S. JOHNSON, and LARRY J. STOCKMEYER. *Some simplified NP-complete graph problems*. Theoret. Comput. Sci., 1(3):237–267, 1976. (Preliminary version in *6th STOC*, 1974). 2

[GK12] DAVID GAMARNIK and DMITRIY KATZ. *Correlation decay and deterministic FPTAS for counting colorings of a graph*. J. Discrete Algorithms, 12:29–47, 2012. (Preliminary version in *18th SODA*, 2007). `arXiv:math/0606143`. 3

[HN99] OLLE HAGGSTROM 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 *Proc. 30th ACM Symp. on Theory of Computing (STOC)*, pages 31–40. 1998. 2, 3, 4, 5, 19

[Hub04] ———. *Perfect sampling using bounding chains*. Ann. Appl. Probab., 14(2):734–753, 2004. `arXiv:math/0405284`. 3, 4, 5

[HV03] THOMAS P. HAYES and ERIC VIGODA. *A Non-Markovian coupling for randomly sampling colorings*. In *Proc. 44th IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 618–627. 2003. 2

[HVV15] THOMAS P. HAYES, JUAN CARLOS VERA, and ERIC VIGODA. *Randomly coloring planar graphs with fewer colors than the maximum degree*. Random Structures Algorithms, 47(4):731–759, 2015. (Preliminary version in *39th STOC*, 2007). `arXiv:0706.1530`. 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):157–166, 1995. 2

[JVV86] MARK JERRUM, LESLIE G. VALIANT, and VIJAY V. VAZIRANI. *Random generation of combinatorial structures from a uniform distribution*. Theoret. Comput. Sci., 43:169–188, 1986. 1, 3

[LPW17] DAVID A. LEVIN, YUVAL PERES, and ELIZABETH L. WILMER. *Markov Chains and Mixing Times*. Amer. Math. Soc., 2nd edition, 2017. 3

[LSS19] JINGCHENG LIU, ALISTAIR SINCLAIR, and PIYUSH SRIVASTAVA. *A deterministic algorithm for counting colorings with 2Δ colors*. In *Proc. 60th IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 1380–1404. 2019. `arXiv:1906.01228`. 3

[LY13] PINYAN LU and YITONG YIN. *Improved FPTAS for multi-spin systems*. In PRASAD RAGHAVENDRA, SOFYA RASKHODNIKOVA, KLAUS JANSEN, and JOSÉ D. P. ROLIM, eds., *Proc. 17th International Workshop on Randomization and Computation (RANDOM)*, volume 8096 of *LNCS*, pages 639–654. Springer, 2013. 3

[MO94] FABIO MARTINELLI and ENZO 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, 4

[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, 1997. `arXiv:cond-mat/9603068`. 2

[Vig00] ERIC VIGODA. *Improved bounds for sampling colorings*. Journal of Mathematical Physics, 41(3):1555–1569, 2000. (Preliminary version in *40th FOCS*, 1999). 2