Entropy Compression on Frugal Colouring

Ziyang Jin[1]

Department of Computer Science
University of Toronto
ziyang@cs.toronto.edu

Abstract. In Molloy's talk[1], it is mentioned that any graph can be 2Δ-coloured such that every colour appears at most ℓ times in each neighborhood, for $\ell = \frac{10 \ln \Delta}{\ln \ln \Delta}$. Molloy mentioned that there is a simple proof using entropy compression. In this note, we will write the proof in details. We will also discuss different colouring procedures. In the end, we will apply the theorem by Achlioptas and Iliopoulos to get the same result.

Keywords: Entropy Compression · Frugal Colouring.

1 The problem

Theorem 1. *We can 2Δ-colour any graph such that each colour appears at most ℓ times in each neighborhood, for*

$$\ell = \frac{10 \ln \Delta}{\ln \ln \Delta}$$

This theorem is adapted from a similar theorem in [2].

By Brooks' theorem, we know that any graph can be coloured using $\Delta + 1$ colours, so 2Δ colours are more than enough to obtain a proper colouring.

The constant 10 here is not the best constant that can be achieved by entropy compression. In fact, we can get it down to 1 following the entropy compression approach.

2 Entropy Compression

To apply entropy compression, there are two parts. One is the colouring algorithm, and the other is the analysis.

2.1 Initial Colouring

First of all, we must start from some initial colouring of the graph. It would be nice that we start from a proper colouring. Otherwise, in our recolouring step, in addition to dealing with a neighborhood with the same colour repeated too many times, we might also need to deal with monochromatic edges.

Since we have 2Δ colours. It is easy to start from any vertex, give it an arbitrary colour, and then we colour the neighbors greedily (where we do not pay attention to how many times the same colour appears in a neighborhood) until we colour the whole, which gives us a proper colouring of the graph to start with.

2.2 Recolouring Algorithm

We need to propose a randomized recolouring algorithm to recolour the vertices until the target property is satisfied. Entropy compression will help us prove that the proposed randomized algorithm terminates with high probability, which means a colouring satisfying the given property exists. We make no attempt to optimize the runtime of the algorithm.

The main idea of our algorithm is to keep recolouring the neighborhood of the vertex who has more than ℓ neighbours of the same colour.

For any vertex $v \in V$, let N_v be the neighborhood of v (not including v). Let $B(v, j, p)$ be the (bad) event that vertex v has ℓ neighbours $u_1, ..., u_\ell$ with the same colour j, where $1 \leq j \leq 2\Delta$. There are $\binom{\Delta}{\ell}$ ways for $u_1, ..., u_\ell$ to lie in the neighborhood of v, and the number p denotes the pth combination out of them.

For any vertex $v \in V$, let L_v be the list of colours that do not appear in its neighborhood, i.e., list of colours available to v if we want to recolour v while still maintaining a proper colouring.

Here is our recolouring algorithm:

```
FIX(G):
    Let S be the set of all B(v,j,p)'s
        in the initial colouring
    While there exists any B(v,j,p) ∈ S:
        Write down "Fix B(v,j,p)" to the log
        RECOLOUR(B(v,j,p))

RECOLOUR(B(v,j,p)):
    For i from 1 to ℓ
        Recolour u_i with a uniformly random colour in L_{u_i}
        Let the new colour assignment be σ(u_i)

    While there exists any bad event B(w,σ(u_i),q) for
    some 1 ≤ i ≤ ℓ and w ∈ N_{u_i}, 1 ≤ q ≤ C(Δ, ℓ-1):
        Write down "Fix B(w,σ(u_i),q)" to the log
        RECOLOUR(B(w,σ(u_i),q))

    Write down "Return" to the log
```

We make three observations:

1. If a vertex v has $l \gg \ell$ neighbours with the same colour j, then we can divide them into $B(v, j, p_1), B(v, j, p_2), ..., B(v, j, p_{l/\ell})$, where we divide l neighbours into disjoint sets of size ℓ.

2. When we recolour the neighbours $u_1, ..., u_\ell$, we recolour them one at a time. After recolouring u_1, the lists $L_{u_2}, ..., L_{u_\ell}$ might be updated if it has an edge connected to u_1. The list of the colours available to the next vertex depends on the colour assignment of previous vertices. This is the key challenge in this algorithm that makes tools like Lovász Local Lemma difficult to apply as we have a series of dependent events, but entropy compression would skip the discussion of dependent events.

3. After we recolour $u_1, ..., u_\ell$, it is possible that we create new bad events. For example, one of u_i's neighbour w originally had exactly $\ell - 1$ neighbours of colour j, and $\sigma(u_i) = j$, and now adding u_i, it just passes the threshold and a new bad event $B(w, \sigma(u_i), q)$ occurs. Since we know one of w's neighbour is u_i, we only need to choose from $\binom{\Delta}{\ell-1}$ combinations to show how the neighbourhood looks like.

2.3 Analysis

In order to make entropy compression work, we need to find a representation such that:

- We can recover the whole history of the colouring
- Our representation will eventually be shorter than the random bits generated by a random number generator

Every time we call `RECOLOUR`, the algorithm will need at least $\ell \log \Delta$ bits from the random number generator since every $|L_{u_i}| \geq \Delta$. This is the target to beat.

Every time we call `RECOLOUR`, we say the algorithm takes a *step*.

A shorter Representation Our representation consists of four parts:

1. Representation of `Fix` $B(v, j, p)$. For every vertex, summing up all the "bad-event" colours in the neighborhood, there are at most $\frac{\Delta}{\ell}$ bad events. So there are at most $|V|\frac{\Delta}{\ell}$ bad events in S. And for each bad event, we need $\log |V|$ bits for v, $\log 2\Delta$ bits for j, and $\log \binom{\Delta}{\ell}$ bits for p, so in total we need $\log |V| + \log 2\Delta + \log \binom{\Delta}{\ell}$ to represent a single event. The while-loop in the `FIX` procedure will iterate at most $|V|\frac{\Delta}{\ell}$ times. So the total amount of bits required to write all the `Fix` $B(v, j, p)$'s through the execution of the algorithm is independent of the number of steps the algorithm have.
2. Representation of `Fix` $B(w, \sigma(u_i), q)$ at every step. This is the main thing we are going to discuss. Please see below.
3. We use a constant number of bits to encode each `RETURN`.
4. At time t, the state-of-the-world, i.e., the current colouring of the graph σ_t. This takes $|V| \log 2\Delta$ bits.

Here is our representation of `Fix` $B(w, \sigma(u_i), q)$:

1. We use a number i ($1 \le i \le \ell(\Delta-1)+1$) to represent the vertex w, which is the ith neighbour in the distance-two neighborhood of the previous vertex v where `RECOLOUR` was called on. When we call `RECOLOUR`, it recolours $u_1, ..., u_\ell$. Recolouring any u_i could potentially cause u_i's neighbour to have a bad event. In addition to the common neighbour v, each u_i can have at most $\Delta - 1$ other neighbours. So in total there are at most $\ell(\Delta-1)+1$ $B(w, \sigma(u_i), q)$'s that can happen. The $+1$ is for bad event on vertex v itself.
2. We use a number p to represent the pth combination of w's neighbours, where $1 \le p \le \binom{\Delta}{\ell-1}$. When we recolour $u_1, ..., u_\ell$, it may cause some neighbour w of some u_i to have a bad event. In addition to known neighbour u_i, we need to choose $\ell - 1$ other neighbours out of at most Δ neighbours of w to fix the pattern of the same colour neighbours.
3. We use a number j to represent the same colour that $u_1, ..., u_\ell$ all have, where $1 \le j \le 2\Delta$ since we have 2Δ colours.

Claim. We are able to recover the colouring of the graph at every step from our representation.

Proof. First, we go through the log from the beginning to the end. It will tell us the sequence of vertices $v_1, v_2, ..., v_t$ where we call `RECOLOUR` on.

Then, we go through the log reversely from the end to the beginning. We have σ_t, the state of the world at step t. And then we look at v_t, and the log `Fix` $B(v_t, c, p)$ where c is the colour and p is the pattern of the ℓ neighbours lying around v_t. By assigning c back to $u_1, ..., u_\ell$, we get σ_{t-1}.

Keep doing this, we will recover $\sigma_{t-2}, \sigma_{t-3}, ..., \sigma_1$, which will tell us all the random bits used in the algorithm. □

Now we need to show that our representation is eventually shorter than the string of random bits. We win if our representation saves some bits at every step. Specifically, what grows with the number of steps is our representation of `Fix` $B(w, \sigma(u_i), q)$. If we use fewer bits than $\ell \log 2\Delta$, then we save some bits at every step. Eventually, we will pay off the overhead (the `Fix` $B(v, j, p)$'s and σ_t) and end up with a shorter representation than the random bits.

Compare with the random bits used at every step Every time `RECOLOUR` is called, our representation for $B(w, \sigma(u_i), q)$ takes:

$$\log(\ell(\Delta-1)+1) + \log\binom{\Delta}{\ell-1} + \log 2\Delta + O(1)$$

bits. Now we try to compare it with $\ell \log 2\Delta$, the random bits used at every step. Note that:

$$\begin{aligned}
&\log(\ell(\Delta-1)+1) + \log\binom{\Delta}{\ell-1} + \log 2\Delta + O(1) \\
\le\ &\log\ell + \log\Delta + (\ell-1)\log\frac{e\Delta}{\ell-1} + \log\Delta + O(1) \qquad (1)\\
\le\ &(\ell+1)\log\Delta + \log\ell - (\ell-1)\log(\ell-1) + \ell\log e + O(1) \\
\le\ &(\ell+1)\log\Delta - \ell\log(\ell-1) + \ell\log e + 2\log\ell + O(1)
\end{aligned}$$

Is it shorter than the random bits generated every step $\ell \log \Delta$?

$$\begin{aligned} & \ell \log \Delta - [(\ell+1)\log \Delta - \ell \log(\ell-1) + \ell \log e + 2\log \ell + O(1)] \\ =& -\log \Delta + \ell \log(\ell-1) - (\log e)\ell - O(\log \ell) \\ =& \ell \log(\ell-1) - \log \Delta - O(\ell) \end{aligned} \tag{2}$$

So it comes down to whether $\ell \log(\ell-1) - \log \Delta > 0$ and $\ell \log(\ell-1) - \log \Delta = O(\log \Delta)$. Take $\ell = \frac{10 \ln \Delta}{\ln \ln \Delta}$, we have

$$\ell \log(\ell-1) = \frac{10 \ln \Delta}{\ln \ln \Delta} \log(\frac{10 \ln \Delta}{\ln \ln \Delta} - 1) = 10 \log \Delta + O(\frac{\ln \Delta \log \ln \ln \Delta}{\ln \ln \Delta})$$

Actually, we can bring ℓ down to as low as $(1+\epsilon)\frac{log\Delta}{\log \log \Delta}$ for any $\epsilon > 0$ and it would still let us save about $\epsilon \log \Delta$ bits at every step.

Therefore, when Δ is big enough, $\ell \log(\ell-1) - \log \Delta - O(\ell) \geq 1$. Our representation roughly saves $O(\log \Delta)$ bits every time. Thus, we succeed with the entropy compression. The randomized recolouring algorithm will terminate with high probability since it is impossible to represent a string of random bits using a shorter representation.

3 A different recolouring algorithm

In the previous section, our recolouring algorithm is mainly recolouring only the same-colour vertices in the neighborhood. In this section, we propose a different recolouring algorithm — we recolour the entire neighborhood (including the "good" vertices), and we will see if we can still get a shorter representation.

For any vertex $v \in V$, a bad colouring is a colouring where there are $\geq \ell$ vertices in the neighborhood of v that have the same colour.

Let $B(v)$ be the bad event that vertex v has $\geq \ell$ neighbours with the same colour, and let $\sigma(N_v)$ be the colouring on the neighborhood of v.

For every vertex $v \in V$, let $d(v)$ be the degree of v. Let the neighborhood of v be $u_1, ..., u_{d(v)}$.

Here is our recolouring algorithm:

```
FIX(G):
    Let S be the set of all B(v)'s
        in the initial colouring
    While there exists any B(v) ∈ S:
        Write down "Fix v,σ(N_v)" to the log
        RECOLOUR(B(v))

RECOLOUR(B(v)):
    For i from 1 to d(v)
        Recolour u_i with a uniformly random colour in L_{u_i}
        Let the new colour assignment be σ(u_i)
```

```
While there exists any bad event B(w) in v or the
distance-two neighbours of v:
    Write down "Fix w,σ(N_w)" to the log
    RECOLOUR(B(w))

Write down "Return" to the log
```

We call it a *step* every time we call `RECOLOUR`.

Benchmark Every step, we need at least $d(v)\log\Delta$ (or $d(w)\log\Delta$ if we call from `RECOLOUR`) bits from the random number generator (since $|L_{u_i}| \geq \Delta$ for $1 \leq i \leq d(v)$).

Our Representation The key point is to represent $\sigma(N_w)$ efficiently.

1. To represent `Fix` $v, \sigma(N_v)$, we need $\log|V|$ bits to index the vertex v, and we need $d(v)\log 2\Delta$ bits to represent $\sigma(N_v)$ (we could find a more efficient representation but there is no need). There are at most $|V|$ such events in the initial colouring, therefore, the total amount of the bits used throughout the entire run of the algorithm is at most $\sum_{v\in V}(\log|V| + d(v)\log 2\Delta)$, which is independent of the number of steps t. This is the overhead the we need to pay off if we are able to save the bits at every step.
2. To represent `Fix` $w, \sigma(N_w)$, there are at most $d(v)(\Delta-1)+1$ new events caused by the recolouring procedure, so we can use an index $1 \leq i \leq d(v)(\Delta-1)+1$ to locate the neighbour w, which takes $\log(d(v)(\Delta-1)+1)$ bits. We also need to represent $\sigma(N_w)$ efficiently. This is the core representation where we want to save some bits as it grows with the number of steps t. We could use $\log 2\Delta$ bits to represent the colour that is repeated more than ℓ times and the colour must appear on some neighbour u_i of v. We could use $\log\binom{\Delta}{\ell-1}$ bits to fix the pattern of neighbours with the same colour, and in the end we could use at most $\log(2\Delta)^{d(w)-\ell} = (d(w)-\ell)\log 2\Delta$ bits to represent the remaining colours. So in total, we will use $\log 2\Delta + \log\binom{\Delta}{\ell-1} + (d(w)-\ell)\log 2\Delta$ to represent $\sigma(N_w)$.
3. We need $O(1)$ number of bits to represent each `Return`.

3.1 Analysis

We win if the number of bits we use to represent `Fix` $w, \sigma(N_w)$ is fewer than $d(w)\log\Delta$.

Length of our representation:

$$\log(d(v)(\Delta-1)+1) + \log 2\Delta + \log\binom{\Delta}{\ell-1} + (d(w)-\ell)\log 2\Delta + O(1)$$

$$\begin{aligned}&\log(d(v)(\Delta-1)+1)+\log\Delta+\log\binom{\Delta}{\ell-1}+(d(w)-\ell)\log 2\Delta+O(1)\\ \leq\ &\log d(v)+2\log\Delta+(\ell-1)\log\frac{e\Delta}{\ell-1}-\ell\log 2\Delta+d(w)+d(w)\log\Delta+O(1)\end{aligned} \tag{3}$$

To make this smaller than $d(w)\log\Delta$, we need

$$\log d(v)+2\log\Delta+(\ell-1)\log\frac{e\Delta}{\ell-1}-\ell\log 2\Delta+d(w)+O(1)<0$$

$$\begin{aligned}&\log d(v)+2\log\Delta+(\ell-1)\log\frac{e\Delta}{\ell-1}-\ell\log 2\Delta+d(w)+O(1)\\ \leq\ &(3-\ell)\log\Delta-\ell+(\ell-1)(\log e+\log\Delta-\log(\ell-1))+O(1)+d(w)\\ \leq\ &2\log\Delta-\ell\log(\ell-1)+O(\ell)+d(w)\end{aligned} \tag{4}$$

If we put $d(w)$ apart, it comes down to whether $\ell\log(\ell-1)$ is greater than $2\log\Delta$. Based on a similar calculation, if we take $\ell=(2+\epsilon)\frac{\log\Delta}{\log\log\Delta}$, then $2\log\Delta-\ell\log(\ell-1)+O(\ell)<0$. However, since we have $d(w)$ and $d(w)$ can be as big as Δ, we will not get a compression.

In terms of the graph, recolouring the whole neighborhood would potentially cause many more bad events than just recolouring the vertices violating the property. Doing this will dilute the chance we move to a flawless colouring.

Another takeaway is that the recolouring algorithm we choose will affect the number of bits we can compress. So picking a good recolouring algorithm is as important as coming up with a creative way to represent the history of the algorithm.

4 Theorem by Achlioptas and Iliopoulos

In 2014, Achlioptas and Iliopoulos [3] came up with a new framework to represent entropy compression based on random walk on directed graphs. Instead of coming up with ad-hoc representations of the algorithm's history, they give a theorem that can be directly applied once we set up the problem using their framework.

4.1 Terminology

Ω is a set of objects. $\sigma\in\Omega$ is a single object.
F is a collection of subsets of Ω. Each subset $f\in F$ ($f\subseteq\Omega$) is called a *flaw*.
f is present in σ if $f\ni\sigma$. An object $\sigma\in\Omega$ is *flawless* if no flaw is present in σ.

For example, given a CNF formula. Each object σ is an assignment of boolean variables $x_1,...,x_n$, so $|\Omega|=2^n$. Let $c_1,...,c_m$ be the clauses. For each clause c_i, we can define a flaw $f_i=\{\sigma_{i_1},...,\sigma_{i_k}\}$ consisting of assignments that make c_i false. For example, $n=5$, and $c_i=(x_1\vee x_2\vee x_3)$, then

$f_i = \{(F,F,F,F,F),(F,F,F,F,T),(F,F,F,T,F),(F,F,F,T,T)\}$. A satisfying assignment for the entire CNF formula is equivalent to a flawless object σ.

D is a directed graph. We call elements $\sigma \in \Omega$ as states.
A state transformation $\sigma \to \tau$ ($\tau \in \Omega$) is taken to address a flaw $f \ni \sigma$.
For each $\sigma \in \Omega$, let $U(\sigma) := \{f \in F : \sigma \in f\}$, i.e., $U(\sigma)$ is the set of flaws present in σ.
For each $\sigma \in \Omega$ and $f \in U(\sigma)$, we require a set $A(f,\sigma) \in \Omega$ that must contain at least one element other than σ which we refer to as the set of possible actions of addressing flaw f in state σ. To address flaw f in state σ we uniformly random select an element $\tau \in A(f,\sigma)$ and walk to state τ. Note that it is possible we select $\tau = \sigma$. We require $A(f,\sigma)$ contains at least one element other than σ.

We represent the set of all possible state transformations as a multi-digraph D on Ω. For each state σ, for each flaw $f \in U(\sigma)$, for each state $\tau \in A(f,\sigma)$, we place an arc $\sigma \xrightarrow{f} \tau$.

The graph D we construct needs to have the following property:

Definition 1 (Atomicity). *D is atomic if for every flaw f and state τ there is at most one arc incoming to τ labelled by f.*

If D is atomic, then the random walk on D can be reconstructed from its final state and the sequence of labels on the arcs traversed. We will have atomicity if the following two conditions are satisfied:
1. Each constraint/flaw forbids exactly one joint value assignment to its underlying variables.
2. Each state transition modified only the variables of the violated constraint/flaw that it addresses.

For example, given a 3CNF formula with 5 variables, suppose $c_i = (x_1 \vee x_2 \vee x_3)$, then the flaw $f_i = \{(F,F,F,F,F),(F,F,F,F,T),(F,F,F,T,F),(F,F,F,T,T)\}$. It forbids exactly one joint value assignment $x_1 = F, x_2 = F, x_3 = F$. Suppose $\sigma = (F,F,F,F,F)$, in order to satisfy condition 2, the state transitions can only move to states where $x_4 = F, x_5 = F$, i.e., it can only change the truth value of the first three variables.

Definition 2 (Potential Causality). *For each arc $\sigma \xrightarrow{f} \tau$ in D and each flaw g present in τ we say that f causes g if $g = f$ or $\sigma \notin g$. If D contains any arc in which f causes g we say that f potentially causes g.*

Definition 3 (Potential Causality Digraph). *The digraph $C = C(\Omega, F, D)$ of the potential causality relation, i.e., the digraph on F where $f \to g$ iff F potentially causes g, is called the potential causality digraph. The neighborhood of a flaw f is $\Gamma(f) = \{g : f \to g$ exists in $C\}$.*

We can assume that C is strongly connected, implying $|\Gamma(f)| \geq 1$ for every $f \in F$.

We assign an arbitrary ordering π of F, and in each flawed state σ we address the greatest flaw according to π in a subset of $U(\sigma)$.

Definition 4 (Amenability). *The amenability of a flaw f is*

$$A_f := \min_{\sigma \in f} |A(f, \sigma)|$$

Theorem 2. *If for every flaw $f \in F$,*

$$\sum_{g \in \Gamma(f)} \frac{1}{A_g} < \frac{1}{e}$$

then for any ordering π of F and any $\sigma_1 \in \Omega$, the uniform random walk on D_π starting at σ_1 reaches a sink within $(\log_2 |\Omega| + |U(\sigma_1)| + s)/\delta$ steps with probability at least $1 - 2^{-s}$, where $\delta = 1 - \max_{f \in F} \sum_{g \in \Gamma(f)} \frac{e}{A_g}$.

4.2 Adapt this to the theorem

Let Ω be the set of all colourings. Let σ be a colouring of the graph. Let f be a flaw represented by $B(v, j, p)$ where v is the index of the vertex in the graph, j is the colour ($1 \leq j \leq 2\Delta$), and p is the pattern of ℓ neighbours of v having the same colour j. Let F be the set of all flaws.

Here is our order π of flaws. We first order by v (index of the vertex), then by j, then by p.

For any flaw f (or g), we would like to compute A_f (or A_g). Since we need our graph D to be atomic, we can only recolour the ℓ neighbours that have the same colour (the second condition). Suppose each neighbour u_i has L_{u_i} colours, note that since we have 2Δ colours and every vertex can have at most Δ neighbours, so $|L_{u_i}| \geq \Delta$, thus we would have

$$\forall f \in F,\ A_f \geq \Delta^\ell$$

For any flaw f, we would like to know $\Gamma(f)$. Note that our graph D needs to be atomic, so we can only recolour the vertices that have the same colour j, so we will recolour ℓ neighbours of v, and each neighbour will have at most $\Delta - 1$ neighbours other than v. So

$$\Gamma(f) = ((\Delta - 1)\ell + 1) \cdot 2\Delta \cdot \binom{\Delta}{\ell - 1}$$

Therefore

$$\sum_{g \in \Gamma(f)} \frac{1}{A_g} = \frac{((\Delta - 1)\ell + 1) \cdot 2\Delta \cdot \binom{\Delta}{\ell-1}}{A_f} \leq \frac{((\Delta - 1)\ell + 1) \cdot 2\Delta \cdot \binom{\Delta}{\ell-1}}{\Delta^\ell}$$

In order to have

$$\frac{((\Delta - 1)\ell + 1) \cdot 2\Delta \cdot \binom{\Delta}{\ell-1}}{\Delta^\ell} < \frac{1}{e}$$

We can take $\ell = (1 + \epsilon)\frac{\ln \Delta}{\ln \ln \Delta}$.

References

1. Michael Molloy. Entropy Compression and the Lovász Local Lemma. YouTube, uploaded by Combinatorics & Optimization University of Waterloo, 24 Aug 2017 https://www.youtube.com/watch?v=GVb0AT0cOSw
2. Michael Molloy and Bruce Reed. 2009. Asymptotically optimal frugal colouring. In Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms (SODA '09). Society for Industrial and Applied Mathematics, USA, 106–114.
3. Dimitris Achlioptas and Fotis Iliopoulos. Random Walks That Find Perfect Objects and the Lovasz Local Lemma. 2014 IEEE 55th Annual Symposium on Foundations of Computer Science, Philadelphia, PA, USA, 2014, pp. 494-503, doi: 10.1109/FOCS.2014.59.