Notes to the LLL Algorithm of Moser
(based on the notes of Robin Moser)

Tibor Szabó
Winter 2015 — Algorithmic Combinatorics

Let $\mathcal{F} = \{A_1, \ldots, A_m\}$ be an arbitrary hypergraph on the vertex set V. For a hyperedge $C \in \mathcal{F}$ denote by $\Gamma^+_{\mathcal{F}}(C) = \{D \in \mathcal{F} : C \cap D \neq \emptyset\}$ the neighborhood of C. Note that $C \in \Gamma^+_{\mathcal{F}}(C)$. Consider the following algorithm

```
Algorithm  $\mathcal{F}$-LLL-painter($\mathcal{G}, c$)
Input sub-hypergraph $\mathcal{G} \subseteq \mathcal{F}$, 2-coloring $c : V \to \{B, R\}$
0 Set $c' := c$
1 while $\mathcal{G}$ is not properly colored by $c'$ do
2         select a monochromatic set $A_i \in \mathcal{G}$ with smallest index
3         change $c'$ by recoloring the vertices of $A_i$ independently and u.a.r.
4         $\mathcal{F}$-LLL-painter($\Gamma^+_{\mathcal{F}}(A_i), c'$)
5 return $c'$
```

Lemma 1 *If $\mathcal{F}$-*`LLL-painter`*$(\mathcal{G}, c)$ ever returns, then $\mathcal{G}$ is properly colored, furthermore, every set $D \in \mathcal{F}$ that intersects a set $C \in \mathcal{F}$ recolored by the algorithm will itself be properly colored.*

Proof: Let $D \in \mathcal{F}$ be such a hyperedge. Let $C \in \mathcal{F}$ be the hyperedge which was recolored *last* by the algorithm among those with $C \cap D \neq \emptyset$. Then, since D is a member of $\Gamma^+_{\mathcal{F}}(C)$, $\mathcal{F}$-`LLL-painter`$(\Gamma^+_{\mathcal{F}}(C), c')$ could not return without properly coloring D. (In fact, immediately after the recoloring of C, D must be properly colored.) After that the vertices of D are not recolored anymore, so D stays properly colored. $\square$

Corollary 2 *Each hyperedge of $\mathcal{G}$ is recolored at most once on the top-level of $\mathcal{F}$-*`LLL-painter`*$(\mathcal{G}, c)$.*

The following infinite rooted ordered tree $\mathcal{T}$ will be a useful environment in picturing the progress of the algorithm. (By *ordered* tree we mean that the children of each vertex have an ordering on them.) The root r of $\mathcal{T}$ has $|\mathcal{G}|$ children each is *labeled* by a different hyperedge of $\mathcal{G}$. Then recursively, each further vertex, whose label is $C \in \mathcal{F}$, has children to $|\Gamma^+_{\mathcal{F}}(C)|$ vertices labeled by the hyperedges in $\Gamma^+_{\mathcal{F}}(C)$. (Note that the same hyperedge C will label infinitely many vertices of the tree. The root has no label.)

Because of the corollary, *any* particular running of the algorithm can be imagined as a subtree of the ordered tree $\mathcal{T}$ containing the root r.

Theorem 3 *Let $\mathcal{F} = \{A_1, \ldots, A_m\} \subseteq \binom{V}{k}$ be a k-uniform hypergraph such that $|\Gamma^+_{\mathcal{F}}(C)| \leq 2^{k-4}$ for every $C \in \mathcal{F}$. Let $c : V \to \{R, B\}$ chosen u.a.r. Then* $\mathcal{F}$-`LLL-painter`*$(\mathcal{F}, c)$ returns a proper coloring in $O(m \log m)$ expected time.*

Proof. We will encode what the algorithm does into a bitstring. To do that we traverse the edges and vertices of the corresponding ordered subtree of $\mathcal{T}$ starting at r. We traverse each edge twice, once downwards and once upwards. Each time we move down on an edge, we append a 1 to our bitstring. Each time we move up, we record a 0. Each time we *first* arrive at a vertex, we append the index of its label in binary and use a further bit to express the monochromatic color `color`(C) of the label C, say write 0 for blue and 1 for red. (Remember, each non-root vertex is labeled by a hyperedge and at the moment when we first arrive to a vertex its label is monochromatic according to c').

For a top-level vertex we need to use $\lceil \log m \rceil$ bits to encode its label, but on lower levels at most $k-4$ bits are enough, as we know that the particular hyperedge is a member of some $\Gamma^+_{\mathcal{F}}(C)$ (and we know which one!).

To do this encoding formally, for some $C \in \mathcal{G}$ let us define `bincode`$(C, \mathcal{G})$ be the number (written in binary) of hyperedges in $\mathcal{G}$ whose index is *smaller* than the index of C. Then here is the algorithm with the formal encoding written into a `log`[1]:

```
    Algorithm  $\mathcal{F}$-LLL-painter$(\mathcal{G}, c)$
Input sub-hypergraph $\mathcal{G} \subseteq \mathcal{F}$, 2-coloring $c : V \to \{B, R\}$
0 Set $c' := c$
1 while $\mathcal{G}$ is not properly colored by $c'$ do
2         select a monochromatic set $A_i \in \mathcal{G}$ with smallest index
3         change $c'$ by recoloring the vertices of $A_i$ independently and u.a.r.
4         append to the log: a bit 1, the bincode$(C, \mathcal{G})$ and the bit color$(C)$
5         run $\mathcal{F}$-LLL-painter$(\Gamma^+_{\mathcal{F}}(A_i), c')$ and set $c'$ to be its output
6         append a bit 0 to the log
7 return $c'$
```

Let us now count the bits. By the Corollary, the number of top-level recolorings is at most m, each requiring $\lceil \log m \rceil$ bits for the label of the set. Each of the lower level recolorings require $k-4$ bits for the label of the set. For each recoloring we must also add one bit expressing the color of the set before the recoloring

[1]There are two completely different logs in this note not to be confused with each other ...

and account for the bit "1" we write when we first enter the corresponding vertex of $\mathcal{T}$ and the bit "0" we write when we leave the vertex towards its parent. So in case of t lower level recolorings, we noted at most $l(t) = m(\lceil \log m \rceil + 3) + t(k-1)$ bits.

Note that the algorithm requires a source of random bits for the coloring: $|V|$ bits for the initial random coloring, and then an additional k bits for every subsequent recoloring. Now comes the intuitive contradiction if the algorithm was running too long: Each vertex of $\mathcal{T}$ which is traversed by the algorithm at some time corresponds to a hyperedge that is monochromatic at the moment the algorithm gets there the first time. This enables us to recover those k bits of the random source which the algorithm uses for the vertices of that hyperedge at that particular time. Note that these k vertices are immediately recolored by the algorithm using the random source, so when further vertices of $\mathcal{T}$ are visited by the algorithm, the k bits recovered there will each time be new. These bits might be not consecutive within the infinite random bitstring, but their location is exactly determined by what we have noted in our log so far.

In case of a top-level recoloring this is not so exciting: we used $\lceil \log m \rceil + 3$ bits in the log, likely much more than the recovered k random bits. For lower level recolorings, however, we use only $k-1$ bits to recover k fully random bits. In other words we *gain* 1 bit of information each time this happens. This cannot go on forever, otherwise information theory collapses! This is the reason the algorithm MUST finish in a finite time with high probability.

If there are t lower level recolorings we recover at least tk random bits. We must then have that $m(\lceil \log m \rceil + 3) + t(k-1) \geq tk$, i.e., $m(\lceil \log m \rceil + 3) \geq t$, is pretty likely. This implies that the number of lower level recolorings is bounded by a function of m.

Let us now see formally all the above. Let T be the random variable counting the number of lower level recolorings of the algorithm. We will prove that $Pr(T = \infty) = 0$ and $E(T | T < \infty) = O(m \log m)$.

Let $\alpha \in \{R, B\}^\infty$ be an infinite bitstring (from the random source) which made the algorithm use at least t lower level recolorings. Let $s = \texttt{log}(\alpha)$ be the the bitstring we produce in our log up to the tth lower level recoloring. Then s consists of at most $l(t)$ bits. Based on s we can reproduce tk very specific(!) places in α. In fact any infinite random bitstring α' which makes us write the same initial segment s into our log **must** have the same bits at these specific kt coordinates. Hence $Pr(s \text{ is the initial segment of our log}) \leq 1/2^{kt}$ for any $s \in S_t$, where S_t is the set of bitstrings that arise as initial segments of the log up to the tth lower level recoloring for some infinite bitstring. Clearly $S_t \subseteq \cup_{i=1}^{l(t)} \{0,1\}^i$. Hence

$$\begin{aligned} Pr(T \geq t) &= \sum_{s \in S_t} Pr(s \text{ is the initial segment of our log}) \leq \sum_{s \in S_t} \frac{1}{2^{kt}} \\ &\leq 2^{l(t)+1-kt} \leq 2^{m(\lceil \log m \rceil + 3)+1-t} \longrightarrow 0 \end{aligned}$$

as $t \to \infty$, so $Pr(T < \infty) = 1$. Then for the expected number of lower level recolorings we have

$$\begin{aligned} E(T|T < \infty) &= \sum_{t=1}^{\infty} Pr(T \geq t | T < \infty) \leq m(\lceil \log m \rceil + 3) + \sum_{t=m(\lceil \log m \rceil + 3)+1}^{\infty} 2^{m(\lceil \log m \rceil + 3)+1-t} \\ &= m(\lceil \log m \rceil + 3) + 2. \end{aligned}$$

Since the number of top level recolorings is at most m for every running of the algorithm and each recoloring requires a function of k many steps, we have shown that the running time is $O(m \log m)$.

$\square$

Remark 1. With a further observation one can also show that the number of recolorings is in fact $O(|V| \log m)$. (HW)
2. In line 2 of the algorithm we could have selected an *arbitrary* monochromatic hyperedge and practically the same analysis would work.
3. The bit 0 which we write in our log after the algorithm is back to the root vertex r is superfluous: we **know** that the next edge is downwards.