Streaming algorithms for 2-coloring uniform hypergraphs

Jaikumar Radhakrishnan, Saswata Shannigrahi

Tata Institute of Fundamental Research, Mumbai, India.
{jaikumar,saswata}@tifr.res.in

Abstract. We consider the problem of two-coloring n-uniform hypergraphs. It is known that any such hypergraph with at most $\frac{1}{10}\sqrt{\frac{n}{\ln n}}2^n$ hyperedges can be two-colored [7]. In fact, there is an efficient (requiring polynomial time in the size of the input) randomized algorithm that produces such a coloring. As stated [7], this algorithm requires random access to the hyperedge set of the input hypergraph. In this paper, we show that a variant of this algorithm can be implemented in the streaming model (with just one pass over the input), using space $O(|V|B)$, where V is the vertex set of the hypergraph and each vertex is represented by B bits. (Note that the number of hyperedges in the hypergraph can be superpolynomial in $|V|$, and it is not feasible to store the entire hypergraph in memory.)

We also consider the question of the minimum number of hyperedges in non-two-colorable n-uniform hypergraphs. Erdös showed that there exist non-2-colorable n-uniform hypegraphs with $O(n^2 2^n)$ hyperedges and $\Theta(n^2)$ vertices. We show that the choice $\Theta(n^2)$ for the number of vertices in Erdös's construction is crucial: any hypergraph with at most $\frac{2n^2}{t}$ vertices and $2^n \exp(\frac{t}{8})$ hyperedges is 2-colorable. (We present a simple randomized streaming algorithm to construct the two-coloring.) Thus, for example, if the number of vertices is at most $n^{1.5}$, then any non-2-colorable hypergraph must have at least $2^n \exp(\frac{\sqrt{n}}{8}) \gg n^2 2^n$ hyperedges. We observe that the dependence of exponential dependence on t in our result is optimal up to constant factors.

Keywords: Property B, hypergraph coloring, streaming algorithm, randomized algorithm

1 Introduction

Two colorability of uniform hypergraphs, also called Property B, is a well-studied problem in hypergraph theory. A hypergraph is called two colorable if all its vertices can be colored by either red or blue colors such that each hyperedge contains vertices of either colors. Erdős [4] first showed by a simple probabilistic argument that any n-uniform hypergraph with fewer than 2^{n-1} hyperedges is 2-colorable. Beck [3] used an algorithm of recoloring the vertices and improved this result to show that any n-uniform hypergraph with at most $n^{1/3-o(1)}2^{n-1}$ hyperedges is 2-colorable. Radhakrishnan and Srinivasan [7] improved on Beck's recoloring algorithm and obtained the best known result on this problem. They

proved that any n-uniform hypergraph with at most $\frac{1}{10}\sqrt{\frac{n}{\ln n}}2^n$ hyperedges can be 2-colored. They also provided a randomized polynomial time algorithm that 2-colors any such hypergraph with high probability. An event happens with high probability if for given any constant $\delta > 0$, the probability of its happening is at least $1 - \delta$. In Section 2, we describe this algorithm which we refer to as *delayed recoloring* algorithm throughout the rest of this paper.

The delayed recoloring algorithm assumes that all vertices and hyperedges of the hypergraph can be stored in the random-access memory (RAM). A processor can access RAM much faster than an external memory, and therefore it is ideal if we can store the entire hypergraph in RAM. Unfortunately, the number of hyperedges can be much larger than what the RAM can afford to store. For example, the number of hyperedges can be as high as $\Omega(2^n)$ when the size of the vertex set is just $O(n)$. This gives rise to the following question. If the RAM has just about enough space to store the vertices, can we still obtain the same coloring that the delayed recoloring algorithm obtains for any n-uniform hypergraph having at most $\frac{1}{10}\sqrt{\frac{n}{\ln n}}2^n$ hyperedges?

Before answering this question, let us first decide how we are going to store a hypergraph in the external memory. Note that each hyperedge is a collection of n vertices. If each vertex is represented by a B-bit long string, each hyperedge can be stored as a sequence of nB bits. We store the hyperedges one after another in the external memory, and store a terminal symbol when there are no more hyperedges. The vertices can be explicitly stored before the hyperedges, but we prefer that they are extracted while reading the hyperedges. This way, a coloring algorithm can start working as soon as the first hyperedge is stored in the external memory.

The algorithms that deal with limited RAM space are known as streaming algorithms in the literature. This algorithmic paradigm have of late become important, especially in the context of large data sets. It is motivated by the fact that the RAM of a computer is limited, and the size of the data set is often much larger than the size of the RAM. The data set can arrive as a continuous data stream or be stored in an external memory, in which case it is sequentially accessible over one or a small number of passes. However, only a minor fraction of the data along with some local variables can be stored in the RAM at any instant.

The challenge for any streaming algorithm is to minimize the following three parameters: the number of passes over the data, maximum RAM requirement at any instant and maximum processing time for any data item. On receiving the complete data set, the algorithm should decide as fast as possible either to start another pass or output an (approximately) accurate answer with high probability. A number of important algorithms have been developed in streaming model of computation, e.g., estimating frequency moments [1] and heavy hitters [8] of a data stream. These algorithms find applications in the context of network routing, where large amount of data flows through any router but each of them have limited memory to process the data. In Section 3, we develop the following streaming algorithm for hypergraph coloring.

Result 1 *We provide a randomized one-pass streaming algorithm to 2-color with high probability any n-uniform hypergraph that has at most $\frac{1}{10}\sqrt{\frac{n}{\ln n}}2^n$ hyperedges. This algorithm requires $O(|V|B)$ RAM space at any instant and $O(nB|V|)$ processing time after reading each hyperedge. On reading the terminal symbol, the algorithm spends $O(nB|V|)$ time and then outputs the coloring if and only if it is valid.*

Note that if the number of edges is much smaller than 2^n, then a random 2-coloring will with high probability be a proper coloring. In that case, we have a trivial streaming algorithm by choosing the coloring first, and verifying that it does properly color all the edges in just one pass.

We next consider the question of the minimum number of hyperedges in a non-2-colorable n-uniform hypergraph. Erdős [5] showed that there is an n-uniform hypergraph with $\Theta(n^2 2^n)$ hyperedges that is not 2-colorable. This construction uses $\Theta(n^2)$ vertices. Erdős and Lovász [6] conjectured that any n-uniform hypergraph with fewer than $n2^n$ hyperedges may be 2-colorable. We study if constructing a counter-example to this conjecture is possible with $o(n^2)$ vertices. Our result in Section 4 shows that with significantly fewer than n^2 vertices, we cannot construct a non-two-colorable hypergraph with fewer than $n2^n$ hyperedges.

Result 2 *For any n-uniform hypergraph with $|V| \leq \frac{n^2}{4 \ln 2n}$ and $|E| < n2^n$, we provide a randomized one-pass streaming algorithm that outputs a 2-coloring with high probability. This algorithm requires $O(\frac{n^2}{\ln n})$ RAM space at any instant and $O(n)$ processing time after reading each hyperedge. On reading the terminal symbol, the algorithm spends $O(1)$ time and then outputs a 2-coloring if and only if it is valid.*

Let us elaborate on the last sentences in each of the results above. When the algorithms terminate, we expect them to produce correct outputs with probability at least $1 - \delta$ for any given constant $\delta > 0$. Moreover, we insist that the algorithms produce one-sided errors, i.e., they either produce correct colorings or do not output any colorings. The motivation for insisting this is the following. Both of our algorithms execute $O(\log \frac{1}{\delta})$ processes in parallel, where each process has a small but constant probability of success. The probability that at least one of these processes is successful then becomes at least $1 - \delta$. To find out the one that is successful, we must examine the correctness of the output of a process after its termination. In each algorithm, we output a coloring if and only if we find a successful process.

Result 1 is obtained by modifying the delayed recoloring algorithm to a streaming version, and then showing that this modification results in the same coloring as before. Result 2 is obtained by a simple application of the union bound in probabilistic method.

1.1 Open questions

We point out two interesting questions that should be settled using existing techniques.

– Can we show that no *deterministic* algorithm can match the performance of our randomized algorithms? In particular, can we show that there are two-colorable hypergraphs with (say) n^2 vertices and (say) 2^n edges such that every deterministic one-pass streaming algorithm that is guaranteed to two-color them requires superpolynomial (in n) RAM space?
– Our result on hypergraphs with $O(\frac{n^2}{t})$ vertices does not improve on the bound provided via the delayed recoloring algorithm when it t is small, say, $O(\log n)$. We believe, it should be possible to combine our argument and the delayed recoloring algorithm to show that if the number of vertices is $o(n^2)$ then we can two-color hypegraphs with strictly more than $\frac{1}{10}\sqrt{\frac{n}{\ln n}}2^n$ hyperedges. We do not have such a result.

2 The Delayed Recoloring Algorithm

We will present the delayed recoloring coloring algorithm in this section. The analysis of the algorithm can be found in Section 2 of [7].

We start by recalling our assumption that the vertices of the hypergraphs are not explicitly stored in the external memory. With this assumption, the delayed recoloring algorithm proceeds as follows. The I/O processor first reads and stores all the hyperedges in RAM, and then extract the vertices from them. Next, one of the $|V|!$ permutations of the vertices is selected uniformly at random. The algorithm then starts with a initial random coloring $\chi_0(V)$ of the vertices, and attempts to flip the color of each vertex once in the order defined by the permutation. A randomly generated boolean vector $b(V)$ is used to indicate whether the color of a vertex is allowed to be flipped ($b(v) = 1$) or not ($b(v) = 0$). If allowed, the color of a vertex is flipped if and only if it belongs to an initially monochromatic hyperedge that continues to be monochromatic till this vertex in considered. Let us now describe the algorithm in detail.

I/O (read). Read all the hyperedges and store them in RAM.

Step 1. Extract the vertices from these hyperedges.

Step 2. Select one of the $|V|!$ permutations of the vertices uniformly at random. Denote this permutation by $P(V) = [v_1, v_2, \ldots v_{|V|}]$.

Step 3. Color each vertex $v \in V$ red or blue independently with probability $\frac{1}{2}$. Denote this coloring by $\chi_0(V)$.

Step 4. Set $b(v) = 1$ with probability p (to be specified below) and $b(v) = 0$ with probability $1 - p$, independently for each vertex $v \in V$. Denote this vector by $b(V)$.

Step 5. Recolor the vertices in $|V|$ steps as follows.

> **Step** 5(1). If v_1 is contained in a hyperedge that is monochromatic in $\chi_0(V)$, then flip the color of v_1 if and only if $b(v_1) = 1$. Denote this coloring by $\chi_1(V)$.
>
> ...

Step $5(i)$. If v_i is contained in a hyperedge that is monochromatic in $\chi_0(V)$ and continues to be monochromatic in $\chi_{i-1}(V)$, then flip the color of v_i if and only if $b(v_i) = 1$. Denote this coloring by $\chi_i(V)$.

I/O (write). Write the coloring $\chi_{|V|}(V)$ in external memory if and only if it is a valid 2-coloring of the hypergraph.

Let $\chi(V) = \chi_{|V|}(V)$. It has been shown in [7] that if $|E| \leq \frac{1}{10}\sqrt{\frac{n}{\ln n}}2^n$, then
1. Pr[there exists a monochromatic hyperedge in $\chi_0(V)$ that remains monochromatic in $\chi(V)$] $\leq \frac{2}{10}\sqrt{\frac{n}{\ln n}}(1-p)^n$.
2. Pr[there exists a non-monochromatic hyperedge in $\chi_0(V)$ which became blue in $\chi(V)$] $\leq$ Pr[there exists a non-monochromatic hyperedge in $\chi_0(V)$ whose red vertices became blue in $\chi(V)$] $\leq \frac{2np}{100 \ln n}$.
3. Pr[there exists a non-monochromatic hyperedge in $\chi_0(V)$ which became red in $\chi(V)$] $\leq$ Pr[there exists a non-monochromatic hyperedge in $\chi_0(V)$ whose blue vertices became red in $\chi(V)$] $\leq \frac{2np}{100 \ln n}$.

Note that the above three events are the only bad events which can make the 2-coloring $\chi(V)$ of a hypergraph invalid. The probability that any of these bad events happens is at most $\frac{2}{10}\sqrt{\frac{n}{\ln n}}(1-p)^n + \frac{4np}{100 \ln n}$. If $p = \frac{\ln n}{2n}$, this probability is at most $\frac{2}{10}\sqrt{\frac{1}{\ln n}} + \frac{2}{100}$. For any $n \geq 2$, this quantity is less than $(\frac{2}{10}\sqrt{\frac{1}{\ln 2}} + \frac{2}{100}) < \frac{1}{2}$. This means that the above algorithm produces a valid 2-coloring with probability at least $\frac{1}{2}$.

To improve this success probability to at least $1-\delta$ for any given constant $\delta > 0$, we need to execute steps $2-5$ of the algorithm $O(\log \frac{1}{\delta})$ times in parallel. It can be seen that the probability that at least one of these parallel processes succeeds in producing a valid coloring is at least $1-\delta$. It takes $O(n|E|B|V|)$ time to check whether a coloring is valid. We choose any one of the successful processes (if one exists) and write the coloring produced by it in the external memory.

Let us verify that the delayed recoloring algorithm can be executed in $O(nB|V||E|)$ time. Step 1 requires $O(nB|V||E|)$ time to identify all distinct vertices and store them in a sequence Q. Step 2 can be implemented in $O(B|V|^2)$ time as follows.

Initialize $P(V)$ to the permutation in which the vertices appear in Q. For each $1 \leq i \leq |V|$, do the following three operations. First, generate a random number j from $[1, i]$. Then, place the i-th vertex of Q at j-th position of permutation $P(V)$. Vertices that used to appear on or after j-th position in $P(V)$ are now shifted one place each to produce a modified $P(V)$.

Steps $3-4$ can be implemented by creating bit-vectors for the set of vertices, and therefore require $O(|V|)$ time each. The k-th position in these bit-vectors corresponds to the k-th position in the permutation $P(V)$. Finally, step $5(i)$ requires $O(nB|E|)$ time for any i, which implies that the total running time for step 5 is $O(nB|V||E|)$.

Note that the I/O(read) step is the reason we require high RAM space requirement in this implementation of the delayed recoloring algorithm. However,

this algorithm can be implemented by just storing the vertices and the hyperedges that are monochromatic after the first random coloring, and therefore requires $O(|V|B + nB\sqrt{\frac{n}{\ln n}})$ RAM space at any instant. This implementation, however, does not verify that the coloring is proper, and might output invalid colorings. The algorithm in the next section is guaranteed never to do this.

3 An Efficient Streaming Algorithm

In this section, we present a modified algorithm that uses $O(|V|B)$ RAM space to store the vertices at any instant, but the total processing time remains asymptotically the same.

Assume that the hyperedges are stored in the external memory in the order $h_1, h_2, \ldots, h_{|E|}$. The I/O processor reads the hyperedges from left to right and stores only the current hyperedge in RAM. New vertices, if any, are extracted form a hyperedge once it is read and stored in the RAM, along with previously extracted vertices. The set of vertices after hyperedge h_i is read is denoted by V_i. Once hyperedge h_i is read, the algorithm first assigns initial colors χ_0 to vertices belonging to $V_i \setminus V_{i-1}$, and then attempts to recolor the vertices belonging to h_i in the order defined by a uniformly random permutation $P(V_i)$ of the vertices extracted so far. As before, a randomly generated boolean vector $b(V_i)$ is used to indicate whether the color of a vertex is allowed to be flipped ($b(u) = 1$) or not ($b(u) = 0$). A vertex is recolored at most once. Let us now describe the modified algorithm in detail.

Initialize $V_0 = \Phi$. For $1 \leq i \leq |E|$, do each of the following steps.

I/O (read). Read the hyperedge h_i and store it in RAM. Delete h_{i-1} from RAM.

Step $1(i)$. Extract and add new vertices of h_i to V_{i-1}, the set of vertices currently stored in RAM. Denote this modified set of vertices by V_i. Note that $V_{|E|} = V$.

Step $2(i)$. Sequentially insert each vertex of $V_i \setminus V_{i-1}$ uniformly at random into the existing random permutation $P(V_{i-1})$ of the vertices belonging to V_{i-1}. If $|V_i| = j$, denote this permutation by $P(V_i) = [v_1, v_2, \ldots v_j]$.

Step $3(i)$. For each vertex $u \in V_i \setminus V_{i-1}$, set $\chi_0(u)$ to red or blue independently with probability $\frac{1}{2}$. Keep $\chi_0(u)$ unchanged for each vertex $u \in V_{i-1}$.

Step $4(i)$. Set $b(u) = 1$ with probability p' (to be chosen later) and $b(u) = 0$ with probability $1 - p'$, independently for each vertex $u \in V_i \setminus V_{i-1}$. Keep $b(u)$ unchanged for each vertex $u \in V_{i-1}$.

Step $5(i)$. If h_i is monochromatic in $\chi_0(V_i)$, find the first (in permutation $P(V_i)$) vertex $u \in h_i$ with $b(u) = 1$. Flip the color of u if and only if it has not been flipped before. Keep the colors of the vertices belonging to $V_i \setminus \{u\}$ unchanged. Denote the new coloring of the vertices by $\chi_i^{'}(V_i)$.

I/O (write). Write the coloring $\chi^{'}_{|E|}(V)$ in external memory if and only if it is a valid 2-coloring of the hypergraph.

Let $\chi'(V) = \chi^{'}_{|E|}(V)$. Recall that the original delayed recoloring algorithm produces a coloring $\chi(V)$ at its termination. We show below that $\chi'(V)$ is the same as $\chi(V)$ for any given $\chi_0(V)$, $P(V)$ and $b(V)$.

Lemma 1. *For each $v \in V$, $\chi'(v) = \chi(v)$ for any given χ_0, P and b.*

Proof. First, let us assume to the contrary that there exists a vertex u with $\chi_0(u) = \chi'(u) \neq \chi(u)$. Let us also assume that h is a hyperedge that necessitated u's recoloring in the original algorithm. In other words, h was one of the monochromatic hyperedges in $\chi_0(V)$ that remained monochromatic till u was considered in one of the sub-steps of step 5. This must have happened due to the fact that the vertices of h that appeared before u in $P(V)$ had their corresponding b bits set to 0. Therefore, when h is considered in the modified algorithm, u must be its first vertex (in permutation $P(V)$) with $b(u) = 1$. The color of u is flipped in this step of the algorithm, if not already flipped in a previous step. Thus, $\chi(u) \neq \chi^{'}(u)$. This contradicts the assumption above.

On the other hand, let us assume to the contrary that there exists a vertex u' with $\chi_0(u') = \chi(u') \neq \chi'(u')$. Let h' be the hyperedge that necessitated the recoloring of u' in the current algorithm. This means that all vertices (of h') before u' (in permutation $P(V)$) had their b bits set at 0. This implies that u' must have been recolored in $\chi(V)$, because h' would have remained monochromatic till u'-th recoloring sub-step of step 5. Thus, $\chi(u') \neq \chi^{'}(u')$. Again, this contradicts the above assumption. □

This lemma implies upper bounds on the probabilities of the following three bad events:
1. Pr[there exists a monochromatic hyperedge in $\chi_0(V)$ that remains monochromatic in $\chi^{'}(V)$] $\leq \frac{2}{10}\sqrt{\frac{n}{\ln n}}(1-p')^n$.
2. Pr[there exists a non-monochromatic hyperedge in $\chi_0(V)$ whose red vertices became blue in $\chi^{'}(V)$] $\leq \frac{2np'}{100 \ln n}$.
3. Pr[there exists a non-monochromatic hyperedge in $\chi_0(V)$ whose blue vertices became red in $\chi^{'}(V)$] $\leq \frac{2np'}{100 \ln n}$.

Note that if none of the above events takes place, the algorithm produces a proper 2-coloring $\chi'(V)$. It can be easily seen that the RAM requirement at any instant is only $O(|V|B)$. Note, however, that the above analysis does not have the desirable property that it outputs only valid colorings. A straight-forward check does not seem possible using $O(|V|B)$ RAM space. We show below that with carefully storing some vertices of a few hyperedges, we can in fact achieve this using only $O(nB)$ RAM space.

Checking whether a coloring is proper using $O(nB)$ RAM space

In order to check whether the first event takes place for a hyperedge h, it is sufficient to look into the corresponding χ_0 and b bits as soon as the hyperedge

is read. The first event takes place for h if and only if it is monochromatic in χ_0 and all its b bits are set at 0. However, it is not immediately possible to determine after reading h whether the second (similarly, the third) event takes place because of it. Therefore, we mark h as a *potentially blue* hyperedge if all its red vertices have their corresponding b bits set at 1. In h is such a hyperedge, we store this subset R_h of red vertices in memory. At the end of $|E|$-th recoloring step, we can check whether all red vertices of h flipped their colors or not. Note that hyperedges those are not potentially blue cannot cause the second event. Similarly, we can mark a hyperedge as *potentially red* and store its subset B_h of blue vertices. For any given h, let us now calculate the expected number of vertices in R_h.

$$E[|R_h|] = \sum_{i=1}^{n} i \cdot \binom{n}{i} \cdot 2^{-n}(p')^i = np'2^{-n} \sum_{i=1}^{n} \binom{n-1}{i-1}(p')^{i-1} \leq np'2^{-n}e^{p'n}.$$

Therefore, the expected total number of vertices in R_h's of all potentially blue hyperedges is at most $\frac{1}{10} \cdot \sqrt{\frac{n}{\ln n}} \cdot np' \cdot e^{p'n}$. By Markov's inequality,

$$\Pr[\text{total number of vertices in } R_h\text{'s of all potentially blue hyperedges} \geq \tfrac{100}{10} \cdot \sqrt{\tfrac{n}{\ln n}} \cdot np' \cdot e^{p'n}] \leq \tfrac{1}{100}.$$

Similarly, we can bound by $\frac{1}{100}$ the probability of the bad event that the total number of vertices in B_h's of all potentially red hyperedges is more than $\frac{100}{10} \cdot \sqrt{\frac{n}{\ln n}} \cdot np' \cdot e^{p'n}$. As a result, the probability that any one of the bad events takes place is at most $\frac{2}{100} + \frac{2}{10}\sqrt{\frac{n}{\ln n}}(1-p')^n + \frac{4np'}{100 \ln n}$. If $p' = \frac{\ln n}{2n}$, the probability of success is still at least $\frac{1}{2}$ as in Section 2. However, the total number of vertices in R_h's of all potentially blue hyperedges is at most $10\sqrt{\frac{n}{\ln n}} \cdot \frac{\ln n}{2} \cdot \sqrt{n} = O(n\sqrt{\ln n})$. To get the claimed bound on RAM space, we need to choose p' more carefully. In particular, we slightly reduce the value of p'.

If $p' = \frac{\ln n}{2n} - \frac{\ln \ln n}{2n}$, the probability that one of the bad events happens is at most $\frac{2}{100} + \frac{2}{10} + \frac{2}{100}$. This means the the success probability is at least $\frac{76}{100}$. With this value of p', the total number of vertices in R_h's of all potentially blue hyperedges is at most $10\sqrt{\frac{n}{\ln n}} \cdot \frac{\ln n}{2} \cdot \sqrt{\frac{n}{\ln n}} = O(n)$. Each vertex takes B bits to store, and so total RAM space required to store all R_h's and R_b's is at most $O(nB)$. This implies that the total RAM space required for the algorithm is $O(|V|B + nB) = O(|V|B)$.

For any given constant $\delta > 0$, we can repeat steps $1(i)$ to $5(i)$ of the algorithm $O(\log \frac{1}{\delta})$ times in parallel to improve the probability of success to at least $1 - \delta$. The RAM requirement for this algorithm then becomes $O(|V|B \log \frac{1}{\delta})$.

It can be easily checked that the processing time for steps $1(i)$ to $5(i)$ is $O(nB|V|)$ for each $1 \leq i \leq |E|$. This implies a total of $O(nB|V||E|)$ processing time for the entire algorithm. On reading the terminal symbol, the algorithm requires to look at the colors of $O(n)$ vertices belonging to R_h's and R_b's to determine the validity of the coloring, and $O(nB|V|)$ processing time is required for this purpose. This completes the proof of Result 1.

4 Coloring Uniform Hypergraphs with $O(\frac{n^2}{\ln n})$ Vertices

In this section, we first show that any hypergraph with at most $\frac{n^2}{4 \ln 2n}$ vertices and fewer than $n2^n$ hyperedges can be two colored. Erdős [5] constructed a random n-uniform hypergraph with $\Theta(n^2)$ vertices and $n^2 2^n$ hyperedges that is not 2-colorable [2]). Erdős and Lovász [6] conjectured that any n-uniform hypergraph with fewer than $n2^n$ hyperedges may be 2-colorable. The following lemma proves that constructing any counterexample to the conjecture of Erdős and Lovász [6] requires more than $\frac{n^2}{4 \ln 2n}$ vertices.

Lemma 2. *Any n-uniform hypergraph with fewer than $n2^n$ hyperedges and at most $\frac{n^2}{4 \ln 2n}$ vertices can be 2-colored.*

Proof. Let us assume that the hypergraph has $2Cn$ vertices (C is a function of n). We partition the vertex set randomly into two parts and color each vertex in one of them by red and the other by blue. Let us calculate the probability that a hyperedge e is monochromatic in this coloring $\chi(V)$. Let p be the probability that e is monochromatic in $\chi(V)$.

$$p = \frac{\binom{2Cn-n}{Cn}}{\binom{2Cn}{Cn}} = \frac{Cn \cdot (Cn-1) \cdot (Cn-2) \cdots (Cn-n+1)}{2Cn \cdot (2Cn-1) \cdot (2Cn-2) \cdots (2Cn-n+1)}$$
$$\leq 2^{-n} \cdot (1 - \frac{1}{4C-1})^{\frac{n}{2}} \leq 2^{-n} \cdot e^{\frac{-n}{8C-2}}.$$

If $C \leq \frac{n}{8 \ln 2n}$, it follows that p is strictly less than $\frac{1}{2n2^n}$. Therefore, the probability that at least one edge is monochromatic in the coloring $\chi(V)$ is at most $|E| \cdot \frac{1}{2n2^n} < \frac{1}{2}$. This implies that there is a proper 2-coloring of the hypergraph. □

Note that the above proof suggests the following result. If $C = \frac{n}{t}$ for a parameter t, then there is a 2-coloring of a hypergraph with at most $2^n \exp(\frac{t}{8})$ hyperedges. This hypergraph has $2Cn = \frac{2n^2}{t}$ vertices. We show below by an argument similar to the one used by Erdős [5] that if $t = o(n)$, there exists a hypergraph with $\frac{2n^2}{t}$ vertices and $O(\frac{n^2}{t} \cdot 2^n \exp(\frac{t}{2}))$ hyperedges that is not 2-colorable.

Consider a 2-coloring of the vertex set, whose size is $\frac{2n^2}{t}$. Pick a random hyperedge of size n. The probability that this hyperedge is monochromatic is at least

$$p = \frac{\binom{\frac{n^2}{t}}{n}}{\binom{\frac{2n^2}{t}}{n}} \geq \left(\frac{\frac{n^2}{t} - n}{\frac{2n^2}{t} - n} \right)^n \approx 2^{-n} \left(1 - \frac{t}{2n} \right)^n \approx 2^{-n} \exp(\frac{-t}{2}).$$

Let $S_1, S_2, \ldots, S_r$ be uniformly and independently chosen hyperedges. The probability that none of these hyperedges is monochromatic is at most

$\left(1 - 2^{-n}\exp(\frac{-t}{2})\right)^r 2^{\frac{2n^2}{t}}$. If $\left(1 - 2^{-n}\exp(\frac{-t}{2})\right)^r 2^{\frac{2n^2}{t}} < 1$, there exists a hypergraph with r hyperedges that is not 2-colorable. Since $(1-x)^r \leq \exp(-xr)$, this inequality is satisfied when $r \geq \frac{2n^2}{t} \cdot 2^n \exp(\frac{t}{2}) \cdot \ln 2$.

Lemma 2 can be extended for k-coloring of n-uniform hypergraphs with vertex set size $O(\frac{n^2}{\ln n})$ but having fewer than nk^n hyperedges.

Lemma 3. *Any n-uniform hypergraph with fewer than nk^n hyperedges drawn from a set of at most $\frac{(k-1)n^2}{4\ln 2n}$ vertices can be k-colored.*

Proof. As before, let us assume that the hypergraph has kCn vertices. We partition the vertex set randomly into k parts and color them by k different colors. Let us calculate the probability that a hyperedge e is monochromatic in this coloring $\chi(V)$. Let p be the probability that e is monochromatic in $\chi(V)$.

$$p = \frac{\binom{kCn-n}{Cn-n} \cdot \binom{(k-1)\cdot Cn}{Cn} \cdots \binom{Cn}{Cn}}{\binom{kCn}{Cn} \cdot \binom{(k-1)\cdot Cn}{Cn} \cdots \binom{Cn}{Cn}} \leq k^{-n} \cdot (1 - \frac{k-1}{2kC-1})^{\frac{n}{2}} \leq k^{-n} \cdot e^{\frac{-n(k-1)}{4Ck-2}}.$$

If $C \leq \frac{(k-1)n}{4k\ln 2n}$, it follows that p is strictly less than $\frac{1}{2nk^n}$. Therefore, the probability that at least one hyperedge is monochromatic in the coloring χ is at most $|E| \cdot \frac{1}{2nk^n} < \frac{1}{2}$. This implies that there is a proper k-coloring of the hypergraph. □

The above two lemmas also show that there exists an *equitable* k-coloring of a hypergraph with fewer than nk^n hyperedges if it has $O(\frac{n^2}{\ln n})$ vertices. In fact, both these proofs can easily be transformed into randomized streaming algorithms to find such colorings. At the beginning, we store the colors of the vertices (half red and remaining half blue) in a bitvector of size $O(\frac{n^2}{\ln n})$. With the arrival of each hyperedge, we just need to check whether it is monochromatic by checking the corresponding bits. If there are fewer than nk^n hyperedges, the probability of success (no hyperedge is monochromatic) is at least $\frac{1}{2}$. By repeating this algorithm $\log \frac{1}{\delta}$ times in parallel, we can improve the probability of success to at least $1 - \delta$. The memory space requirement for this algorithm is $O(\frac{n^2}{\ln n} \log \frac{1}{\delta})$.

In the following, we derandomize this algorithm to k-color any hypergraph with at most $\frac{(k-1)n^2}{4\ln 2n}$ vertices and fewer than nk^n hyperedges. We derandomize using conditional expectations, in a way it is used to derandomize the algorithm to find a large cut in a graph [10]. We first provide the algorithm for $k = 2$. Let us assume that $|V|$ is even and the vertices are $v_1, v_2, \ldots, v_{|V|}$. The order of vertices in which they are colored will be denoted by $u_1, u_2, \ldots, u_{|V|}$.

Step 1: Color v_1 by red, and call this vertex u_1.

Step 2: For $j = 2$ to n, calculate the expected number of monochromatic hyperedges conditioned on coloring v_j by red. Select the vertex that gives the lowest expectation and color it by red. Call this vertex u_2.

Step $i \leq \frac{|V|}{2}$: After the $(i-1)$-th step, the colors of $u_1, u_2, \ldots, u_{i-1}$ are red. For $j = i$ to n, calculate the expected number of monochromatic hyperedges

conditioned on coloring v_i by red. Select the i-th red vertex as the one that gives the lowest expectation and call it u_i.

Step $\frac{|V|}{2} + 1$: Color all the remaining vertices by blue.

It can be easily seen that such an algorithm produces a deterministic 2-coloring in time $O(|V|^2|E|)$. The space requirement, however, becomes $O(|E|B)$.

A similar deterministic algorithm exists for k-coloring as well. We first find the vertices with color1, followed by color2, ..., colork.

5 Streaming and the Lovász Local Lemma

Radhakrishnan and Srinivasan [7] considered the local version of the problem of 2-coloring. In this section, we mention a streaming algorithm for this version. In particular, we observe from the parallel version of the algorithm of Moser and Tardos [11] that for any given constant $\epsilon > 0$, there exists an $O(\log |V|)$-pass streaming algorithm to 2-color any n-uniform hypergraph none of whose hyperedges intersects more than $\frac{(1-\epsilon)2^{n-1}}{e} - 1$ other hyperedges. This algorithm requires $O(|V|B)$ memory space. For such hypergraphs, it is an interesting open problem to find an $O(1)$-pass streaming algorithm that uses asymptotically just enough RAM space to store the vertices.

Moser and Tardos [11] recently proposed a parallel algorithm for Lovász Local Lemma, which we describe below. Let X be a finite set of events determined by a finite set P of mutually independent random variables, such that each event of X is determined by a subset of the variables in P. Let G_X denote the dependency graph of the events, i.e., two events A and B are connected by an edge if and only if they share common variables. For any event $A \in X$, we denote by $N(A)$ the events which are neighbors of A in G_X. An assignment or evaluation of the variables *violates* an event A if it makes A happen. With these notations, we explain their algorithm:

Step 1. Evaluate each variable in P independently at random.

Step 2. If there exists at least one violated event in X, construct a maximal independent set M of the sub-graph (of G_X) induced by the violated events in X. Independently perform random re-evaluation of each variable that belongs to one of the events of M.

Step 3. If there are no violated events, output the current evaluation of the variables. Otherwise, go to Step 2.

If a local condition is assumed to hold for each of the events, the following theorem bounds the expected number of times Step 2 of the algorithm is executed.

Theorem 1. *[11] If $\epsilon > 0$ and there exists real number assignments $x : X \to (0, 1)$ such that*

$$\forall A \in X : \Pr[A] \leq (1 - \epsilon)x(A) \prod_{B \in N(A)} (1 - x(B)), \tag{1}$$

then the algorithm executes step 2 *an expected* $O(\frac{1}{\epsilon}\log\sum_{A\in X}\frac{x(A)}{1-x(A)})$ *number of times before it finds an evaluation of* P *violating no event in* X*.*

For each hyperedge h of a hypergraph H, let X_h denote the event that h is monochromatic in a 2-coloring of H and let $X = \{X_h : h \in H\}$. Therefore, $\Pr[X_h] = 2^{1-n}$. If each hyperedge of H intersects at most $\frac{(1-\epsilon)2^{n-1}}{e} - 1$ other hyperedges, we can assign $x(H_h) = \frac{e}{(1-\epsilon)2^{n-1}}$ to satisfy Equation 1 for all $X_h \in X$. (We use $(1-\frac{1}{r+1})^r \geq e^{-1}$ for any $r \geq 1$.)

In our streaming algorithm, we start with a uniformly random coloring of the vertices. Thereafter, Step 2 of the algorithm can be executed once in each pass as follows. Whenever an hyperedge arrives, we mark and store its vertices in memory if and only if it is monochromatic in the current coloring and does not intersect with any of the previously marked hyperedges. Since at most $\frac{|V|}{n}$ disjoint hyperedges can be marked in one pass, only $O(|V|B)$ memory space is required to store all their vertices. At the end of each pass, we randomly re-evaluate the colors of each of the marked vertices and return to the beginning of the secondary memory. The algorithm stops if and only if there are no monochromatic hyperedges in the current coloring. By Theorem 1, this algorithm terminates after $O(\log\frac{|E|}{2^{n-1}}) = O(\log|V|)$ expected number of passes.

References

1. N. Alon, Y. Matias and M. Szegedy. The space complexity of approximating the frequency moments. *Proceedings of the ACM symposium on Theory of computing (STOC)*, 20-29, 1996.
2. N. Alon and J. H. Spencer. The Probabilistic Method. Wiley-Interscience Series, John Wiley and Sons, Inc., New York, 1992.
3. J. Beck. On 3-chromatic hypergraphs. *Discrete mathematics*, 24: 127-137, 1978.
4. P. Erdős. On a combinatorial problem. *Nordisk Mat. Tidsskr*, 11: 5-10, 1963.
5. P. Erdős. On a combinatorial problem, II. *Acta Mathematica of the Academy of Sciences*, Hungary, 15: 445-447, 1964.
6. P. Erdős and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. Colloq. Math. Soc. Janos Bolyai, Vol. 10, 609-627.
7. J. Radhakrishnan and A. Srinivasan. Improved bounds and algorithms for hypergraph 2-coloring. *Random Structures Algorithms* 16 (1): 4-32, 2000; also in FOCS 1998.
8. R. M. Karp, C. H. Papadimitriou and S. Shenker. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems* 28: 51-55, 2003.
9. A. Kostochka. Coloring uniform hypergraphs with few colors. *Random Structures Algorithms* 24 (1): 1-10, 2004.
10. M. Mitzenmacher and E. Upfal. Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press, 2005.
11. R. A. Moser and G. Tardos. A constructive proof of the general Lovász local lemma. *Journal of the ACM* 57 (2): 1-15, 2010.
12. R. Yuster. Equitable coloring of k-uniform hypergraphs. *SIAM Journal on Discrete Mathematics* 16 (4): 524-532, 2003.