# A Fast Coloring Oracle for Average Case Hypergraphs

**Cassandra Marcussen**
Harvard University, Cambridge, MA, USA

**Edward Pyne**
MIT, Cambridge, MA, USA

**Ronitt Rubinfeld**
MIT, Cambridge, MA, USA

**Asaf Shapira**
Tel Aviv University, Israel

**Shlomo Tauber**
Tel Aviv University, Israel

## Abstract

Hypergraph 2-colorability is one of the classical NP-hard problems. Person and Schacht [SODA'09] designed a deterministic algorithm whose expected running time is polynomial over a uniformly chosen 2-colorable 3-uniform hypergraph. Lee, Molla, and Nagle recently extended this to $k$-uniform hypergraphs for all $k \geq 3$. Both papers relied heavily on the regularity lemma, hence their analysis was involved and their running time hid tower-type constants.

Our first result in this paper is a new simple and elementary deterministic 2-coloring algorithm that reproves the theorems of Person–Schacht and Lee–Molla–Nagle while avoiding the use of the regularity lemma. We also show how to turn our new algorithm into a randomized one with average expected running time of only $O(n)$.

Our second and main result gives what we consider to be the ultimate evidence of just how easy it is to find a 2-coloring of an average 2-colorable hypergraph. We define a *coloring oracle* to be an algorithm which, given vertex $v$, assigns color red/blue to $v$ while inspecting as few edges as possible, so that the answers to any sequence of queries to the oracle are consistent with a single legal 2-coloring of the input. Surprisingly, we show that there is a coloring oracle that, on average, can answer **every** vertex query in time $O(1)$.

**2012 ACM Subject Classification** Theory of computation → Streaming, sublinear and near linear time algorithms

**Keywords and phrases** average-case algorithms, local computation algorithms, graph coloring

**Digital Object Identifier** 10.4230/LIPIcs.APPROX/RANDOM.2025.61

**Category** RANDOM

**Related Version** *Full Version*: `https://arxiv.org/abs/2507.10691`

**Funding** *Cassandra Marcussen*: Supported in part by an NDSEG fellowship, and by NSF Award 2152413 and a Simons Investigator Award to Madhu Sudan.
*Edward Pyne*: Supported by a Jane Street Graduate Research Fellowship and NSF awards CCF-2310818 and CCF-2127597.
*Ronitt Rubinfeld*: Supported by NSF awards DMS-2022448 and CCF-2310818.
*Asaf Shapira*: Supported in part by ERC Consolidator Grant 863438.
*Shlomo Tauber*: Supported in part by ERC Consolidator Grant 863438.

## 1 Introduction

Graph partition problems are among the most well-studied topics in algorithmic graph theory. These problems ask if a graph can be partitioned so that a certain *global* property holds. Among these properties, probably the most well-studied one is graph and hypergraph

colorability. Let us quickly recall the relevant definitions. A $k$-uniform hypergraph $\mathcal{H} = (V(\mathcal{H}), E(\mathcal{H}))$ consists of a vertex set $V(\mathcal{H})$ and edge set $E(\mathcal{H})$ where every edge $e \in E(\mathcal{H})$ is a subset of $V(\mathcal{H})$ of size $k$. For brevity, we call a $k$-uniform hypergraph a $k$-graph, noting that a 2-graph coincides with the usual notion of a graph. We denote $\left|V(\mathcal{H})\right| = n$. A $k$-graph $\mathcal{H}$ is $d$-colorable if $V(\mathcal{H})$ has a $d$-coloring so that in each edge at least two vertices are colored differently. When a $k$-graph is 2-colorable, we will also call it *bipartite*[1]. While graph $d$-colorability is NP-hard for $d \geq 3$, Lovász [29] famously proved that for $k \geq 3$ deciding if a $k$-graph is 2-colorable is also NP-hard. Hypergraph 2-colorability has been extensively studied in combinatorics [14, 30, 36], with notable contributions leading to the development of the renowned Lovász Local Lemma [30]. In computer science, hypergraph coloring has received significant attention due to its strong connections with fundamental problems in graph coloring and satisfiability of Boolean formulas [15, 31]. By leveraging approximation techniques from graph coloring, several works [4, 10, 23, 24] have proposed algorithms for properly coloring 2-colorable hypergraphs, where the number of colors depends on the size of the hypergraph.

Given the hardness of deciding graph and hypergraph colorability problems, it is natural to ask if these problems are easy on average. This question is a natural one in the study of average-case complexity of NP-hard problems. The first result in this direction was obtained by Turner [40] who proved that there is an algorithm that can find a $d$-coloring of almost all $d$-colorable graphs. Note that this does *not* give an algorithm whose average running time is polynomial over the set of $d$-colorable graphs. Such a result was obtained in a highly influential paper of Dyer and Frieze [17]. It is of course natural to design such algorithms for $k$-graphs. The first to so were Person and Schacht [34, 35] who gave an average case polynomial time algorithm for 3-graph 2-colorability. Their result was recently extended to arbitrary $k \geq 3$ by Lee, Molla, and Nagle [26] who gave an average case $O(n^k)$ algorithm. The algorithms of [26, 34, 35] rely on graph/hypergraph regularity lemmas. As a result, they were difficult to analyze, used algorithmic versions of the regularity lemma [3] as a black box, and their running times hid tower-type constants.

## 1.1 New Classical Algorithms

Our main results in this paper improve upon [26, 34, 35] in several ways. In what follows, a 2-coloring algorithm is one that is guaranteed to find a 2-coloring of every 2-colorable $k$-graph. Throughout the rest of the paper we assume that $k \geq 3$ is an absolute constant. Therefore, while in many places the dependence on $k$ can be improved significantly, we chose not to do so for the sake of simplifying the proofs.

Our first result in this paper is a new deterministic 2-coloring algorithm with polynomial average case running time. The algorithm is simple and completely elementary. In particular, it does not use any graph or hypergraph regularity lemma.

▶ **Theorem 1.** *There is a deterministic 2-coloring algorithm whose average case running time over the set of 2-colorable $k$-graphs is $n^{O(k)}$.*

To be precise, and to set the stage for the introduction of the coloring oracle of Theorem 4 below, we state the above theorem as follows. If $A$ is a deterministic 2-coloring algorithm, then $T_A(\mathcal{H})$ denotes the running time of $A$ on input $\mathcal{H}$, and $T_A(n)$ denotes the average of $T_A(\mathcal{H})$ over all 2-colorable $k$-graphs on $n$ vertices. Theorem 1 thus states that there is a 2-coloring algorithm $A$ satisfying $T_A(n) = n^{O(k)}$.

[1] In extremal combinatorics, 2-colorability is also called *Property B*.

Our second result shows that if we allow the 2-coloring algorithm to use randomization, then we can significantly improve the average case expected running time.

▶ **Theorem 2.** *There is a randomized 2-coloring algorithm whose average case expected running time over the set of 2-colorable $k$-graphs is $O(n)$.*

If $A$ is a randomized 2-coloring algorithm, then $T_A(\mathcal{H})$ now denotes the *expected* running time of $A$ on input $\mathcal{H}$, and $T_A(n)$ denotes the average of $T_A(\mathcal{H})$ over all 2-colorable $k$-graphs on $n$ vertices. Theorem 2 thus states that there is a randomized 2-coloring algorithm $A$ satisfying $T_A(n) = O(n)$. This average case running time is optimal since we need $O(n)$ time just to output the coloring of $\mathcal{H}$.

## 1.2 A Fast Coloring Oracle

We now turn to introducing our main result in this paper, which shows that hypergraph 2-colorability is even easier on average than merely being solvable in polynomial time as in Theorems 1 and 2. (In fact, the algorithms and techniques for this result directly imply Theorems 1 and 2 as well.) To this end we will introduce a *local* coloring algorithm in Definition 3 below. Our main inspiration for this definition are the notion of a *Partition Oracles* from the area of sublinear time algorithms and the emerging area of *Local Computing Algorithms (LCA)*. Let us now describe Partition Oracles, postponing to Subsection 1.2.1 the discussion of LCAs and prior relevant work. The notion of partition oracle was first introduced in [22] and was further studied in [27, 25]. These are sublinear, in fact $O(1)$, time algorithms that supply oracle access to a vertex partition of every planar graph[2] so that every component is of size $O(1)$ and there are $o(n)$ edges connecting these components. As is the case in sublinear time algorithms, we assume that the algorithm can quickly query the input object. In our case, the algorithm can query whether any $k$-tuple of vertices is an edge in $\mathcal{H}$. Given the discussion above, we introduce the following definition.

▶ **Definition 3** (Coloring-Oracle). *A coloring-oracle is a randomized algorithm $A$ whose input is a vertex $u$ in a 2-colorable $k$-graph $\mathcal{H}$. The algorithm $A$ can access $\mathcal{H}$ using edge queries of the form "is $(v_1, \ldots, v_k)$ an edge in $\mathcal{H}$"? The algorithm should always return a color $0/1$ for $u$. Furthermore:*

- **(i)** *For every sequence of queries $u_1, u_2 \ldots$, the answers of $A$ are consistent with a single legal 2-coloring of $\mathcal{H}$.*
- **(ii)** *The oracle uses only $o(n)$ memory for keeping information between different calls.*

Observe that without requirement $(i)$ the definition would be trivial since the oracle could just answer 0 for every vertex, as every vertex can be colored 0 in some 2-coloring of $\mathcal{H}$. Requirement $(ii)$ is also necessary, since if the algorithm is allowed to keep $n$ bits of information, then when it is first called it can find a legal 2-coloring, store it in memory, and then use it to answer subsequent calls. A moment's reflection reveals[3] that there is in fact a (not very efficient) algorithm satisfying Definition 3.

Given Theorems 1 and 2 and the success of partition oracles, it is natural to ask if it is possible to design a coloring oracle that is efficient on average. Let us then introduce the following definition. If $A$ is a coloring oracle, then we use $T_A(\mathcal{H}, u)$ to denote the expected

[2] The results actually hold for more general "minor closed" families of graphs.
[3] Indeed, given $u$ the algorithm asks about all possible edges of $\mathcal{H}$, then looks for the lexicographically first 2-coloring of $\mathcal{H}$ and returns the color it assigns to $u$.

time it takes $A$ to return a color for $u$ in $\mathcal{H}$. Since we are considering average case behavior, it might seem natural to define $T_A(\mathcal{H})$ as the average of $T_A(\mathcal{H}, u)$ over all vertices of $\mathcal{H}$, but we instead make a stronger requirement and define it as the *worst case* over all vertices, that is $T_A(\mathcal{H}) = \max_u T_A(\mathcal{H}, u)$. We finally define $T_A(n)$ as the average of $T_A(\mathcal{H})$ over all 2-colorable $k$-graphs on $n$ vertices. Since hypergraph 2-colorability is NP-hard we certainly do not expect to have a coloring oracle $A$ for which $T_A(\mathcal{H})$ is sub-exponential for every $\mathcal{H}$. Surprisingly, there is a coloring oracle that on average is as efficient as possible.

▶ **Theorem 4.** *There is a coloring oracle $A$ satisfying $T_A(n) = O(1)$. Furthermore, the oracle does not use* **any** *memory for keeping information between successive calls.*

We find it quite surprising that it is possible to not use any shared memory between successive calls, and still answer every query consistently in average case $O(1)$ time. Thus the main conceptual contribution of this work is demonstrating just how much more efficient can Oracles and LCAs be on average, compared to their worst case behavior.

Comparing coloring oracles and partition oracles, in addition to differing algorithmic goals[4], note that a coloring oracle handles all inputs and solves the 2-coloring problem exactly, while a partition oracle only handles planar graphs and only solves an approximate problem. On the other hand, partition oracles always work in $O(1)$ time, while a coloring oracle only works in $O(1)$ time on average over the set of colorable graphs.

### 1.2.1 Transforming the Coloring Oracle into an LCA

In recent years, there has been extensive work on a new model of distributed computing known as Local Computation Algorithms (LCAs) [38, 5, 7]. In this model, the algorithm is given probe access to the input object (in this case, the $k$-graph) and a fixed random string, and must answer queries regarding a particular combinatorial structure defined on it (in this case, the color of a vertex $u$ in a 2-coloring). The answers must be globally consistent, each query must be answered with sublinear work, and there is no persistent memory between queries. We observe that our result implies an *average-case* LCA for coloring, which we now discuss.

LCAs for vertex coloring (over worst-case graphs) have been the subject of extensive study. Much of this has focused on $(\Delta + 1)$-coloring, where $\Delta$ is the maximum degree of the input graph. First, it is known that $r$-round algorithms in the distributed LOCAL model over graphs with maximum degree $\Delta$ can be used to construct LCAs with query processing time $O(\Delta^r)$ using a reduction of Parnas and Ron [33]. Applying this to the result of [8] in the distributed LOCAL model yields a $\Delta^{O(\log^*(\Delta))} \cdot \log(n)$ time $(\Delta + 1)$-coloring LCA. Notably, [9] later constructed a $\Delta^{O(1)} \cdot O(\log(n))$ time LCA for $(\Delta + 1)$-coloring, also proving results in other distributed and sublinear models. For hypergraphs, when a two-coloring is guaranteed by the Lovász Local Lemma, the work of [16] gives an LCA that answers queries in polylogarithmic time. Note that the latter result applies when there is a bound on the degree of the hypergraph, whereas the graphs we consider are usually dense. Recent papers have also explored LCAs for 3-coloring and 2-coloring graphs and hypergraphs typically with additional assumptions made, such as access to linear preprocessing probes [13], or answering only up to polylogarithmic many queries [2].

Graph and hypergraph coloring have been studied in other sublinear access models [6] and through the lens of property testing [11, 12, 1]. Additionally, substantial effort has gone into designing (worst-case) LCAs for a variety of other fundamental problems, including maximal independent set [19, 20, 18, 28], and maximal matching [28, 41, 32].

[4] A partition oracle seeks to partition the graph into $O(n)$ sets with as few edges as possible between them, while a coloring oracle's goal is to partition the graph into $O(1)$ sets with no edges inside them.

A recent paper [7] initiated the study of LCAs over average-case inputs. An oracle is an *average-case LCA* for a problem $\Pi$ over a distribution $\mathcal{G}$ over objects if, with probability at least $(1 - \frac{1}{n})$ over $G \leftarrow \mathcal{G}$, the oracle is an LCA for $\Pi$ on $G$. For hypergraph 2-coloring, the key distinction between average-case LCAs and coloring oracles is the allowed failure probability (the LCA can fail on some inputs, while the coloring oracle must always return a 2-coloring).

We observe that our result implies a very efficient average-case LCA for coloring, which also holds for 2-colorable 2-graphs due to the different failure criterion.

▶ **Theorem 5.** *For all $k \geq 2$, there is an average-case LCA for the uniform distribution over 2-colorable $k$-graphs with worst-case probe complexity of $O(1)$ and runtime per query of $O(1)$[5]. Moreover, the average-case LCA does not require shared randomness between queries.*

We view this as the most nontrivial example of an average-case LCA so far [7]. We present the proof in the journal version of the paper.

## 1.3 Key New Idea and Comparison to Prior Works

The key idea in many average case algorithms is to define a property $\mathcal{P}$ which is useful in the following sense: on one hand almost all objects (graphs, hypergraphs, etc) satisfy $\mathcal{P}$, and on the other hand every object satisfying $\mathcal{P}$ is easy to solve. As we observed earlier, it is not enough for $\mathcal{P}$ to hold for $(1 - o(1))$-fraction or even for $(1 - 2^{-n/2})$-fraction of the objects since that could result in an exponential average case running time if the objects without property $\mathcal{P}$ take exponential time. It is interesting to note that a similar challenge of coming up with a useful property appears also in the design of regularity lemmas for graphs [39] and hypergraphs [21, 37]. There, the goal is to come up with a property that is strong enough for the purposes of applying the lemma, and weak enough to be satisfied by every graph. It is thus no coincidence that the algorithms of Person and Schacht [34, 35] and Lee, Molla, and Nagle [26] used useful properties involving notions related to graph and hypergraph regularity. Hence, they also used the graph/hypergraph regularity lemmas and algorithms [3], leading to huge hidden constants and a complicated analysis. We show there is a very simple-to-state useful property, which we describe in Section 2. While it takes some work to show that most hypergraphs satisfy it, designing an efficient algorithm for hypergraphs satisfying it is almost trivial. Moreover, the property directly leads to the coloring oracle of Theorem 4. Very roughly, the property states that there is a small substructure that has a unique coloring (what we call $K_{\ell,\ell}$ in Section 2) such that the (unique) coloring of this structure uniquely determines the color of *all* vertices of the hypergraph. Most importantly, the coloring of this small substructure forces the colors of all other vertices via "paths" of short length (actually length 2) and there are in fact "many" such paths, making it easy to find one using sampling. We can also take advantage of this property in order to avoid using any memory between successive calls and still maintain the consistency of the coloring. The most (actually, only) technically demanding part of the paper is thus not the design or the analysis of the algorithms, but the proof of Lemma 8 regarding properties of typical 2-colorable $k$-graphs. See the end of Section 2 for a brief description of the proof of this lemma, and the journal version of the paper for the full proof.

[5] This is in the LCA model where the algorithm is given access to a random string consisting of *words* i.e. random entries in $[n]$. A random word corresponds to getting $O(\log n)$ bits of randomness in a step.

## 1.4 Paper Overview

In Section 2 we formally define the useful hypergraph property that underpins all our algorithms here and state the key probabilistic fact regarding this property, see Lemma 8. In Section 3 we prove Theorem 1. To this end, we present a new average case polynomial time deterministic 2-coloring algorithm which relies on the useful property introduced in Section 2. In Section 4 we prove Theorems 2 and 4. To do so, we make subtle adjustments to the algorithm presented in Section 3.

# 2 A Useful Property for Coloring Hypergraphs

Our goal in this section is to define the useful hypergraph property alluded to in the proof overview above. An *independent set* $I$ in a $k$-graph is a set of vertices so that none of the edges of the hypergraph is fully contained in $I$. Note that a 2-coloring of a hypergraph naturally partitions its vertex set into two independent sets, namely those that are colored red and those that are colored blue. Given a 2-colorable $k$-graph $\mathcal{H}$, when we refer to the two independent sets of $\mathcal{H}$, we mean the two independent sets given by some 2-coloring of $\mathcal{H}$. Note that (unless specified otherwise) we do not assume that $\mathcal{H}$ has a unique 2-coloring. For a vertex $u \in V(\mathcal{H})$ and a set of vertices $A \subseteq V(\mathcal{H})$, we use $N(u, A)$ to denote the set of $(k-1)$-tuples of vertices in $A$ which form an edge together with $u$. So $N(u, A)$ are the neighbors of vertex $v$ in the set $A$, but as opposed to 2-graphs, where the neighbors of a vertex are also vertices, now the neighbors are $(k-1)$-tuples of vertices. For an integer $\ell \geq k-1$ the $k$-graph $K^{(k)}_{\ell,\ell}$ is the one consisting of two vertex sets $A, B$ of size $\ell$ and of all the edges that have a single vertex in either $A$ or $B$ and $k-1$ vertices in the other set. To simplify the notation, we will use $K_{\ell,\ell}$ instead of $K^{(k)}_{\ell,\ell}$. We first observe the following simple fact.

▷ Claim 6. For every $\ell \geq 2k-3$, the $k$-graph $K_{\ell,\ell}$ has a unique 2-coloring.

Proof. Let $A, B$ be the color classes in the definition of $K_{\ell,\ell}$, and consider any other 2-coloring. Assume without loss of generality that two vertices $a, a' \in A$ are assigned different colors. Since $\ell \geq 2k-3$ there are $k-1$ vertices in $A$ that received the same color.[6] Suppose this is color 1, that $a$ is colored 1, and that $a'$ is colored 0. It follows from the definition of $K_{\ell,\ell}$ that all vertices in $B$ must be colored 0, since each $b \in B$ forms an edge with the $k-1$ vertices of $A$ that are colored 1. But then every edge containing $k-1$ vertices in $B$ and vertex $a'$ is not colored properly. ◁

Given a copy of $K_{\ell,\ell}$ we will use $A, B$ to denote the two colors classes in its unique 2-coloring, namely the two sets used in the definition of $K_{\ell,\ell}$. We are now ready to define the useful property, which we call *good*.

▶ **Definition 7** (Good $k$-Graphs). *Suppose $\mathcal{H}$ is an $n$-vertex 2-colorable $k$-graph. Set*

$$\ell = \ell(k) = 5k. \tag{1}$$

*Then $\mathcal{H}$ is good if it satisfies:*

**(i)** *The number of copies of $K_{\ell,\ell}$ in $\mathcal{H}$ is at least $n^{2\ell}/2^{2^{10k}}$.*

[6] Note that the assumption $\ell \geq 2k-3$ is indeed needed since when $\ell = 2k-4$ there are different 2-colorings of $K_{\ell,\ell}$ obtained by coloring $k-2$ of the vertices in $A, B$ with color 0/1.

**(ii)** *The following holds for every copy $K$ of $K_{\ell,\ell}$ in $\mathcal{H}$. Suppose $A, B$ are the independent sets of $K$. Let $N_A$ be the vertices $v$ satisfying $N(v, A) \neq \emptyset$ and $N_B$ be the vertices $v$ satisfying $N(v, B) \neq \emptyset$. Then every vertex $u$ in $\mathcal{H}$ satisfies either $|N(u, N_A)| \geq n^{k-1}/k^{4k}$ or $|N(u, N_B)| \geq n^{k-1}/k^{4k}$.*

See Figures 1 and 2 for illustrations of the good property and how the good property assists the algorithms.

Recall that for a property to be useful for our purposes here we first need all but a negligible fraction of the 2-colorable $k$-graphs to satisfy it. This is precisely the statement of the following lemma, whose (somewhat technical) proof appears in the journal version of the paper. In what follows $\mathcal{T}_n^{(k)}$ is the set of all 2-colorable $k$-graphs on $n$ vertices, and $\mathcal{H} \sim \mathcal{T}_n^{(k)}$ means that $\mathcal{H}$ is uniformly chosen from $\mathcal{T}_n^{(k)}$.

▶ **Lemma 8.** *If $\mathcal{H} \sim \mathcal{T}_n^{(k)}$, then for all large enough $n$, $\mathcal{H}$ is good with probability at least $1 - 2^{-2n}$.*

The main idea of the proof of Lemma 8 is the following. Consider an alternative probabilistic model for picking a random 2-colorable $k$-graph on a set $V$ of $n$ vertices, which we denote $\mathcal{T}_{S,n}^{(k)}$: in this case we preselect two "planted" independent sets $S$ and $V \setminus S$ (where we will add no edges) and pick each edge intersecting $S$ and $V \setminus S$ with probability $1/2$. It is easy to show (see the journal version of the paper), using classical tail bounds, that a $k$-graph generated by $\mathcal{T}_{S,n}^{(k)}$ such that $n/4 \leq |S| \leq 3n/4$ is good with probability at least $1 - 2^{-3n}$. What is also "clear" is that $\mathcal{H} \sim \mathcal{T}_n^{(k)}$ is "similar" to $\mathcal{H} \sim \mathcal{T}_{S,n}^{(k)}$ when $|S|$ is close to $n/2$. Perhaps surprisingly, there is a very short proof making this intuition precise. The proof of Lemma 8 will appear in the journal version of the paper.

## 3 Coloring Bipartite Hypergraphs

Our goal in this section is to prove Theorem 1. Throughout this section we use $\ell = 5k$ as in Definition 7. By Lemma 8 only a tiny fraction of the bipartite $k$-graphs are not good. It is thus easy to see that in order to prove Theorem 1 it is enough to design a deterministic algorithm which finds in polynomial time a 2-coloring of every good 2-colorable $k$-graphs. For notational reasons we will sometimes use the term "bipartite" instead of "2-colorable".

Before presenting the concrete algorithm, we give an informal description of it. The algorithm first uses exhaustive search in order to find a copy of $K_{\ell,\ell}$. If $\mathcal{H}$ is good then we know that it has many copies of $K_{\ell,\ell}$. Letting $A, B$ be the independent sets in the unique 2-coloring the $K_{\ell,\ell}$ the algorithm found, it then colors $A$ with 0 and $B$ with 1. The algorithm now looks for all vertices $v$ which form an edge with $k-1$ vertices from $A$ or $B$. Note that the color of such a $v$ is uniquely determined by the coloring of $A$ and $B$ and thus we color them accordingly. The algorithm now looks for all vertices $v$ which form an edge with $k-1$ vertices that were colored 0 or 1 in the previous step. Again, the color of such a $v$ is uniquely determined by the coloring we made in the previous step. If $\mathcal{H}$ is good we thus color all of its vertices. If any step of the algorithm fails, it just uses exhaustive search to find a legal 2-coloring. We remind the reader that $N(u, A)$ denotes the set of $(k-1)$-tuples of vertices in $A$ which form an edge with $u$.

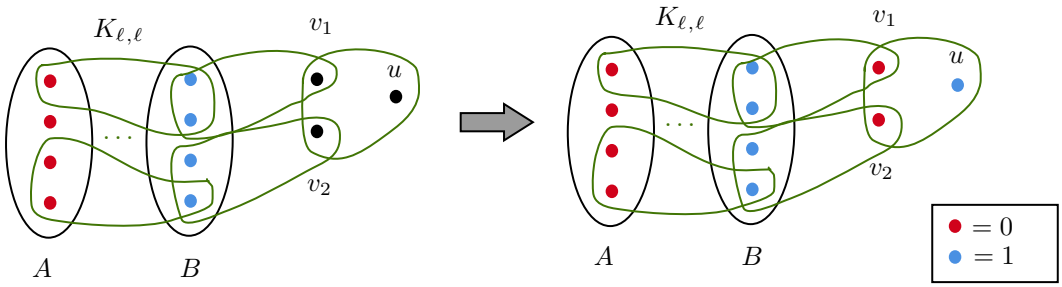


**Figure 1** Illustration of how vertices are colored in *good* 3-graphs (Definition 7) by Algorithm 1. Once a copy of $K_{\ell,\ell}$ is found and (uniquely) colored, all other vertices will be within a path length of two of this copy and are colored consistently relative to the $K_{\ell,\ell}$ copy.

**Algorithm 1** Deterministic Coloring Algorithm for Bipartite $k$-Graphs.

**Input:** A bipartite $k$-graph $\mathcal{H}$.
**Output:** A proper 2-coloring of $\mathcal{H}$.
**Procedure:** COLOR-BIPARTITE-HYPERGRAPH($\mathcal{H}$)

1. Search for a copy of $K_{\ell,\ell}$ using exhaustive search.
2. **If** no copy of $K_{\ell,\ell}$ was found, proceed to Step 13.
3. Let $A, B$ be the independent sets of the copy of $K_{\ell,\ell}$ found in Step 1.
4. Color the vertices in $A$ with color 0 and those in $B$ with color 1.
5. $\forall u \in V(H)$ **do**
6. **If** $N(u, B) \neq \emptyset$, then color $u$ with color 0.
7. **If** $N(u, A) \neq \emptyset$, then color $u$ with color 1.
8. Let $C_0$ (resp. $C_1$) be the vertices colored 0 (resp. 1) thus far.
9. $\forall u \in V(H) \setminus \{C_0 \cup C_1\}$ **do**
10. **If** $N(v, C_1) \neq \emptyset$, then color $u$ with color 0.
11. **If** $N(v, C_0) \neq \emptyset$, then color $u$ with color 1.
12. **If** all vertices were colored then return this 2-coloring, **Else** go to Step 13.
13. Exhaustively search for a legal 2-coloring of $\mathcal{H}$. Return the first one found.

We begin by demonstrating that COLOR-BIPARTITE-HYPERGRAPH indeed produces a proper 2-coloring.

▶ **Lemma 9.** *Suppose $\mathcal{H}$ is a bipartite $k$-graph. Then the algorithm* COLOR-BIPARTITE-HYPERGRAPH *returns a proper 2-coloring of $\mathcal{H}$.*

**Proof.** If a copy of $K_{\ell,\ell}$ is not found, then we run Step 13 which finds a legal 2-coloring (since one exists) so the claim holds in this case. Suppose then that a copy of $K_{\ell,\ell}$ was found, and fix a legal 2-coloring $c : V(\mathcal{H}) \mapsto \{0, 1\}$ of the vertices of $\mathcal{H}$. Recall that by Claim 6 $K_{\ell,\ell}$ has unique 2-coloring. Hence, $c$ assigns all vertices of $A$ (resp. $B$) the same color. Assume without loss of generality that these are colors 0 and 1 as in our coloring. Clearly, $c$ colors the vertices colored in Steps 6/7 in the same color as the algorithm does. Similarly, $c$ colors the vertices colored in Steps 10/11 in the same color as the algorithm does. Hence, if we colored all of the vertices of $\mathcal{H}$ then our coloring agrees with $c$ which is a legal 2-coloring of $\mathcal{H}$ (which means that $c$ is the unique 2-coloring of $\mathcal{H}$). If the algorithm did not color all the vertices then it again resorts to exhaustively looking for a legal 2-coloring. ◀

For a given 2-coloring algorithm $A$ and input $\mathcal{H}$, let $T_A(\mathcal{H})$ denote the running time of $A$ on input $\mathcal{H}$. Let $T_A(n)$ be the average of $T_A(\mathcal{H})$ over all $n$-vertex bipartite $k$-graphs $\mathcal{H}$, that is, the average case running time of $A$ over the $n$-vertex bipartite $k$-graphs.

▶ **Lemma 10.** *The algorithm* Color-Bipartite-Hypergraph *satisfies*

$$T_{\textsc{Color-Bipartite-Hypergraph}}(n) = n^{O(k)} .$$

**Proof.** The proof can be found in the journal version of the paper. ◀

**Proof of Theorem 1.** By Lemma 9 Color-Bipartite-Hypergraph always returns a legal 2-coloring, and by Lemma 10 its average running time is $n^{O(k)}$. It is also clear that the algorithm does not use any memory to keep information between successive calls to it. ◀

## 4 A Coloring Oracle for Bipartite Hypergraphs

In this section we prove Theorems 4 and 2. Throughout this section we use $\ell = 5k$ as in Definition 7. To this end, we will modify the algorithm Color-Bipartite-Hypergraph presented in the previous section in order to turn it into a coloring oracle and subsequently into a randomized algorithm with linear expected running time. Let us describe the key ideas needed to make the running time $O(1)$ on average and how to avoid using any memory between successive calls and still maintain a consistent coloring. Observe that in order to get running time $O(1)$ on average it is enough to obtain this running time for good hypergraphs. For such inputs, we can in fact find a copy of $K_{\ell,\ell}$ in $O(1)$ time since a positive proportion of all $2\ell$-tuples contain a $K_{\ell,\ell}$ (see Step 2 of Algorithm 2 below). It is easy to see that if $\mathcal{H}$ is good then a coloring of a copy of $K_{\ell,\ell}$ forces a coloring of every vertex $u$ in $\mathcal{H}$. In fact, there are many $(k-1)$-tuples $v_1, \ldots, v_{k-1}$ which witness this fact, so the color of $u$ can be deduced in $O(1)$ time (as in Steps 4-5). The challenge is that the Oracle should answer consistently for *every* input, hence we need a mechanism for solving this issue. This is achieved by Step 2, in which we not only try to find a copy of $K_{\ell,\ell}$ but we also demand that this copy forces a color for vertex 1. We always assume that vertex 1 is colored 0 (see below why), so in this sense vertex 1 forces the colors of $A$ and $B$. The colors of $A$ and $B$ then force the colors of other vertices $u$ (but *not* necessarily all of them) in Steps 4-5. If we think of a 2-coloring of $\mathcal{H}$ as a 0/1 string of length $n$, then the color of vertex 1 is the most significant bit, hence the lexicographically first legal 2-coloring of $\mathcal{H}$ must assign vertex 1 the color 0 (since flipping the colors of a legal coloring is also a legal coloring). This is why it is convenient to also look for a coloring giving 1 the color 0.

▶ **Lemma 11.** *Algorithm* Coloring-Oracle *is a 2-coloring oracle. That is, if $\mathcal{H}$ is a bipartite $k$-graph, then for every sequence of queries $u_1, u_2, \ldots$ the oracle's answers are consistent with the same legal 2-coloring of $\mathcal{H}$.*

**Proof.** Let $c$ be the lexicographically first legal 2-coloring of $\mathcal{H}$. Note that such a coloring must assign vertex 1 the color 0. We claim that for any sequence of queries, the oracle's answers are always consistent with $c$. This is certainly the case if when applied to vertex $u$, the algorithm ever resorts to Step 7, so suppose it does not. In this case we know that the algorithm found a copy of $K_{\ell,\ell}$, a $(k-1)$-tuple of vertices satisfying conditions $(ii.1)$ or $(ii.2)$ and a $(k-1)$-tuple satisfying either Step 4 or 5. Suppose wlog that steps $(ii.1)$ and 4 are the ones that were satisfied (the other 3 options are identical). First recall that by Claim 6 every legal 2-coloring of $\mathcal{H}$ gives the vertices of $A$ the same color and to those in $B$ the other color. If $(ii.1)$ holds then each of the vertices $y_1, \ldots, y_{k-1}$ forms an edge with a $(k-1)$-tuple in $A$ implying that in any legal 2-coloring of $\mathcal{H}$ the vertices $y_1, \ldots, y_{k-1}$ are colored as $B$. Since $(ii.1)$ further assumes that $(1, y_1, \ldots, y_{k-1}) \in E(\mathcal{H})$ we conclude that in any legal 2-coloring of $\mathcal{H}$ the set $A$ is assigned the same color as vertex 1. In particular,

**Algorithm 2** Coloring Oracle for Bipartite $k$-Graphs.

**Input:** Vertex $u$ in some bipartite $k$-graph $\mathcal{H}$, and oracle access to $E(\mathcal{H})$.
**Output:** Color assignment to $u$.

1. **Procedure:** Coloring-Oracle($\mathcal{H}$,$u$)
2. Uniformly sample $(2\ell+k-1)$-tuples $(x_1,\ldots,x_{2\ell},y_1,\ldots,y_{k-1})$ of vertices until $(i)$ and $(ii)$ hold:
   - $(i)$ Vertices $x_1,\ldots,x_{2\ell}$ span a copy of $K_{\ell,\ell}$. Set $A,B$ to be the independent sets of this copy.
   - $(ii)$ Vertices $y_1,\ldots,y_{k-1}$ satisfy one of the following:
     - $(ii.1)$ : $(1,y_1,\ldots,y_{k-1})\in E(\mathcal{H})$ and $N(y_i,A)\neq\emptyset$ for every $i\in[k-1]$. Set $c_A=0,c_B=1$
     - $(ii.2)$ : $(1,y_1,\ldots,y_{k-1})\in E(\mathcal{H})$ and $N(y_i,B)\neq\emptyset$ for every $i\in[k-1]$. Set $c_A=1,c_B=0$

   **If** all $(2\ell+k-1)$-tuples were inspected, and none of them satisfies $(i),(ii)$, goto Step 7.
3. Uniformly sample $(k-1)$-tuples of vertices $v_1,\ldots,v_{k-1}$:
4. **If** $(u,v_1,\ldots,v_{k-1})\in E(\mathcal{H})$ and $N(v_i,A)\neq\emptyset$ for every $i\in[k-1]$ then return $c_A$.
5. **If** $(u,v_1,\ldots,v_{k-1})\in E(\mathcal{H})$ and $N(v_i,B)\neq\emptyset$ for every $i\in[k-1]$ then return $c_B$.
6. **If** all $(k-1)$-tuples have been inspected, and none of them satisfies $4-5$, then goto Step 7.
7. Exhaustively search for the lexicographically first legal 2-coloring $c$ of $\mathcal{H}$. Return $c(v)$.

this means that $c$ assigns $A$ the color 0 and $B$ the color 1. Hence we set $c_A=0,c_B=1$ to indicate this fact. By an identical argument, if Step 4 holds then in any legal 2-coloring of $E(\mathcal{H})$, and in particular in $c$, vertex $u$ receives the same color as $A$. Hence returning color $c_A$ for $u$ is consistent with $c$. ◀

Recall that if $A$ is a coloring oracle, $\mathcal{H}$ is a bipartite $k$-graph and $u\in V(\mathcal{H})$, then we use $T_A(\mathcal{H},u)$ to denote the expected running time it takes $A$ to return a color for $u$. We further set $T_A(\mathcal{H})=\max_{u\in V(\mathcal{H})}T_A(\mathcal{H},u)$, and $T_A(n)$ as the average of $T_A(\mathcal{H})$ over all bipartite $n$-vertex $k$-graphs $\mathcal{H}$.

▶ **Lemma 12.** *The algorithm Coloring-Oracle satisfies*

$$T_{\text{Coloring-Oracle}}(n)=O(1)\ .$$

**Proof.** The proof can be found in the journal version of the paper. ◀

**Proof (of Theorem 4).** By Lemma 11 the algorithm Coloring-Oracle is indeed a coloring oracle, and by Lemma 12 it satisfies $T_{\text{Coloring-Oracle}}(n)=O(1)$. Finally, note that the random string used on query $u$ is local and does not need to be maintained between queries, so the algorithm requires no persistent storage (and the responses remain consistent by Lemma 11). ◀

**Proof (of Theorem 2).** Suppose we color $\mathcal{H}$ by invoking Coloring-Oracle on all vertices of $\mathcal{H}$. By linearity of expectation, for a given $\mathcal{H}$ the expected running time of the algorithm is at most $n\cdot T_{\text{Coloring-Oracle}}(\mathcal{H})$. By Theorem 4, this means that the average running time of the algorithm over $\mathcal{H}\sim\mathcal{T}_n^{(k)}$ is at most

$$n\cdot T_{\text{Coloring-Oracle}}(n)=n\cdot O(1)=O(n)\ .$$ ◀

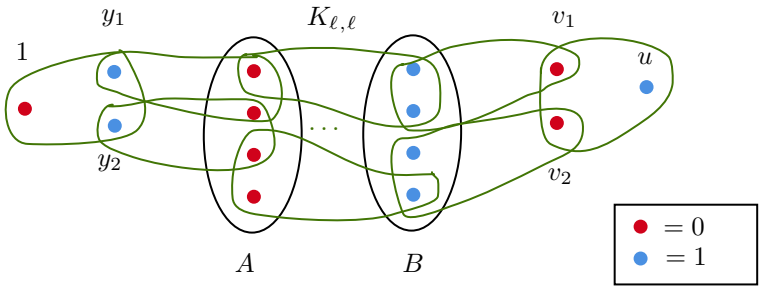


**Figure 2** Illustration of how vertices are colored in *good* 3-graphs (Definition 7) by Algorithm 2. First, once a copy of $K_{\ell,\ell}$ is found, we must determine its coloring relative to how vertex 1 is colored. Therefore, a path of length up to two to vertex 1 is found, and $K_{\ell,\ell}$ is colored to be consistent with vertex 1 being colored 0 (red). All other vertices will be within a path length of two of this copy and are colored consistently relative to the $K_{\ell,\ell}$ copy.

▶ Remark 13. The coloring oracle algorithm immediately yields an (average-case) LCA (as defined in [7]). In the journal version of the paper, we show how to obtain a slightly simpler average-case LCA by modifying the coloring oracle.

## References


1 Hugo Aaronson, Gaia Carenini, and Atreyi Chanda. Property testing in bounded degree hypergraphs. *CoRR*, abs/2502.18382, 2025. doi:10.48550/arXiv.2502.18382.

2 D. Achlioptas, T. Gouleakis, and F. Iliopoulos. Simple local computation algorithms for the general Lovász Local Lemma. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '20, pages 1–10. ACM, 2020. doi:10.1145/3350755.3400250.

3 N. Alon, R. A. Duke, H. Lefmann, V. Rödl, and R. Yuster. The algorithmic aspects of the regularity lemma. *Journal of Algorithms*, 16:80–109, 1994. doi:10.1006/JAGM.1994.1005.

4 Noga Alon, Pierre Kelsen, Sanjeev Mahajan, and Hariharan Ramesh. Coloring 2-colorable hypergraphs with a sublinear number of colors. *Nordic J. of Computing*, 3(4):425–439, December 1996.

5 Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1132–1139. SIAM, 2012. doi:10.1137/1.9781611973099.89.

6 S. Assadi, Y. Chen, and S. Khanna. Sublinear algorithms for ($\Delta$+ 1) vertex coloring. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 767–786. SIAM, 2019. doi:10.1137/1.9781611975482.48.

7 Amartya Shankha Biswas, Ruidi Cao, Cassandra Marcussen, Edward Pyne, Ronitt Rubinfeld, Asaf Shapira, and Shlomo Tauber. Beyond worst case local computation algorithms. *CoRR*, abs/2403.00129, 2025. doi:10.48550/arXiv.2403.00129.

8 Y. Chang, W. Li, and S. Pettie. An optimal distributed ($\Delta$+ 1)-coloring algorithm? In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC '18, pages 445–456. ACM, 2018. doi:10.1145/3188745.3188964.

9 Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The complexity of ($\delta$+ 1) coloring in congested clique, massively parallel computation, and centralized local computation. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 471–480, 2019. doi:10.1145/3293611.3331607.

10 Hui Chen and Alan M. Frieze. Coloring bipartite hypergraphs. In *Proceedings of the 5th International IPCO Conference on Integer Programming and Combinatorial Optimization*, pages 345–358, Berlin, Heidelberg, 1996. Springer-Verlag. doi:10.1007/3-540-61310-2_26.

11 A. Czumaj and C. Sohler. Testing hypergraph coloring. In *Automata, Languages and Programming: 28th International Colloquium, ICALP 2001*, volume 2076 of *Lecture Notes in Computer Science*, pages 493–505. Springer, 2001. doi:10.1007/3-540-48224-5_41.

12 A. Czumaj and C. Sohler. Abstract combinatorial programs and efficient property testers. *SIAM Journal on Computing*, 34(3):580–615, 2005. doi:10.1137/S009753970444199X.

13 Artur Czumaj, Yishay Mansour, and Shai Vardi. Sublinear graph augmentation for fast query implementation. In *Approximation and Online Algorithms: 16th International Workshop, WAOA 2018, Helsinki, Finland, August 23-24, 2018, Revised Selected Papers*, pages 181–203, Berlin, Heidelberg, 2018. Springer-Verlag. doi:10.1007/978-3-030-04693-4_12.

14 Artur Czumaj and Christian Scheideler. Coloring non-uniform hypergraphs: a new algorithmic approach to the general lovász local lemma. In David B. Shmoys, editor, *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA*, pages 30–39. ACM/SIAM, 2000. URL: http://dl.acm.org/citation.cfm?id=338219.338229.

15 Artur Czumaj and Christian Scheideler. A new algorithm approach to the general lovász local lemma with applications to scheduling and satisfiability problems (extended abstract). In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 38–47, New York, NY, USA, 2000. Association for Computing Machinery. doi:10.1145/335305.335310.

16 A. Dorobisz and J. Kozik. Local Computation Algorithms for Hypergraph Coloring-Following Beck's Approach. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, LIPIcs, Vol. 261, pages 48:1–48:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.48.

17 M. E. Dyer and A. M. Frieze. The solution of some random NP-hard problems in polynomial expected time. *J. Algorithms*, 10(4):451–489, December 1989. doi:10.1016/0196-6774(89)90001-1.

18 J. Ghaffari and J. Uitto. Sparsifying Distributed Algorithms with Ramifications in Massively Parallel Computation and Centralized Local Computation. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 1636–1653. SIAM, 2019. doi:10.1137/1.9781611975482.99.

19 Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 270–277, USA, 2016. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611974331.CH20.

20 Mohsen Ghaffari. Local computation of maximal independent set. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 438–449. IEEE, 2022. doi:10.1109/FOCS54457.2022.00049.

21 W. T. Gowers. Hypergraph regularity and the multidimensional Szemerédi theorem. *Annals of Mathematics*, 166(3):897–946, 2007.

22 Avinatan Hassidim, Jonathan A. Kelner, Huy Ngoc Nguyen, and Krzysztof Onak. Local graph partitions for approximation and testing. *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 22–31, 2009. doi:10.1109/FOCS.2009.77.

23 Michael Krivelevich, Ram Nathaniel, and Benny Sudakov. Approximating coloring and maximum independent sets in 3-uniform hypergraphs. *Journal of Algorithms*, 41(1):99–113, 2001. doi:10.1006/jagm.2001.1173.

24 Michael Krivelevich and Benny Sudakov. Approximate coloring of uniform hypergraphs. *Journal of Algorithms*, 49(1):2–12, 2003. 1998 European Symposium on Algorithms. doi:10.1016/S0196-6774(03)00077-4.

25 Akash Kumar, C. Seshadhri, and Andrew Stolman. Random walks and forbidden minors III: poly(d/$\epsilon$)-time partition oracles for minor-free graph classes. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 257–268. IEEE, 2021. doi:10.1109/FOCS52979.2021.00034.

26 Boyoon Lee, Theodore Molla, and Brendan Nagle. On two-coloring bipartite uniform hypergraphs. *CoRR*, 2404.05026, 2024. arXiv:2404.05026.

27 Reut Levi and Dana Ron. A quasi-polynomial time partition oracle for graphs with an excluded minor. *ACM Transactions on Algorithms*, 11(3), January 2015. doi:10.1145/2629508.

28 Reut Levi, Ronitt Rubinfeld, and Anak Yodpinyanee. Local computation algorithms for graphs of non-constant degrees. *Algorithmica*, 77(4):971–994, 2017. doi:10.1007/s00453-016-0126-y.

29 László Lovász. Coverings and colorings of hypergraphs. *Proc. 4th Southeastern Conference of Combinatorics, Graph Theory, and Computing*, pages 3–12, 1973. URL: https://cir.nii.ac.jp/crid/1572261549354589440.

30 P. Erdos-L Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. *Coll Math Soc J Bolyai*, 1974. URL: https://api.semanticscholar.org/CorpusID:6519125.

31 Chi-Jen Lu. Deterministic hypergraph coloring and its applications. *SIAM J. Discret. Math.*, 18:320–331, 1998. URL: https://api.semanticscholar.org/CorpusID:1265012.

32 Yishay Mansour and Shai Vardi. A local computation approximation scheme to maximum matching. In Prasad Raghavendra, Sofya Raskhodnikova, Klaus Jansen, and José D. P. Rolim, editors, *APPROX/RANDOM 2013*, volume 8096 of *Lecture Notes in Computer Science*, pages 260–273. Springer, 2013. doi:10.1007/978-3-642-40328-6_19.

33 M. Parnas and D. Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 382:183–196, 2007.

34 Y. Person and M. Schacht. Almost all hypergraphs without Fano planes are bipartite. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 217–226. SIAM, 2009. doi:10.1137/1.9781611973068.25.

35 Yury Person and Mathias Schacht. An expected polynomial time algorithm for coloring 2-colorable 3-graphs. *Electronic Notes in Discrete Mathematics*, 13:465–469, January 2011. doi:10.1016/j.endm.2009.07.077.

36 J. Radhakrishnan and A. Srinivasan. Improved bounds and algorithms for hypergraph two-coloring. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No.98CB36280)*, pages 684–693, 1998. doi:10.1109/SFCS.1998.743519.

37 V. Rödl, B. Nagle, J. Skokan, M. Schacht, and Y. Kohayakawa. The hypergraph regularity method and its applications. *Proceedings of the National Academy of Sciences of the United States of America*, 102(23):8109–8113, 2005.

38 Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In Bernard Chazelle, editor, *Innovations in Computer Science - ICS 2011, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 223–238. Tsinghua University Press, 2011. URL: http://conference.iiis.tsinghua.edu.cn/ICS2011/content/papers/36.html.

39 E. Szemerédi. Regular partitions of graphs. In *Problèmes combinatoires et théorie des graphes (Colloq. Internat. CNRS, Univ. Orsay, Orsay, 1976)*, volume 260 of *Colloques Internationaux du CNRS*, pages 399–401, Paris, 1978. CNRS.

40 J. S. Turner. Almost all $k$-colorable graphs are easy to color. *J. Algorithms*, 9:63–82, 1988. doi:10.1016/0196-6774(88)90005-3.

41 Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. An improved constant-time approximation algorithm for maximum matchings. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, pages 225–234, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1536414.1536447.