Approximate Hypergraph Partitioning and Applications

Eldar Fischer* Arie Matsliah† Asaf Shapira‡

Abstract

We show that any *partition-problem* of hypergraphs has an $O(n)$ time approximate partitioning algorithm and an efficient property tester. This extends the results of Goldreich, Goldwasser and Ron who obtained similar algorithms for the special case of graph partition problems in their seminal paper [16].

The partitioning algorithm is used to obtain the following results:

- We derive a surprisingly simple $O(n)$ time algorithmic version of Szemerédi's regularity lemma. Unlike all the previous approaches for this problem [3, 10, 14, 15, 21], which only guaranteed to find partitions of tower-size, our algorithm will find a small regular partition in the case that one exists;
- For any $r \geq 3$, we give an $O(n)$ time randomized algorithm for constructing regular partitions of r-uniform hypergraphs, thus improving the previous $O(n^{2r-1})$ time (deterministic) algorithms [8, 15].

The property testing algorithm is used to unify several previous results, and to obtain the partition densities for the above problems (rather than the partitions themselves) using only poly$(1/\epsilon)$ queries and constant running time.

*Faculty of Computer Science, Technion – Israel institute of technology, Technion City, Haifa 32000, Israel. Email: eldar@cs.technion.ac.il. Research supported in part by an ISF grant number 1011/06.

†Faculty of Computer Science, Technion – Israel institute of technology, Technion City, Haifa 32000, Israel. Email: ariem@cs.technion.ac.il.

‡Microsoft Research. Email: asafico@tau.ac.il.

1 Introduction and general background

Graph partition problems are some of the most well-studied problems both in graph theory and in computer-science. Standard examples of partition problems include k-colorability, Max-Clique and Max-Cut. Most of these problems are computationally hard even to approximate, but it was observed in the 90's [5, 11] that many of these partition problems have good approximations when the input graph is dense. In this paper we introduce an efficient $O(n)$ algorithm for partitioning *hypergraphs*, with an accompanying $O(1)$ query complexity test for the existence of a partition with given parameters. We show that several previous results on graph and hypergraph partition problems follow as special cases of our main result. In some cases the our results will actually improve upon the previously known ones.

Our framework for studying hypergraph partition problems generalizes the framework of graph partition problems that was introduced by Goldreich, Goldwasser and Ron [16]. Let us briefly discuss the graph partitioning algorithm of [16]. A graph *partition-instance* Ψ is composed of an integer k specifying the number of sets in the required partition $V_1, \ldots, V_k$ of the graph's vertex set, and intervals specifying the allowed ranges for the number of vertices in every V_i and the number of edges between every V_i and V_j for $i \leq j$.

Goldreich, Goldwasser and Ron [16] showed that for any partition-instance Ψ with k parts, and for any fixed ϵ, there is an $O(2^{(k/\epsilon)^{O(k)}} + (k/\epsilon)^{O(k)}n)$ time algorithm that produces a partition of an input graph that is ϵ-close to satisfying Ψ, assuming that a satisfying partition exists (the distance is measured by the differences between the actual densities and the required ones). Note that one can formulate many problems, such as k-colorability, Max-Cut and Max-Clique, in this framework of partition-instances. Therefore, the algorithm of [16], which we will henceforth refer to as the *GGR-algorithm*, implies for example that there is an $O(\exp(1/\epsilon^3) + (1/\epsilon^2)n)$ time algorithm that approximates the size of the maximum cut of a graph to within an additive error of ϵn^2. It also implies that for any $\epsilon > 0$ there is an $O(\exp(1/\epsilon^3) + (1/\epsilon^2)n)$ time algorithm that, given a graph G, either reports that G is not 3-colorable, or 3-colors the vertices of G in such a way that all but at most ϵn^2 of the edges are properly colored.

The second main result of [16] was that in fact, for any partition problem Ψ as above, there is a randomized algorithm making only $(k/\epsilon)^{O(k)}$ many queries to the input graph, and running in time $2^{(k/\epsilon)^{O(k)}}$, which distinguishes with probability $\frac{2}{3}$ between graphs satisfying Ψ and graphs that are ϵ-far from satisfying Ψ. Thus the result of [16] implies that one can distinguish in *constant time* between graphs satisfying a partition instance and graphs that are far from satisfying it.

Here we extend the main results of [16] to the case of hypergraphs by showing that there is a randomized algorithm for the general hypergraph partition-problem. The running time of our algorithm is $O(n)$ (where n is the number of vertices, making this sublinear) and it has the following property: Given an input hypergraph H, which satisfies the partition problem, the algorithm produces a partition of H that is "close" to satisfying it. In the case that no such partition of H exists, the algorithm rejects the input. We also obtain property testing algorithms for such problems making only poly$(1/\epsilon)$ queries, and with a constant running time.

We present several applications of our result, and in the foremost a new application related to Szemerédi's regularity lemma [27]. By using an appropriate hypergraph modeling of the problem we design a surprisingly simple $O(n)$ time algorithm for constructing regular partitions of graphs. An added benefit is that unlike the previous approaches for constructing regular partitions [3, 10, 14,

15, 21], which proved the lemma "algorithmically", our algorithm will find a small regular partition in the case that one exists in our input graph, rather than being guaranteed to find only a partition of the tower-size upper bound given by Szemerédi's lemma itself.

We also design an $O(n)$ time randomized algorithm for constructing regular partitions of r-uniform hypergraphs for any $r \geq 3$. This improves over the previous (deterministic) algorithms [8, 15] that have a running time of $O(n^{2r-1})$.

Our property testing algorithm provides a common generalization for many previously known results. We show how special cases of the now-testable partition problem can be easily used to derive some results that were previously proved using specialized methods, namely testing properties of hypergraphs [7, 23] and estimating k-CNF satisfiability [1].

2 Extension of the GGR-algorithm

Our main result in this paper is a generalization of the GGR-algorithm to the case of hypergraphs. Let us start by defining our framework for studying hypergraph partition problems. We consider directed hypergraphs $H = (V, E_1, E_2, \ldots, E_s)$ with n vertices and s directed edge sets (the generalization to several edge sets also has a role in our main application). Every edge set $E_i(H) \subseteq V^{r_i}$ is a set of *ordered* r_i-tuples (ordered sets with possible repetitions) over V. [1] That is, for every edge set $E_i(H)$ we think of any of the edges $e \in E_i$ as an ordered subset of $V(H)$ of size r_i. Let us put $r = \max_i\{r_i\}$. Note that the usual notion of a directed graph corresponds to hypergraphs with only one edge set E whose edges are of size 2. For an r_i-tuple $e = (v_1, \ldots, v_{r_i}) \in E_i$ we say that v_j is in the j^{th} *place* of e (or is simply the j^{th} *vertex* of e). We use $[k]$ to denote the set $\{1, \ldots, k\}$, and we use $a = b \pm c$ as a shorthand for $b - c \leq a \leq b + c$.

2.1 The partition property

Let H be a hypergraph as above and let $\Pi = \{V_1^\Pi, \ldots, V_k^\Pi\}$ be a partition of $V(H)$. Let us introduce a notation for counting the number of edges of $E_i(H)$ with a specific placement of their vertices within the partition classes of Π (remember that the edges are ordered). For every $i \in [s]$ we denote by Φ_i the set of all possible mappings $\phi : [r_i] \to [k]$. We think of every $\phi \in \Phi_i$ as mapping the vertices of an r_i-tuple to the components of Π. We denote by $E_{i,\phi}^\Pi \subseteq E_i$ the following set of r_i-tuples:

$$E_{i,\phi}^\Pi = \{(v_1, \ldots, v_{r_i}) \in E_i : v_j \in V_{\phi(j)}^\Pi,\ \forall\ 1 \leq j \leq r_i\}$$

We now introduce a notion that generalizes the partition instances of graphs that were discussed earlier in the context of graphs. A *density tensor* is a sequence $\psi = \left\langle \langle \rho_j \rangle_{j\in[k]}, \langle \mu_{i,\phi} \rangle_{i\in[s],\phi\in\Phi_i} \right\rangle$ of reals (between 0 and 1), specifying the presumed normalized sizes of $|V_i^\Pi|$ and $|E_{i,\phi}^\Pi|$ of a partition of a hypergraph H. In particular, given a partition $\Pi = \{V_1^\Pi, V_2^\Pi, \ldots, V_k^\Pi\}$ of a hypergraph H, we set ψ^Π to be the density tensor $\left\langle \langle \rho_j^\Pi \rangle_{j\in[k]}, \langle \mu_{i,\phi}^\Pi \rangle_{i\in[s],\phi\in\Phi_i} \right\rangle$ with the property that for all j, $\rho_j^\Pi = \frac{1}{n} \cdot |V_j^\Pi|$ and for all i and ϕ, $\mu_{i,\phi}^\Pi = \frac{1}{n^{r_i}} \cdot |E_{i,\phi}^\Pi|$.

For a fixed hypergraph H, a set Ψ of general density tensors (with respect to k, s and $r_1, \ldots, r_s$) defines a property of $V(H)$'s partitions as follows. We say that a partition Π of $V(H)$ (exactly) *satisfies* Ψ if there exists a density tensor $\psi = \left\langle \langle \rho_j \rangle_{j\in[k]}, \langle \mu_{i,\phi} \rangle_{i\in[s],\phi\in\Phi_i} \right\rangle \in \Psi$, such that ψ and the

[1] The logicians among the readers should recognize these as arity r_i relations.

density tensor ψ^{Π} of Π are equal. Namely, the following equalities hold: for all $j \in [k]$, $\rho_j^{\Pi} = \rho_j$; and for all $i \in [s]$ and $\phi \in \Phi_i$, $\mu_{i,\phi}^{\Pi} = \mu_{i,\phi}$.

We say that Π ϵ-*closely* satisfies Ψ if there exists $\psi \in \Psi$ such that: for all $j \in [k]$, $\rho_j^{\Pi} = \rho_j$; and for all $i \in [s]$ and $\phi \in \Phi_i$, $\mu_{i,\phi}^{\Pi} = \mu_{i,\phi} \pm \epsilon$. In addition, we say that Π ϵ-*approximately* satisfies Ψ if there exists $\psi \in \Psi$ such that: for all $j \in [k]$, $\rho_j^{\Pi} = \rho_j \pm \epsilon$; and for all $i \in [s]$ and $\phi \in \Phi_i$, $\mu_{i,\phi}^{\Pi} = \mu_{i,\phi} \pm \epsilon$.

By extension (and with a slight abuse of notation), we say that the hypergraph H itself *satisfies* the property Ψ if there exists a partition Π of H's vertices that satisfies Ψ, and similarly we say that H itself ϵ-*closely* (ϵ-*approximately*) satisfies the property Ψ if there exists a partition of H's vertices that ϵ-closely (ϵ-approximately) satisfies the property Ψ. In addition, we may refer to a specific density tensor ψ as the singleton set $\{\psi\}$, and accordingly as a property of partitions.

In the following we make some computational assumptions on the representation of the considered set Ψ, which may be infinite. In particular, we assume that given a density tensor ψ, computing whether ψ is close to some $\psi' \in \Psi$ can be done efficiently.

Definition 2.1 *For a set Ψ of density tensors, we say that Ψ* is checkable for proximity in time $O(t)$, *or shortly $TC(\Psi) = O(t)$, if there exists an algorithm that for any density tensor ψ and any $\epsilon > 0$ decides in time at most $O(t)$ whether the tensor ψ ϵ-approximately satisfies Ψ, and if so, outputs a density tensor $\psi_T \in \Psi$ which is ϵ-approximated by ψ.*

We say that a set Ψ of density tensors is efficiently checkable for proximity *if $TC(\Psi)$ is bounded by some polynomial in s and k^r.*

Note that the sets Ψ resulting from upper and lower bound constraints as in [16] are indeed efficiently checkable for proximity. For instance, given a density tensor $\psi = \left\langle \langle \rho_j \rangle_{j\in[k]}, \langle \mu_{i,\phi} \rangle_{i\in[s],\phi\in\Phi_i} \right\rangle$, one can verify in time $O(k + s \cdot k^r)$ whether ψ ϵ-approximately satisfies Ψ by going over the $k + s \cdot k^r$ values of the parameters of ψ, and checking if all of these values satisfy the lower and upper bounds within the allowed deviation of ϵ. It is easy to verify that the sets of density tensors defined in the proofs of Theorem 3 and Theorem 4 are also efficiently checkable for proximity.

Also, we assume throughout that a uniformly random choice of a vertex $v \in V$, as well as an edge query, can all be done in constant time. In the following we make no attempt to optimize the coefficients, only the function types.

We are now ready to state the main technical theorem of this paper.

Theorem 1 (Hypergraph Partitioning Algorithm) *Let $H = (V, E_1, \ldots, E_s)$ be a hypergraph with n vertices and s edge sets of maximal size r, let Ψ be an efficiently checkable set of density tensors, of partitions into k sets, and let $\epsilon > 0$ and $\delta > 0$ be fixed constants. There is a randomized algorithm A such that*

- *If H satisfies Ψ then with probability at least $1-\delta$ the algorithm A outputs a partition of $V(H)$ that ϵ-closely satisfies Ψ.*
- *For every H, the algorithm A outputs a partition of $V(H)$ that* does not *ϵ-closely satisfies Ψ with probability at most δ. In particular, if H does not ϵ-closely satisfy Ψ, then the algorithm returns "No Partition" with probability at least $1 - \delta$.*

Furthermore, the running time of A is bounded by $\log^2(\frac{1}{\delta}) \cdot \exp\left((\frac{r}{\epsilon})^{O(s\cdot r\cdot k^r)}\right) + n \cdot \mathrm{poly}(k^r, s, 1/\epsilon)$.

Observe that the above algorithm has only poly-logarithmic dependence on the success probability δ, a fact that will be important in the applications of Theorem 1 we discuss later.

Let us now discuss the property testing variant of Theorem 1. To this end, we define one additional measure of closeness to the property Ψ in the property testing manner. Let $H = (V, E_1, \dots E_s)$ and $H' = (V, E'_1, \dots E'_s)$ be two hypergraphs over the same vertex set V, with matching arities of edge sets (i.e. for all $i \in [s]$, $r_i = r'_i$). Let $\Delta(E_i, E'_i)$ denote the size of the symmetric difference between E_i and E'_i. The distance between H and H' is defined as $dist(H, H') = \frac{1}{s}\sum_{i\in[s]} \Delta(E_i, E'_i)/n^{r_i}$. The distance of a hypergraph H from the property Ψ is defined as $dist(H, \Psi) = \min_{H'}\{dist(H, H') : H'$ satisfies $\Psi\}$. For $\epsilon > 0$ we say that H is *ϵ-far* from satisfying the property Ψ when $dist(H, \Psi) > \epsilon$, and otherwise, H is *ϵ-close* to Ψ. The testing algorithm follows immediately from the following.

Theorem 2 (Testing) *Let $H = (V, E_1, \dots, E_s)$ be a hypergraph with n vertices and s edge sets of maximal size r, let Ψ be an efficiently checkable set of density tensors, of partitions into k sets, and let $\epsilon > 0$ and $\delta > 0$ be fixed constants. There is a randomized algorithm A_T such that*

- *If H satisfies Ψ then with probability at least $1 - \delta$ the algorithm A_T outputs* Accept*, and in addition provides a density tensor $\psi_T \in \Psi$ such that ψ_T is ϵ-closely satisfied by H.*
- *If H does not even ϵ-closely satisfy the property Ψ then with probability at least $1 - \delta$ the algorithm A_T outputs* Reject*.*

The query complexity of the algorithm A_T is bounded by $\log^2(\frac{1}{\delta}) \cdot \mathrm{poly}(k^r, s, 1/\epsilon)$, and its running time is bounded by $\log^2(\frac{1}{\delta}) \cdot \exp\left((\frac{r}{\epsilon})^{O(s\cdot r\cdot k^r)}\right)$.

To see why the algorithm above can be used as a testing algorithm in the traditional sense, just note that a hypergraph that ϵ/k^r-closely satisfies a property is clearly ϵ-close to it. Note that the GGR-algorithm [16] follows from Theorem 1 by setting $r = 2$ and $s = 1$ in the statement of the theorem. Similarly, the property testing algorithm of [16] follows easily from Theorem 2.

3 Immediate applications of the theorems

Just as the GGR-algorithm [16] and its testing variant can be used to give $O(n)$ time algorithms and testing algorithms for graph partition problems like Max-Cut, k-colorability and Max-Clique, so can Theorems 1 and 2 be used to give analogous algorithms for the corresponding problems in hypergraphs. For example, consider the following immediate corollary of Theorems 1 and 2.

Corollary 3.1 *For any fixed r and c, the following algorithms immediately follow from using Theorems 1 and 2 for the corresponding special cases:*

- *An ϵ-test of an n-vertex r-uniform (simple) hypergraph for the property of being c-colorable (with no monochromatic edges), making $\mathrm{poly}(1/\epsilon)$ queries and taking $\exp(\mathrm{poly}(1/\epsilon))$ time, and an accompanying approximate coloring algorithm taking $\exp(\mathrm{poly}(1/\epsilon)) + \mathrm{poly}(1/\epsilon)n$ time.*
- *An ϵ-test of an n-vertex r-uniform hypergraph for the property of having an independent set of size at least αn (for any fixed α), making $\mathrm{poly}(1/\epsilon)$ queries and taking $\exp(\mathrm{poly}(1/\epsilon))$ time, and an accompanying approximate algorithm taking $\exp(\mathrm{poly}(1/\epsilon)) + \mathrm{poly}(1/\epsilon)n$ time and finding a set of this size spanning less than ϵn^r edges.*

- *An algorithm for approximating the maximum r-way cut of an n-vertex r-uniform hypergraph up to an ϵn^r additive error, making* $\mathrm{poly}(1/\epsilon)$ *queries and taking* $\exp(\mathrm{poly}(1/\epsilon))$ *time, and an accompanying algorithm taking* $\exp(\mathrm{poly}(1/\epsilon)) + \mathrm{poly}(1/\epsilon)n$ *time for finding a cut witnessing the approximate maximum.*

We note that some of these results are not new. For example, a test for a maximum independent set was first obtained by Langberg [23], and a test for hypergraph c-colorability was first obtained by Czumaj and Sohler [7]. Moreover, the ad-hoc versions of the results of [7] and [23] have a better dependence on $1/\epsilon$ than the algorithms derivable from our method of obtaining general hypergraph partitions, although we could also improve our dependency by doing ad-hoc optimizations in the proof for the special cases. The main motivation for the above proposition is to show that our result can be thought of as a common generalization of many previous results.

3.1 Estimating the maximum number of satisfiable clauses in a k-CNF

Up to now it would have been enough to prove a version of our main theorem that would apply only to undirected hypergraphs. The next application, related to the work of [1], shows an instance in which the ordering of the edges is important.

Corollary 3.2 *For a fixed k, the following follows from using Theorems 1 and 2 for the appropriate setup: An algorithm for approximating the minimum number of unsatisfiable clauses in an n-variable k-CNF up to an ϵn^k additive error, making* $\mathrm{poly}(1/\epsilon)$ *queries and taking* $\exp(\mathrm{poly}(1/\epsilon))$ *time, and an accompanying algorithm taking* $\exp(\mathrm{poly}(1/\epsilon))+\mathrm{poly}(1/\epsilon)n$ *time for finding an assignment witnessing the approximate minimum.*

Proof. We need to use here $k+1$ "edge sets", $E_0, \ldots, E_k$, where E_i will correspond to all clauses for which exactly i of the k literals are negated. Moreover, we order each clause in E_i so that the i negated literals are first. We consider the n variables as "vertices", and seek a partition of them into two sets V_0 and V_1.

We now formulate the property Ψ_α, as the property that in the desired partition no more than a total of αn^k edges are in some E_i while their first i vertices are in V_1 and all their other vertices are in V_0. It is now not hard to see that a partition witnessing Ψ_α can be converted to an assignment leaving no more than αn^k clauses unsatisfied, by assigning to each variable x_i the value j_i for which the corresponding vertex v_i is in V_{j_i}.

Finally, to estimate the minimum number, we run our partitioning algorithm for any α which is an integer multiple of $\epsilon/2$, with approximation parameter $\epsilon/2$ and confidence parameter $\epsilon/6$ (so with probability at least $\frac{2}{3}$ we make no error on any run of the partitioning algorithm). ■

We note that here again the degree of our polynomial is worse than the one in [1], but also here we could have made it better with ad-hoc optimizations. Also, the original result of [1] refers to general k-CSP instances rather than k-CNF ones, but it is not hard to either reduce the former to the latter, or use a slightly more complicated instance of the partitioning problem.

4 Finding a regular partition of a graph

4.1 Background and statement

The most interesting application we have of Theorem 1 is a new algorithmic approach for constructing a regular partition of a graph (in the sense of Szemerédi [27]). This approach leads to an algorithm that improves upon the previous algorithms for this problem both in the running time (note that we make a tradeoff here by constructing a sublinear time *probabilistic* algorithm), and in the guarantee that a small regular partition will be found if one exists, with a running time that corresponds to the actual output size. Although this is a result on graphs, it cannot be derived from the graph version of Theorem 1 (i.e. the original GGR-algorithm) because a key feature of the algorithm is that it considers relations between more than two vertices of the graph.

The regularity lemma of Szemerédi [27] is one of the most important results in graph theory, as it guarantees that every graph has an ϵ-approximation of constant descriptive size, namely a size that depends only on ϵ and not on the size of the graph. This approximation "breaks" the graph into a constant number of pseudo-random bipartite graphs. This is very useful in many applications since dealing with random-like graphs is much easier than dealing with arbitrary graphs. For a comprehensive survey of the applications of the lemma, the interested reader is referred to [22].

We first state the lemma. Recall that for two nonempty disjoint vertex sets A and B of a graph G, we define $E(A,B)$ to be the set of edges of G between A and B. The *edge density* of the pair is defined by $d(A,B) = |E(A,B)|/(|A||B|)$. The original notion of regularity, to which we refer to as *subset regularity*, was defined based on an easy to observe property of random graphs.

Definition 4.1 (ϵ-subset-regular pair) *A pair (A,B) is* ϵ-subset-regular *if for every $A' \subseteq A$ and $B' \subseteq B$ satisfying $|A'| \geq \epsilon|A|$ and $|B'| \geq \epsilon|B|$, we have $d(A',B') = d(A,B) \pm \epsilon$.*

An ϵ-regular pair can be thought of as a pseudo-random bipartite graph in the sense that it behaves almost as we would expect from a random bipartite graph of the same density, see e.g. [10]. The intuition behind the above definition is that if (A,B) behaves like a random bipartite graph with edge density d, then all large enough sub-pairs should have similar densities.

A partition $V_1,\ldots,V_k$ of the vertex set of a graph is called an *equipartition* if $|V_i|$ and $|V_j|$ differ by no more than 1 for all $1 \leq i < j \leq k$ (so in particular every V_i has one of two possible sizes). The *order* of an equipartition denotes the number of partition classes (k above). An equipartition $V_1,\ldots,V_k$ of the vertex set of a graph is called *ϵ-subset-regular* if all but at most $\epsilon\binom{k}{2}$ of the pairs (V_i,V_j) are ϵ-subset-regular. Szemerédi's regularity lemma can be formulated as follows.

Lemma 4.2 ([27]) *For every $\epsilon > 0$ there exists $T = T_{4.2}(\epsilon)$, such that any graph with $n \geq 1/\epsilon$ vertices has an ϵ-subset-regular equipartition of order $\hat{k}$, where $1/\epsilon \leq \hat{k} \leq T$.*

It is far from surprising that a lemma that supplies an approximation of constant size for arbitrarily large graphs will also have algorithmic applications. The original proof of the regularity lemma was non-constructive, in the sense that it did not supply an efficient polynomial time algorithm for obtaining a regular partition of the graph. The first polynomial time algorithm for constructing such a partition of a graph was obtained by Alon et al. [2]. Additional algorithmic versions of the lemma were later obtained in [3, 10, 14, 15, 21].

The main observation used in most of the above was that although subset-regularity is computationally hard even to detect, it is essentially equivalent to an alternative definition that is based on a simple count of local structures.

Given (A, B), we denote by $d_{C_4}(A, B)$ the density of C_4 instances (cycles of size 4) between A and B, namely, the number of copies of C_4 whose vertices alternate between A and B, divided by their possible maximum number $\binom{|A|}{2}\binom{|B|}{2}$. [2]

Definition 4.3 (ϵ-locally-regular pair and partition) *A pair (A, B) of vertex sets is* ϵ-locally-regular *if $d_{C_4}(A, B) = d^4(A, B) \pm \epsilon$.*

An equipartition $V_1, \ldots, V_k$ of the vertex set of a graph is called ϵ-locally-regular *if all but at most $\epsilon\binom{k}{2}$ of the pairs (V_i, V_j) are ϵ-locally-regular.*

In the sequel we will refer to local-regularity simply as *regularity* whenever there is no ambiguity in the context. The main observation about this definition is that an $\epsilon^{O(1)}$-regular pair is also ϵ-subset-regular, and the other direction similarly holds. This was first proved in [2]. Our algorithm will center on finding partitions in which the pairs conform to local regularity.

The main drawback of the regularity lemma is that the bounds that the lemma guarantees have an enormous dependence on ϵ. More precisely, denote the tower of exponents function as $\text{Tower}(i) = 2^{\text{Tower}(i-1)} = 2^{\overset{\cdot^{\cdot^{\cdot^{2}}}}{\left.\right\} i}}$. Then the bounds on $T_{4.2}(\epsilon)$ are given by $\text{Tower}(1/\epsilon^5)$. Furthermore, Gowers [17] proved that these bounds are not far from the truth (in the qualitative sense), as he constructed a graph that has no ϵ-regular partition of size less than $\text{Tower}(1/\epsilon^{1/16})$.

Of course, this takes a huge toll on the dependency on ϵ of the running time of all the previous algorithms for constructing regular partitions. Also, as these algorithms essentially follow the original proof of Szemerédi's lemma (while substituting the newer and easier to check definition of regularity as in Definition 4.3), they are not guaranteed to run faster even if the graph admits a very small regular partition, as it may be overlooked considering the way the lemma was proved.

In contrast, the algorithm we design in this paper takes a different approach, and manages to directly check the graph (up to some tolerance) for the existence of a regular partition with a prescribed order k. Thus, a smaller regular partition, if one exists, will indeed be returned. For the same reason, the running time is also reduced. For a fixed ϵ, the running time will be linear in the size of the actual output (making this a sublinear time algorithm in the size of the input). The worst case running time will only occur if the graph is indeed a worst-case graph, such as the one constructed in [17]. Another added feature is that if we only care about the densities of the regular partition and not the partition itself, then there is a variant of the algorithm that takes constant time, assuming that a uniformly random choice of a vertex and an edge query can be done in constant time. The following is the new algorithmic version of Lemma 4.2.

Theorem 3 *There is a randomized algorithm, that given $\epsilon > 0$ and an n vertex graph G, that has a $\frac{1}{2}\epsilon$-regular partition of order $\hat{k} \geq 1/\epsilon$, [3] produces with high probability an ϵ-regular partition of G of order k, where $1/\epsilon \leq k \leq \hat{k}$.*

[2] When counting the number of C_4 instances between A and B we only consider the edges connecting A and B. Therefore, $0 \leq d_{C_4}(A, B) \leq 1$.

[3] The reason for the requirement that $\hat{k} \geq 1/\epsilon$ is that the same requirement is needed in Lemma 4.2. For the same reason we return a partition of size $k \geq 1/\epsilon$.

The expected running time of the algorithm is $O(\exp(\exp(O(\hat{k}^5)))+n\cdot\mathrm{poly}(\hat{k}))$. *Also, the densities of the partition can be found and output in time* $\exp(\exp(O(\hat{k}^5)))$, *independently of* n.

We stress that in Theorem 3, the algorithm does *not* receive the number $\hat{k}$ as part of the input. Also, the hidden constants in the running time are *absolute* and do not depend on ϵ or $\hat{k}$, and correspondingly there is no unconditional Tower-dependence on ϵ as in the previous algorithms. In addition, it is not hard to see that the constant $\frac{1}{2}$ in the statement can be replaced with any constant smaller than 1. Finally, although the algorithm does not know $\hat{k}$ in advance, its running time does depend, in a good sense, on $\hat{k}$. That is, if $\hat{k}$ is small then the running time of the algorithm will also be small, compared to the unconditional Tower-type dependence on ϵ of the previous algorithms.

Note that if one takes $\hat{k}$ in Theorem 3 to be the upper bound on the size of a regular partition that must exist in any graph, that is $\hat{k} = T_{4.2}(\epsilon) = \mathrm{Tower}(\mathrm{poly}(1/\epsilon))$, then we simply get a randomized algorithm for constructing a regular partition of a graph. An interesting feature of this algorithm is that its running time is $O(n)$ for any fixed ϵ. This algorithm is faster than the previous algorithms that ran in time $\Omega(n^2)$ (and higher). The price that we pay is that our algorithm is randomized while the previous algorithms were deterministic.

4.2 The proof idea

Our main idea is that instead of trying to reprove the regularity lemma "algorithmically" as in the previous approaches, we take Lemma 4.2 for a fact and just try to find the smallest regular partition that the graph has. Starting from $k = 1/\epsilon$ we try to find a regular partition of G of order k, and if none exists we move to $k+1$. Lemma 4.2 guarantees that we will eventually find a regular partition. To implement the above approach, we need an algorithm that will produce a regular partition of G of order k if one exists, and will not output any partition if none exists. Let's see how we can use Theorem 1 to obtain such an algorithm.

A key difference between a partition-instance in the sense of [16] and a regular partition is that in a partition-instance we only care about the density of each pair (V_i, V_j), while in a regular partition we care about the *distribution* of the edges between V_i and V_j given in terms of a sort of a second moment condition, that of Definition 4.3. The framework of the graph partition problems of [16] thus cannot by itself provide a check for regularity (unless "negations" of partition properties are checked as in [12] and in the hypergraph regularity algorithms here, but these again may lead to a small regular partition being overlooked). However, as we show below, a hypergraph partition theorem such as Theorem 1 is very useful for checking the regularity condition of Definition 4.3.

Given a graph G let us *implicitly* construct a hypergraph $H = H(G)$ on the vertex set $V(H) = V(G)$, that contains the 2-edges of G as well as 4-edges corresponding to the 4-cycles of G. This is not an undirected hypergraph, so we need at least some part of the additional generality provided in Theorem 1. Suppose that G has an ϵ-regular partition $V_1, \ldots, V_k$. Then, for any pair (V_i, V_j) which is ϵ-regular, Definition 4.3 says that there is a real d such that $d = d(A, B)$ and the number of 4-cycles spanned by (A, B) is $(d^4 \pm \epsilon)\binom{|V_i|}{2}\binom{|V_j|}{2}$. Hence, the same partition $V_1, \ldots, V_k$ when considered on H, would have the property that if (V_i, V_j) is regular (in G) with density d, then the density of 4-edges in H connecting a pair of vertices from V_i with a pair of vertices from V_j is $d^4 \pm \epsilon$.

Therefore, our algorithm for constructing a regular partition of G simply looks for a partition of H into k sets $V_1, \ldots, V_k$ such that most pairs (V_i, V_j) have the property that the density of 4-edges connecting them is close to $d^4(V_i, V_j)$. By the above discussion we know that if a graph G has an

ϵ-regular partition then H satisfies the partition-instance. One should be careful at this point, as Theorem 1 only guarantees that if the hypergraph satisfies the partition-instance then the algorithm will return a possibly *different* partition that is close to satisfying the partition-instance. Luckily, it is not difficult to see that if the algorithm returns a partition of H that is close to satisfying the above partition-instance, then it is in fact regular with slightly worse parameters.

We finally note that the above explanation describes the method of designing an algorithm whose running time is as stated with *high probability* and not in *expectation*. To maintain a low expected running time, without knowing the value of $\hat{k}$ (which may be very large), we need one final trick: As we go and try higher and higher values of k, we go back and try again the lower values, to get another chance at small partitions that may have been missed during the first try. We note that in this use of Theorem 1 we rely on the fact that the dependence on δ in Theorem 1 is only poly-logarithmic. Thus we execute the more costly iterations with sufficiently small probability relative to their cost.

4.3 Proof of Theorem 3

We first formally describe the reduction, through which we reduce the problem of finding a regular partition of a graph G to the problem of finding a partition of a hypergraph $H = H(G)$ satisfying certain density conditions. Given a graph G let us define the following hypergraph $H = H(G)$.

Definition 4.4 *For a graph G, the hypergraph $H = H(G)$ has the same vertex set as G, and two sets of edges, a set E of 2-edges and a set C of (ordered) 4-edges. $E(H)$ is made identical to $E(G)$ (if we insist on the ordered edge setting of Theorem 1, then for every $\{u, v\} \in E(G)$ we have both (u, v) and (v, u) in H). We set C to include all sequences (u_1, u_2, u_3, u_4) that form a 4-cycle in that order in G.*

We note that also C has symmetries due to redundancy, as for example $(u_1, u_2, u_3, u_4) \in C$ if and only if $(u_2, u_3, u_4, u_1) \in C$. If we want to keep a hypergraph-like structure without redundancy, then we should define $E(H)$ as a set of unoredered pairs (just like $E(G)$) and $C(H)$ as a set of unordered pairs of unordered pairs, where the cycle (u_1, u_2, u_3, u_4) would be represented by the pair of pairs $\{\{u_1, u_3\}, \{u_2, u_4\}\}$. This is the representation that we adopt from now on (moving back to the fully ordered representation would just entail adding a constant coefficient in Definition 4.5 below).

Let us note that while discussing regular partitions, we measure the density of edges and C_4 between sets V_i, V_j relative to the size of V_i and V_j, see for example Definitions 4.1 and 4.3. On the other hand, when discussing the partition problems related to Theorem 1, we considered the density of edges relative to the number of vertices in the entire graph. In order to keep the following discussion aligned with the definitions of Subsection 2.1, let us use the convention of Theorem 1 of considering densities relative to the number of vertices in the entire graph. Therefore, in our density tensors we set $\mu(i, j)$ as the number of edges in $E(H)$ between V_i and V_j divided by n^2 and set $\mu_{C_4}(i, j)$ as the number of directed edges in $C(H)$ consisting of a pair from V_i and a pair from V_j, divided by n^4. Our hypergraph partition property is now defined as the following.

Definition 4.5 *Let $\Psi(k, \epsilon)$ denote the set of density tensors for partitions $(V_1, \ldots, V_k)$ which are equipartitions, and in addition for at least a $1 - \epsilon$ fraction of the pairs $1 \leq i < j \leq k$ we have $\mu_{C_4}(i, j) = \frac{1}{4}k^4\mu^4(i, j) \pm \frac{\epsilon}{4k^4}$.*

It is not hard to see that the above Ψ is efficiently checkable for proximity as per Definition 7.1, as required for Theorem 1 and Theorem 2.

We claim that a partition satisfying Ψ over H indeed satisfies the (local) regularity condition over G. For simplicity, we ignore additive $O(\frac{k}{n})$ factors all throughout the following.

Claim 4.6 *Given a graph G, let $H = H(G)$ be the hypergraph defined above. Then, a partition of $V(G)$ into sets $V_1, \ldots, V_k$ is ϵ-regular if and only if the same partition of $V(H)$ satisfies $\Psi(k, \epsilon)$. Also, if a partition of $V(H)$ $\frac{\epsilon}{16k^4}$-closely satisfies $\Psi(k, \frac{1}{2}\epsilon)$ then the same partition of $V(G)$ is ϵ-regular.*

Proof: For two disjoint vertex sets W_1, W_2, Let us define by $E(W_1, W_2)$ the set of edges of $E(H)$ from W_1 to W_2, and by $C_4(W_1, W_2)$ the set of 4-edges of $C(H)$ of type $\{\{u_1, u_2\}, \{v_1, v_2\}\}$ with $u_1, u_2 \in V_i$ and $v_1, v_2 \in V_j$. It is now not hard to see that (up to $O(\frac{k}{n})$ additive factors) we have $d(V_i, V_j) = \frac{|E(V_i,V_j)|}{(n/k)^2} = k^2\mu(i,j)$, and similarly $d_{C_4}(V_i, V_j) = \frac{|C(V_i,V_j)|}{\binom{n/k}{2}^2} = 4k^4\mu_{C_4}(i,j)$. Now, the ϵ-regularity condition of Definition 4.3 requires that $d_{C_4}(V_i, V_j) = d(V_i, V_j)^4 \pm \epsilon$. By the above this is equivalent to the condition $\mu_{C_4}(i,j) = \frac{1}{4}k^4\mu^4(i,j) \pm \frac{\epsilon}{4k^4}$, as needed.

As for the second assertion of the lemma, note that $\Psi(k, \frac{1}{2}\epsilon)$ requires that for all but at most $\frac{1}{2}\epsilon\binom{k}{2}$ of the pairs (V_i, V_j) we have $\mu_{C_4}(i,j) = \frac{1}{4}k^4\mu^4(i,j) \pm \frac{\epsilon}{8k^4}$. If a partition $\frac{\epsilon}{16k^4}$-closely satisfies this condition, it means that for all these pairs (V_i, V_j) we have $\mu_{C_4}(i,j) = \frac{1}{4}k^4\mu^4(i,j) \pm \frac{\epsilon}{4k^4}$, which means by the above discussion that this partition is ϵ-regular. ■

The algorithm: We only prove the version that provides the actual partition, as proving the version that provides densities in constant running time is almost word-for-word identical, only instead of Theorem 1 one would use Theorem 2 respectively. We start by describing a version of the algorithm that will run in the stated time with high probability (say, 3/4) rather than in expectation. We will later add one more idea that will give the required bound on the expectation. Given a graph G and a parameter $\epsilon > 0$, our goal is to produce an ϵ-regular partition of G of size at least $1/\epsilon$ and at most $\hat{k}$, assuming that G has a $\frac{1}{2}\epsilon$-regular partition of size at most $\hat{k}$ (remember that the algorithm does not receive $\hat{k}$ as part of the input). Starting from $k = 1/\epsilon$, we execute the hypergraph partitioning algorithm on the hypergraph $H = H(G)$ that was described at the beginning of this section with the partition instance $\Psi(k, \frac{1}{2}\epsilon)$, with success probability δ_k (that will be specified later) and with approximation parameter $\frac{\epsilon}{16k^4}$. Note that each call to the algorithm of Theorem 1 is done with a different value of k. Let us name the step where we call the partition algorithm with partition-instance $\Psi(k, \frac{1}{2}\epsilon)$ the k^{th} *iteration* of the algorithm.

A crucial point here is that we *do not* explicitly construct H as that may require time $\Theta(n^4)$. Rather, whenever the hypergraph partition algorithm of Theorem 1 asks whether a pair of vertices $\{v_1, v_2\}$ is connected to another pair of vertices $\{u_1, u_2\}$, we just answer by inspecting the four corresponding edges of G (in constant time). If for some integer k the hypergraph partition algorithm returns a partition of $V(H)$, then we return the same partition for $V(G)$. Otherwise, we move to the next integer $k + 1$. If we reached $k = T_{4.2}(\frac{1}{2}\epsilon) = \text{Tower}(\text{poly}(1/\epsilon))$ we stop and return "fail".[4]

[4]The choice of the maximum k is because by Lemma 4.2 we know that any graph has a $\frac{1}{2}\epsilon$-regular partition of size at most $T_{4.2}(\frac{1}{2}\epsilon)$. Therefore, if we reached that value of k then we know that we made a mistake on the way.

Correctness: Let us now prove the correctness of the algorithm. As above we denote by δ_k the error probability with which we apply the partition algorithm of Theorem 1 at the k^{th} iteration. Remember that the algorithm does not know r in advance, and it may be the case that r is as large as Tower$(1/\epsilon)$ (due to the lower bound of Gowers [17]). Therefore, one way to resolve this is to take each δ_k to be $1/\text{Tower}(1/\epsilon)$. However, that would mean that the running time of the partition algorithm would be Tower$(1/\epsilon)$ even if r is small (as the running time depends on δ_k).

A more economic way around this problem is to take $\delta_k = \frac{1}{4} \cdot 2^{-k}$. This way the probability of making an error when considering all possible values of k is bounded by $\sum_{k=1}^{\infty} \frac{1}{4} \cdot 2^{-k} \leq \frac{1}{4}$. Note that this in particular means that with high probability we will not return a partition of G that does not $\frac{\epsilon}{16k^4}$-closely satisfy $\Psi(k, \frac{1}{2}\epsilon)$. Combined with the second assertion of Claim 4.6 we get that with high probability the algorithm returns only partitions that are ϵ-regular.

We thus only have to show that if G has a $\frac{1}{2}\epsilon$-regular partition of size $\hat{k}$, then the algorithm will find an ϵ-regular partition of G of size at most $\hat{k}$. Suppose then that G has such a partition of size $\hat{k}$. We show that with high probability the algorithm will find such a partition during the $\hat{k}^{th}$ iteration (of course it may be the case that it will find a smaller partition during one of the previous iterations). Let $H = H(G)$ be the hypergraph defined above. By Claim 4.6 we know that as G has a $\frac{1}{2}\epsilon$-regular partition of size $\hat{k}$, H satisfies $\Psi(\hat{k}, \frac{1}{2}\epsilon)$. Therefore, with probability at least $1 - \delta_{\hat{k}} \geq 3/4$, the partition algorithm will return a partition of H that $\frac{\epsilon}{16\hat{k}^4}$-closely satisfies $\Psi(\hat{k}, \frac{1}{2}\epsilon)$. By the second assertion of Claim 4.6 we know that such a partition is ϵ-regular.

Running time with high probability: Let us bound the running time of the algorithm. Since with high probability the hypergraph partition algorithm will generate an ϵ-regular partition of G when reaching the $\hat{k}^{th}$ iteration (or earlier), we get that with high probability the algorithm will terminate after at most $\hat{k}$ iterations. When executing the algorithm of Theorem 1 on $\Psi(\hat{k}, \frac{1}{2}\epsilon)$ we need to set $s = 1$, $r = 4$, $\delta = \delta_k = \frac{1}{4} \cdot 2^{-k}$. The number of partitions is k while the proximity parameter should be $\frac{\epsilon}{16k^4} \geq \frac{1}{16k^5}$ (remember that we start with $k = 1/\epsilon$). It follows from Theorem 1 that the running time of the hypergraph partition algorithm in the k^{th} iteration is $O(2^{2^{O(k^5)}} + n \cdot \text{poly}(k))$. As the running time of k iterations is clearly dominated by the running time of the k^{th} one, we get that with probability at least $\frac{3}{4}$ both the answer will be correct and the running time will be bounded by $O(2^{2^{O(\hat{k}^5)}} + n \cdot \text{poly}(\hat{k}))$.

Bounding the expected running time: The version of the algorithm described above runs in time $O(2^{2^{O(\hat{k}^5)}} + n \cdot \text{poly}(\hat{k}))$ with high probability, but this is not its expectation. The reason is that the error of not finding one small existing regular partition could be very costly, as it could be followed with many iterations of searching in vain for higher values of k (remember that the larger k is, the more costly it is to execute the algorithm of Theorem 1). Suppose then that we partition the execution of the algorithm into *phases*, where in the k^{th} phase we execute the algorithm described above from the first iteration till iteration k, only now we use $\delta_k = 2^{-2^{b \cdot k^6}}$ for *all* k iterations, with b a constant to be chosen.

The modified algorithm clearly has a larger probability of outputting the required regular partition and a smaller probability of returning a wrong partition (because $2^{-2^{b \cdot k^6}} \leq k^{-1}2^{-k}$). Let us compute the expected running time, which we bound using $\sum_{k=m}^{\infty} p_k \cdot t_k$ where p_k is the probability of the algorithm performing the k^{th} phase, and t_k is the "cost" of the k^{th} phase, that is the running time

of that phase. Similarly to what we have discussed above, the time it takes to execute the iterations $1,\ldots,k$ of the original algorithm, even with the new δ_k, is bounded by $O(2^{2^{O(k^5)}} + n \cdot \mathrm{poly}(k))$. The contribution of the first $\hat{k}$ phases is clearly dominated by this expression for $k = \hat{k}$.

Now, let us focus on p_k for some $k > \hat{k}$. Suppose that the graph has a $\frac{1}{2}\epsilon$-regular partition of size $\hat{k}$. To reach phase k for some $k > \hat{k}$, the algorithm must have in particular failed the attempt made in phase $k-1$ to find a partition of size $\hat{k}$. Remember that the failure probability of that attempt is at most δ_{k-1}. Therefore, the probability of reaching phase k is at most $\delta_{k-1} = 2^{-2^{b\cdot(k-1)^6}}$. Hence, the total time expectancy for this algorithm (where we set b to be a large enough constant) is bounded by

$$\begin{aligned}
\sum_{k=1/\epsilon}^{\infty} p_k \cdot t_k &= \sum_{k=1/\epsilon}^{\infty} p_k \cdot O(2^{2^{O(k^5)}} + n \cdot \mathrm{poly}(k)) \\
&= O(2^{2^{O(\hat{k}^5)}} + n \cdot \mathrm{poly}(\hat{k})) + \sum_{k=\hat{k}+1}^{\infty} 2^{-2^{b\cdot(k-1)^6}} \cdot O(2^{2^{O(k^5)}} + n \cdot \mathrm{poly}(k)) \\
&= O(2^{2^{O(\hat{k}^5)}} + n \cdot \mathrm{poly}(\hat{k})) + O(n) \\
&= O(2^{2^{O(\hat{k}^5)}} + n \cdot \mathrm{poly}(\hat{k}))\ .
\end{aligned}$$

5 Regular partition of a hypergraph

Another main application of Theorem 1 is finding regular partitions of r-uniform hypergraphs. [5] Just like the algorithms for graph regularity have many applications for graph problems, the algorithms for hypergraph regularity have applications for hypergraph problems.

Hypergraph regularity has more than one version. The version we investigate here is the one discussed e.g. in [15] (the "vertex partition" version), which is not strong enough for some applications such as proving Szemerédi's Theorem on r-term arithmetic progressions [26], but still has many applications. For example, it was used in [15] in order to obtain additive approximations for all Max-SNP problems. This regularity is defined in an analog manner to the definition of ϵ-subset-regularity for graphs. The following is a quick rundown of the relevant definitions.

Definition 5.1 *For an r-uniform hypergraph H and an r-tuple $(U_1,\ldots,U_r)$ of vertex sets of H, the* edge density $d(U_1,\ldots,U_r)$ *is defined by the number of edges with one vertex from every U_i, divided by $\prod_{i=1}^{r}|U_i|$. An r-tuple $(U_1,\ldots,U_r)$ as above is called* ϵ-regular*, if every r-tuple $U'_1,\ldots,U'_r$ such that $U'_i \subseteq U_i$ and $|U'_i| \geq \epsilon|U_i|$ satisfies $d(U'_1,\ldots,U'_r) = d(U_1,\ldots,U_r) \pm \epsilon$.*

Finally, an equipartition $V_1,\ldots,V_k$ of the vertices of H is called ϵ-regular *if for all but at most ϵ of the possible r-tuples $1 \leq i_1 < i_2 < \cdots < i_r \leq k$, we have that $(V_{i_1},\ldots,V_{i_r})$ is ϵ-regular.*

As is the case for graphs, an ϵ-regular partition of an r-uniform hypergraph into a bounded number of sets always exists. However, obtaining an algorithmic version of the hypergraph version

[5] A hypergraph $H = (V,E)$ is r-uniform if all its edges have exactly r distinct vertices of V. These edges are *unordered*, that is, they are just subsets of $V(H)$ of size r. Hence a simple graph is just a 2-uniform hypergraph.

of the regularity lemma turns out to be more involved than the graph case, and in particular here we can no longer guarantee to find a small regular partition if one exists.

Theorem 4 *For any fixed r and $\epsilon > 0$, there exists an $O(n)$ time probabilistic algorithm that finds an ϵ-regular partition of an r-uniform hypergraph with n vertices. Moreover, if we only want to find the densities of an ϵ-regular partition, then there exists an algorithm obtaining them in constant time.*

The improvement over previous results that comes from the linearity of the algorithm in its output becomes more apparent here: While the previous algorithms (see, for example, [8, 15]) for constructing a regular partition of an r-uniform hypergraph have running time $O(n^{2r-1})$ (note that this is close to being quadratic in the input size), our algorithm has running time $O(n)$ for any r. Like the previous algorithms, and unlike the algorithm given in Theorem 3, the algorithm in Theorem 4 still has a tower-type dependence on ϵ.

Unlike the algorithm for graph regularity from the previous section that directly finds a regular partition, the algorithm of Theorem 4 makes use of some aspects of the iterative procedure for proving the existence of a regular partition, which repeatedly refines a partition of the hypergraph until a regular one is obtained. A direct implementation of such a procedure would suffer from NP-Completeness issues, and even if this is resolved in the style of previous works it would still take at least $\Omega(n^r)$ time as all deterministic algorithms have to (and in the previous works this actually takes longer). The crux of our proof is to apply an idea from [12] along with Theorem 1 in order to implement (an aspect of) the iterative procedure in time $O(n)$.

5.1 Proof sketch for Theorem 4

The main result in [12] required a way to quickly find the densities of a regular partition of a graph.[6] While the full result there does not seem translatable to our current body on knowledge on hypergraphs, the part about finding a regular partition is indeed translatable to a proof of Theorem 4. In the following we describe how to adapt the appropriate arguments from [12] to hypergraph regularity.

Equipartitions and the index function: In all that follows, we consider an r-uniform simple and unordered hypergraph H with a vertex set V, and we assume throughout that $|V|$ is large enough as a function of the other parameters (for a smaller $|V|$, we can just do an exhaustive search). An *equipartition* $V_1, \ldots, V_k$ of the vertex set V is defined as before. The main tool in proving regularity is the following numeric function, that in some sense measures how much the densities of the r-tuples in the partition vary.

Definition 5.2 *For an r-tuple of vertex sets $(W_1, \ldots, W_r)$ of an r-uniform hypergraph G, we define the* density $d(W_1, \ldots, W_r)$ *of the r-tuple as the number of edges with one vertex from every W_i, divided by $\prod_{j=1}^{r} |W_j|$.*

For an equipartition $\mathcal{A} = (V_1, \ldots, V_k)$ of an r-uniform hypergraph G into l sets, its index $\mathrm{ind}(\mathcal{A})$ *is defined as $k^{-r} \sum_{1 \le i_1 < i_2 < \cdots < i_r \le k} (d(V_{i_1}, \ldots, V_{i_r}))^2$.*

The index of any equipartition is clearly always between 0 and 1. One immediate but useful observation is the continuity of the index function with respect to the densities.

[6] In fact, in [12] a partition conforming to a strengthened notion of regularity was required.

Claim 5.3 *If an equipartition $\mathcal{A} = (V_1, \ldots, V_k)$ of G and an equipartition $\mathcal{A}' = (V'_1, \ldots, V'_k)$ of G' satisfy $|d(V_{i_1}, \ldots, V_{i_r}) - d(V'_{i_1}, \ldots, V'_{i_r})| \leq \epsilon$ for all $1 \leq i_1 < i_2 < \cdots < i_r \leq k$, then the respective indexes satisfy $|\mathrm{ind}(\mathcal{A}) - \mathrm{ind}(\mathcal{A}')| \leq 4\epsilon$.*

Robustness, finality, and regularity: The main technical observation of Szemerédi in [27] is that non-regular partitions can be refined in a way that substantially increases their index, while (because the index is bounded by 1) there must be partitions which cannot be refined that way. This observation carries to hypergraphs. Let us state this formally.

Definition 5.4 *A* refinement *of an equipartition $\mathcal{A} = (V_1, \ldots, V_k)$ is an equipartition $\mathcal{B} = (W_1, \ldots, W_l)$ such that every set W_j is fully contained in some V_{i_j}.*

Given $\delta > 0$ and a function $f : N \to N$, we say that an equipartition $\mathcal{A} = (V_1, \ldots, V_k)$ is (δ, f)-robust *if there exists no refinement $\mathcal{B} = (W_1, \ldots, W_l)$ for which $k \leq l \leq f(k)$ and $\mathrm{ind}(\mathcal{B}) \geq \mathrm{ind}(\mathcal{A}) + \delta$. The equipartition $\mathcal{A} = (V_1, \ldots, V_k)$ is called* (δ, f)-final *if there exists no $\mathcal{B} = (W_1, \ldots, W_l)$ as above regardless of whether it is a refinement of $\mathcal{A}$ or not.*

It is immediate to see that any (δ, f)-final partition is also (δ, f)-robust. We work with the definition of a final partition because it is somewhat easier to handle in the framework of Theorem 1. By using the fact that the index is always bounded between 0 and 1, it is not hard to see the following.

Claim 5.5 *For every $\delta > 0$, t and $f : N \to N$ there exists $T = T_{5.5}(\delta, f, t)$ such that every graph G with $n > T$ vertices has an equipartition $\mathcal{A} = (V_1, \ldots, V_k)$ with $t \leq k \leq T$ which is (δ, f)-final.*

The main technical step in [27] is using the "defect form" of the Cauchy-Schwartz inequality to derive a connection between robustness and regularity. The proof is based on refining the equipartition according to the algebra generated by the regularity counter examples for the irregular r-tuples, and works almost word for word for hypergraphs. We will not reproduce it in this version as it has been proved in several previous papers, see e.g. [6] and [8].

Lemma 5.6 *For every r and ϵ there exist $\delta = \delta_{5.6}(r, \epsilon)$ and $f(k) = f_{5.6}^{(r,\epsilon)}(k)$ so that any equipartition of an r-uniform hypergraph that is (δ, f)-robust is also ϵ-regular.*

The above lemma immediately suggests a way to generate a regular partition. Starting from an arbitrary partition, we can repeatedly refine the partition until a regular one is obtained. Indeed, that is the main idea in the original proofs of Szemerédi's theorem in graphs and hypergraph. The first problem with implementing this strategy algorithmically is that it is not clear how to efficiently detect if an r-tuple is not regular. This problem, however, can be overcome (see e.g. [2] and [8]). The more relevant problem to our investigation here is that just calculating edge counts for a partition of the hypergraph already takes time $\Theta(n^r)$, and our goal is an $O(n)$ time algorithm. As we explain in the sequel, if we are looking for a partition satisfying a somewhat stronger condition than regularity, then one such partition can be produced in $O(n)$.

Finding a regular partition: The proof of Theorem 4 now manifests itself as checking a sequence of density tensor sets that each describes a partition into k sets which is (δ, f)-final for the parameters we obtain from Lemma 5.6. However, finality (or robustness) by itself cannot be described by a set of density tensors. What we can do is test for density tensors which conform to a possible value of the index function, and then try to verify that an equipartition is final by testing for finer partitions with somewhat higher indexes and *rejecting* our equipartition if they exist. Claim 5.3 allows us to use our approximate algorithms, while ensuring that the "negation" procedure will also work (i.e. that for an equipartition that is not final we will indeed detect this).

For an integer k and $0 \leq \alpha \leq 1$, we let $\Psi^{(k,\alpha)}$ be the set of all density tensors of possible equipartitions into k sets whose index is at least α. We set $\delta = \delta_{5.6}(r, \epsilon)$ and $f(k) = f_{5.6}^{(r,\epsilon)}(k)$, and now for every $1/\epsilon \leq k \leq f(T_{5.5}(\frac{1}{3}\delta, f, 1/\epsilon))$ and every $\alpha \in \{0, \frac{1}{6}\delta, \frac{2}{6}\delta, \ldots, 1\}$ we run either the algorithm of Theorem 1 or the algorithm of Theorem 2 (depending on whether we want to find the actual partition or just the densities) on $\Psi^{(k,\alpha)}$, with approximation parameter $\frac{1}{24}k^{-r}\delta$, and with success probability sufficient to ensure that with probability at least $\frac{2}{3}$ none of the iterations provides an erroneous answer.[7]

We are now interested in finding k_0 and $\alpha_0 = \frac{\ell}{6}\delta$ for which we received a positive answer for $\Psi^{(k_0,\alpha_0)}$ and received a negative answer for all $\Psi^{(k,\alpha_0+2\delta/3)}$ for $k_0 \leq k \leq f(k_0)$. To conclude the proof, it is enough to convince ourselves that the following observations hold assuming that none of our iterations has provided an erroneous answer.

- For every k we can approximate the maximum index of an equipartition into k parts up to an error of $\frac{1}{6}\delta$. The lower bound follows directly from our procedure, while the upper bound follows from Claim 5.3
- A k_0 as above is obtained. This is because the k' for which there is a $(\frac{1}{3}\delta, f)$-final partition into k' sets (which exists by Claim 5.5) will in particular be detected due to the first item above (this does not mean that we will necessarily obtain $k_0 = k'$).
- For the k_0 that is obtained, the corresponding witnessing equipartition is (δ, f)-final and hence ϵ-regular. This is again due to the bounds of the first item above on the error in our index estimates.

The last two items above imply that we can then (by extracting the required data about the partition witnessing a maximum index for k_0) find our regular partition.

6 Overview of the proof of Theorem 1 and Theorem 2

Before going to the formal proof of Theorem 1 and Theorem 2 we briefly describe the general idea of how it is done.

The outermost layer of the proof borrows from the proof of [16] for graph partitions. Assuming that the hypergraph admits a partition $\Pi = \{V_1, \ldots, V_k\}$ of the vertices with the desired densities, we first split V into $\ell = O(1/\epsilon)$ parts $Y^1, Y^2, \ldots, Y^\ell$ of equal size. Then, for every part Y^i, a sample set $U \subset V \setminus Y^i$ is chosen at random with the hope of obtaining sufficiently many vertices

[7] The additional k^{-r} factor in the approximation parameter is because of the difference in the normalization between the definition of the density tensors in Ψ and the normalization of the density measure $d(W_1, \ldots, W_r)$.

from every V_j. Assume for now that we know the intersection of U with Π, i.e. the partition $\Pi_U = \{U \cap V_1, \ldots, U \cap V_k\}$. Then we try to reconstruct Π from its intersection with U as follows. First, the vertices in Y^i are clustered into a finite number of clusters, according to their degrees in the various components of Π_U (the degree of v is the number of hyperedges in which v participates, counted for all sets of edges and all possible configurations with respect to Π_U). Intuitively, each cluster groups together the vertices that look similar with respect to U and Π_U. Assume in addition that for every cluster $C \subset Y^i$ of similar vertices we also know their approximate distribution in the components of Π, i.e. we know some approximate $\bar{\beta} = \{\beta_1, \ldots, \beta_k\}$, where for every j, $\beta_j \approx \frac{|C \cap V_j|}{|C|}$. Then we partition every cluster of Y^i into the k components in a way that preserves the normalized intersection sizes, according to the corresponding $\bar{\beta}$.

Since we do not know the intersection Π_U, we simply try every possible partition of U. Similarly for the second assumption, since we do not know the intersection sizes for each cluster, we try all possibilities (up to some allowed deviation). For every such combination we get a different partition of V, and we test (by sampling) if one of them would give a reconstruction with the required densities.

The motivation for this kind of reconstruction is based on the following central observation. Suppose that $Y \subset V$ is a small enough set (but still linear in n) containing vertices that have similar degrees in the various V_i's. Assume that $\Pi' = \{V'_1, \ldots, V'_k\}$ is a new partition of V that is constructed from Π by redistributing the vertices of Y arbitrarily, so long as the component sizes are almost preserved (i.e. for all i, $|V'_i \cap Y| \approx |V_i \cap Y|$). Then the densities of the partition Π' are almost similar to the densities of Π. We should also note that, like in [16], it is not possible to classify all vertices of V at once while maintaining a small approximation error, so every Y^i is classified according to a partition of a different sample set U^i.

Although the overall strategy of the algorithm is similar to the GGR-algorithm, there are several differences between our analysis and the original one in [16]. One of the reasons is that there are many ways in which an edge with r vertices can intersect the k vertex sets, complicating the procedure of classifying the vertices according to the edges they have with the sets of the partition. This reflects on which statistics we keep track of, and also on how we obtain an approximation thereof. Another reason is that Theorem 1 can have as an input a general set of density tensors, rather than intervals of allowed densities as in the GGR-algorithm [16]. This extension of the GGR formalism allows us to use the algorithm more easily and efficiently in our applications.

7 The proof of Theorem 1 and Theorem 2

7.1 Definitions

We start by recalling some of the definitions from Section 2, and introducing a few new ones. We consider directed hypergraphs $H = (V, E_1, E_2, \ldots, E_s)$ with n vertices and s directed edge sets. Every edge set $E_i \subseteq V^{r_i}$ is a set of ordered r_i-tuples (ordered sets with possible repetitions) over V. Let $r = \max_i\{r_i\}$.

Let $\Pi = \{V_1^\Pi, V_2^\Pi, \ldots, V_k^\Pi\}$ be a partition of V. For every $i \in [s]$ we denote by Φ_i the set of all possible mappings $\phi : [r_i] \to [k]$. Each $\phi \in \Phi_i$ maps the vertices of an r_i-tuple to the components of Π. We denote by $E_{i,\phi}^\Pi \subseteq E_i$ the following set of r_i-tuples:

$$E_{i,\phi}^\Pi = \{(v_1, \ldots, v_{r_i}) \in E_i : \forall j \; v_j \in V_{\phi(j)}^\Pi\}$$

The *density tensor* of Π is the sequence of reals between 0 and 1 specifying the normalized sizes of $|V_i^\Pi|$ and $|E_{i,\phi}^\Pi|$. Namely, $\psi^\Pi = \left\langle \langle \rho_j^\Pi \rangle_{j\in[k]}, \langle \mu_{i,\phi}^\Pi \rangle_{i\in[s],\phi\in\Phi_i} \right\rangle$ is the density tensor of Π where $\rho_j^\Pi \triangleq \frac{1}{n} \cdot |V_j^\Pi|$ and $\mu_{i,\phi}^\Pi \triangleq \frac{1}{n^{r_i}} \cdot |E_{i,\phi}^\Pi|$.

For a fixed hypergraph H, a set Ψ of general density tensors (with respect to k, s and $r_1, \ldots, r_s$) defines a property of the partitions of V as follows. We say that a partition Π of V (and accordingly the density tensor ψ^Π) *satisfies* Ψ if there exists a density tensor $\psi = \left\langle \langle \rho_j \rangle_{j\in[k]}, \langle \mu_{i,\phi} \rangle_{i\in[s],\phi\in\Phi_i} \right\rangle \in \Psi$, such that ψ and the density tensor ψ^Π of Π are equal. Namely, the following equalities hold.

- for all $j \in [k]$, $\quad \rho_j^\Pi = \rho_j$
- for all $i \in [s]$ and $\phi \in \Phi_i$, $\quad \mu_{i,\phi}^\Pi = \mu_{i,\phi}$

We say that Π ϵ-*closely* satisfies Ψ if there exists $\psi \in \Psi$ such that

- for all $j \in [k]$, $\quad \rho_j^\Pi = \rho_j$
- for all $i \in [s]$ and $\phi \in \Phi_i$, $\quad \mu_{i,\phi}^\Pi = \mu_{i,\phi} \pm \epsilon$

In addition, we say that Π ϵ-*approximately* satisfies Ψ if for some $\psi \in \Psi$ the following holds.

- for all $j \in [k]$, $\quad \rho_j^\Pi = \rho_j \pm \epsilon$
- for all $i \in [s]$ and $\phi \in \Phi_i$, $\quad \mu_{i,\phi}^\Pi = \mu_{i,\phi} \pm \epsilon$

We extend the above notions to hypergraphs (by the existence of a corresponding partition) after Section 2, and accordingly we refer to Ψ as a property of hypergraphs too.

As we mentioned earlier, we consider only the sets Ψ for which the proximity of any density tensor ψ can be checked efficiently. The following definition is a formalization of what we alluded to in Section 2.

Definition 7.1 *For a set Ψ of density tensors, we say that Ψ* is checkable for proximity in time t*, or shortly $TC(\Psi) = t$, if there exists an algorithm that for any density tensor ψ and any $\epsilon > 0$ decides in time at most t whether the tensor ψ ϵ-approximately satisfies Ψ, and if so, outputs a density tensor $\psi_T \in \Psi$ which is ϵ-approximated by ψ.*

We say that a set Ψ of density tensors is efficiently checkable for proximity *if $TC(\Psi)$ is bounded by some polynomial in s and k^r.*

Note that the sets Ψ resulting from upper and lower bound constraints as in [16] are indeed efficiently checkable for proximity. For instance, given a density tensor $\psi = \left\langle \langle \rho_j \rangle_{j\in[k]}, \langle \mu_{i,\phi} \rangle_{i\in[s],\phi\in\Phi_i} \right\rangle$, one can verify in time $O(k + s \cdot k^r)$ whether ψ ϵ-approximately satisfies Ψ by going over the $k + s \cdot k^r$ values of the parameters of ψ, and checking if all of these values satisfy the lower and upper bounds within the allowed deviation of ϵ. It is easy to verify that the sets of density tensors defined in the proofs of Theorem 3 and Theorem 4 are also efficiently checkable for proximity.

7.2 Restating and proving the main theorems

We restate here Theorem 1 and Theorem 2, and then move to their proofs. We remind that we make no attempt here to optimize constants and coefficients, but only the function types, e.g. polynomial versus exponential (See however Subsection 7.4.6 below for a sketch of some optimizations tailored for special cases such as hypergraph colorability).

Theorem 1 (Hypergraph Partitioning Algorithm) *Let $H = (V, E_1, \ldots, E_s)$ be a hypergraph with n vertices and s edge sets of maximal size r, let Ψ be an efficiently checkable set of density tensors, of partitions into k sets, and let $\epsilon > 0$ and $\delta > 0$ be fixed constants. There is a randomized algorithm A such that*

- *If H satisfies Ψ then with probability at least $1-\delta$ the algorithm A outputs a partition of $V(H)$ that ϵ-closely satisfies Ψ.*
- *For every H, the algorithm A outputs a partition of $V(H)$ that* does not *ϵ-closely satisfies Ψ with probability at most δ. In particular, if H does not ϵ-closely satisfy Ψ, then the algorithm returns "No Partition" with probability at least $1-\delta$.*

Furthermore, the running time of A is bounded by $\log^2(\frac{1}{\delta}) \cdot \exp\left((\frac{r}{\epsilon})^{O(s \cdot r \cdot k^r)}\right) + n \cdot \mathrm{poly}(k^r, s, 1/\epsilon)$.

Theorem 2 (Testing Algorithm) *Let $H = (V, E_1, \ldots, E_s)$ be a hypergraph with n vertices and s edge sets of maximal size r, let Ψ be an efficiently checkable set of density tensors, of partitions into k sets, and let $\epsilon > 0$ and $\delta > 0$ be fixed constants. There is a randomized algorithm A_T such that*

- *If H satisfies Ψ then with probability at least $1-\delta$ the algorithm A_T outputs* Accept*, and in addition provides a density tensor $\psi_T \in \Psi$ such that ψ_T is ϵ-closely satisfied by H.*
- *If H does not even ϵ-closely satisfy the property Ψ then with probability at least $1-\delta$ the algorithm A_T outputs* Reject*.*

The query complexity of the algorithm A_T is bounded by $\log^2(\frac{1}{\delta}) \cdot \mathrm{poly}(k^r, s, 1/\epsilon)$, and its running time is bounded by $\log^2(\frac{1}{\delta}) \cdot \exp\left((\frac{r}{\epsilon})^{O(s \cdot r \cdot k^r)}\right)$.

7.2.1 Proof of Theorem 1 and Theorem 2

First we define the notion of partitioning oracles, and state the central lemma that implies the proof of Theorem 1 and Theorem 2.

Definition 7.2 *A mapping $\pi : V(H) \to [k]$ is called a* (q, c)-partition oracle *if:*

- *for any $v \in V(H)$, the query complexity of computing $\pi(v)$ is bounded by $O(q)$*
- *for any $v \in V(H)$, the time complexity of computing $\pi(v)$ is bounded by $O(c)$*

For a subset $Y \subset V(H)$, we define partial partition oracles *$\pi : Y \to [k]$ similarly.*

Given a sequence $\{\pi_i\}_{i=1}^m$ of m partition oracles, we say that the sequence $\{\pi_i\}_{i=1}^m$ has shared query complexity *f, if for every $v \in V(H)$ computing the m values $\pi_1(v), \pi_2(v), \ldots, \pi_m(v)$ requires at most f queries in all.*

Now we are ready to state the main lemma required for the proof of Theorems 1 and 2.

Lemma 7.3 *Let $H = (V, E_1, \ldots, E_s)$ be a hypergraph, let $0 < \epsilon < 1$ be a fixed constant and let Ψ be a set of density tensors. There exist $m = m_{7.3}(\epsilon, s, r, k) = \exp\left(\epsilon^{-O(s \cdot r \cdot k^r)}\right)$ and $f = f_{7.3}(\epsilon, s, r, k) = \mathrm{poly}(k^r, s, 1/\epsilon)$ for which there is a randomized algorithm A_{R} that generates a sequence of size m of (f, f)-partition oracles $\{\pi_i : V(H) \to [k]\}_{i=1}^m$ with shared query complexity f, such that:*

- *If H satisfies Ψ then with probability at least $\frac{1}{2}$ one or more of the m partition oracles induces a partition which ϵ-approximately satisfies Ψ.*
- *The algorithm A_R makes* no *queries to H, and it is not affected by the property Ψ.*

The running time of algorithm A_R is bounded by $O(m)$.

Note that the algorithm A_R does not depend on H or Ψ at all. It only depends on the general parameters $(s, r_1, \ldots, r_s, \epsilon, k)$ of the problem, and none of the parameters of the specific input.

Informally, our partition oracles will be constructed as follows.

- **(representation)** every partition oracle π will be associated with some small subset $U \subset V(H)$ of H's vertices, a partition Π_U of U into k components, and a vector $\bar{\beta}$. The set U will be usually shared among the partition oracles within a sequence.
- **(query complexity)** for every vertex $v \in V(H)$, some subset of edges within $U \cup \{v\}$ will be queried. As we mentioned above, in case of sequences of partition oracles the set U will be shared, and hence the *shared query complexity* of the considered sequences will be small (when U is small).
- **(operation)** every vertex $v \in V(H)$ is classified according to the outcomes of the queries in the previous stage and the partition Π_U. Then it is mapped to some $j \in [k]$ at random, according to the vector $\bar{\beta}$ which is interpreted as a distribution.

Before proving Lemma 7.3, we state two additional lemmas and through them prove Theorem 1 and Theorem 2.

Lemma 7.4 *Let H be a hypergraph and let Ψ be a set of density tensors. For every $\epsilon > 0$, if H ϵ'-approximately satisfies Ψ, where $\epsilon' \triangleq \frac{\epsilon}{2r}$, then H also ϵ-closely satisfies Ψ.*

Lemma 7.5 *Let Ψ be a property that is checkable for proximity in time $TC(\Psi)$, and let $\{\pi_i\}_{i=1}^m$ be a sequence of m (q, c)-partition oracles with shared query complexity f. For every $0 < \delta, \epsilon < 1$, there is a randomized algorithm A_S that satisfies the following.*

- *If one of the partition oracles π_i defines a partition Π_i that $\epsilon/2$-approximately satisfies the property Ψ, then with probability at least $1 - \frac{\delta}{2}$ algorithm A_S outputs i and a density tensor $\psi \in \Psi$ which is ϵ-approximated by ψ^{Π_i} – the true density tensor of Π_i.*
- *If for every π_i the induced partition Π_i does not even ϵ-approximately satisfies Ψ, then algorithm A_S outputs* False *with probability at least $1 - \frac{\delta}{2}$.*

Furthermore, the query complexity of A_S is bounded by

$$O\left((r \cdot f + s) \cdot (\frac{1}{\epsilon})^2 \cdot \log(\frac{m \cdot sk^r}{\delta})\right)$$

and the time complexity of A_S is bounded by

$$O\left(m \cdot c \cdot (r \cdot f + s) \cdot (\frac{1}{\epsilon})^2 \cdot \log(\frac{m \cdot sk^r}{\delta})\right) + m \cdot TC(\Psi)$$

We proceed with proving Theorem 1 and Theorem 2, and postpone the prof of Lemma 7.4 and Lemma 7.5 to Subsection 7.5.1 and Subsection 7.5.2 respectively.

Proof of Theorem 1

First we apply Lemma 7.3 with accuracy parameter $\epsilon' = \frac{\epsilon}{4r}$, repeating it $2\log(1/\delta)$ independent times, and thus we obtain $m' = m \cdot 2\log(1/\delta)$ partition oracles with shared query complexity $2\log(1/\delta) \cdot f$, where we know that if H satisfies Ψ, then with probability at least $1 - \frac{1}{2}\delta$ at least one of the m' partition oracles induces a partition which ϵ'-approximately satisfies Ψ. Then we can apply the algorithm from Lemma 7.5 on all m' partitions and write down a $2\epsilon'$-approximately satisfying partition if we find one, and output "No Partition" otherwise. In case we found a $2\epsilon'$-approximately satisfying partition oracle π, we partition all vertices of H in time $O(n \cdot f)$. Then we modify the resulting partition by making a minimal number of vertex moves according to Lemma 7.4 (see its proof below), so if one of the oracles indeed $2\epsilon'$-approximately satisfies Ψ then with probability $1 - \delta/2$ Lemma 7.5 detected it and hence its corresponding modified partition ϵ-closely satisfies Ψ. As a conclusion, the time complexity of algorithm A is bounded by

$$O\Big(m' \cdot \Big[f_{7.3}^2 \cdot \Big(\frac{1}{\epsilon'}\Big)^2 \cdot \log\Big(\frac{m' s k^r}{\delta}\Big) \cdot r \cdot s + TC(\Psi)\Big] + n \cdot f_{7.3}\Big) =$$

$$\exp\Big((r/\epsilon)^{O(s \cdot r \cdot k^r)}\Big) \cdot \log^2(1/\delta) + n \cdot \mathrm{poly}(k^r, s, 1/\epsilon)$$

and the probability of success is at least $1 - \delta$, matching the assertions of Theorem 1. ■

Proof of Theorem 2

First we apply Lemma 7.3 with $\epsilon' = \frac{\epsilon}{4r}$, repeating it $2\log(1/\delta)$ independent times and obtaining $m' = m \cdot 2\log(1/\delta)$ partition oracles with shared query complexity $2\log(1/\delta) \cdot f$, where with probability at least $1 - \frac{1}{2}\delta$, at least one of them induces an ϵ'-approximately satisfying partition in the case that H satisfies Ψ. Then we can apply the algorithm from Lemma 7.5 on all m' partitions and output the density tensor $\psi \in \Psi$ if we received one from algorithm A_S. Observe that by Lemma 7.4, the density tensor ψ (in the case that the algorithm A_S did not err) is ϵ-closely satisfied by H. Hence the total probability of success is at least $1 - \delta$. The time complexity of algorithm A_T is bounded by

$$\exp\Big((r/\epsilon)^{O(s \cdot r \cdot k^r)}\Big) \cdot \log^2(1/\delta)$$

and its query complexity is bounded by

$$\mathrm{poly}(k^r, s, 1/\epsilon) \cdot \log^2(1/\delta)$$

matching the assertions of Theorem 2. ■

7.3 Proof of Lemma 7.3

As we mentioned in Section 6, in order to maintain a small approximation error the partitioning algorithm A_R is going to classify the vertices of H incrementally, by first splitting $V(H)$ into sets $Y^1, Y^2, \ldots, Y^{8/\epsilon}$ and then partitioning every set Y^i separately. Therefore, we first state a technical lemma related to this partial partitioning algorithm, and using it we prove Lemma 7.3.

Lemma 7.6 *Let $H = (V, E_1, \ldots, E_s)$ be a hypergraph, let $0 < \alpha < \epsilon < 1$ be fixed constants and let Ψ be a set of density tensors. Let in addition $\Pi = \{V_1, \ldots, V_k\}$ be a partition of H's vertices that α-approximately satisfies the property Ψ and let $Y \subset V(H)$ be a set of size $\frac{\epsilon}{8}n$.*

There exists $\hat{m} = \hat{m}_{7.6}(\epsilon, s, k, r) = \exp\left(\epsilon^{-O(s \cdot k^r)}\right)$ and $f = f_{7.6}(\epsilon, s, k, r) = \text{poly}(k^r, s, 1/\epsilon)$ for which there is a randomized algorithm $A_{1/\epsilon}$ that outputs a sequence of $\hat{m}$ partial *(f, f)-partition oracles for Y with shared query complexity f,* without looking at Π*, such that with probability $1 - \frac{\epsilon}{16}$ at least one of the partition oracles induces a partition $\Pi_U = \{Y_1, \ldots, Y_k\}$ such that combining it with the original partition Π gives a partition $\hat{\Pi} = \{\hat{V}_1, \ldots, \hat{V}_k\}$ (where for every i, $\hat{V}_i = ((V_i \setminus Y) \cup Y_i)$), which $(\alpha + \frac{\epsilon^2}{8})$-approximately satisfies Ψ.*

Furthermore, the partial partitioning algorithm $A_{1/\epsilon}$ makes no *queries to H at all, is not dependent on Ψ, and its time complexity is bounded by $O(\hat{m})$.*

Supposing that we have such a *partial* partitioning algorithm $A_{1/\epsilon}$, we show how to construct the main partitioning algorithm A_{R} from Lemma 7.3.

Proof (of Lemma 7.3). Assume that H has a partition Π that satisfies the property Ψ. First we arbitrarily split V into $l = 8/\epsilon$ equal-sized sets $\{Y^1, \ldots, Y^l\}$. Then we execute for every Y^i the partial partitioning algorithm $A_{1/\epsilon}$, and write down the $\hat{m}$ partition oracles of each Y^i. Recall that the partial partitioning algorithm $A_{1/\epsilon}$ does not require knowledge of the existing partition Π. Using these $\hat{m} \cdot l$ partial partition oracles $\{(\pi_1^1, \ldots, \pi_{\hat{m}}^1), \ldots, (\pi_1^l, \ldots, \pi_{\hat{m}}^l)\}$ we construct a set of $\hat{m}^l$ *complete* partition oracles for V by going over all possible combinations. Namely, we combine every possible sequence $\pi_{i_1}^1, \pi_{i_2}^2, \ldots, \pi_{i_l}^l$ of l partial oracles into a complete oracle, that partitions all of H's vertices into k components. We claim that with probability at least $\frac{1}{2}$, one of these $\hat{m}^l$ complete partition oracles induces a partition that ϵ-approximately satisfies Ψ.

Assume that we start with an implicit partition $\Pi^0 = \{V_1^0, \ldots, V_k^0\}$ of H that satisfies the property Ψ, or equivalently α-approximately satisfies Ψ for $\alpha = 0$. Since every execution of $A_{1/\epsilon}$ fails with probability at most $\frac{\epsilon}{16} = \frac{1}{2l}$, with probability at least $1/2$ we have a sequence of l "correct" partial partition oracles that admits the following construction. To construct the correct complete partition oracle, we take the first "correct" induced partition $\Pi_{Y^1} = \{Y_1^1, \ldots, Y_k^1\}$ of Y^1, and build (implicitly) a new complete partition $\Pi^1 = \{V_1^1, \ldots, V_k^1\}$, where $V_i^1 = (V_i^0 \setminus Y^1) \cup Y_i^1$, and where we now know (by Lemma 7.6) that Π^1 is a partition that $\alpha + \frac{\epsilon^2}{8} = \frac{\epsilon^2}{8}$-approximately satisfies Ψ. Now we continue with a "correct" induced partition of Y^2 with respect to Π^1, and build a complete partition Π^2 that $2 \cdot \frac{\epsilon^2}{8}$-approximately satisfies Ψ and so on. Eventually, with probability $1/2$, there is a "right" sequence of the induced partial partitions Π_{Y^i}, from which we get a complete partition Π^l which (due to the triangle inequality) $\frac{8}{\epsilon} \cdot \frac{\epsilon^2}{8} = \epsilon$-approximately satisfies Ψ as required. Moreover, the way that the partitions are constructed is such that for this to work (by trying out all combinations of the provided partition oracles of $Y^1, \ldots, Y^l$) we require no knowledge on Π at all, apart from that it exists. ■

7.4 Algorithm $A_{1/\epsilon}$ - Proof of Lemma 7.6

7.4.1 Overview

The main idea behind the partial partitioning algorithm is the following. Let $\Pi = \{V_1, \ldots, V_k\}$ be the (unknown) partition which approximately satisfies the property Ψ, and let $\tilde{Y}$ be some small enough

set of vertices. We refer to the *type* of an edge $e = (v_1, \ldots, v_{r_i})$ as its configuration with respect to Π, i.e. the number of vertices that it has in every component, and their positions in e. If almost all vertices in $\tilde{Y}$ participate in the same number of edges of any type (with respect to $V_1 \setminus \tilde{Y}, \ldots, V_k \setminus \tilde{Y}$), then we can redistribute $\tilde{Y}$ in *any way*, as long as the component sizes are almost preserved, and still get a partition Π' which is almost as good as Π.

Once we convinced ourselves that this is true, we can think of a somewhat larger set $Y \subset V$, and its own partition $\{Y_1, \ldots, Y_c\}$ where in every Y_i (similarly to the set $\tilde{Y}$ above) almost all vertices are "similar" (note that this partition of Y is not related to the main partition Π, and the partition size c depends among other things on the level of accuracy in our notion of "almost"). Now, if we redistribute each Y_i in a way that preserves the component sizes in Π, then still (after the c redistributions) we get a partition which approximately satisfies the property Ψ.

In the following section we define the notion of "almost similar vertices" more precisely, and show how to cluster a given set in a way that groups almost similar vertices together.

7.4.2 Clustering vertices

Let $Y^0 \subset V$ be a fixed set of vertices and let $\Pi = \{V_1, \ldots, V_k\}$ be some partition of V. Denote by $\{W_1, \ldots, W_k\}$ the partition that Π induces on $V \setminus Y^0$, i.e. $\{V_1 \setminus Y^0, \ldots, V_k \setminus Y^0\}$. Our aim in this section is to cluster every vertex $v \in Y^0$ according to its relative density scheme with respect to $\{W_1, \ldots, W_k\}$. Referring to our discussion above, we are going to group the "similar" vertices together. The clustering procedure is described precisely below, but first we start with some definitions.

We denote by Φ_i^p the restriction of Φ_i to the domain $[r_i] \setminus \{p\}$. Namely, Φ_i^p is the set of all mappings from $[r_i] \setminus \{p\}$ to $[k]$. Fix an edge set E_i, an index $p \in [r_i]$, a vertex $v \in Y^0$ and a mapping $\phi \in \Phi_i^p$. Let $\Gamma_{i,\phi}^p(v)$ denote the set

$$\left\{(v_1, \ldots, v_{r_i}) \in E_i : \left(\forall_{j \in [r_i] \setminus \{p\}}\ v_j \in W_{\phi(j)}\right) \wedge \left(v_p = v\right)\right\}$$

Intuitively, for every extension $\phi' \in \Phi_i$ of ϕ, the size of $\Gamma_{i,\phi}^p(v)$ measures how the density $\mu_{i,\phi'}^{\pi}$ is affected when putting the vertex v in the set $V_{\phi'(p)}$. Since we are interested in the normalized densities of such edges, we define

$$\gamma_{i,\phi}^p(v) = \frac{|\Gamma_{i,\phi}^p(v)|}{n^{r_i - 1}}$$

Let $\gamma(v) = \langle \gamma_{i,\phi}^p(v) \rangle_{i \in [s], p \in [r_i], \phi \in \Phi_i^p}$ denote the density tensor of the vertex v considering all possible edge sets, places and mappings with respect to the partition $\{W_1, \ldots, W_k\}$. This density tensor encodes all information on v that we will use in the following (going back to our previous discussion, "similar" vertices are the vertices that have "similar" density tensors). Now we build an auxiliary set of quantitative density tensors. First let $\mathcal{Q}_\epsilon = \{0, \frac{\epsilon}{32}, \frac{2\epsilon}{32}, \ldots, 1\}$ be the set of integer multiples of $\frac{\epsilon}{32}$ in $[0, 1]$. Then the set of possible clusters is defined as

$$\mathcal{A} = \{\bar{\alpha} = \langle \alpha_{i,\phi}^p \rangle_{i \in [s], p \in [r_i], \phi \in \Phi_i^p} : \forall_{i \in [s], p \in [r_i], \phi \in \Phi_i^p}\ \ \alpha_{i,\phi}^p \in \mathcal{Q}_\epsilon\}$$

In this set of $(\frac{1}{\epsilon})^{O(k^r \cdot s \cdot r)}$ clusters (recall that r is the maximal arity of the edge sets) we group together the vertices that have approximately (up to $\frac{\epsilon}{16}$) the same density tensors. The precise clustering of a vertex $v \in Y^0$ is performed as follows. We say that the tensor $\gamma(v)$ of v is *close* to the cluster $\bar{\alpha} \in \mathcal{A}$

if for every i, ϕ, p we have $\gamma^p_{i,\phi}(v) = \alpha^p_{i,\phi} \pm \epsilon/32$. Note that one tensor $\gamma(v)$ can be close to multiple clusters. The final clustering will be based on sampled approximate tensors of H's vertices (rather than the real tensors), and all we will require is that almost every vertex v goes to a cluster that is close to $\gamma(v)$.

In the following we show that redistributing vertices from the same cluster cannot change the resulting edge densities from those of the initial partition of V by much.

7.4.3 Redistributing vertices from the same cluster

Let $\Pi = \{V_1, \ldots, V_k\}$ be any fixed partition of V, and let $Y^0 \subset V$ be a set of at most $\frac{\epsilon}{8}n$ vertices. As above, denote by $\{W_1, \ldots, W_k\}$ the partition induced by Π on $V \setminus Y^0$. Let $Y^{0,\bar{\alpha}}$ be a subset of Y^0 such that all but at most $\xi|Y^{0,\bar{\alpha}}|$ vertices $v \in Y^{0,\bar{\alpha}}$ are close to the same cluster $\bar{\alpha}$ with respect to the partition $\{W_1, \ldots, W_k\}$. This means that there is a tensor $\bar{\alpha} = \langle \alpha^p_{i,\phi} \rangle_{i\in[s], p\in[r_i], \phi\in\Phi^p_i}$ such that for all but at most $\xi|Y^{0,\bar{\alpha}}|$ vertices $v \in Y^{0,\bar{\alpha}}$ we have $\forall_{i\in[s],p\in[r_i],\phi\in\Phi^p_i}\ \gamma^p_{i,\phi}(v) = \alpha^p_{i,\phi} \pm \epsilon/32$. We claim that if the vertices of such a $Y^{0,\bar{\alpha}}$ are redistributed over the k components of Π in a way that approximately preserves the number of vertices in every component, then the edge densities (according to any index i and mapping $\phi \in \Phi_i$) remain almost the same as before the redistribution of $Y^{0,\bar{\alpha}}$. Formally:

Claim 7.7 *Let $(Y^{0,\bar{\alpha}}_1, \ldots, Y^{0,\bar{\alpha}}_k)$ be any partition of $Y^{0,\bar{\alpha}}$ that for each $1 \le j \le k$ satisfies $||V_j \cap Y^{0,\bar{\alpha}}| - |Y^{0,\bar{\alpha}}_j|| < \eta|Y^{0,\bar{\alpha}}|$, and let $\hat{\Pi} = \{\hat{V}_1, \ldots, \hat{V}_k\} = \{W_1 \cup Y^{0,\bar{\alpha}}_1, \ldots, W_k \cup Y^{0,\bar{\alpha}}_k\}$ be the redistributed partition of V. Then the following holds.*

- $\forall_{i\in[k]},\ ||V_j| - |\hat{V}_j|| < \eta|Y^{0,\bar{\alpha}}|$
- $\forall_{i\in[s],\phi\in\Phi_i},\quad \left||E^{\Pi}_{i,\phi}| - |E^{\hat{\Pi}}_{i,\phi}|\right| \le (\frac{3\epsilon}{16} + \xi + \eta)|Y^{0,\bar{\alpha}}|n^{r_i-1}$

Proof. The first inequality follows directly from our assumptions, so we move to the second one. Fix i and ϕ. We call an edge e *relevant* with respect to Π, i and ϕ if $e \in E^{\Pi}_{i,\phi}$. Our purpose is to show that for all i and ϕ, the size of $E^{\Pi}_{i,\phi}$ is close to the size of $E^{\hat{\Pi}}_{i,\phi}$. The amount of relevant edges which have no vertices in $Y^{0,\bar{\alpha}}$ at all is preserved. We partition the rest of the relevant edges into two types:

$$I_1 = \{e \in (E^{\Pi}_{i,\phi} \cup E^{\hat{\Pi}}_{i,\phi}) : |e \cap Y^{0,\bar{\alpha}}| = 1\}$$

$$I_{\ge 2} = \{e \in (E^{\Pi}_{i,\phi} \cup E^{\hat{\Pi}}_{i,\phi}) : |e \cap Y^{0,\bar{\alpha}}| \ge 2\}$$

Clearly the size of $I_{\ge 2}$ is bounded by $|Y^{0,\bar{\alpha}}|^2 n^{r_i-2} \le \frac{\epsilon}{8}|Y^{0,\bar{\alpha}}|n^{r_i-1}$. As for I_1, its influence on the change in the density may be caused by one of the following.

- At most $\xi|Y^{0,\bar{\alpha}}|$ vertices that do not reside in the same cluster as the others. These vertices can participate in at most $\xi|Y^{0,\bar{\alpha}}|n^{r_i-1}$ relevant edges.
- The difference between the vertices that reside in the same cluster. This can change the number of relevant edges by at most $\frac{\epsilon}{16}|Y^{0,\bar{\alpha}}|n^{r_i-1}$.
- The difference in the sizes of the components before and after the redistribution. Since these differences are bounded by $\eta|Y^{0,\bar{\alpha}}|$, the change in the number of crossing edges due to them is at most $\eta|Y^{0,\bar{\alpha}}|n^{r_i-1}$.

Summing up, the difference in the number of relevant edges before and after the redistribution is at most $(\frac{3\epsilon}{16} + \xi + \eta)|Y^{0,\bar{\alpha}}|n^{r_i-1}$. ■

7.4.4 Redistributing general sets of vertices

Until now we referred to sets $Y^{0,\bar{\alpha}}$ in which almost all vertices are similar. Now we turn to the case where we need to redistribute an arbitrary set Y^0 of size at most $\frac{\epsilon}{8}n$ (let the V_j's, W_j's and the $\hat{V}_j$'s denote the sets of the above partitions with respect to Y^0). Assume for now that we have an oracle that provides us the following information:

(i) A clustering $\{Y^{0,\bar{\alpha}_1},\ldots,Y^{0,\bar{\alpha}_c}\}$ of Y^0, such that all but at most an $\epsilon/4$ fraction of the vertices in Y^0 are assigned to a close cluster with respect to $\{W_1,\ldots,W_k\}$

(ii) For every $Y^{0,\bar{\alpha}_i}$ and V_j, the approximate normalized size of $|Y^{0,\bar{\alpha}_i}\cap V_j|$. Namely, for each $Y^{0,\bar{\alpha}_i}$ and V_j define $\zeta_j^{\bar{\alpha}_i}\triangleq\frac{|Y^{0,\bar{\alpha}_i}\cap V_j|}{|Y^{0,\bar{\alpha}_i}|}$ as the true intersection size. Then the oracle provides a sequence $(\beta_j^{\bar{\alpha}_i})_{i\in[c],j\in[k]}$ of reals such that

$$\sum_{j=1}^{k}\sum_{i=1}^{c}\left(|\zeta_j^{\bar{\alpha}_i}-\beta_j^{\bar{\alpha}_i}|\cdot|Y^{0,\bar{\alpha}_i}|\right)<\frac{\epsilon}{4}\cdot|Y^0|$$

Denoting by ξ_h the fraction of vertices in every $Y^{0,\bar{\alpha}_h}$ that are not close to the cluster $\bar{\alpha}_h$, we have $\sum_h\xi_h|Y^{0,\bar{\alpha}_h}|\leq\frac{\epsilon}{4}|Y^0|$, and denoting by η_h the sum $\sum_{j=1}^{k}|\zeta_j^{\bar{\alpha}_h}-\beta_j^{\bar{\alpha}_h}|$ we have $\sum_h\eta_h|Y^{0,\bar{\alpha}_h}|\leq\frac{\epsilon}{4}|Y^0|$. Then, after any redistribution of the vertices of each $Y^{0,\bar{\alpha}_h}$ according to the sequence $(\beta_j^{\bar{\alpha}_i})_{i\in[c],j\in[k]}$, from (ii) we have:

$$\forall_{j\in[k]}:||V_j|-|\hat{V}_j||<\frac{\epsilon}{4}|Y^0|\leq\frac{\epsilon^2}{32}n$$

and according to (i), by Claim 7.7 we have:

$$\forall_{i\in[s],\phi\in\Phi_i}:\left||E_{i,\phi}^{\Pi}|-|E_{i,\phi}^{\hat{\Pi}}|\right|\leq\sum_h(\frac{3\epsilon}{16}+\xi_h+\eta_h)|Y^{0,\bar{\alpha}_h}|n^{r_i-1}\leq\frac{\epsilon}{2}|Y^0|n^{r_i-1}\leq\frac{\epsilon^2}{16}n^{r_i}$$

In terms of density tensors, we have: $\forall_{j\in[k]}:|\rho_j^{\Pi}-\rho_j^{\hat{\Pi}}|\leq\frac{\epsilon^2}{32}$ and $\forall_{i\in[s],\phi\in\Phi_i}:|\mu_{i,\phi}^{\Pi}-\mu_{i,\phi}^{\hat{\Pi}}|\leq\frac{\epsilon^2}{16}$.

The only thing left is to describe how to simulate the oracles for (i) and (ii) above. As for the second item (ii), we can just provide corresponding partitions for all possible sets of intersection sizes as follows. Let

$$\mathcal{B}=\left\{\bar{\beta}=(\beta_1^{\bar{\alpha}},\ldots,\beta_k^{\bar{\alpha}})_{\bar{\alpha}\in\mathcal{A}}:(\beta_j^{\bar{\alpha}}\in\{0,\frac{\epsilon}{32},\frac{2\epsilon}{32},\ldots,1\})\wedge(\sum_j\beta_j^{\bar{\alpha}}=1)\right\}$$

The size of $\mathcal{B}$ is bounded from above by $(32/\epsilon)^{k\cdot|\mathcal{A}|}=\exp\left(\epsilon^{-O(s\cdot r\cdot k^r)}\right)$. For every possible $\bar{\beta}\in\mathcal{B}$, we partition the vertices of every $Y^{0,\bar{\alpha}}$ into the k components according to the distribution defined by $(\beta_1^{\bar{\alpha}},\ldots,\beta_k^{\bar{\alpha}})$, i.e. when queried about a vertex $v\in Y^{0,\bar{\alpha}}$ we put it into V_j with probability $\beta_j^{\bar{\alpha}}$, for every $j\in[k]$. Observe that there exists $\bar{\beta}=(\beta_1^{\bar{\alpha}},\ldots,\beta_k^{\bar{\alpha}})_{\bar{\alpha}\in\mathcal{A}}\in\mathcal{B}$ which approximates the true intersection values $(\zeta_1^{\bar{\alpha}},\ldots,\zeta_k^{\bar{\alpha}})_{\bar{\alpha}\in\mathcal{A}}$ even closer than needed, and then for $n=\omega(s\cdot k^r)$, with probability $1-o(1)$, redistributing according to this vector $\bar{\beta}$ will make the intersection sizes as close as required. Note that this avoids having to know the actual sizes of the intersections of $Y^{0,\bar{\alpha}}$ with the components $V_1,\ldots,V_k$ while constructing a partial partition oracle for the vertices in Y^0.

For the first item (i), we are going to choose a random set U of vertices in $V \setminus Y^0$, and approximate the clusters of the vertices in Y^0 according to their density tensors with respect to the partition of U induced by Π. Once again, since we do not know the partition Π, we shall try all possible partitions of U as follows. Let $U \in (V \setminus Y^0)$ be a set of size T. We denote by $\mathcal{P}_U$ the set of all possible k-way partitions of U, namely $\mathcal{P}_U = \{\Pi_U : \Pi_U \text{ is a } k\text{-way partition of } U\}$. Note that the size of $\mathcal{P}_U$ is at most k^T. We are going to partition the set U according to each one of the partitions in $\mathcal{P}_U$, and clearly one of them is the correct one, which is the one that is induced by the initial partition $\Pi = \{V_1, \ldots, V_k\}$. The following lemma states that sampling from a small set U gives a good approximation of the clusters for most of the vertices in Y^0.

Lemma 7.8 *Let $\Pi = \{V_1, \ldots, V_k\}$ be a partition of V, let $Y^0 \subset V$ be a set of size $\frac{\epsilon}{8}n$, and let $U \subset V \setminus Y^0$ be a uniformly chosen random set of $r \cdot t = r \cdot \frac{2^{16}}{\epsilon^2} \log \frac{l \cdot s \cdot r \cdot k^r}{\epsilon}$ vertices (possibly with repetitions). There is a (deterministic) algorithm that outputs a sequence of $k^{rt} \cdot |\mathcal{B}| = \exp\left(\epsilon^{-O(s \cdot r \cdot k^r)}\right)$ partial oracles, that have shared query complexity $s \cdot t$ and that query only edges contained in $U \cup Y^0$ to provide presumed density tensors for vertices in Y^0. With probability at least $1 - 1/(2l)$ over the choice of U at least one of the oracles clusters the vertices of Y^0 so that all but at most $\frac{\epsilon}{4}|Y^0|$ of them are clustered correctly with respect to Π and $\mathcal{A}$. Moreover, both the production and the operation of these oracles do not depend on Π at all.*

7.4.5 Estimating the clusters by sampling – Proof of Lemma 7.8

Let $\Pi = \{V_1, \ldots, V_k\}$ (as we mentioned above, we first assume here that the underlying partition of the vertices is known), let $\{W_1, \ldots, W_k\} = \{V_1 \setminus Y^0, \ldots, V_k \setminus Y^0\}$, and let us fix a vertex $v \in Y^0$, an edge set E_i, $i \in [s]$, a position $p \in [r_i]$ and a mapping $\phi \in \Phi_i^p$. We are going to estimate the value of $\gamma_{i,\phi}^p(v)$ by sampling random sets of vertices from V, and calulating the fraction of the relevant edges within the chosen random subsets and our fixed vertex v. Recall that

$$\gamma_{i,\phi}^p(v) \triangleq \frac{1}{n^{r_i - 1}} \left| \left\{ (v_1, \ldots, v_{r_i}) \in E_i : \left(\forall_{j \in [r_i] \setminus \{p\}} \; v_j \in W_{\phi(j)} \right) \wedge \left(v_p = v \right) \right\} \right|$$

We should now note the following: If we choose uniformly, independently and with repetition a sequence of $r_i - 1$ vertices $v_1, \ldots, v_{p-1}, v_{p+1}, \ldots, v_{r_i}$, and denote by F the event that for every $i \neq p$ between 1 and r_i we have $v_i \in W_{\phi(i)}$ and in addition $(v_1, \ldots, v_{p-1}, v, v_{p+1}, \ldots, v_{r_i}) \in E_i$, then the probability of F is exactly $\gamma_{i,\phi}^p(v)$. Now, we would like to repeat this step $t = \frac{2^{16}}{\epsilon^2} \log \frac{l \cdot s \cdot r \cdot k^r}{\epsilon}$ times independently, and compute $\hat{\gamma}_{i,\phi}^p(v)$, which is the fraction of the number of times where the event F occurred (later we will show what to do here without prior knowedge of $W_1, \ldots, W_k$). Applying the additive Chernoff bound we have

$$\Pr\left[|\hat{\gamma}_{i,\phi}^p(v) - \gamma_{i,\phi}^p(v)| > \frac{\epsilon}{32} \right] < 2 \cdot \exp(-2(\frac{\epsilon}{32})^2 t) < \frac{\epsilon}{32 \cdot l \cdot s \cdot r \cdot k^r}$$

To be able to classify v completely by our oracle, we first choose uniformly and independently (with repetitions, although those would clearly occur with probability $o(1)$) a sequence $U = \{w_1, \ldots w_{(r-1)t}\}$ of $(r-1)t$ vertices. For a requested vertex v, we can for every $i \in [s]$, $p \in [r_i]$ and $\phi \in \Phi_i^p$ compute the estimation $\hat{\gamma}_{i,\phi}^p(v)$ by dividing U into subsequences of length $r_i - 1$ and using the first t of them in the estimation procedure above. By the union bound, the probability for a given v that there exists *any* i, p and ϕ for which the deviation is more than $\frac{\epsilon}{32}$ is bounded by

$\frac{\epsilon}{32l}$. Hence, by the Markov inequality, with probability at least $1 - \frac{1}{3l}$ the sequence U is such that for all vertices $v \in Y^0$ but at most an $\epsilon/4$ fraction of them, we would get $\hat{\gamma}^p_{i,\phi}(v) = \gamma^p_{i,\phi}(v) \pm \frac{\epsilon}{32}$ for all i, p and ϕ. We also note that the number of input queries that we need in order to produce the classification of v is clearly bounded by $t \cdot s \cdot r \cdot k^r$

Now, we still have to deal with the problem of not knowing $W_1, \ldots, W_k$. We do know which vertices are not in any W_i at all (because we know Y^0 which is given by the algorithm and is not dependent on the input hypergraph), so we can classify those vertices from U which are not in those sets. For the other vertices, we have no choice but to construct appropriate oracles for any possibility of partitioning them into k sets, as one of those partitions would be the correct one conforming to $W_1, \ldots, W_k$. Here it is important that U is chosen only once, and then re-used for all our oracles in all of their oracle queries.

7.4.6 Some possible ad-hoc optimizations for properties

In all our treatment, we made no attempt to optimize the dependency functions themselves but only to optimize the function types. Moreover, we applied the general versions of Theorem 1 and Theorem 2 also for properties where clearly our requirements on the partition tensor ψ^{Π} are rather sparse. Before we conclude the proof details for these theorems, here is a good place to sketch how we can use such specific features of special cases to substantially optimize the algorithm.

For several of the special cases discussed in this paper there is a way to ignore many of the values $\gamma^p_{i,\phi}(v)$, thus reducing the size of $\mathcal{A}$ and leading to much shorter running times. The impact of these optimizations will be discussed in full in a future version of this paper, and here we only briefly sketch how they can be done for two of the properties.

k-colorability of r-uniform hypergraphs: We note that this partition property depends only on the density of edges *inside* every partition component V_j. These are only affected by edges including a vertex v whose other $r-1$ vertices are in the same V_j. In other words, we need only to count the values $\gamma^p_{1,\phi}(v)$ for which $\phi : [r] \setminus \{p\} \to [k]$ is a constant function, and there are only rk of those. Moreover, taking into consideration that this is a property of simple (undirected) hypergraphs, we can also do away with the index p. Thus, the power of $1/\epsilon$ that is hidden in the complexity estimates of the colorability algorithm would be $\tilde{O}(k)$ rather than depend on k^r (only multiplicative coefficients would depend on r).

Similar optimizations, to a somewhat lesser extent, are also possible for the problem of estimating the number of satisfiable clauses in a k-CNF instance.

Optimizations for Theorem 3: The property Ψ used in its proof (see Section 4.3) concerns hypergraphs with 4-edges, but these are always between two sets. Therefore, we only need to consider values $\gamma^p_{2,\phi}(v)$ where the function ϕ has a range of size 2. Along with some additional care taken in the proof itself, we believe that it is possible to replace $\hat{k}^5$ in the estimates there with $\hat{k}^{2+o(1)}$.

7.5 The last missing details of the proof

7.5.1 From ϵ'-approximation to ϵ-closeness – Proof of Lemma 7.4

In this section we show that any partition Π, which $\epsilon' = \frac{\epsilon}{2r}$-approximately satisfies Ψ, can be turned into a partition which ϵ-closely satisfies Ψ by correcting the number of vertices in each component, where the correction is done by moving around a minimal number of vertices according to some close density tensor in Ψ. The idea behind this observation is that a partition which ϵ'-approximately satisfies Ψ can be corrected (by definition) by moving vertices in a way that every V_j either gains up to $\epsilon' n$ vertices or loses up to $\epsilon' n$ vertices (but not both). Now we show that as a consequence of such moves the difference in the measured edge densities before and after the correction are small enough.

Formally, let Π be a partition which ϵ'-approximately satisfies Ψ, let ψ^Π be the density tensor of Π and let $\psi \in \Psi$ be the density tensor which is ϵ'-approximately satisfied by Π. By definition,

- for all $j \in [k]$, $\quad \rho_j^\Pi = \rho_j \pm \epsilon'$
- for all $i \in [s]$ and $\phi \in \Phi_i$, $\quad \mu_{i,\phi}^\Pi = \mu_{i,\phi} \pm \epsilon'$

Let $\hat{\Pi}$ be the *corrected* partition, which results from the following process: Let S be a temporary set of vertices, which is initially empty. For each $j \in [k]$ such that $\rho_j^\Pi > \rho_j$, we take $|V_j^\Pi| - \rho_j \cdot n$ vertices from component V_j^Π of Π, and put them into the set S. Then, for each $j \in [k]$ such that $\rho_j^\Pi < \rho_j$, we move $\rho_j \cdot n - |V_j^\Pi|$ vertices from S to the component V_j.

We are going to prove that the partition $\hat{\Pi}$ ϵ-closely satisfies ψ. Recall that by definition of ϵ-closely satisfying, we need to show that:

- for all $j \in [k]$, $\quad \rho_j^{\hat{\Pi}} = \rho_j$
- for all $i \in [s]$ and $\phi \in \Phi_i$, $\quad \mu_{i,\phi}^{\hat{\Pi}} = \mu_{i,\phi} \pm \epsilon$

The first item holds by the definition of $\hat{\Pi}$, so we just need to prove the second one. Let us fix $i \in [s]$ and $\phi \in \Phi_i$ and prove that $\mu_{i,\phi}^{\hat{\Pi}} = \mu_{i,\phi} \pm \epsilon$.

Let $j_h = \phi(h)$ for all $h \in [r_i]$, and let $V_{j_1}, V_{j_2}, \ldots, V_{j_{r_i}}$ be the set (with possible repetitions) of components of Π that participate in the edges from $|E_{i,\phi}^\Pi|$. Observe that any vertex $v \in V_{j_h}$ can participate in the h'th place of at most $n^{r_i - 1}$ edges of $|E_{i,\phi}^\Pi|$. On the other hand, the number of vertices that where added or removed from each V_{j_h} is at most $\epsilon' n$. Therefore, $\left||E_{i,\phi}^\Pi| - |E_{i,\phi}^\Pi|\right| \leq \epsilon' \cdot r_i \cdot n^{r_i}$, and consequently, $\mu_{i,\phi}^{\hat{\Pi}} = \mu_{i,\phi}^\Pi \pm \epsilon/2$. Combining it with $\mu_{i,\phi}^\Pi = \mu_{i,\phi} \pm \frac{\epsilon}{2r}$ we conclude that $\mu_{i,\phi}^{\hat{\Pi}} = \mu_{i,\phi} \pm \epsilon$ as required. ■

7.5.2 Testing proximity to Ψ – Proof of Lemma 7.5

Lemma 7.5 (restated) *Let Ψ be a property that is checkable for proximity in time $TC(\Psi)$, and let $\{\pi_i\}_{i=1}^m$ be a sequence of m (q,c)-partition oracles with shared query complexity f. For every $0 < \delta, \epsilon < 1$, there is a randomized algorithm A_S that satisfies the following.*

- *If one of the partition oracles π_i defines a partition Π_i that $\epsilon/2$-approximately satisfies the property Ψ, then with probability at least $1 - \frac{\delta}{2}$ algorithm A_S outputs i and a density tensor $\psi \in \Psi$ which is ϵ-approximated by ψ^{Π_i} – the true density tensor of Π_i.*
- *If for every π_i the induced partition Π_i does not even ϵ-approximately satisfies Ψ, then algorithm A_S outputs* False *with probability at least $1 - \frac{\delta}{2}$.*

Furthermore, the query complexity of A_S is bounded by

$$O\left((r \cdot f + s) \cdot (\frac{1}{\epsilon})^2 \cdot \log(\frac{m \cdot sk^r}{\delta})\right)$$

and the time complexity of A_S is bounded by

$$O\left(m \cdot c \cdot (r \cdot f + s) \cdot (\frac{1}{\epsilon})^2 \cdot \log(\frac{m \cdot sk^r}{\delta})\right) + m \cdot TC(\Psi)$$

Proof. The main idea is straightforward. Fix one of the partition oracles π, and let Π denote the partition induced by it. Our algorithm will estimate the following measures by sampling:

- for all $j \in [k]$, we compute a value ρ_j^S that estimates the normalized size of the component V_j in the partition Π.
- for every $i \in [s]$ and $\phi \in \Phi_i$, we compute a value $\mu_{i,\phi}^S$ that estimates the corresponding hyperedge density.

In the next step, we check these estimated values as follows. Let ψ^S denote the density tensor corresponding to the estimated values above. If ψ^S $\frac{3}{4}\epsilon$-approximately satisfies Ψ, then we output $\psi \in \Psi$ that is $\frac{3}{4}\epsilon$-approximated by ψ^S (recall that finding such ψ takes $O(TC(\Psi))$ time). Otherwise, we move to the next partition oracle from the sequence $\{\pi_i\}_{i=1}^m$. If we did not find any $\frac{3}{4}\epsilon$-approximately satisfying partition, then we output *False*.

We now define and analyze the estimation procedure formally. First we choose a sequence T_v (with possible repetitions) of $t = (\frac{4}{\epsilon})^2 \cdot \log\left(\frac{4m \cdot (sk^r + k)}{\delta}\right)$ vertices uniformly at random, and we choose one additional sequence T_e of $r \cdot t$ vertices in a similar fashion. Next we make the following queries and write down their results.

- For every partition oracle π and every vertex $v \in T_v \cup T_e$ we compute $\pi(v)$. This requires making $O(r \cdot t \cdot f)$ queries.
- For every $i \in [s]$ we split the first $r_i \cdot t$ vertices of T_e into t consequent r_i-tuples $\{\xi^h = (v_1^h, \ldots, v_{r_i}^h)\}_{h=1}^t$, and for every tuple $\xi^h = (v_1^h, \ldots, v_{r_i}^h)$ we check if $(v_1^h, \ldots, v_{r_i}^h) \in E_i$. This requires making $O(t \cdot s)$ queries.

Summing up, the total number of queries we make is bounded by

$$O\Big(t \cdot (r \cdot f + s)\Big) = O\left((r \cdot f + s) \cdot (\frac{1}{\epsilon})^2 \cdot \log(\frac{m \cdot sk^r}{\delta})\right)$$

Now, once we made all required queries, for every partition oracle π we perform the following.

- For every $j \in [k]$ we set
$$\rho_j^S = \frac{1}{t}|\{v \in T_v : \pi(v) = j\}|$$
- For every $i \in [s]$ and $\phi \in \Phi_i$, we set
$$\mu_{i,\phi}^S = \frac{1}{t}\left|\left\{\xi^h = (v_1^h, \ldots, v_{r_i}^h) : \left((v_1^h, \ldots, v_{r_i}^h) \in E_i\right) \wedge \left(\forall_{j\in[r_i]}\pi(v_j) = \phi(j)\right)\right\}\right|$$

For each $j \in [k]$, let F_j denote the event $|\rho_j^S - \rho_j^\Pi| \geq \epsilon/4$. Similarly, for every $i \in [s]$ and $\phi \in \Phi_i$, let $F_{i,\phi}$ denote the event $|\mu_{i,\phi}^S - \mu_{i,\phi}^\Pi| \geq \epsilon/4$. Observe that if with probability $1 - \frac{\delta}{2}$ none of these "bad" events occur for any one of the partition oracles π, then algorithm A_S satisfies the assertions of the lemma. We concentrate on a somewhat simpler case (which implies this one by the union bound), where we want to bound the probability for one specific π and one specific event F of this type by $\frac{\delta}{2m(sk^r+k)}$.

Note that for every $v \in T_v$ and $j \in [k]$, $\Pr[\pi(v) = j] = \rho_j^\Pi$. Similarly, for every r_i-tuple $\xi^h = (v_1^h, \ldots, v_{r_i}^h)$ from T_e, $i \in [s]$ and $\phi \in \Phi_i$,

$$\Pr\left[\left((v_1^h, \ldots, v_{r_i}^h) \in E_i\right) \wedge \left(\forall_{j\in[r_i]}\pi(v_j) = \phi(j)\right)\right] = \mu_{i,\phi}^\Pi$$

Since the vertices in T_v and T_e were chosen independently at random, we can apply Chernoff's inequality to bound the probability of the event F.

$$\Pr[F = \text{True}] \leq 2 \cdot \exp(-2 \cdot (\epsilon/4)^2 \cdot t) \leq \frac{\delta}{2m \cdot (s \cdot k^r + k)}$$

as required.

Since the operations of A_S consume negligible running time, its time complexity is bounded by m times the query complexity, plus m times the time required for checking proximity to Ψ. So in total, the time complexity of A_S is bounded by

$$O\left(m \cdot c \cdot (r \cdot f + s) \cdot \left((\frac{1}{\epsilon})^2 \cdot \log(\frac{m \cdot sk^r}{\delta})\right)\right) + m \cdot TC(\Psi)$$

■

8 Concluding Remarks and Open Problems

Strong hypergraph regularity: Very recently, a lot of attention was given to obtaining hypergraph regularity lemmas [17, 24, 25, 28] that are strong enough to reprove Szemerédi's number-theoretic theorem [26] and to prove combinatorial deletion results. As it turns out, in these lemmas one needs to consider not only partitions of the vertices of the graph, but also partitions of pairs of vertices, triples of vertices and so on. Furthermore, one needs to consider various requirements on the interactions between the partitions of different arities concerning their densities and beyond (e.g. on the count of certain small substructures). See for example [19] for an algorithmic version of a strong regularity lemma for 3-uniform hypergraphs.

We believe that we can extend our main result about vertex partitions to the case of partitions of pairs and beyond, but (regretfully) we currently cannot use them to give algorithmic versions of the new variants of the regularity lemma discussed above. The main reason is that our "control" of the interactions between partitions of different arities is not strong enough to match the one typically required for a strong hypergraph regularity lemma. As we could not find another very interesting application of this generalization we opted to describe here the vertex partition as in Theorem 1. It would be very interesting to obtain a generalization of Theorem 1 that will be strong enough to yield algorithmic versions of the results of [17, 24, 25, 28].

Improving the output for hypergraph regularity: While we can apply Theorem 1 in a simple way to obtain an algorithmic version of the regularity lemma for graphs (given in Theorem 3), the generalization for hypergraphs (Theorem 4) is more complicated, and has worse output guarantees as it will generally not find small regular partitions if they exist. It would be interesting to see if one can prove a variant of Theorem 4 similar in nature to Theorem 3, perhaps by formulating and proving an adequate notion of "local regularity" first.

Acknowledgments

We are grateful to Vojtech Rödl for answering numerous questions on hypergraph regularity, and to Artur Czumaj for helpful discussions. We also thank Ilan Newman for his collaboration at an early stage of this work. This work emerged from discussions at a DIMACS Workshop[8], and we would like to thank the organizers for their invitation.

References

[1] N. Alon, W. F. de la Vega, R. Kannan and M. Karpinski, Random sampling and approximation of MAX-CSP problems, Proc. of the 34 ACM STOC, ACM Press (2002), 232-239. Also: JCSS 67 (2003), 212-243.

[2] N. Alon, R. A. Duke, H. Lefmann, V. Rödl and R. Yuster, The algorithmic aspects of the regularity lemma, J. of Algorithms 16 (1994), 80-109.

[3] N. Alon and A. Naor, Approximating the cut-norm via Grothendieck's inequality, SIAM J. on Computing 35 (2006), 787-803.

[4] N. Alon and J. H. Spencer, **The Probabilistic Method**, Second Edition, Wiley, New York, 2000.

[5] S. Arora, D. R. Karger and M. Karpinski, Polynomial time approximation schemes for dense instances of NP-Hard problems, J. Comput. Syst. Sci. 58 (1999), 193-210.

[6] F.R.K. Chung, Regularity lemmas for hypergraphs and quasi-randomness, Random Structures and Algorithms, 2 (1991), 241-252.

[8] *Properties of large graphs: From combinatorics to statistical physics and back*, organized by L. Lovász and B. Sudakov, 16.10.06–21.6.06, Rutgers University, New Brunswick NJ, USA.

[7] A. Czumaj and C. Sohler, Testing hypergraph colorability, Theor. Comput. Sci. 331 (2005), 37-52.

[8] A. Czygrinow and V. Rödl, An algorithmic regularity lemma for hypergraphs, SIAM Journal on Computing, 30 (2000), 1041-1066.

[9] R. Diestel, **Graph Theory**, Third Edition, Springer-Verlag, 2005.

[10] R. Duke, H. Lefman and V. Rödl, A fast approximation algorithm for computing the frequencies of subgraphs in a given graph, SIAM J. on Computing 24 (1995) 598-620.

[11] W. Fernandez de la Vega, Max-Cut has a randomized approximation scheme in dense graphs, Random Struct. Algorithms 8 (1996), 187-198.

[12] E. Fischer and I. Newman, Testing versus estimation of graph properties, Proc. of STOC 2005, 138-146. Also, SICOMP, to appear.

[13] A. Frieze and R. Kannan, Quick approximation to matrices and applications, *Combinatorica* 19 (1999), 175-220.

[14] A. Frieze and R. Kannan, A simple algorithm for constructing Szemerédi's regularity partition, Electr. J. Comb. 6: (1999).

[15] A. Frieze and R. Kannan, The regularity lemma and approximation schemes for dense problems, Proc. of FOCS 1996, 12-20.

[16] O. Goldreich, S. Goldwasser and D. Ron, Property testing and its connection to learning and approximation, JACM 45(4): 653-750 (1998).

[17] T. Gowers, Lower bounds of tower type for Szemerédi's uniformity lemma, GAFA 7 (1997), 322-337.

[18] T. Gowers, Hypergraph regularity and the multidimensional Szemerédi theorem, manuscript, 2006.

[19] P. Haxell, B. Nagle and V. Rödl, An algorithmic version of the hypergraph regularity method, Proc of FOCS 2005, 439-448.

[20] Y. Kohayakawa, V. Rödl and J. Skokan, Hypergraphs, quasi-randomness, and conditions for regularity, J. of Combinatorial Theory A, 97 (2002), 307-352.

[21] Y. Kohayakawa, V. Rödl and L. Thoma, An optimal algorithm for checking regularity, SIAM J. on Computing 32 (2003), no. 5, 1210-1235.

[22] J. Komlós and M. Simonovits, Szemerédi's regularity lemma and its applications in graph theory. In: *Combinatorics, Paul Erdős is Eighty*, Vol II (D. Miklós, V. T. Sós, T. Szönyi eds.), János Bolyai Math. Soc., Budapest (1996), 295–352.

[23] M. Langberg, Testing the independence number of hypergraphs, Proc. of RANDOM 2004, 405-416.

[24] B. Nagle, V. Rödl and M. Schacht, The counting lemma for regular k-uniform hypergraphs, Random Structures and Algorithms 28 (2006), 113-179.

[25] V. Rödl and J. Skokan, Regularity lemma for k-uniform hypergraphs, Random Structures and Algorithms 25 (2004), 1-42.

[26] E. Szemerédi, Integer sets containing no k elements in arithmetic progression, Acta Arith. 27 (1975), 299-345.

[27] E. Szemerédi, Regular partitions of graphs, In: *Proc. Colloque Inter. CNRS* (J. C. Bermond, J. C. Fournier, M. Las Vergnas and D. Sotteau, eds.), 1978, 399–401.

[28] T. Tao, A variant of the hypergraph removal lemma, J. Combin. Theory, Ser. A 113 (2006), 1257-1280.