APPROXIMATE HYPERGRAPH PARTITIONING AND APPLICATIONS*

ELDAR FISCHER†, ARIE MATSLIAH‡, AND ASAF SHAPIRA§

Abstract. Szemerédi's regularity lemma is a cornerstone result in extremal combinatorics. It (roughly) asserts that any dense graph is composed of a finite number of pseudorandom graphs. The regularity lemma has found many applications in theoretical computer science and thus a lot of attention was given to designing algorithmic versions of this lemma. Our main results in this paper are the following:

- We introduce a new approach to the problem of constructing regular partitions of graphs, which results in a surprisingly simple $O(n)$ time algorithmic version of the regularity lemma, thus improving over the previous $O(n^2)$ time algorithms. Furthermore, unlike all the previous approaches for this problem [3, 10, 14, 15, 21], which only guaranteed to find tower-size partitions, our algorithm will find a small regular partition, if one exists in the graph.
- For any constant $r \geq 3$ we give an $O(n)$ time randomized algorithm for constructing regular partitions of r-uniform hypergraphs, thus improving the previous $O(n^{2^{r-1}})$ time (deterministic) algorithms [8, 15].

The above results are obtained as an application of an efficient algorithm for approximating *partition problems* of hypergraphs which we obtain here.

- Given a (directed) hypergraph with bounded edge arities, a set of constraints on the set sizes and densities of a possible partition of its vertex set, and an approximation parameter, we provide in $O(n)$ time a partition approximating the constraints if a partition satisfying them exists. We can also test in $O(1)$ time for the existence of such a partition given the approximation parameter.

This algorithm extends the result of Goldreich, Goldwasser and Ron for graph partition problems [16], and encompasses more recent hypergraph related results such as the maximal constraint satisfaction approximation of [5].

*A PRELIMINARY VERSION OF THIS PAPER APPEARED IN PROC. OF THE 48^{TH} ANNUAL IEEE SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE (FOCS) 2007, 579-589.

†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.

‡CWI, Amsterdam. Email: ariem@cwi.nl.

§Georgia Institute of Technology. Email: asafico@math.gatech.edu. Research supported by NSF grant DMS-0901355.

1. Introduction and General Background.

1.1. Background on Szemerédi's regularity lemma. The regularity lemma of Szemerédi [26] is one of the most important results in graph theory. It guarantees that any dense graph can be partitioned into a bounded number of bipartite graphs that are "pseudorandom". In other words, the lemma guarantees that every graph has an ϵ-approximation of constant descriptive size. This fact is very useful in many applications since dealing with random-like graphs is much easier than dealing with arbitrary graphs. 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 nonconstructive, 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]. For a comprehensive survey of the applications of the lemma, the interested reader is referred to [22].

Before stating the regularity lemma we need some basic notation. 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 $d(A,B) = |E(A,B)|/(|A||B|)$. We use the notation $a = b \pm c$ as a shorthand for $b - c \leq a \leq b + c$. The original notion of regularity, to which we refer to as *subset regularity*, is defined as follows.

DEFINITION 1.1 (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 $d(A',B') = d(A,B) \pm \epsilon$.*

Note that a random bipartite graph with a positive edge density is $o(1)$-subset-regular with high probability. Thus, the smaller ϵ is, the more "random"-like an ϵ-subset-regular pair is.

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 1.2 ([26]). *For every $\epsilon > 0$ there exists $T = T_{1.2}(\epsilon)$, such that any graph with $n \geq 1/\epsilon$ vertices has an ϵ-subset-regular equipartition of order $\hat{k}$, where $1/\epsilon \leq \hat{k} \leq T$.*

We now define an equivalent notion of regularity. 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 $\binom{|A|}{2}\binom{|B|}{2}$. [1]

DEFINITION 1.3 (ϵ-locally-regular pair and partition). *A pair (A,B) of vertex sets is* ϵ-locally-regular *if $d_{C_4}(A,B) = (d(A,B))^4 \pm \epsilon$.*

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

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

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 proof of the regularity lemma is fairly simple. One starts with any partition of the vertices of the input graph into $k = 1/\epsilon$ sets $V_1, \ldots, V_k$. If this partition is ϵ-regular then we are done. Otherwise, there are many pairs (V_i, V_j) which are not ϵ-regular. For any such pair V_i, V_j, there are thus[2] two subsets $U_i \subseteq V_i$ and $U_j \subseteq V_j$ such that $d(U_i, U_j) \neq d(V_i, V_j) \pm \epsilon$. One then shows how to use the collection of pairs (U_i, U_j) in order to refine the partition $V_1, \ldots, V_k$ into a new partition $V_1, \ldots, V_{k'}$ in such a way that a certain potential function increases by at least ϵ^5. Since this potential function is bounded from above by 1 this process must terminate after $1/\epsilon^5$ steps. The only nonconstructive part of this proofs is the task of finding the pairs (U_i, U_j). Therefore, the most natural way to obtain an algorithmic version of the regularity lemma is to find a polynomial-time algorithm for solving this problem. Indeed, *all* the previous algorithmic versions of the regularity lemma [3, 10, 14, 15, 21] used this scheme; they only differed in the details on how one finds the pairs (U_i, U_j). The first polynomial-time algorithm given by Alon et. al [2] was based on Boolean matrix multiplication. The algorithm of Frieze and Kannan [14] used singular value decomposition. The algorithm of Alon and Naor was based on an approximation algorithm for the cut-norm [3]. Finally, the algorithm of Kohayakawa, Rödl and Thoma [21] was based on performing random walks on expanders, being the first to obtain a running time of $O(n^2)$.

As is well known, 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 function by $\text{Tower}(i) = 2^{\text{Tower}(i-1)} = 2^{\cdot^{\cdot^{\cdot^{2}}}\}i}$. Then the bounds on $T_{1.2}(\epsilon)$ 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 $\text{Tower}(1/\epsilon^{1/16})$.

All previous algorithmic versions of the regularity lemma had two drawbacks due to their shared principles. First, since they reprove the regularity lemma algorithmically, they could only be guaranteed to find the worst-case regular partition. By the result of Gowers [17] mentioned above, this worst case partition can be of size $\text{Tower}(1/\epsilon)$. Therefore, even if the graph has a small regular partition these algorithms are only guaranteed to return a partition of size $\text{Tower}(1/\epsilon)$. Second, although the regularity lemma is approximative in nature, these algorithms traditionally read the entire graph to provide the partition, making their running time equal to $\Omega(n^2)$ or worse for a fixed ϵ.

1.2. A new approach for the constructive regularity lemma. In this paper we take a completely different approach to the problem of constructing a regular partition. Instead of algorithmically proving the lemma, we take the lemma "for granted" and design an algorithm for checking if a graph has a regular partition with a certain number of classes. Thus, our algorithm will find a small partition in the graph if one exists, as opposed to all previous algorithms, which can only guarantee to find the worst case partition of size $\text{Tower}(1/\epsilon)$. Our second improvement over the previous $O(n^2)$ algorithms is that the running time of our algorithm is $O(n)$ for any fixed ϵ. Additionally, even the hidden constants in the $O(n)$ running time will only

[2]By the equivalence between the two notions of regularity we have defined above.

depend on the size of the smallest regular partition. Note that this running time is *sublinear* in the size of the input (which is $O(n^2)$), and is optimal as it is linear in the output size – just "writing down" a partition of the vertices takes $O(n)$ time.

THEOREM 1.4 (Main Result). *There is a randomized algorithm, that given $\epsilon > 0$ and an n vertex graph G, that has a $\epsilon/2$-regular partition of order $\hat{k} \geq 1/\epsilon$,* [3] *produces with high probability an ϵ-regular partition of G of order k, where $1/\epsilon \leq k \leq \hat{k}$. The expected running time of the algorithm is*

$$n \cdot \mathrm{poly}(\hat{k}) + 2^{2^{\mathrm{poly}(\hat{k})}} . \tag{1.1}$$

Additionally, if only the densities of the regular partition rather than the partition itself need to be produced, then this can be done in time $2^{2^{\mathrm{poly}(\hat{k})}}$.

We stress that in Theorem 1.4, 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-type dependence on ϵ as in the previous algorithms. As it turns out, a simple inspection of the proof we provide can show that the constant $\frac{1}{2}$ in the regularity parameter can be replaced by any constant smaller than 1; we just use the constant $\frac{1}{2}$ for maintaining the simplicity of the presentation. 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.

An interesting aspect of our new algorithmic version of graph regularity is that it is obtained as a simple application of a general *hypergraph* partitioning algorithm which we construct in this paper. This aspect of the paper is discussed in the next subsection.

1.2.1. Hypergraph regularity. Similarly to graphs, (weak) subset regularity for r-tuples of vertex sets in r-uniform hypergraph is defined as not admitting large subsets $U_1 \subseteq V_1, \ldots, U_r \subseteq V_r$ so that the edge densities of $V_1, \ldots, V_r$ and $U_1, \ldots, U_r$ differ greatly. Regular partitions of hypergraphs are then defined in an analogous manner to that of graphs.

Our main technical tool of hypergraph partitioning is also useful in finding (weakly) regular partitions of r-uniform hypergraphs, when combined with some ideas similar to [12]. For any fixed r and ϵ, we design an $O(n)$ time (randomized) algorithm for finding an ϵ-regular partition of an r-uniform hypergraph. In this case the improvement in the running time is more substantial compared to the previous algorithms, whose running time was $O(n^{2^{r-1}})$. However, as opposed to the graph case, in this case our algorithm is not guaranteed to find small partitions if they exist.

1.3. Hypergraph partition problems. 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 [6, 11] that many of these partition problems have good approximations when the input graph is dense. The main tool that we develop in this paper is an efficient $O(n)$ algorithm for partitioning *hypergraphs*, with an accompanying $O(1)$-query test for the existence of a partition with given parameters.

[3] The reason for the requirement that $\hat{k} \geq 1/\epsilon$ is that the same requirement is needed in Lemma 1.2. For the same reason we return a partition of size $k \geq 1/\epsilon$.

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 $i \leq j$. For instance, 2-colorability can be expressed by a simple partition instance, where $k = 2$, the (trivial) constraints on the sizes of V_1 and V_2 are $0 \leq |V_1|, |V_2| \leq n$, and the constraints on the edges are $|E(V_1)| = |E(V_2)| = 0$ and $0 \leq |E(V_1, V_2)| \leq \binom{n}{2}$.

Goldreich, Goldwasser and Ron [16] showed that for any partition instance Ψ with k parts, and for any positive ϵ, there is an $(2^{(k/\epsilon)^{O(k)}} + (k/\epsilon)^{O(k)}n)$-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 edge 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[4] for example that there is an $(2^{O(1/\epsilon^3)} + O(n/\epsilon^2))$-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 $(2^{O(1/\epsilon^3)} + O(n/\epsilon^2))$-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 a property testing algorithm for Ψ. For any partition problem Ψ as above, it provided a randomized algorithm making only $(k/\epsilon)^{O(k)}$ many queries to the input graph (where each query takes a pair u, v of vertices and answers whether (u, v) is an edge in the 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 for any fixed Ψ and $\epsilon > 0$ one can distinguish in *constant time*[5] 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. For any fixed Ψ and $\epsilon > 0$, 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 a property testing algorithm for such problems, making only poly$(1/\epsilon)$ many queries and running in constant time, to distinguish with probability $\frac{2}{3}$ hypergraphs satisfying Ψ from hypergraphs that are ϵ-far from Ψ.

1.3.1. Prior work on hypergraph partition algorithms. Algorithms similar to the hypergraph partition algorithm that we obtain here have been considered in some earlier papers. Most notably, Frieze and Kannan [13] and Andersson and En-

[4]To be precise, the exact bounds we quote for Max-Cut and 3-colorability follow from a more specialized argument given in [16] and not directly from the general GGR algorithm. See [16] for more details.

[5]More accurately, it does so in constant time plus the time required to make a constant number of queries.

gebretsen [5] obtained partition algorithms for some partition problems of k-uniform hypergraphs as well as testing algorithms for such properties.

Their results differ in some ways from ours. First, while our algorithm can handle hypergraphs with edges of different arities (sizes), the algorithms of [13, 5] can handle (as stated) only hypergraphs with edges of a one arity. As we explain in Section 3, being able to simultaneously handle edges of different arities is crucial to obtaining the algorithm stated in Theorem 1.4. Also, our algorithm is more general in the properties it handles. The algorithm of [5] comes close, handling maximum constraint satisfaction problems, however our algorithm also handles properties not definable by a maximum of densities. In the proof we will highlight the part that allows this added generality.

Another difference is that for any fixed r, if all edges in the input are of size at most r, then the running time of our algorithm (stated in Theorem 2.6) is $O(n)$. The running time of the partition algorithm of [5] is $O(n^r)$. In [13] it is mentioned (see the end of subsection 2.2 in [13]) that their constant time testing algorithms can be turned into explicit partitions in time that is linear in the size of the input, which is also $\Theta(n^r)$. Having a partition algorithm with $O(n)$ running time is crucial to us since we apply this algorithm to obtain the $O(n)$ time algorithmic version of the graph regularity lemma (stated in Theorem 1.4) as well as the $O(n)$ time algorithmic version of the hypergraph regularity lemma (see Section 4). However in itself the running time difference could possibly be bridged by importing some arguments from our proof into the proofs of [5] or [13] rather than proving the entire result from scratch.

1.4. Organization. The rest of the paper is organized as follows. In Section 2 we explicitly define the hypergraph partition problems we consider here, and state the hypergraph partition algorithm we obtain. In Section 3 we prove Theorem 1.4 by reducing the task of finding a regular partition of a graph to the task of finding an appropriate partition of a hypergraph. In Section 4 we obtain an algorithmic version of the regularity lemma for r-uniform hypergraphs. In Section 5 we give an overview of the proof of the hypergraph partition algorithm, and in Section 6 we give the full details of this algorithm. Section 7 contains some concluding remarks and open problems.

2. The Hypergraph Partition Algorithm. We now formally define the hypergraph problem that we will study in this paper.

2.1. General notation. We use $[k]$ to denote the set $\{1, \ldots, k\}$, and use $a = b \pm c$ as a shorthand for $b - c \leq a \leq b + c$. For two (multivariate) functions $f, g : \prod_{i=1}^{k} D_i \to \mathbb{R}^+$, we say that $f = O(g)$ if there exist constants $c, d > 0$ such that for all $x_1 \in D_1, \ldots, x_k \in D_k$ we have $f(x_1, \ldots, x_k) \leq c \cdot g(x_1, \ldots, x_k) + d$. Similarly we write $f = \text{poly}(g)$ is there exist constants $c, d > 0$ such that $f(x_1, \ldots, x_k) \leq (g(x_1, \ldots, x_k))^c + d$ for all $x_1, \ldots, x_k$, and write $f = \exp(g)$ if there exist constants for which $f(x_1, \ldots, x_k) \leq 2^{(g(x_1,\ldots,x_k))^c+d}$ for all $x_1, \ldots, x_k$. We also use the shorthand $f = \text{poly}(g_1, \ldots, g_l)$ for $f = \text{poly}(\prod_{i=1}^{l} g_i)$. In the following we make no attempt to optimize the coefficients involved, only the function types.

2.2. Hypergraphs. 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[6] (ordered sets with possible repetitions) over V. That is, for every edge set $E_i(H)$ we think of any of the edges $e \in E_i$ as a length r_i sequence of members of

[6]The readers with a background in Logic would recognize these as arity r_i relations.

$V(H)$. We say that a hypergraph H is of *type* $(s, r_1, \ldots, r_s)$ if it has exactly s edge sets $E_1, \ldots, E_s$, and for each $i \in [s]$, the set E_i contains edges of size exactly r_i. Note that the usual notion of a directed graph corresponds to hypergraphs of type $(1, 2)$. 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 also set $r = \max_i\{r_i\}$.

For the algorithms that we develop here, we generalize the usual edge queries used in the dense graph model of [16] as follows. Given a hypergraph $H = (V, E_1, \ldots, E_s)$ of type $(s, r_1, \ldots, r_s)$, an edge oracle for H is a function $Q : [s] \times V^{\leq r} \to \{0, 1\}$ such that $Q(i, v_1, \ldots, v_{r_i}) = 1$ if and only if $(v_1, \ldots, v_{r_i}) \in E_i$.

When analysing our testing algorithm 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.

2.3. The partition property. Let H be a hypergraph of type $(s, r_1, \ldots, r_s)$ 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:

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

For example, let $G = (V, E)$ be a hypergraph of type $(1, 2)$, that is a usual directed graph. For some partition $\Pi = \{V_1^\Pi, V_2^\Pi\}$ of V and the identity function $\phi : [2] \to [2]$, $\phi(i) \triangleq i$, the set $E_{1,\phi}^\Pi \subseteq E$ contains all edges $(u, v) \in E$ directed from $u \in V_1^\Pi$ to $v \in V_2^\Pi$.

We now introduce a notion that generalizes the partition instances that were discussed in [16] in the context of graphs.

DEFINITION 2.1 (Density tensors). *A* density tensor *of* order k *and* type $(s, r_1, \ldots, r_s)$ *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* k*-wise partition of a hypergraph of type* $(s, r_1, \ldots, r_s)$ *(whenever* k *and* $(s, r_1, \ldots, r_s)$ *are clear from the context, we call* ψ *simply a* density tensor*). 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, $\mu_{i,\phi}^\Pi = \frac{1}{n^{r_i}} \cdot |E_{i,\phi}^\Pi|$.

DEFINITION 2.2 (Partition properties induced by density tensors). *For a fixed hypergraph* H *of type* $(s, r_1, \ldots, r_s)$*, a set* Ψ *of density tensors (of order* k *and type* $(s, r_1, \ldots, r_s)$*) defines a property of* $V(H)$*'s* k*-wise 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 density tensor* ψ^Π *of* Π *are equal. Namely,* Π satisfies Ψ *if there is* $\psi = \left\langle \langle \rho_j \rangle_{j\in[k]}, \langle \mu_{i,\phi} \rangle_{i\in[s],\phi\in\Phi_i} \right\rangle \in \Psi$ *such that*

- *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 extend this notion of satisfying partitions (and equivalence between density tensors) in two ways as follows.

DEFINITION 2.3 (being ϵ-tightly satisfying/equal). *A partition* Π ϵ-tightly *satisfies* Ψ *if there is* $\psi = \left\langle \langle \rho_j \rangle_{j\in[k]}, \langle \mu_{i,\phi} \rangle_{i\in[s],\phi\in\Phi_i} \right\rangle \in \Psi$ *such that*

- *for all* $j \in [k]$, $\quad \rho_j^\Pi = \rho_j$

- *for all* $i \in [s]$ *and* $\phi \in \Phi_i$, $\quad \mu^{\Pi}_{i,\phi} = \mu_{i,\phi} \pm \epsilon$.

In this case ψ^{Π} *is* ϵ-tightly equal *to* ψ.

DEFINITION 2.4 (being ϵ-loosely satisfying/equal). *A partition* Π ϵ-loosely *satisfies* Ψ *if there is* $\psi = \left\langle \langle \rho_j \rangle_{j\in[k]}, \langle \mu_{i,\phi} \rangle_{i\in[s],\phi\in\Phi_i} \right\rangle \in \Psi$ *such that*

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

Here ψ^{Π} *is* ϵ-loosely equal *to* ψ.

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-*tightly* (respectively ϵ-*loosely*) satisfies the property Ψ if there exists a partition of H's vertices that ϵ-tightly (respectively ϵ-loosely) satisfies the property Ψ. In addition, we may consider a specific density tensor ψ as a singleton set $\Psi = \{\psi\}$, and accordingly as a property of partitions.

Usually the sets Ψ of density tensors will not be given explicitly (as a set), but by some compact representation. For example, let us define a set Ψ_{2COL} of density tensors corresponding to bipartite graphs. Ψ_{2COL} is of order $k = 2$ and type $(s = 1, r_1 = 2)$, and it contains all density tensors $\psi = \left\langle \langle \rho_j \rangle_{j\in[2]}, \langle \mu_\phi \rangle_{\phi:[2]\to[2]} \right\rangle$ that satisfy:

- $0 \le \rho_1, \rho_2 \le 1$;
- $0 \le \mu_\phi \le 1$ if ϕ is injective, and $\mu_\phi = 0$ otherwise.

Clearly Ψ contains infinitely many density tensors, but it has a very small representation. In addition, it is easy to see that given a density tensor ψ and $\epsilon > 0$, computing whether ψ is equal to some $\psi' \in \Psi$, or is ϵ-tightly or ϵ-loosely equal to one, can be done efficiently. We will always make these computational assumptions on the representations of the considered sets Ψ, as formalized next.

DEFINITION 2.5. *Given a set* Ψ *of density tensors of order* k *and type* $(s, r_1, \ldots, r_s)$, *we say that* Ψ is checkable for proximity in time $t = t\Big(s, k, r \triangleq \max_i\{r_i\}\Big)$, *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 *is* ϵ*-loosely equal to some* $\psi_T \in \Psi$, *and if so, the algorithm outputs that tensor* ψ_T.

We say that a family of sets $\mathcal{T} = \{\Psi_{k,(s,r_1,\ldots,r_s)}\}$ *is* efficiently checkable for proximity *if there exists a polynomial* $t = t(x, y)$ *such that for every* $\Psi_{k,(s,r_1,\ldots,r_s)} \in \mathcal{T}$ *of order* k *and type* $(s, r_1, \ldots, r_s)$, *both the representation of* $\Psi_{k,(s,r_1,\ldots,r_s)}$ *and* $TC(\Psi_{k,(s,r_1,\ldots,r_s)})$ *are bounded by* $t(s, k^r)$.

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 is ϵ-loosely equal to some tensor in 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. This will be also the case in our context. From now on, we will only consider (even if not explicitly mentioned) sets of density tensors that are efficiently checkable for proximity.

We are now ready to state the theorem about the hypergraph partitioning algorithm that we develop in this paper.

THEOREM 2.6 (Hypergraph Partitioning Algorithm). *For every* $k \in \mathbb{N}$, *hypergraph type* $(s, r_1, \ldots, r_s)$ *(with* $r = \max\{r_i\}$*) and set* Ψ *of density tensors of order* k *and type* $(s, r_1, \ldots, r_s)$ *(taken from a family* $\mathcal{T}$ *efficiently checkable for proximity), there exists a randomized algorithm* A *taking as inputs a hypergraph* H *of type* $(s, r_1, \ldots, r_s)$ *and two parameters* $\epsilon, \delta > 0$ *such that*

- *the algorithm A either outputs a k-wise partition of $V(H)$, or declares "No Partition";*
- *if H satisfies Ψ then with probability at least $1-\delta$ the algorithm A outputs a partition of $V(H)$ that ϵ-tightly satisfies Ψ;*
- *the algorithm A outputs a partition of $V(H)$ that* does not *ϵ-tightly satisfies Ψ with probability at most δ. In particular, if H does not ϵ-tightly satisfy Ψ, then the algorithm declares "No Partition" with probability at least $1-\delta$.*

The running time of A is bounded by

$$\log^3(1/\delta) \cdot \exp\left(\left(\frac{r}{\epsilon}\right)^{s \cdot r \cdot k^r}\right) + n \cdot \mathrm{poly}(k^r, s, 1/\epsilon) \; .$$

Let us now discuss the property testing variant of Theorem 2.6. To this end, we define one additional measure of closeness to the property Ψ. Let $H = (V, E_1, \ldots E_s)$ and $H' = (V, E'_1, \ldots E'_s)$ be two hypergraphs of type $(s, r_1, \ldots, r_s)$, over the same vertex set V. 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 $dist(H, \Psi) = \min_{H'}\{dist(H, H') : H' \text{ satisfies } \Psi\}$. 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 2.7 (Testing Version of Theorem 2.6). *For every $k \in \mathbb{N}$, hypergraph type $(s, r_1, \ldots, r_s)$ (with $r = \max\{r_i\}$) and set Ψ of density tensors of order k and type $(s, r_1, \ldots, r_s)$ (taken from a family $\mathcal{T}$ efficiently checkable for proximity), there exists a randomized algorithm A_T taking as inputs two parameters $\epsilon, \delta > 0$ and an oracle access to a hypergraph H of type $(s, r_1, \ldots, r_s)$, 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-tightly satisfied by H;*
- *if H does not even ϵ-tightly satisfy the property Ψ then with probability at least $1-\delta$ the algorithm A_T outputs* Reject.

The query complexity of A_T is bounded by $\log^3(\frac{1}{\delta}) \cdot \mathrm{poly}(k^r, s, 1/\epsilon)$, and its running time is bounded by $\log^3(\frac{1}{\delta}) \cdot \exp\left((\frac{r}{\epsilon})^{s \cdot r \cdot k^r}\right)$.

The algorithm above can be used as a testing algorithm in the traditional sense due to the following trivial observation.

CLAIM 2.8. *Any hypergraph that ϵ/k^r-tightly satisfies a partition property Ψ is also ϵ-close to satisfying it.* ∎

Observe that the GGR algorithm [16] follows from the special case of Theorem 2.6 for $r = 2$ and $s = 1$ in the statement of the theorem. Similarly, the property testing algorithm of [16] follows from a special case of Theorem 2.7.

3. Algorithmic Version for Graph Regularity. We start with an overview of the proof of Theorem 1.4. As we have previously mentioned, the main idea is that instead of trying to "algorithmically reprove" the regularity lemma as in all the previous approaches, we take Lemma 1.2 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 1.2 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. Here is how we can use the algorithm stated in Theorem 2.6 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 1.3. The framework of the graph partition problems of [16] thus cannot by itself provide a check for regularity. However, as we show below, a hypergraph partition theorem such as Theorem 2.6 is very useful for checking the regularity condition of Definition 1.3.

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 ordered 4-edges corresponding to the 4-cycles of G. This is not an undirected hypergraph (and also not uniform), so we need at least some of the additional generality provided in Theorem 2.6. 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 1.3 says that there is a real d such that $d = d(V_i, V_j)$ and the number of 4-cycles spanned by (V_i, V_j) 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 $(d(V_i, V_j))^4$. Note here another reason for needing the added generality of Theorem 2.6, as this property is not expressible in terms of individual density constraints alone.

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 2.6 only guarantees that if the hypergraph satisfies the partition instance then the algorithm will return a possibly *different* partition that is close (in the sense of Definition 2.3) to satisfying the partition instance. However, asserting that a partition of H that is close to satisfying the above partition instance is in fact regular with slightly worse parameters turns out to be an easy claim.

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 their initial try. Thus we execute the more costly iterations with sufficiently small probability relative to their cost.

3.1. Proof of Theorem 1.4. We first formally describe the construction, 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 3.1. *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 2.6, 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 unordered 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 3.2 below).

Let us note that while discussing regular partitions, we measure the density of edges and C_4 copies between sets V_i, V_j relative to the size of V_i and V_j, see for example Definitions 1.1 and 1.3. On the other hand, when discussing the partition problems related to Theorem 2.6, 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 2.3, let us use the convention of Theorem 2.6 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 3.2. *Let $\Psi(k,\epsilon)$ denote the set of density tensors for partitions $(V_1,\ldots,V_k)$ which are equipartitions, where in addition for at least a $1-\epsilon$ fraction of the pairs $1 \le i < j \le k$ 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 sets Ψ are efficiently checkable for proximity as per Definition 2.5, matching the implicit assumption in Theorem 2.6 and Theorem 2.7.

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 (remember that in our application k does not depend on n).

CLAIM 3.3. *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 $V(H)$ $\frac{\epsilon}{16k^4}$-tightly 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 1.3 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 $\frac{\epsilon}{16k^4}$-tightly 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 2.6 one would use Theorem 2.7 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 2.6 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 2.6 asks whether a pair of vertices $\{v_1, v_2\}$ is connected to another pair of vertices $\{u_1, u_2\}$ (by C), 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 $k = T_{1.2}(\frac{1}{2}\epsilon) = \text{Tower}(\text{poly}(1/\epsilon))$ we stop and return "fail".[7]

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 2.6 at the k^{th} iteration. Remember that the algorithm does not know $\hat{k}$ in advance, and it may be the case that $\hat{k}$ is as large as $\text{Tower}(1/\epsilon)$ (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 $\text{Tower}(1/\epsilon)$ even if $\hat{k}$ 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 $\frac{\epsilon}{16k^4}$-tightly satisfy $\Psi(k, \frac{1}{2}\epsilon)$. Combined with the second assertion of Claim 3.3 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 3.3 we know that as G has a $\frac{1}{2}\epsilon$-regular partition of size $\hat{k}$, this 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 $\frac{\epsilon}{16\hat{k}^4}$-tightly satisfies $\Psi(\hat{k}, \frac{1}{2}\epsilon)$. By the second assertion of Claim 3.3 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 2.6 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 2.6

[7] The choice of the maximum k is because by Lemma 1.2 we know that any graph has a $\frac{1}{2}\epsilon$-regular partition of size at most $T_{1.2}(\frac{1}{2}\epsilon)$. Therefore, if we reached that value of k then we know that we made an error along the way.

that the running time of the hypergraph partition algorithm in the k^{th} iteration is[8]

$$2^{2^{O(k^5)}} + n \cdot \text{poly}(k).$$

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 $2^{2^{O(\hat{k}^5)}} + n \cdot \text{poly}(\hat{k})$.

Bounding the expected running time:. The version of the algorithm described above runs in time $2^{2^{O(\hat{k}^5)}} + n \cdot \text{poly}(\hat{k})$ 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 2.6). 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 $2^{-2^{b \cdot k^6}} \leq \frac{1}{4} \cdot 2^{-k}$). 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 $2^{2^{O(k^5)}} + n \cdot \text{poly}(k)$. 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

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

4. Regular Partitions of Hypergraphs. As we have mentioned earlier we can also use our hypergraph partitioning algorithm to devise an algorithmic version

[8]Note that the running time we get from Theorem 2.6 is actually $O(2^{2^{O(k^4 \log k)}} + n \cdot \text{poly}(k))$ but for the rest of the analysis we opted for the simpler (somewhat larger) bound.

of the regularity lemma for r-uniform hypergraphs[9]. 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 [25], 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 4.1. *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 an ϵ fraction 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 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 4.2. *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 $O(n^{2^{2r-1}})$ (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 1.4, the algorithm in Theorem 4.2 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 4.2 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 2.6 in order to implement (an aspect of) the iterative procedure in time $O(n)$.

4.1. Proof sketch for Theorem 4.2. The main result in [12] required a way to quickly find the densities of a regular partition of a graph.[10] While the full result

[9] 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.

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

there does not seem immediately translatable to our current body on knowledge on hypergraphs, the part about finding a regular partition is indeed translatable to a proof of Theorem 4.2. 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 4.3. *Recall that 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 k 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.

CLAIM 4.4. *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 $|d(V_{i_1}, \ldots, V_{i_r}) - d(V'_{i_1}, \ldots, V'_{i_r})| \le \epsilon$ for all $1 \le i_1 < i_2 < \cdots < i_r \le k$, then the respective indexes satisfy $|\mathrm{ind}(\mathcal{A}) - \mathrm{ind}(\mathcal{A}')| \le 4\epsilon$.* The proof of this claim follows directly from the definitions of density and index.

Robustness, finality, and regularity:. The main technical observation of Szemerédi in [26] 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 4.5. *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 $k \le l \le f(k)$ and $\mathrm{ind}(\mathcal{B}) \ge \mathrm{ind}(\mathcal{A}) + \delta$. 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 2.6. By using the fact that the index is always bounded between 0 and 1, it is not hard to see the following.

CLAIM 4.6. *For every $\delta > 0$, t and $f : N \to N$ there exists a computable function $T = T_{4.6}(\delta, f, t)$, such that every graph G with $n > T$ vertices has an equipartition $\mathcal{A} = (V_1, \ldots, V_k)$ with $t \le k \le T$ which is (δ, f)-final.*

The main technical step in [26] 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. [7] and [8].

LEMMA 4.7. *For every r and ϵ there exist computable functions $\delta = \delta_{4.7}(r, \epsilon)$*

and $f(k) = f_{4.7}^{(r,\epsilon)}(k)$ *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 4.2 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 4.7. 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 4.4 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. It is easy to verify that the sets $\Psi^{(k,\alpha)}$ are efficiently checkable (as per Definition 2.5), since comparing indices can be trivially done even in linear time. We set $\delta = \delta_{4.7}(r, \epsilon)$ and $f(k) = f_{4.7}^{(r,\epsilon)}(k)$, and now for every $1/\epsilon \leq k \leq f(T_{4.6}(\frac{1}{3}\delta, f, 1/\epsilon))$ and every $\alpha \in \{0, \frac{1}{6}\delta, \frac{2}{6}\delta, \ldots, 1\}$ we run either the algorithm of Theorem 2.6 or the algorithm of Theorem 2.7 (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.[11]

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 4.4
- 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 4.6) 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

[11] 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 $d(W_1, \ldots, W_r)$.

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.

5. Overview of the proof of Theorem 2.6 and Theorem 2.7. Before going to the formal proof of Theorem 2.6 and Theorem 2.7 we briefly describe the general idea of how it is done.

The algorithm (claimed in Theorem 2.6) that we develop in the proof can be divided into three stages as follows.

Stage 1 (running time independent of n) This is the most crucial stage, in which the algorithm generates a sequence of $\exp\left(\left(\frac{r}{\epsilon}\right)^{s\cdot r\cdot k^r}\right)$ k-way partitions of $V(H)$, such that if H satisfies Ψ then with high probability at least one of these partitions will ϵ-tightly satisfy Ψ. In fact, the partitions are not explicitly written down (this would take $\Omega(n)$ time), but rather *partition oracles* are generated. Every partition oracle specifies the conditions that a given vertex of H must satisfy in order to be mapped to a certain component of the corresponding partition. Roughly speaking, for every vertex $v \in V(H)$ the partition oracle specifies a procedure, which essentially consists of checking the various degrees of v within some small subset $U \subseteq V(H)$, and given the outcome of this procedure it specifies which of the k components should contain v. A more detailed outline of Stage 1 is provided below in Section 5.1.

Stage 2 (running time independent of n) The second part is fairly simple – it takes the sequence of partition oracles generated in Stage 1 and checks if one of them induces a partition that ϵ-tightly satisfies Ψ. This is done by sampling and approximating the densities in a straightforward manner. The probability of error while checking each partition is sufficiently low, so that with high probability a good partition will be detected if and only if the sequence from Stage 1 contained one.

Stage 3 (running time linear in n) The third stage is now trivial. It takes the partition oracle (if found) from Stage 2 and computes the final partition by evaluating it on every $v \in V(H)$. Notice that this part is *not* needed for the testing version of the algorithm (Theorem 2.7).

The main part of the proof deals with showing that if the hypergraph H satisfies Ψ, then with high probability the sequence of partition oracles from Stage 1 will contain an oracle that induces an ϵ-tightly satisfying partition. Intuitively, this is done by showing that there is an ϵ-net of k-way partitions of $V(H)$ (with distance corresponding to Definition 2.3) of size independent of n. In fact, Stage 1 *takes neither n nor H as input*, but just k and the type $(s, r_1, \ldots, r_s)$. Nevertheless, to make the arguments concrete, throughout the proof we will always refer to the input hypergraph H. In the following subsection we give a more detailed outline of Stage 1. Stage 2 is formalized in Lemma 6.4 and its proof is given in Section 6.4.2.

5.1. Outline of Stage 1. The outermost layer in the analysis of Stage 1 borrows from the proof of [16] for graph partitions. Assuming that the hypergraph H admits a partition $\Pi = \{V_1, \ldots, V_k\}$ of the vertices which satisfies Ψ, we generate the partition oracles as follows. First we 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 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 separately 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 can 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 Π_U, the projection of Π on U, we simply try every possible partition of U. Similarly for the second assumption, since we do not know the vector $\bar{\beta}$ of intersection sizes for each cluster, we try all possible vectors up to some allowed deviation. For every such combination of Π_U and $\bar{\beta}$ we get a different *partial* partition oracle, only for the vertices in Y^i. Finally, the sequence of *complete* partition oracles for $V(H)$ will consist of all possible combinations of the partial oracles for the subsets $Y^1, Y^2, \ldots, Y^\ell$. The crucial part is showing that if a satisfying Π as above exists, then one of these partition oracles induces a partition that *loosely* satisfies Ψ. We then show (see Lemma 6.3 and its proof in Section 6.4.1) that a loosely satisfying partition can be easily converted to a tightly satisfying one, with slightly worse parameters.

The intuition behind the correctness of this approach is based on the following central observation. Suppose that $C \subset V$ is a small enough set (say, $|C| = \epsilon n/100$) 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 C arbitrarily, so long as the component sizes are almost preserved (i.e. for all i, $|V'_i \cap C| \approx |V_i \cap C|$). Then the densities of the partition Π' are 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 we first arbitrarily split V into $Y^1, \ldots, Y^l$ and then classify the vertices of each Y^i separately, according to a different sample set U^i.

Although the overall strategy in Stage 1 of the algorithm is similar to the GGR algorithm, there are several differences between our analysis and the original one in [16]. 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. Also, Theorem 2.6 can use a general set of density tensors, rather than intervals of allowed densities as in the GGR algorithm [16] (for the special case of graphs, the original GGR algorithm actually can be naturally extended to include general density tensor sets). This extension of the GGR formalism allows us to use the algorithm more easily and efficiently in our applications.

6. The proof of Theorem 2.6 and Theorem 2.7. We restate here Theorem 2.6 and Theorem 2.7, 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 Comment 6.9 below for a sketch of some optimizations tailored for special cases such as hypergraph colorability).

Theorem 2.6 (Hypergraph Partitioning Algorithm). For every $k \in \mathbb{N}$, hypergraph type $(s, r_1, \ldots, r_s)$ (with $r = \max\{r_i\}$) and set Ψ of density tensors of order k and type $(s, r_1, \ldots, r_s)$ (taken from a family $\mathcal{T}$ efficiently checkable for proximity), there

exists a randomized algorithm A taking as inputs a hypergraph H of type $(s, r_1, \ldots, r_s)$ and two parameters $\epsilon, \delta > 0$ such that

- *the algorithm A either outputs a k-wise partition of $V(H)$, or declares "No Partition";*
- *if H satisfies Ψ then with probability at least $1 - \delta$ the algorithm A outputs a partition of $V(H)$ that ϵ-tightly satisfies Ψ;*
- *the algorithm A outputs a partition of $V(H)$ that* does not *ϵ-tightly satisfies Ψ with probability at most δ. In particular, if H does not ϵ-tightly satisfy Ψ, then the algorithm declares "No Partition" with probability at least $1 - \delta$.*

The running time of A is bounded by

$$\log^3(1/\delta) \cdot \exp\left(\left(\frac{r}{\epsilon}\right)^{s \cdot r \cdot k^r}\right) + n \cdot \mathrm{poly}(k^r, s, 1/\epsilon) \; .$$

Theorem 2.7 (Testing Version of Theorem 2.6). For every $k \in \mathbb{N}$, hypergraph type $(s, r_1, \ldots, r_s)$ (with $r = \max\{r_i\}$) and set Ψ of density tensors of order k and type $(s, r_1, \ldots, r_s)$ (taken from a family $\mathcal{T}$ efficiently checkable for proximity), there exists a randomized algorithm A_T taking as inputs two parameters $\epsilon, \delta > 0$ and an oracle access to a hypergraph H of type $(s, r_1, \ldots, r_s)$, 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-tightly satisfied by H;*
- *if H does not even ϵ-tightly satisfy the property Ψ then with probability at least $1 - \delta$ the algorithm A_T outputs* Reject.

The query complexity of A_T is bounded by $\log^3(\frac{1}{\delta}) \cdot \mathrm{poly}(k^r, s, 1/\epsilon)$, and its running time is bounded by $\log^3(\frac{1}{\delta}) \cdot \exp\left((\frac{r}{\epsilon})^{s \cdot r \cdot k^r}\right)$.

6.1. Proof of Theorem 2.6 and Theorem 2.7. First we define the notion of partitioning oracles, and state the central lemma that implies the proof of Theorem 2.6 and Theorem 2.7.

DEFINITION 6.1. *A (q, c) k-*wise partition oracle *is a randomized algorithm, making edge queries to a hypergraph H and computing a mapping $\pi : V(H) \to [k]$ such that:*

- *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 (q, c) k-wise partition oracles *$\pi : Y \to [k]$ similarly.*

Given a sequence $\{\pi_i\}_{i=1}^m$ of m (q, c) k-wise 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 $\pi_1(v), \pi_2(v), \ldots, \pi_m(v) \in [k]$ requires at most f queries to H in total. Naturally $f \geq q$.*

Now we are ready to state the main lemma required for the proof of Theorems 2.6 and 2.7.

LEMMA 6.2. *For every $k \in \mathbb{N}$ and hypergraph type $(s, r_1, \ldots, r_s)$ (with $r = \max\{r_i\}$) there exists a randomized algorithm A_{R} that on input $\epsilon > 0$ generates a sequence of length m of (f, f) k-wise partition oracles $\{\pi_i\}_{i=1}^m$ such that:*

- *$m = m_{6.2}(\epsilon, s, r, k) = \exp\left(\epsilon^{-s \cdot r \cdot k^r}\right)$ and $f = f_{6.2}(\epsilon, s, r, k) = \mathrm{poly}(k^r, s, 1/\epsilon)$;*
- *the shared query complexity of $\{\pi_i\}_{i=1}^m$ is f;*

- *for any hypergraph H of type $(s, r_1, \ldots, r_s)$ and any set Ψ of density tensors of order k and type $(s, r_1, \ldots, r_s)$, if H satisfies Ψ then with probability at least $\frac{1}{2}$ one or more of the m partition oracles generated by A_{R} induces a partition of $V(H)$ which ϵ-loosely satisfies Ψ.*

The running time of the algorithm A_{R} is bounded by $O(m)$.

Note that the algorithm A_{R} does not depend on any specific H or Ψ at all. It only depends on the general parameters $(s, r_1, \ldots, r_s, \epsilon, k)$ of the problem. Informally, the partition oracles generated by A_{R} will be constructed as follows (although the output of A_{R} does not depend on any specific hypergraph, for simplicity we refer to H as the hypergraph in mind).

- **(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 set of distributions (one distribution for each possible classification of v).

Before proving Lemma 6.2, we state two additional lemmas and through them prove Theorem 2.6 and Theorem 2.7.

LEMMA 6.3. *Let H be a hypergraph and let Ψ be a set of density tensors (both of type $(s, r_1, \ldots, r_s)$). For every $\epsilon > 0$, if H ϵ'-loosely satisfies Ψ, where $\epsilon' \triangleq \frac{\epsilon}{2r}$, then H also ϵ-tightly satisfies Ψ.*

LEMMA 6.4. *Let Ψ be a set of density tensors that is checkable for proximity in time $TC(\Psi)$, and let $\{\pi_i\}_{i=1}^m$ be a sequence of m (q, c) k-wise partition oracles with shared query complexity f. There is a randomized algorithm A_S that on input $0 < \delta, \epsilon < 1$ and an oracle access to a hypergraph H of type $(s, r_1, \ldots, r_s)$ does the following.*

- *If one of the partition oracles π_i defines a partition Π_i of $V(H)$ that $\epsilon/2$-loosely satisfies the property Ψ, then with probability at least $1 - \frac{\delta}{2}$ the algorithm A_S outputs i and a density tensor $\psi \in \Psi$ which is ϵ-loosely equal to ψ^{Π_i} – the true density tensor of Π_i.*
- *If for every π_i the induced partition Π_i does not even ϵ-loosely satisfies Ψ, then the 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 2.6 and Theorem 2.7, and postpone the prof of Lemma 6.3 and Lemma 6.4 to Subsection 6.4.1 and Subsection 6.4.2 respectively.

Proof of Theorem 2.6. First we apply Lemma 6.2 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'-loosely satisfies Ψ. Then we can apply the algorithm from Lemma 6.4 on all m' partitions and write down a $2\epsilon'$-loosely satisfying partition if we find one, and output "No Partition" otherwise. In case we found a $2\epsilon'$-loosely 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 6.3 (see its proof below), so if one of the oracles indeed $2\epsilon'$-loosely satisfies Ψ then with probability $1 - \delta/2$ Lemma 6.4 detected it and hence its corresponding modified partition ϵ-tightly satisfies Ψ. As a conclusion, the time complexity of algorithm A is bounded by

$$O\Big(m' \cdot \Big[f_{6.2}^2 \cdot \Big(\frac{1}{\epsilon'}\Big)^2 \cdot \log\Big(\frac{m'sk^r}{\delta}\Big) \cdot r \cdot s + TC(\Psi)\Big] + n \cdot f_{6.2}\Big) =$$

$$\exp\Big((r/\epsilon)^{s \cdot r \cdot k^r}\Big) \cdot \log^3(1/\delta) + n \cdot \mathrm{poly}(k^r, s, 1/\epsilon)$$

and the probability of success is at least $1 - \delta$, matching the assertions of Theorem 2.6. ∎

Proof of Theorem 2.7. First we apply Lemma 6.2 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'-loosely satisfying partition in the case that H satisfies Ψ. Then we can apply the algorithm from Lemma 6.4 on all m' partitions and output the density tensor $\psi \in \Psi$ if we received one from algorithm A_S. Observe that by Lemma 6.3, the density tensor ψ (in the case that the algorithm A_S did not err) is ϵ-tightly satisfied by H. Hence the total probability of success is at least $1 - \delta$. The time complexity of algorithm A_T is bounded by

$$\exp\Big((r/\epsilon)^{s \cdot r \cdot k^r}\Big) \cdot \log^3(1/\delta)$$

and its query complexity is bounded by

$$\mathrm{poly}(k^r, s, 1/\epsilon) \cdot \log^3(1/\delta)$$

matching the assertions of Theorem 2.7. ∎

6.2. Proof of Lemma 6.2. COMMENT 6.5. *As we mentioned in Section 5, although the sequence of partition oracles generated by A_{R} does not depend on any specific hypergraph or density tensor, it will be convenient to refer to some fixed H and Ψ throughout the proof. The reason for this is that in the proof we mostly argue about the partition oracles themselves (rather than the algorithm A_{R}), whose outputs depend on the hypergraph in mind.*

In order to maintain a small approximation error, the partition oracles generated by algorithm A_{R} are 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 the partial partitioning algorithm for each Y^i, and using it we prove Lemma 6.2.

LEMMA 6.6. *For every $k \in \mathbb{N}$ and hypergraph type $(s, r_1, \ldots, r_s)$ (with $r = \max\{r_i\}$) there exists a randomized algorithm $A_{1/\epsilon}$ that on input $0 < \alpha < \epsilon < 1$ and $n \in \mathbb{N}$ generates a sequence of $\hat{m}$ partial (f, f) k-wise partition oracles $\{\pi_i : [\frac{\epsilon}{8}n] \to [k]\}_{i=1}^{\hat{m}}$ such that:*

- *$\hat{m} = \hat{m}_{6.6}(\epsilon, s, k, r) = \exp\left(\epsilon^{-s \cdot r \cdot k^r}\right)$ and $f = f_{6.6}(\epsilon, s, k, r) = \text{poly}(k^r, s, 1/\epsilon)$;*
- *the shared query complexity of $\{\pi_i\}_{i=1}^{\hat{m}}$ is f;*
- *for any Ψ of order k and type $(s, r_1, \ldots, r_s)$, any hypergraph H of order n and type $(s, r_1, \ldots, r_s)$ having a partition $\Pi = \{V_1, \ldots, V_k\}$ that α-loosely satisfies Ψ, and any subset $Y \subset V(H)$ of size $\frac{\epsilon}{8}n$, with probability $1 - \frac{\epsilon}{16}$ at least one of the $\hat{m}$ 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, $\hat{V}_i = \big((V_i \setminus Y) \cup Y_i\big)$), which $(\alpha + \frac{\epsilon^2}{8})$-loosely satisfies Ψ.*

Furthermore, the running time of $A_{1/\epsilon}$ 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 6.2.

Proof (of Lemma 6.2).. 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-loosely 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-loosely 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 6.6) that Π^1 is a partition that $\alpha + \frac{\epsilon^2}{8} = \frac{\epsilon^2}{8}$-loosely 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 $2 \cdot \frac{\epsilon^2}{8}$-loosely 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) $\frac{8}{\epsilon} \cdot \frac{\epsilon^2}{8} = \epsilon$-loosely 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. ■

6.3. Algorithm $A_{1/\epsilon}$ - Proof of Lemma 6.6.

6.3.1. Overview. The main idea behind the partial partitioning algorithm is the following. Let $\Pi = \{V_1, \ldots, V_k\}$ be the (unknown) partition which loosely 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.

6.3.2. 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 scheme* of the vertex v considering all possible edge sets, places and mappings with respect to the partition $\{W_1, \ldots, W_k\}$. This density scheme 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 schemes). Now we build an auxiliary set of quantitative density schemes. First let $\mathcal{Q}_\epsilon = \left\{0, \frac{\epsilon}{32}, \frac{2\epsilon}{32}, \ldots, 1\right\}$ 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 schemes. The precise clustering of a vertex $v \in Y^0$ is performed as follows. We say that the density scheme $\gamma(v)$ of v is *close* to the cluster $\bar{\alpha} \in \mathcal{A}$ if for every i, ϕ, p we have $\gamma_{i,\phi}^p(v) = \alpha_{i,\phi}^p \pm \epsilon/32$. Note that one density scheme $\gamma(v)$ can be close to multiple clusters. The final clustering will be based on sampled approximate density schemes

of H's vertices (rather than the real ones), 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.

6.3.3. 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 density scheme $\bar{\alpha} = \langle \alpha^p_{i,\phi} \rangle_{i\in[s],p\in[r_i],\phi\in\Phi_i^p}$ such that for all but at most $\xi|Y^{0,\bar{\alpha}}|$ vertices $v \in Y^{0,\bar{\alpha}}$ we have $\forall_{i\in[s],p\in[r_i],\phi\in\Phi_i^p}\ \gamma^p_{i,\phi}(v) = \alpha^p_{i,\phi} \pm \epsilon/32$. 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 6.7. *Let $(Y_1^{0,\bar{\alpha}}, \ldots, Y_k^{0,\bar{\alpha}})$ be any partition of $Y^{0,\bar{\alpha}}$ that for each $1 \le j \le k$ satisfies $\left||V_j \cap Y^{0,\bar{\alpha}}| - |Y_j^{0,\bar{\alpha}}|\right| < \eta|Y^{0,\bar{\alpha}}|$, and let $\hat{\Pi} = \{\hat{V}_1, \ldots, \hat{V}_k\} = \{W_1 \cup Y_1^{0,\bar{\alpha}}, \ldots, W_k \cup Y_k^{0,\bar{\alpha}}\}$ be the redistributed partition of V. Then the following holds.*

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

Proof. The first inequality follows form the definition of the sets $\hat{V}_j$ and fact that $\left||V_j \cap Y^{0,\bar{\alpha}}| - |Y_j^{0,\bar{\alpha}}|\right| < \eta|Y^{0,\bar{\alpha}}|$ for each $1 \le j \le k$, 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 $(\frac{3\epsilon}{16} + \xi + \eta)|Y^{0,\bar{\alpha}}|n^{r_i-1}$. ■

6.3.4. 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 $\epsilon/16$ 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 $\left(\beta_j^{\bar{\alpha}_i}\right)_{i\in[c],j\in[k]}$ of reals such that

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

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 $\sum_h\xi_h|Y^{0,\bar{\alpha}_h}|\leq\frac{\epsilon}{16}|Y^0|$, and denoting by η_h the sum $\sum_{j=1}^{k}\left|\zeta_j^{\bar{\alpha}_h}-\beta_j^{\bar{\alpha}_h}\right|$ 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 $\left(\beta_j^{\bar{\alpha}_i}\right)_{i\in[c],j\in[k]}$, from (ii) we have:

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

and according to (i), by Claim 6.7 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

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

The size of $\mathcal{B}$ is bounded from above by $(32k/\epsilon)^{k\cdot|\mathcal{A}|}=\exp\left(\epsilon^{-s\cdot r\cdot k^r}\right)$. 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 $\left(\beta_1^{\bar{\alpha}},\ldots,\beta_k^{\bar{\alpha}}\right)$, 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 $\bar{\beta}=\left(\beta_1^{\bar{\alpha}},\ldots,\beta_k^{\bar{\alpha}}\right)_{\bar{\alpha}\in\mathcal{A}}\in\mathcal{B}$ which approximates the true intersection values $\left(\zeta_1^{\bar{\alpha}},\ldots,\zeta_k^{\bar{\alpha}}\right)_{\bar{\alpha}\in\mathcal{A}}$ 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 schemes 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 6.8. *There is a (deterministic) algorithm that given subsets $Y^0 \subset V$ of size $\frac{\epsilon}{8}n$, and $U \subset V \setminus Y^0$ of size at most $r \cdot t = r \cdot \frac{2^{16}}{\epsilon^2} \log \frac{l \cdot s \cdot r \cdot k^r}{\epsilon}$, outputs a sequence of $k^{rt} \cdot |\mathcal{B}| = \exp\left(\epsilon^{-s \cdot r \cdot k^r}\right)$ partial k-wise partition oracles $\{\pi_i : Y^0 \to [k]\}_{i=1}^{k^{rt} \cdot |\mathcal{B}|}$ with shared query complexity $s \cdot t$ such that*

- *the partition oracles query only edges contained in $U \cup Y^0$;*
- *if U is chosen at random (by choosing $r \cdot t$ vertices uniformly at random, possibly with repetitions), then for any partition $\Pi = \{V_1, \ldots, V_k\}$ of V, with probability at least $1 - \frac{1}{2l}$ over the choice of U at least one of the partition oracles clusters the vertices of Y^0 so that all but at most $\frac{\epsilon}{16}|Y^0|$ of them are clustered correctly with respect to Π and $\mathcal{A}$.*

6.3.5. Estimating the clusters by sampling – Proof of Lemma 6.8. 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 calculating 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 $1 - \frac{1}{2l}$ the sequence U is such that for all vertices $v \in Y^0$ but at most an $\epsilon/16$ fraction of them, we would get $\hat{\gamma}_{i,\phi}^p(v) = \gamma_{i,\phi}^p(v) \pm \frac{\epsilon}{32}$ 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 reused for all our oracles in all of their oracle queries.

COMMENT 6.9. *In all our treatment, we made no attempt to optimize the dependency functions themselves but only to optimize the function types. However, there are properties for which the requirements on the density tensor ψ^Π are rather sparse, allowing us to ignore some of the values $\gamma^p_{i,\phi}(v)$ in the density scheme for a vertex v. Doing this decreases substantially the number of clusters in the classification, leading to a much decreased time and query complexity.*

Before we continue, this is a good place to sketch how this can be done for the particular property that we use for proving Theorem 1.4. The property Ψ used in its proof (see Section 3) concerns hypergraphs with 4-edges, but these are always with two vertices in one set V_i and the other two vertices in one other V_j. 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 $\hat{k}^{2+o(1)}$.

6.4. The last missing details of the proof.

6.4.1. Proof of Lemma 6.3. In this section we show that any partition Π, which $\epsilon' = \frac{\epsilon}{2r}$-loosely satisfies Ψ, can be turned into a partition which ϵ-tightly 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 an appropriate density tensor in Ψ. The idea behind this observation is that a partition which ϵ'-loosely 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'-loosely satisfies Ψ, let ψ^Π be the density tensor of Π and let $\psi \in \Psi$ be the density tensor which is ϵ'-loosely equal to ψ^Π. By definition,

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

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 $\rho^\Pi_j > \rho_j$, we take $|V^\Pi_j| - \rho_j \cdot n$ vertices from component V^Π_j of Π, and put them into the set S. Then, for each $j \in [k]$ such that $\rho^\Pi_j < \rho_j$, we move $\rho_j \cdot n - |V^\Pi_j|$ vertices from S to the component V_j.

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

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

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 $\mu^{\hat{\Pi}}_{i,\phi} = \mu_{i,\phi} \pm \epsilon$.

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 $|E^{\Pi}_{i,\phi}|$. 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 $|E^{\Pi}_{i,\phi}|$. On the other hand, the number of vertices that were added or removed from each V_{j_h} is at most $\epsilon' n$. Therefore, $\left||E^{\hat{\Pi}}_{i,\phi}| - |E^{\Pi}_{i,\phi}|\right| \leq \epsilon' \cdot r_i \cdot n^{r_i}$, and consequently, $\mu^{\hat{\Pi}}_{i,\phi} = \mu^{\Pi}_{i,\phi} \pm \epsilon/2$. Combining it with $\mu^{\Pi}_{i,\phi} = \mu_{i,\phi} \pm \frac{\epsilon}{2r}$ we conclude that $\mu^{\hat{\Pi}}_{i,\phi} = \mu_{i,\phi} \pm \epsilon$ as required. ■

6.4.2. Testing proximity to Ψ – Proof of Lemma 6.4.

Lemma 6.4 (restated). Let Ψ be a set of density tensors that is checkable for proximity in time $TC(\Psi)$, and let $\{\pi_i\}_{i=1}^m$ be a sequence of m (q,c) k-wise partition oracles with shared query complexity f. There is a randomized algorithm A_S that on input $0 < \delta, \epsilon < 1$ and an oracle access to a hypergraph H of type $(s, r_1, \ldots, r_s)$ does the following.

- *If one of the partition oracles π_i defines a partition Π_i of $V(H)$ that $\epsilon/2$-loosely satisfies the property Ψ, then with probability at least $1 - \frac{\delta}{2}$ the algorithm A_S outputs i and a density tensor $\psi \in \Psi$ which is ϵ-loosely equal to ψ^{Π_i} – the true density tensor of Π_i.*
- *If for every π_i the induced partition Π_i does not even ϵ-loosely satisfy Ψ, then the 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^S_j 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 $\mu^S_{i,\phi}$ 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 is $\frac{3}{4}\epsilon$-loosely equal to some tensor $\psi \in \Psi$, then we output that ψ (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 $\frac{3}{4}\epsilon$-loosely 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 an 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 $O(r \cdot t \cdot f)$ queries.

- For every $i \in [s]$ we split the first $r_i \cdot t$ vertices of T_e into t consequent r_i-tuples $\{\tau^h = (v_1^h, \ldots, v_{r_i}^h)\}_{h=1}^t$, and for every tuple $\tau^h = (v_1^h, \ldots, v_{r_i}^h)$ we check if $(v_1^h, \ldots, v_{r_i}^h) \in E_i$. This requires $O(t \cdot s)$ queries.

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

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

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

$$\mu_{i,\phi}^S = \frac{1}{t}\Big|\Big\{\tau^h = (v_1^h, \ldots, v_{r_i}^h) : \Big((v_1^h, \ldots, v_{r_i}^h) \in E_i\Big) \wedge \Big(\forall_{j \in [r_i]} \pi(v_j) = \phi(j)\Big)\Big\}\Big|$$

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 $\tau^h = (v_1^h, \ldots, v_{r_i}^h)$ from T_e, $i \in [s]$ and $\phi \in \Phi_i$,

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

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.

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

as required.

Since the comparison 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

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

■

7. Concluding Remarks and Open Problems.

Strong hypergraph regularity:. Very recently, a lot of attention was given to obtaining hypergraph regularity lemmas [17, 23, 24, 27] that are strong enough to reprove Szemerédi's number-theoretic theorem [25] 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 partitions of pairs and beyond, but (regrettably) 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 (i.e. the density restriction types that we can check for) is not strong enough to match the one typically required for a strong hypergraph regularity lemma. It would be very interesting to obtain a generalization of Theorem 2.6 that will be strong enough to yield algorithmic versions of the results of [17, 23, 24, 27].

Improving the output for hypergraph regularity:. While we can apply Theorem 2.6 in a simple way to obtain an algorithmic version of the regularity lemma for graphs (given in Theorem 1.4), the generalization for hypergraphs (Theorem 4.2) 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 4.2 similar in nature to Theorem 1.4, 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 started out from discussions during a DIMACS Workshop[12], and we would like to thank the organizers for providing us with this opportunity. We also thank the editor and anonymous referees for their useful comments.

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] G. Andersson and L. Engebretsen, Property testers for dense constraint satisfaction programs on finite domains, Random Structures and Algorithms 21 (2002), 14-32.

[6] 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.

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

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

[12] *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.

[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, SIAM J. on Computing, 37 (2007), 482-501.
[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] B. Nagle, V. Rödl and M. Schacht, The counting lemma for regular k-uniform hypergraphs, Random Structures and Algorithms 28 (2006), 113-179.
[24] V. Rödl and J. Skokan, Regularity lemma for k-uniform hypergraphs, Random Structures and Algorithms 25 (2004), 1-42.
[25] E. Szemerédi, Integer sets containing no k elements in arithmetic progression, Acta Arith. 27 (1975), 299-345.
[26] 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.
[27] T. Tao, A variant of the hypergraph removal lemma, J. Combin. Theory, Ser. A 113 (2006), 1257-1280.