Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Code Sparsification and its Applications*

Sanjeev Khanna† Aaron (Louie) Putterman‡ Madhu Sudan§

November 6, 2023

Abstract

We introduce a notion of *code* sparsification that generalizes the notion of cut sparsification in graphs. For a (linear) code $\mathcal{C} \subseteq \mathbb{F}_q^n$ of dimension k a $(1 \pm \epsilon)$-*sparsification* of size s is given by a weighted set $S \subseteq [n]$ with $|S| \leq s$ such that for every codeword $c \in \mathcal{C}$ the projection $c|_S$ of c to the set S has (weighted) hamming weight which is a $(1 \pm \epsilon)$ approximation of the hamming weight of c. We show that for every code there exists a $(1 \pm \epsilon)$-sparsification of size $s = \widetilde{O}(k \log(q)/\epsilon^2)$. This immediately implies known results on graph and hypergraph cut sparsification up to polylogarithmic factors (with a simple unified proof) — the former follows from the well-known fact that cuts in a graph form a linear code over $\mathbb{F}_2$, while the latter is obtained by a simple encoding of hypergraph cuts. Further, by connections between the eigenvalues of the Laplacians of Cayley graphs over $\mathbb{F}_2^k$ to the weights of codewords, we also give the first proof of the existence of spectral Cayley graph sparsifiers over $\mathbb{F}_2^k$ by Cayley graphs, i.e., where we sparsify the set of generators to nearly-optimal size. Additionally, this work can be viewed as a continuation of a line of works on building sparsifiers for constraint satisfaction problems (CSPs); this result shows that there exist near-linear size sparsifiers for CSPs over $\mathbb{F}_p$-valued variables whose unsatisfying assignments can be expressed as the zeros of a linear equation modulo a prime p. As an application we give a full characterization of ternary Boolean CSPs (CSPs where the underlying predicate acts on three Boolean variables) that allow for near-linear size sparsification. This makes progress on a question posed by Kogan and Krauthgamer (ITCS 2015) asking which CSPs allow for near-linear size sparsifiers (in the number of variables).

At the heart of our result is a codeword counting bound that we believe is of independent interest. Indeed, extending Karger's cut-counting bound (SODA 1993), we show a novel decomposition theorem of linear codes: we show that every linear code has a (relatively) small subset of coordinates such that after deleting those coordinates, the code on the remaining coordinates has a smooth upper bound on the number of codewords of small weight. Using the deleted coordinates in addition to a (weighted) random sample of the remaining coordinates now allows us to sparsify the whole code. The proof of this decomposition theorem extends Karger's proof (and the contraction method) in a clean way, while enabling the extensions listed above without any additional complexity in the proofs.

*The full version of the paper can be accessed at `https://arxiv.org/pdf/2311.00788.pdf`

†School of Engineering and Applied Sciences, University of Pennsylvania, Philadelphia, PA. Email: `sanjeev@cis.upenn.edu`. Supported in part by NSF awards CCF-1934876 and CCF-2008305.

‡School of Engineering and Applied Sciences, Harvard University, Cambridge, Massachusetts, USA. Supported in party by the Simons Investigator Fellowship of Boaz Barak, NSF grant DMS-2134157, DARPA grant W911NF2010021, and DOE grant DE-SC0022199. Supported in part by the Simons Investigator Award of Madhu Sudan and NSF Award CCF 2152413. Email: `aputterman@g.harvard.edu`.

§School of Engineering and Applied Sciences, Harvard University, Cambridge, Massachusetts, USA. Supported in part by a Simons Investigator Award and NSF Award CCF 2152413. Email: `madhu@cs.harvard.edu`.

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

1 Introduction.

In this work we introduce a notion of sparsification for linear codes, prove that "nearly linear-size" sparsifications exist for every linear code, and give some applications beyond coding theory. We start by recalling the notion of sparsification and give some background.

Sparsification of data with respect to a certain class of queries alludes to a compression of the data that allows all queries in the class to be answered (or estimated) correctly. It has emerged as a fundamental concept in theoretical computer science, with graph sparsification for cut-queries being a central example. In seminal works, Karger [13] and Benczúr and Karger [2] showed how graphs may be sparsified by carefully sampling a subset of its edges and assigning weights to them, so that for every cut, the cut-size in the sampled weighted graph gives a good estimate of the cut size in the input graph. Crucially the number of sampled edges was nearly linear-sized in the number of vertices of the graph. This work led to many strengthenings (in particular to allowing the sample to be linear sized [1]), extensions (in particular to more general spectral notions of sparsification [1, 22], to hypergraphs [5, 11, 17]), more recently to sparsifying sums of norms [9], and applications to a host of problems including solvers for max-flow and min-cut [15, 20, 21], as well as to better solvers for structured linear systems [6, 10, 18, 19, 22] and applications to clustering [4]. In the streaming and sketching settings, small representations of graphs are equally important and work on graph sparsifiers has played a key role.

A natural question that arises is which other classes of structures and queries allow for such sparsification. In this work we explore this question in a new terrain, namely linear codes, where the queries specify a message and the goal is to estimate the Hamming weight of its encoding under the linear code. We describe our setting formally first before motivating the problem.

1.1 Code Sparsification.

Throughout this paper q will be a prime power and $\mathbb{F}_q$ will denote the finite field on q elements. A linear code $\mathcal{C}$ is a $\mathbb{F}_q$-linear subspace of $\mathbb{F}_q^n$, and we will assume it is the image of a linear map $E : \mathbb{F}_q^k \to \mathbb{F}_q^n$. The Hamming weight of a vector $v \in \mathbb{F}_q^n$, denoted $\mathrm{wt}(v)$ is the number of non-zero coordinates of v. Given a sequence of non-negative (integer) weights $w = (w_1, \ldots, w_n)$, the weighted Hamming weight of $v = (v_1, \ldots, v_n)$, denoted $\mathrm{wt}_w(v)$, equals $\sum_{i|v_i \neq 0} w_i$. For a vector $v \in \mathbb{F}_q^n$ and set $S \subseteq [n]$ the puncturing of v to the set S, denoted $v|_S$ is the vector $(v_i)_{i \in S}$. The puncturing of a code $\mathcal{C} \subseteq \mathbb{F}_q^n$ to the coordinates $S \subseteq [n]$ is the code $\mathcal{C}|_S \subseteq \mathbb{F}_q^{|S|}$ given by $\mathcal{C}|_S = \{v|_S : v \in \mathcal{C}\}$. We are now ready to define code sparsifiers.

Definition 1.1. (Code Sparsifier) *For integer s, real $\epsilon > 0$ and a linear code $\mathcal{C} \subseteq \mathbb{F}_q^n$, a $(1 \pm \epsilon)$-sparsifier of size s for the code $\mathcal{C}$ is a subset $S \subseteq [n]$ with $|S| \leq s$ along with weights $w_S = (w_i)_{i \in S}$ such that for every codeword $v \in \mathcal{C}$, we have*

$$(1 - \epsilon)\mathrm{wt}(v) \leq \mathrm{wt}_w(v|_S) \leq (1 + \epsilon)\mathrm{wt}(v).$$

(In other words the weighted Hamming weight of every codeword in the punctured code roughly equals its weight in the unpunctured code.)

The vanilla representation of a linear code would involve kn elements of $\mathbb{F}_q$. The sparsification reduces the representation size to sk field elements which may be significantly smaller if $s \ll n$. In several applications we consider later, the code $\mathcal{C}$ itself is obtained by puncturing a known fixed mother code $M \subseteq \mathbb{F}_q^N$. In such cases the vanilla representation of $\mathcal{C}$ (to someone who knows M) would require $n \log N$ bits while the sparsification would require only $s \log N$ bits to describe. Thus in both cases the sparsification definitely compresses the representation of $\mathcal{C}$. And if we fix any linear encoding scheme $E : \mathbb{F}_q^k \to \mathbb{F}_q^n$ such that $\mathcal{C} = \{E(m) | m \in \mathbb{F}_q^k\}$, then the sparsification allows us to estimate the hamming weight of the encoding $E(m)$ of every message $m \in \mathbb{F}_q^k$. Thus, the definition of code sparsifiers fits the general notion of sparsification, and so we turn to the motivation for studying this concept.

1.2 Motivation.

Our initial motivation for studying code sparsification is that it abstracts and generalizes cut sparsification in graphs. Specifically for every graph there is a linear code over $\mathbb{F}_2$ such that codewords of this code are indicator vectors of the edges crossing cuts in the graph. (This code is obtained by viewing the edge-vertex incidence matrix as the generator of the code.) Thus, a sparsifier for this code corresponds to a cut sparsifier for the associated graph. Existence of graph sparsifiers is typically proved by combinatorial or spectral analysis — tools that are less amenable to application over codes. Thus the exploration of code sparsification forces us to revisit methods for constructing graph sparsifiers and extract the essential elements in this toolkit,

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

One broad class of sparsifiers that overlap significantly with code sparsifiers are *CSP sparsifiers*, introduced by Kogan and Krauthgamer [17] and studied further by Filtser and Krauthgamer [7] and Butti and Zivný [3]. Constraint Satisfaction Problems (CSPs) have as instances n constraints on k variables where each constraint operates on a constant number of variables (called the arity of the constraint). A CSP sparsifier aims to compress an instance of the CSP into a smaller (weighted) one on the same set of variables such that for every assignment to the variables, the sparsified CSP satisfies roughly the same number of constraints as the original one. When the variables take on values in a finite set $\mathbb{F}_q$ and the constraints are linear constraints over $\mathbb{F}_q$, then the sparsification task is a code sparsification task. (Note that code sparsification allows q as well as the arity of the constraints to be non-constant and so code sparsification is not a subclass of CSP sparsification.) In particular, cut sparsification is also a special case of CSP sparsification. Prior work had shown how to get nearly linear size sparsifiers for CSPs beyond cut sparsifiers. Specifically, in [17], it was shown that r-SAT instances on a universe of k Boolean variables admit sparsifiers of size $\widetilde{O}(kr/\varepsilon^2)$. This was improved to $\widetilde{O}(k/\varepsilon^2)$ by Chen, Khanna and Nagda [5], but the family of CSPs for which this result holds was not broadened. The works [3, 7] completely classify all binary CSPs (i.e., with arity two), that allow for nearly linear size sparsification, but a classification beyond $r = 2$ remains wide open. Indeed [7], pose this as an open question, and [3] highlight the challenge of sparsifying r-XOR CSPs (for $r \geq 3$) as a central problem that remains unaddressed by graph and hypergraph sparsification techniques. Thus, code sparsification seems like the natural next frontier in CSP sparsification and worthy of further attention.

Finally we also give some new applications of code sparsification in this paper itself. In particular, we give a simple reduction from hypergraph cut sparsification to code sparsification that makes the former a special case of the latter (although there may be some polylogarithmic losses in the size of the sparsification). We also show how code sparsification can be used to derive structured sparsification of Cayley graphs over $\mathbb{F}_2^k$, by other Cayley graphs! We elaborate on these new connections and their implications after describing our main results.

1.3 Main Results. Our main theorem in this paper shows that every (possibly weighted) linear code $\mathcal{C} \subseteq \mathbb{F}_q^n$ of dimension k has a nearly-linear sized sparsifier, i.e., one of size $\widetilde{O}(\frac{k}{\epsilon^2} \log q)$.[1]

THEOREM 1.1. *For every $\epsilon > 0$, prime power q, positive integers k and n, every (possibly weighted) linear code $\mathcal{C} \subseteq \mathbb{F}_q^n$ of dimension k has a $(1 \pm \epsilon)$-code sparsifier of size $\widetilde{O}\left(\frac{k}{\epsilon^2} \log q\right)$.*

Note that the theorem is essentially optimal up to polylogarithmic factors in k (for constant ϵ and q) in that codes require $\Omega(k)$-sized sparsifiers. Note also that our result qualitatively reproduces the existential part of the sparsification result in [2] while extending it vastly. Of particular interest is the fact that our result does not require the generator matrix of the code to be sparse, something that was evidently true of all previous works on sparsification, and potentially used as a core ingredient in many proofs. In fact, the theorem as stated above is completely independent of the choice of the generator matrix of the code, whereas previous analyses even for the graph-theoretic codes seem to rely on the use of a specific generator matrix.

A central tool in the cut sparsifiers of [2, 8, 13, 14] is "Karger's cut counting bound" [12, 14] which asserts that every graph on k vertices whose minimum cut is c has at most $k^{2\alpha}$ cuts of size at most αc, for every integer α. This bound can be interpreted in coding terms — the minimum cut size is the minimum distance of the corresponding code, and distance is of course a central concept in coding theory. However the lemma is patently false for general codes. Specifically for codes of minimum distance $\Omega(n)$ the bound would suggest that there are at most $n^{O(1)}$ codewords in the code and every asymptotically good code (those which $k = \Omega(n)$) are counterexamples to this potential extension. In view of the centrality of this bound though, it is natural to ask what weaker bound one can get for general codes. Our next theorem gives a simple weakening that essentially suggests that the only counterexamples come from "good" codes embedded in $\mathcal{C}$.

THEOREM 1.2. *For every prime power q, parameters d, k and n and every linear code $\mathcal{C} \subseteq \mathbb{F}_q^n$, the following holds: there exists a subset $T \subseteq [n]$ with $|T| \leq k \cdot d$ such that for $S = [n] \setminus T$ the code $\mathcal{C}|_S$ satisfies the condition that for every integer $\alpha \geq 1$ the code $\mathcal{C}|_S$ has at most $q^\alpha \cdot \binom{k}{\alpha}$ codewords of weight at most αd.*

Note that in the above theorem, d does not refer to the distance, but rather is a parameter of our choosing.

[1] In this paper we use the notation $\widetilde{O}(\cdot)$ to hide poly logarithmic factors in the argument.

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

In other words while $\mathcal{C}$ may have many relatively small weight codewords, they come from a sub-code $\mathcal{C}|_T$ contained on a small set of coordinates while the rest of the code $\mathcal{C}|_S$ has a smooth growth in the number of codewords of small weight. We note that this basic theorem about linear spaces does not seem to have been noticed before and could be of independent interest.

While it is immediate that Theorem 1.2 can be used to get some sparsification for some codes, it is not clear how to use it to go all the way to Theorem 1.1. Indeed in the case of graph sparsification for preserving cuts, known proofs utilize additional notions from graph theory to identify importance of a coordinate (edge). For instance, [2] utilizes the notion of edge strengths while [8] relies on edge connectivity to determine sampling probabilities, and in both cases, the analysis uses graph-theoretic structure to establish correctness of the resulting sparsifiers. We show nevertheless that a simple recursive scheme can be applied to sparsify every code. Indeed even the specialization of this proof to the graph-theoretic case of cut sparsifiers seems new and we describe this simpler proof in the appendix of the full version [16].

We remark that one weakness of our results (or a major open question) is that our results are existential and we do not have efficient algorithms to produce the sparsifiers that we show exist. The difficulty roughly emerges from the difficulty of finding and counting low weight codewords in a code which are known hard problems in coding theory.

1.4 Applications.

Hypergraph Cut Sparsification. A cut sparsifier for a hypergraph is a simple extension of the notion of a cut sparsifier for graphs. Specifically it is a weighted subgraph of the input hypergraph such that for every 2-coloring of the vertices, the number of bichromatic edges in the original hypergraph is approximately the same as the weight of the bichromatic edges in the subgraph. Previous works by Kogan and Krauthgamer [17] (in the constant arity hyperedge case) and ultimately Chen, Khanna and Nagda [5] (in the unbounded arity case) have given cut-sparsifiers of size $O(k\log(k)/\epsilon^2)$ for every hypergraph on k vertices. We are able to recover their result qualitatively (up to polylogarithmic factors in k and $1/\epsilon$) with a very simple reduction. (Specifically we note that if we choose q to be a large enough prime and associate an r-vertex hyperedge with the vector $(q-r+1,1,\ldots,1,0,\ldots,0) \in \mathbb{F}_q^k$ then the only coordinates in encodings of messages in $\{0,1\}^k$ that are 0 are the monochromatic edges.) See Remark 8.1 for more details. Indeed by applying this reduction to Theorem 1.2 we also obtain a structural decomposition theorem for hypergraphs that does not seem to have been noticed before.

THEOREM 1.3. *For every integer $d \geq 1$, every hypergraph H on k vertices has a set of at most kd hyperedges such that upon their removal, the resulting hypergraph satisfies the condition that for every integer $\alpha \geq 1$ it has at most $(2k)^{2\alpha}$ cuts of size $\leq \alpha d$.*

Indeed, as in the setting of codes, an analog of "Karger's cut-counting bound" does not hold in the realm of hypergraphs [17]. Thus, our analysis of code sparsifiers provides a more universal counting bound which decomposes codes and hypergraphs alike.

Cayley Graph Sparsifiers. A well-studied notion extending that of a cut-sparsifier for graphs is a spectral sparsifier. Formally a spectral sparsifier of a graph is a weighted subgraph whose Laplacian has eigenvalues close to that of Laplacian of the original graph. (The Laplacian of a graph $G = (V, E)$, denoted L_G, is a $|V| \times |V|$ matrix, whose diagonal entries $L_{G_{i,i}}$ are the degrees of the ith vertex, and whose off diagonal entries $L_{G_{i,j}}$ are $-w_{i,j}$ where w_{ij} is the weight of the edge (i,j) in G.) Informally, a spectral sparsifier allows us to estimate the quadratic form $x^T L_G x$ for every real vector x, whereas a cut-sparsifier allows us to estimate this form only for $x \in \{0,1\}^{|V|}$. Most of the results in this paper only extend the notion of cut-sparsifiers, but not spectral sparsifiers. The only exception is for spectral sparsifiers of "Cayley graphs" on $\mathbb{F}_2^k$. In this special setting the vertex set of the Cayley graph is $\mathbb{F}_2^k$, and the edges are specified by a "generating" set $\Gamma \subseteq \mathbb{F}_2^k$. Two vertices $x, y \in \mathbb{F}_2^k$ are adjacent if $x - y \in \Gamma$.

While the general theory of spectral sparsification of course holds for Cayley graphs, this may not lead to a compressed representation of the graph, since the generating set Γ can be much smaller than $|V|$ (and Γ specifies the Cayley graph completely). A natural question in this context would be whether there can be a compression of Cayley graphs that is also a Cayley graph (so leads to a compressed representation of the original graph). While this question remains open for general groups, in the setting of $\mathbb{F}_2^k$, our main theorem, Theorem 1.1 effectively resolves this positively.

A folklore connection between the eigenvectors of the Cayley graphs and the code generated by Γ (where Γ

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

is viewed an $n \times k$ matrix over $\mathbb{F}_2$ whose columns generate a code contained in $\mathbb{F}_2^n$) allows us to show that the weight distribution of a code-sparsifier of Γ closely matches that of Γ, and so leads to a new generating set for the Cayley graph with nearly matching eigenvalue profile. This leads to the following theorem, whose proof may be found in § 6.

THEOREM 1.4. (CAYLEY GRAPH SPECTRAL SPARSIFIER) *For every (possibly weighted) Cayley graph G on $\mathbb{F}_2^k$ with generating set $\Gamma \subseteq \mathbb{F}_2^k$, there exists a weighted sparsifier $\hat{\Gamma} \subseteq \Gamma$, such that for the Cayley graph $\hat{G}$ generated by $\hat{\Gamma}$,*

$$(1-\varepsilon)L_G \preceq L_{\hat{G}} \preceq (1+\varepsilon)L_G.$$

Further, $|\hat{\Gamma}| \leq \widetilde{O}(k/\varepsilon^2)$.

We stress that this is the first existential result of its kind. For comparison, the work of [1] showed the existence of spectral sparsifiers for any graph on n vertices to size $O(n/\varepsilon^2)$. In the setting of Cayley graphs, this implies the existence of spectral sparsifiers of Cayley graphs over $\mathbb{F}_2^k$ with $O(2^k/\varepsilon^2)$ edges, leading to an average degree which is approximately $O(1/\varepsilon^2)$. *However,* the key distinction is that the sparsifier returned by [1] is *not* guaranteed to still be a Cayley graph and indeed it can not be. In fact, even just to maintain connectivity, a Cayley graph on $\mathbb{F}_2^k$ requires $\Omega(k)$ generators. Our sparsifiers have degree $\widetilde{O}(k)$ (for constant ϵ), but now our resulting sparsified graph is a Cayley graph.

CSP Sparsification. As described earlier, cut sparsification can be viewed as a special case of CSP sparsification (corresponding to a CSP where constraints apply to two binary variables and require that their XOR be 1). Furthermore when restricted to fields of constant size and generator matrices with exactly r non-zero elements per row (where the columns of the generator matrix generate the code), the code sparsification problem is also a special case of CSP sparsification. Thus this interpretation already leads to a new broad class of CSPs that admit nearly linear sparsifiers.

We say that a predicate $P : \mathbb{F}_q^r \to \{0,1\}$ is an *affine* predicate if there exist elements $a_0, a_1, \ldots, a_r \in \mathbb{F}_q$ such that $P(b_1, \ldots, b_r) = 0$ if and only if $a_0 + \sum_i a_i b_i = 0$ (over $\mathbb{F}_q$). Equivalently, the predicate $P(b_1, \ldots b_r)$ is evaluating $\sum_i a_i b_i \neq -a_0$.

Now, let $\mathcal{P}$ be a collection of predicates of any arity, and define CSP($\mathcal{P}$) to be the family of CSPs where each constraint is a predicate from $\mathcal{P}$ applied to any appropriately sized tuple of variables. The following theorem asserts that CSPs over affine predicates are sparsifiable.

THEOREM 1.5. *Let $\mathcal{P} = \{P : P$ is an affine predicate over $\mathbb{F}_q\}$. Then, any CSP in CSP($\mathcal{P}$) admits a nearly linear size $(1 \pm \varepsilon)$ sparsification, namely of size $\widetilde{O}_q(k/\epsilon^2)$, where k denotes the number of variables in the instance (and $O_q(\cdot)$ hides factors of q).*

Note that the above theorem has no dependence on the value r, and in fact, we can sparsify affine predicates even when $r = k$, and simultaneously sparsify affine predicates of different arities as long as they are affine with respect to the same field $\mathbb{F}_q$.

One immediate application of the above theorem is to any r-XOR constraint. Indeed, the unsatisfying instances of an XOR constraint form a linear subspace over $\mathbb{F}_2$, and so we give the first proof of the sparsifiability of XOR constraints to nearly linear size. This addresses one of the open questions of [3], who showed the fundamental inexpressability of XOR constraints in terms of hypergraphs.

To illustrate the power of the theorem above, we also extend a result of [7] to predicates on 3 Boolean variables, giving an exact classification of which Boolean predicates on up to three variables are sparsifiable to near-linear size. We say that a predicate $P : \{0,1\}^r \to \{0,1\}$ has *an affine projection to AND* if there exists a function $\pi : [r] \to \{0, 1, x, \neg x, y, \neg y\}$ such that $AND(x, y) = P(\pi(1), \ldots, \pi(r))$.

THEOREM 1.6. *For a predicate $P : \{0,1\}^3 \to \{0,1\}$, all possible CSPs of P on subsets of k variables are $(1 \pm \varepsilon)$ sparsifiable to size $\widetilde{O}(k/\varepsilon^2)$ if and only if P has no affine projection to AND.*

We remark that this classification does not yet extend to arbitrary ternary predicates (specifically over non-Boolean variables), and thus does not extend the result in [3].

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

1.5 Proof Techniques. In our view, one of the strengths of this paper is that the proofs are conceptually simple and short even as they generalize known results and provide some new applications. Indeed once we determine the right form of the weight counting bound, namely Theorem 1.2 (later as Theorem 2.2), its proof is not very hard.

Roughly, we build a contraction procedure in linear spaces analogous to Karger's contraction method in [12, 14] in graphs. It is not immediately clear what should be the appropriate analog to Karger's contraction in our setting. Karger's method is inherently very reliant on the underlying graph structure, as each contraction "merges" two vertices with an edge between them. In a general linear code, while one can think of each row of the generating matrix as corresponding to an edge, and each column of the generating matrix as corresponding to a vertex, the analogy quickly breaks down as each row can be "involved" with many columns. As such, the perspective we end up taking for "contracting" the generating matrix (and one that extends to the graphical case for edges) is a contraction on a single coordinate j of the generating matrix which is not always 0. We decompose the entire column space of the generating matrix into the entire linear subspace of codewords ($\subseteq \mathbb{F}_q^n$) which is zero on this coordinate (i.e. such that the generating matrix for this subspace is entirely 0 in the jth row), and a single vector which is non-zero in the jth coordinate. The original linear space is the span of these two separate components, and for us, contracting on a coordinate corresponds to keeping only the linear subspace which is zero on this coordinate.

Using this perspective, we show that this contraction procedure when applied to a random, not constantly 0 coordinate of the code (which corresponds to an edge in the underlying graph in Karger's result) is likely to keep low-weight codewords intact. Thus if we focus on a particular low-weight codeword, repeated application of the contraction procedure should output this codeword with probability at least $q^{-\alpha}\binom{k}{\alpha}^{-1}$ *as long as the support of our code never gets too small.* We use support here to refer to the total number of remaining not all zero rows of the generating matrix, or equivalently the number of coordinates in the code which are not always 0.

While Karger's analysis is able to explicitly lower bound the support size (in his case the number of remaining edges) at every iteration in the algorithm by considering the minimum cut size, we can do no such thing as there can exist codes with large dimension and small support contained in our code. Instead, we build a parameterized trade-off saying that whenever our algorithm is unable to make progress in the contraction process, it is because there is a non-trivially high-dimensional code contained in our code that is supported on relatively few coordinates. (See Theorem 2.1 for a precise formulation.) Removing this code and the coordinates it is supported on, and continuing leads to a proof of Theorem 1.2.

The proof of Theorem 1.1 given Theorem 1.2 is also not hard, and we believe this is a simpler proof than previous sparsification results for special cases of graphs and hypergraphs. To prove the existence of near-linear size code sparsifiers, suppose we start with a code $\mathcal{C}$ of dimension k, and length k^2. We first invoke the above theorem with $d = \sqrt{k}$. Intuitively, this breaks the code into two parts: In one part, we have no guarantee on the distribution of weights of the codewords, but the support size is bounded by $k \cdot d = k^{3/2}$. In the other part, we have a strong bound on the distribution of weights of codewords, but the support can still be as large as k^2. However, the fact that this code has roughly at most k^α codewords of weight $\alpha\sqrt{k}$ suggests that if we sample $k^{3/2}$ coordinates of this code and give each a weight of $\sqrt{k}$ then we get a pretty good sparsification of this part of the code. Gluing the two parts together gives an $O(k^{3/2})$ size sparsifier of the whole code. To get better sparsifications we can continue to apply this procedure recursively. For instance, if we repeat the process one more level (separately on each of the sparsified codes above), we can now use $d = k^{1/4}$ instead of $d = \sqrt{k}$ and this leads to 4 codes, each of size roughly $k^{5/4}$. Likewise, we can glue these 4 codes together, yielding an $O(k^{5/4})$ size sparsifier. Repeating enough times, and optimizing the parameters carefully leads to the final result. There are additional caveats when moving to codes whose length is super-polynomial in their dimension, and for these we utilize a few additional ideas to get our final sparsification result.

1.6 Organization. In § 2, we introduce an analog of Karger's contraction algorithm using Gaussian elimination to prove our decomposition theorem (Theorem 1.2) for linear codes. Then, in § 3 we introduce some basic facts and definitions about codes, graphs, and CSPs, which we will utilize in our construction of code sparsifiers and their applications. In § 4 we formalize the argument we gave above regarding sparsifying any code whose length is polynomial in its dimension. Finally, in § 5, we use the algorithm from § 4 as a sub-routine along with several code decomposition tricks to remove any dependence on the length of the code, and present the existence of near-linear size sparsifiers for all linear codes, proving Theorem 1.1. In § 6, we prove Theorem 1.4 by a simple

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

reduction to codes. In § 7 we prove Theorem 1.5 and Theorem 1.6, extending the classification of near-linearly sparsifiable CSPs again by reducing to the coding case. Finally, in § 8, we interpret hypergraphs in the setting of codes, proving our decomposition result, namely Theorem 1.3.

2 A Counting Bound for Codewords.

2.1 Preliminaries. First, we introduce a few basic definitions. These definitions will be used throughout the paper and are all that is required for this section.

DEFINITION 2.1. *A linear code $\mathcal{C}$ of dimension k and length n is a k-dimensional linear subspace of $\mathbb{F}_q^n$. We often associate with $\mathcal{C}$ a generator matrix $G \in \mathbb{F}_q^{n\times k}$, which maps vectors $x \in \mathbb{F}_q^k$ to codeword $\in \mathbb{F}_q^n$.*

DEFINITION 2.2. *For a codeword $c \in \mathbb{F}_q^n$, the weight of a codeword is the number of non-zero entries in c. This is will be denoted as* $\mathrm{wt}(c)$.

DEFINITION 2.3. *For a linear code $\mathcal{C} \subseteq \mathbb{F}_q^n$, we say the support size is*

$$\mathrm{Supp}(\mathcal{C}) = |\{i \in [n] : \exists c \in \mathcal{C} \text{ such that } c_i \neq 0\}|.$$

*In words, it is the number of coordinates of the code which are not always zero. Likewise, we say that for a code $\mathcal{C} \subseteq \mathbb{F}_q^n$ along with a generating matrix $G \in \mathbb{F}_q^{n\times k}$, a coordinate $j \in [n]$ is **non-zero** if there exists a codeword $c \in \mathcal{C}$ such that $c_j \neq 0$.*

DEFINITION 2.4. *For a linear code $\mathcal{C} \subseteq \mathbb{F}_q^n$, a **subcode** of $\mathcal{C}$ is a linear subspace of $\mathcal{C}$.*

DEFINITION 2.5. *For a linear code $\mathcal{C} \subseteq \mathbb{F}_q^n$, we say that the **density** of a code is*

$$\mathit{Density}(\mathcal{C}) = \frac{\mathit{Dim}(\mathcal{C})}{\mathrm{Supp}(\mathcal{C})}.$$

We say that the density of the empty code is 0.

2.2 Overview. In his seminal work, Karger ([12, 14]) showed that for a graph on k vertices with minimum cut-value c, there are at most $k^{2\alpha}$ cuts with size $\leq \alpha c$, for any integer α. As discussed in the introduction, while an analogous statement does not hold for codes in general, we are able to prove a generalization that establishes a bound on the distribution of codewords. Roughly speaking, we show that for a code $\mathcal{C} \subseteq \mathbb{F}_q^n$ of dimension k and a parameter $d \in \mathbb{Z}^+$ of our choosing, either there exists a dense subcode of $\mathcal{C}$ contained on a small support (i.e. a subcode whose density is $\geq \frac{1}{d}$), or the code satisfies a Karger-style bound on the distribution of codeword weights with parameter d (i.e. there are at most $(qk)^\alpha$ codewords of weight $\leq \alpha d$). An exact statement of this result is given in Theorem 2.1. We will then show that Theorem 2.1 in fact implies Theorem 1.2 described in the introduction; that is, there exists a set of at most kd coordinates, such that upon removing these coordinates (equivalently removing the corresponding rows from the generating matrix), the resulting code satisfies the Karger-style bound with parameter d. An exact statement is provided in Theorem 2.2.

We will prove this bound by describing a "contraction" algorithm akin to that of Karger. Intuitively, the algorithm takes in a generating matrix of a code of dimension k and length n. At random, the algorithm chooses one of the non-zero coordinates of this generating matrix and performs Gaussian elimination to "zero" out all but one of the columns in this coordinate. Then, the algorithm removes the remaining column of the generating matrix that is non-zero in this coordinate, reducing the dimension of the generating matrix by 1. This process is repeated until the dimension of the generating matrix is sufficiently small.

We call this Gaussian elimination step on a random non-zero coordinate of the generating matrix a "contraction". We will show that as long as the support of the generating matrix is sufficiently large in every iteration, then any low-weight codeword will "survive" (i.e. remain in the span of the generating matrix) after many contractions with high probability. We can then conserve probability mass to argue that in fact, under the condition that the support is never too small, the number of lightweight codewords cannot be too large. This naturally leads to the statement of Theorem 2.1 as either there exists a subcode of sufficiently high dimension with small support or during our contraction algorithm, the support is always large enough to get a strong bound on the number of codewords.

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

REMARK 2.1. *When the mother code is the cut-code of a graph, the procedure of choosing a random row and eliminating all but* 1 *of the non-zero entries is in fact equivalent to Karger's contraction based algorithm ([12,14]).*

First, we will prove some facts about the following algorithm, which takes as input a generating matrix $G \in \mathbb{F}_q^{n \times k}$ for a code of dimension k:

Algorithm 1: Contract(G, α)

Let G_i be the ith column of G, and let k be the number of columns of G, and n the number of rows.
while $dim(G) \geq \alpha + 1$ **do**
  Choose a random non-zero coordinate $j \in [n]$ of G.
  Let G_a be the first column of G where the jth coordinate is non-zero, and let $G_{b_1}, \dots G_{b_p}$ be the remaining columns where the jth coordinate is non-zero.
  Remove column G_a from G, and add $-G_{j,a}^{-1} G_{b_i,a} G_a$ to each $G_{b_i}, i \in [p]$.
end

CLAIM 2.1. *Given a matrix G of dimension k, after i contractions, the dimension of the column span of G is $k - i$.*

Proof. For the base case, note that after 0 contractions, the dimension is indeed k, as the columns of G are indeed a basis. Now, suppose the claim holds inductively. We will assume that after i contractions, the rank is $k - i$, and show that this holds after the $i + 1$st contraction. This is clear however, as there are $k - i - 1$ columns after the $i + 1$st contraction, and if we added the column that we removed back into the matrix, the rank would still be $k - i$ (as every column operation was invertible). If adding one vector can bring the dimension to $k - i$, the dimension before adding the vector must be at least $k - i - 1$. ∎

CLAIM 2.2. *For any non-zero coordinate $j \in [n]$ of G that we contract to get G', the span of G' is exactly all codewords in the column span of G that are zero in coordinate j.*

Proof. First, note that if a code of dimension k' is non-zero in some coordinate j, then it is non-zero in this coordinate in exactly $(q-1) \cdot q^{k'-1}$ codewords (and zero in this coordinate in exactly $q^{k'-1}$ codewords). After we contract on this jth coordinate, we get a new generating matrix G' of dimension $k' - 1$, where the jth row is all 0. Finally, because G' is made by adding columns of G together, the span of G' is contained in the span of G. This means that the span of G' is a $k' - 1$ dimensional subspace of G where the jth row is 0, which is exactly all codewords generated by G that are 0 in their jth coordinate, as the span of G' will have $q^{k'-1}$ codewords. ∎

CLAIM 2.3. *Consider a code $\mathcal{C}$ of dimension k, and a codeword $c \in \mathcal{C}$. If we never contract on any coordinate j where c is non-zero, then c is still in the span of the resulting contracted code.*

Proof. Let G' be the result of performing all the aforementioned contractions. Let the sequence of coordinates we contract on be $j_1, \dots j_k$. We know that after each contraction on j_i, the span of the new generator matrix is exactly all remaining codewords of G that are zero in coordinate j_i. Since c is zero on all of $j_1, \dots j_k$ (because we only ever contract on coordinates where c is 0 by assumption), then c always remains in the span of G after each contraction as the columns we removed are non-zero on these coordinates. So, after all the contractions are performed, c is still in the span of G. ∎

2.3 A Karger-Style Bound for Codewords.

Here, we will prove the following theorem:

THEOREM 2.1. *For a linear code $\mathcal{C} \subseteq \mathbb{F}_q^n$ of dimension k, and any integer $d \geq 1$, at least one of the following is true:*

1. *There exists a linear sub-code $\mathcal{C}' \subset \mathcal{C}$ such that Density$(\mathcal{C}') > \frac{1}{d}$.*
2. *For all integers α, there are at most $q^\alpha \cdot \binom{k}{\alpha}$ codewords of weight $\leq \alpha d$.*

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

REMARK 2.2. *Note that Theorem 2.1 implies Karger's original cut-counting bound as a special case with $q = 2$. For any $c \geq 1$, in any graph with minimum cut size c, the number of edges is necessarily at least $(nc)/2$. So Condition 1 above never arises once we set $d = c/2$, allowing us to recover Karger's original cut-counting bound.*

We first prove some sub-claims that will make this easier.

LEMMA 2.1. *Suppose we run Algorithm 1, and for some $d \in \mathbb{Z}^+$, after every contraction Density(Span(G)) $\leq \frac{1}{d}$. Then, the probability some codeword $c \in \mathcal{C}$ of weight $\leq \alpha d$ (for $\alpha \in \mathbb{Z}^+$) is still in the span of the final contracted G is at least $\binom{k}{\alpha}^{-1}$.*

Proof. By Claim 2.3, it follows that if c is in the span of the generating matrix G after i iterations, and we contract on a coordinate where c is non-zero, then c will still be in the span of the contracted generating matrix. So, it follows that the probability c survives is:

$$\Pr[\text{survives first contraction}] \cdot \dots \cdot \Pr[\text{survives } (k-\alpha)\text{th contraction}].$$

Using the fact that before the ith contraction, the dimension is $k - i + 1$, we know that the support must be at least $(k - i + 1) \cdot d$ by the assumed density. Because the codeword c survives if we contract on a coordinate where c is 0, the probability of survival in the ith contraction is at least $1 - \frac{\alpha \cdot d}{(k-i+1)\cdot d}$. Thus,

$$\Pr[\text{survives all contractions}] \geq (1 - \alpha/k)(1 - \alpha/(k-1)) \dots (1 - \alpha/(\alpha+1)) \tag{2.1}$$

$$= \frac{k-\alpha}{k} \cdot \frac{k-1-\alpha}{k-1} \cdot \dots \cdot \frac{\alpha+1-\alpha}{\alpha+1} = \binom{k}{\alpha}^{-1}. \tag{2.2}$$

∎

We are now ready to prove our main claim.

Proof. [Proof of Theorem 2.1] Suppose that condition 1 does not hold. Then, every linear sub-code $\mathcal{C}' \subseteq \mathcal{C}$ satisfies Density$(\mathcal{C}') \leq \frac{1}{d}$. We can then invoke Lemma 2.1 to conclude that any codeword with weight $\leq \alpha \cdot d$ is in the span of the contracted matrix with probability $\geq \binom{k}{\alpha}^{-1}$. Further, note that because the dimension of the contracted matrix is α, there are at most q^α codewords in this span. Because there are q^α codewords in the span of each contracted matrix, the sum of all the probabilities of low weight codewords surviving must be $\leq q^\alpha$. This means there can be at most $q^\alpha \cdot \binom{k}{\alpha}$ codewords of weight $\leq \alpha \cdot d$. ∎

THEOREM 2.2. *For a linear code $\mathcal{C}$ of dimension k and length n over $\mathbb{F}_q$, for any integer $d \geq 1$, there exists a set of at most $k \cdot d$ coordinates, such that upon their removal, in the resulting code, for any integer $\alpha \geq 1$ there are at most $q^\alpha \cdot \binom{k}{\alpha}$ codewords of weight $\leq \alpha d$.*

Proof. As long as Condition 1 of Theorem 2.1 continues to hold, we can write the matrix in the form $\begin{bmatrix} A & B \\ 0 & C \end{bmatrix}$, where the coordinates corresponding to A, B are the subcode of density $> \frac{1}{d}$. We can then remove all these coordinates corresponding to this subspace, and continue repeating this process until Condition 1 no longer holds. Once condition 2 holds, the dimension of the new code will be some value $\leq k$, so the number of codewords of weight $\leq \alpha d$ will be at most $q^\alpha \cdot \binom{k}{\alpha}$. To see why the number of coordinates we remove is at most kd, whenever the subspace we remove has dimension k', we remove at most $k'd$ coordinates, so after this removal, the resulting code is of dimension $k - k'$. It follows that we can remove at most kd coordinates before the dimension of the code is 0.

Note that after removing coordinates, the resulting code will have dimension $\leq k$, so in particular, the generating matrix will have a non-trivial nullspace. This means that there will be several messages that map to the *same* codeword. However, if the encoding of two messages is the same, i.e. yielding the same codeword, we do not count these as separate instances. Instead, this bound treats this as a single codeword. ∎

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

3 Preliminaries.

3.1 Codes.

DEFINITION 3.1. *For a code $\mathcal{C} \subseteq \mathbb{F}_q^n$, its **distance** is*

$$\min_{c \in \mathcal{C}: c \neq 0} \mathrm{wt}(c).$$

DEFINITION 3.2. *For a code $\mathcal{C} \subseteq \mathbb{F}_q^n$, the **coordinates** of the code are $[n]$. When we refer to the* number of coordinates*, this is interchangeable with the* length *of the code, which is exactly n.*

In this work, we will be concerned with code sparsifiers as defined below.

DEFINITION 3.3. *For a code $\mathcal{C} \subseteq \mathbb{F}_q^n$ with associated generating matrix G, a $(1 \pm \varepsilon)$-sparsifier for $\mathcal{C}$ is a subset $S \subseteq n$, along with a set of weights $w_S : S \to \mathbb{R}^+$ such that for any $x \in \mathbb{F}_q^k$*

$$(1-\varepsilon)\mathrm{wt}(Gx) \leq \mathrm{wt}_S(G|_S x) \leq (1+\varepsilon)\mathrm{wt}(Gx).$$

Here, wt_S is meant to imply that if the codeword is non-zero in its coordinate corresponding to an element $i \in S$, then it contributes $w_S(i)$ to the weight. We will often denote $G|_S$ with the corresponding weights as $\widetilde{G}$.

We next present a few simple results for code sparsification that we will use frequently.

CLAIM 3.1. *For a vertical decomposition of a generating matrix,*

$$G = \begin{bmatrix} G_1 \\ G_2 \\ \vdots \\ G_k \end{bmatrix},$$

if we have a $(1 \pm \varepsilon)$ sparsifier to codeword weights in each G_i, then their union is a $(1 \pm \varepsilon)$ sparsifier for G.

Proof. Consider any codeword $c \in \mathrm{Span}(G)$. Let c_i denote the restriction to each G_i in the vertical decomposition. It follows that if in the sparsifier $\mathrm{wt}(\hat{c}_i) \in (1 \pm \varepsilon)\mathrm{wt}(c_i)$, then $\mathrm{wt}(\hat{c}) = \sum_i \mathrm{wt}(\hat{c}_i) \in (1 \pm \varepsilon) \sum_i \mathrm{wt}(c_i) = (1 \pm \varepsilon)\mathrm{wt}(c)$. ∎

CLAIM 3.2. *Suppose $\mathcal{C}'$ is $(1 \pm \delta)$ sparsifier of $\mathcal{C}$, and $\mathcal{C}''$ is a $(1 \pm \varepsilon)$ sparsifier of $\mathcal{C}'$, then $\mathcal{C}''$ is a $(1-\varepsilon)(1-\delta), (1+\varepsilon)(1+\delta)$ approximation to $\mathcal{C}$ (i.e. preserves the weight of any codeword to a factor $(1-\varepsilon)(1-\delta)$ below and $(1+\varepsilon)(1+\delta)$ above).*

Proof. Consider any codeword $\mathcal{C}x$. We know that $(1-\varepsilon)\mathrm{wt}(\mathcal{C}x) \leq \mathrm{wt}(\mathcal{C}'x) \leq (1+\varepsilon)\mathrm{wt}(\mathcal{C}x)$. Additionally, $(1-\delta)\mathrm{wt}(\mathcal{C}'x) \leq \mathrm{wt}(\mathcal{C}''x) \leq (1+\delta)\mathrm{wt}(\mathcal{C}'x)$. Composing these two facts, we get our claim. ∎

Finally, we will use the following claim many times implicitly in our arguments, as we will freely change the generating matrix of the code we are looking at.

CLAIM 3.3. *Suppose generating matrices G and G' both generate the same dimension k code $\mathcal{C}$ of length n. Then, if some weighted subset of the rows of G yields a $(1 \pm \varepsilon)$ sparsifier $G|_S = \hat{G}$, the same weighted subset of the rows of G' yields a $(1 \pm \varepsilon)$ sparsifier $G'|_S = \hat{G}'$.*

Proof. Consider any codeword $c \in \mathcal{C}$. By construction, there is an x, x' such that $Gx = c, G'x' = c$. Now, $G|_S x = c|_S$ and $G'|_S x' = c|_S$. Hence, if $\hat{G}$ is a $(1 \pm \varepsilon)$ sparsifier of codewords in $\mathcal{C}$, then so too is $\hat{G}'$. ∎

3.2 A Probabilistic Bound.

We will frequently utilize the following probabilistic bound.

CLAIM 3.4. ([8]) *Let $X_1, \ldots X_\ell$ be random variables such that X_i takes on value $1/p_i$ with probability p_i, and is 0 otherwise. Also, suppose that $\min_i p_i \geq p$. Then, with probability at least $1 - 2e^{-0.38\varepsilon^2 \ell p}$,*

$$\sum_i X_i \in (1 \pm \varepsilon)\ell.$$

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

3.3 Graphs and Graph Sparsification. We recall here a few basic concepts about graphs and graph sparsification.

DEFINITION 3.4. *A* cut *in a graph $G = (V, E)$ is a subset $S \subseteq V$. We will specify the size of a cut $|\delta_G(S)|$ to be the number of edges that cross from S to $V - S$ (i.e. the number of edges that go from a set S to the rest of the vertices). When a graph $G = (V, E)$ also has an associated weight function $w : E \to \mathbb{R}^+$, we take $|\delta_G(S)|$ to be the sum over all the edges that cross from S to $V - S$ of the weights of these crossing edges.*

DEFINITION 3.5. *A $(1 \pm \varepsilon)$* cut*-sparsifier for a weighted graph $G = (V, E)$ is a new, weighted graph $\hat{G} = (V, \hat{E})$, with associated weight function w, such that*

1. *$\hat{E} \subseteq E$.*
2. *For every cut $S \subseteq V$,*
$$(1 - \varepsilon)|\delta_G(S)| \leq |\delta_{\hat{G}}(S)| \leq (1 + \varepsilon)|\delta_G(S)|.$$

The famous result of Karger relates the size of the minimum cut to the number of cuts of other sizes.

DEFINITION 3.6. *The* minimum cut *of a graph $G = (V, E)$ is*
$$\min_{S \subseteq V : S \neq V, S \neq \emptyset} |\delta_G(S)|.$$

THEOREM 3.1. (KARGER'S CUT-COUNTING BOUND) *[12, 14] Suppose a graph $G = (V, E)$ has n vertices, and minimum cut value c. Then, for any integer α, the number of cuts of size at most αc is at most $n^{2\alpha}$.*

DEFINITION 3.7. (CUT CODE) *For intuition, we will several times refer to the "cut code" of a corresponding graph $G = (V, E)$. Intuitively, this is the code over $\mathbb{F}_2$ with a generating matrix on $|V|$ columns, where for every edge $e = (u, v) \in E$, we add a row in the generating matrix which has a 1 in the columns corresponding to u and v. If we denote this generating matrix by G', it can be verified that for any cut $(S, V - S)$,*
$$|\delta_G(S)| = \mathrm{wt}(G'\mathbf{1}_S).$$

That is, the weight of the codeword corresponding to the encoding of the indicator vector of S is exactly the number of edges crossing the cut $(S, V - S)$.

3.4 CSPs and CSP Sparsification. We formally define here CSPs and CSP sparsification. We start by introducing the notion of a predicate.

DEFINITION 3.8. *A predicate P of arity r is a function going from $\{0,1\}^r \to \{0,1\}$.*

We will look at CSPs which are defined using a single predicate.

DEFINITION 3.9. *A CSP over k variables with predicate P, is a collection of constraints of the form $P(x_1^{(i)}, x_2^{(i)}, \ldots x_r^{(i)})$ where r is the arity of P, and i ranges from 1 to m.*

DEFINITION 3.10. *The value obtained by the CSP on an assignment x is correspondingly*
$$\sum_{i=1}^{m} P(x_1^{(i)}, x_2^{(i)}, \ldots x_r^{(i)})$$

In some cases, the CSP has a corresponding weight *$w_i \in \mathbb{R}^+$ for each constraint. In this case, the value of the CSP on assignment x is*
$$\sum_{i=1}^{m} w_i \cdot P(x_1^{(i)}, x_2^{(i)}, \ldots x_r^{(i)}).$$

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

DEFINITION 3.11. *A $(1 \pm \varepsilon)$-sparsifier for a CSP C on m constraints, is a new CSP $\hat{C}$ specified by a subset $T \subseteq [m]$, along with weights $(w_i)_{i \in T}$, such that for any assignment $x \in \{0,1\}^k$,*

$$(1-\varepsilon)\sum_{i=1}^{m} P(x_1^{(i)}, x_2^{(i)}, \dots x_r^{(i)}) \leq \sum_{i \in T} w_i P(x_1^{(i)}, x_2^{(i)}, \dots x_r^{(i)}) \leq (1+\varepsilon)\sum_{i=1}^{m} P(x_1^{(i)}, x_2^{(i)}, \dots x_r^{(i)}).$$

In words, we choose a weighted subset of the constraints of C, such that for any assignment x, the value of the CSP is preserved to a $(1 \pm \varepsilon)$ factor.

Note that in some works, sparsifying a CSP is meant to only preserve satisfiability while reducing the number of constraints. In our setting, the goal is to approximately preserve the value of the satisfied constraints.

DEFINITION 3.12. *For a universe of variables $x \in \{0,1\}^k$, an affine projection is a restriction of the variables of the form $x_i = 0, x_i = 1, x_i = x_j$ or $x_i = \neg x_j$.*

REMARK 3.1. *We say that an affine projection of a predicate P of arity r yields an AND of arity 2 if there exists a function $\pi : [r] \to \{0, 1, x, \neg x, y, \neg y\}$ such that $\mathrm{AND}(x, y) = P(\pi(1), \dots, \pi(r))$.*

For instance, the predicate $P : \{0,1\}^3 \to \{0,1\}$, with the only satisfying assignments $000, 001$ is equal to an AND of arity 2 under affine projections. If we consider the restriction R which sets the third variable equal to 0, we get a new predicate $P|_R$ whose only satisfying assignment is 00. Thus, this predicate $P|_R(y_1, y_2) = \neg y_1 \wedge \neg y_2$.

4 Near-linear Size Sparsifiers for Polynomially-bounded Codes.

In this section, we will prove the existence of near-linear size sparsifiers for codes of length $k^{O(1)}$, where k is the dimension of the code. That is, when $\mathcal{C}$ is a linear code of dimension k and length $k^{O(1)}$. We will also assume that $\mathcal{C}$ is unweighted (or that every coordinate has the same weight). The main theorem to be proved in this section is Theorem 4.1 showing the result for polynomially length (which is a specific case of the more general Theorem 4.2 which applies to codes of any length, but loses extra factors, also proved here). In particular, this is a special case of Theorem 1.1 proved in the introduction.

Intuitively, the proof will use Theorem 2.2 to repeatedly decompose the code $\mathcal{C}$. In each application, we invoke Theorem 2.2 with a specific choice of parameter d to decompose the generator matrix for the code $\mathcal{C}$ into the form

$$\begin{bmatrix} A & B \\ 0 & C \end{bmatrix},$$

where A is exactly the low-dimensional subcode with bounded support. We will argue that we can keep all of the coordinates of A because the support of A is sufficiently bounded. In turn, this means that B is effectively preserved as well, so as long as we can get a $(1 \pm \varepsilon)$ approximation to C of sufficiently small size, we will be okay. To deal with C, we argue that the distribution of the weights of codewords in C is sufficiently smooth, such that we may subsample the coordinates at rate roughly $1/d$.

This now yields two separate codes, each with size that is strictly smaller than the starting size. We operate inductively, and recursively break down these two smaller codes. Ultimately, in each recursive step, we take a code of size $k \cdot k^{\gamma}$, and return two codes of size roughly $k \cdot k^{\gamma/2}$. For an initial code of length $k^{O(1)}$ (i.e. whose length is polynomial in the dimension), after $\log \log k$ levels of recursion, we have $\log k$ codes, each of length roughly k. In turn, we can glue these codes back together, and return a sparsifier for our original code.

4.1 Code Decomposition.

First, we consider Algorithm 2:

Algorithm 2: CodeDecomposition$(\mathcal{C}, d)$

1 Let k be the dimension of $\mathcal{C}$.
2 Let S be the set of coordinates to be removed as specified by Theorem 2.2.
3 Let $\mathcal{C}'$ be the code $\mathcal{C}$ after removing the set of coordinates S.
4 **return** $S, \mathcal{C}'$

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

CLAIM 4.1. *1. After the termination of the Algorithm 2, $|S| \leq k \cdot d$.*

2. The final resulting $\mathcal{C}'$ of Algorithm 2 satisfies condition 2 of Theorem 2.1.

Proof. For the details of the proof, please refer to [16]. ∎

CLAIM 4.2. *To get a $(1 \pm \varepsilon)$ code sparsifier for $\mathcal{C}$, it suffices to get $S, \mathcal{C}'$ from Algorithm 2, and then sample all of the indices in S with probability 1, and get a $(1 \pm \varepsilon)$-sparsifier for $\mathcal{C}'$.*

Proof. For the details of the proof, please refer to [16]. ∎

4.2 Code Sparsification Algorithm. In this section we will present the code sparsification algorithm, and argue its correctness and its sparsity.

Algorithm 3: CodeSparsify($\mathcal{C} \subseteq \mathbb{F}_q^n, k, \varepsilon, \eta$)

Let n be the length of $\mathcal{C}$.
if $n \leq 100 \cdot k \cdot \eta \log(k) \log(q)/\varepsilon^2$ **then**
  return $\mathcal{C}$
end
Let $d = \frac{n\varepsilon^2}{\eta \cdot k \log(k) \log(q)}$.
Let $S, \mathcal{C}' = \text{CodeDecomposition}(\mathcal{C}, \sqrt{d} \cdot \eta \cdot \log(k) \log(q)/\varepsilon^2)$. Let $\mathcal{C}_1 = \mathcal{C}|_S$. Let $\mathcal{C}_2$ be the result of sampling every coordinate of $\mathcal{C}'$ at rate $1/\sqrt{d}$.
return $\text{CodeSparsify}(\mathcal{C}_1, k, \varepsilon, \eta) \cup \sqrt{d} \cdot \text{CodeSparsify}(\mathcal{C}_2, k, \varepsilon, \eta)$

We will use the following fact, which is a simple extension of a result from Karger [13]. In Karger's work, it was noted that for a graph with minimum cut value c, one can roughly sample the edges at rate $\log(n)/(c\varepsilon^2)$ and scale the weights of the sampled edges up by $c\varepsilon^2/\log(n)$ while still maintaining a $(1 \pm \varepsilon)$ approximation to the cuts in the graph. In the following claim, we adapt this fact to codes which satisfy a smooth bound on the number of codewords of a given weight.

CLAIM 4.3. *Suppose $\mathcal{C}$ is a code of dimension k over $\mathbb{F}_q$, and let $b \geq 1$ be an integer such that for any integer $\alpha \geq 1$, the number of codewords of weight $\leq \alpha b$ is at most $(qk)^\alpha$. Suppose further that the minimum distance of the code $\mathcal{C}$ is b. Then, sampling the coordinates of $\mathcal{C}$ at rate $\frac{\log(k)\log(q)\eta}{b\varepsilon^2}$ with weights $\frac{b\varepsilon^2}{\log(k)\log(q)\eta}$ yields a $(1 \pm \varepsilon)$ sparsifier with probability $1 - 2^{-(0.19\eta - 110)\log k} \cdot k^{-101}$.*

We can then state the main theorem from this section:

THEOREM 4.1. *For a code $\mathcal{C}$ on alphabet $\mathbb{F}_q$ of dimension k, and length $k^{O(1)}$, Algorithm 3 creates a $(1 \pm \varepsilon)$ sparsifier for $\mathcal{C}$ of size $O(k\eta \log^2(k) \log(q) (\log\log(k))^2/\varepsilon^2)$ with probability $1 - 2^{-(0.19\eta - 110)\log k} \cdot k^{-100}$.*

Proof. See [16] for the full proof. ∎

THEOREM 4.2. *For a code $\mathcal{C}$ of dimension k, and length n over $\mathbb{F}_q$, Algorithm 3 creates a $(1 \pm \varepsilon)$ sparsifier for $\mathcal{C}$ with probability $1 - \log(n) \cdot 2^{-(0.19\eta - 110)\log k} \cdot k^{-100}$ with at most*

$$O(k\eta \log(k) \log(q) \log^2(n) (\log\log(n))^2/\varepsilon^2)$$

coordinates.

Proof. See [16] for the full proof. ∎

However, as we will address in the next section, this result is not perfect:

1. For large enough n, there is no guarantee that this probability is ≥ 0 unless η depends on n.
2. For large enough n, $\log^2(n)$ may even be larger than k.

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

5 Nearly Linear Size Sparsifiers for Codes of Arbitrary Length.

In this section, our goal is to prove Theorem 1.1 (the exact version proved will be Theorem 5.1). We will do this by using Theorem 4.2 as a sub-routine in another algorithm.

In the previous section, we saw an algorithm which produces a near-linear size sparsifier for codes of length polynomial in the dimension. However, if we start with a code of arbitrary length n, simply applying the algorithm from the previous section led to spurious $\log(n)$ factors, which unfortunately can dwarf k. In this section, we will show how we can be a little more careful with our sparsifier to avoid these extra $\log(n)$ factors. To do this, in § 5.1, we will showcase a simple one-shot algorithm that returns a size $O(k^2 \log(q)/\varepsilon^2)$ weighted $(1 \pm \varepsilon)$ sparsifier of any code of dimension k and length n. Ideally, we could simply use this sparsification and compose on top of it Algorithm 3. However, as written, Algorithm 3 only works for *unweighted* codes (or codes where every coordinate has the same weight).

Further, the ratio of the weights that are returned by this sparsifier is unbounded in k (and in many cases will be as large as $\Omega(n)$). Naive notions of turning the weighted code into an unweighted code unfortunately do not work, as if we try to replace coordinates of weight w with w unweighted coordinates, we will no longer be guaranteed that length of the code is polynomial, and we will not have gained anything.

Instead, we will show that once we have a weighted sparsifier of size polynomial in the dimension, we can group coordinates together by their weights. That is, we set a parameter $\alpha = \text{poly}(k/\varepsilon)$ sufficiently large, and set the ith group to contain coordinates with weights between $[\alpha^{i-1}, \alpha^i]$. Our key observation is that if a codeword is non-zero in any coordinate in the ith group, then for an appropriately chosen α, the total weighted contribution from any coordinates in groups $i-2, i-3, \ldots$ is much less than an $\varepsilon/100$ fraction of the weight coming from group i. Thus if we let i be the largest integer such that a codeword is non-zero in the ith weight group, we can effectively ignore all the coordinates corresponding to weight groups $i-2, \ldots$ when sparsifying this codeword. Now, starting with the largest i, we decompose the code into codewords which are non-zero in group i and those which are zero in group i. For those which are non-zero, we can effectively ignore all the coordinates from groups $i-2, i-3, \ldots$. This means that all the coordinates we are concerned with have weights in the range $[\alpha^{i-2}, \alpha^i]$. To turn this into an unweighted code, we simply pull out a factor of α^{i-2}, and now for a coordinate of weight w, we can repeat it roughly w times. Because the weights are polynomial in the dimension, the resulting unweighted code is also polynomial in dimension, and we can invoke the results from the previous section.

5.1 Simple Quadratic Size Sparsifiers.

In this section, we will introduce a one-shot method for sparsifying a code of dimension k and length n on alphabet $\mathbb{F}_q$ that maintains $O(k^2 \log(q)/\varepsilon^2)$ indices of the original code. We state the algorithm here, and then analyze the space complexity and correctness of this algorithm.

Algorithm 4: QuadraticSparsify($\mathcal{C} \subseteq \mathbb{F}_q^n, k, \varepsilon$)

Let n be the length of $\mathcal{C}$. **for** $i = 1, \ldots n$ **do**
  Let w_i be $\min_{c \in \mathcal{C} : c_i \neq 0} \text{wt}(c)$.
end
Let $\mathcal{C}'$ be the result of sampling every coordinate of $\mathcal{C}$ with probability $\min(1, a \cdot k \log(q)/(\varepsilon^2 w_i))$, and weight $1/\min(1, a \cdot k/(\varepsilon^2 w_i))$.
return $\mathcal{C}'$

5.1.1 Correctness.

First, we prove the correctness of this algorithm.

LEMMA 5.1. *For a code $\mathcal{C}$ of dimension k, and a fixed codeword $c \in \mathcal{C}$, Algorithm 4 returns a sparsifier $\mathcal{C}'$ for $\mathcal{C}$, such that the new weight of c is a $(1 \pm \varepsilon)$ approximation to the old weight with probability at least $1 - 2^{-2k}$.*

Proof. See [16] for the full proof. ∎

5.1.2 Size Analysis.

Next, we bound the space taken by this algorithm. To do this, we will first need to take advantage of some structural results about codes.

FACT 5.1. *For any linear code, there exists a basis such that codewords of weight ℓ can be written as the sum of codewords of weight $\leq \ell$ from the basis.*

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

REMARK 5.1. *Fix such a basis $b_1, \dots b_k$ as specified by the previous fact. Now, consider any coordinate of this code. If we look at the weights of all codewords that are non-zero in this coordinate, the minimum weight codeword must be with one of the basis vectors. This follows easily: any codeword non-zero in its ith coordinate must be the sum of at least 1 basis vector which is non-zero in its ith coordinate. This means that the weight of the codeword must be greater than the weight of the corresponding basis vectors in its sum.*

By the previous remark, when we try to bound the possible values attained by $\max_x \frac{\langle r_i, x\rangle}{\mathrm{wt}(\mathcal{D}x)}$, it will suffice to analyze the possible value attained just by looking at the basis codewords for this special basis. This simplifies things, as we do not have to look at what can happen with possible linear combinations of the codewords. Instead of looking at a basis, we may simply consider a matrix of dimension $n \times k$.

CLAIM 5.1. *Fix an $n \times k$ matrix A. Let c_i denote the ith coordinate of c, and let A_j denote the jth column of A. Then,*

$$\sum_{i=1}^{n} \max_{j \in [k]:(A_j)_i = 1} \frac{1}{\mathrm{wt}(c_j)} \leq k.$$

CLAIM 5.2. *For any linear code $\mathcal{C}$ of dimension k and length n, with a generator matrix G consisting of rows r_i,*

$$\sum_{i=1}^{n} \max_{c \in \mathcal{C}:c_i = 1} \frac{1}{\mathrm{wt}(c)} \leq k.$$

Proof. This follows by taking the specified codeword basis from Remark 5.1 and invoking Claim 5.1. ∎

Finally, we can prove a bound on the size of the sketch.

LEMMA 5.2. *With probability $1 - 2^{-k}$, Algorithm 4 does not sample more than $O(k^2 \log(q)/\varepsilon^2)$ coordinates.*

Proof. See [16] for the full proof. ∎

5.2 Removing the $O(\log n)$ factors. Similar to [5], we want to remove the extra factors of $\log n$. To this end, we suggest the following procedure upon being given a code of length n and dimension k in Algorithm 5.

Algorithm 5: WeightClassDecomposition$(\mathcal{C}, \varepsilon, k)$

1 Let $\mathcal{C}'$ =QuadraticSparsify$(\mathcal{C}, k, \varepsilon/4)$.
2 Let $\alpha = \frac{k^3 \log(q)}{\varepsilon^3}$.
3 Let E_i be all coordinates of $\mathcal{C}'$ that have weight between $[\alpha^{i-1}, \alpha^i]$.
4 Let $\mathcal{D}_{\text{odd}} = E_1 \cup E_3 \cup E_5 \cup \dots$, and let $\mathcal{D}_{\text{even}} = E_2 \cup E_4 \cup E_6 \cup \dots$.
5 **return** $\mathcal{D}_{odd}, \mathcal{D}_{even}$.

Next, we mention some facts about this algorithm.

LEMMA 5.3. *Consider a code $\mathcal{C}$ of dimension k and length n. Let*

$$\mathcal{D}_{odd}, \mathcal{D}_{even} = \text{WeightClassDecomposition}(\mathcal{C}, \varepsilon, k).$$

To get a $(1 \pm \varepsilon)$-sparsifier for $\mathcal{C}$, it suffices to get a $(1 \pm \varepsilon/4)$ sparsifier to each of $\mathcal{D}_{odd}, \mathcal{D}_{even}$.

Proof. See [16] for the full proof. ∎

Because of the previous lemma, it is now our goal to create sparsifiers for $\mathcal{D}_{\text{odd}}, \mathcal{D}_{\text{even}}$. Without loss of generality, we will focus our attention only on $\mathcal{D}_{\text{even}}$, as the procedure for $\mathcal{D}_{\text{odd}}$ is exactly the same (and the proofs will be the same as well). At a high level, we will take advantage of the fact that

$$\mathcal{D}_{\text{even}} = E_2 \cup E_4 \cup \dots,$$

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

where each E_i contains edges of weights $[\alpha^{i-1}, \alpha^i]$, for $\alpha = \frac{k^3 \log(q)}{\varepsilon^3}$. Because the returned result from Quadratic sparsify has at most $O(k^2 \log(q)/\varepsilon^2)$ edges with high probability, whenever a codeword $c \in \mathcal{C}'$ has a 1 in a coordinate corresponding to E_i, we can effectively ignore all coordinates of lighter weights $E_{i-2}, E_{i-4}, \ldots$. This is because any coordinate in $E_{\leq i-2}$ has weight at most a $\frac{\varepsilon^3}{k^3 \log(q)}$ fraction of any single coordinate in E_i. Because there are at most $O(k^2 \log(q)/\varepsilon^2)$ coordinates in $\mathcal{C}'$, it follows that the total possible weight of all coordinates in $E_{\leq i-2}$ is still at most a $O(\varepsilon/k)$ fraction of the weight of a single coordinate in E_i. Thus, we will argue that when we are creating a sparsifier for codewords that have a 1 in a coordinate corresponding to some E_i, we will be able to effectively ignore all coordinates corresponding to $E_{\leq i-2}$. To argue this, we will first have to show how to decompose the code into blocks that are non-zero in coordinates in E_i. So, consider the following algorithm:

Algorithm 6: SingleSpanDecomposition($\mathcal{D}_{\text{even}}, \alpha, i$)

Let E_i be all coordinates of $\mathcal{D}_{\text{even}}$ with weights between α^{i-1} and α^i.
Let G be a generating matrix for $\mathcal{D}_{\text{even}}$.
Let k' be the rank of $G|_{E_i}$.
Let $b_1, \ldots b_{k'}$ be k' linearly independent columns in $G|_{E_i}$.
Permute the columns of $\mathcal{D}_{\text{even}}$ so $b_1, \ldots b_{k'}$ become the first k' columns of $G|_{E_i}$.
Perform column operations on G to cancel out all remaining columns in $G|_{E_i}$.
return $G|_{E_i}$, $G|_{\bar{E}_i}$, k'

Now, we can continue to decompose the code by repeating Algorithm 6 multiple times.

Algorithm 7: SpanDecomposition($\mathcal{D}_{\text{even}}, \alpha$)

Let $\mathcal{D}'_{\text{even}} = \mathcal{D}_{\text{even}}$.
Let $S = \{\}$.
while $\mathcal{D}'_{even}$ *is not empty* **do**
  Let k be the dimension of $\mathcal{D}'_{\text{even}}$. Let i be the largest integer such that E_i is non-empty in $\mathcal{D}'_{\text{even}}$.
  Let $G|_{E_i}, G|_{\bar{E}_i}, k' =$SingleSpanDecomposition($\mathcal{D}'_{\text{even}}, \alpha, i$).
  Let H_i be the first k' columns of $G|_{E_i}$.
  Let $\mathcal{D}'_{\text{even}}$ be the span of the final $k - k'$ columns of $G|_{\bar{E}_i}$.
  Add i to S.
end
return S, H_i *for every* $i \in S$

CLAIM 5.3. *Let S, H_i be as returned by Algorithm 7. Then, $\sum_{i \in S} \text{rank}(H_i) = \text{rank}(\mathcal{D}_{even})$.*

Proof. For the details of the proof, please refer to [16]. ∎

LEMMA 5.4. *Suppose we have a code of the form $\mathcal{D}_{even}$ created by Algorithm 5. Then, if we run Algorithm 7 on $\mathcal{D}_{even}$, to get $S, H_i \forall i \in S$, it suffices to get a $(1 \pm \varepsilon/2)$ sparsifier for each of the H_i in order to get a $(1 \pm \varepsilon)$ sparsifier for $\mathcal{D}_{even}$.*

Proof. See [16] for the full proof. ∎

Dealing with Bounded Weights. Let us consider any H_i that is returned by Algorithm 7, when called with $\alpha = k^3 \log(q)/\varepsilon^3$. By construction, H_i will contain weights only in the range $[\alpha^{i-1}, \alpha^i]$ and will have at most $O(k^2 \log(q)/\varepsilon^2)$ coordinates. In this subsection, we will show how we can turn H_i into an unweighted code with at most $O(k^5 \log^2(q)/\varepsilon^6)$ coordinates. First, note however, that we can simply pull out a factor of α^{i-1}, and treat the remaining graph as having weights in the range of $[1, \alpha]$. Because multiplicative approximation does not change under multiplication by a constant, this is valid. Formally, consider the following algorithm:

LEMMA 5.5. *Consider a code $\mathcal{C}$ with weights bounded in the range $[1, \alpha]$. To get a $(1 \pm \varepsilon)$ sparsifier for $\mathcal{C}$ it suffices to return a $(1 \pm \varepsilon/10)$ sparsifier for $\mathcal{C}'$ weighted by $\varepsilon/10$, where $\mathcal{C}'$ is the result of calling Algorithm 8 on $\mathcal{C}, \alpha, 1, \varepsilon$.*

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Algorithm 8: MakeUnweighted($\mathcal{C}, \alpha, i, \varepsilon$)

Divide all edge weights in $\mathcal{C}$ by α^{i-1}.
Make a new unweighted code $\mathcal{C}'$ by duplicating every coordinate of $\mathcal{C}$ $\lfloor 10w(r)/\varepsilon \rfloor$ times.
return $\mathcal{C}, \alpha^{i-1} \cdot \varepsilon/10$

Proof. See [16] for the full proof. ∎

CLAIM 5.4. *Suppose a code $\mathcal{C}$ of length n has weight ratio bounded by α, and minimum weight α^{i-1}. Then, calling Algorithm 8 with error parameter ε yields a new unweighted code of length $O(n\alpha/\varepsilon)$.*

Proof. Each coordinate is repeated at most $O(\alpha/\varepsilon)$ times. ∎

5.3 Final Algorithm. Finally, we state our final algorithm in Algorithm 9, which will create a $(1 \pm \varepsilon)$ sparsifier for any code $\mathcal{C} \subseteq \mathbb{F}_q^n$ of dimension k preserving only $\widetilde{O}(k \log(q)/\varepsilon^2)$ coordinates.

Algorithm 9: FinalCodeSparsify($\mathcal{C}, \varepsilon$)

1 Let k be the dimension of $\mathcal{C}$.
2 Let $\alpha = k^3 \log(q)/(\varepsilon/2)^3$, and $\mathcal{D}_{\text{odd}}, \mathcal{D}_{\text{even}}$ =WeightClassDecomposition($\mathcal{C}, \varepsilon, k$).
3 Let $S_{\text{even}}, \{H_{\text{even},i}\}$ =SpanDecomposition($\mathcal{D}_{\text{even}}, \alpha$).
4 Let $S_{\text{odd}}, \{H_{\text{odd},i}\}$ =SpanDecomposition($\mathcal{D}_{\text{odd}}, \alpha$).
5 **for** $i \in S_{even}$ **do**
6 Let $\widehat{H}_{\text{even},i}, w_{\text{even},i}$ = MakeUnweighted($H_{\text{even},i}, \alpha, i, \varepsilon/8$).
7 Let $\widetilde{H}_{\text{even},i}$ =CodeSparsify($\widehat{H}_{\text{even},i}, \text{rank}(\widehat{H}_{\text{even},i}), \varepsilon/80, 100(\log(k/\varepsilon)\log\log(q))^2$).
8 **end**
9 **for** $i \in S_{odd}$ **do**
10 Let $\widehat{H}_{\text{odd},i}, w_{\text{odd},i}$ = MakeUnweighted($H_{\text{odd},i}, \alpha, i, \varepsilon/8$).
11 Let $\widetilde{H}_{\text{odd},i}$ =CodeSparsify($\widehat{H}_{\text{odd},i}, \text{rank}(\widehat{H}_{\text{odd},i}), \varepsilon/80, 100(\log(k/\varepsilon)\log\log(q))^2$).
12 **end**
13 **return** $\bigcup_{i \in S_{even}} \left(w_{even,i} \cdot \widetilde{H}_{even,i} \right) \cup \bigcup_{i \in S_{odd}} \left(w_{odd,i} \cdot \widetilde{H}_{odd,i} \right)$

First, we analyze the space complexity. WLOG we will prove statements only with respect to $\mathcal{D}_{\text{even}}$, as the proofs will be identical for $\mathcal{D}_{\text{odd}}$.

CLAIM 5.5. *Suppose we are calling Algorithm 9 on a code $\mathcal{C}$ of dimension k. Let $k_{even,i} = \text{rank}(\widehat{H}_{even,i})$ from each call to the for loop in line 5.*

For each call $\widetilde{H}_{even,i}$ =CodeSparsify($\widehat{H}_{even,i}, \text{rank}(\widehat{H}_{even,i}), \varepsilon/10, 100(\log(k/\varepsilon)\log\log(q))^2$) in Algorithm 9, the resulting sparsifier has

$$O\left(k_{even,i} \log(k_{even,i}) \log^2(k/\varepsilon) \cdot \log^2(k/\varepsilon) \log(q) (\log\log(k/\varepsilon)\log\log(q))^2/\varepsilon^2\right)$$

coordinates with probability at least $1 - \log(k\log(q)/\varepsilon) \cdot 2^{-\Omega(\log^2(k/\varepsilon)(\log\log(q))^2)}$.

Proof. See [16] for the full proof. ∎

LEMMA 5.6. *In total, the combined number of coordinates over $i \in S_{even}$ of all of the $\widetilde{H}_{even,i}$ is at most $\widetilde{O}(k\log(q)/\varepsilon^2)$ with probability at least $1 - \log(k\log(q)/\varepsilon) \cdot 2^{-\Omega(\log^2(k/\varepsilon)(\log\log(q))^2)}$.*

Proof. See [16] for the full proof. ∎

Now, we will prove that we also get a $(1 \pm \varepsilon)$ sparsifier for $\mathcal{D}_{\text{even}}$ when we run Algorithm 9.

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

LEMMA 5.7. *After combining the $\widehat{H}_{even,i}$ from Lines 5-8 in Algorithm 9, the result is a $(1 \pm \varepsilon/4)$-sparsifier for $\mathcal{D}_{even}$ with probability at least $1 - \log(k\log(q)/\varepsilon) \cdot 2^{-\Omega(\log^2(k/\varepsilon)(\log\log(q))^2)}$.*

Proof. See [16] for the full proof. ∎

The reason for our choice of η is a little subtle. For Theorem 4.2, the failure probability is characterized in terms of the dimension of the code that is being sparsified. However, when we call Algorithm 3 as a sub-routine in Algorithm 9, we have no guarantee that the rank is $\omega(1)$. Indeed, it is certainly possible that the decomposition in H_i creates k different matrices all of rank 1. Then, choosing η to only be a constant, as stated in Theorem 4.2, the failure probability could be constant, and taking the union bound over k choices, we might not get anything meaningful. To amend this, instead of treating η as a constant in Algorithm 3, we set $\eta = 100(\log(k/\varepsilon)\log\log(q))^2$, where now k is the *overall* rank of the code $\mathcal{C}$, *not* the rank of the current code that is being sparsified H_i. With this modification, we can then attain our desired probability bounds.

THEOREM 5.1. *For any code $\mathcal{C}$ of dimension k and length n, Algorithm 9 returns a $(1 \pm \varepsilon)$ sparsifier to $\mathcal{C}$ with $\widetilde{O}(k\log(q)/\varepsilon^2)$ coordinates with probability $\geq 1 - 2^{-\Omega((\log(k/\varepsilon)\log\log(q))^2)} - 2^{-k}$.*

Proof. See [16] for the full proof. ∎

Extension to the weighted case. Finally, note that as stated, this section proves the existence of near-linear size code sparsifiers for *unweighted* codes of any length. However, this extends simply to *weighted* codes of any length, as we can simply repeat coordinates in accordance with their weights, and then sparsify the resulting unweighted code.

We make this more rigorous below:

CLAIM 5.6. *Suppose we are given a weighted code $\mathcal{C}$ of dimension k, where the weight of coordinate i is $w_i \in \mathbb{R}^+$, and the smallest weight of any coordinate is w. Then, if we create a new unweighted code $\mathcal{C}'$ by repeating coordinate i $\lfloor \frac{100w_i}{\varepsilon w} \rfloor$ times, then for any $c \in C$, the corresponding $c' \in \mathcal{C}'$ satisfies*

$$\frac{w\varepsilon}{100} \cdot \mathrm{wt}(c') \in (1 \pm \varepsilon/10)\mathrm{wt}(c).$$

Further, a $(1 \pm \varepsilon/10)$-sparsifier to $\mathcal{C}'$ when weighted by $\frac{w\varepsilon}{100}$ yields a $(1 \pm \varepsilon)$-sparsifier for $\mathcal{C}$. Hence, there exist sparsifiers of size $\widetilde{O}(k\log(q)/\varepsilon^2)$ for any weighted code $\mathcal{C}$.

Proof. See [16] for the full proof. ∎

6 Application to Cayley Graph Sparsifiers.

In this section, we will explore the connection between the weights of codewords for an error correcting code $\mathcal{C}$ of dimension k and the eigenvalues of the Laplacian of a Cayley graph on $\mathbb{F}_2^k$. At a high level, it is well known that there exist Cayley graph expanders on $\mathbb{F}_2^k$ with constant expansion when the degree of the graph is $\Omega(k)$. Any such expander H can be viewed as a sparsifier to the complete Cayley graph G on $\mathbb{F}_2^k$ (where we take the set of generators S to be exactly $\mathbb{F}_2^k$), under the constraint that the resulting sparsifier still has a Cayley graph structure, and $L_G \approx_\varepsilon L_H$. At a high level, our result says that for *any* Cayley graph G over $\mathbb{F}_2^k$, there exists a sparsifier H with at most $\widetilde{O}(k/\varepsilon^2)$ edges, such that H is still a Cayley graph, and $L_H \approx_\varepsilon L_G$.

Note that by prior work [1], we know that there exist sparsifiers H for any Cayley graph G such that $L_H \approx_\varepsilon L_G$, however this is the first work which proves the existence of such graphs under the restriction that H is also a Cayley graph and of nearly-linear size.

Preliminaries. First, we introduce some definitions related to Cayley graphs. In this section, we will fix a binary linear code $\mathcal{C}$, as well as a generating matrix for $\mathcal{C}$, denoted by $G_\mathcal{C}$. Let r_i denote the ith row of $G_\mathcal{C}$.

DEFINITION 6.1. *A Cayley graph G is a graph with algebraic structure; its vertex set is defined to be a group, and the edges correspond to a set of generators S, along with weight $(w_i)_{i \in S}$. For every element in $s \in S$, and for every vertex v, there is an edge from v to $v + s$ of weight w_s.*

DEFINITION 6.2. *Let $\chi_x(r) = (-1)^{\langle x,r \rangle}$, where the inner product is taken modulo 2, and $x, r \in \mathbb{F}_2^k$.*

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

FACT 6.1. *For a Cayley graph G defined over $\mathbb{F}_2^k$ with generating set S, there is exactly one eigenvalue of G for every vertex x in its vertex set (which is $\mathbb{F}_2^k$). The corresponding eigenvalue (of the adjacency matrix) is*

$$\lambda_x(G) = \sum_{r \in S} w_r \chi_x(r).$$

The corresponding eigenvector is χ_x. Note that this means that any Cayley graph defined on the same vertex set has the same eigenvectors.

Going forward, we will let G be a Cayley graph over $\mathbb{F}_2^k$, where its set of generators S is exactly $\{r_1, \dots r_n\}$, where these are the rows of the generating matrix $G_{\mathcal{C}}$.

FACT 6.2. *For a message $x \in \mathbb{F}_2^k$, we have that*

$$\text{Bias}(G_{\mathcal{C}} x) = \mathbb{E}_{r \in S} \chi_x(r_i).$$

The above statement is very intuitive. $\chi_x(r_i)$ is 1 if the ith bit in the codeword corresponding to x is 0, and is -1 if the ith bit is 1. As a result, this expectation is exactly measuring how many more 0's there are than 1's.

FACT 6.3. *We can generalize the previous fact to the eigenvalues of the Laplacian. In this way we get that*

$$\lambda_x(L_G) = n - \sum_{r \in S} \chi_x(r) = n(1 - \text{Bias}(G_{\mathcal{C}} x)) = 2 \cdot \text{wt}(G_{\mathcal{C}} x).$$

CLAIM 6.1. *By preserving the weight of every codeword of $\mathcal{C}$ to a $(1 \pm \varepsilon)$ factor, we preserve the eigenvalues of L_G to a $(1 \pm \varepsilon)$ factor.*

Proof. This follows exactly from Fact 6.3. If we have a code sparsifier $\hat{C}$ for $\mathcal{C}$, then for any $x \in \mathbb{F}_2^k$, $\text{wt}(G_{\hat{\mathcal{C}}}) \in (1 \pm \varepsilon)\text{wt}(G_{\mathcal{C}} x)$. Because the codeword weights of the generating set and the eigenvalues of the Cayley graph are exactly equal, this $(1 \pm \varepsilon)$ approximation to the codeword weights implies that a Cayley graph with the same weighted generating set as used by the code sparsifier would be a $(1 \pm \varepsilon)$ spectral sparsifier by Fact 6.3. ∎

7 Applications to Sparsifying CSPs.

In this section, we show how to use our result on the sparsifiability of codes in the setting of CSPs. Specifically we first show that all affine predicates are sparsifiable, thus proving Theorem 1.5. Then we use this theorem to classify all Boolean ternary CSPs, CSPs on variables that take values in $\{0, 1\}$ where the predicate applies on three variables. This leads to a proof of Theorem 1.6.

7.1 Affine CSPs.

Recall that a predicate $P : \mathbb{F}_q^r \to \{0, 1\}$ is an *affine* predicate if there exist elements $a_0, a_1, \dots, a_r \in \mathbb{F}_q$ such that $P(b_1, \dots, b_r) = 0$ if and only if $a_0 + \sum_i a_i b_i = 0$ (over $\mathbb{F}_q$). We say further that P is *linear* if $a_0 = 0$.

The proof of Theorem 1.5 is completely straightforward if P is linear, given the definition of a linear code. The extension to the affine case uses a simple reduction from the affine case to the linear case (with one extra variable).

Proof. [Proof of Theorem 1.5] Given an instance Φ of CSP($\mathcal{P}$) with variables $x_1, \dots, x_k$ and constraints $C_1, \dots, C_n$ where $C_j = P^{(j)}(x_{(j),1}, \dots, x_{(j),r_j})$, we will create a code $\mathcal{C} \subseteq \mathbb{F}_q^n$ of dimension k generated by the matrix $G \in \mathbb{F}_q^{n \times k}$, where each row of the generating matrix corresponds to a single constraint. Let $P^{(j)} \in \mathcal{P}$ denote the predicate showing up in the jth constraint of our CSP instance. We start with the case that $P^{(j)}$ is linear with elements $a_{(j),1}, \dots, a_{(j),r_j} \in \mathbb{F}_q$ being such that $P^{(j)}(b_1, \dots, b_{r_j}) = 0$ if and only if $\sum_i a_{(j),i} b_i = 0$. Then, in the corresponding jth row of the generating matrix, for each $i \in [r_j]$, we place $a_{(j),i}$ in the column corresponding to variable $x_{(j),i}$, and leave all other entries in the row to be 0. It is straightforward to verify that for an assignment $x \in \mathbb{F}_q^k$, $(Gx)_j = 0$ if and only if C_j is unsatisfied. Thus $\text{wt}(Gx)$ counts the number of satisfied constraints for assignment x and thus a code sparisifier for $\mathcal{C}$ is a sparsifier for the instance Φ of CSP(P).

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Now considering the case of a general affine $P^{(j)}$ given by $P^{(j)}(b_1,\ldots,b_r)=0$ if and only if $a_{(j),0}+\sum_i a_{(j),i}b_i = 0$. Now let $\widehat{P^{(j)}}(b_0,\ldots,b_r)=\sum_{i=0}^{r} a_{(j),i}b_i$. Note that $\widehat{P^{(j)}}$ is linear. Given an instance Φ of CSP($\mathcal{P}$) on variables $x_1,\ldots,x_k$ with constraints $C_1,\ldots,C_n$ where $C_j = P^{(j)}(x_{(j),1},\ldots,x_{(j),r_j})$, let $\hat{\Phi}$ be the instance of CSP($\mathcal{P}$) on variables $x_0,\ldots,x_k$ with constraints $\hat{C}_1,\ldots,\hat{C}_n$ given by $\hat{C}_j = \widehat{P^{(j)}}(x_0,x_{(j),1},\ldots,x_{(j),r})$. We note that an assignment $x\in\mathbb{F}_q^k$ for Φ corresponds to the assignment $(1,x)\in\mathbb{F}_q^{k+1}$ to $\hat{\Phi}$ and so a sparsifier for $\hat{\Phi}$ (available from the previous paragraph) also sparsifies Φ. This concludes the proof. ∎

A major open question from the work of [3] was the sparsifiability of XOR predicates. Even on 3 variables, it was not known if the predicate $P(x_1,x_2,x_3)=x_1\oplus x_2\oplus x_3$ was sparsifiable to near-linear size. As a consequence of Theorem 1.5, we get the following result that resolves this question.

COROLLARY 7.1. *On a universe of k variables, any CSP with r-XOR predicates for $1\le r\le k$ is $(1\pm\varepsilon)$ sparsifiable to size $\widetilde{O}(k/\varepsilon^2)$.*

7.2 Ternary Boolean Predicates. We now turn to the classification of ternary Boolean predicates. Recall that a predicate $P:\{0,1\}^r\to\{0,1\}$ has *an affine projection to AND* if there exists a function $\pi:[r]\to\{0,1,x,\neg x,y,\neg y\}$ such that $\mathrm{AND}(x,y)=P(\pi(1),\ldots,\pi(r))$. We wish to prove that $P:\{0,1\}^3\to\{0,1\}$ is sparsifiable nearly linear size if and only if it has no affine projection to AND (Theorem 1.6).

The hardness result follows from a well-known result showing that the dicut problem is not sparsifiable to subquadratic size [7], which in our language is equivalent to saying that the binary AND predicate is not sparsifiable. (We include a precise statement and proof below for completeness — see Lemma 7.1.) Extending this to all predicates that have an affine projection to AND is simple (and holds for general r). This is stated and proved as Lemma 7.2 below. The bulk of the section then does a case analysis and shows that all ternary Boolean predicates that do not have an affine projection to AND can be sparsified by appealing to Theorem 1.5.

Non-sparsifiability of predicates with affine projection to AND.

LEMMA 7.1. ([7]) *For every $\epsilon\in[0,1)$, every $(1\pm\epsilon)$-sparsifier of size s for CSP(AND) on k variables requires $s=\Omega(k^2)$.*

The proof is actually more general and shows that any "sketch" of an instance Φ of CSP(AND) requires $\Omega(k^2)$ bits.

Proof. Let S and w_S be a $(1\pm\epsilon)$ sparsifier of size s of a CSP instance Φ on variables $x_1,\ldots,x_k$. Given a sparsifier, i.e., a subset of the constraints S and a pair $(i,j)\in\binom{k}{2}$, consider the weight of the constraints satisfied by the assignment x^{ij} given by $x^{ij}_k=1$ if $k\in\{i,j\}$ and 0 otherwise. This weight is positive in Φ if and only if the constraint $x_i\wedge x_j$ appears in Φ. Thus this weight is positive in the weighted sparsified instance using constraints from S if and only if the constraint $x_i\wedge x_j$ appears in Φ (since $\epsilon<1$); and furthermore the weight is positive in the unweighted sparsfied instance on S if and only if the constraint $x_i\wedge x_j$ appears in Φ. (The presence of the weights only affect the weight of the constraints that are satisfied, but not whether the number is positive or not.) Since this is true to every $(i,j)\in\binom{k}{2}$, it follows that S allows us to reconstruct $\Omega(k^2)$ independent bits of information about Φ and thus by the pigeonhole principle $|S|\ge\binom{k}{2}=\Omega(k^2)$ (for some instance Φ). ∎

LEMMA 7.2. *For every r, if a predicate $P:\{0,1\}^r\to\{0,1\}$ has an affine projection to AND, then for every $\epsilon\in[0,1)$, every $(1\pm\epsilon)$-sparsifier of size s for CSP(P) requires $s=\Omega_r(k^2)$.*

Proof. See [16] for the full proof. ∎

Sparsifying 3-CSPs with no affine projections to AND. Finally we show that if a predicate $P:\{0,1\}^3\to\{0,1\}$ has no affine projection to AND, then there exists linear-size sparsifiers for P. We will make use of the following corollary of Theorem 1.5:

COROLLARY 7.2. *Suppose there exists a linear equation $E(x_1,x_2,x_3)=ax_1+bx_2+cx_3+d \mod p$ (for p a prime) such that the unsatisfying assignmments to a predicate $P(x_1,x_2,x_3):\{0,1\}^3\to\{0,1\}$ are exactly the assigments to x_1,x_2,x_3 such that E evaluates to 0, then a valued CSP containing this predicate can be sparsified to size $\widetilde{O}(k\log(p)/\varepsilon^2)$.*

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

We note that the corollary is immediate from Theorem 1.5, which even allows the variables to take values in all of $\mathbb{Z}_p$ while we only need them to take values in $\{0,1\}$. Restricting the set of assignments preserves sparsification and so the sparsifier from Theorem 1.5 certainly suffices to get Corollary 7.2.

With this in hand, we will now use a case by case analysis to show how to write any predicate $P : \{0,1\}^3 \to \{0,1\}$ with no affine projection to AND as a linear equation modulo some prime. We state four claims that cover the different cases, and prove them in turn. Given the four claims, and Lemma 7.2, the proof of Theorem 1.6 is immediate.

CLAIM 7.1. *If $P : \{0,1\}^3 \to \{0,1\}$ has zero, six, seven or eight satisfying assignments then P is sparsifiable to nearly linear size.*

CLAIM 7.2. *If $P : \{0,1\}^3 \to \{0,1\}$ has five satisfying assignments and P has no affine projections to AND then P is sparsfiable to nearly linear size.*

CLAIM 7.3. *If $P : \{0,1\}^3 \to \{0,1\}$ has four satisfying assignments and P has no affine projections to AND then P is sparsfiable to nearly linear size.*

CLAIM 7.4. *If $P : \{0,1\}^3 \to \{0,1\}$ has one, two or three satisfying assignments then P has an affine projection to AND.*

Proof. [Proof of Theorem 1.5] If P has an affine projection to AND, then by Lemma 7.2, P has no subquadratic sized sparsifiers. So assume P has no affine projections to AND. Then by Claim 7.4 (in contrapositive form) P has at least four satisfying assignments. And Claim 7.1-Claim 7.3 show that in all remaining cases P is sparsifiable to nearly linear size. ∎

Thus all that remains is to prove Claim 7.1-Claim 7.4. For these proofs, please see [16].

8 Application to Hypergraph Cut Sparsifiers.

In this section, we will show how our result implies the existence of near-linear size hypergraph cut sparsifiers. First, we introduce the definition of a hypergraph.

DEFINITION 8.1. *A* hypergraph $G = (V, E)$ *is a set of n vertices V, along with a set of hyperedges E. Each hyperedge e is a subset of V, of any size.*

Next, we introduce the definition of a *cut* in a hypergraph.

DEFINITION 8.2. *For a hypergraph $G = (V, E)$, a cut in the hypergraph is a non-empty subset $S \subset V$. The size of the cut S is the number of hyperedges in E that are not completely contained in S or $V - S$. Intuitively, this is the number of edges that cross between $S, V - S$. We use $\delta_S(G)$ to denote the crossing edges corresponding to S in G.*

With this, we can then state a consequence of our main result in the setting of hypergraphs.

COROLLARY 8.1. *For a hypergraph $G = (V, E)$ on k vertices, there exists a weighted sub-hypergraph $\hat{G}$ of G with $\widetilde{O}(k/\varepsilon^2)$ hyperedges, such that for any subset $S \subseteq V$,*

$$(1-\varepsilon)\mathrm{wt}(\delta_G(S)) \leq \mathrm{wt}(\delta_{\hat{G}}(S)) \leq (1+\varepsilon)\mathrm{wt}(\delta_G(S)).$$

At a high level, our proof takes advantage of the fact that we showed the existence of code sparsifiers over *any* arbitrary field $\mathbb{F}_q$. In particular, for a hypergraph on k vertices, we will choose a prime q between k and $2k$. Then, we will create a generating matrix for a code over $\mathbb{F}_q$ where each row of the generating matrix corresponds to a hyperedge in the hypergraph. To start, we create a generating matrix with k columns. Now, for any hyperedge by e, we denote its size by $|e|$. If we analyze the row of the generating matrix corresponding to edge e, we then place a 1 in the columns corresponding to vertices $e_1, \ldots e_{|e|-1}$. For $e_{|e|}$ (i.e. the final vertex contained in the hyperedge), we place the value $q - |e| + 1$. In doing so, the row-sum of any row of the generating matrix will be exactly 0. Indeed, for any $\{0,1\}$ weighted linear combination of the columns of this generating matrix, a row is identically 0 if either *all* of the vertices corresponding to the hyperedge are included in the linear combination, or none of them are. This is exactly the definition of a hypergraph cut.

First, we use a basic number theoretic result:

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

FACT 8.1. (BERTRAND'S POSTULATE) *For any positive integer n, there exists a prime between n and $2n$.*

Thus, for any hypergraph on k vertices, we can find a prime number between $k, 2k$. Next, we define more specifically the generating matrix corresponding to a hypergraph:

DEFINITION 8.3. *For a hypergraph $H = (V, E)$ on k vertices, let q be a prime between $k, 2k$ as guaranteed by Fact 8.1. Let G be a generating matrix of a code defined over $\mathbb{F}_q^{|E|}$, and let G have k columns. Now, for any hyperedge $e = v_1, \dots v_{|e|}$, let G_{e,v_i} be 1 if $i \leq |e| - 1$, and $q - |e| + 1$ if $i = |e|$. All other entries in the row are zero. Call this generating matrix G the* generating matrix associated with H.

REMARK 8.1. *Let G be the generating matrix associated with a hypergraph H. Let $S \subseteq [k]$, and let x be the indicator vector for S. Then,*

$$\text{wt}(\delta_H(S)) = \text{wt}(Gx).$$

Proof. Consider any such $S \subseteq [k]$. $\text{wt}(\delta_H(S))$ is exactly the number of hyperedges that are not completely contained in S or $V - S$. Now, $\text{wt}(Gx)$ is the number of non-zero entries in Gx. By our construction of G, the only way for a $\{0, 1\}$ row-sum of G to be zero is when a $\{0, 1\}$ vector assigns either all 0's, or all 1's to the corresponding vertices of the hyperedge. This is exactly the same as the hyperedge being completely contained in S or $V - S$. ∎

CLAIM 8.1. *Let G be the generating matrix associated with a hypergraph H. If, there exists a sparsifier $\hat{G}$ such that for every message $x \in \mathbb{F}_q^k$, $\text{wt}(\hat{G}x) \in (1 \pm \varepsilon)\text{wt}(Gx)$, then if we select the corresponding edges of H with the same weights as in $\hat{G}$, we will recover a $(1 \pm \varepsilon)$ hypergraph cut sparsifier for H.*

Proof. By the previous Remark, for any set $S \subseteq [k]$, and x being the indicator vector for S,

$$\text{wt}(\delta_H(S)) = \text{wt}(Gx).$$

Further, given $\hat{G}$, we can create the corresponding hypergraph $\hat{H}$. It is still true that

$$\text{wt}(\delta_{\hat{H}}(S)) = \text{wt}(Gx).$$

Thus, we conclude that for any $S, x = \mathbf{1}[S]$,

$$(1-\varepsilon)\text{wt}(\delta_H(S)) = (1-\varepsilon)\text{wt}(Gx) \leq \text{wt}(\hat{G}x) = \text{wt}(\delta_{\hat{H}}(S)) \leq (1+\varepsilon)\text{wt}(Gx) = (1+\varepsilon)\text{wt}(\delta_H(S)).$$

It follows that the hypergraph associated with $\hat{G}$ is indeed a hypergraph cut-sparsifier for H. ∎

COROLLARY 8.2. *For any hypergraph H on k vertices, there exists a hypergraph cut sparsifier of H with $\widetilde{O}(k/\varepsilon^2)$ weighted hyperedges.*

Proof. Let G be the generating matrix associated with H. Let $\hat{G}$ be the $(1 \pm \varepsilon)$ code-sparsifier for the code generated by G. Note that because $q \leq 2k$, the number of rows in $\hat{G}$ is

$$\widetilde{O}(k \log q/\varepsilon^2) = \widetilde{O}(k/\varepsilon^2).$$

By the previous claim, we can then let $\hat{H}$ be the hypergraph associated with $\hat{G}$. ∎

Finally, by using Corollary 2.2, we can state a novel fact about the decomposition of hypergraphs.

COROLLARY 8.3. *For any hypergraph H on n vertices, for any integer $d \geq 1$, there exists a set of at most nd hyperedges, such that upon their removal, the resulting hypergraph has at most $(2n)^{2\alpha}$ cuts of size $\leq \alpha d$.*

Proof. Let G be the generating matrix associated with H. By the previous corollary, it follows that there exists a prime $q \leq 2n$, and a set of at most nd rows we can remove from G such that the number of codewords of weight $\leq \alpha d$ is at most $q^\alpha \cdot \binom{k}{\alpha}$. However, each such codeword corresponds with a possible cut in the graph of size at most αd. Hence, the number of possible cuts in the graph of size $\leq \alpha d$ is at most $(2n)^{2\alpha}$.

Note that again, if two separate vertex cuts $S_1, V - S_1$ and $S_2, V - S_2$ lead to exactly the same hyperedges being cut, we do not consider these to be separate cuts. Indeed, when we bound the number of cuts, we are bounding the number of distinct sets of hyperedges being cut of a given size. ∎

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

9 Conclusions.

In this work, we showed that for any linear code $\mathcal{C} \subseteq \mathbb{F}_q^n$ of dimension k, there exists a weighted set S of $\widetilde{O}(k \log(q)/\varepsilon^2)$ coordinates, such that for any codeword $c \in \mathcal{C}$, the quantity $\mathrm{wt}(c|_S)$ is a $(1 \pm \varepsilon)$ approximation to $\mathrm{wt}(c)$. This result provides a unified approach to recover known results about existence of near-linear size graph and hypergraph cut-sparsifiers, as well as some new results that include near-linear size Cayley-graph sparsifiers of Cayley graphs over $\mathbb{F}_2^k$, and near-linear size sparsifiers for a broader class of CSPs than were previously known.

The existential nature of our sparsification result raises the following natural question. Is there a poly-time algorithm to find the decomposition of Theorem 2.1? Alternately, is there a poly-time algorithm to compute code sparsifiers of near-linear size? The NP-hardness of the Minimum Distance Problem (MDP) for linear codes makes it particularly hard to implement Algorithm 4 (which explicitly calculates the minimum distance) and likewise finding the decomposition of Theorem 2.1 for arbitrary d can in *some* cases solve the MDP as well.

Another interesting direction for future work is to extend our classification theorem for sparsifiability of CSPs to predicates of arity greater than 3.

Acknowledgments.

We thank Salil Vadhan for pointing out the connection between codes and the eigenvalues of the Laplacians of Cayley graphs over $\mathbb{F}_2^k$.

References

[1] J. D. Batson, D. A. Spielman, and N. Srivastava, *Twice-ramanujan sparsifiers*, in Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009, M. Mitzenmacher, ed., ACM, 2009, pp. 255–262.

[2] A. A. Benczúr and D. R. Karger, *Approximating* s-t *minimum cuts in* $\tilde{\mathrm{O}}(\mathrm{n}^2)$ *time*, in Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996, G. L. Miller, ed., ACM, 1996, pp. 47–55.

[3] S. Butti and S. Zivný, *Sparsification of binary csps*, SIAM J. Discret. Math., 34 (2020), pp. 825–842.

[4] J. Chen, H. Sun, D. P. Woodruff, and Q. Zhang, *Communication-optimal distributed clustering*, in Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain, D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, eds., 2016, pp. 3720–3728.

[5] Y. Chen, S. Khanna, and A. Nagda, *Near-linear size hypergraph cut sparsifiers*, in 61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020, S. Irani, ed., IEEE, 2020, pp. 61–72.

[6] M. B. Cohen, R. Kyng, G. L. Miller, J. W. Pachocki, R. Peng, A. B. Rao, and S. C. Xu, *Solving SDD linear systems in nearly* $\mathrm{m}\log^{1/2}\mathrm{n}$ *time*, in Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014, D. B. Shmoys, ed., ACM, 2014, pp. 343–352.

[7] A. Filtser and R. Krauthgamer, *Sparsification of two-variable valued constraint satisfaction problems*, SIAM J. Discret. Math., 31 (2017), pp. 1263–1276.

[8] W. S. Fung, R. Hariharan, N. J. Harvey, and D. Panigrahi, *A general framework for graph sparsification*, in Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing, STOC '11, New York, NY, USA, 2011, Association for Computing Machinery, p. 71–80.

[9] A. Jambulapati, J. R. Lee, Y. P. Liu, and A. Sidford, *Sparsifying sums of norms*, CoRR, abs/2305.09049 (2023).

[10] A. Jambulapati and A. Sidford, *Ultrasparse ultrasparsifiers and faster laplacian system solvers*, CoRR, abs/2011.08806 (2020).

[11] M. Kapralov, R. Krauthgamer, J. Tardos, and Y. Yoshida, *Towards tight bounds for spectral sparsification of hypergraphs*, in STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021, S. Khuller and V. V. Williams, eds., ACM, 2021, pp. 598–611.

[12] D. R. Karger, *Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm*, in Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA, V. Ramachandran, ed., ACM/SIAM, 1993, pp. 21–30.

[13] ———, *Using randomized sparsification to approximate minimum cuts*, in Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia, USA, D. D. Sleator, ed., ACM/SIAM, 1994, pp. 424–432.

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited

[14] ———, *Random sampling in cut, flow, and network design problems*, Math. Oper. Res., 24 (1999), pp. 383–413.
[15] J. A. Kelner, Y. T. Lee, L. Orecchia, and A. Sidford, *An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations*, in Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014, C. Chekuri, ed., SIAM, 2014, pp. 217–226.
[16] S. Khanna, A. L. Putterman, and M. Sudan, *Code sparsification and its applications*, CoRR, abs/2311.00788 (2023).
[17] D. Kogan and R. Krauthgamer, *Sketching cuts in graphs and hypergraphs*, in Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015, T. Roughgarden, ed., ACM, 2015, pp. 367–376.
[18] R. Kyng, Y. T. Lee, R. Peng, S. Sachdeva, and D. A. Spielman, *Sparsified cholesky and multigrid solvers for connection laplacians*, in Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016, D. Wichs and Y. Mansour, eds., ACM, 2016, pp. 842–850.
[19] Y. T. Lee and H. Sun, *Constructing linear-sized spectral sparsification in almost-linear time*, SIAM J. Comput., 47 (2018), pp. 2315–2336.
[20] R. Peng, *Approximate undirected maximum flows in* O*(*m*polylog(*n*)) time*, in Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016, R. Krauthgamer, ed., SIAM, 2016, pp. 1862–1867.
[21] J. Sherman, *Nearly maximum flows in nearly linear time*, in 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA, IEEE Computer Society, 2013, pp. 263–269.
[22] D. A. Spielman and S. Teng, *Spectral sparsification of graphs*, SIAM J. Comput., 40 (2011), pp. 981–1025.

Downloaded 03/01/25 to 173.48.140.147 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © 2024 by SIAM
Unauthorized reproduction of this article is prohibited