Streaming complexity of CSPs with randomly ordered constraints*

Raghuvansh R. Saxena† Noah Singer‡ Madhu Sudan§ Santhoshini Velusamy¶

Abstract

We initiate a study of the streaming complexity of constraint satisfaction problems (CSPs) when the constraints arrive in a random order. We show that there exists a CSP, namely Max-DICUT, for which random ordering makes a provable difference. Whereas a $4/9 \approx 0.445$ approximation of DICUT requires $\Omega(\sqrt{n})$ space with adversarial ordering, we show that with random ordering of constraints there exists a 0.483-approximation algorithm that only needs $O(\log n)$ space. We also give new algorithms for Max-DICUT in variants of the adversarial ordering setting. Specifically, we give a two-pass $O(\log n)$ space 0.483-approximation algorithm for general graphs and a single-pass $\widetilde{O}(\sqrt{n})$ space 0.483-approximation algorithm for bounded-degree graphs.

On the negative side, we prove that CSPs where the satisfying assignments of the constraints support a one-wise independent distribution require $\Omega(\sqrt{n})$-space for any non-trivial approximation, even when the constraints are randomly ordered. This was previously known only for adversarially ordered constraints. Extending the results to randomly ordered constraints requires switching the hard instances from a union of random matchings to simple Erdős-Renyi random (hyper)graphs and extending tools that can perform Fourier analysis on such instances.

The only CSP to have been considered previously with random ordering is Max-CUT where the ordering is not known to change the approximability. Specifically it is known to be as hard to approximate with random ordering as with adversarial ordering, for $o(\sqrt{n})$ space algorithms. Our results show a richer variety of possibilities and motivate further study of CSPs with randomly ordered constraints.

1 Introduction

In this paper we consider the streaming complexity of solving constraint satisfaction problems (CSPs) approximately with randomly ordered constraints. We introduce these terms below before turning to the context and our work. Readers familiar with these topics may safely skip to Section 1.1.

Constraint satisfaction problems: A *constraint satisfaction problem (CSP)* is described by a family of predicates $\mathcal{F} \subseteq \{f : \mathbb{Z}_q^k \to \{0,1\}\}$ where $k, q \in \mathbb{N}$ and $\mathbb{Z}_q = \{0, \dots, q-1\}$. Given such a family $\mathcal{F}$, an instance Ψ of the problem $\mathsf{Max\text{-}CSP}(\mathcal{F})$ on n variables is described by m constraints $C_1, \dots, C_m$ where for $i \in [m]$, $C_i = (f_i, \mathbf{j}(i) = (j_1(i), \dots, j_k(i)))$ with $f_i \in \mathcal{F}$ and $\mathbf{j}(i)$ is a sequence of k distinct elements of $[n]$. An assignment to the n variables is given by $\mathbf{a} \in \mathbb{Z}_q^n$. The assignment satisfies C_i if $C_i(\mathbf{a}) := f_i(a_{j_1(i)}, \dots, a_{j_k(i)}) = 1$ and the value of the assignment on the instance Ψ is given by $\mathsf{val}_\Psi(\mathbf{a}) = \frac{1}{m}\sum_{i \in [m]} C_i(\mathbf{a})$. The goal of $\mathsf{Max\text{-}CSP}(\mathcal{F})$ is to compute $\mathsf{val}_\Psi := \max_{\mathbf{a} \in \mathbb{Z}_q^n}\{\mathsf{val}_\Psi(\mathbf{a})\}$. We will also be interested in approximation algorithms $\mathbf{ALG}$: Given $\alpha \in [0,1]$, an α-approximation algorithm to $\mathsf{Max\text{-}CSP}(\mathcal{F})$ is one whose output satisfies $\alpha \cdot \mathsf{val}_\Psi \leq \mathbf{ALG}(\Psi) \leq \mathsf{val}_\Psi$ for every instance Ψ.

Many natural problems can be expressed as CSPs. One example of particular interest to this paper is the problem $\mathsf{Max\text{-}DICUT} = \mathsf{Max\text{-}CSP}(\{\mathsf{DICUT}\})$, where the predicate $\mathsf{DICUT} : \mathbb{Z}_2^2 \to \{0,1\}$ is defined by $\mathsf{DICUT}(x,y) = (1-x)y$ (with the arithmetic being over $\mathbb{Z}_2$). Max-DICUT can equivalently be viewed as a *graph* problem in which variables correspond to vertices and constraints correspond to edges. The goal is then to estimate the size of the highest-value "directed partition" (i.e., $\{0,1\}$-assignment) of the vertices, where the value of a partition is the number of edges from 1-vertices to 0-vertices.

*The full version of the paper can be accessed at `https://arxiv.org/abs/2207.07158`. In this version, we omit many of the proofs.

†Microsoft Research. Email: `raghuvansh.saxena@gmail.com`

‡Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, and Harvard College, Harvard University, Cambridge, MA, USA. Supported by an NSF Graduate Research Fellowship (Award DGE2140739). Email: `ngsinger@cs.cmu.edu`.

§School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA. Supported in part by a Simons Investigator Award and NSF Awards CCF 1715187 and CCF 2152413. Email: `madhu@cs.harvard.edu`.

¶School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA. Supported in part by a Google Ph.D. Fellowship, a Simons Investigator Award to Madhu Sudan, and NSF Awards CCF 1715187 and CCF 2152413. Email: `svelusamy@g.harvard.edu`.

Streaming Algorithms: The class of algorithms we consider (and rule out) are randomized streaming algorithms. Inputs to these algorithms arrive as a stream of elements, in our case a stream of constraints. We consider algorithms that use some bounded amount of space, denoted $s(n)$, to process the stream and produce their output. They may toss their own coins to process the stream. In this work we focus mainly on algorithms whose inputs are *randomly ordered*, i.e., given an instance m on variables with constraints $C_1, \ldots, C_m$, a permutation $\pi : [m] \to [m]$ is chosen uniformly at random and the constraints arrive in the order $C_{\pi(1)}, \ldots, C_{\pi(m)}$. We say that an algorithm is correct if it outputs a correct answer[1] with probability 2/3, where the probability is both over internal coin tosses and over the random arrival order of the input.

1.1 Previous work.

The recent years have seen a significant amount of research on the streaming complexity of approximating CSPs with adversarial order of arrival. We refer the reader to Chou, Golovnev, Sudan and Velusamy [5] for some of the history. (See also [21] and [23] for some broader surveys.) One key result from this line of research is a dichotomy result for "sketching algorithms" to approximate all CSPs, while getting dichotomies in the more general streaming context for many subclasses. A sketching algorithm is a streaming algorithm that works by compressing substreams into small summaries called sketches with the feature that the sketch of a concatenation of two streams can be obtained from sketches of the two component streams. All known algorithms for CSPs (with proven guarantees on approximation) are sketching algorithms motivating the current work. In this work we consider a weakening of the input model, to random ordering of constraints, to explore the possibility of other algorithms, or to rule them out.

Turning to random order in graph streaming problems, Kapralov, Khanna, and Sudan [13] gave a polylog(n)-space random-order streaming algorithm for polylog(n)-approximating the maximum matching problem; Kapralov, Mitrović, Norouzi-Fard, and Tardos [16] improved the exponent in the approximation factor. Another line of works [17, 19] explores "generic" ways in which sublinear-time algorithms for graph problems can be transformed into random-ordering streaming algorithms; the latter work establishes provable separations for random-ordering streaming from adversarial-order streaming for problems including estimating the number of connected components and the minimum spanning tree weight. Most relevantly, in the context of CSPs, Kapralov, Khanna, and Sudan [14] showed that the CSP $\mathsf{Max\text{-}CUT} = \mathsf{Max\text{-}CSP}(\{\mathsf{CUT}\})$ where $\mathsf{CUT} : \mathbb{Z}_2^2 \to \{0,1\}$ is defined by $\mathsf{CUT}(a,b) = a + b$ cannot be nontrivially approximated by $o(\sqrt{n})$-space streaming algorithms even in the random-order setting. But other than [14], the previous works on random-order streaming have not studied CSPs, and in particular, none of the previous works suggest that random order of arrival could lead to any algorithmic improvement.

1.2 Main results.

In this paper, we present both positive (i.e., algorithmic) and negative (i.e., hardness) results on the usefulness of randomly-ordered streams for approximating CSPs, in comparison to adversarially-ordered streams.

1.2.1 Positive results.

Our main positive result asserts that there exists a constraint satisfaction problem where random arrival of constraints provably leads to better approximation with $o(\sqrt{n})$ space.

THEOREM 1.1. *There exists a $O(\log n)$-space streaming algorithm that outputs a .483-approximation to the $\mathsf{Max\text{-}DICUT}$ value of directed graphs on n vertices whose edges arrive in a random order.*

This theorem is restated as Theorem 3.1 and proved in Section 3.1.

The result above should be contrasted with the result of Chou, Golovnev and Velusamy [6] who show that for every $\epsilon > 0$, a streaming algorithm that achieves a $(4/9 + \epsilon)$-approximation of $\mathsf{Max\text{-}DICUT}$ requires $\Omega(\sqrt{n})$ space when the constraints are ordered *adversarially.* (Note $4/9 = 0.444\ldots$.) Their lower bound holds in the general setting of streaming algorithms, with a matching upper bound using a sketching algorithm. Our algorithm is not a sketching algorithm. This is the only result to our knowledge for a streaming CSP (even with assumptions on arrival order) where a non-sketching algorithm outperforms known sketching algorithms.

Indeed the ideas from this algorithm help in contexts other than just the random arrival order and we describe some of these consequences next.

[1]Recall that approximation algorithms are not required to output any one fixed answer. An answer is correct on input Ψ if it lies in the interval $[\alpha \cdot \mathsf{val}_\Psi, \mathsf{val}_\Psi]$.

1.2.2 Positive results in other streaming models.

The algorithm used to prove Theorem 1.1 can also be straightforwardly adapted to the setting of *two-pass* algorithms with adversarial order as asserted below.

Theorem 1.2. *There exists a $O(\log n)$-space two-pass streaming algorithm that outputs a .483-approximation to the Max-DICUT value of directed graphs on n vertices under adversarial ordering of edges.*

This theorem is restated in more detail as Theorem 3.2 and proved in Section 3.2. The 2-pass algorithm answers an open question in [5], perhaps with an unexpected answer.

Finally, we also show how the algorithm can be further modified to get the same approximation to Max-DICUT using $\widetilde{O}(\sqrt{n})$ space with a single-pass streaming algorithm in *bounded-degree* graphs with adversarial ordering of edges.

Theorem 1.3. *There exists a $\widetilde{O}(\sqrt{n})$-space streaming algorithm that outputs a .483-approximation to the Max-DICUT value of bounded-degree directed graphs on n vertices under adversarial ordering of edges.*

Theorem 3.3 actually gives a more detailed relationship between the space needed and the maximum degree of the graph. It implies the theorem above and is proved in Section 3.3. We remark that [6] show that $o(\sqrt{n})$ space algorithms cannot get better than a 4/9-approximation and their proof actually holds even when the input graphs are of bounded-degree. Thus Theorem 1.3 establishes the significance of the $\sqrt{n}$-space threshold — again a result that may be somewhat surprising.

1.2.3 Negative results.

Returning to our main quest of understanding streaming CSPs in the random-ordering setting and motivated by the algorithmic potential demonstrated by Theorem 1.1 above, we re-explore negative results on streaming to see whether they might also apply to the random arrival ordering model. We show that for a broad class of constraint satisfaction problems, the known hardness results on streaming algorithms with adversarial ordering do indeed extend (with substantial changes in the analysis) to the case of randomly ordered constraints. We define the class of problems considered and the approximation lower bound achieved below, starting with the latter.

We say that an algorithm is *trivial* if its output is a constant (independent of the input). For a class of constraints $\mathcal{F}$, define $\rho_{\min}(\mathcal{F})$ to be the minimum (strictly, infimum) value val_Ψ over all instance Ψ of Max-CSP($\mathcal{F}$). (A priori, $\rho_{\min}(\mathcal{F})$ might not be computable given $\mathcal{F}$, but it is shown to be computable in [5].) Clearly, an algorithm that outputs $\rho = \rho_{\min}(\mathcal{F})$ on every instance is a valid, but trivial, ρ-approximation algorithm for Max-CSP($\mathcal{F}$). Motivated by this [5] define a problem to be *approximation-resistant* to a class of algorithms if for every $\epsilon > 0$ it does not have a $(\rho + \epsilon)$-approximation within the class. Our next theorem proves a broad class of CSPs to be approximation-resistant to $o(\sqrt{n})$-space single pass streaming algorithms, even with a random ordering of constraints.

We now turn to the class of problems covered by our theorem. We say a distribution $\mathcal{D}$ over $\mathbb{Z}_q^k$ has *uniform marginals* if when we sample $\mathbf{a} = (a_1, \ldots, a_k) \sim \mathcal{D}$, each a_i is distributed uniformly over $\mathbb{Z}_q$. We say a predicate $f : \mathbb{Z}_q^k \to \{0,1\}$ *supports one-wise independence* if there exists a distribution $\mathcal{D}$ supported on $f^{-1}(1)$ whose marginals are uniform. We say a family $\mathcal{F}$ *supports one-wise independence* if every $f \in \mathcal{F}$ supports one-wise independence. We say a family $\mathcal{F}$ *weakly supports one-wise independence* if there exists $\mathcal{F}' \subseteq \mathcal{F}$ supporting one-wise independence with $\rho_{\min}(\mathcal{F}') = \rho_{\min}(\mathcal{F})$. Our theorem below asserts the approximation resistance of Max-CSP($\mathcal{F}$) on randomly ordered instances when $\mathcal{F}$ weakly supports one-wise independence.

Theorem 1.4. *For every $k, q \in \mathbb{N}$ and $\mathcal{F}$ s.t. $\mathcal{F} \subseteq \{f : \mathbb{Z}_q^k \to \{0,1\}\}$ that weakly supports one-wise independence, Max-CSP($\mathcal{F}$) is approximation resistant to $o(\sqrt{n})$-space streaming algorithms in the random order model. That is, for every $\epsilon > 0$, there exists $\tau > 0$ such that every streaming algorithm which $(\rho_{\min}(\mathcal{F}) + \epsilon)$-approximates Max-CSP($\mathcal{F}$) in the random-order model uses at least $\tau\sqrt{n}$ space on instances with n variables.*

We assert that all known families that are known to be approximation-resistant to $o(\sqrt{n})$-space single pass streaming algorithms, even under adversarial ordering, weakly support one-wise independence [5]. Such problems include Max-CUT (and thus our result subsumes that of [14]), Max-qUniqueGames, Max-qColoring, and Max-kOR.[2] Our result thus strengthens our understanding of approximation resistance for the broadest class of problems where it was previously understood.

[2]Indeed, the question of proving random-ordering approximation-resistance for Max-qUniqueGames was posed by Guruswami and Tao [8, §5] when they proved adversarial-ordering hardness, and thus, we answer an open question of theirs.

1.3 Technical contributions.

Next, we describe some of the technical contributions of our results, beginning with the positive (algorithmic) side.

1.3.1 Positive results.

All streaming algorithms for CSPs in previous works [9, 6, 4, 5, 2] have been based on measuring generalizations of the "total bias" of CSP instances defined originally in [9]; this quantity, even in its richest form from [5], is a sum over the variables of some form of "bias", and can be computed using norm-sketching algorithms [10, 12, 1]. Bias, in turn, roughly measures whether, considering each constraint in which a variable appears independently, the variable prefers to take one value more often than others. More concretely, in the case of Max-DICUT, given a directed graph, the $\mathsf{bias}(i)$ of vertex i is simply $\frac{\mathsf{out\text{-}deg}(i)-\mathsf{in\text{-}deg}(i)}{\mathsf{out\text{-}deg}(i)+\mathsf{in\text{-}deg}(i)}$, where $\mathsf{out\text{-}deg}(i)$ and $\mathsf{in\text{-}deg}(i)$ denote the out- and in-degrees of i, respectively. Thus, $\mathsf{bias}(i)$ is a real number between -1 and 1; if $\mathsf{bias}(i) \approx 1$, i has mostly out-edges, so we should assign it to 0, while if $\mathsf{bias}(i) \approx -1$, it has mostly in-edges, so we should assign i to 1. The *total bias* of a Max-DICUT instance sums variables' biases weighted by their total degree. This quantity was defined in [9], and was observed to equal the ℓ_1-norm of the vector whose i-th entry is $\mathsf{out\text{-}deg}(i) - \mathsf{in\text{-}deg}(i)$, and thus it can be measured in the streaming setting.

Thus, a key contribution of our work is the first new *algorithmic* paradigm for streaming CSPs since the approach of [9]. This should be contrasted with the fact that many works [8, 15, 6, 4, 5, 22, 3] have made significant progress on the hardness front. Instead of estimating the *total* bias of the input graph, we build a *snapshot* of the graph: Specifically we merge vertices with (roughly) the same bias and estimate the fraction of edges that go from vertices of different bias. To get this snapshot information, we look at a representative sample of edges and consider the biases of their endpoints. Here is where we use the random arrival order of edges: We can sample typical edges at the beginning of the stream, and then we measure the bias of their endpoints over the rest of the stream. (So really our algorithm just needs the first few edges to be random, and the rest of the stream could even be ordered adversarially!)

Using this bias information to produce a cut is not trivial, but fortunately for us a previous work of Feige and Jozeph [7] analyzed exactly this question. They studied "oblivious algorithms" for Max-DICUT, which are algorithms which randomly assign each vertex independently based solely on its bias, and showed the existence of an α_{FJ}-approximation algorithm for some $\alpha_{\mathrm{FJ}} \in (0.483, 0.4899)$. Our theorem follows by appealing to their result. We remark that based on the trivial reduction from Max-CUT, Max-DICUT's approximability for $o(\sqrt{n})$-space algorithms with randomly ordered constraints is at most 1/2. And while [7] showed that oblivious algorithms cannot do better than 0.4899-approximations, it is quite possible that other quantities that can be easily estimated with random arrival orders (such as the number of copies of $O(1)$-vertex subgraphs, such as paths) could lead to 1/2-approximation algorithms.

The idea of computing a snapshot of the graph and then using that (via the Feige-Jozeph analysis [7]) to approximate the Max-DICUT value of a graph turns out to work in other streaming models as well. For instance, in the two-pass setting with adversarial ordering of the edges, we can pick a random sample of edges in the first pass and then use the second pass to compute the bias of the endpoints of the edges. This leads to an $O(\log n)$ space streaming 2-pass algorithm achieving the same approximation factor for Max-DICUT even in the adversarial arrival setting. In the case of bounded-degree graphs we are also able to compute snapshots with $\widetilde{O}(\sqrt{n})$-space when the edge arrival order is adversarial. This does require some additional care, as there's no obvious way to sample random edges. Instead, the general plan is to sample roughly $\sqrt{n}$ random (positive-degree) *vertices*, and estimate the snapshot based on edges in the induced subgraph on these vertices. One important implication of the bounded-degree assumption here is that it limits the dependence between which edges are included, and after some careful analysis this leads to a single-pass $\widetilde{O}(\sqrt{n})$-space algorithm for bounded-degree graphs achieving the same approximation factor for Max-DICUT.

1.3.2 Negative results.

Turning to the negative results that form the technical meat of this paper, we comment briefly on where previous works used the adversarial ordering and what we need to do to overcome it. Starting with [14], all hardness results for streaming Max-CSP($\mathcal{F}$) problems have been based on constructing so-called "**YES**" and "**NO**" distributions over instances which have high and low values, respectively (with high probability), and showing that these are indistinguishable by reducing from a one-way communication problem. Designing these distributions is typically a trade-off between desired properties for the streaming lower bound (e.g., optimizing the value gap between **YES** and **NO** instances) and technical considerations in terms of how to prove the appropriate communication lower bounds (and whether they even hold at all!). The distributions themselves result from a two-fold process: First, sample a random hypergraph, and then treat each hyperedge as a CSP constraint by labeling it with an appropriate predicate $f \in \mathcal{F}$. Indeed, this "labeling" is the only difference between the **YES** and **NO** distributions; typically, in the **NO** distribution the labels are completely random, while in the **YES** distribution they are selected to be consistent with some global assignment.

Now, consider the communication problem in which we split up hypergraph edges and labels among $T = O(1)$ of "players", and the players must distinguish between the **YES** and **NO** cases. At a high level, the technical complexity of such problems is closely connected to the structure of the hypergraphs that the players receive. In particular, it becomes necessary to analyze a counting problem involving $\mathbb{Z}_q$-labelings of edge-vertex incidences with sum constraints at vertices and density constraints on edges (see Eq. (5.4) below for a technical statement). In previous works aside from [14], each player's input hypergraph was a random (partial) *hypermatching.* Crucially, hypermatchings (of any particular size) are unique up to renaming of vertices. While this significantly simplifies the combinatorial analysis, it is not appropriate for proving random-ordering streaming lower bounds. This is because, in the communication-to-streaming reduction, the resultant stream of constraints is the concatenation of constraints contributed by each player; these streams will have the property that in each successive "chunk" of $\approx 1/T$ constraints, no variables are repeated, which is unlikely in a randomly-ordered stream. Thus, it is necessary to draw the players' input hypergraphs from a different distribution. In the case of $\mathsf{Max\text{-}CUT}$, with alphabet size $q = 2$ and arity $k = 2$, Kapralov *et al.* [14] instead worked with general random graphs. Such graphs are no longer unique up to renaming of vertices; there are many different equivalence classes, and each behaves differently in the proof of the lower bound. However, [14] manages this difficulty by showing that (1) cycles are unlikely, and (2) conditioned on cycle-freeness, each equivalence class corresponds to a union of paths with a certain length profile. It turns out that both the $k = 2$ and $q = 2$ assumptions are significantly helpful the analysis of [14]. If $k > 2$, we lose the decomposition into unions of paths, while if $q > 2$, we need to worry about different $\mathbb{Z}_q$-labelings even of the same path, and thus the length of paths comes into play.

Nevertheless, in our work, we manage to generalize to arbitrary $k, q \in \mathbb{N}$ by conducting a careful combinatorial analysis of connected component sizes in random hypergraphs (see Section 6). This allows us to develop streaming hardness results for all CSPs weakly supporting one-wise independence (Theorem 1.4). Indeed, we show that perfectly satisfiable instances (i.e., those with value 1) are indistinguishable from random instances with independent, uniformly random constraints!

1.4 Subsequent work. In a subsequent work [20], we design an algorithm which 0.483-approximates the $\mathsf{Max\text{-}DICUT}$ value of *arbitrary* directed graphs under adversarial ordering of edges. This extends the result of this paper for bounded-degree graphs (Theorem 1.3). The core of the algorithm in [20] is still the "bias snapshot" algorithm of Feige-Jozeph [7]. However, the challenge is now dealing with edges whose endpoints have widely varying degrees. In [20] the algorithm now considers many sparsifications of the input graph in parallel, with a wide range of parameters; correspondingly, the analysis of [20] roughly shows that the Feige-Jozeph algorithm [7] is "robust" to a variety of errors which result in this process, and is much more involved.

2 Preliminaries

For $n > 0$, we use 0^n to denote the all zeros vector of length n and $\mathcal{S}(n)$ to denote the set of all permutations mapping the set $[n]$ to itself. Let Σ be a set, $n \in \mathbb{N}$, and $\pi \in \mathcal{S}(n)$ be a permutation. For $\boldsymbol{\sigma} \in \Sigma^n$ and $i \in [n]$, we use σ_i to denote coordinate i of $\boldsymbol{\sigma}$ and $\pi(\boldsymbol{\sigma})$ to denote the vector $\sigma_{\pi(1)}, \sigma_{\pi(2)}, \ldots, \sigma_{\pi(n)}$. For $\boldsymbol{\sigma} \in \Sigma^*$, we use $|\boldsymbol{\sigma}|$ to denote the number of coordinates in $\boldsymbol{\sigma}$.

For a set S, we use $\Delta(S)$ to denote the set of all distributions whose support is S. We use the name of a distribution to denote its probability mass function (i.e., for $s \in S$ and $\mathcal{D} \in \Delta(S)$, $\mathcal{D}(s)$ denotes $\Pr_{t \sim \mathcal{D}}[s = t]$). For $k > 0$ and sets $S_1, S_2, \ldots, S_k$, we use $\Delta_{\mathsf{unif}}(S_1, S_2, \ldots, S_k)$ to denote the set of all distributions on the product set $S = S_1 \times S_2 \times \ldots \times S_k$ for which the marginal distribution on the set S_i, for all $i \in [k]$ is uniform. We simply write $\Delta_{\mathsf{unif}}(S)$ if the product decomposition is clear from context.

2.1 Definitions.

2.1.1 The random-order streaming model. Let Σ be an alphabet set. A Σ-*stream* is a string $\boldsymbol{\sigma} \in \Sigma^*$. A *deterministic streaming algorithm* **ALG** for Σ-streams is defined by the tuple:

$$\mathbf{ALG} = (S, \mathsf{mdfy}, \mathsf{out}),$$

where: (1) $S = \|\mathbf{ALG}\|$ is the space/memory required by the algorithm **ALG**, (2) $\mathsf{mdfy} = \Sigma \times \{0,1\}^S \to \{0,1\}^S$ is the function the algorithm uses to update its state upon reading a symbol from the stream, and (3) $\mathsf{out} = \{0,1\}^S \to \{0,1\}^S$ is the function the algorithm uses to compute its output from its state at the end of the stream. We shall suppress arguments on the right hand side when they are clear from context. We define a *randomized streaming algorithm* on Σ-streams to be a distribution over deterministic streaming algorithms. Additionally, the space required by a randomized streaming algorithm is the maximum space required by a deterministic algorithm in its support.

Execution of a streaming algorithm. Let Σ be an alphabet set and **ALG** be a (deterministic) algorithm for Σ-streams. For a stream $\boldsymbol{\sigma} \in \Sigma^*$ with $m = |\boldsymbol{\sigma}|$, the algorithm **ALG** acts on $\boldsymbol{\sigma}$ in m steps as follows. At the beginning (before step 1), the algorithm is the state $s_0 = 0^S$. Then, for $i \in [m]$, the algorithm reads the symbol σ_i and uses it to update its state by defining $s_i = \mathsf{mdfy}(\sigma_i, s_{i-1})$. Finally, after m steps, the algorithm outputs the value $\mathsf{out}(s_m)$.

Note that all the states of the algorithm and its final output are determined by its input $\boldsymbol{\sigma}$. For $i \in [m]$, we write $\mathbf{ALG}(\boldsymbol{\sigma}, i) \in \{0,1\}^S$ to denote the state after step i of the algorithm on input $\boldsymbol{\sigma}$. We define $\mathbf{ALG}(\boldsymbol{\sigma}, 0) = 0^S$ for convenience. Finally, we write $\mathbf{ALG}(\boldsymbol{\sigma}) \in \{0,1\}$ to denote the output of the algorithm on input $\boldsymbol{\sigma}$.

Computation using streaming algorithms. Let Σ be an alphabet set and $f : \Sigma^* \to \{0,1\}$ be a (possibly partial) function. For $p > 0$, we say that a randomized streaming algorithm $\mathcal{A}$ *computes* the function f in the random-order streaming model with probability p if for all $\boldsymbol{\sigma} \in \Sigma^*$, we have:

$$\Pr_{\mathbf{ALG} \sim \mathcal{A}, \pi \sim \mathcal{S}(|\boldsymbol{\sigma}|)}(\mathbf{ALG}(\pi(\boldsymbol{\sigma})) = f(\boldsymbol{\sigma})) \geq p.$$

Distinguishing using streaming algorithms. Let Σ be an alphabet set and $(\mathcal{Y}, \mathcal{N})$ be a pair of distributions over Σ^*. For $\delta \geq 0$, we say that a deterministic streaming algorithm **ALG** *distinguishes* between $\mathcal{Y}$ and $\mathcal{N}$ with *advantage* δ in the random-order streaming model if

$$\left| \Pr_{\boldsymbol{\sigma} \sim \mathcal{Y}, \pi \sim \mathcal{S}(|\boldsymbol{\sigma}|)}(\mathbf{ALG}(\pi(\boldsymbol{\sigma})) = 1) - \Pr_{\boldsymbol{\sigma} \sim \mathcal{N}, \pi \sim \mathcal{S}(|\boldsymbol{\sigma}|)}(\mathbf{ALG}(\pi(\boldsymbol{\sigma})) = 1) \right| \geq \delta.$$

We say that **ALG** distinguishes between $\mathcal{Y}$ and $\mathcal{N}$ with advantage δ in the worst case streaming model if:

$$\left| \Pr_{\boldsymbol{\sigma} \sim \mathcal{Y}}(\mathbf{ALG}(\boldsymbol{\sigma}) = 1) - \Pr_{\boldsymbol{\sigma} \sim \mathcal{N}}(\mathbf{ALG}(\boldsymbol{\sigma}) = 1) \right| \geq \delta.$$

We may sometimes refer to a pair $(\mathcal{Y}, \mathcal{N})$ of distributions as a streaming problem and say that "**ALG** solves the $(\mathcal{Y}, \mathcal{N})$-problem" instead of saying that "**ALG** distinguishes between $\mathcal{Y}$ and $\mathcal{N}$". The two notions of distinguishability are equivalent if the distributions $\mathcal{Y}$ and $\mathcal{N}$ are sufficiently symmetric:

LEMMA 2.1. *Let Σ be an alphabet set and $\mathcal{D}$ be a distribution over Σ^* such that for all $\boldsymbol{\sigma} \in \Sigma^*$ and $\pi \sim \mathcal{S}(|\boldsymbol{\sigma}|)$, we have $\mathcal{D}(\boldsymbol{\sigma}) = \mathcal{D}(\pi(\boldsymbol{\sigma}))$. Then, for all $\boldsymbol{\tau} \in \Sigma^*$, we have:*

$$\Pr_{\boldsymbol{\sigma} \sim \mathcal{D}}(\boldsymbol{\sigma} = \boldsymbol{\tau}) = \Pr_{\boldsymbol{\sigma} \sim \mathcal{D}, \pi \sim \mathcal{S}(|\boldsymbol{\sigma}|)}(\pi(\boldsymbol{\sigma}) = \boldsymbol{\tau}).$$

Proof. Let $\mathcal{D}'$ be the distribution on $\mathbb{N}$ obtained by sampling $\boldsymbol{\sigma}$ from $\mathcal{D}$ and outputting $|\boldsymbol{\sigma}|$. We can view the process of sampling $\boldsymbol{\sigma}$ from $\mathcal{D}$ and then sampling π from $\mathcal{S}(|\boldsymbol{\sigma}|)$ as the process of first sampling an integer $m \geq 0$ from $\mathcal{D}'$, then sampling a permutation π from $\mathcal{S}(m)$ and finally, a string $\boldsymbol{\sigma}$ from $\mathcal{D}$ conditioned on the fact that

$|\boldsymbol{\sigma}| = m$. Moreover, as $\pi(\boldsymbol{\sigma}) = \boldsymbol{\tau}$ can happen only if $m = |\boldsymbol{\tau}|$, we get (using $m = |\boldsymbol{\tau}|$):

$$\begin{aligned}
\Pr_{\boldsymbol{\sigma}\sim\mathcal{D},\pi\sim\mathcal{S}(|\boldsymbol{\sigma}|)}(\pi(\boldsymbol{\sigma}) = \boldsymbol{\tau}) &= \mathcal{D}'(m) \cdot \Pr_{\pi\sim\mathcal{S}(m),\boldsymbol{\sigma}\sim\mathcal{D}|_{|\boldsymbol{\sigma}|=m}}(\pi(\boldsymbol{\sigma}) = \boldsymbol{\tau}) \\
&= \mathcal{D}'(m) \cdot \frac{1}{m!} \cdot \sum_{\pi\in\mathcal{S}(m)} \Pr_{\boldsymbol{\sigma}\sim\mathcal{D}|_{|\boldsymbol{\sigma}|=m}}(\pi(\boldsymbol{\sigma}) = \boldsymbol{\tau}) \\
&= \mathcal{D}'(m) \cdot \frac{1}{m!} \cdot \sum_{\pi\in\mathcal{S}(m)} \Pr_{\boldsymbol{\sigma}\sim\mathcal{D}|_{|\boldsymbol{\sigma}|=m}}\big(\boldsymbol{\sigma} = \pi^{-1}(\boldsymbol{\tau})\big) \\
&= \mathcal{D}'(m) \cdot \frac{1}{m!} \cdot \sum_{\pi\in\mathcal{S}(m)} \Pr_{\boldsymbol{\sigma}\sim\mathcal{D}|_{|\boldsymbol{\sigma}|=m}}(\boldsymbol{\sigma} = \boldsymbol{\tau}) \\
&= \mathcal{D}'(m) \cdot \Pr_{\boldsymbol{\sigma}\sim\mathcal{D}|_{|\boldsymbol{\sigma}|=m}}(\boldsymbol{\sigma} = \boldsymbol{\tau}) \\
&= \Pr_{\boldsymbol{\sigma}\sim\mathcal{D}}(\boldsymbol{\sigma} = \boldsymbol{\tau}).
\end{aligned}$$

□

Corollary 2.1. (Random order to worst-case) *Let Σ be an alphabet set and $(\mathcal{Y}, \mathcal{N})$ be a pair of distributions over Σ^* such that for all $\boldsymbol{\sigma} \in \Sigma^*$ and $\pi \sim \mathcal{S}(|\boldsymbol{\sigma}|)$, we have $\mathcal{Y}(\boldsymbol{\sigma}) = \mathcal{Y}(\pi(\boldsymbol{\sigma}))$ and $\mathcal{N}(\boldsymbol{\sigma}) = \mathcal{N}(\pi(\boldsymbol{\sigma}))$. Then, for all $\delta \geq 0$ and any deterministic streaming algorithm* **ALG** *from Σ-streams, we have that* **ALG** *distinguishes between $\mathcal{Y}$ and $\mathcal{N}$ with advantage δ in the random-order streaming model if and only if* **ALG** *distinguishes between $\mathcal{Y}$ and $\mathcal{N}$ with advantage δ in the worst case streaming model.*

We shall also need the following connection between computation and distinguishing using streaming algorithms (which is sometimes referred to generally as "Yao's minimax principle"):

Fact 2.1. *Let Σ be an alphabet set, $f : \Sigma^* \to \{0, 1\}$ be a partial function, and $p > 0$. If there exists a randomized streaming algorithm $\mathcal{A}$ that computes the function f in the random-order streaming model with probability p, then for all distributions $\mathcal{Y}$ and $\mathcal{N}$ supported on $f^{-1}(1)$ and $f^{-1}(0)$ respectively, we have a deterministic streaming algorithm* **ALG**, $\|\mathbf{ALG}\| \leq \|\mathcal{A}\|$ *such that* **ALG** *distinguishes between $\mathcal{Y}$ and $\mathcal{N}$ with advantage $2 \cdot \left(p - \frac{1}{2}\right)$ in the random-order streaming model.*

2.2 The $\mathsf{Max\text{-}CSP}(\cdot)$ problem. Throughout this subsection, we let $q, k \in \mathbb{N}$ and $\mathcal{F}$ be a non-empty set of *predicate* functions mapping $\mathbb{Z}_q^k \to \{0, 1\}$. Let $n \geq k \in \mathbb{N}$. An *instance* Ψ of $\mathsf{Max\text{-}CSP}_n(\mathcal{F})$ is given by a sequence:

$$\Psi = (f_i, M_i)_{i>0} \in \left(\mathcal{F} \times \{0,1\}^{k\times n}\right)^*,$$

where, for all $i \in [|\Psi|]$, the matrix M_i is a $k \times n$ partial permutation matrix, *i.e.*, a matrix with $0, 1$ entries and exactly one 1 in each row and at most one 1 in every column. Let $m = |\Psi|$. Intuitively, Ψ can be seen as a sequence of m constraints, with constraint $i \in [m]$ requiring that the function f_i when applied to the k variables indicated by M_i evaluates to 1. Here, for $j \in [k]$ the j^{th} variable indicated by M_i is the unique column that has the 1 in row j of M_i.

Value of Ψ. For an assignment $\mathbf{x} \in \mathbb{Z}_q^n$ of the n variables, the fraction of satisfied constraints is given by:

$$\mathsf{val}_\Psi(\mathbf{x}) = \frac{1}{m} \cdot \sum_{i\in[m]} f_i(M_i\mathbf{x}). \tag{2.1}$$

We define the *value* of Ψ to be the largest fraction of the constraints that can be satisfied by an assignment:

$$\mathsf{val}_\Psi = \max_{\mathbf{x}\in\mathbb{Z}_q^n} \mathsf{val}_\Psi(\mathbf{x}). \tag{2.2}$$

The function $\rho_{\min}(\cdot)$. The best lower bound on the value of *every* instance of $\mathsf{Max\text{-}CSP}(\mathcal{F})$ is given by:

$$\rho_{\min}(\mathcal{F}) = \inf_{\substack{n\in\mathbb{N} \\ \Psi \text{ instance of } \mathsf{Max\text{-}CSP}_n(\mathcal{F})}} \mathsf{val}_{\Psi}. \tag{2.3}$$

The following lemma, taken from [5], gives an equivalent formulation of the function $\rho(\cdot)$ above that is slightly more amenable to analysis.

LEMMA 2.2. ([5], PROPOSITION 2.12) *Let $q, k \in \mathbb{N}$ be given and $\mathcal{F}$ be a non-empty set of functions mapping $\mathbb{Z}_q^k \to \{0,1\}$. It holds that:*

$$\rho_{\min}(\mathcal{F}) = \min_{D\in\Delta(\mathcal{F})} \max_{D'\in\Delta(\mathbb{Z}_q)} \mathop{\mathbb{E}}_{\substack{f\sim D \\ \mathbf{a}\sim D'^k}} [f(\mathbf{a})].$$

Approximation resistance. Let $n \geq k \in \mathbb{N}$ and $\epsilon > 0$. Define the partial function $\mathsf{aprx}_{\mathcal{F},n,\epsilon}$ on instances Ψ of $\mathsf{Max\text{-}CSP}_n(\mathcal{F})$ to be 1 if $\mathsf{val}_\Psi = 1$ and 0 if $\mathsf{val}_\Psi \leq \rho_{\min}(\mathcal{F}) + \epsilon$. We are now ready to define the notion of approximation resistance.

DEFINITION 2.1. (APPROXIMATION RESISTANCE) *Let $q, k \in \mathbb{N}$ be given and $\mathcal{F}$ be a non-empty set of functions mapping $\mathbb{Z}_q^k \to \{0,1\}$. Let $s : \mathbb{N} \to \mathbb{R}$ be a monotone function. We say that $\mathsf{Max\text{-}CSP}(\mathcal{F})$ is approximation resistant to $o(s)$ space in the random order streaming model if for all $\epsilon > 0$ and $p > \frac{1}{2}$, there exists $\tau > 0$ such that for all $n \in \mathbb{N}$ and all randomized streaming algorithms $\mathcal{A}$ that compute $\mathsf{aprx}_{\mathcal{F},n,\epsilon}$ in the random-order streaming model with probability p, we have $\|\mathcal{A}\| \geq \tau \cdot s(n)$.*

One-wise independence. We say that a function $f : \mathbb{Z}_q^k \to \{0,1\}$ *supports one-wise independence* if there exists a distribution $D \in \Delta_{\mathsf{unif}}(\mathbb{Z}_q^k)$ that is supported on $f^{-1}(1)$. Similarly, we say that a family $\mathcal{F}$ of functions *(strongly) supports one-wise independence* if all functions in the family support one-wise independence. Finally, we say that a family $\mathcal{F}$ *weakly supports one-wise independence* if there exists a non-empty sub-family $\mathcal{F}' \subseteq \mathcal{F}$ that strongly supports one-wise independence and satisfies $\rho_{\min}(\mathcal{F}) = \rho_{\min}(\mathcal{F}')$.

2.3 One-way communication protocols.

Let $\mathcal{X}^A$ and $\mathcal{X}^B$ be two sets. We will treat these sets as the inputs sets for Alice and Bob respectively. We now define *one-way communication protocols* between Alice and Bob, where the inputs of the parties come from the sets $\mathcal{X}^A$ and $\mathcal{X}^B$ respectively, and Alice sends a single message to Bob. We start by defining deterministic protocols. Such a protocol is defined by a tuple:

$$\Pi = (L, \mathsf{msg}, \mathsf{out}),$$

where: (1) $L = \|\Pi\|$ is the length of the protocol Π, (2) $\mathsf{msg} : \mathcal{X}^A \to \{0,1\}^L$ is the function Alice uses to compute her message, and (3) $\mathsf{out} : \mathcal{X}^B \times \{0,1\}^L \to \{0,1\}$ is the function Bob uses to compute his output. We shall suppress the arguments on the right hand side when they are clear from context. We define a randomized protocol to be a distribution over deterministic protocols with the same input sets. The length of a randomized protocol is defined to be the maximum length of the deterministic protocols in its support.

Execution of a protocol. Let $\mathcal{X}^A$ and $\mathcal{X}^B$ be sets and Π be a deterministic protocol with inputs sets $\mathcal{X}^A$ and $\mathcal{X}^B$. For $x^A \in \mathcal{X}^A$ and $x^B \in \mathcal{X}^B$, we define the output $\Pi(x^A, x^B) \in \{0,1\}$ of the protocol Π on inputs x^A and x^B as:

$$\Pi(x^A, x^B) = \mathsf{out}\big(x^B, \mathsf{msg}\big(x^A\big)\big).$$

This is because, when the inputs are x^A and x^B, the string $\mathsf{msg}\big(x^A\big)$ is the message sent by Alice to Bob, and therefore, $\mathsf{out}\big(x^B, \mathsf{msg}\big(x^A\big)\big)$ is the output computed by Bob upon receiving this message.

One-way communication problems. We define a communication problem to be a pair of distributions[3] $(\mathcal{Y}, \mathcal{N})$ on the same product set $\mathcal{X}^A \times \mathcal{X}^B$. A protocol for the $(\mathcal{Y}, \mathcal{N})$-problem is a one way communication protocol where Alice's input comes from the set $\mathcal{X}^A$ and Bob's input comes from the set $\mathcal{X}^B$. Let $(\mathcal{Y}, \mathcal{N})$ be a communication problem and $\mathsf{\Pi}$ be a randomized communication protocol for the $(\mathcal{Y}, \mathcal{N})$-problem. For $\delta \geq 0$, we say that $\mathsf{\Pi}$ solves the $(\mathcal{Y}, \mathcal{N})$-problem with advantage δ if we have:

$$\left| \Pr_{\substack{(x^A,x^B)\sim\mathcal{Y} \\ \Pi\sim\mathsf{\Pi}}} \big(\Pi(x^A, x^B) = 1\big) - \Pr_{\substack{(x^A,x^B)\sim\mathcal{N} \\ \Pi\sim\mathsf{\Pi}}} \big(\Pi(x^A, x^B) = 1\big) \right| \geq \delta.$$

[3]Note that this matches our notation for distributional streaming problems. Nonetheless, the difference will be clear from context.

2.4 Analytical tools.

We now introduce some tools from probability and Fourier analysis which will be helpful in the proofs of the lower bounds.

2.4.1 Random variables.

Let $\|\mathcal{Y} - \mathcal{N}\|_{\text{tv}}$ denote the total variation distance between two distributions $\mathcal{Y}$ and $\mathcal{N}$.

Lemma 2.3. (Triangle inequality) *Let $\mathcal{Y}, \mathcal{N}, \mathcal{Z} \in \Delta(\Omega)$. Then*

$$\|\mathcal{Y} - \mathcal{N}\|_{\text{tv}} \geq \|\mathcal{Y} - \mathcal{Z}\|_{\text{tv}} - \|\mathcal{Z} - \mathcal{N}\|_{\text{tv}}.$$

Lemma 2.4. (Data processing inequality) *Let Y, N be random variables with sample space Ω, and let Z be a random variable with sample space Ω' which is independent of Y and N. If $g : \Omega \times \Omega' \to \Omega''$ is any function, then*

$$\|Y - N\|_{\text{tv}} \geq \|g(Y, Z) - g(N, Z)\|_{\text{tv}}.$$

We will use the following concentration inequality from [15].

Lemma 2.5. ([15, Lemma 2.5]) *Let $X = \sum_{i=1}^{n} X_i$, where X_i are Bernoulli $\{0, 1\}$-valued random variables satisfying, for every $k \in [n]$, $\mathbb{E}[X_k \mid X_1, \ldots, X_{k-1}] \leq p$ for some $p \in (0, 1)$. Let $\mu = np$. Then for all $\Delta > 0$,*

$$\Pr[X \geq \mu + \Delta] \leq \exp\left(-\frac{\Delta^2}{2(\mu + \Delta)}\right).$$

We also need the following concentration inequality that we prove using Lemma 2.5.

Lemma 2.6. *Let $X = \sum_{i=1}^{n} X_i$, where X_i are Bernoulli $\{0, 1\}$-valued random variables satisfying, for every $k \in [n]$, $\mathbb{E}[X_k \mid X_1, \ldots, X_{k-1}] \geq p$ for some $p \in (0, 1)$. Let $\mu = np$. Then for all $\Delta > 0$,*

$$\Pr[X \leq \mu - \Delta] \leq \exp\left(-\frac{\Delta^2}{2(n - (\mu - \Delta))}\right).$$

Proof. Follows immediately from Lemma 2.5 on the random variables $Y_i = 1 - X_i$, $q = 1 - p$, and $\nu = nq$ (since $X \leq \mu - \Delta$ is equivalent to $Y \geq \nu + \Delta$). $\square$

2.4.2 Fourier analysis over $\mathbb{Z}_q$.

Let $q \geq 2 \in \mathbb{N}$, and let $\omega \stackrel{\text{def}}{=} e^{2\pi i/q}$ denote a (fixed primitive) q-th root of unity. Here, we summarize relevant aspects of Fourier analysis over $\mathbb{Z}_q^n$; see e.g. [18, §8] for details.[4] Given a function $f : \mathbb{Z}_q^n \to \mathbb{C}$ and $\mathbf{s} \in \mathbb{Z}_q^n$, we define the *Fourier coefficient*

$$\widehat{f}(\mathbf{s}) \stackrel{\text{def}}{=} \sum_{\mathbf{x} \in \mathbb{Z}_q^n} \omega^{-\mathbf{s} \cdot \mathbf{x}} f(\mathbf{x})$$

where $\cdot$ denotes the inner product over $\mathbb{Z}_q$. For $p \in (0, \infty)$, we define f's *p-norm*

$$\|f\|_p \stackrel{\text{def}}{=} \left(\sum_{\mathbf{x} \in \mathbb{Z}_q^n} |f(\mathbf{x})|^p\right)^{1/p}.$$

We also define f's 0-norm

$$\|f\|_0 \stackrel{\text{def}}{=} \sum_{\mathbf{x} \in \mathbb{Z}_q^n} \mathbb{1}_{f(\mathbf{x}) \neq 0}$$

(a.k.a. the size of its support and the Hamming weight of its "truth table"). Also, for $\ell \in \{0\} \cup [n]$, we define the *level-ℓ Fourier (2-)weight* as

$$\mathsf{W}^{\ell}[f] \stackrel{\text{def}}{=} \sum_{\mathbf{s} \in \mathbb{Z}_q^n : \|\mathbf{s}\|_0 = \ell} |\widehat{f}(\mathbf{s})|^2.$$

These weights are closely connected to f's 2-norm:

[4][18] uses a different normalization for norms and inner products, essentially because it considers expectations instead of sums over inputs.

PROPOSITION 2.1. (PARSEVAL'S IDENTITY) *For every $q, n \in \mathbb{N}$ and $f : \mathbb{Z}_q^n \to \mathbb{C}$, we have*

$$\|f\|_2^2 = q^n \sum_{\ell=0}^{n} \mathsf{W}^\ell[f].$$

Moreover, let $\mathbb{D} \overset{\text{def}}{=} \{w \in \mathbb{C} : |w| \le 1\}$ denote the (closed) unit disk in the complex plane. The following lemma bounding the low-level Fourier weights for functions mapping into $\mathbb{D}$ is derived from hypercontractivity theorems in [3]:

LEMMA 2.7. ([3, LEMMA 2.11]) *There exists $\zeta > 0$ such that the following holds. Let $q \ge 2, n \in \mathbb{N}$ and consider any function $f : \mathbb{Z}_q^n \to \mathbb{D}$. If for $c \in \mathbb{N}$, $\|f\|_0 \ge q^{n-c}$, then for every $\ell \in \{1, \dots, 4c\}$, we have*

$$\frac{q^{2n}}{\|f\|_0^2} \mathsf{W}^\ell[f] \le \left(\frac{\zeta c}{\ell}\right)^\ell.$$

LEMMA 2.8. *Let $\mathcal{U} = \mathcal{U}(\mathbb{Z}_q^m)$. Then for all $\mathcal{Z} \in \Delta(\mathbb{Z}_q^m)$,*

$$\|\mathcal{Z} - \mathcal{U}\|_{\text{tv}}^2 \le q^{2m} \sum_{\ell=1}^{m} \mathsf{W}^\ell[\mathcal{Z}].$$

Finally, we need the following useful inequality, which bounds the total variation distance between a distribution $\mathcal{Z}$ and the uniform distribution by Fourier weights of the probability mass function of $\mathcal{Z}$:

Proof. We have

$$\|\mathcal{Z} - \mathcal{U}\|_{\text{tv}} = \frac{q^m}{2} \|\mathcal{Z} - \mathcal{U}\|_1.$$

Thus by Cauchy-Schwartz,

$$\|\mathcal{Z} - \mathcal{U}\|_{\text{tv}}^2 \le q^{2m} \|\mathcal{Z} - \mathcal{U}\|_2^2.$$

Finally, we apply Parseval and observe that $\widehat{\mathcal{Z}}(\mathbf{0}) = \widehat{\mathcal{U}}(\mathbf{0}) = 1$ while for all $\mathbf{s} \ne \mathbf{0}$, $\widehat{\mathcal{U}}(\mathbf{s}) = 0$ by symmetry. $\square$

2.5 Hypergraphs. Let $2 \le k, n \in \mathbb{N}$. A *$k$-hyperedge* on $[n]$ is a k-tuple $\mathbf{e} = (e_1, \dots, e_k) \in [n]^k$ of distinct indices, and a *k-hypergraph* (a.k.a. "k-uniform hypergraph") G on $[n]$ is a sequence $(\mathbf{e}(1), \dots, \mathbf{e}(m))$ of (not necessarily distinct) k-hyperedges. For $\alpha \in (0,1), n \in \mathbb{N}$, let $\mathcal{G}_{k,\alpha}(n)$ denote the uniform distribution over k-hypergraphs on $[n]$ with αn hyperedges.

Given a graph G with m edges $\mathbf{e}(1), \dots, \mathbf{e}(m)$, we associate each hyperedge $\mathbf{e}(i)$ with a partial permutation matrix $M_i \in \{0,1\}^{k \times n}$, such that for each $j \in [k]$, row j has a 1 only in position $e(j)_i$. We associate to G an *adjacency matrix* $M \in \{0,1\}^{km \times n}$ by stacking together $M_1, \dots, M_m$. Since they encode the same information, we will often treat adjacency matrices and k-hypergraphs as interchangeable (and speak of drawing a matrix M from $\mathcal{G}_{k,\alpha}(n)$).

For a k-hypergraph G on vertex-set $[n]$ with hyperedges $(\mathbf{e}(1), \dots, \mathbf{e}(m))$, we define the *vertex-hyperedge incidence graph* B_G, which is a bipartite graph defined as follows: The left vertex-set is $[n]$, the right vertex-set is $[m]$, and there is an edge between $i \in [n]$ and $j \in [m]$ iff $i \in \mathbf{e}(j)$.

2.6 Reservoir sampling in the streaming setting. Reservoir sampling is a term used to refer to a family of randomized streaming algorithms that are used to sample uniform k random elements from the stream without prior knowledge on the length of the stream. The simplest algorithm, known as *Algorithm R*, was created by Alan Waterman in 1975. The algorithm runs in $O(k)$ space and works as follows: it maintains a "reservoir" of size k. Initially, the first k elements in the stream are stored in the reservoir. For $i > k$, when the i-th element of the stream, denoted by a_i, arrives, the algorithm generates a random number j between 1 and i, and if $j \le k$, it replaces the j-th element in the reservoir with a_i. It is not hard to show that if m elements have arrived in the stream so far, then the probability of any one of them being in the reservoir is exactly k/m (see [24] for more details).

2.7 k-wise independent hash family. A *k-wise independent hash family* is a family of hash functions $\mathsf{H}(n,m) = \{h : [n] \to [m]\}$ that satisfies the following properties:

- For every $x \in [n]$ and $a \in [m]$, and $h \sim \mathsf{H}$ uniformly, $\Pr[h(x) = a] = \frac{1}{m}$, and
- For every distinct $x_1, \ldots, x_k \in [n]$, and $h \sim \mathsf{H}$ uniformly, $h(x_1), \ldots, h(x_k)$ are independent random variables.

For every fixed k, it is well-known that for $m = 2^\ell$ (or indeed any prime power) and any n, there are k-wise independent hash functions $\mathsf{H}(n,m)$ which can be sampled using only $O(\log n + \log m)$ randomness [11].

3 Algorithms for Max-DICUT

We review the definition of Max-DICUT as an optimization problem on unweighted directed graphs. Let $\mathcal{G} = (V, E)$ be an unweighted directed (multi)graph. $\mathcal{G}$'s Max-DICUT value, denoted $\mathsf{val}_\mathcal{G}$, is defined as the size of the largest directed cut in the graph. Formally,

$$\mathsf{val}_\mathcal{G} \overset{\text{def}}{=} \max_{L,R:V=L\sqcup R} |E_{L\to R}|,$$

where $E_{L\to R} = \{(i,j) \in E : i \in L \text{ and } j \in R\}$. In this section, we prove the following three theorems for a constant $\alpha_{\text{FJ}} \geq 0.483$:

Theorem 3.1. (Random-ordering algorithm) *Let $\epsilon > 0$ and $c > 0$ be constants. There exists an $O(\log n)$-space single-pass streaming algorithm* **ALG** *such that for every directed graph $\mathcal{G} = (V, E)$ with $|V| = n$ and $|E| \leq n^c$, the following holds: On input the edges of $\mathcal{G}$ in a uniformly random order,* **ALG** *outputs an $(\alpha_{\text{FJ}} - \epsilon)$-approximation to $\mathsf{val}_\mathcal{G}$ with probability at least $2/3$.*

Theorem 3.2. (Two-pass algorithm) *Let $\epsilon > 0$ and $c > 0$ be constants. There exists an $O(\log n)$-space* two-pass *streaming algorithm* **ALG** *such that for every directed graph $\mathcal{G} = (V, E)$ with $|V| = n$ and $|E| \leq n^c$, the following holds: On input the edges of $\mathcal{G}$ in adversarial order,* **ALG** *outputs an $(\alpha_{\text{FJ}} - \epsilon)$-approximation to $\mathsf{val}_\mathcal{G}$ with probability at least $2/3$.*

Theorem 3.3. (Bounded-degree algorithm) *Let $\epsilon > 0$, $c > 0$ be constants. There exists an $O(D^{3/2}\sqrt{n}\log^2 n)$-space single-pass streaming algorithm* **ALG** *such that for every directed graph $\mathcal{G} = (V, E)$ with $|V| = n$, $|E| \leq n^c$, and max-degree at most D, the following holds: On input the edges of $\mathcal{G}$ in adversarial order,* **ALG** *outputs an $(\alpha_{\text{FJ}} - \epsilon)$-approximation to $\mathsf{val}_\mathcal{G}$ with probability at least $2/3$.*

But first, we build some notation. The *bias* of a vertex $i \in V$ with respect to a directed graph $\mathcal{G} = (V, E)$, denoted $\mathsf{bias}_\mathcal{G}(i)$, is defined as $\mathsf{bias}_\mathcal{G}(i) = \frac{\mathsf{out\text{-}deg}_\mathcal{G}(i) - \mathsf{in\text{-}deg}_\mathcal{G}(i)}{\mathsf{out\text{-}deg}_\mathcal{G}(i) + \mathsf{in\text{-}deg}_\mathcal{G}(i)}$, where $\mathsf{out\text{-}deg}_\mathcal{G}(i), \mathsf{in\text{-}deg}_\mathcal{G}(i)$ respectively denote the out-degree and in-degree of i in $\mathcal{G}$. Now, we define the *density matrix* of a graph with respect to a partition of its vertices into bias intervals. Given any vector $\mathbf{t} = (t_1, \ldots, t_\ell) \in [-1,1]^\ell$ satisfying $-1 = t_1 < \cdots < t_\ell = 1$, consider the partition of the vertices V into blocks of vertices $V = V_1 \sqcup \cdots \sqcup V_\ell$ where for every $r \in [\ell - 1]$, $V_r = \{i : \mathsf{bias}_\mathcal{G}(i) \in [t_r, t_{r+1})\}$, and $V_\ell = \{i : \mathsf{bias}_\mathcal{G}(i) = 1\}$. Now the density matrix of $\mathcal{G}$ with respect to $\mathbf{t}$, denoted by $M_{\mathcal{G},\mathbf{t}}$, is an $\ell \times \ell$ matrix of natural numbers defined as $M_{\mathcal{G},\mathbf{t}}(i,j) = |E_{V_i \to V_j}|$, for every $i, j \in [\ell]$, i.e., the (i,j)-th entry of $M_{\mathcal{G},\mathbf{t}}$ counts the number of edges in $\mathcal{G}$ between vertices with biases in the intervals $[t_i, t_{i+1})$ (or 1 if $i = \ell$) and $[t_j, t_{j+1})$ (or 1 if $j = \ell$).

The following lemma was proved in [7] and it shows that there exists a vector $\mathbf{t}$ such that for every directed graph $\mathcal{G}$, the density matrix of $\mathcal{G}$ with the respect to the canonical partition $\mathcal{P}_{\mathcal{G},\mathbf{t}}$ can be used to get a good approximation to the Max-DICUT value of $\mathcal{G}$.

Lemma 3.1. ([7]) *There exists a constant $\alpha_{\text{FJ}} \in (0.483, 0.4899)$, $\ell_{\text{FJ}} \in \mathbb{N}$, a vector of bias thresholds $\mathbf{t}_{\text{FJ}} = (t_1, \ldots, t_\ell) \in [-1,1]^{\ell_{\text{FJ}}}$, and a vector of probabilities $\mathbf{p}_{\text{FJ}} = (p_1, \ldots, p_\ell) \in [0,1]^\ell$ such that for every directed graph $\mathcal{G}$,*

$$\alpha_{\text{FJ}} \cdot \mathsf{val}_\mathcal{G} \leq \sum_{i,j=1}^{\ell_{\text{FJ}}} p_i(1 - p_j) M_{\mathcal{G},\mathbf{t}}(i,j) \leq \mathsf{val}_\mathcal{G}.$$

We observe that algorithmically, the estimate for $\mathsf{val}_{\mathcal{G}}$ in this lemma corresponds to assigning each vertex in block V_i to L w.p. p_i and R w.p. $1-p_i$, independently of all other vertices.

As a corollary of Lemma 3.1, we show that in order to get an $(\alpha_{\mathrm{FJ}}-\epsilon)$-approximation for the Max-DICUT value of $\mathcal{G}$, it suffices to obtain an additive $\pm\epsilon' m$ approximation for every element of $M_{\mathcal{G},\mathbf{t}}$, for $\epsilon' = O(\epsilon)$.

Corollary 3.1. *Let $\alpha_{\mathrm{FJ}}, \ell_{\mathrm{FJ}}, \mathbf{t}_{\mathrm{FJ}}, \mathbf{p}_{\mathrm{FJ}}$ be as in Lemma 3.1. Let $\mathcal{G}$ be a directed graph and let m denote the number of edges in $\mathcal{G}$. Let $\epsilon \in (0, \alpha_{\mathrm{FJ}})$ and $\epsilon' = \frac{\epsilon}{8(\ell_{\mathrm{FJ}})^2}$. If there exists $N \in \mathbb{R}^{\ell_{\mathrm{FJ}}\times\ell_{\mathrm{FJ}}}$ such that for every $i,j \in [\ell_{\mathrm{FJ}}]$,*

$$M_{\mathcal{G},\mathbf{t}}(i,j) - \epsilon' m \le N(i,j) \le M_{\mathcal{G},\mathbf{t}}(i,j) + \epsilon' m\,,$$

then

$$(\alpha_{\mathrm{FJ}}-\epsilon)\mathsf{val}_{\mathcal{G}} \le \sum_{i,j\in[\ell_{\mathrm{FJ}}]} p_i(1-p_j)N(i,j) - \frac{\epsilon}{8}m \le \mathsf{val}_{\mathcal{G}}.$$

Proof. Omitted; see full version. ◻

In the following subsections, we describe how to estimate $M_{\mathcal{G},\mathbf{t}}$ in a number of different settings: $O(\log n)$-space single-pass streaming algorithm under random ordering of edges (Section 3.1), $O(\log n)$-space two-pass streaming algorithm under adversarial ordering (Section 3.2), and $O(D^{3/2}\sqrt{n}\log^2 n)$-space single-pass streaming algorithm for degree-D bounded graphs under adversarial ordering (Section 3.3). These algorithms share the same central principle: First, let $\mathcal{H} = (V, E')$ be a subgraph of $\mathcal{G} = (V, E)$ (i.e., $E' \subseteq E$). Given bias thresholds $-1 = t_1 < \cdots < t_\ell = 1$, let $M_{\mathcal{H}\subseteq\mathcal{G},\mathbf{t}} \in \mathbb{N}^{\ell\times\ell}$ denote the matrix with entries $M_{\mathcal{H}\subseteq\mathcal{G},\mathbf{t}}(i,j) = |E'_{V_i\to V_j}|$ where $V_1 \sqcup \cdots \sqcup V_\ell$ is the partition of V with respect to bias *in* $\mathcal{G}$. (Note that this is distinct from the matrices $M_{\mathcal{G},\mathbf{t}}$ and $M_{\mathcal{H},\mathbf{t}}$ because it counts *edges* in $\mathcal{H}$ but measures *bias* with respect to $\mathcal{G}$.) Now the strategy of all three algorithms is to somehow sample a "representative" subgraph $\mathcal{H}$ of $\mathcal{G}$, and then estimate $M_{\mathcal{G},\mathbf{t}}$ from $M_{\mathcal{H}\subseteq\mathcal{G},\mathbf{t}}$ simply by multiplying every entry by a scale factor $\frac{m(\mathcal{G})}{m(\mathcal{H})}$ (where $m(\mathcal{G}) = |E|$ and $m(\mathcal{H}) = |E'|$). There are two questions associated with this approach, which we answer differently in each setting:

1. *How do we sample a subgraph $\mathcal{H}$ with "representative" proportions of edges from $E_{V_i\to V_j}$ for each $i,j\in[\ell]$?* In Sections 3.1 and 3.2, $\mathcal{H}$ consists of random edges from $\mathcal{G}$, while in Section 3.3, $\mathcal{H}$ is the subgraph induced on random vertices from $\mathcal{G}$. In both cases, we show that (for a sufficiently large sample size), $\mathcal{H}$ is "sufficiently representative" with high probability using concentration bounds.

2. *How do we remember the "global bias" (i.e., the bias in $\mathcal{G}$) of vertices we sample in $\mathcal{H}$?* In the single-pass setting, we measure biases "online": Each time we see a new vertex appear as an endpoint in an edge, we decide whether to track its bias over the rest of the stream or not, and if we decide not to, it cannot have positive degree in $\mathcal{H}$. The two-pass setting obviates this limitation, since we can decide which vertices to track in the first pass and then actually track them in the second pass.

3.1 $O(\log n)$-space random-ordering (single-pass) algorithm. In this subsection, we prove Theorem 3.1 by showing that Algorithm 1 is an $(\alpha_{\mathrm{FJ}}-\epsilon)$-approximation streaming algorithm for computing Max-DICUT value when the edges of the input graph $\mathcal{G}$ are randomly ordered and uses space at most $O(\log n)$. Algorithm 1 uses Algorithm 2 as a subroutine to estimate $M_{\mathcal{G},\mathbf{t}}$ within a small additive error and then uses this estimate to compute an $(\alpha_{\mathrm{FJ}}-\epsilon)$-approximation to the Max-DICUT value of $\mathcal{G}$. We now describe and analyse Algorithm 1 and Algorithm 2.

Algorithm 1 Random-Order-Dicut$_\epsilon(n, \boldsymbol{\sigma})$:

```
Input: $n \in \mathbb{N}$ and a stream $\boldsymbol{\sigma} = (\mathbf{e}(1), \ldots, \mathbf{e}(m))$ representing randomly ordered edges of $\mathcal{G}$ on $n$ vertices.
1: Let $\ell_{\mathrm{FJ}}$, $\mathbf{t}_{\mathrm{FJ}}$, $\mathbf{p}_{\mathrm{FJ}}$ be from Lemma 3.1. Let $k$ and $m_0$ be fixed according to Lemma 3.2 corresponding to $\ell_{\mathrm{FJ}}, \mathbf{t}_{\mathrm{FJ}}$, and $\epsilon' = \frac{\epsilon}{8(\ell_{\mathrm{FJ}}^2)}$.
2: Store the first $m_0$ edges that arrive in the stream.
3: Let $N \leftarrow$ Random-Order-Estimate-$M_{\mathcal{G},\mathbf{t}}(n, \boldsymbol{\sigma}, \mathbf{t}_{\mathrm{FJ}}, k)$.
4: if $m < m_0$ then
5:     Compute $M_{\mathcal{G},\mathbf{t}_{\mathrm{FJ}}}$ directly from the stored edges and $N \leftarrow M_{\mathcal{G},\mathbf{t}_{\mathrm{FJ}}}$.
6: Output $\sum_{i,j=1}^{\ell_{\mathrm{FJ}}} p_i(1-p_j)N(i,j) - \frac{\epsilon}{8}m$.
```

We are now ready to describe our first algorithm for estimating $M_{\mathcal{G},\mathbf{t}}$.

```
Algorithm 2 Random-Order-Estimate-$M_{\mathcal{G},\mathbf{t}}(n, \boldsymbol{\sigma}, \mathbf{t}, k)$
Input: the number $n$ of vertices of a directed graph $\mathcal{G}$, a stream $\boldsymbol{\sigma} = (\mathbf{e}(1), \dots, \mathbf{e}(m))$ representing randomly
    ordered edges of $\mathcal{G}$, bias thresholds $-1 = t_1 < \cdots < t_\ell = 1$, and a parameter $k \in \mathbb{N}$.
1: Store the first $k$ edges $(\mathbf{e}(1), \dots, \mathbf{e}(k))$ of the stream. Let $\mathcal{H}$ denote the corresponding subgraph.
2: Over the remainder of the stream, track the following:
       • for every vertex $i$ with positive degree in $\mathcal{H}$, the degrees $\textsf{out-deg}_{\mathcal{G}}(i)$ and $\textsf{in-deg}_{\mathcal{G}}(i)$,
       • and the total number $m$ of edges in the stream.
3: After the stream ends, compute the following:
       • for every $i$ with positive degree in $\mathcal{H}$, $\textsf{bias}_{\mathcal{G}}(i)$,
       • and the matrix $M_{\mathcal{H}\subseteq\mathcal{G},\mathbf{t}}$.
Output: $N \in \mathbb{R}^{\ell\times\ell}$, where for every $i, j \in [\ell]$, $N(i,j) = \frac{m}{k} M_{\mathcal{H}\subseteq\mathcal{G},\mathbf{t}}(i,j)$.
```

Now the following lemma asserts the correctness of the estimate in Algorithm 2 for a sufficiently large choice of k:

LEMMA 3.2. *For every $\ell \in \mathbb{N}$ and threshold vector $\mathbf{t} \in [-1,1]^\ell$ and $\epsilon' > 0$, there exists $k, m_0 \in \mathbb{N}$ such that for every directed graph $\mathcal{G} = (V, E)$ with $m = |E| \geq m_0$ edges, with probability $\frac{2}{3}$, the matrix N output by Algorithm 2 on input $\mathcal{G}$ satisfies, for every $i, j \in [\ell]$, the inequalities*

$$M_{\mathcal{G},\mathbf{t}}(i,j) - \epsilon' m \leq N(i,j) \leq M_{\mathcal{G},\mathbf{t}}(i,j) + \epsilon' m.$$

Proof. Omitted; see full version. ☐

Finally, we prove Theorem 3.1.

Proof. [Proof of Theorem 3.1] Consider Algorithm 1. We fix $\ell_{\mathrm{FJ}}, \mathbf{t}_{\mathrm{FJ}}, \mathbf{p}_{\mathrm{FJ}}, \alpha_{\mathrm{FJ}}$ according to Lemma 3.1. For the choice of $k \in \mathbb{N}$ in Lemma 3.2 that corresponds to $\ell_{\mathrm{FJ}}, \mathbf{t}_{\mathrm{FJ}}$, and $\epsilon' = \frac{\epsilon}{8(\ell_{\mathrm{FJ}})^2}$, we run Algorithm 2 with the parameters $\mathbf{t}_{\mathrm{FJ}}, k$ on the input graph $\mathcal{G}$. For $m \geq m_0$, Lemma 3.2 implies that with probability $\frac{2}{3}$, the output N of Algorithm 2 entrywise approximates $M_{\mathcal{G},\mathbf{t}_{\mathrm{FJ}}}$ up to an additive $\pm\epsilon' m$. For $m < m_0$, Algorithm 1 computes $M_{\mathcal{G},\mathbf{t}_{\mathrm{FJ}}}$ exactly. Now Corollary 3.1 implies that the output of Algorithm 1 is an $(\alpha_{\mathrm{FJ}} - \epsilon)$-approximation to the Max-DICUT value of $\mathcal{G}$ as desired.

Finally, we show that Algorithm 1 can be implemented in $O(\log n)$ space. Since m_0 is a constant, it takes only $O(\log n)$ space to store the first m_0 edges. Algorithm 2 can be implemented in $O(\log n)$ space since it takes $O(\log n)$ space to store k edges and we use a simple counter in step 2 that uses $O(\log n)$ space for m that is bounded by $\mathrm{poly}(n)$. ☐

3.2 Two-pass $O(\log n)$-space adversarial-ordering algorithm.

In this subsection, we show how the random-ordering algorithm presented in Section 3.1 can be modified to work with adversarial input ordering given *two* passes over the input stream to prove Theorem 3.2.

Proof. [Proof of Theorem 3.2] Let $\mathbf{ALG}$ denote the $(\alpha_{\mathrm{FJ}} - \epsilon)$-approximation algorithm for Max-DICUT in the random ordering setting (Algorithm 1). Consider the following algorithm $\mathbf{ALG}'$: In the first pass $\mathbf{ALG}'$ uses reservoir sampling (see Section 2.6) to randomly sample k edges from the stream; this requires $O(k)$ space.[5] In the second pass, it runs the remainder of Algorithm 2 with parameters $\mathbf{t}_{\mathrm{FJ}}, k$ to obtain N and outputs $\sum_{i,j=1}^{\ell_{\mathrm{FJ}}} p_i(1-p_j)N(i,j) - \frac{\epsilon}{8}m$. The same proof of correctness, as well as the space analysis for Algorithm 1 works here as well. We conclude that with probability at least 2/3, $\mathbf{ALG}'$ outputs an $(\alpha_{\mathrm{FJ}} - \epsilon)$-approximation to the Max-DICUT value of $\mathcal{G}$. ☐

[5] Note that if the length of the stream is known *a priori*, there is a simpler sampling procedure. In the first pass, $\mathbf{ALG}'$ can sample every edge in the stream with probability $\frac{2k}{m}$. Let S denote the number of edges that were sampled. With high probability, $|S| \geq k$. Now, $\mathbf{ALG}'$ can choose a random subset of k edges from S.

3.3 $O(D^{3/2}\sqrt{n}\log^2 n)$-space adversarial-ordering algorithm for degree-D bounded graphs. In this subsection, we prove Theorem 3.3 by showing that Algorithm 3 is an $(\alpha_{\mathrm{FJ}}-\epsilon)$-approximation streaming algorithm for computing Max-DICUT value of degree-D bounded graphs and uses space at most $O(D^{3/2}\sqrt{n}\log^2 n)$. The basic idea is to sample a subset of the vertices of the input graph $\mathcal{G}$ and estimate $M_{\mathcal{G},\mathbf{t}}$ using the density matrix for the induced subgraph $M_{\mathcal{H}\subseteq\mathcal{G},\mathbf{t}}$. However, there are a few issues that ensue. Firstly, we need to deal with the case where most of $\mathcal{G}$'s vertices are isolated (i.e., they have degree zero); we manage this by only sampling vertices which have positive degree, by using a hash function on these vertices. This, in turn, requires estimating the number m of edges in the stream, which is not known *a priori*. For an estimate $\widehat{m}$ that satisfies $\widehat{m} \le m < 2\widehat{m}$, with high probability, Algorithm 4 estimates $M_{\mathcal{G},\mathbf{t}}$ correctly within a small additive error. Algorithm 3 runs Algorithm 4 for various estimates of m and using the correct output from Algorithm 4, it computes an $(\alpha_{\mathrm{FJ}}-\epsilon)$-approximation to the Max-DICUT value of $\mathcal{G}$. We now describe and analyse Algorithm 3 and Algorithm 4.

Algorithm 3 Bounded-Degree-Dicut$_D(n,\boldsymbol{\sigma})$:

```
Input: $n \in \mathbb{N}$ and a stream $\boldsymbol{\sigma} = (\mathbf{e}(1),\ldots,\mathbf{e}(m))$ representing randomly ordered edges of $\mathcal{G}$ on $n$ vertices.
1: Let $\ell_{\mathrm{FJ}}$, $\mathbf{t}_{\mathrm{FJ}}$, $\mathbf{p}_{\mathrm{FJ}}$ be from Lemma 3.1. Let $C_1$ and $k$ be fixed according to Lemma 3.3 corresponding to $\ell_{\mathrm{FJ}},\mathbf{t}_{\mathrm{FJ}}$, and $\epsilon' = \frac{\epsilon}{8(\ell_{\mathrm{FJ}}^2)}$.
2: Store the first $2C_1^2D$ edges that arrive in the stream.
3: for every integer $b$ from 0 to $\lfloor\log(nD/2)\rfloor$ do
4:     $\widehat{N}_b \leftarrow$ Bounded-Degree-Estimate-$M_{\mathcal{G},\mathbf{t}}(n,\boldsymbol{\sigma},\mathbf{t}_{\mathrm{FJ}},k,2^b)$
5:     if $\widehat{N}_b$ is not Fail then
6:         $N \leftarrow \widehat{N}_b$.
7: if $m < 2C_1^2D$ then
8:     Compute $M_{\mathcal{G},\mathbf{t}_{\mathrm{FJ}}}$ directly from the stored edges and $N \leftarrow M_{\mathcal{G},\mathbf{t}_{\mathrm{FJ}}}$.
9: Output $\sum_{i,j=1}^{\ell_{\mathrm{FJ}}} p_i(1-p_j)N(i,j) - \frac{\epsilon}{8}m$.
```

Algorithm 4 Bounded-Degree-Estimate-$M_{\mathcal{G},\mathbf{t}}(n,\boldsymbol{\sigma},\mathbf{t},k,\widehat{m})$

```
Input: the number $n$ of vertices of a directed graph $\mathcal{G}$, a stream $\boldsymbol{\sigma} = (\mathbf{e}(1),\ldots,\mathbf{e}(m))$ representing adversarially ordered edges, a vector $\mathbf{t} = (t_1,\ldots,t_\ell) \in [-1,1]^\ell$, and parameters $k,\widehat{m} \in \mathbb{N}$, where $\widehat{m}$ is a power of 2.
1: Sample a random hash function $\pi : [n] \to [\widehat{m}]$ from a 4-wise independent hash family $\mathsf{H}(n,\widehat{m})$ (see Section 2.7).
2: For the remainder of the stream, track the number of edges $m$ that arrive.
3: Define $s \leftarrow k\sqrt{\widehat{m}}$.
4: Initialize $\widehat{n} \leftarrow 0$.
5: Initialize $\mathcal{H} \leftarrow (V,\emptyset)$, where $V$ is the vertex set of $\mathcal{G}$.
6: for each edge $\mathbf{e}(t) = (u,v)$ in the stream do
7:     if $\pi(u) \le s$ then
8:         Track the bias of $u$. Increase $\widehat{n}$ by 1 if this is the first edge incident on $u$.
9:     if $\pi(v) \le s$ then
10:        Track the bias of $v$. Increase $\widehat{n}$ by 1 if this is the first edge incident on $v$.
11:    if $\pi(u) \le s$ and $\pi(v) \le s$ then
12:        Add $\mathbf{e}$ to $\mathcal{H}$.
13:    if $\widehat{n} > (5s \cdot \min\{n, 4\widehat{m}\})/\widehat{m}$ then
14:        Halt and output Fail.
15: if $m < \widehat{m}$ or $m \ge 2\widehat{m}$ then
16:     Halt and output Fail.
Output: $N \in \mathbb{R}^{\ell\times\ell}$, where for every $i,j \in [\ell]$, $N(i,j) = \frac{m}{\mu}M_{\mathcal{H}\subseteq\mathcal{G},\mathbf{t}}(i,j)$ where $\mu = ms^2/\widehat{m}^2$.
```

The correctness of Algorithm 4 conditioned on the estimate $\widehat{m}$ being approximately accurate is asserted in the following lemma:

LEMMA 3.3. *For every ℓ, threshold vector $\mathbf{t} \in [-1,1]^\ell$, and $\epsilon' > 0$, there exists $C_1 = C_1(\epsilon') > 0$ such that the following holds. Let $\mathcal{G}$ be a graph with n vertices, m edges, and max-degree $\leq D$ such that $m \geq 2C_1^2 D$, and let $\widehat{m} \in \mathbb{N}$ be such that $\widehat{m} \leq m < 2\widehat{m}$. Then with probability $\frac{2}{3}$ (over the choice of the permutation π), the matrix N output by Algorithm 4 on input $\mathcal{G}$ (with parameters $k = C_1\sqrt{D}, \widehat{m}$) satisfies, for every $i, j \in [\ell]$, the inequalities*

$$M_{\mathcal{G},\mathbf{t}}(i,j) - \epsilon' m \leq N(i,j) \leq M_{\mathcal{G},\mathbf{t}}(i,j) + \epsilon' m.$$

Proof. Omitted; see full version. ◻

Finally, we prove Theorem 3.3.

Proof. [Proof of Theorem 3.3] Consider Algorithm 3. We fix $\ell_{\mathrm{FJ}}, \mathbf{t}_{\mathrm{FJ}}, \mathbf{p}_{\mathrm{FJ}}, \alpha_{\mathrm{FJ}}$ according to Lemma 3.1 and k according to Lemma 3.3 corresponding to $\ell_{\mathrm{FJ}}, \mathbf{t}_{\mathrm{FJ}}$, and $\epsilon' = \frac{\epsilon}{8(\ell_{\mathrm{FJ}})^2}$. Since the max-degree of $\mathcal{G}$ is at most D, the number of edges m is at most $nD/2$. Observe that for every m, there is a unique $b \in [0, \lfloor \log(nD/2) \rfloor]$ such that $2^b \leq m < 2^{b+1}$. Namely, for $\widehat{b} = \lfloor \log m \rfloor$, we have $2^{\widehat{b}} \leq m < 2^{\widehat{b}+1}$. For $b = \widehat{b}$, the algorithm executes Algorithm 4 with $\widehat{m} = 2^{\widehat{b}}$. For $m \geq 2C_1^2 D$, Lemma 3.3 implies that with probability $\frac{2}{3}$, the output N of Algorithm 4 entrywise approximates $M_{\mathcal{G},\mathbf{t}_{\mathrm{FJ}}}$ up to an additive $\pm\epsilon' m$. For $m < 2C_1^2 D$, Algorithm 3 computes $M_{\mathcal{G},\mathbf{t}_{\mathrm{FJ}}}$ exactly. Now Corollary 3.1 implies that output of Algorithm 3 is an $(\alpha_{\mathrm{FJ}} - \epsilon)$-approximation to the Max-DICUT value of $\mathcal{G}$ as desired.

Finally, we show that Algorithm 3 can be implemented in $O(D^{3/2}\sqrt{n}\log^2 n)$ space. The first $2C_1^2 D$ edges in the stream can be stored in $O(D \log n)$ space. Since Algorithm 3 executes Algorithm 4 $O(\log n)$ times, it suffices to prove that Algorithm 4 can be implemented in $O(D^{3/2}\sqrt{n}\log n)$ space. Firstly, it takes $O(\log n)$ space to store π (see Section 2.7 for an example construction). Moreover, we can maintain the counter for the number of edges using $O(\log m)$ space. We have $\widehat{n} \leq (5s \cdot \min\{n, 4\widehat{m}\})/\widehat{m}$. Every tracked vertex contributes only $O(D \log n)$ space to store its degree and neighborhood. Therefore, Algorithm 4 requires at most $O\left(D^{3/2}\log n \cdot \min\{n, \widehat{m}\}/\sqrt{\widehat{m}}\right) \leq O(D^{3/2}\log n \cdot \sqrt{n})$ space. Hence, Algorithm 3 can be implemented in $O(D^{3/2}\log^2 n\sqrt{n})$ space. ◻

4 Lower bounds for Max-CSP in the random-ordering setting

4.1 The generalized Uniform Randomized Mask Detection (RMD) problem. We now define the Generalized-Uniform-RMD problem, the main focus of our lower bound. We shall define both a communication version and a streaming version. In either case, we need to define a pair of distributions. As the two pairs are rather closely related, we define them together.

DEFINITION 4.1. (Generalized-Uniform-RMD) *Let $q, k \in \mathbb{N}$ be given and $\mathcal{F}$ be a non-empty set of functions mapping $\mathbb{Z}_q^k \to \{0,1\}$. Let $\alpha > 0$ and $n \in \mathbb{N}$ be parameters and $\mathcal{D}_Y \in \Delta\big(\mathcal{F} \times \Delta_{\mathsf{unif}}\big(\mathbb{Z}_q^k\big)\big)$ be a distribution with finite support[6]. For all integers $0 \leq t \leq \alpha n$ and both versions, we now define a distribution $\mathcal{H}_{\mathcal{F},\mathcal{D}_Y,\alpha}(n,t)$ as follows:*

1. *For both versions:*
   (a) *Sample a vector $\mathbf{x}^*$ uniformly at random from $\mathbb{Z}_q^n$.*
   (b) *For all $i \in [\alpha n]$, sample a matrix $M_i \in \{0,1\}^{k \times n}$ uniformly and independently from the set of all partial permutation matrices[7].*
   (c) *For all $i \in [\alpha n]$, sample a pair (f_i, D_i) independently from $\mathcal{D}_Y$.*
   (d) *For all $i \in [\alpha n]$, sample a vector $\mathbf{b}(i) \in \mathbb{Z}_q^k$ independently from D_i if $i \leq t$ and uniformly and independently from the set $\mathbb{Z}_q^k$ if $i > t$.*
   (e) *For all $i \in [\alpha n]$, set $\mathbf{z}(i) = M_i\mathbf{x}^* - \mathbf{b}(i)$.*

[6] Observe that $\mathcal{D}_Y$ is a finite support distribution over pairs, the second element of which is itself a distribution.

[7] Recall that a partial permutation matrix is a matrix with $0, 1$ entries and exactly one 1 in each row and at most one 1 in every column.

2. Output as follows:

(a) For the communication version, define M (respectively, $\mathbf{z}$) to be the matrix (resp., vector) obtained by stacking all the M_i (resp., $\mathbf{z}(i)$) on top of each other. Also, define the vector $\mathbf{D}$ to be the vector consisting of the pairs $(f_i, D_i)_{i\in[\alpha n]}$. Output the pair $(\mathbf{x}^*, (M, \mathbf{z}, \mathbf{D}))$. (The first element of the pair $\mathbf{x}^*$ forms the input for Alice and the second element $(M, \mathbf{z}, \mathbf{D})$ forms the input for Bob.)

(b) For the streaming version, output the stream $(f_i, M_i, \mathbf{z}(i))_{i\in[\alpha n]}$. (Note that the length of the stream is αn and each symbol is a triple $(f_i, M_i, \mathbf{z}(i))$.)

For both versions, the problem $\mathsf{Generalized\text{-}Uniform\text{-}RMD}_{\mathcal{F},\mathcal{D}_Y,\alpha}(n)$ *is defined to be the pair of distributions $(\mathcal{H}_{\mathcal{F},\mathcal{D}_Y,\alpha}(n, \alpha n), \mathcal{H}_{\mathcal{F},\mathcal{D}_Y,\alpha}(n, 0))$. We shall often refer to $\mathcal{H}_{\mathcal{F},\mathcal{D}_Y,\alpha}(n, \alpha n)$ as the "yes" distribution and denote it by $\mathcal{Y}$ and $\mathcal{H}_{\mathcal{F},\mathcal{D}_Y,\alpha}(n, 0)$ as the "no" distribution and denote it by $\mathcal{N}$. The remaining distributions will only be needed for the streaming version and will be used as "hybrids".*

We note that in the communication version of Definition 4.1, the matrix M given to Bob is the adjacency matrix of a graph sampled from the distribution $\mathcal{G}_{k,\alpha}(n)$ (see Section 2.5).

We now define what it means to solve the Generalized-Uniform-RMD communication problem arising from the pair $(\mathcal{F}, \mathcal{D}_Y)$ with *non-trivial advantage*. The main emphasis of the definition is the advantage one can get as $\alpha \to 0$. It is natural to expect the advantage to shrink with α, and the definition below requires that the advantage only shrinks linearly with α.

DEFINITION 4.2. (SOLVING Generalized-Uniform-RMD WITH NON-TRIVIAL ADVANTAGE) *Let $q, k \in \mathbb{N}$ be given and $\mathcal{F}$ be a non-empty set of functions mapping $\mathbb{Z}_q^k \to \{0, 1\}$. Let $\mathcal{D}_Y \in \Delta\big(\mathcal{F} \times \Delta_{\mathsf{unif}}\big(\mathbb{Z}_q^k\big)\big)$ be a distribution with finite support and $s : \mathbb{N} \to \mathbb{R}$ be a function. We say that the pair $(\mathcal{F}, \mathcal{D}_Y)$ can be solved with non-trivial advantage using $o(s)$ communication if there exists $\delta > 0$ such that for all $\alpha, \tau > 0$, there exist infinitely many $n \in \mathbb{N}$ for which there exists a (randomized) protocol Π that solves the* $\mathsf{Generalized\text{-}Uniform\text{-}RMD}_{\mathcal{F},\mathcal{D}_Y,\alpha}(n)$*-problem with advantage $\delta \cdot \alpha$ and satisfies $\|\Pi\| \le \tau \cdot s(n)$.*

4.2 Proof of Theorem 1.4. In this section, we state two theorems that together imply Theorem 1.4. These theorems are then proved in the following sections. First, we have the following communication lower bound on the Generalized-Uniform-RMD problem.

THEOREM 4.1. *Let $q, k \in \mathbb{N}$ be given and $\mathcal{F}$ be a non-empty set of functions mapping $\mathbb{Z}_q^k \to \{0, 1\}$. Let $\mathcal{D}_Y \in \Delta\big(\mathcal{F} \times \Delta_{\mathsf{unif}}\big(\mathbb{Z}_q^k\big)\big)$ be a distribution with finite support. Then, $(\mathcal{F}, \mathcal{D}_Y)$ cannot be solved with non-trivial advantage using $o(\sqrt{n})$ communication.*

We also show why the above communication lower bound implies that certain CSPs are approximation resistant.

THEOREM 4.2. *Let $q, k \in \mathbb{N}$ be given and $\mathcal{F}$ be a non-empty set of functions mapping $\mathbb{Z}_q^k \to \{0, 1\}$ and weakly supporting one-wise independence. There exists a distribution $\mathcal{D}_Y \in \Delta\big(\mathcal{F} \times \Delta_{\mathsf{unif}}\big(\mathbb{Z}_q^k\big)\big)$ with a finite support such that if $(\mathcal{F}, \mathcal{D}_Y)$ cannot be solved with non-trivial advantage using $o(\sqrt{n})$ communication, then Max-CSP$(\mathcal{F})$ is approximation resistant to $o(\sqrt{n})$ space in the random order streaming model.*

An (outline of) the proof of Theorem 4.1 is in the following section. We omit the proof of Theorem 4.2; see the full version.

5 Proof of Theorem 4.1

In this section we prove that the Generalized-Uniform-RMD communication problem arising from $(\mathcal{F}, \mathcal{D}_Y)$ cannot be solved with non-trivial advantage using $o(\sqrt{n})$ communication. The central element in the proof is to look at the distribution of Bob's input $\mathbf{z}$ conditioned on Alice's message and the matrix M, and to argue that the distributions are close in the **YES** and **NO** cases. By definition, the distribution in the **NO** case is uniform over $\mathbb{Z}_q^{km}$ and so what needs to be really shown is that in the **YES** case also this distribution is close to uniform.

Note that Alice's message specifies a set $A \subseteq \mathbb{Z}_q^n$ such that $\mathbf{x}^* \sim \mathsf{Unif}(A)$. Lemma 5.1 roughly relates the distance of the conditional distribution of $\mathbf{z}$ (in the **YES** case) to the Fourier spectrum of the indicator of the set

A and to a somewhat complex combinatorial parameter associated with the random hypergraph described by M (see Eq. (5.4)). More precisely Lemma 5.1 bounds this distance provided M is "cycle-free" according to a natural notion of cycle-freeness for hypergraphs that we introduce below. We then state two lemmas upper-bounding the expectation of the combinatorial parameter (Lemma 5.2) and the probability of a cycle (Lemma 5.3), whose proofs are deferred to Section 6. We use these bounds to complete the proof of Theorem 4.1.

The proof outline described above follows the same structure as that of [14] with two significant differences. First, the definition of cycle-freeness is different in our work and this difference has a quantitative effect in that the probability of being cycle-free increases to $\Theta(\alpha^2)$ in our setting compared to $\Theta(\alpha^3)$ in their work. This difference is significant in the context of "non-trivial advantage". Directly following the proof in [14] would have led to a $\Theta(\alpha)$ advantage and we make some changes in the proof of Theorem 4.1 to show that despite the higher probability of cycle-freeness, protocols with non-trivial advantage require $\Omega(\sqrt{n})$ communication. The second difference is in the combinatorial quantity of interest which sees differences due to the higher values of k and q, and the richness of the distributions $\mathcal{D}_Y$ that we need to handle. The analysis of the combinatorial quantity is also more complex and we describe the differences in the next section.

5.1 Indististinguishability via Fourier analysis. Conditioned on a set $A \subset \mathbb{Z}_q^n$ of $\mathbf{x}^*$'s corresponding to an Alice message, a k-hypergraph $M \in \{0,1\}^{k\alpha n \times n}$, and a vector $\mathbf{D} = ((f_1, D_1), \ldots, (f_m, D_m)) \in (\mathcal{F} \times \Delta_{\mathsf{unif}}(\mathbb{Z}_q^k))^m$, let $\mathcal{Z}_{A,M,\mathbf{D}} \in \Delta(\mathbb{Z}_q^{k\alpha n})$ denote the conditional distribution of Bob's input $\mathbf{z}$ in the **YES** case, i.e.,

$$\mathcal{Z}_{A,M,\mathbf{D}}(\mathbf{z}) = \Pr_{\mathbf{x}^* \sim \mathcal{U}(A), \mathbf{b} \sim D_1 \times \cdots \times D_m}[\mathbf{z} = M\mathbf{x}^* - \mathbf{b}].$$

For a k-hypergraph G, let $\mathsf{cf}(G)$ denote the event that G is *cycle-free* in the sense that its point-hyperplane incidence graph B_G contains no cycles. Let $S_{\neq 1} \stackrel{\text{def}}{=} \{\mathbf{s} \in (\mathbb{Z}_q^k)^{\alpha n} : \forall i \in [\alpha n], \|\mathbf{s}(i)\|_0 \neq 1\}$. Then for $\ell \in [n]$, we define the quantity

$$h_{k,\alpha}(\ell, n) \stackrel{\text{def}}{=} \max_{\mathbf{v} \in \mathbb{Z}_q^n, \|\mathbf{v}\|_0 = \ell} \left(\mathop{\mathbb{E}}_{M \sim \mathcal{G}_{k,\alpha}(n)} \left[\mathbb{1}_{\mathsf{cf}(M)} \cdot \left| \{ \mathbf{s} \in S_{\neq 1} : M^\top \mathbf{s} = \mathbf{v} \} \right| \right] \right). \tag{5.4}$$

Lemma 5.1. (Fourier-analytic reduction) *Fix $n \in \mathbb{N}$, $\alpha \in (0, 1/100k)$, and a vector*

$$\mathbf{D} = ((f_1, D_1), \ldots, (f_m, D_M)) \in (\mathcal{F} \times \Delta_{\mathsf{unif}}(\mathbb{Z}_q^k))^m.$$

Then

$$\mathop{\mathbb{E}}_{M \sim \mathcal{G}_{k,\alpha}(n)}[\mathbb{1}_{\mathsf{cf}(M)} \cdot \|\mathcal{Z}_{A,M,\mathbf{D}} - \mathcal{U}(\mathbb{Z}_q^{k\alpha n})\|_{\mathrm{tv}}^2] \leq \frac{q^{2n}}{|A|^2} \sum_{\ell=1}^{k\alpha n} h_{k,\alpha}(\ell, n) \mathsf{W}^\ell[\mathbb{1}_A]$$

where $h_{k,\alpha}(\ell, n)$ is defined as in Eq. (5.4).

Proof. Omitted; see full version. ☐

5.2 Properties of random hypergraphs. Now we state two lemmas about the distribution $\mathcal{G}_{k,\alpha}(n)$ which we will prove in Section 6 below:

Lemma 5.2. *For all $2 \leq q, k \in \mathbb{N}$, there exists $c_h < \infty$ and $\alpha_0 > 0$ such that for all $\alpha \in (0, \alpha_0)$,*

$$h_{k,\alpha}(\ell, n) \leq \left(\frac{c_h \ell}{n} \right)^{\ell/2}.$$

Lemma 5.3. *For every $k \geq 2$, there exists $c_{\mathsf{cf}} < \infty$ and $\alpha_0 \in (0,1)$ such that for all $n \geq k$ and $\alpha \in (0, \alpha_0)$,*

$$\Pr_{G \sim \mathcal{G}_{k,\alpha}(n)}[\neg \mathsf{cf}(G)] \leq c\alpha^2.$$

5.3 Putting the ingredients together Modulo these lemmas, we can now prove Theorem 4.1:

Proof. [Proof of Theorem 4.1] Omitted; see full version. □

REMARK 5.1. *Even a weaker bound in Lemma 5.2 of $(c_h\ell^2/n)^{\ell/2}$ would have sufficed for us to prove Theorem 4.1. On the other hand, we also note that the lemma can be strengthened even further and our proof could actually yield any $c_h > 0$ by choosing α_0 small enough. We omit this optimization in Section 6.*

6 Hypergraph analyses

In this section we analyze the quantities of interest in random hypergraphs. In Section 6.1 we analyze the probability that a random hypergraph has a cycle — this analysis is straightforward (and included mainly for completeness). In Section 6.2 we analyze the quantity $h_{k,\alpha}(\ell, n)$ which takes more work. An overview is included in the beginning of Section 6.2.

6.1 Proving Lemma 5.3: Upper-bounding the probability of cycles.

PROPOSITION 6.1. *Let $2 \le k \le n \in$ and $\alpha \in (0,1)$. For every $u, v \in [n]$ and $j \in [\alpha n]$,*

$$\Pr_{G\sim\mathcal{G}_{k,\alpha}(n)}[u, v \in \mathbf{e}(j)] = \frac{\binom{k}{2}}{\binom{n}{2}},$$

where G has hyperedges $\mathbf{e}(1), \ldots, \mathbf{e}(\alpha n)$.

Proof. Omitted; see full version. □

Proof. [Proof of Lemma 5.3] Omitted; see full version. □

6.2 Proving Lemma 5.2: Upper-bounding $h_{k,\alpha}(\ell, n)$. In what follows we fix a vector $\mathbf{v} \in \mathbb{Z}_q^n$ with support $U \subseteq [n]$ and upper bound the quantity $\mathbb{E}_{M\sim\mathcal{G}_{k,\alpha}(n)}\left[\mathbb{1}_{\mathsf{cf}(M)} \cdot \left|\left\{\mathbf{s} \in S_{\neq 1} : M^\top \mathbf{s} = \mathbf{v}\right\}\right|\right]$. For $M \in \mathsf{supp}(\mathcal{G}_{k,\alpha}(n))$ let $X(M) = \mathbb{1}_{\mathsf{cf}(M)} \cdot \left|\left\{\mathbf{s} \in S_{\neq 1} : M^\top \mathbf{s} = \mathbf{v}\right\}\right|$ so that the quantity of interest is $\mathbb{E}_{M\sim\mathcal{G}_{k,\alpha}(n)}[X(M)]$. To analyze this expectation, first in Proposition 6.2 we give combinatorial conditions on M under which $X(M) = 0$. Further we give a simpler upper bound on $X(M)$ in terms of the connected component structure of M when $X(M)$ is potentially non-zero. Roughly, this proposition bounds $X(M)$ by some function of the size of the connected components of M that are incident to the set U. Lemmas 6.1 to 6.4 then analyze the probability that the components have large size. The resulting bounds are put together to prove Lemma 5.2 at the end of this section.

We now turn to proving Lemma 5.2. Throughout this section, the vertex-hyperedge incidence graph $B = B_M$ corresponding to a k-hypergraph M (from Section 2.5) will be the central object of interest. While we refer to vertices of M as “vertices”, the vertices of B are referred to as either “left vertices” (corresponding to vertices of M) or “right vertices” (corresponding to hyperedges of M). Similarly we use “hyperedges” to refer to edges of M and “edges” to refer to edges of B. In this interpretation, the i-th hyperedge $\mathbf{e}(i)$ of M is the neighborhood of the i-th right vertex of B. Thus, sampling a random hypergraph $M \sim \mathcal{G}_{k,\alpha}(n)$ is equivalent to sampling B by setting each right vertex’s neighborhood to be a uniform and independent subset of k left vertices. The vector $\mathbf{v}$ can be viewed as a $\mathbb{Z}_q$-labelling of the left vertices of B, while the vector $\mathbf{s}$ is a $\mathbb{Z}_q$-labelling of B’s edges. The condition $\mathbf{s} \in S_{\neq 1}$ means that no right-vertex of B has degree exactly one, and the condition $M^\top\mathbf{s} = \mathbf{v}$ implies that the left vertices of B are each labelled by the sum (modulo q) of the labels of incident edges of B. The condition that U is the support of $\mathbf{v}$ implies that U is exactly the set of left vertices with non-zero labels.

Now consider the connected component decomposition of B, which induces a partition $V_1, \ldots, V_{t'}$ of B’s left vertices $[n]$. Since $U \subseteq [n]$ is a subset of B’s left vertices, B’s partition of $[n]$ further induces a partition of U into subsets $U_1, \ldots, U_t$ for $t \le t'$. (This partition is given by intersecting each V_i with U and throwing it away if the intersection is empty. Thus, each component U_i of U is contained in a single connected component of B.)

Note that this partition (given U and B) is essentially unique up to renaming of the parts. We formalize this as follows. We say that $U_1, \ldots, U_t$ is a *canonical partition* of U if each U_i contains the least numbered vertex of U that is not contained in $\cup_{j<i}U_j$. (Note that every partition $U_1, \ldots, U_t$ can be converted into a canonical one by renumbering the parts. Furthermore given U and B this partition is unique.) We let $\mathsf{cc\text{-}part}(B, U)$, for

"connected component partition", denote this canonical partition of U induced by B. We say that B *partitions* U *into* t *connected components* if $\mathsf{cc\text{-}part}(B,U)$ has t parts.

Given a subset $U' \subseteq U$ contained in a unique connected component of B, we say it has *L-type* ℓ if $\ell = |U'|$, and *R-type* r if the connected component of B containing U' has exactly r right vertices. These numbers satisfy the inequality $\ell \leq kr$ since every left vertex must touch at least one right vertex. More generally, if B partitions U into connected components $\mathsf{cc\text{-}part}(B,U) = (U_1,\dots,U_t)$, we say $\mathsf{cc\text{-}part}(B,U)$ is of *L-type* $(\ell_1,\dots,\ell_t)$ if $\ell_i = |U_i|$ for every $i \in [t]$. We say $\mathsf{cc\text{-}part}(B,U)$ is *valid* if $\ell_i \geq 2$ for every i. We say $\mathsf{cc\text{-}part}(B,U)$ is of *R-type* $(r_1,\dots,r_t)$ if in B, the connected component containing U_i has exactly r_i right vertices for every $i \in [t]$, and $\mathsf{cc\text{-}part}(B,U)$ is of *R-total-type* r if $\sum_{i\in[t]} r_i = r$.

The following proposition fixes a graph M and give conditions on when the quantity $\mathbb{1}_{\mathsf{cf}(M)} \cdot \left|\left\{\mathbf{s} \in S_{\neq 1} : M^\top \mathbf{s} = \mathbf{v}\right\}\right|$ is non-zero; moreover, when it is non-zero, it gives an upper bound.

PROPOSITION 6.2. *For a fixed* $\mathbf{v} \in \mathbb{Z}_q^n$ *with support* $U \subseteq [n]$ *and a fixed k-hypergraph* M*, the quantity* $\mathbb{1}_{\mathsf{cf}(M)} \cdot \left|\left\{\mathbf{s} \in S_{\neq 1} : M^\top \mathbf{s} = \mathbf{v}\right\}\right|$ *is non-zero only if* M *is cycle-free, and* $\mathsf{cc\text{-}part}(B,U)$ *is a valid partition. Furthermore, for every* $r \in \mathbb{N}$*, if* M *is cycle-free and* $\mathsf{cc\text{-}part}(B,U)$ *is a valid partition of R-total-type* r*, we have* $\mathbb{1}_{\mathsf{cf}(M)} \cdot \left|\left\{\mathbf{s} \in S_{\neq 1} : M^\top \mathbf{s} = \mathbf{v}\right\}\right| \leq q^{kr}$.

Proof. Omitted; see full version. □

Now, we state several lemmas regarding the probability of a random graph M partitioning sets in various ways, building towards Lemma 6.4 below which bounds the probability that $\mathsf{cc\text{-}part}(B_M,U)$ is a valid partition of R-total-type r.

LEMMA 6.1. *Let* $n/2+1 \leq n' \leq n$ *and* $\alpha \in (0,1)$*. Let* $M \sim \mathcal{G}_{k,\alpha'}(n')$ *for* $\alpha' = \alpha n/n'$ *and* $B = B_M$*. Then for every* $u \in [n']$,

$$\Pr_M[B \text{ places } u \text{ in a component of R-type at least } r_1] \leq (2ek^2\alpha)^{r_1}.$$

Proof. Omitted; see full version. □

LEMMA 6.2. *Let* $\alpha \leq 1/(2e^3k^2)$, $n/2+1 \leq n' \leq n$ *and* $r_1 \in \mathbb{N}$*. Fix a set* $U_1 \subseteq [n']$ *with* $|U_1| = \ell_1$*. Let* $M \sim \mathcal{G}_{k,\alpha'}(n')$ *for* $\alpha' = \alpha n/n'$ *and let* $B = B_M$*. Then*

$$\Pr_M[B \text{ partitions } U_1 \text{ into a single connected component of R-type } r_1] \leq (2ek^2\alpha)^{r_1/2}(2k(\ell_1-1)/n)^{\ell_1-1}.$$

Proof. Omitted; see full version. □

LEMMA 6.3. *Let* $\alpha \leq 1(2e^3k^2)$ *and* $n \geq 4$*. Fix* $r \in \mathbb{N}$*, a set* $U \subseteq [n]$ *and a canonical partition* $U_1,\dots,U_t$ *of* U*. Let* $\ell = |U|$ *and* $\ell_i = |U_i|$*. Let* $M \sim \mathcal{G}_{k,\alpha}(n)$*. We have*

$$\Pr[\mathsf{cc\text{-}part}(B,U) = (U_1,\dots,U_t) \text{ with R-total-type } r] \leq (32ek^2\alpha)^{r/2}(2k/n)^{\ell-t}\prod_{i=1}^{t}(\ell_i-1)^{\ell_i-1}.$$

Proof. Omitted; see full version. □

LEMMA 6.4. *Let* $\alpha \leq 1(2e^3k^2)$, $n \geq 4$ *and* $\ell \leq n/(4ek)$*. Fix* $r \in \mathbb{N}$*, a set* $U \subseteq [n]$ *with* $|U| = \ell$*. Let* $M \sim \mathcal{G}_{k,\alpha}(n)$ *and* $B = B_M$*. Then*

$$\Pr_M[\mathsf{cc\text{-}part}(B,U) \text{ is valid and has R-total-type } r] \leq 2(32ek^2\alpha)^{r/2}(32ek\ell/n)^{\ell/2}.$$

Proof. Omitted; see full version. □

Given these lemmas, we are now prepared to prove Lemma 5.2.

Proof. [Proof of Lemma 5.2] Omitted; see full version. □

References

[1] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming Algorithms via Precision Sampling. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 363–372. doi:10.1109/FOCS.2011.82.

[2] Joanna Boyland, Michael Hwang, Tarun Prasad, Noah Singer, and Santhoshini Velusamy. On sketching approximations for symmetric Boolean CSPs. In Amit Chakrabarti and Chaitanya Swamy, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 245 of *LIPIcs*, pages 38:1–38:23. Schloss Dagstuhl — Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX/RANDOM.2022.38.

[3] Chi-Ning Chou, Alexander Golovnev, Madhu Sudan, Ameya Velingker, and Santhoshini Velusamy. Linear Space Streaming Lower Bounds for Approximating CSPs. In *Proceedings of the 54th Annual ACM Symposium on Theory of Computing*.

[4] Chi-Ning Chou, Alexander Golovnev, Madhu Sudan, and Santhoshini Velusamy. Approximability of all Boolean CSPs with linear sketches. URL: https://arxiv.org/abs/2102.12351, arXiv:2102.12351v7.

[5] Chi-Ning Chou, Alexander Golovnev, Madhu Sudan, and Santhoshini Velusamy. Approximability of all finite CSPs with linear sketches. In *Proceedings of the 62nd Annual IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society. doi:10.1109/FOCS52979.2021.00117.

[6] Chi-Ning Chou, Alexander Golovnev, and Santhoshini Velusamy. Optimal Streaming Approximations for all Boolean Max-2CSPs and Max-$k$SAT. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science*, pages 330–341. IEEE Computer Society. doi:10.1109/FOCS46700.2020.00039.

[7] Uriel Feige and Shlomo Jozeph. Oblivious Algorithms for the Maximum Directed Cut Problem. 71(2):409–428. doi:10.1007/s00453-013-9806-z.

[8] Venkatesan Guruswami and Runzhou Tao. Streaming Hardness of Unique Games. In Dimitris Achlioptas and László A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 145 of *LIPIcs*, pages 5:1–5:12. Schloss Dagstuhl — Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.5.

[9] Venkatesan Guruswami, Ameya Velingker, and Santhoshini Velusamy. Streaming Complexity of Approximating Max 2CSP and Max Acyclic Subgraph. In Klaus Jansen, José D. P. Rolim, David Williamson, and Santosh S. Vempala, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 81 of *LIPIcs*, pages 8:1–8:19. Schloss Dagstuhl — Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.8.

[10] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. 53(3):307–323. Conference version in FOCS 2000. doi:10.1145/1147954.1147955.

[11] A. Joffe. On a Set of Almost Deterministic \$k\$-Independent Random Variables. 2(1):161–162. doi:10.1214/aop/1176996762.

[12] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. On the Exact Space Complexity of Sketching and Streaming Small Norms. In *Proceedings of the 2010 Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1161–1178. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611973075.93.

[13] Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 734–751. Society for Industrial and Applied Mathematics. doi:10.5555/2634074.2634129.

[14] Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Streaming lower bounds for approximating MAX-CUT. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1263–1282. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611973730.84.

[15] Michael Kapralov and Dmitry Krachun. An optimal space lower bound for approximating MAX-CUT. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 277–288. Association for Computing Machinery. doi:10.1145/3313276.3316364.

[16] Michael Kapralov, Slobodan Mitrović, Ashkan Norouzi-Fard, and Jakab Tardos. Space efficient approximation to maximum matching size from uniform edge samples. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1753–1772. Society for Industrial and Applied Mathematics. doi:10.5555/3381089.3381196.

[17] Morteza Monemizadeh, S. Muthukrishnan, Pan Peng, and Christian Sohler. Testable Bounded Degree Graph Properties Are Random Order Streamable. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming*, volume 80 of *LIPIcs*, pages 131:1–131:14. Schloss Dagstuhl — Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2017.131.

[18] Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 1st edition edition.

[19] Pan Peng and Christian Sohler. Estimating Graph Parameters from Random Order Streams. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611975031.157.

[20] Raghuvansh R. Saxena, Noah Singer, Madhu Sudan, and Santhoshini Velusamy. Streaming beyond sketching for Maximum Directed Cut. In submission. arXiv:2211.03916.

[21] Noah Singer. On streaming approximation algorithms for constraint satisfaction problems. URL: https://nrs.harvard.edu/URN-3:HUL.INSTREPOS:37371750.

[22] Noah Singer, Madhu Sudan, and Santhoshini Velusamy. Streaming approximation resistance of every ordering CSP. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 207 of *LIPIcs*, pages 17:1–17:19. Schloss Dagstuhl — Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX/RANDOM.2021.17.

[23] Madhu Sudan. Streaming and Sketching Complexity of CSPs: A survey (Invited Talk). In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming*, volume 229 of *LIPIcs*, pages 5:1–5:20. Schloss Dagstuhl — Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2022.5.

[24] Jeffrey S. Vitter. Random sampling with a reservoir. 11(1):37–57. doi:10.1145/3147.3165.