2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS) | 979-8-3503-1894-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/FOCS57990.2023.00055

Improved Streaming Algorithms for Maximum Directed Cut via Smoothed Snapshots

Raghuvansh R. Saxena
Tata Institute of Fundamental Research
Mumbai, Maharashtra, India
raghuvansh.saxena@gmail.com

Noah G. Singer
Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA, USA
0000-0002-0076-521X
ngsinger@cs.cmu.edu

Madhu Sudan
School of Engineering and Applied Sciences
Harvard University
Cambridge, MA, USA
0000-0003-3718-6489
madhu@cs.harvard.edu

Santhoshini Velusamy
Toyota Technological Institute at Chicago
Chicago, IL, USA
0000-0002-0294-5425
santhoshini@ttic.edu

***Abstract*—We give an $\widetilde{O}(\sqrt{n})$-space single-pass 0.483-approximation streaming algorithm for estimating the maximum directed cut size (Max-DICUT) in a directed graph on n vertices. This improves over an $O(\log n)$-space $4/9 < 0.45$ approximation algorithm due to Chou, Golovnev, and Velusamy (FOCS 2020), which was known to be optimal for $o(\sqrt{n})$-space algorithms. Max-DICUT is a special case of a *constraint satisfaction problem* (CSP). In this broader context, we give the *first* CSP for which algorithms with $\widetilde{O}(\sqrt{n})$ space can provably outperform $o(\sqrt{n})$-space algorithms.**

The key technical contribution of our work is development of the notions of a first-order snapshot of a (directed) graph and of estimates of such snapshots. These snapshots can be used to simulate certain (non-streaming) Max-DICUT algorithms, including the "oblivious" algorithms introduced by Feige and Jozeph (Algorithmica, 2015), who showed that one such algorithm achieves a 0.483-approximation.

Previous work of the authors (SODA 2023) studied the restricted case of bounded-degree graphs, and observed that in this setting, it is straightforward to estimate the snapshot with ℓ_1 errors and this suffices to simulate oblivious algorithms. But for unbounded-degree graphs, even defining an achievable and sufficient notion of estimation is subtle. We describe a new notion of snapshot estimation and prove its sufficiency using careful smoothing techniques, and then develop an algorithm which sketches such an estimate via a delicate process of intertwined vertex- and edge-subsampling.

Prior to our work, the only streaming algorithms for *any* CSP on general instances were based on generalizations of the $O(\log n)$-space algorithm for Max-DICUT, and can roughly be characterized as based on "zeroth" order snapshots. Our work thus opens the possibility of a new class of algorithms for approximating CSPs by demonstrating that more sophisticated snapshots can outperform cruder ones in the case of Max-DICUT.

***Index Terms*—constraint satisfaction problem, approximation algorithms, sublinear algorithms, streaming algorithms**

N.G.S. was supported by an NSF Graduate Research Fellowship (Award DGE2140739). M.S. was supported in part by a Simons Investigator Award and NSF Award CCF 2152413. S.V. was supported in part by a Google Ph.D. Fellowship, a Simons Investigator Award to Madhu Sudan, and NSF Award CCF 2152413.

I. Introduction

We consider approximating the maximum directed cut value of a directed graph by a streaming algorithm presented with a stream of edges in an arbitrary (worst-case) order. Our main result is a single-pass algorithm using $\widetilde{O}(\sqrt{n})$-space that gives a $.483$ approximation algorithm. Along the way we develop the notions of snapshots of graphs and estimates of such snapshots, which introduce new tools for approximating graph theoretic quantities and more generally for approximating Constraint Satisfaction Problems (CSPs). In what follows we explain the background of the directed cut problem, the significance of the result, and the techniques used to achieve this result.

A. Background

We begin by defining the maximum directed cut (Max-DICUT) problem in a directed graph $\mathcal{G}$. (These definitions will all be informal; see Section II for formal definitions.) Given a graph $\mathcal{G}$ on n vertices, labeled $1, \dots, n$, *cut* of $\mathcal{G}$ is a binary string $\mathbf{x} \in \{0,1\}^n$, assigning a bit to every vertex in $\mathcal{G}$. We say $\mathbf{x}$ *cuts* a directed edge (u, v) if $x_u = 1$ and $x_v = 0$. (Note the asymmetry between u and v.) The *value* $\mathsf{val}_{\mathcal{G}}(\mathbf{x})$ of a cut $\mathbf{x}$ is the total fraction of edges it cuts, and the *value* $\mathsf{val}_{\mathcal{G}}$ of $\mathcal{G}$ is the maximum value of any cut. A uniformly random cut has value $\frac{1}{4}$ in expectation, so every graph has value at least $\frac{1}{4}$.

We consider *streaming* algorithms for the problem of estimating the Max-DICUT value $\mathsf{val}_{\mathcal{G}}$ of a directed graph $\mathcal{G}$, given a stream $\boldsymbol{\sigma} = (e_1 = (u_1, v_1), \dots, e_m = (u_m, v_m))$ of the graph's edges in arbitrary order. We say an algorithm is an α-*approximation* for the Max-DICUT problem if its output $\widehat{v}$ satisfies $\alpha \cdot \mathsf{val}_{\mathcal{G}} \le \widehat{v} \le \mathsf{val}_{\mathcal{G}}$ (with high probability). We say an algorithm is a *space-$s(n)$ streaming algorithm* (where n is the number of vertices in $\mathcal{G}$) if it reads the stream of edges $\boldsymbol{\sigma}$ in sequential order and uses $s(n)$ space.

979-8-3503-1894-4/23/$31.00 ©2023 IEEE

DOI 10.1109/FOCS57990.2023.00055

The Max-DICUT problem is one example of a so-called *constraint satisfaction problem (CSP)*. We omit a full definition as we do not require it, but these problems are basically defined by two things: (1) a "global" space of allowed "assignments" to "variables" and (2) a collection of "local" constraints, each of which specifies allowed values for a small subset of variables. For Max-DICUT, variables are vertices, assignments are cuts, and constraints are edges; we will use these terms interchangeably. The "symmetric version" of Max-DICUT is another CSP called *maximum cut* (Max-CUT), in which a cut $\mathbf{x}$ cuts an edge (u, v) if $x_u \neq x_v$; we mention it here as it serves a useful point of comparison for Max-DICUT.

B. Recent work

Over the last decade, there has been extensive work on the approximability of various CSPs in various streaming models [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13]; see also the surveys [14], [15].

Max-DICUT has emerged as the central benchmark for algorithms among CSPs in the streaming setting. It was the first problem shown to admit a non-trivial approximation in sublinear (in n) space in the work of Guruswami, Velingker, and Velusamy [3]. Subsequent work of Chou, Golovnev, and Velusamy [7] gave an improved algorithm for Max-DICUT along with a tight bound on the approximability — pinning the approximability of Max-DICUT for $o(\sqrt{n})$-space streaming at $\frac{4}{9}$.

Theorem I.1 ([7]). *For every $\epsilon > 0$, there is a streaming algorithm (in fact, a linear sketching algorithm) which $(4/9 - \epsilon)$-approximates the* Max-DICUT *value of a graph in $O_\epsilon(\log n)$ space. Conversely, every $(4/9 + \epsilon)$-approximation streaming algorithm for* Max-DICUT *uses $\Omega_\epsilon(\sqrt{n})$ space.*

Both the algorithms in [3] and [7] are what previous works have called "bias-based" algorithms, or what we will call a "zeroth-order snapshot" algorithms. Roughly, the bias of a vertex captures the ratio of its in-degree to its out-degree, and a zeroth-order snapshot computes a histogram of the bias of vertices in the graph and uses this histogram (and no other information) to approximate the Max-DICUT value of a graph. Strikingly, the work of [7] shows that zeroth-order snapshot based algorithms are optimal among $o(\sqrt{n})$-space streaming algorithms.

Subsequent work of Chou, Golovnev, Sudan, and Velusamy [8] showed that this result is part of a broader landscape for $o(\sqrt{n})$-space streaming complexity of CSPs. In particular, Chou, Golovnev, Sudan, and Velusamy [8] proved a *dichotomy theorem* for all finite CSPs. The understanding of Max-DICUT plays a central role in their results. In particular, they generalize the zeroth-order snapshot based Max-DICUT algorithm of [7] to all CSPs. Their lower bounds also generalize the lower bounds from [7] with some notions ("padded one-wise independent problems") that are direct abstractions of Max-DICUT and share tight lower bounds.

One might ask, given a particular CSP, if there are any algorithms that outperform zeroth-order snapshot algorithms studied in [8]. For a wide class of CSPs, including Max-CUT, the answer is "NO" — there are recent $\Omega(n)$-space lower bounds ruling out all nontrivial approximations [6], [10].[1][2] Thus to make advances one has to restrict the problems considered, and in this work we focus on the simplest remaining problem after Max-CUT, namely, Max-DICUT.

For Max-DICUT, till this work and a recent related work by the authors [13] it was conceivable that there were no improvements possible in $o(n)$ space. But at the same time the above mentioned lower bound from [7] did not extend to this setting and it was unclear whether this was due to a limitation of the lower bounds techniques or if better algorithms exist.

In a previous work [13], the authors gave some evidence for the possibility that better algorithms for Max-DICUT do indeed exist. To be precise, recall that the sketching algorithm of [7] is a $\frac{4}{9} \approx 0.444$-approximation, which uses $O(\log n)$ space and is optimal among $o(\sqrt{n})$-space streaming algorithms (Theorem I.1). In [13] we proved that for Max-DICUT, the algorithm of [7] *can* be beaten in certain restricted models such as when the input stream is *randomly* (instead of adversarially) ordered, or the graph has constant maximum-degree. In particular:

Theorem I.2 ([13]). *For every $d \in \mathbb{N}$, there is a streaming algorithm which 0.483-approximates the* Max-DICUT *value of a graph with maximum degree d in $\widetilde{O}_d(\sqrt{n})$ space.*

In doing so the work of [13] introduces the notion that we call a "first-order snapshot" — where information about the input graph is "compressed" to a histogram of edges based on the biases of their two endpoints. (See Definition III.1 below for the definition of a snapshot. We refer here to "first-order" snapshots since higher-order snapshots might maintain a histogram of longer length paths or other subgraphs with more than one edge.) For bounded-degree graphs, mapping a graph to its snapshot is clearly a compression (since the number of possible biases is finite), and this compressed information can be estimated, under fairly natural notions of estimation, by an $\widetilde{O}(\sqrt{n})$-space streaming algorithm.

However, Theorem I.2 does not answer the question of whether the Max-DICUT algorithm of [7] can be beaten on *general* graphs in $o(n)$ space. Indeed, their algorithm breaks down in a fundamental way on general graphs, so it could be considered evidence only that more sophisticated lower bound techniques are necessary to rule out such algorithms. We further discuss why we believe that Theorem I.2 was far from a resolution to this question in Section I-D below.

C. Main result

Our main theorem gives an algorithm that uses slightly more than $\sqrt{n}$ space and outperforms the algorithm of [7]:

[1] $\Omega(n)$ space is tight up to logarithmic factors because randomly sparsifying down to $O(n/\epsilon^2)$ constraints gives $(1-\epsilon)$-approximations.

[2] The condition for inapproximability given in [10] for a predicate $f : \mathbb{Z}_q^k \to \{0, 1\}$ is termed "width", and states that f's support contains some translate of the diagonal $\{(a, \ldots, a) : a \in \mathbb{Z}_q^k\}$. More broadly, the strongest known hardness results for CSPs (e.g., also in [8]) seem to rely on "niceness" properties of the support of f.

Theorem I.3 (Main theorem)**.** *There is a streaming algorithm which* 0.483*-approximates the* Max-DICUT *value of an arbitrary (multi)graph in* $\widetilde{O}(\sqrt{n})$ *space.*

See Theorem IV.2 below for the fully detailed statement.

The fact that we achieve the same approximation factor as [13] is not a coincidence. Both works obtain their final algorithm by constructing a first-order snapshot of the input graph, and then observing that this information suffices to simulate the performance of "oblivious" algorithms on the given input, and finally using a result of Feige and Jozeph [16] that gives an oblivious algorithm to approximate Max-DICUT to a factor of $\approx .483$. (Roughly, oblivious algorithms randomly and independently assign vertices to either the 0-side or the 1-side where the probability of choosing a side depends on the bias of the vertex, and these probabilities are chosen to optimize the expected number of edges crossing the cut — a quantity that can be optimized using just the first-order snapshot information.) While this chain of reasoning is similar, every step becomes more complex in the unbounded-degree setting. Indeed as we explain below, designing algorithms for bounded-degree graphs is and has been substantially easier than the general case.

D. Beyond bounded-degree instances

Before turning to our setting with $\widetilde{O}(\sqrt{n})$ space, we first remark on the role of degree in the earlier works of [3], [7], [8]. The algorithms in all these works work for general degree graphs and use powerful norm estimation algorithms as black boxes. If one were to consider the simpler case of their problems in the bounded-degree setting, these algorithms could have been implemented without reliance on these subroutines. Specifically, their algorithms only need an estimate of the absolute value of "bias times the degree" for a random vertex, and this could be estimated by simply picking a random sample of the vertices and computing their bias and degree as the stream passes by. For general CSPs (even on non-Boolean domains) also such a process would suffice, and this would not only simplify the algorithms significantly, it even would achieve a space bound of $O(\log n)$ which is better than the current bounds given in [8] for general CSPs.

Digging deeper into this analogy one can consider ℓ_p norm estimation problems themselves. For this class of problems also one can define a bounded-degree version of the problem — where one is trying to compute the ℓ_p norm of a vector in $\{-C, \ldots, C\}^n$ in the turnstile update model. In this bounded-degree setting, the ℓ_p norm can be trivially computed by randomly sampling an $O_C(1)$-sized subset of the coordinates and maintaining their values. Thus ℓ_p norms can be estimated in $O(\log n)$ space for every p in this bounded-degree setting, whereas in the general case it is well-known that ℓ_p norm estimation requires polynomial in n space for $p > 2$.

Thus, the bounded-degree setting can be vastly easier to solve and results in this setting may best be viewed as a proof of concept — though even this "proof of concept" may be misleading, as exemplified by the ℓ_p norm estimation problem.

Turning to our specific goal — that of computing (first-order) snapshots of a graph in $\widetilde{O}(\sqrt{n})$ time — our prior work [13] again manages to estimate this snapshot in the bounded-degree setting by sampling $\widetilde{O}(\sqrt{n})$ vertices and maintaining the bias of the sampled vertices as well as the induced subgraph on these vertices. We discuss why this is reasonable in the bounded-degree setting in the following subsection. But such a simple algorithm is definitely not going to work in the general setting! In particular, computing a good estimate of the snapshot is at least as hard as computing the ℓ_1 norm of a vector in the turnstile model with unit updates. Indeed, "snapshot" estimation seems to be a "higher-level" challenge than simple norm estimation and roughly requires computing some "two-wise" marginals of the graph updates, whereas bias corresponded to "one-wise" marginals. Black-box use of norm-estimation algorithms no longer seems to suffice to solve these "two-wise" marginal problems, which seem to need new algorithmic ideas. We feel this class of problems and the ideas used here to deal with them may be of even broader interest than the application to Max-DICUT.

E. Technical overview

Our goal is to approximate the Max-DICUT value of a graph $\mathcal{G}$ by estimating its snapshot $\mathsf{Snap}_{\mathcal{G},\mathfrak{t}}$.

Setting aside the streaming model momentarily, the "gold standard" way to estimate the snapshot would be to sample a small set E of edges uniformly and independently at random, measure the biases of the endpoints of every edge in E, and use this to estimate the snapshot. Unfortunately, since the stream is adversarially-ordered, there is no obvious way to implement this procedure since by the time a "random" edge appears in our stream, many of the edges incident to its endpoints might have already appeared, and thus, we may not know its endpoints' biases.[3]

To get an algorithm for adversarially-ordered streams, we could hope to somehow sample a set E of edges in a way which maintains the property that for every edge in E, we know the bias of its endpoints. While E may not be a uniformly random set of edges, we could still hope for an estimate of the snapshot if E is "sufficiently" random. A natural approach proposed in [13] for doing this is the following. We sample a uniform set S of *vertices* upfront, i.e., before the stream, by uniformly including every vertex with some probability p independently. Then, during the stream, we measure the bias of every vertex in S and store the *induced subgraph* on S as E. Since S is sampled before the stream begins, this approach has the advantage that even though the graph is adversarially-ordered, we end up knowing the biases of the endpoints of every edge in E. Here is where the $\sqrt{n}$ space dependency comes from: By a "birthday paradox" argument[4], since E is the induced subgraph on S, in order to

[3] As observed in [13], when the edges in the stream are *randomly* ordered this simple setup does give an algorithm: One can simply set E to be the first $O(1)$ edges in the stream and then observe the biases of the endpoints over the remainder of the stream.

[4] It suffices to consider only sparse graphs (see Lemma II.9).

expect to even see any edges in E we will need $|S| = \Omega(\sqrt{n})$. But we are very far from done at this point, because there is a crucial issue as compared to the gold standard case: The edges which are included in E are no longer independent! In particular, if two edges e and e' share a common endpoint (or two, in the case of a multigraph!), then conditioning on $e \in E$ increases the probability that $e' \in E$. Here is where the maximum-degree assumption in [13] comes in: If $\mathcal{G}$ has maximum-degree D, $e \in E$ is independent of all but $\leq 2D+1$ events $e' \in E$. It turns out that when $D = O(1)$, this lets us get enough control on the variance of which edges show up in E to give a correct estimate. But for larger D, this approach completely breaks down, and for good reason: In the extreme example of a *star graph* (i.e., a graph where one vertex is connected to all other vertices), we must store the center of the star in S, or otherwise E will be empty! But if we place *every* vertex in S we will use linear space — we want to store the center with probability 1, but the other (low-degree) vertices with probability only $O(1/\sqrt{n})$. Thus, in order to extend this simple estimator to general graphs, we will very roughly want to place vertices in S with probability which increases as a function of their degree. Implementing this in the streaming setting creates numerous challenges, and solving these is a main focus of this paper.

1) Vertex-sampling in the general case: Our goal now is to extend the vertex sampling approach described above to general graphs. We remark that even in the general case, we can assume WLOG that the number of edges in the graph is $\Omega(\sqrt{n})$ and $O(n)$.[5]

As we mentioned in the previous subsection, we would like to sample a set S of vertices, such that every vertex is included independently with probability which increases with the degree. This is the first step towards estimating the snapshot, which will also require sampling edges between these vertices; we focus on the former task for now, and address the latter in the following subsection.

Instead of sampling one set S of vertices, we will aim for a slightly more detailed goal, which is to sample a set U_a of vertices of degree between d_{a-1} and d_a, where $1 = d_1 < \cdots < d_k = O(n)$ is some partition of the possible degrees in the graph. (For concreteness, we use $d_a = 2^{a-1}$.) We envision the graph as consisting of k *layers*; a vertex of degree between d_{a-1} and d_a is in layer a (and has "degree class" a, in analogy to the bias classes).

Consider the task of sampling U_a, a uniform set of vertices in a fixed layer a. Recall that in the previous subsection (the bounded-degree case), we placed all vertices in S with some fixed probability p independently. Now, we would like to place layer-a vertices into U_a with some probability p_a independently.[6] To fit within our space bound, we only require $|U_a| = O(\sqrt{n})$; this turns out to imply that p_a can grow as a function of a. For instance, if $d_a = \Omega(\sqrt{n})$ then there can only be $O(\sqrt{n})$ vertices in layer a, so we can even afford $p_a = 1$. (Making p_a this large in high layers is actually necessary for good estimates, as shown by the "star" example.) But there is a seeming paradox in this plan: *When a vertex v first appears in the stream, we would like to know its layer a, so that we can toss an appropriately biased coin (i.e., Bernoulli-p_a) to determine whether it goes into U_a; but as this is the* first *appearance of v, we know nothing about its degree besides that it is at least* 1*!*

One natural way to deal with this problem is to *defer* deciding whether to a vertex has high degree until we see many edges touching it. To do this, we take advantage of subsampling *edges* as well as vertices. To layer a we also associate an "edge-subsampling probability" q_a and a "subsampled graph" $\mathcal{G}_a$ which includes every edge in $\mathcal{G}$ independently with probability q_a. We choose $q_a = Cd_a^{-1}$ for a large constant C, meaning that vertices with degree d_a in $\mathcal{G}$ have degree roughly C in $\mathcal{G}_a$.[7] This allows us to sample U_a in the streaming setting: We sample $\mathcal{G}_a$ on the fly, and then we add to U_a with probability p_a each new vertex with $\mathcal{G}_a$-degree roughly between $0.49C$ and $1.01C$.[8] Note that $\mathcal{G}_a$ is too large to store — in particular, $\mathcal{G}_1$ has m edges, and more generally $\mathcal{G}_a$ has $q_a m$ edges in expectation — so we will need to carefully choose which edges to store when crafting our estimate in the following subsection.

Now U_a will contain a random sample of layer-a vertices, but — and this is crucial — it may also contain other randomly sampled vertices, like those in layers $a-1$ or $a+1$. E.g., consider a vertex of degree $d_a + 1$, which is technically in layer $a+1$, but is also likely to have $\mathcal{G}_a$-degree under $q_a d_a$; indeed, one cannot differentiate between this vertex and a layer-a vertex with high probability based on $\mathcal{G}_a$-degree. To put it another way, by the time we see the first incident edge to a vertex in $\mathcal{G}_a$, many of its incident edges in $\mathcal{G}$ may have already passed by in the stream, meaning we *cannot track* its "global" bias or degree exactly. This creates a substantial technical issue in even defining the type of estimates we are trying to achieve, which we have to resolve by certain

[5]If the graph has $O(\sqrt{n})$ edges we can afford to store the entire graph within our space bound. A standard sparsification argument (see Lemma II.9) shows replacing $\mathcal{G}$ with a random subsample of $O(n/\epsilon^2)$ edges changes the Max-DICUT value by only ϵ.

[6]There is a technical reason from switching S to U here to denote sets of vertices: It is convenient to think of first sampling a set S_a before the stream to include *all* vertices w.p. p_a (even those not in layer a), and then U_a is the intersection of S_a with the set of vertices in layer a, which are the vertices we actually want to track.

[7]Actually, in order for adequate concentration of the degree in $\mathcal{G}_a$, we will need $C = \Omega(\log n)$, but we ignore this for simplicity. Also, if $d_a < C$, we set $q_a = 1$, i.e., we need no edge-subsampling. This is equivalent to the bounded-degree case we already analyzed.

[8]We are cheating slightly here: We do not know the degree of such a vertex when it first appears, so we cannot decide whether it has degree falling in this range. Instead, during the stream we can store a set $\widehat{U}_a$ containing each vertex with positive $\mathcal{G}_a$-degree w.p. p_a. Then, after the stream, we set U_a to be the set of vertices in $\widehat{U}_a$ with $\mathcal{G}_a$-degree in the appropriate range. This point will come up again when we want to store $\mathcal{G}_a$-edges associated to these vertices in the following subsection; we will have to store edges for every vertex in N_a and then use "cutoffs" to stop storing edges once we know they cannot be in U_a. We ignore these details in this overview.

"smoothing" arguments which we defer until the subsection after the following (Section I-E3).

2) Sampling edges for the estimate: In the previous section, we described a scheme based on vertex- and edge-subsampling which samples uniform sets U_a of layer-a vertices (perhaps along with "borderline" vertices in layer $a+1$ and $a-1$, which we ignore for now). But our ultimate target is estimating the *snapshot*, which counts edges between vertices of different biases; in this sense, these sets of vertices are only means to an end, and we need to also describe how we sample edges *between* these sampled vertices and use them to estimate the snapshot. Recall that in each layer we subsampled a graph $\mathcal{G}_a$, which was still too large to store; the edges we use to produce the estimate will be a subset of $\mathcal{G}_a$'s edges which we can actually store.

Since our algorithm now breaks down vertices by their degrees as well as their biases, it turns out that the natural object to aim to estimate is not the snapshot itself, but instead what we will call the *refined snapshot* of $\mathcal{G}$, denoted $\mathsf{RSnap}_{\mathcal{G},\mathbf{d},\mathbf{t}}$ (see Definition III.3 below). This is a four-dimensional array whose (a,b,i,j)-th entry contains the fraction of edges in $\mathcal{G}$ that go from vertices in bias class i and degree class a to vertices in bias class j and degree class b. Note that the refined snapshot $\mathsf{RSnap}_{\mathcal{G},\mathbf{d},\mathbf{t}}$ is only more granular than the snapshot $\mathsf{Snap}_{\mathcal{G},\mathbf{t}}$ which we actually want to estimate — in particular, $\mathsf{Snap}_{\mathcal{G},\mathbf{t}}$ can be computed from $\mathsf{RSnap}_{\mathcal{G},\mathbf{d},\mathbf{t}}$ by "projecting" the latter to its third and fourth coordinates. Note also that the snapshot's dimensions are constant, while the refined snapshot's dimensions are polylogarithmic (because the number of layers is $k = \Theta(\log n)$). We will abbreviate $A = \mathsf{RSnap}_{\mathcal{G},\mathbf{d},\mathbf{t}}$ for convenience and focus on estimating A by inspecting edges between these sets U_a.

a) Estimating edges within each layer: There is a class of entries (a,b,i,j) of A which are relatively simple to estimate: The "degree diagonal" $a = b$, or in other words, entries which correspond to edges *within a single layer*. For this, we can just store the induced subgraph of $\mathcal{G}_a$ on U_a. Looking at these induced subgraphs — modulo the issue of estimating the bias and degree of sampled vertices — will be roughly equivalent to the bounded-degree case, essentially because vertices in U_a have small degree in $\mathcal{G}_a$. However, this is only a small subset of the entries of A which we need to estimate. Given $a \neq b \in [k]$, how can we estimate the "cross edges" between layers a and b?

b) Estimating cross edges: The difficulty with estimating cross-edges, in comparison to the in-layer edges discussed before, is that there can be wide discrepancies between the degrees of the edges' endpoints (both in the global graph and in any particular layer). That is, for $a < b \in [k]$, vertices in degree class b are expected to have a high degree in layer a (as they are expected to have d_b edges and we subsample with probability Cd_a^{-1}) while the vertices in degree class a are expected to have almost no edges in layer b. This makes the concentration analysis more subtle than the bounded-degree case, but we can still get by with two crucial observations:

1) When looking at the layer a, we can strengthen the algorithm to remember *all* edges in $\mathcal{G}_a$ that are incident on a vertex in U_a (and not just the induced subgraph on U_a).
2) Secondly, as we mentioned in the previous subsection, as the layer a increases, the maximum number of vertices in $\mathcal{G}$ in layer a decreases. This means that we can afford for the probability p_a that any particular layer-a vertex is stored in U_a to increase, and for instance when $d_a = \Omega(\sqrt{n})$ we can even afford $p_a = 1$ as there are only $O(\sqrt{n})$ such vertices.

We claim that, with the above modification, one can estimate cross edges between layers a and b by looking at the graph $\mathcal{G}_a$ and counting the number of edges in this graph that go from vertices in U_a to vertices in U_b. To see why this works, we consider $a = 1$ and two cases for b:

- **When** $d_b = \Omega(\sqrt{n})$**:** In this case, by Item 2, U_b is sufficiently large as to contain all the vertices in $\mathcal{G}$ with degree class b. Given the fact that U_b has all these vertices and we have Item 1, whether or not a cross edge $e = (u,v)$, where u has degree class a and v has degree class b, is counted depends only on whether or not $u \in U_a$ and whether or not $e \in \mathcal{G}_a$. The latter is independent across all edges while the former has only a small amount of dependence, as the vertices in degree class a have low degree in $\mathcal{G}_a$, and does not harm the concentration inequalities too much.
- **When** $d_b = o(\sqrt{n})$**:** The argument above will not directly work in this case, as now whether or not a cross edge is counted also depends on whether or not $v \in U_b$. As the vertices in degree class b have high degree in $\mathcal{G}_a$, this creates a lot of dependencies (depending on d_b/d_a) and breaks the concentration bounds.
  What saves us here is that in Items 1 and 2, we sample all edges in $\mathcal{G}_a$ that are incident on U_a and also are relatively likely to remember any particular vertex in U_b. Thus, the number of cross edges between a and b that are remembered in $\mathcal{G}_a$ is much larger than $O(1)$ (which was the number obtained in the bounded-degree case). Having a larger number of cross edges also means we can also afford to deviate by more without affecting the multiplicative guarantee, and this larger deviation will help us deal with the extra dependencies in this case.

3) The analysis via windowed averaging and smoothing: Even with the modifications above, there is a major problem that we still have to overcome. This problem arises because we do not compute the degrees and biases of the vertices in $\mathcal{G}$ exactly, and instead *estimate* them from the sampled graphs $\mathcal{G}_a$. These estimates will always be slightly off, and this can wreak havoc in the analysis. As we mentioned above, if for instance the degree of a vertex is at the "boundary" of degree classes a and $a+1$, it is impossible to determine with high probability, which entries of the estimate for A will the edges which touch this vertex contribute to — in some subsamples, the vertex could "appear to be" in degree class

a and in others it could be in $a+1$. But morally, if the partition is sufficiently fine, these mistakes should not matter too much anyhow because it is possible to "slightly tweak" the bias of vertices to put them into neighboring classes without significantly changing the Max-DICUT value.

The approach we come up with to circumvent these issues, which may be of independent interest, is the following: Instead of trying to estimate entries of A individually, we group them into "windows" and estimate the average of all the entries in the window instead. For example, when trying to estimate whether a vertex has degree class a, we instead take a windowing parameter w and estimate over all degree classes $\{a-w,\ldots,a+w\}$.[9] Therefore, our algorithm targets a certain kind of "windowed" estimate as opposed to a simple ℓ_1-estimate. In particular, we define a novel (and incomparable) notion of estimating an array which we call *pointwise smooth estimation* (Definition III.16). In this notion of estimation, for an estimate $\widehat{A}$ of A, as δ gets arbitrarily small, we do not require that each entry of $\widehat{A}$ approaches one fixed value; instead, it must approach an *interval* (which gets arbitrarily narrow as w gets arbitrarily large). Sufficiency of this kind of estimate relies on exactly the kind of informal "tweaking" ("smoothing") analysis we mentioned before.

Informally, the intervals are defined as follows: Consider the (a,b,i,j)-th entry of $\widehat{A}$, and any edge e from bias class i' and degree class a' to bias class j' and degree class b'. Then we declare:

- "Inner" edges: If $\|(a,b,i,j)-(a',b',i',j')\|_\infty \le w-1$, then e *must* count in $\widehat{A}(a,b,i,j)$.
- "Outer" edges: If $\|(a,b,i,j)-(a',b',i',j')\|_\infty \ge w+1$, then e *must not* count in $\widehat{A}(a,b,i,j)$.
- "Borderline" edges: If $\|(a,b,i,j)-(a',b',i',j')\|_\infty = w$, then e may or may not count in $\widehat{A}(a,b,i,j)$.

The key lemma in the analysis then states:

Lemma I.4 (Informal version of Lemmas III.11 and III.17)**.** *If an approximation ratio α to* Max-DICUT *can be achieved by looking at the snapshot of a graph $\mathcal{G}$, and $\widehat{A}$ is a (w,δ)-pointwise estimate of the refined snapshot $A=\mathsf{RSnap}_{\mathcal{G},\mathbf{d},\mathbf{t}}$, then $\widehat{A}$ can be used to achieve an $(\alpha-\epsilon)$-approximation to* Max-DICUT *for $\epsilon=O(\delta(k\ell)^2+w\lambda+1/w)$, where k, ℓ, and λ are the number of degree classes in $\mathbf{d}$, the number of bias classes in $\mathbf{t}$, and the maximum width of any interval in $\mathbf{t}$, respectively.*

The three terms in the error ϵ come from, respectively: Switching from an entrywise (ℓ_∞) error guarantee for the refined snapshot $\widehat{A}$ to a global (ℓ_1) error guarantee for the snapshot; a surface area-to-volume ratio bounding errors from the "borderline"; and the "smoothing" operation which tweaks the biases of vertices. Note that there is a significant interplay of parameters here: To achieve any fixed $\epsilon=O(1)$, we'll have to set $w=O(1/\epsilon)$, $\lambda=O(1/w)=O(1/\epsilon^2)$, and since $k=\log n$, we ultimately need $\delta=O(\epsilon^5/\log^2 n)$.[10] Complementarily, Lemma IV.3 is the key correctness statement, stating that such a pointwise smoothed estimate is achieved by our

[9]One technical issue with this approach is we have to handle the "degenerate" cases where, e.g. $a<w$ so the set of allowed degree classes is smaller than $2w+1$. We correspondingly have to weight the entries in the matrix to equalize the contributions of different edges, and this introduces some more potential errors in the algorithm as these "weighting factors" for sampled edges can also be estimated incorrectly. We ignore these details in this introduction.

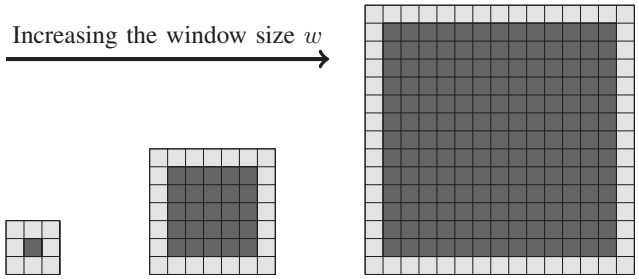

Fig. 1: A depiction of how larger windows reduce the "borderline" effects (in two dimensions). As w becomes larger and larger, a $w\times w$ rectangle (dark gray) dominates its "boundary" (light gray) more and more. Geometrically, a rectangle is two-dimensional while its boundary is "essentially one-dimensional". However, for estimating the Max-DICUT value in a graph, smoothing over size-w windows for large w introduces errors from the use of "continuity" results (i.e., Lemma III.11 below). The right choice of w strikes a balance between these two forces.

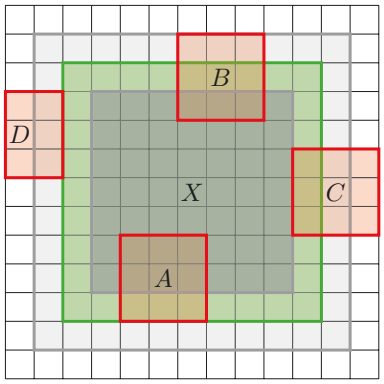

Fig. 2: Consider estimating a (two-dimensional) matrix with "off-by-one" errors, wherein the mass of each entry may shift to one of 8 neighboring entries (red boxes). If we estimate an average over a window of size $w=4$ in taxicab distance around an entry X (green rectangle): (i) "Outer" entries, such as the one marked D, beyond distance $w+1=5$ from the center (light gray) can *never* contribute. (ii) "Inner" entries, such as the one marked A, within distance at most $w-1=3$ from the center (dark gray) *always* contribute. (iii) "Borderline" entries, such as B or C, *may or may not contribute*, depending on the specific error pattern.

[10]Such a guarantee is believable because in the "gold-standard" case where we sample *independently* the random edges and look at their biases (and degrees), the deviations would be $O(1/\sqrt{n})$ by a Chernoff bound.

algorithm. For simplicity, in the algorithm, we only handle a fixed partition degree partition $\mathbf{d}$ (i.e., where $d_a = 2d_{a-1}$). To make λ arbitrarily small, we use the fact that a bias partition can be subdivided arbitrarily while maintaining the approximation ratio of what can be deduced from the snapshot.

F. Future directions

Via the trivial reduction from Max-CUT, it is known that for all $\epsilon > 0$, streaming algorithms which $(\frac{1}{2}+\epsilon)$-approximate Max-DICUT require $\Omega_\epsilon(n)$ space (cf. [10]). There are a number of interesting alternatives for what could happen between $\omega(\sqrt{n})$ and $o(n)$ space. Three main scenarios are:

Scenario 1. "Algorithms beating Theorem I.3": There are $(\frac{1}{2}-\epsilon)$-approximations in $O(\sqrt{n})$ space.

Scenario 2. "First-order snapshots give optimal sublinear-space algorithms": Beating 0.483 requires $\Omega(n)$ space.[11]

Scenario 3. "Approximation vs. space tradeoff": Beating 0.483 can be achieved in $o(n)$ space, but $(\frac{1}{2}-\epsilon)$-approximations for $\epsilon > 0$ require arbitrarily close to $\Omega(n)$ space for arbitrarily small ϵ.

We are particularly interested in the possibility that for Max-DICUT, one can beat the first-order snapshot algorithms we consider here by estimating instead "higher-order snapshots", where by a t-order snapshot we mean a histogram of bias patterns among subgraphs with t edges. These seem to correspond to $n^{1-1/(t+1)}$-space algorithms in the bounded-degree case; is it possible to build on the techniques in this paper to estimate t-order snapshots within this space bound? Conversely, could there be matching "dichotomy" lower bounds — e.g., for Max-DICUT, could first-order snapshot algorithms be optimal in $o(n^{2/3})$ space? Finally, we mention that Singer [17] gives oblivious algorithms for Max-kAND beating the $o(\sqrt{n})$-space approximation ratios calculated in [11]; could our snapshot estimation techniques be extended to work for these problems, or even for all finite CSPs?

Outline of the paper

In the publication version of this paper, we omit many technical proofs, which are left to the full version available on `arXiv` [18]. In Section II we introduce some notation and review some background material. The main technical content of the paper is from Section III onwards, which can be divided into two independent steps. In the first step, we roughly reduce achieving an approximation factor of 0.483 on a graph $\mathcal{G}$ to a problem which we call "pointwise smoothed estimation" of a graph. The basic definitions and statements here are in Section III; we define "continuous snapshot algorithms" (Definition III.18), one of which achieves a 0.483-approximation, and show how they can be simulated given "pointwise smoothed estimates" (Definition III.16); as mentioned above, the reduction itself is divided between Lemmas III.11 and III.17. (The proofs of these lemmas can be found in the full version [18].) In the second step, we show how such a "pointwise smoothed estimate" can be achieved via a streaming algorithm which implements the "edge-and-vertex-subsampling" paradigm outlined above. We present the algorithm in Section IV; the key correctness lemma, Lemma IV.3, which states that (under certain niceness conditions) we achieve a pointwise smoothed estimate, is proven in the full version [18].

II. Preliminaries and notation

$[\ell]$ denotes the set of natural numbers $\{1, \dots, \ell\}$. We use standard asymptotic notation $O(\cdot), o(\cdot)$, etc., with the convention that subscripts (e.g., $f(x,y) = O_y(g(x))$) denote arbitrary dependence in the implicit constant.

A. Matrices and arrays

For $\ell \in \mathbb{N}$, we let $\mathbb{M}^\ell \stackrel{\text{def}}{=} \mathbb{R}^{\ell\times\ell}$ denote the space of real $\ell\times\ell$ matrices, $\mathbb{M}^\ell_{\geq 0} \subseteq \mathbb{M}^\ell$ the space of matrices with nonnegative entries, and $\mathbb{M}^\ell_\Delta \subseteq \mathbb{M}^\ell$ matrices with nonnegative entries summing to 1. For $i, j \in [\ell]$, $M(i,j)$ denotes the (i,j)-th entry of M. Given two matrices $M, N \in \mathbb{M}^\ell$, we let $\|M-N\|_1$ and $\|M-N\|_\infty$ denote their entrywise 1- and ∞-norms, respectively, i.e.,

$$\|M-N\|_1 \stackrel{\text{def}}{=} \sum_{i,j=1}^{\ell} |M(i,j)-N(i,j)|$$

and

$$\|M-N\|_\infty \stackrel{\text{def}}{=} \max_{i,j\in[\ell]} |M(i,j)-N(i,j)|.$$

For $k, \ell \in \mathbb{N}$, we define analogues of this notation for four-dimensional arrays: $\mathbb{A}^{k,\ell} \stackrel{\text{def}}{=} \mathbb{R}^{k\times k\times\ell\times\ell}$ denotes $k\times k\times\ell\times\ell$ arrays, $\mathbb{A}^{k,\ell}_{\geq 0}$ nonnegative arrays, and $\mathbb{A}^{k,\ell}_\Delta$ nonnegative arrays summing to 1; we also define 1- and ∞-norms for arrays. We typically use the letters A and B for four-dimensional arrays, and M and N for (two-dimensional) matrices.

B. (Directed) graphs, degrees, biases, and (directed) cuts

In this paper, we consider directed graphs without self-loops.[12] It will be convenient to use two related definitions, "weighted graphs" and "multigraphs", corresponding to nonnegative real and nonnegative integer edge weights, respectively. In particular, multigraphs will be convenient to encode input to our streaming algorithm, while the more general notion of weighted graphs will be convenient in the analysis.

A *weighted graph* on a vertex-set $V = V(\mathcal{G})$ is defined by an *adjacency matrix* $\mathsf{AdjMat}_\mathcal{G} \in \mathbb{M}^{V\times V}_{\geq 0}$ with zeros on the diagonal. We let $m_\mathcal{G} = \sum_{u,v\in V} \mathsf{AdjMat}_\mathcal{G}(u,v)$ denote the total weight in a weighted graph $\mathcal{G}$.

Given a vertex $v \in V$ in a weighted graph $\mathcal{G}$, we define its *out-* and *in-degrees*

$$\mathsf{deg\text{-}out}_\mathcal{G}(v) \stackrel{\text{def}}{=} \sum_{u\in V} \mathsf{AdjMat}_\mathcal{G}(v,u)$$

[11] Actually, 0.483 is not *exactly* the best we can do; rather, we are interested in the best ratio achievable by "continuous snapshot algorithms" (see Definition III.18 below).

[12] We avoid self-loops because, from the perspective of Max-DICUT (which we are about to define), a self-loop edge is never satisfied by any assignment and is therefore uninteresting from an algorithmic perspective.

and

$$\mathsf{deg\text{-}in}_{\mathcal{G}}(v) \stackrel{\text{def}}{=} \sum_{u \in V} \mathsf{AdjMat}_{\mathcal{G}}(u,v),$$

and its *(total) degree*

$$\mathsf{deg}_{\mathcal{G}}(v) \stackrel{\text{def}}{=} \mathsf{deg\text{-}out}_{\mathcal{G}}(v) + \mathsf{deg\text{-}in}_{\mathcal{G}}(v).$$

If $\mathsf{deg}_{\mathcal{G}}(v) = 0$, we say v is *isolated*; otherwise, we define v's *bias*

$$\mathsf{bias}_{\mathcal{G}}(v) \stackrel{\text{def}}{=} \frac{\mathsf{deg\text{-}out}_{\mathcal{G}}(v) - \mathsf{deg\text{-}in}_{\mathcal{G}}(v)}{\mathsf{deg}_{\mathcal{G}}(v)} \in [-1,1].$$

Finally, for a "cut" $\mathbf{x} \in \{0,1\}^V$, we define its *value* in $\mathcal{G}$

$$\mathsf{val}_{\mathcal{G}}(\mathbf{x}) \stackrel{\text{def}}{=} \frac{1}{m_{\mathcal{G}}} \sum_{u,v \in V} x_v(1-x_u) \cdot \mathsf{AdjMat}_{\mathcal{G}}(u,v),$$

and the overall Max-DICUT value of $\mathcal{G}$ as the maximum value of any cut:

$$\mathsf{val}_{\mathcal{G}}(\mathbf{x}) \stackrel{\text{def}}{=} \max_{\mathbf{x} \in \{0,1\}^V} \mathsf{val}_{\mathcal{G}}(\mathbf{x}).$$

A *multigraph* is a weighted graph where the entries of the adjacency matrix are all integers; equivalently, the graph is specified by a *multiset* of edges $E(\mathcal{G}) \subseteq \{(u,v) : u \neq v \in V(\mathcal{G})\}$, and entries of the matrix equal multiplicities of each edge. Our streaming algorithms will be presented a multigraph with its edges enumerated in arbitrary (adversarial) order, with the goal of achieving an approximation to the Max-DICUT value of a graph.[13]

C. Concentration

We write $\exp(x) = e^{-x}$. We shall need a number of concentration inequalities which operate in different parameter regimes of interest. We list several well-known inequalities as well as some convenient corollaries.

Lemma II.1 (Chernoff upper bound)**.** *Let $X_1, \ldots, X_n$ be independent $\{0,1\}$-valued random variables, and let $X = \sum_{i=1}^n X_i$. Then for all $\delta > 0$,*

$$\Pr[X \geq (1+\delta)\mathbb{E}[X]] \leq \exp(-\delta^2 \mathbb{E}[X]/(2+\delta)).$$

Lemma II.2 (Chernoff lower bound)**.** *Let $X_1, \ldots, X_n$ be independent $\{0,1\}$-valued random variables, and let $X = \sum_{i=1}^n X_i$. Then for all $0 \leq \delta \leq 1$,*

$$\Pr[X \leq (1-\delta)\mathbb{E}[X]] \leq \exp(-\delta^2 \mathbb{E}[X]/2).$$

Corollary II.3 (Two-sided Chernoff bound)**.** *Let $X_1, \ldots, X_n$ be independent $\{0,1\}$-valued random variables, and let $X = \sum_{i=1}^n X_i$. Then for all $0 \leq \delta \leq 1$,*

$$\Pr[|X - \mathbb{E}[X]| \geq \delta \mathbb{E}[X]] \leq 2\exp(-\delta^2 \mathbb{E}[X]/3).$$

[13] As is standard in the streaming and sketching literature, we will have to assume that the length of the stream $m \leq \mathrm{poly}(n)$. Also, one could consider a more general input model, where we get an arbitrary sequence of edges and (nonnegative real) weights, where the edges are possibly repeated, and the maximum and minimum weights are $w_{\max} \leq \mathrm{poly}(n)$ and $w_{\min} \geq 1/\mathrm{poly}(n)$ respectively. In this model, we can only handle unit weights, but this is essentially without loss of generality because one can multiply by roughly $w_{\max}/(w_{\min}\epsilon)$ and then "round" every weight to the nearest integer while preserving the Max-DICUT value up to $O(\epsilon)$.

Corollary II.4 (Chernoff upper bound, high deviation form)**.** *Let $X_1, \ldots, X_n$ be independent $\{0,1\}$-valued random variables, and let $X = \sum_{i=1}^n X_i$. Then for all $\eta \geq 3\mathbb{E}[X]$,*

$$\Pr[X \geq \eta] \leq \exp(-\eta/8).$$

Lemma II.5 (Weighted Chernoff bound [19, cf. Theorem 3.3])**.** *Let $X_1, \ldots, X_n$ be independent $\{0,1\}$-valued random variables. Let $0 < \nu_1, \ldots, \nu_n$ be weights, and let $X = \sum_{i=1}^n \nu_i X_i$. Let $\lambda_0 = \max_i\{\nu_i\}$ and $\lambda_2 = \sum_{i=1}^n \nu_i^2 \mathbb{E}[X_i]$. Then for all $\delta > 0$,*

$$\Pr[X \geq (1+\delta)\mathbb{E}[X]] \leq \exp(-\delta^2 \mathbb{E}[X]^2/2\lambda_2)$$

and

$$\Pr[X \leq (1-\delta)\mathbb{E}[X]] \leq \exp(-\delta^2 \mathbb{E}[X]^2/(2\lambda_2 + \lambda_0 \delta \mathbb{E}[X])).$$

Corollary II.6 (Two-sided weighted Chernoff bound, low weights)**.** *Let $X_1, \ldots, X_n$ be independent $\{0,1\}$-valued random variables. Let $0 < \nu_1, \ldots, \nu_n \leq 1$ be weights, and let $X = \sum_{i=1}^n \nu_i X_i$. Then for all $\delta > 0$,*

$$\Pr[|X - \mathbb{E}[X]| \geq \delta \mathbb{E}[X]] \leq 2\exp(-\delta^2 \mathbb{E}[X]/3).$$

Proof. Follows from the previous lemma since $\lambda_2 \leq \sum_{i=1}^n \nu_i \mathbb{E}[X_i] = \mathbb{E}[X]$ and $\lambda_0 \leq 1$. □

Lemma II.7 (Chebyshev bound)**.** *Let $X_1, \ldots, X_n$ be random variables, and let $X = \sum_{i=1}^n X_i$. Then for all $\eta > 0$,*

$$\Pr[|X - \mathbb{E}[X]| \geq \eta] \leq \frac{\mathsf{Var}[X]}{\eta^2}.$$

Corollary II.8 (Chebyshev with limited independence)**.** *Let $X_1, \ldots, X_n$ be random variables such that $0 \leq X_1, \ldots, X_n \leq 1$, and let $X = \sum_{i=1}^n X_i$. Further, suppose that each X_i is independent (pairwise) of all but D variables $\{X_j\}_{j \in [n]}$. Then for all $\eta > 0$,*

$$\Pr[|X - \mathbb{E}[X]| \geq \eta] \leq \frac{D \cdot \mathbb{E}[X]}{\eta^2}.$$

In particular, if the variables are pairwise independent, then

$$\Pr[|X - \mathbb{E}[X]| \geq \eta] \leq \frac{\mathbb{E}[X]}{\eta^2}.$$

Proof. Follows using $\mathsf{Var}[X] = \sum_{i,j=1}^n \mathbb{E}[X_i X_j] - \mathbb{E}[X_i]\mathbb{E}[X_j]$, the limited independence assumption, and the fact that for all $i, j \in [n]$, $\mathbb{E}[X_i X_j] \leq \mathbb{E}[X_i]$ (using $0 \leq X_i, X_j \leq 1$). □

D. Sparsification for Max-DICUT

The following lemma is a standard statement about sparsification for the Max-DICUT problem, which essentially lets us reduce to considering graphs with linearly many edges. We include the proof in the appendix to the full version [18] for completeness.

Lemma II.9 (Linear sparsification preserves Max-DICUT values)**.** *There exists a universal constant $C_{\mathrm{spar}} > 0$ such that the following holds. For every $\epsilon_{\mathrm{spar}} \in (0,1)$ and $n, m \in \mathbb{N}$, suppose $C_{\mathrm{spar}} n/(\epsilon_{\mathrm{spar}}^2 m) \leq p_{\mathrm{spar}} \leq 1$. Then for every*

(multi)graph $\mathcal{G}$ on n vertices with m edges, if we let $\mathcal{G}_{\mathrm{spar}}$ be the random multigraph resulting from throwing away every edge of $\mathcal{G}$ independently with probability $1 - p_{\mathrm{spar}}$, then with probability $99/100$ over the choice of $\mathcal{G}_{\mathrm{spar}}$, we have $|\mathsf{val}_{\mathcal{G}} - \mathsf{val}_{\mathcal{G}_{\mathrm{spar}}}| \le \epsilon_{\mathrm{spar}}$ and $|m_{\mathcal{G}_{\mathrm{spar}}} - p_{\mathrm{spar}} m| \le \epsilon_{\mathrm{spar}} p_{\mathrm{spar}} m$.

E. k-wise independent hash families

The following definition of a k-wise independent hash family will play a role in the algorithm we present in Section IV below.

Definition II.10. *A family of hash functions $\mathcal{H} = \{h : [n] \to [m]\}$ is* k-wise independent *if it satisfies the following properties:*

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

Lemma II.11 ([20], see e.g. [13, §2.6])**.** *For every $k, n, m = 2^\ell \in \mathbb{N}$, there exists a family of k-wise independent hash functions $\mathcal{H}_k = \{h : [n] \to [m]\}$ such that a uniformly random hash function can be sampled with $O_k(\log n + \log m)$ bits of randomness.*

III. Reducing Max-DICUT approximation to snapshot estimation

In this section, we develop some machinery to reduce the Max-DICUT approximation problem for a graph $\mathcal{G}$ to a problem of estimating a "pointwise snapshot estimate" of $\mathcal{G}$ in the sense of Definition III.16 below. To begin, we formally define snapshots (Definition III.1 below). Then, we define various useful notions of smoothing matrices and arrays. Eventually, the statements of the key reduction lemmas are Lemmas III.11 and III.17. We also discuss how the measuring the snapshot implies approximation algorithms with factor at least 0.483 (Lemma III.19 below).

A. Snapshots

Let $\mathbb{T}^\ell \subseteq \mathbb{R}^{\ell+1}$ denote the space of vectors $\mathbf{t} = (t_0, \ldots, t_\ell)$ such that $t_0 < \cdots < t_\ell$. We call such a vector a *threshold vector of length ℓ*. Given a threshold vector $\mathbf{t} \in \mathbb{T}^\ell$, for any $x \in [t_0, t_\ell]$, we define x's *index* $\mathsf{ind}^{\mathbf{t}}(x)$ (w.r.t. $\mathbf{t}$) as the unique $i \in [\ell]$ such that $t_{i-1} \le x < t_i$ (and if $x = t_\ell$ then $\mathsf{ind}^{\mathbf{t}}(x) = \ell$).

Let $\mathbb{T}^\ell_{\pm 1} \subseteq \mathbb{T}^\ell$ denote the subset of threshold vectors with $t_0 = -1$ and $t_\ell = 1$. We think of such vectors as defining partitions of biases in graphs. For shorthand, given a weighted graph $\mathcal{G}$ and a (nonisolated) vertex v, we write $\mathsf{b\text{-}ind}^{\mathbf{t}}_{\mathcal{G}}(v) \stackrel{\mathrm{def}}{=} \mathsf{ind}^{\mathbf{t}}(\mathsf{bias}_{\mathcal{G}}(v)) \in [\ell]$ for the index representing the "bias class" containing v, and given a pair of nonisolated vertices u, v, we write $\mathsf{b\text{-}ind}^{\mathbf{t}}_{\mathcal{G}}(u, v) \stackrel{\mathrm{def}}{=} (\mathsf{ind}^{\mathbf{t}}(\mathsf{bias}_{\mathcal{G}}(u)), \mathsf{ind}^{\mathbf{t}}(\mathsf{bias}_{\mathcal{G}}(v))) \in [\ell]^2$ for their pair of bias classes. We say $\mathbf{t}$ is λ-*wide* if for every $i \in [\ell]$, $\lambda/2 \le t_i - t_{i-1} \le \lambda$. The width of the partition turns out to factor into the error bound in Lemma III.11 below. (One should think of $\lambda \approx 1/\ell$.)

Definition III.1 (Snapshot ("snapshot" in [13]))**.** *Given a weighted graph $\mathcal{G}$ and threshold vector $\mathbf{t} \in \mathbb{T}^\ell_{\pm 1}$, we define the* snapshot $\mathsf{Snap}_{\mathcal{G},\mathbf{t}} \in \mathbb{M}^\ell_\Delta$ *by*

$$\mathsf{Snap}_{\mathcal{G},\mathbf{t}}(i,j) \stackrel{\mathrm{def}}{=} \frac{1}{m_{\mathcal{G}}} \sum_{u,v=1}^{n} \mathsf{AdjMat}_{\mathcal{G}}(u,v) \mathbb{1}_{\mathsf{b\text{-}ind}^{\mathbf{t}}_{\mathcal{G}}(u,v)=(i,j)}.$$[14]

In other words, the matrix $\mathsf{Snap}_{\mathcal{G},\mathbf{t}}$ counts the weight fraction of edges in the graph between each pair of bias classes. Note that this is a *normalized* matrix, i.e., its entries sum to 1, unlike the adjacency matrix $\mathsf{AdjMat}_{\mathcal{G}}$.

Next, we introduce a new version of a snapshot of a graph $\mathcal{G}$ which also takes into account the degrees of the vertices, which we call the *refined snapshot*. Suppose we also have a threshold vector $\mathbf{d} \in \mathbb{T}^k$ which partitions vertex degrees in $\mathcal{G}$, in the following sense: all nonisolated vertices in $\mathcal{G}$ have degree between d_0 and d_k. We define similar notations: For nonisolated v, we write $\mathsf{d\text{-}ind}^{\mathbf{d}}_{\mathcal{G}}(v) \stackrel{\mathrm{def}}{=} \mathsf{ind}^{\mathbf{d}}(\mathsf{deg}_{\mathcal{G}}(v))$ for the "degree class" of v. (For notational convenience, if $\mathsf{deg}_{\mathcal{G}}(v) = 0$ we will write $\mathsf{d\text{-}ind}^{\mathbf{d}}_{\mathcal{G}}(v) = -\infty$.) For nonisolated u, v, we define:

$$\mathsf{db\text{-}ind}^{\mathbf{d},\mathbf{t}}_{\mathcal{G}}(u,v) = (\mathsf{d\text{-}ind}^{\mathbf{d}}_{\mathcal{G}}(u), \mathsf{d\text{-}ind}^{\mathbf{d}}_{\mathcal{G}}(v), \mathsf{b\text{-}ind}^{\mathbf{t}}_{\mathcal{G}}(u), \mathsf{b\text{-}ind}^{\mathbf{t}}_{\mathcal{G}}(v)). \tag{III.2}$$

This lets us define:

Definition III.3 (Refined snapshot)**.** *Given a weighted graph $\mathcal{G}$ and threshold vectors $\mathbf{t} \in \mathbb{T}^\ell_{\pm 1}$, $\mathbf{d} = (d_0, \ldots, d_k)$, such that every nonisolated vertex $v \in V(\mathcal{G})$ has $d_0 \le \mathsf{deg}_{\mathcal{G}}(v) \le d_k$, we define the* refined snapshot $\mathsf{RSnap}_{\mathcal{G},\mathbf{d},\mathbf{t}} \in \mathbb{A}^{k,\ell}_\Delta$ *by*

$$\begin{aligned}&\mathsf{RSnap}_{\mathcal{G},\mathbf{d},\mathbf{t}}(a,b,i,j)\\ &\stackrel{\mathrm{def}}{=} \frac{1}{m_{\mathcal{G}}} \sum_{u,v=1}^{n} \mathsf{AdjMat}_{\mathcal{G}}(u,v) \mathbb{1}_{\mathsf{db\text{-}ind}^{\mathbf{d},\mathbf{t}}_{\mathcal{G}}(u,v)=(a,b,i,j)}.\end{aligned} \tag{III.4}$$

This array is only more informative than the snapshot; in particular, the snapshot can be recovered via a "projection":

Definition III.5 (Projecting arrays into matrices)**.** *Given an array $A \in \mathbb{A}^{k,\ell}$, we define a matrix $\mathsf{Proj}(A) \in \mathbb{M}^\ell$ by projecting onto the third and fourth coordinates, i.e.,*

$$(\mathsf{Proj}(A))(i,j) = \sum_{a,b=1}^{k} A(a,b,i,j).$$

Fact III.6. *Let $\mathbf{d} = (d_0, \ldots, d_k) \in \mathbb{T}^k$ be a degree partition and let $\mathcal{G}$ be a weighted graph such that all nonisolated vertices have degree between d_0 and d_k. Then $\mathsf{Proj}(\mathsf{RSnap}_{\mathcal{G},\mathbf{d},\mathbf{t}}) = \mathsf{Snap}_{\mathcal{G},\mathbf{t}}$.*

B. Defining windows

To present our formalism for the smoothing analysis, we begin with defining some notations for "windows" around entries in (1-dimensional) vectors, (2-dimensional) matrices,

[14] Note that $\mathsf{b\text{-}ind}^{\mathbf{t}}_{\mathcal{G}}(u,v)$ is not defined if v or u is isolated. But in either case, $\mathsf{AdjMat}_{\mathcal{G}}(u,v)$ vanishes, so we adopt the convention of discarding these terms.

and (4-dimensional) arrays. These will correspond to indices to $[\ell]$, $[\ell]^2$, and $[k]^2 \times [\ell]^2$, respectively, where $k, \ell \in \mathbb{N}$. In each case, windows will correspond to a ball of a certain radius in the ∞-norm.

More concretely, we make the following definitions:

Definition III.7 (Windows)**.** *Suppose $w < \ell \in \mathbb{N}$. For $i \in [\ell]$, let*

$$\mathsf{Win}^{w,\ell}(i) \stackrel{\text{def}}{=} \{i' \in [\ell] : |i' - i| \leq w\}$$

denote the 1-dimensional window *around i. For $i, j \in [\ell]$, let*

$$\begin{aligned}\mathsf{Win}^{w,\ell}(i,j) &\stackrel{\text{def}}{=} \mathsf{Win}^{w,\ell}(i) \times \mathsf{Win}^{w,\ell}(j) \\ &= \{(i',j') \in [\ell]^2 : \max\{|i'-i|, |j'-j|\} \leq w\}\end{aligned}$$

denote the 2-dimensional window *around (i,j). Given also $k > w \in \mathbb{N}$, for $a, b \in [k]$ and $i, j \in [\ell]$, let*

$$\begin{aligned}\mathsf{Win}^{w,k,\ell}(a,b,i,j) &\stackrel{\text{def}}{=} \mathsf{Win}^{w,k}(a,b) \times \mathsf{Win}^{w,\ell}(i,j) \\ &= \{(a',b',i',j') \in [k]^2 \times [\ell]^2 : \max\{|a-a'|, |b-b'|, \\ &\qquad |i-i'|, |j-j'|\} \leq w\}\end{aligned}$$

denote the 4-dimensional window *around (a,b,i,j).*

C. Smoothed estimates (of matrices)

We now define what it means to *smooth* a matrix $M \in \mathbb{M}^\ell$ over windows of size w.

Definition III.8 (Smoothing matrices)**.** *Let $\ell \in \mathbb{N}$ and $M \in \mathbb{M}^\ell$. For $w < \ell \in \mathbb{N}$, we define a* smoothed *matrix $M^{\sim w} \in \mathbb{M}^\ell$ by*

$$M^{\sim w}(i,j) = \sum_{(i',j') \in \mathsf{Win}^{w,\ell}(i,j)} \nu^{\sim w,\ell}(i',j') \cdot M(i',j'),$$

where $\nu^{\sim w,\ell}(i',j') \stackrel{\text{def}}{=} 1/|\mathsf{Win}^{w,\ell}(i',j')|$ is a normalization factor.

Note that the normalization factors $\nu^{\sim w,\ell}(i',j')$ do not depend on the matrix M. Informally, their importance is as follows. Suppose ℓ is even and $\ell > 2w$. Consider the entries $M(1,1)$ and $M(\ell/2, \ell/2)$. The former will contribute to $\approx w^2$ entries in $M^{\sim w}$ — in particular, the indices $\{1, \ldots, w+1\} \times \{1, \ldots, w+1\}$ — while the latter will contribute to $\approx 4w^2$ entries — in particular, $\{\ell/2 - (w+1), \ldots, \ell/2 + (w+1)\} \times \{\ell/2 - (w+1), \ldots, \ell/2 + (w+1)\}$. We will be interested in smoothing snapshots, i.e., $M = \mathsf{Snap}_{\mathcal{G},\mathbf{t}}$, and we would like for the resulting matrices to "resemble" snapshots, at least in the sense of still having entries summing to 1. In particular, the choice of normalization factors ensures that the following holds:

Proposition III.9 (Smoothing preserves entry sum)**.** *For every $M \in \mathbb{M}^\ell$, $\sum_{i,j=1}^{\ell} M^{\sim w}(i,j) = \sum_{i,j=1}^{\ell} M(i,j)$. In particular, if $M \in \mathbb{M}^\ell_\Delta$ then $M^{\sim w} \in \mathbb{M}^\ell_\Delta$.*

The proof is by a simple double-counting argument, and it is included in the full version of this paper [18].

Definition III.10 (Smoothed estimates)**.** *Let $M, \widehat{M} \in \mathbb{M}^\ell$ be matrices, $w \in \mathbb{N}$, and $\epsilon > 0$. $\widehat{M}$ is a (w,ϵ)-*smoothed estimate *of M if $\|\widehat{M} - M^{\sim w}\|_1 \leq \epsilon$.*

For us, the sufficiency of this notion of smoothed estimation is given by the following lemma, which roughly states that anything which can be deduced about $\mathsf{val}_{\mathcal{G}}$ from $\mathsf{Snap}_{\mathcal{G},\mathbf{t}}$ can also be deduced from a (w,ϵ)-smoothed estimate of $\mathsf{Snap}_{\mathcal{G},\mathbf{t}}$ up to an additive factor $O(\lambda w + \epsilon)$. A proof of this lemma is left to the full version [18] of this paper.

Lemma III.11 (Smoothed estimate of snapshot suffices)**.** *There exists a universal constant $C_{\text{smooth}} > 0$ such that the following holds. Let $w < \ell \in \mathbb{N}$. Let $\mathbf{t} \in \mathbb{T}^\ell$ be λ-wide. Let $\mathcal{G}$ be any weighted graph with snapshot $M := \mathsf{Snap}_{\mathcal{G},\mathbf{t}}$. Then there exists a weighted graph $\mathcal{H}$ with snapshot $N := \mathsf{Snap}_{\mathcal{H},\mathbf{t}}$ such that $|\mathsf{val}_{\mathcal{G}} - \mathsf{val}_{\mathcal{H}}| \leq C_{\text{smooth}} \lambda w$, and $\|N - M^{\sim w}\|_1 \leq C_{\text{smooth}} \lambda w$.*

D. Smoothed pointwise estimates (of arrays)

Unfortunately, we will not be able to directly estimate entries of the smoothed snapshot $\mathsf{Snap}^{\sim w}_{\mathcal{G},\mathbf{t}}$ using our sampling-based algorithm. Roughly, this is because there may be huge discrepancy between degrees of vertices (indeed, from $O(1)$ to $\Omega(n)$), and the subsampling parameters will depend on the degree. So, we reduce to the problem of estimating more refined quantities: smoothed versions of the *array* $\mathsf{RSnap}_{\mathcal{G},\mathbf{d},\mathbf{t}}$. Our hope is that, if the degree partition is fine enough, these quantities can actually be estimated.

Definition III.12 (Smoothing arrays)**.** *Let $k, \ell \in \mathbb{N}$ and $A \in \mathbb{A}^{k,\ell}$. For $w < k, \ell \in \mathbb{N}$, we define a* smoothed *array $A^{\sim w} \in \mathbb{A}^{k,\ell}$ by*

$$A^{\sim w}(a,b,i,j) \stackrel{\text{def}}{=} \sum_{(a',b',i',j') \in \mathsf{Win}^{w,k,\ell}(a,b,i,j)} \nu^{\sim w,k,\ell}(a',b',i',j') \cdot A(a',b',i',j'),$$

where $\nu^{\sim w,k,\ell}(a',b',i',j') \stackrel{\text{def}}{=} 1/|\mathsf{Win}^{w,k,\ell}(a',b',i',j')|$ is a normalization factor.

Let $A \in \mathbb{A}^{k,\ell}_{\geq 0}$ be an array with nonnegative entries. For the final step in our reduction, we define a certain notion of a "pointwise smoothed estimate" for an array (Definition III.16 below), and give a statement (Lemma III.17 below) which roughly says that such an estimate implies a smoothed estimate for the snapshot in the sense of Definition III.10 above. Such a "pointwise" estimate will be what we actually aim to achieve in the algorithm, with $A = \mathsf{RSnap}_{\mathcal{G},\mathbf{d},\mathbf{t}}$.

The notion of "pointwise estimate" relies on the definition of two additional arrays $A^{-w}, A^{+w} \in \mathbb{A}^{k,\ell}_{\geq 0}$ which satisfy the inequality $A^{-w} \leq A^{\sim w} \leq A^{+w}$ entrywise and account for "off-by-one" errors when estimating $A^{\sim w}$. We first describe these arrays informally as the actual definitions may appear quite technical. Consider an "estimation" function $\xi : [k]^2 \times [\ell]^2 \to [k]^2 \times [\ell]^2$ for the graph $\mathcal{G}$, which takes as input the index (a,b,i,j) of an entry in the array A, and outputs a tuple $\xi(a,b,i,j)$ which is promised to differ from (a,b,i,j) by at most 1 in each entry. (This will correspond to issues with "borderline" vertices when $A = \mathsf{RSnap}_{\mathcal{G},\mathbf{d},\mathbf{t}}$, as described in

Section I-E above.) Now suppose we tried to estimate the entry $A^{\sim w}(a,b,i,j)$ using the expression

$$\sum_{\xi(a',b',i',j')\in \mathsf{Win}^{w,k,\ell}(a,b,i,j)} \nu^{\sim w,k,\ell}(a',b',i',j')\cdot A(a',b',i',j').$$

The quantities $A^{-w}(a,b,i,j)$ and $A^{+w}(a,b,i,j)$ are lower- and upper-bounds for this expression, respectively, based on the "worst possible" function ξ.

To be more precise, we define the arrays $A^{-w}, A^{+w} \in \mathbb{A}^{k,\ell}_{\geq 0}$ as follows:

Definition III.13 (Upper- and lower-bound normalization factors)**.** *Let $w < k, \ell \in \mathbb{N}$. Then for $a', b' \in [k], i', j' \in [\ell]$, we define*

$$\nu^{-w,k,\ell}(a',b',i',j') \stackrel{\text{def}}{=} \min_{(a'',b'',i'',j'')\in \mathsf{Win}^{1,k,\ell}(a',b',i',j')} \nu^{\sim w,k,\ell}(a'',b'',i'',j''),$$

and

$$\nu^{+w,k,\ell}(a',b',i',j') \stackrel{\text{def}}{=} \max_{(a'',b'',i'',j'')\in \mathsf{Win}^{1,k,\ell}(a',b',i',j')} \nu^{\sim w,k,\ell}(a'',b'',i'',j'').$$

Definition III.14 (Upper- and lower-bound arrays)**.** *Let $w < k, \ell \in \mathbb{N}$ and $A \in \mathbb{A}^{k,\ell}$. We define two arrays $A^{-w}, A^{+w} \in \mathbb{A}^{k,\ell}_{\geq 0}$ by defining, for all $a, b \in [k]$ and $i, j \in [\ell]$:*

$$A^{-w}(a,b,i,j) \stackrel{\text{def}}{=} \sum_{(a',b',i',j')\in \mathsf{Win}^{w-1,k,\ell}(a,b,i,j)} \nu^{-w,k,\ell}(a',b',i',j')A(a',b',i',j'),$$

and

$$A^{+w}(a,b,i,j) \stackrel{\text{def}}{=} \sum_{(a',b',i',j')\in \mathsf{Win}^{w+1,k,\ell}(a,b,i,j)} \nu^{+w,k,\ell}(a',b',i',j')A(a',b',i',j').$$

Note that by definition $\nu^{-w,k,\ell}(a',b',i',j') \leq \nu^{\sim w,k,\ell}(a',b',i',j') \leq \nu^{+w,k,\ell}(a',b',i',j')$ (since $(a',b',i',j') \in \mathsf{Win}^{1,k,\ell}(a',b',i',j')$). Hence, (since A has nonnegative entries) we have $A^{-w}(a,b,i,j) \leq A^{\sim w}(a,b,i,j) \leq A^{+w}(a,b,i,j)$, since they sum over windows around (a,b,i,j) of sizes $w-1$, w, and $w+1$, respectively, using increasing normalization factors. We also have the following simple lem:

Lemma III.15. *Let $w < k, \ell \in \mathbb{N}$ and $A \in \mathbb{A}^{k,\ell}_{\Delta}$. For all $a, b \in [k]$ and $i, j \in [\ell]$, we have*

$$A^{-w}(a,b,i,j) \leq A^{+w}(a,b,i,j) \leq 1.$$

Proof. For the first inequality, see above. For the second, note from Definitions III.12 and III.13 that $\nu^{+w,k,\ell}(a',b',i',j') \leq 1$ implying $A^{+w}(a,b,i,j) \leq \sum_{(a',b',i',j')\in \mathsf{Win}^{w+1,k,\ell}(a,b,i,j)} A(a',b',i',j') \leq 1$. □

Now given the definitions of these upper- and lower-bound arrays A^{+w} and A^{-w}, we are prepared to define our notion of pointwise estimation:

Definition III.16 (Pointwise smoothed estimates)**.** *Let $A, \widehat{A} \in \mathbb{A}^{k,\ell}$ be arrays, $w \in \mathbb{N}$, and $\delta > 0$. $\widehat{A}$ is a (w,δ)-*pointwise smoothed estimate *of A if for every $a, b \in [k], i, j \in [\ell]$,*

$$A^{-w}(a,b,i,j) - \delta \leq \widehat{A}(a,b,i,j) \leq A^{+w}(a,b,i,j) + \delta.$$

Such an estimate (where A is the refined snapshot of a graph) is precisely what is achieved by our algorithm; see Lemma IV.3 below. Correspondingly, we can state our key lemma reducing smoothed estimation of a matrix (in our application, the snapshot) to pointwise smoothed estimation of an array (in our application, the refined snapshot) which projects to that matrix in the sense of Definition III.5:

Lemma III.17 (Pointwise smoothed estimate of array $\Longrightarrow$ smoothed estimate of matrix)**.** *There exists a universal constant $C_{\text{win}} > 0$ such that the following holds. Let $A, \widehat{A} \in \mathbb{A}^{k,\ell}$ be arrays, $w \in \mathbb{N}$, and $\delta > 0$. Suppose $\widehat{A}$ is a (w,δ)-pointwise smoothed estimate of A. Then for $M := \mathsf{Proj}(A)$ and $\widehat{M} := \mathsf{Proj}(\widehat{A})$, $\widehat{M}$ is a (w,ϵ)-smoothed estimate of M, for $\epsilon := \delta(k\ell)^2 + C_{\text{win}}/w$.*

A proof of this lemma is in the full version of this paper [18]. We remark that there is a new factor of $(k\ell)^2$ in the error term corresponding to the product of dimensions of the array; this is because we are switching from an ℓ_∞-type guarantee to an ℓ_1-type guarantee. Recall also that in our application (where A is the refined snapshot), we will have $k = \Theta(\log n)$; this means we need δ to be shrinking as a function of n to apply the lemma, but this turns out to be achievable.

E. Snapshot algorithms

The final piece of the puzzle is actually identifying how snapshots let us reason about the Max-DICUT value of a graph. We make the following definition for algorithms which do exactly this:

Definition III.18 (Continuous snapshot algorithms)**.** *A* continuous snapshot algorithm *is defined by a threshold vector $\mathbf{t} \in \mathbb{T}^\ell$ and a function $\mathcal{A} : \mathbb{M}^\ell \to [0,1]$ such that:*

1) Correctness: *For every weighted graph $\mathcal{G}$, $\mathcal{A}(\mathsf{Snap}_{\mathcal{G},\mathbf{t}}) \leq \mathsf{val}_{\mathcal{G}}$.*
2) Continuity: $|\mathcal{A}(M) - \mathcal{A}(N)| \leq \|M - N\|_1$.

We say $\mathcal{A}$ is an α-approximation *if for every weighted graph $\mathcal{G}$, $\alpha\,\mathsf{val}_{\mathcal{G}} \leq \mathcal{A}(\mathsf{Snap}_{\mathcal{G},\mathbf{t}})$.*

Our aim will be to implement any such algorithm as a streaming algorithm via the snapshot estimation machinery we described above. In particular, Feige and Jozeph [16] showed that one such algorithm achieves a 0.483-approximation:

Lemma III.19 (Implied by Feige and Jozeph [16])**.** *There exists a constant $\alpha_{\text{FJ}} \in (0.4835, 0.4899)$, $\ell_{\text{FJ}} \in \mathbb{N}$, a vector of bias thresholds $\mathbf{t}_{\text{FJ}} \in \mathbb{T}^{\ell_{\text{FJ}}}_{\pm 1}$, and a vector of probabilities $\mathbf{r}_{\text{FJ}} = (r_1, \ldots, r_\ell) \in [0,1]^{\ell_{\text{FJ}}}$ such that the function $\mathcal{A}(M) := \sum_{i,j=1}^{\ell_{\text{FJ}}} r_i(1-r_j)M(i,j)$ is a continuous snapshot algorithm achieving a α_{FJ}-approximation.*

Though we will not need this fact, we remark that the estimate for $\mathsf{val}_{\mathcal{G}}$ given in the lemma corresponds to a

simple randomized assignment for $\mathcal{G}$, namely the so-called "oblivious" assignment which assigns each nonisolated vertex $v \in V(\mathcal{G})$ to 1 with probability r_i and 0 otherwise, where $i = \mathsf{b\text{-}ind}^{\mathbf{t}}_{\mathcal{G}}(v)$ is its bias class.

One final notion we will need regarding snapshot algorithms is the following. Suppose we have a (continuous) snapshot algorithm $\mathcal{A}_{\mathtt{orig}}$ with a threshold vector $\mathbf{t}_{\mathtt{orig}}$, and we want to reduce the *width* of $\mathbf{t}_{\mathtt{orig}}$, i.e., the size of the largest interval, yielding a new (continuous) snapshot algorithm with a smaller-width threshold vector. (We will want to do this in order to apply Lemma III.11.) Consider greedily packing intervals of width $\leq \lambda$ within each interval in $\mathbf{t}$ in some canonical way; the resulting partition, which we denote by $\mathtt{RefinePart}(\mathbf{t}_{\mathtt{orig}})$, has $\ell' \leq \ell + 1/\lambda$ intervals. There is a natural corresponding snapshot algorithm, which given an $\ell' \times \ell'$ snapshot collapses it to the corresponding $\ell \times \ell$ snapshot and runs $\mathcal{A}$, and remains an α-approximation. We denote this algorithm by $\mathtt{RefineAlg}_{\lambda}(\mathbf{t}_{\mathtt{orig}}, \mathcal{A}_{\mathtt{orig}})$.

IV. The algorithm

In this section, we present an algorithm that approximates the Max-DICUT value of an arbitrary multigraph, thereby proving Theorem I.3 (in its full version Theorem IV.2 below). The algorithm itself has to deal with a number of technical issues, so we begin by outlining the algorithm and giving a number of pointers which hopefully make it easier to digest.

A. Overview

a) Informal recap: At the highest level, we want to be able to sample random edges in a multigraph $\mathcal{G}$ and estimate the biases of their endpoints. In particular, we fix some global partition $\mathbf{t} \in \mathbb{T}^{\ell}_{\pm 1}$ of the interval $[-1, +1]$ of possible biases into ℓ "bias classes". We want to estimate the snapshot $\mathsf{Snap}_{\mathcal{G},\mathbf{t}}$ of the input graph $\mathcal{G}$, which is a matrix whose entries count the number of edges between each pair of bias classes. This can be used to simulate a "continuous snapshot algorithm" applied to $\mathcal{G}$ (Definition III.18), and one such algorithm achieves a 0.483-approximation (Lemma III.19).

To do this, we fix some global partition $d_0 \leq \cdots \leq d_k$ of degrees in $\mathcal{G}$ dividing the graph's vertices into "layers", where layer a is the vertices of degree between d_{a-1} and d_a, and aim to estimate the "refined snapshot" $\mathsf{RSnap}_{\mathcal{G},\mathbf{d},\mathbf{t}}$ (Definition III.3), which counts the number of edges whose endpoints are in particular bias classes and layers. Towards this, we hope to obtain representative samples $U_1, \ldots, U_k$ of vertices, U_a containing vertices in layer (roughly) a, such that (1) we (roughly) know the bias classes of every sampled vertex and (2) we also have representative samples of edges between the layers.

More precisely, when we say "roughly" we mean "up to $w\pm 1$" allowing us to apply the "smoothing" techniques developed in Section III. Concretely, this means that we target producing a "pointwise smoothed estimate" of the refined snapshot in the sense of Definition III.16; the claim that we can achieve this is Lemma IV.3 below.

b) Outline of the algorithm: The outline of our scheme is as follows. For each "layer" $a \in \{1, \ldots, k\}$, we fix some probabilities $q_a, p_a \in [0, 1]$. We subsample the edges of $\mathcal{G}$ with probability q_a and then subsample positive-degree vertices in $\mathcal{G}_a$ with probability p_a to get a subset of vertices, which we denoted $\mathtt{vStored}_a$ in the algorithm. We hope to store the $\mathcal{G}_a$-neighborhoods of these vertices to get a subset of edges denoted $\mathtt{eStored}_a$. Ideally, we can use $\mathtt{eStored}_a$ to estimate the degrees and biases of vertices in $\mathtt{vStored}_a$, and further, if we see an edge stored in $\mathtt{eStored}_a$ between vertices in $\mathtt{vStored}_a$ and $\mathtt{vStored}_b$, we can count it in the appropriate entry of the refined snapshot. However, note that $\mathtt{eStored}_a$ cannot necessarily store all neighbors for every vertex in $\mathtt{vStored}_a$; in particular, while our edge subsampling ensures that $\mathtt{vStored}_a$ is unlikely to contain vertices of $\mathcal{G}$-degree much less than d_a, it may still contain vertices of $\mathcal{G}$-degree much *greater* than d_a, in which case we can not hope to store all its neighbors.

We write the algorithm in a particularly "well-factored" form called a "sketching algorithm"; while we omit the full definition of such an algorithm, we remark that it essentially means that the algorithm behaves "independently" on each edge arriving in the stream. In particular, we can construct a "sketch" corresponding to each individual edge e, which consists of a pair $(\mathtt{vStored}_a, \mathtt{eStored}_a)$ for each layer a, where $\mathtt{vStored}_a$ contains either endpoint with probability p_a independently and $\mathtt{eStored}_a$ with probability q_a. Then, we can "compose" the sketches for the entire stream together by taking the unions of the sets in these sketches, up to appropriate cutoffs.

It is useful to recall that the subsampling probabilities q_a (for edges) and p_a (for vertices) decrease and increase, respectively, as a function of the layer number a.

c) Some pointers: It is very helpful to remember that there are two distinct sources of randomness in the algorithm: We "sparsify" the graph by subsampling edges, and then subsample which vertices we actually store. In the analysis we will first analyze this edge-subsampling, and then conditioned on certain "good outcomes" for the edge-subsampling we will analyze the vertex-subsampling. Correspondingly, it is useful to keep in mind the graph $\mathcal{G}_a$ consisting of all the edges sampled in layer a (each is sampled w.p. q_a) and the set $\mathcal{N}_a$ of positive-degree vertices in $\mathcal{G}_a$. (However, the actual streaming algorithm is not necessarily be able to store these sets as they can grow too large; it only stores subsets $\mathtt{eStored}_a$ and $\mathtt{vStored}_a$, respectively.)

When we actually see an edge $e = (u, v)$ in $\mathcal{G}_a$, how do we know what to do with it (i.e., which entry of the matrix should it contribute to)? Note that $\mathbb{E}[\mathsf{deg}_{\mathcal{G}_a}(v)] = q_a \mathsf{deg}_{\mathcal{G}}(v)$. Thus, we can hope to use $q_a^{-1}\mathsf{deg}_{\mathcal{G}_a}(v)$ (which we call v's "apparent degree") as an estimate for $\mathsf{deg}_G(v)$. Roughly, this should work out if $\mathsf{deg}_{\mathcal{G}}(v)$ is decently large; for instance, we can use the Chernoff bound to show that w.h.p. $\mathsf{deg}_{\mathcal{G}}(v)/2 \leq q_a^{-1}\mathsf{deg}_{\mathcal{G}_a}(v) \leq 2\mathsf{deg}_{\mathcal{G}}(v)$, so the apparent degree moves by at most 1 interval relative to the actual degree. The same sort of analysis is necessary to analyze the bias of a vertex and

ultimately prove that we get a so-called "pointwise smoothed estimate".

Finally, one remaining technical issue stems from the fact that when we want to subsample a set of nonisolated vertices in the subsampled graph $\mathcal{G}_a$, but we do not know the nonisolated vertices ahead of time. In particular, each time we see a *new* nonisolated vertex we want to toss a p-biased coin — but if we decide *not* to store a vertex, we need to "remember" this decision if we happen to see it again. This would be manageable if the algorithm had random access to the results of n biased coin flips, but this model would be somewhat nonstandard. Instead, as in [13], we observe that when proving concentration it is sufficient to have four-wise independence in the vertex-subsampling procedure, and thus, we decide whether to store a vertex by plugging it into a previously sampled four-wise independent hash function (see Lemma II.11).

B. Describing the algorithm

The goal of this section is to prove Theorem I.3. We begin by presenting an algorithm (Algorithm 1 below) for estimating the Max-DICUT value of a stream of edges corresponding to a graph $\mathcal{G}$ given an estimate for the number of edges in $\mathcal{G}$. This algorithm first produces the sketch for the stream containing the sampled vertices and edges (via a "sketch" subroutine, Algorithm 2, and a "compose" subroutine, Algorithm 3), and then feeds this sketch into another subroutine, Algorithm 4, which estimates degrees and biases among the sampled vertices and counts sampled edges, creates an estimate for the refined snapshot of the graph, and uses this to approximate the graph's Max-DICUT value. The key correctness lemma, Lemma IV.3 below, states that this estimate for the refined snapshot is a "pointwise smoothed estimate", allowing us to apply the machinery from Section III.

We begin with several tables containing definitions of parameters to be used in the algorithms.

Notation	Value	Description
$w \in \mathbb{N}$	$1/\epsilon$	Size of windows for smoothing
$\lambda > 0$	ϵ/w	Maximum width of intervals in the refinement $\mathbf{t}$ of $\mathbf{t}_{\mathtt{orig}}$ (see next line)
$\mathbf{t} \in \mathbb{T}^{\ell}$	$\mathtt{RefinePart}_{\lambda}(\mathbf{t}_{\mathtt{orig}})$	Refinement of $\mathbf{t}_{\mathtt{orig}}$ into ℓ intervals of width at most λ
$\ell \in \mathbb{N}$	$\leq \ell_{\mathtt{orig}} + 1/\lambda$	Number of intervals in $\mathbf{t}$
$\mathcal{A}$	$\mathtt{RefineAlg}_{\lambda}(\mathbf{t}_{\mathtt{orig}}, \mathcal{A}_{\mathtt{orig}})$	Corresponding refinement of $\mathcal{A}_{\mathtt{orig}}$

TABLE I: Global parameters determined by ϵ alone.

Notation	Value	Description
$m_{\mathtt{min}} \in \mathbb{N}$	$\sqrt{n}$	Minimum number of edges handled
$m_{\mathtt{max}} \in \mathbb{N}$	$C_{\mathrm{spar}} n/(\epsilon)^2$ where C_{spar} is as in Lemma II.9	Maximum number of edges handled (see Lemma II.9)
$k^* \in \mathbb{N}$	$6 \log \log n$	Number of degree intervals before we begin subsampling edges
$D \in \mathbb{N}$	2^{k^*+w+2}	W.h.p. bound on max-degree of "counted vertices" in subsampled graphs (parameter for space bound), note that $D = O_{\epsilon}(\log^6 n)$
$\mathtt{eCutoff}$	$\log^7 n$	Maximum number of stored neighbors per vertex

TABLE II: Global parameters determined by ϵ and n.

Notation	Value	Description
$k \in \mathbb{N}$	$\log(2\widehat{m})$	Number of degree intervals (we will have $\widehat{m} \leq m_{\mathtt{max}}$ and thus, $k = O_{\epsilon}(\log n)$)
$\rho > 0$	$1000\sqrt{D} \cdot (k\ell)^3/\epsilon$	Factor controlling space usage, note that $\rho = O_{\epsilon}(\log^6 n)$ (assuming $\widehat{m} \leq m_{\mathtt{max}}$)
$p_0 > 0$	$\rho/\sqrt{\widehat{m}}$	Factor in vertex-subsampling probability
$\mathtt{vCutoff}$	$10\rho\sqrt{2\widehat{m}}$	Maximum number of stored vertices per layer, note that $\mathtt{vCutoff} = O_{\epsilon}(\sqrt{n}\log^6 n)$ (assuming $\widehat{m} \leq m_{\mathtt{max}}$)

TABLE III: Global parameters determined by ϵ, n, and $\widehat{m}$.

Algorithm 1 Our algorithm.

Input: A multigraph $\mathcal{G}$ and an estimate $\widehat{m}$ for the number of edges.

1: For all $a \in [k]$, sample a hash function $\pi_a : [n] \to [1/p_a]$ from $\mathcal{H}_4(n, 1/p_a)$ (see Lemma II.11).
2: Initialize an empty sketch $(m, ((\mathtt{vStored}_a, \mathtt{eStored}_a)_{a\in[k]})$.
3: **for** each edge (u, v) in the stream **do**
4: Use Algorithm 3 to combine the current sketch with the sketch of (u, v) according to Algorithm 2, and store the result in the current sketch.
5: **end for**
6: Run Algorithm 4 on the final sketch to obtain the output.

Notation	Value	Description
$d_a \in \mathbb{N}$	2^a	Degree partition. We also define $\mathbf{d} = (d_0, \ldots, d_k)$ where $d_0 = 1$.
$q_a \in [0, 1]$	$\min\{2^{k^*-a}, 1\}$	Edge-sampling probability
$p_a \in [0, 1]$	$\min\{p_0 q_a^{-1}, 1\}$	Vertex-sampling probability

TABLE IV: "Per-layer" parameters defined for all $a \in [k]$ and determined by ϵ, n, and $\widehat{m}$.

Algorithm 2 Sketch for an input edge.

Input: An edge (u, v), an estimate $\widehat{m}$ of the total number of edges, and hash functions $\{\pi_a : [n] \to [1/p_a]\}$.

Set $m \leftarrow 1$.
for $a = 1, \ldots, k$ **do**
  Toss a biased coin which is 1 with probability q_a, and let z denote its output.
  if $z = 1$ **then**
    $\texttt{vStored}_a = \{v' \mid v' \in \{u, v\} \wedge \pi_a(v') = 1\}$.
    If $\texttt{vStored}_a \neq \emptyset$, set $\texttt{eStored}_a \leftarrow \{(u, v)\}$. Otherwise, set $\texttt{eStored}_a \leftarrow \emptyset$.
  else
    $\texttt{vStored}_a, \texttt{eStored}_a \leftarrow \emptyset$.
  end if
end for
Output the sketch $\left(m, (\texttt{eStored}_a, \texttt{vStored}_a)_{a\in[k]}\right)$.

Algorithm 3 Combining two sketches.

Input: Two sketches $(m^{(1)}, (\texttt{eStored}_a^{(1)}, \texttt{vStored}_a^{(1)})_{a\in[k]})$ and $(m^{(2)}, (\texttt{eStored}_a^{(2)}, \texttt{vStored}_a^{(2)})_{a\in[k]})$.

Set $m \leftarrow m^{(1)} + m^{(2)}$.
for $a = 1, \ldots, k$ **do**
  Set $\texttt{vStored}_a \leftarrow \perp$ if either $\texttt{vStored}_a^{(1)}$ or $\texttt{vStored}_a^{(2)}$ is $\perp$, or if $|\texttt{vStored}_a^{(1)} \cup \texttt{vStored}_a^{(2)}| > \texttt{vCutoff}$. Otherwise, set $\texttt{vStored}_a \leftarrow \texttt{vStored}_a^{(1)} \cup \texttt{vStored}_a^{(2)}$.
  If $\texttt{vStored}_a = \perp$, then set $\texttt{eStored}_a \leftarrow \perp$. Else, set $\texttt{eStored}_a \leftarrow \texttt{eStored}_a^{(1)} \cup \texttt{eStored}_a^{(2)}$ (multi-set union). Iteratively remove all edges (u, v) from $\texttt{eStored}_a$ for which we have for all $v' \in \{u, v\}$ that either $v' \notin \texttt{vStored}_a$ or $\deg_{\texttt{eStored}_a}(v') > \texttt{eCutoff}$.
end for
Output the sketch $\left(m, ((\texttt{eStored}_a, \texttt{vStored}_a))_{a\in[k]}\right)$.

Algorithm 4 Computing the output from a sketch.

Input: A sketch $\left(m, (\texttt{eStored}_k, \texttt{vStored}_k)_{a\in[k]}\right)$.

If $\exists a \in [k] : \texttt{eStored}_a = \perp$, then return $\perp$.
For all $a \in [k]$ and $v \in \texttt{vStored}_a$, define:

$$\texttt{dEst}_a(v) = \min\{q_a^{-1} \cdot \deg_{\texttt{eStored}_a}(v), d_k\}$$

and

$$\texttt{bEst}_a(v) = \mathsf{bias}_{\texttt{eStored}_a}(v).$$

For all $a \in [k]$ and $i \in [\ell]$, define:

$$\texttt{vEst}_{a,i} = \Big\{v \in \texttt{vStored}_a \mid \deg_{\texttt{eStored}_a}(v) < \texttt{eCutoff} \wedge \mathsf{ind}^{\mathbf{d}}(\texttt{dEst}_a(v)) \in \mathsf{Win}^{w,k}(a) \wedge \mathsf{ind}^{\mathbf{t}}(\texttt{bEst}_a(v)) \in \mathsf{Win}^{w,\ell}(i)\Big\}.$$

For all $a, b \in [k]$ and $i, j \in [\ell]$, define:

$$\texttt{AEst}_{a,b,i,j} = \sum_{(u,v)\in\texttt{eEst}_{a,b,i,j}} \nu\texttt{Est}_{a,b}^{w,k,\ell}(u, v),$$

where:

$$\texttt{eEst}_{a,b,i,j} = \texttt{eStored}_{\min\{a,b\}} \cap (\texttt{vEst}_{a,i} \times \texttt{vEst}_{b,j}),$$

and

$$\nu\texttt{Est}_{a,b}^{w,k,l}(u, v) = \nu^{\sim w,k,\ell}\big(\mathsf{ind}^{\mathbf{d}}(\texttt{dEst}_a(u)), \mathsf{ind}^{\mathbf{d}}(\texttt{dEst}_b(v)), \mathsf{ind}^{\mathbf{t}}(\texttt{bEst}_a(u)), \mathsf{ind}^{\mathbf{t}}(\texttt{bEst}_b(v))\big).$$

Define the array $\widehat{A} \in \mathbb{A}_{\geq 0}^{k,\ell}$ as:

$$\widehat{A}(a, b, i, j) = \frac{\texttt{AEst}_{a,b,i,j}}{m q_{\min\{a,b\}} p_a p_b}. \tag{IV.1}$$

Define $\widehat{M} \leftarrow \mathsf{Proj}(\widehat{A}) \in \mathbb{M}^\ell$ and output $\widehat{v} \leftarrow \mathcal{A}(\widehat{M}) - \frac{\epsilon}{4}$.

C. Analyzing the algorithm

In this subsection, we prove our main result (Theorem I.3) which asserts that we can achieve a $\widetilde{O}(\sqrt{n})$-space 0.483-approximation algorithm. The essence of this algorithm is Algorithm 1; unfortunately we cannot quite use the latter directly, because it requires an estimate for the number of edges in $\mathcal{G}$ which we do not have *a priori*, but this can be fixed with some standard tricks.

In fact, we prove the following more detailed theorem:

Theorem IV.2. *For every fixed continuous snapshot algorithm $(\mathbf{t}_{\texttt{orig}}, \mathcal{A}_{\texttt{orig}})$ achieving a ratio of α, the following holds. Let $\epsilon > 0$. There is a sketching algorithm which, given the edges (in adversarial order) of a multigraph $\mathcal{G}$ on n vertices and m edges, uses $2^{O(1/\epsilon)} \cdot \sqrt{n} \cdot \log^{O(1)}(n + m)$ space and, with probability at least $9/10$, outputs a value $\widehat{v}$ satisfying $(\alpha - \epsilon)\mathsf{val}_{\mathcal{G}} \leq \widehat{v} \leq \mathsf{val}_{\mathcal{G}}$.*

For correctness of the algorithm, we will need the following lemma, whose proof we defer to later:

Lemma IV.3. *Let $\epsilon > 0$, $n, m, \widehat{m} \in \mathbb{N}$, and $\mathcal{G}$ be a multigraph on n vertices and $m_{\texttt{min}} \leq m \leq m_{\texttt{max}}$ edges and such that*

$m/2 \le \widehat{m} \le 2m$. *Let* $A = \mathsf{RSnap}_{\mathcal{G},\mathbf{d},\mathbf{t}}$ *and* $\widehat{A}$ *be as computed by Eq.* (IV.1) *in the call to Algorithm 4 in Algorithm 1. Letting* w, k, ℓ *be defined as in Tables I and III, we have with probability at least* $99/100$ *that* $\widehat{A}$ *is a* $(w, \epsilon/(k\ell))^2$*-pointwise smoothed estimate of* A*.*

Returning to the main claim:

Proof. We assume for simplicity that we know an *a priori* bound $m < n^C$. The algorithm we use will run the following sketches in parallel:

- A sketch to count the number of edges m in the input graph.
- A "buffer" sketch to store up to $m_{\mathtt{min}}$ edges from the input graph (and $\perp$ if there are more).
- For $t \in \left\{\log_{1.9} m_{\mathtt{min}}, \dots, \log_{1.9}(n^C)\right\}$, set $\widehat{m}(t) \leftarrow 1.9^t$, $\widehat{m}_{\mathrm{spar}}(t) \leftarrow \min(m_{\mathtt{max}}, \widehat{m}(t))$, and $p_{\mathrm{spar}}(t) \leftarrow \widehat{m}_{\mathrm{spar}}(t)/\widehat{m}(t)$. Obtain a graph stream $\widehat{\mathcal{G}}(t)$ by including each edge of $\widehat{\mathcal{G}}$ with probability $p_{\mathrm{spar}}(t)$ independently and run Algorithm 1 with $\widehat{\mathcal{G}}(t)$ and $\widehat{m}_{\mathrm{spar}}(t)$ to get an output.

After the stream, if $m \le m_{\mathtt{min}}$ we solve the instance (which we have stored) exactly. Otherwise, we choose the unique t such that $\widehat{m}(t) \le m < 1.9 \cdot \widehat{m}(t)$ and return the output corresponding to this t.

The proof of the theorem now proceeds in several steps.

a) Space bound: In the buffer we store at most $m_{\mathtt{min}}$ edges, which takes $\widetilde{O}(m_{\mathtt{min}}) = \widetilde{O}(\sqrt{n})$ space. Then, we invoke Algorithm 1 on $O(C \log n)$ values of t, so it suffices to show the bound separately for each value of t. For any such t, note that each sketch created by Algorithm 2 for a single edge is at most $k = O(\log n)$ copies of the edge and its endpoints, and therefore this sketch has size $O(\log^2 n)$. We now bound the size of the sketch obtained by combining the sketches using Algorithm 3. which means that it suffices to bound the size of the pairs $(\mathtt{eStored}_1, \mathtt{vStored}_1), \dots, (\mathtt{eStored}_k, \mathtt{vStored}_k)$. For this note that the number a of such pairs is $O(\log n)$ and thus, it suffices to bound the size of each pair. For any such pair a, the number of vertices in $\mathtt{vStored}_a$ is at most $\mathtt{vCutoff}$ (due to Algorithm 3) and the number of edges is at most $\mathtt{vCutoff} \cdot \mathtt{eCutoff}$ (due to Algorithm 3). Finally, we observe that $\mathtt{eCutoff} = \log^{O(1)} n$ and, since $\widehat{m}_{\mathrm{spar}}(t) \le m_{\mathtt{max}}$, we have $\mathtt{vCutoff} = 2^{O(1/\epsilon)} \cdot \sqrt{n} \cdot \log^{O(1)} n$.

b) Reducing to "correct" $\widehat{m}$: If $m \le m_{\mathtt{min}}$ we solve the instance exactly. Otherwise, we return the output for t satisfying $\widehat{m}(t) \le m < 1.9 \cdot \widehat{m}(t)$. For this t, we have:

$$\widehat{m}_{\mathrm{spar}}(t) \le m \cdot p_{\mathrm{spar}}(t) \le 1.9 \cdot \widehat{m}_{\mathrm{spar}}(t).$$

Using this and Lemma II.9 with $\epsilon_{\mathrm{spar}} = \epsilon$ (note that the conditions of Lemma II.9 are satisfied as either $p_{\mathrm{spar}}(t) = 1$ and the lemma is trivial or $p_{\mathrm{spar}}(t) = m_{\mathtt{max}}/\widehat{m}(t) \ge \frac{C_{\mathrm{spar}} n}{(\epsilon)^2 m}$), we have with probability at least $99/100$ that $\left|\mathsf{val}_{\mathcal{G}} - \mathsf{val}_{\widehat{\mathcal{G}}(t)}\right| \le \epsilon$ and the number of edges in $\widehat{\mathcal{G}}(t)$ is between $\widehat{m}_{\mathrm{spar}}(t)/2$ and $2\widehat{m}_{\mathrm{spar}}(t)$.

c) Applying the reduction: Define $\widehat{\mathcal{G}} := \widehat{\mathcal{G}}(t)$. By Lemma IV.3, $\widehat{A}$ is a $(w, \epsilon/(k\ell)^2)$-pointwise smoothed estimate of $A = \mathsf{RSnap}_{\widehat{\mathcal{G}},\mathbf{d},\mathbf{t}}$ with probability $99/100$. Conditioning on this event, we can then apply Lemma III.17 to conclude that $\widehat{M}$ is a $(w, \epsilon + C_{\mathrm{win}}/w)$-smoothed estimate of $M := \mathsf{RSnap}_{\widehat{\mathcal{G}},\mathbf{t}}$, i.e., $\|\widehat{M} - M^{\sim w}\|_1 \le \epsilon + C_{\mathrm{win}}/w$.

By Lemma III.11, there exists a weighted graph $\mathcal{H}$ with snapshot $N := \mathsf{Snap}_{\mathcal{H},\mathbf{t}}$ such that $\|N - M^{\sim w}\|_1 \le C_{\mathrm{smooth}} \lambda w$ and $|\mathsf{val}_{\widehat{\mathcal{G}}} - \mathsf{val}_{\mathcal{H}}| \le C_{\mathrm{smooth}} \lambda w$. By the triangle inequality, $\|N - \widehat{M}\|_1 \le C_{\mathrm{smooth}} \lambda w + C_{\mathrm{win}}/w + \epsilon$ and $|\mathsf{val}_{\mathcal{G}} - \mathsf{val}_{\mathcal{H}}| \le C_{\mathrm{smooth}} \lambda w + \epsilon$.

Finally, we assumed $\alpha \mathsf{val}_{\mathcal{H}} \le \mathcal{A}(N) \le \mathsf{val}_{\mathcal{H}}$. By continuity of the snapshot algorithm $\mathcal{O}$, we have $|\mathcal{A}(\widehat{M}) - \mathcal{A}(N)| \le C_{\mathrm{smooth}} \lambda w + C_{\mathrm{win}}/w + \epsilon$. Re-parametrizing ϵ finishes the proof. □

Theorem I.3 follows from Theorem IV.2 by instantiating the latter with the specific continuous snapshot algorithm in Lemma III.19.

REFERENCES

[1] D. Kogan and R. Krauthgamer, "Sketching cuts in graphs and hypergraphs," in *Proceedings of the 6th Annual Conference on Innovations in Theoretical Computer Science (ITCS 2015, Rehovot, Israel, January 11-13, 2015)*. Association for Computing Machinery, 2015, pp. 367–376.

[2] M. Kapralov, S. Khanna, and M. Sudan, "Approximating matching size from random streams," in *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014, Portland, OR, USA, January 5-7, 2014)*. USA: Society for Industrial and Applied Mathematics, Jan. 2014, pp. 734–751.

[3] V. Guruswami, A. Velingker, and S. Velusamy, "Streaming Complexity of Approximating Max 2CSP and Max Acyclic Subgraph," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX 2017, Berkeley, CA, USA, August 16-18, 2017)*, ser. LIPIcs, K. Jansen, J. D. P. Rolim, D. Williamson, and S. S. Vempala, Eds., vol. 81. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, Aug. 2017, pp. 8:1–8:19.

[4] M. Kapralov, S. Khanna, M. Sudan, and A. Velingker, "$(1 + \omega(1))$-approximation to MAX-CUT requires linear space," in *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017, Barcelona, Spain, January 16-19, 2017)*. Society for Industrial and Applied Mathematics, Jan. 2017, pp. 1703–1722.

[5] V. Guruswami and R. Tao, "Streaming Hardness of Unique Games," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX 2019, Cambridge, MA, USA, September 20-22, 2019)*, ser. LIPIcs, D. Achlioptas and L. A. Végh, Eds., vol. 145. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, Sep. 2019, pp. 5:1–5:12.

[6] M. Kapralov and D. Krachun, "An optimal space lower bound for approximating MAX-CUT," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC 2019, Phoenix, AZ, USA, June 23-26, 2019)*. Association for Computing Machinery, Jun. 2019, pp. 277–288.

[7] C.-N. Chou, A. Golovnev, and S. Velusamy, "Optimal Streaming Approximations for all Boolean Max-2CSPs and Max-kSAT," in *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS 2020, Virtual, November 16-19, 2020)*. IEEE Computer Society, Nov. 2020, pp. 330–341.

[8] C.-N. Chou, A. Golovnev, M. Sudan, and S. Velusamy, "Approximability of all finite CSPs with linear sketches," in *Proceedings of the 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2021, Denver, CO, USA, February 7-10, 2022)*. IEEE Computer Society, 2021.

[9] N. Singer, M. Sudan, and S. Velusamy, "Streaming approximation resistance of every ordering CSP," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX 2021, August 16-18, 2021)*, ser. LIPIcs, M. Wootters and L. Sanità, Eds., vol. 207. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, Jul. 2021, pp. 17:1–17:19.

[10] C.-N. Chou, A. Golovnev, M. Sudan, A. Velingker, and S. Velusamy, "Linear Space Streaming Lower Bounds for Approximating CSPs," in *Proceedings of the 54th Annual ACM Symposium on Theory of Computing*, 2022.

[11] J. Boyland, M. Hwang, T. Prasad, N. Singer, and S. Velusamy, "On sketching approximations for symmetric Boolean CSPs," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX 2022, Virtual, September 19-21, 2022)*, ser. LIPIcs, A. Chakrabarti and C. Swamy, Eds., vol. 245. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, Jul. 2022, pp. 38:1–38:23.

[12] C.-N. Chou, A. Golovnev, A. Shahrasbi, M. Sudan, and S. Velusamy, "Sketching Approximability of (Weak) Monarchy Predicates," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 2022.

[13] R. R. Saxena, N. Singer, M. Sudan, and S. Velusamy, "Streaming complexity of CSPs with randomly ordered constraints," in *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms*, 2023.

[14] N. Singer, "On streaming approximation algorithms for constraint satisfaction problems," Undergraduate Thesis, Harvard University, Cambridge, MA, Mar. 2022.

[15] M. Sudan, "Streaming and Sketching Complexity of CSPs: A survey (Invited Talk)," in *49th International Colloquium on Automata, Languages, and Programming*, ser. LIPIcs, M. Bojańczyk, E. Merelli, and D. P. Woodruff, Eds., vol. 229. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2022, pp. 5:1–5:20.

[16] U. Feige and S. Jozeph, "Oblivious Algorithms for the Maximum Directed Cut Problem," *Algorithmica*, vol. 71, no. 2, pp. 409–428, Feb. 2015.

[17] N. G. Singer, "Oblivious algorithms for the Max-kAND problem," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX 2023, Atlanta, GA, USA, September 11-13, 2023)*, ser. LIPIcs, N. Megow and A. D. Smith, Eds., vol. 275, May 2023.

[18] R. R. Saxena, N. G. Singer, M. Sudan, and S. Velusamy, "Improved streaming algorithms for Maximum Directed Cut via smoothed snapshots," May 2023.

[19] F. Chung and L. Lu, "Concentration Inequalities and Martingale Inequalities: A Survey," *Internet Mathematics*, vol. 3, no. 1, pp. 79–127, Jan. 2006.

[20] A. Joffe, "On a Set of Almost Deterministic k-Independent Random Variables," *The Annals of Probability*, 1974.