Quantum Space Advantage for Streaming Approximations

Ryan Anselm David Joy

December 14, 2025

1 Introduction

We present a unified survey of a line of works culminating in a provably exponential quantum space advantage in the streaming model for approximating the value of the maximum directed cut problem (MAX-DICUT).

This is quite a notable result in quantum computing because it is rare to have a quantum resource advantage that is simultaneously exponential, unconditionally provable, and for a problem of independent interest. For example, Simon's problem provides an unconditionally provable and exponential separation, but was artificially contrived for the sake of demonstrating the separation and is of no independent interest; Shor's factoring algorithm is believed to be exponential and is of independent interest, but it has not been rigorously proven that no efficient classical factoring algorithm exists; and Grover search is provable and of great independent interest, but only gives a quadratic speedup. In order to fully appreciate this result, however, an understanding of both classical and quantum streaming algorithm design, as well as streaming lower bound techniques, is required. This survey collects together a streamlined presentation of key results scattered across several papers, namely:

- A $\tilde{O}(\sqrt{n})$ space classical streaming algorithm for 0.483-approximating MAX-DICUT using an object known as the *snapshot*, developed in sections 2 and 3. [FJ10; Sax+23b]
- A $\Omega(\sqrt{n})$ classical space lower bound on approximating MAX-DICUT better than $\frac{4}{9} \approx 0.444$ in the streaming model, presented in section 4. [CGV21]
- Finally, a "quantization" of the classical streaming algorithm that achieves a space reduction to polylog(n) space when allowed to used quantum memory in additional to classical, presented in section 5. [KPV23; KPV24]

Definition 1.1 (Maximum Directed Cut)**.** *Let $G = (V, E)$ be a directed graph where by convention $n = |V|$ and $m = |E|$. Then*

$$\textit{MAX-DICUT}(G) = \max_{x \in \{0,1\}^n} |\{\vec{uv} \in E : x_u = 0, x_v = 1\}|.$$

MAX-DICUT is closely related to the more well-known Maximum Cut (MAX-CUT) problem, which is defined in the same way except that the graph G is undirected. Note that MAX-CUT can be reduced to MAX-DICUT by defining a directed graph with two edges $\vec{uv}$ and $\vec{vu}$ for every undirected edge uv in the MAX-CUT instance.

Streaming Problems. In the streaming model, the input to an algorithm is larger than the memory the algorithm is allowed to store. To even be able to take in this enormous input, the input is progressively fed to the algorithm one piece at a time in a stream. Algorithms which work in the non-streaming setting may no longer work because they require too much space or need to query arbitrary parts of the input on-demand. This means that novel algorithmic techniques are needed for streaming problems that may be well-studied in non-streaming settings such as Max-Cut and Max-DiCut. Simultaneously, imposing limitations on the memory sublinear in the size of the input opens the avenue for unconditionally proving impossibility results and lower bounds in the streaming setting via information-theoretic limitations.

The input model we use is the standard *insertion-only, (non-adaptive) adversarial-order, single-pass* graph streaming model. This is the model where an exponential quantum space advantage is achieved for Max-DiCut. Specifically, suppose that for directed unweighted graph G, its edges arrive one at a time to the algorithm as ordered pairs $(u, v)_i \in E$ for $1 \leq i \leq m$ where $u, v \in V$. At the end, the algorithm achieves an α-approximation for Max-DiCut(G) for some $0 < \alpha < 1$ if it outputs an x such that $\alpha \cdot$ Max-DiCut$(G) \leq x \leq$ Max-DiCut(G) with high probability. Each edge only arrives once, and can arrive in an arbitrary order that is fixed before the algorithm starts running. Note that the goal is thus to approximate the *value* of the maximum cut; outputting the cut itself is not required.

2 Oblivious Techniques for Maximum Directed Cut

2.1 Background and Definitions

The trivial algorithm of ignoring the stream and randomly assigning each vertex uses constant space and achieves an approximation ratio of $\frac{1}{4}$ (since edges are directed, the probability of cutting an edge is $\frac{1}{4}$, not $\frac{1}{2}$). It is not immediately obvious how space insufficient to store the graph could improve this: we need to find features in the graph that are useful but small. In this section, we will lay the groundwork through an analysis of oblivious algorithms.

Definition 2.1 (Bias)**.** *The bias of a vertex is its out-degree divided by its degree.*

The out-degree of a vertex v is the number of $\overrightarrow{uv}$ edges (edges to v, as opposed to $\overrightarrow{vu}$ edges). Thus a "giver" vertex with all incident edges pointing away from it has bias 1, while a "taker" vertex with all incident edges pointing into it has bias 0. Note that vertices of degree 0 have no bearing on a cut problem, so we can assume they do not exist.

This definition of bias comes from [FJ10]. Other authors such as [Sax+23b] define bias as $\frac{\text{out}-\text{in}}{\text{degree}}$, giving it a range of $[-1, 1]$. This choice does not affect the algorithms or complexity results, as the two definitions can easily be converted, but can be annoying when reading the literature.

Definition 2.2 (Oblivious Algorithm)**.** *An oblivious algorithm defined by a function $f : [0, 1] \to [0, 1]$. It independently assigns each vertex $x_v = 0$ with probability $f(bias(v))$ (and $x_v = 1$ otherwise).*

A key insight is that this is a local, parallel operation: Processing a vertex only requires knowledge of the directions of incident edges; this is oblivious of the graph's overall structure or the assignment of other vertices. The trivial algorithm described at the start of the section is oblivious with $f(b) = \frac{1}{2}$.

2.2 A .375 approximation

We will now show the .375 approximation from Feige and Jozeph [FJ10]. They gave a proof based on modifying and reflecting the graph, but we have written an LP-based proof, as this technique will be needed in the next section. Consider the "three-bucket" function

$$f_3(b) = \begin{cases} 0 & b \in [0, \frac{1}{3}) \\ \frac{1}{2} & b \in [\frac{1}{3}, \frac{2}{3}] \\ 1 & b \in (\frac{2}{3}, 1] \end{cases}$$

This function makes sense: the higher the bias, the more edges point away from the vertex, and thus the more edges that require $x_v = 0$ to cut. We will prove it beats the trivial algorithm.

Theorem 2.3. *An oblivious algorithm using f_3 achieves an approximation ratio of $\frac{3}{8} = .375$.*

Proof: Split the vertices into six sets: S_1, S_2, and S_3 are vertices for which OPT assigns $x_v = 0$ and that are in the first, second, and third buckets of f_3, respectively. S_4, S_5, S_6 are the corresponding vertices that OPT assigns $x_v = 1$. Define

$$x_{ij} = \frac{1}{\textsc{Max-DiCut}(G)} |\overrightarrow{uv} \in E : u \in S_i \wedge v \in S_j|$$

Lastly, let ℓ_i and u_i denote the lower and upper bounds on the bias of vertices in S_i; let f_i denote the value of $f(b)$ on these biases. This gives the following LP:

$$\text{minimize} \sum_{i\in[6]} \sum_{j\in[6]} f_i(1-f_j)x_{ij} \qquad \text{subject to}$$

$$\sum_{i\in[3]} \sum_{j\in[4,6]} x_{ij} = 1 \tag{1}$$

$$\ell_i \sum_{j\in[6]} x_{ij} + x_{ji} \le \sum_{j\in[6]} x_{ij} \qquad \forall i \in [6] \tag{2}$$

$$u_i \sum_{j\in[6]} x_{ij} + x_{ji} \ge \sum_{j\in[6]} x_{ij} \qquad \forall i \in [6] \tag{3}$$

$$x_{ij} \ge 0 \qquad \forall i \in [6], j \in [6] \tag{4}$$

(1) ensures the variables are scaled by $\frac{1}{\textsc{Max-DiCut}(G)}$; this makes the objective function be the expected value of the oblivious algorithm's solution divided by $\textsc{Max-DiCut}(G)$, which is its approximation ratio. (2) and (3) ensure that the average bias in each S_i is within the bounds. Note, however, that this is merely a guarantee on the average bias, and that it allows equality on all bounds. This means that while every graph can be written as an LP solution, not every LP solution has a corresponding graph. However, this just means that the optimal LP value is a lower bound on the true approximation ratio, exactly the type of guarantee we want.

Solving this LP yields an optimal value of .375, proving that for every graph, expected value of the algorithm's solution is at least .375OPT.

2.3 A .483 approximation

The LP-based analysis from the previous section works to analyze any oblivious algorithm based on a piecewise constant function: If there are k cases, the analysis will have $2k$ sets, giving an LP with $4k^2$ variables and $4k^2 + 2k + 1$ constraints, which can be efficiently solved. Using an LP solver, this gives a way rapidly evaluate many such algorithms mechanically instead of having to come up with proofs.

Feige and Jozeph came up with a piecewise constant function with 102 cases: the function they use, which we shall term f_{102}, is an approximation to

$$g(b) = \begin{cases} 0 & b \in [0, \frac{1}{4}) \\ 2b - \dfrac{1}{2} & b \in [\frac{1}{4}, \frac{3}{4}] \\ 1 & b \in (\frac{3}{4}, 1] \end{cases}$$

where the middle case is approximated using 100 equal-sized intervals. Using the LP-based proof creates an enormous 41616-variable LP. Nevertheless, this can be solved with the aid of a computer, yielding an optimal value above .483, proving that f_{102} gives a .483-approximation [FJ10].

It is known that .5 is an upper bound for oblivious algorithms [FJ10], so this is a quite good result. To see this bound, consider a $2n$-length cycle: The optimal cut has value n, but every vertex has an identical bias of .5, so an oblivious algorithm's expected value is $2nf(.5)(1 - f(.5)) \le .5n$.

3 Lifting Oblivious Algorithms to Streaming Algorithms

Recall that in the streaming setting, we no longer are interested in producing an assignment of vertices, but instead wish to merely estimate the *value* of the maximum cut. A straightforward approach would be to keep track of the in-degree and out-degree of each vertex during the stream, then calculate biases and use the oblivious algorithm. However, this requires linear space, and we seek $o(n)$-space streaming algorithms. The primary question in creating a streaming algorithm out of an oblivious algorithm is thus ”how much information can we lose while maintaining the performance guarantee from the oblivious algorithm?” The following definitions and theorem answer this question.

Definition 3.1 (Zeroth-Order Snapshot)**.** *A zeroth order snapshot is parameterized by* $\ell + 1$ *thresholds* $0 = t_0 < t_1 ... < t_\ell = 1$ *and consists of bias classes* $V_1 \cup V_2 ... V_\ell = V$ *where* V_i *contains vertices with biases between* t_{i-1} *and* t_i*.*

In order to use the oblivious algorithm from the previous section, we take $\ell = 102$ and set the thresholds to match the cases of f_{102}. Thus, $f_{102}(v)$ is constant for all all vertices in each V_i; we will denote this value as $f_{102}(V_i)$.

Definition 3.2 (First-Order Snapshot)**.** *A first order snapshot is parameterized by the same thresholds and consists of an* $\ell \times \ell$ *matrix* M_G *where*

$$(M_G)_{i,j} = |\overrightarrow{uv} \in E : u \in F_i \wedge v \in F_j|$$

A first-order snapshot is a count of how many edges go between each pair of bias classes. There is no clear way to calculate this exactly in sublinear space, but the following theorem shows that there is value in approximating it:

Theorem 3.3. *If* N *approximates* M_G *such that* $|N_{i,j} - (M_G)_{i,j}| \leq \epsilon m$ *for every entry, then*

$$(.483 - \epsilon)\textsc{Max-DiCut}(G) \leq \sum_{i,j \in \ell} f_{102}(V_i)(1 - f_{102}(V_j))N_{i,j} - \frac{\epsilon m}{8} \leq \textsc{Max-DiCut}(G)$$

Note that although N has 102^2 entries, this is a constant independent of the graph size; the space complexity will come from our method to create N. For simplicity, we have slightly weakened this theorem from Corollary 3.1 in [Sax+23a]. The proof of this theorem is lengthy and can be found in the reference. The takeaway is that as long as we can estimate the first-order snapshot to additive error, we will get an approximation ratio close to that of f_{102}. The loss in the approximation ratio is bounded by the error in approximating the first-order snapshot, giving us a clear goal to design an algorithm around. This has essentially reduced the maximum-cut problem to one of edge-counting.

There were many intermediate steps in creating a full $O(\sqrt{n})$ space streaming algorithm, which we will recap using three theorems. Each theorem is the results of significant technical work in [Sax+23a] and [Sax+23b], so we can merely provide some intuition behind each one and direct the interested reader to read the references in full.

Theorem 3.4. *If the stream were randomly (instead of adversarially) ordered, there exists an* $O(\log n)$*-space algorithm that creates the* S *required by* ***Theorem 3.3*** *with probability at least* $\frac{2}{3}$*.*

The idea here, detailed in Algorithm 2 of [Sax+23a], is to store the first k (which is a constant) edges appearing in the stream, yielding a subset of $\leq 2k$ vertices. Since this is a constant, these vertices' biases may be calculated exactly in $O(\log n)$ space. Lemma 3.2 of the paper then proves that for large enough k (the exact value depends on ϵ), this provides enough information to estimate the first-order snapshot for the entire graph.

This critically depends on the randomness restriction: The proofs utilize the first k edges of the stream being a random sample of the graph, which is not true in the adversarial setting.

Theorem 3.5. *If we were allowed two passes (instead of one) there exists an $O(\log n)$-space algorithm that creates the S required by* ***Theorem 3.3*** *with probability at least $\frac{2}{3}$*

This follows from the algorithm in the previous theorem: Given two passes, the first pass may be used to randomly sample edges. The second pass runs the algorithm using these random edges. Thus, a second pass seems to give similar power to changing an adversarial ordering to random.

Theorem 3.6. *If the degree of every vertex in G is at most d, a $O\left(d^{\frac{3}{2}}\sqrt{n}\log^2 n\right) = \tilde{O}_d(\sqrt{n})$-space algorithm that creates the S required by* ***Theorem 3.3*** *with probability at least $\frac{2}{3}$*

The key insight here is that the maximum number of edges is bounded, and that certain techniques are unlocked if the number of edges is (approximately) known in advance. Algorithm 3 of [Sax+23a] runs many $\tilde{O}(\sqrt{n})$ estimators in parallel, each with a different guess of the number of edges, then looks at the closest one once the stream ends. The workhorse is Lemma 3.3, which states that the required error bound will be met so long as the true number of edges is within a factor of 2 of the estimator's guess. This allows a logarithmic number of estimators.

Due to the $d^{\frac{3}{2}}$ dependence, this is only helpful in the case where d is small. For $d = n$, this algorithm takes quadratic space, worse than the straightforward "calculate all biases exactly" algorithm!

Each of these three theorems, and the corresponding algorithms, appeared in [Sax+23a]. Thus, the state of the art after this paper was that a .483 approximation algorithm for Max-DiCut was known in the streaming model, but only if the graph had bounded degree. This degree bound was soon removed in the authors' follow-up work [Sax+23b]. This work is even more technical and involved than the last. The main intuition is to use higher-order snapshots, plus many technical tricks and workarounds, to combat the worst adversarial cases stemming from unbounded degree. This yields the following theorem as the conclusion of this section.

Theorem 3.7. *There exists a $\tilde{O}_d(\sqrt{n})$-space classical algorithm that .483-approximates Max-DiCut in the streaming model with probability at least $\frac{2}{3}$*

4 Streaming Approximation Lower Bound

Unconditional lower bounds in the standard input model of computation are usually hard to come by. In the *streaming model*, however, lower bounds on space requirements have successfully been shown unconditionally. We present one such streaming lower bound on space due to [CGV21] which states that, classically, any streaming algorithm that achieves better than a $\frac{4}{9} \approx 0.444$ approximation ratio for Max-DiCut requires $\Omega(\sqrt{n})$ memory. This result serves to juxtapose with the following section, which presents a *quantum* streaming algorithm for Max-DiCut which achieves an approximation ratio of $0.483 > 0.444$, while using only polylog(n) qubits of quantum memory. The combination of proven classical hardness and proven quantum easiness is what makes a separation "provable".

Max-DiCut belongs to a class of problems known as maximum constraint satisfaction problems (Max-CSPs), and more specifically, it is a type of Max-2CSP, meaning that each constraint clause takes in 2 variables as input. Clauses in Max-DiCut take on the form $(x_i \wedge \neg x_j)$ for some choices of $i \neq j \in [n]$ where $x_i, x_j \in \{0, 1\}$ are the labels of the tail and head vertices, respectively, of an edge in a directed graph G. Max-DiCut is a special case of Max-2EAND, which is a Max-2CSP such that all its clauses are of the form $(c_i \wedge c_j)$ where $c_i \in \{x_i, \neg x_i\}$, $c_j \in \{x_j, \neg x_j\}$ for some $i \neq j$. We will show a reduction from a one-way communication problem with a known communication lower bound called **Distributional Boolean Hidden Partition (DBHP)** to Max-2EAND in the streaming setting. This reduction implies that Max-2EAND, and also Max-DiCut, have a streaming space lower bound for achieving a $4/9$-approximation.

Definition 4.1 (DBHP)**.** *Let $n \in \mathbb{N}, \beta \in (0, 1/16)$ be parameters. $G \sim G(n, 2\beta/n)$ is an Erdős-Rényi random graph (with n vertices and where each potential edge is independently included w.p. $2\beta/n$). Let $X^* \in \{0,1\}^n$ be a n-bit vector sampled uniformly at random. Let r be the number of edges in G, and $M \in \{0,1\}^{r \times n}$ be the edge-vertex incidence matrix of G. The two players, Alice and Bob, are given the following inputs:*

- *Alice receives the randomly chosen vector $X^* \in \{0,1\}^n$.*
- *Bob receives the edge-vertex incidence matrix M and a vector $w \in \{0,1\}^r$, and is promised that w is sampled from one of two possible distributions with equal probability. Either $w = MX^*$ (YES distribution) or $w = \mathbf{1} + MX^*$ (NO distribution) where $\mathbf{1} \in \mathbb{F}_2^r$ is the all-ones vector.*

Alice is allowed to send a message m to Bob, where the complexity of the communication protocol is the length of the message, $|m|$. The goal of the players is for Bob to distinguish between the YES and NO distributions. The success probability is formally defined as

$$\Pr_{w \sim \textit{YES}}[\textit{Bob says YES}]/2 + \Pr_{w \sim \textit{NO}}[\textit{Bob says NO}]/2.$$

There is a one-way communication complexity lower bound for the DBHP problem which we will state without proof.

Lemma 4.2 (Lemma 4.4 of [CGV21], based on Lemma 5.1 of [KKS14])**.** *Let $\beta \in (n^{-1/10}, 1/16)$ and $s \in (n^{-1/10}, 1)$ be parameters. Any protocol for DBHP that has $|m| \le s\sqrt{n}$ bits of communication cannot distinguish between the YES and NO distributions with success probability more than $1/2 + c \cdot (\beta^{3/2} + s)$ for some constant $c > 0$ and sufficiently large n.*

Reduction from DBHP to Max-2EAND. The graph G in the definition of DBHP will be very sparse, so to make it more amenable to performing a reduction we first densify it by a factor T. Alice still samples a single $X^* \in \{0,1\}^n$, but Bob is now given T instances of (M_i, w_i) for $1 \le i \le T$, each sampled i.i.d. from either the YES or NO distributions (all T pairs are sampled from the same distribution). The goal is the same, for Bob to distinguish between the two cases, using some communication bits from Alice.

The reduction is defined by two algorithms, $\mathcal{A}$ and $\mathcal{B}$, to be specified. Alice runs $\mathcal{A}$ on X^* and outputs a set of Max-2EAND clauses. Bob runs $\mathcal{B}$ on each of his (M_i, w_i)s and outputs T sets of Max-2EAND clauses. The overall Max-2EAND instance is the union of all the clauses, $\mathcal{A}(X^*) \cup \mathcal{B}(M_1, w_1) \cup \cdots \cup \mathcal{B}(M_T, w_T)$. This reduction from $\mathcal{A}$ and $\mathcal{B}$ induces two different distributions of MAX-2EAND instances, $\mathcal{D}^Y$ and $\mathcal{D}^N$, corresponding to the underlying YES and NO distributions respectively. For some values v^Y and v^N, if

$$\Pr_{\Psi \sim \mathcal{D}^Y}[\textsc{Max-2EAND}(\Psi) \ge v^Y] = 1 - o(1) \text{ and } \Pr_{\Psi \sim \mathcal{D}^N}[\textsc{Max-2EAND}(\Psi) \le v^N] = 1 - o(1),$$

then if there were an α-approximation algorithm for Max-2EAND for $\alpha > \frac{v^N}{v^Y}$, then this approximation algorithm could be used to distinguish instances of Max-2EAND sampled from $\mathcal{D}^Y$ from instances of Max-2EAND sampled from $\mathcal{D}^N$, thus enabling a way to distinguish between the YES and NO distributions.

Remark 4.3. *A streaming algorithm for distinguishing $\mathcal{D}^Y$ and $\mathcal{D}^N$ that uses S bits of space can be turned into a one-way communication protocol for DBHP that uses S bits of communication.*

Under the adversarial-order streaming model, a streaming algorithm is promised to work on any ordering of stream elements. We will use this by having the stream start with the clauses in $\mathcal{A}(X^*)$. Alice could compute $\mathcal{A}(X^*)$ from her input X^* and simulate the streaming algorithm on it. Since the streaming algorithm uses only S bits of space, Alice will have an S-bit memory state that she can send to Bob as the message m. Bob then continues the simulation of the streaming algorithm on the clauses $\mathcal{B}(M_1, w_1) \cup \cdots \cup \mathcal{B}(M_T, w_T)$ he computes based on his input, using the S-bit memory state received from Alice as the starting memory. At the end, since the streaming algorithm distinguishes between $\mathcal{D}^Y$ and $\mathcal{D}^N$, then Bob is able to distinguish between YES and NO.

Here are the definitions of algorithms $\mathcal{A}$ and $\mathcal{B}$. Let $c > 0$ be the constant from Lemma 4.2. For $\varepsilon > 0$, let $T = (10000/\varepsilon^2)^3 \cdot (10c)^2$ and $\beta = \frac{1}{(10cT)^{2/3}}$, so that $\beta T = 10000/\varepsilon^2$.

- $\mathcal{A}(X^*)$: Sample $\beta nT/4$ independent pairs $(i, j) \in X^* \times \overline{X^*}$, and for each pair include the clause $(x_i \wedge \neg x_j)$ in the set of clauses $\mathcal{A}(X^*)$.
- For each $\mathcal{B}(M_i, w_i)$: Let r be the number of rows in M_i. For each $1 \le l \le r$ such that $(w_i)_l = 1$, let the two 1s in the lth row of M_i be in the jth and kth positions. Include the clauses $(x_j \wedge \neg x_k)$ and $(x_k \wedge \neg x_j)$ in the set of clauses $\mathcal{B}(M_i, w_i)$.

Lemma 4.4. *For any $\varepsilon \in (0, 1)$, let $\mathcal{D}^Y$ and $\mathcal{D}^N$ be defined in terms of the $\beta, T, \mathcal{A}, \mathcal{B}$ as above. For a* Max-2EAND *instance Ψ, let m_Ψ denote the number of clauses in Ψ. Then*

$$\Pr_{\Psi \sim \mathcal{D}^Y}[\textsc{Max-2EAND}(\Psi) \ge (3/5 - \varepsilon) \cdot m_\Psi] = 1 - o(1)$$

and

$$\Pr_{\Psi \sim \mathcal{D}^N}[\textsc{Max-2EAND}(\Psi) \le (4/15 + \varepsilon) \cdot m_\Psi] = 1 - o(1).$$

A direct consequence of this lemma is that $v^N = (4/15 + \varepsilon) \cdot m_\Psi$ and $v^Y = (3/5 - \varepsilon) \cdot m_\Psi$, so α-approximating Max-2EAND for any $\alpha > \frac{4/15+\varepsilon}{3/5-\varepsilon} > 4/9$ implies the ability to distinguish YES and NO.

Proof Sketch.

- Suppose we are in the YES distribution/$\mathcal{D}^Y$. It suffices to show the existence of an assignment which will w.h.p. satisfy a $3/5$ fraction of clauses.

  Consider the assignment $\sigma = X^* \in \{0, 1\}^n$ to the variables of Ψ, exactly using Alice's input X^*. Every single clause $(x_i \wedge \neg x_j)$ generated by $\mathcal{A}(X^*)$ will be satisfied by σ because $(i, j) \in X^* \times \overline{X^*}$ definitionally. This contributes $\beta nT/4$ satisfied clauses. Furthermore, for every pair of clauses $(x_j \wedge \neg x_k)$ and $(x_k \wedge \neg x_j)$ generated by $\mathcal{B}$, exactly one of them will be satisfied by σ because of the way these clauses are constructed for each entry of w that equals 1, which occurs when $w_l = (MX^*)_l = (X^*)_j + (X^*)_k = 1$ when j and k are defined as the positions of the two 1s in the lth row of M.

  The amount contributed by $\mathcal{B}$ turns out to be the number of entries over all w_i equal to 1, which will turn out to be closely concentrated around a value of $\frac{\beta nT}{2}$. Therefore, with high probability, σ satisfies around $\frac{\beta nT}{4} + \frac{\beta nT}{2} = \frac{3\beta nT}{4}$ clauses out of a total of around $\frac{\beta nT}{4} + \beta nT = \frac{5\beta nT}{4}$ clauses (both closely concentrated around these values). Hence, w.h.p. the fraction of clauses satisfied by $\sigma = X^*$ will be greater than $(3/5 - \varepsilon)$.

- Suppose we are in the NO distribution/$\mathcal{D}^N$. In this case we have to show that w.h.p. *none* of the assignments satisfy more than a $4/15$ fraction of clauses.

  Let $X^* \in \{0, 1\}^n$ define a hidden partition of the vertices. The key intuition is that in the NO/$\mathcal{D}^N$ case, two clauses are added by $\mathcal{B}$ iff $(w_i)_\ell = 1$, which occurs only if $(X^*)_j = (X^*)_k$, i.e. when x_j and x_k are on the same side of the hidden partition. The clauses are of the form $(x_j \wedge \neg x_k)$ and $(x_k \wedge \neg x_j)$.

Thus clauses from $\mathcal{B}$ reward assignments $\sigma \in \{0,1\}^n$ such that $\sigma_j \neq \sigma_k$ whenever $(X^*)_j = (X^*)_k$, i.e. maximizing cutting edges that don't cross the hidden partition. Meanwhile, clauses from $\mathcal{A}$ only rewards assignments σ that assign many vertices in X^* to 1 and many vertices in $\overline{X^*}$ to 0 (so as to satisfy many sampled $(x_i \wedge \neg x_j)$ with $(i,j) \in X^* \times \overline{X^*}$), i.e. maximizing cutting edges that cross the hidden partition.

Another way this can be viewed is that $\mathcal{B}$-clauses incentive σ being different from X^*, while $\mathcal{A}$-clauses incentive σ being similar to X^*. These two objectives compete with one another. One can make this quantitative by parameterizing σ by its overlaps

$$p := \Pr_{i \in X^*}[\sigma_i = 1], \qquad q := \Pr_{j \in \overline{X^*}}[\sigma_j = 0].$$

Then, it is possible to show that the expected contribution of $\mathcal{A}$ scales like $pq \cdot (\beta n T/4)$, while the expected number of marked edges cut *within* the sides of the hidden partition scales like

$$\big(p(1-p) + q(1-q)\big) \cdot (\beta n T/2),$$

because inside X^* you cut a random within-edge with probability $\approx 2p(1-p)$, and similarly inside $\overline{X^*}$. Since the number of clauses still is around $(5/4)\beta n T$, this yields a rough upper bound on the fraction of satisfied clauses of the form

$$\frac{pq + 2p(1-p) + 2q(1-q)}{5}.$$

Maximizing over $p, q \in [0,1]$ shows this expression is maximized at $p = q = 2/3$ with value $4/15$. Hence, w.h.p. over $\Psi \sim \mathcal{D}^N$, no assignment achieves more than a $(4/15 + \varepsilon)$ fraction of clauses.

5 Construction of a Quantum Streaming Algorithm

Theorem 5.1 ([KPV23], Theorem 1)**.** *There is a quantum streaming algorithm which* 0.483*-approximates the* Max-DiCut *value of an input graph G with probability* $1 - \delta$*. The algorithm uses* $O(\log^5 n \log \frac{1}{\delta})$ *qubits of space.*

The quantum streaming algorithm of [KPV23] relies on the same fundamental idea as that of the classical streaming algorithm of [Sax+23b]: they approximate the maximum directed cut by estimating the snapshot of the graph. The novel contribution that they provide is a way of estimating the snapshot (or rather, a suitable approximation to it called a pseudosnapshot) in only polylog(n) quantum space compared to the $\tilde{O}(\sqrt{n})$ classical space of the original using a primitive called the *quantum pair sketch*.

We break the construction of an algorithm for approximating Max-DiCut into several stages which build on top of one another. First, we describe the quantum pair sketch, an inherently quantum primitive. We give an example application of the base quantum pair sketch for the Boolean Hidden Matching problem. We then construct a new primitive called a heavy edge counter from the quantum pair sketch interface. Following this, we give an idea of how to use the heavy edge counter to estimate the pseudosnapshot of a graph stream.

5.1 Quantum Pair Sketch

We briefly describe the functionality of the quantum pair sketch, treating it as a black box which can be used without knowing its implementation. The implementation details of the sketch require some knowledge of quantum computing, which we give only a high-level picture of for the sake of flow and keeping prerequisites minimal. A more formal exposition on the implementation can be found in section 3 of [KPV24].

The quantum pair sketch $\mathcal{Q}_T$ summarizes a set $T \subseteq \{0,1\}^l$ of up to $L = 2^l$ elements using $l = \log L$ qubits of quantum memory. The benefit of an exponential reduction in space requirements for quantumly storing the set T compared to the naive classical approach comes at the cost of it being *probabilistically* queried. It has the following operations:

- **create**(T): instantiates the sketch $\mathcal{Q}_T$ from a set $T \subseteq \{0,1\}^l$.
- **update**($\pi, \mathcal{Q}_T$): For an input permutation $\pi : \{0,1\}^l \to \{0,1\}^l$, transforms $\mathcal{Q}_T \mapsto \mathcal{Q}_{\pi(T)}$ where $\pi(T) = \{\pi(x) : x \in T\}$.
- **query-one**($x, \mathcal{Q}_T$): probabilistically checks whether $x \in T$ or not by returning 1 or 0 respectively.
  If $x \in T$, then
  - w.p. $1/|T|$ returns 1 and destroys the sketch (meaning the sketch cannot be used anymore after this).
  - w.p. $1 - 1/|T|$ returns 0 and replaces $\mathcal{Q}_T$ with $\mathcal{Q}_{T\setminus\{x\}}$.

  If $x \notin T$, always returns 0 ($\mathcal{Q}_T$ is unaltered).
  Note: Getting 1 implies that $x \in T$, while getting 0 could mean x was or was not *originally* in T, but going forward T has been updated so that x is for certain not there.
- **query-pair**($x, y, \mathcal{Q}_T$): probabilistically checks whether $\{x, y\} \subseteq T$. Has possible return values of $+1$, -1, and 0.
  If $\{x, y\} \subseteq T$, then
  - w.p. $2/|T|$ returns $+1$ and destroys the sketch.
  - w.p. $1 - 2/|T|$ returns 0 and replaces $\mathcal{Q}_T$ with $\mathcal{Q}_{T\setminus\{x,y\}}$

  If $x \in T$ or $y \in T$ but NOT both ($|\{x, y\} \cap T| = 1$), then
  - w.p. $1/|T|$ returns $+1$ or -1 at random with equal probability (each has probability $1/(2|T|)$) and destroys the sketch.
  - w.p. $1 - 1/|T|$ returns 0 and replaces $\mathcal{Q}_T$ with $\mathcal{Q}_{T\setminus\{x,y\}}$

  If $x \notin T$ and $y \notin T$, always returns 0 ($\mathcal{Q}_T$ is unaltered).
  Note: When $\{x, y\} \subseteq T$, $\mathbb{E}[\text{query-pair}(\{x, y\}, \mathcal{Q}_T)] = 2/|T|$. In the other cases, when $\{x, y\} \not\subseteq T$, $\mathbb{E}[\text{query-pair}(\{x, y\}, \mathcal{Q}_T)] = 0$. This provides a means of querying the presence of a pair by maintaining k copies of the sketch in parallel for sufficiently large k and looking at the average of the outcomes.

Many features of the quantum pair sketch are forced by the underlying quantum mechanical implementation. For instance, the **update** operation comes in the form of a permutation because all non-measurement operations on the underlying quantum state must be *reversible*, in the sense that knowing the operation and the output, there is a uniquely specified input. The property of destroying the sketch or removing the element we wanted to check for most of the time after a query is inconvenient, but is a necessary consequence of performing the underlying quantum measurement, which according to the laws of quantum mechanics collapses the original state storing a (potentially exponentially large) set to a state storing a smaller set that is consistent with the outcome observed.

The utility of "quantumness" over classical randomness in this sketch arises in the **query-pair** operation in the case where $\{x, y\} \subseteq T$. In that case, the -1 outcome never occurs due to destructive interference, while the $+1$ outcome occurs with higher probability due to constructive interference. This feature is the *key advantage* quantum pair sketch has over a randomized classical sketch, and we will see how it used to obtain an exponential quantum space advantage in the next section.

5.2 Toy Problem: Boolean Hidden Matching

To build intuition for the quantum pair sketch, we first apply it to the Boolean Hidden Matching problem. This example isolates the core principles that yield an exponential separation in space complexity between quantum and classical methods. The Boolean Hidden Matching streaming problem is defined as follows: For a set of n vertices $v \in [n]$, each has a label $x_v \in \{0, 1\}$. There is also a partial matching $M \subseteq [n] \times [n]$ where $|M| = \alpha n$ for $0 < \alpha \leq 1/2$ on the n vertices such that each vertex appears in at most one pair $(u, v) \in M$ and pairs are between distinct vertices $u \neq v$. Each pair $(u, v) \in M$ has a label $z_{uv} \in \{0, 1\}$. It is promised that one of two conditions holds:

1. For every $(u, v) \in M$, $x_u \oplus x_v = z_{uv}$, or
2. For every $(u, v) \in M$, $x_u \oplus x_v \neq z_{uv}$.

The algorithm receives a stream of vertex-label pairs (v, x_v) and matching-label pairs $((u, v), z_{uv})$, where the two types of updates are intermingled in any order. By the end of the stream, all vertices should have appeared in the stream alongside their x label once, and all vertex pairs in M should have appeared in the stream alongside their z label once. The goal is to decide by the end of the stream which condition is true.

Classical Streaming Hardness. Without remembering at least one complete set of x_u, x_v, and z_{uv}, a classical algorithm can do no better than succeeding $1/2$ of the time by guessing the correct condition randomly. The optimal classical strategy for the Boolean Hidden Matching streaming problem is essentially to save some random choice of x or z labels and hope to "get lucky" in seeing the corresponding z or x labels, respectively, later on in the stream to obtain at least one complete set of x_u, x_v, and z_{uv}. However, in a stream of $\Theta(n)$ updates, you would need to save a non-trivial amount of labels to have a high probability of getting lucky. Specifically, the birthday paradox phenomenon tells us that you would need to save $\Omega(\sqrt{n})$ items to have a constant probability of seeing at least one complete set of x_u, x_v, and z_{uv} labels. Indeed, this problem is classically hard to decide with high probability in the streaming model unless the algorithm is allowed to use $\Omega(\sqrt{n})$ memory. The full proof of this uses Fourier-analytic techniques to show the lower bound rigorously.

Quantum Streaming Algorithm. First, the algorithm initializes the quantum pair sketch with a set

$$T = \{(u, 0) \mid u \in [n]\}$$

by calling **create**(T). Each element of T can be written with $l = \lceil \log n \rceil + 1$ bits, so $l = \lceil \log n \rceil + 1$ qubits are required for the sketch. Each element of T has two labels: the first corresponds to a choice of vertex u and the second label to a vertex label x_u which is initially set to 0, but will be updated as the algorithm receives elements from the stream. For the two different kinds of stream updates, here is how the algorithm responds:

- Updates of the form (v, x_v): Let v' be the vertex seen in this update and $x_{v'}$ be the label seen. If $x_{v'} = 0$, do nothing. If $x_{v'} = 1$, then consider the permutation $\pi_{v'}$ which maps
  - $(v', 0) \mapsto (v', 1)$
  - $(v', 1) \mapsto (v', 0)$

  while mapping every other element to itself. The second mapping is necessary to preserve bijectivity of the permutation, though it is not actually used for anything. The algorithm calls **update**$(\pi_{v'}, \mathcal{Q}_T)$, which is used to update the label of v' in T from 0 to 1.

- Updates of the form $((u,v),z_{uv})$: When getting an update like this, the algorithm tries to recover the labels of u and v. It picks two bits $a,b \in \{0,1\}$ independently and uniformly at random as vertex label guesses and calls

  $$\textbf{query-pair}((u,a),(v,b),\mathcal{Q}_T).$$

  If this call returns $+1$, we classically save the values of a and b used in the call returning $+1$ (i.e. if **query-pair**$((u,0),(v,1),\mathcal{Q}_T)$ returns $+1$ then it saves 0 and 1) as well as the label z_{uv} revealed during the update. If the call returns -1, return $\perp$ and abort the algorithm. If it returns 0 then do nothing and continue with the next stream update. Note that when ± 1 is returned the sketch is destroyed so it is no longer useful beyond that point.

Let us analyze the $((u,v),z_{uv})$ update further. Whatever the current state of $\mathcal{Q}_T$ may be, exactly *one* of the four possible **query-pair** calls of the form **query-pair**$(x,y,\mathcal{Q}_T)$ will have $\{x,y\} \subseteq T$, *two* will have $|\{x,y\} \cap T| = 1$, and *one* will have $x,y \notin T$. For the $\{x,y\} \subseteq T$ query, $+1$ is returned with probability $2/|T|$, for the two $|\{x,y\} \cap T| = 1$ queries, $+1$ is returned with probability $1/(2|T|)$ each, and for the $x,y \notin T$ query, $+1$ is returned with probability 0.

What is the probability that a **query-pair** call which returned $+1$ had vertex label guesses a,b which are the same as the *current* labels of u and v stored in the quantum sketch ($x_{u,\text{curr}}, x_{v,\text{curr}}$), such that $a = x_{u,\text{curr}}$ and $b = x_{v,\text{curr}}$? Let c be the event that $a = x_{u,\text{curr}}$ and $b = x_{v,\text{curr}}$ for the a and b chosen, and let $+1$ be the event that the **query-pair** call using a and b as the vertex label guesses returned $+1$. By Bayes' rule

$$\Pr[c \mid +1] = \frac{\Pr[+1 \mid c\,] \cdot \Pr[c]}{\Pr[+1]} = \frac{\frac{2}{|T|} \cdot \frac{1}{4}}{\left(\frac{1}{4} \cdot \frac{2}{|T|} + \frac{1}{2} \cdot \frac{1}{2|T|}\right)} = 2/3.$$

Note that if either of the (u,x_u) or (v,x_v) updates did not arrive before the $((u,v),z_{uv})$ update, the *current* vertex labels $x_{u,\text{curr}}, x_{v,\text{curr}}$ may not reflect the *true* vertex labels x_u, x_v. However, correcting for this can be handled by classically updating the saved value of $x_{u,\text{curr}}$ or $x_{v,\text{curr}}$ if these updates are seen later in the stream. For example: suppose that after seeing update $((u,v),z_{uv}=1)$, getting a return value of $+1$, and saving $x_{u,\text{curr}} = 0, x_{v,\text{curr}} = 0, z_{uv} = 1$, an update $(v,1)$ arrives. The algorithm would then classically set $x_{v,\text{curr}} = 1$. If another update $(u,1)$ arrives, the algorithm would classically set $x_{u,\text{curr}} = 1$. Finally, it sets $x_u := x_{u,\text{curr}}$ and $x_v := x_{v,\text{curr}}$ at the end of the stream after all updates have been processed.

At the end of the stream, if values x_u, x_v, and z_{uv} have been classically saved, then we check whether $x_u \oplus x_v = z_{uv}$ or $x_u \oplus x_v \neq z_{uv}$ and report which one is true. If no values were saved, then return $\perp$. Recall that $|M| = \alpha n$. The probability of any given update of the form $((u,v),z_{uv})$ returning $+1$ is $\Theta(1/n)$, and there are $\Theta(\alpha n)$ many updates of this form, so with $\Theta(\alpha)$ probability the algorithm will report a non-$\perp$ answer, that will be correct $2/3$ of the time. By repeating this sketch $\Theta(\frac{1}{\alpha})$ times in parallel and taking a majority vote of the decision for the outputs that are not $\perp$, the algorithm will output the correct answer $\geq 2/3$ of the time. Each sketch requires $O(\log n)$ quantum space, so the overall quantum space complexity is $O(\frac{1}{\alpha}\log n)$, achieving an *exponential space advantage* against the $\Omega(\sqrt{n})$ classical lower bound.

5.3 Constructing a Heavy Edge Counter from Quantum Pair Sketch

We will construct a new primitive out of quantum pair sketch called a *heavy edge counter*. Suppose that for a directed graph $G = (V,E)$, the input arrives as a stream of directed edges $\vec{uv}$ in G. We show how to construct a quantum streaming algorithm that counts heavy edges in G. More specifically, let $d_v^{\leq \vec{uv}}$ be the number of edges incident to v that arrived before $\vec{uv}$ in a stream (and analogously for $d_u^{\leq \vec{uv}}$). For some $\varepsilon > 0$ and constants d_h, d_t, the goal of the Heavy Edges problem is to estimate from the graph stream the number of edges $\vec{uv} \in E$ within $\pm \varepsilon m$ of the true value such that

$$d_u^{\leq \vec{uv}} \geq d_h \text{ and } d_v^{\leq \vec{uv}} \geq d_t.$$

Quantum Streaming Algorithm. First, the algorithm initializes a quantum pair sketch with the set

$$T = \{(i, s) \mid i \in [4m]\}$$

where $\{s, h, t\}$ are labels indicating "scratch", "head", and "tail" respectively by calling **create**(T).

Permutation Construction. For the i-th directed edge $\vec{uv}$ that arrives in the stream, we utilize four unique scratch states from T, denoted $s^h_{i,u}, s^t_{i,u}, s^h_{i,v}, s^t_{i,v}$, which are equal to elements $(4i-3, s)$, $(4i-2, s)$, $(4i-1, s)$, $(4i, s)$ respectively. We construct the permutation $\pi_{i,uv}$ as a product of four disjoint cycles. For each $(w, q) \in \{(u, h), (u, t), (v, h), (v, t)\}$, the permutation cycles the scratch state through the counter states (w, j, q) for $j \in \{0, \ldots, d_q\}$:

$$\pi_{i,uv} : \quad s^q_{i,w} \;\mapsto\; (w, 0, q) \;\mapsto\; (w, 1, q) \;\mapsto\; \ldots \;\mapsto\; (w, d_q, q) \;\mapsto\; s^q_{i,w}.$$

This implements a modular increment of the counter for w with respect to the label q, using the scratch state to expand the cycle length. All other states remain fixed. This permutation is applied to the quantum pair sketch by calling **update**($\pi_{i,uv}, \mathcal{Q}_T$) when the ith directed edge $\vec{uv}$ arrives.

Immediately after the **update** call, the algorithm calls

$$\textbf{query-pair}((u, d_h, h), (v, d_t, t), \mathcal{Q}_T).$$

If this returns ± 1, terminate the algorithm and return $\pm 2m$, depending on which value of ± 1 was returned by **query-pair**. If it returns 0, then continue taking in stream updates, unless this is the last item, in which case the algorithm returns 0.

Analysis. Observe that (u, d_h, h) and (v, d_t, t) are present in the set T exactly when at least d_h and d_t edges adjacent to u and v, respectively, have arrived in the stream prior to $\vec{uv}$ by maintaining a "stack" of counters that accumulates up to d_h or d_t. Each time an edge adjacent to a vertex u arrives, it increments all the counters of the form (u, i, q) for $i \in \{0, \ldots, d_q\}$ and $q \in \{h, t\}$. If **query-pair** returns 0, then (u, d_h, h) and (v, d_t, t) are removed from T (or possibly they were never there to begin with). However, for the next edge that arrives incident on u or v, its **update** call will shift the stack back up again so that (u, d_h, h) or (v, d_t, t) is in T again since the elements $(u, d_h - 1, h)$ or $(v, d_t - 1, t)$ were present in the stack and upon the new **update** call get shifted back up.

If $d_u^{\leq \vec{uv}} \geq d_h$ and $d_v^{\leq \vec{uv}} \geq d_t$, then the expected value of each call to **query-pair** is $\frac{2}{|T|} = \frac{1}{2m}$, but otherwise the expected value of **query-pair** is 0. Using linearity of expectation, the final expected value of the algorithm is the sum of the expected contributions of each edge $\vec{uv}$ such that $d_u^{\leq \vec{uv}} \geq d_h$ and $d_v^{\leq \vec{uv}} \geq d_t$ (Note: the analysis is a little bit more complicated, as when elements are removed from T it boosts the probability of encountering the true element, but this increase cancels out exactly with the decrease in e.v. due to the probability of the quantum sketch being destroyed before it processes a stream update). Each edge meeting the criteria $d_u^{\leq \vec{uv}} \geq d_h$ and $d_v^{\leq \vec{uv}} \geq d_t$ adds $\frac{1}{2m}$ to the total expected value. By multiplying the algorithm's output by $2m$, we obtain a noisy estimator for the number of edges $\vec{uv}$ in the stream such that $d_u^{\leq \vec{uv}} \geq d_h$ and $d_v^{\leq \vec{uv}} \geq d_t$. We can obtain a $\pm \varepsilon m$ estimate of the quantity desired with high probability by running many parallel copies of quantum pair sketch and averaging their results to reduce the variance.

Space Complexity. How many parallel copies of quantum pair sketch need to be run to get an estimate within $\pm \varepsilon m$ with high probability? The output of one sketch takes on a value $X \in \{-2m, 0, 2m\}$, so $\text{Var}[X] \leq \mathbb{E}[X^2] \leq 4m^2 = O(m^2)$. By Chebyshev's inequality, to stay within $\pm \varepsilon m$ of the true value with probability $\geq 2/3$ requires using the average of $\Theta(\frac{1}{\varepsilon^2})$ copies of the quantum pair sketch. Therefore the overall quantum space complexity required for the Heavy Edges problem is $O(\frac{1}{\varepsilon^2} \log n)$.

5.4 Constructing a (pseudo)snapshot estimator from Heavy Edge Counter

We are finally in a position to give a high-level picture of how to quantumly estimate a *pseudosnapshot*, which can then be used to approximate $\textsc{Max-DiCut}(G)$ via the techniques developed in sections 2 and 3.

Degree Range Count Estimator. Suppose that instead of counting heavy edges, we wanted to count edges $\vec{uv}$ whose endpoint degrees were within specific *ranges*, or that $d_u^{\leq \vec{uv}} \in [d_h, d'_h)$ and $d_v^{\leq \vec{uv}} \in [d_t, d'_t)$. This can be achieved by maintaining four separate heavy edge counters that serve as estimators for the following values:

1. $A := |\{\vec{uv} \in E : d_u^{\leq \vec{uv}} \geq d_h$ and $d_v^{\leq \vec{uv}} \geq d_t\}|$
2. $B := |\{\vec{uv} \in E : d_u^{\leq \vec{uv}} \geq d'_h$ and $d_v^{\leq \vec{uv}} \geq d_t\}|$
3. $C := |\{\vec{uv} \in E : d_u^{\leq \vec{uv}} \geq d_h$ and $d_v^{\leq \vec{uv}} \geq d'_t\}|$
4. $D := |\{\vec{uv} \in E : d_u^{\leq \vec{uv}} \geq d'_h$ and $d_v^{\leq \vec{uv}} \geq d'_t\}|$

The true number of edges $\vec{uv}$ such that $d_u^{\leq \vec{uv}} \in [d_h, d'_h)$ and $d_v^{\leq \vec{uv}} \in [d_t, d'_t)$ will be $A - B - C + D$ by an inclusion-exclusion argument. This is because when we subtract B (edges with $d_u^{\leq \vec{uv}} \geq d'_h$) and C (edges with $d_v^{\leq \vec{uv}} \geq d'_t$) from A, the edges counted in D (those satisfying both $d_u^{\leq \vec{uv}} \geq d'_h$ and $d_v^{\leq \vec{uv}} \geq d'_t$) have been subtracted twice, so D must be added back to correct for the double subtraction.

Generalizing from degrees to biases. We would like to estimate the count of edges in a *bias* class $[b_h, b'_h) \times [b_t, b'_t)$ rather than a *degree* class $[d_h, d'_h) \times [d_t, d'_t)$ in order to actually estimate the snapshot. Recall that the bias of a vertex is defined as its outdegree divided by its degree. We could in principle make a small modification to the heavy edge counter to check for outdegree instead of degree: for an arriving edge $\vec{uv}$, have it only increment the two permutation cycles with elements of the form (u, i, h) and (u, i, t) instead of all four. However, we need to estimate the outdegree and degree *simultaneously* to estimate the bias.

A naive approach that explicitly maintains two independent counters per vertex (one for outdegree, one for degree) would require keeping a Cartesian product collection of counter states, which would significantly increase the size of the underlying set T and thereby decrease the success probability of the quantum queries. Instead, we *probabilistically* encode outdegree information into the *higher-order bits* of the same counter that tracks degree.

Note that we do not need to know (d_u^{out}, d_u) exactly to obtain something close to the bias. For technical reasons, [KPV23] adds random noise to the counts to smooth out the estimates, so that they are estimating a different, but closely related value called the *pseudobias*. We only need enough information to decide which *pseudobias* interval $[t_i, t_{i+1})$ the endpoint falls into at the moment an edge arrives for a constant number of pseudobias intervals defined by thresholds $0 < t_1 < \cdots < t_i < \cdots < 1$. For bias estimation, the trick is to treat outdegree only through a coarse, randomized encoding: whenever an out-edge of u is observed, the algorithm performs a "jump" of size k in the same counter with probability $1/k$ (for an appropriate parameter k), so that the *expected* contribution of the jumps is proportional to d_u^{out} while keeping the counter's state space small. Intuitively, this turns the counter value into

$$\text{(degree)} \; + \; k \cdot \text{(a sampled proxy for outdegree)},$$

implemented inside one register rather than as an explicit pair of counters.

Once outdegree is encoded via these rare k-jumps, the algorithm checks outdegree thresholds by probing the stack at *multiple offsets*: instead of only querying at d_h, it queries at positions $d_h,\ d_h + k,\ d_h + 2k, \dots$ (and similarly on the tail side), where the offset index corresponds to the relevant pseudobias interval. As in the degree-only setting, the algorithm does not estimate a bias class directly, but instead runs a constant number of heavy edge counters at appropriately chosen offset pairs and combines their outputs by inclusion–exclusion. Each such heavy edge counter tests whether an edge's endpoints simultaneously exceed the required degree threshold and the required outdegree proxy threshold. By aggregating these estimates, the algorithm obtains an unbiased estimate for the number of edges whose endpoints lie in the specified pseudobias class.

Estimating all pseudobias classes. The discussion above explains how to estimate a *single* pseudosnapshot entry in $O(\log n)$ space, corresponding to one fixed pair of head/tail degree ranges and one fixed pair of head/tail pseudobias thresholds. To approximate MAX-DICUT, however, the algorithm must estimate the pseudosnapshot over all relevant parameter combinations. There are only $O(1)$ bias intervals $[t_i, t_{i+1})$ needed by the final approximation algorithm, but the pseudosnapshot estimator of [KPV23; KPV24] internally refines these intervals using a discretization parameter $\kappa = \tilde{\Theta}(\log n)$. Concretely, within each degree-range pair (α, β), the estimator performs queries indexed by $(i, j) \in [\kappa]^2$, corresponding to κ^2 possible offset pairs used to probe different pseudobias thresholds. The other source of polylogarithmic overhead comes from the degree discretization. The algorithm partitions degrees into

$$L \;=\; \lfloor \log_{1+\varepsilon^3} n \rfloor + 1 \;=\; \Theta\!\left(\frac{\log n}{\varepsilon^3}\right)$$

multiplicative ranges, and it estimates the pseudosnapshot separately for each ordered pair of head/tail degree ranges, yielding $L^2 = \tilde{O}(\log^2 n)$ degree-scale subproblems. For each such subproblem indexed by (α, β) and each of the κ^2 pseudobias index pairs, a single quantum pair sketch instance provides an unbiased but high-variance estimator. The algorithm therefore runs $\tilde{O}(1/\varepsilon^2)$ independent copies per entry (with an additional $\log(1/\delta)$ factor for high success probability) and aggregates them using standard concentration techniques. Overall, the total quantum space in terms of n is

$$O(\log n) \times \tilde{O}(L^2) \times \tilde{O}(\kappa^2) \;=\; \tilde{O}(\log^5 n),$$

which matches the space bound stated in Theorem 1 of [KPV23].

References

[CGV21] Chi-Ning Chou, Alexander Golovnev, and Santhoshini Velusamy. *Optimal Streaming Approximations for all Boolean Max-2CSPs and Max-kSAT*. 2021. arXiv: `2004.11796 [cs.CC]`. URL: `https://arxiv.org/abs/2004.11796`.

[FJ10] Uriel Feige and Shlomo Jozeph. *Oblivious Algorithms for the Maximum Directed Cut Problem*. 2010. arXiv: `1010.0406 [cs.DS]`. URL: `https://arxiv.org/abs/1010.0406`.

[KKS14] Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. *Streaming Lower Bounds for Approximating MAX-CUT*. 2014. arXiv: `1409.2138 [cs.DS]`. URL: `https://arxiv.org/abs/1409.2138`.

[KPV23] John Kallaugher, Ojas Parekh, and Nadezhda Voronova. *Exponential Quantum Space Advantage for Approximating Maximum Directed Cut in the Streaming Model*. 2023. arXiv: `2311.14123 [quant-ph]`. URL: `https://arxiv.org/abs/2311.14123`.

[KPV24] John Kallaugher, Ojas Parekh, and Nadezhda Voronova. *How to Design a Quantum Streaming Algorithm Without Knowing Anything About Quantum Computing*. 2024. arXiv: `2410.18922 [quant-ph]`. URL: `https://arxiv.org/abs/2410.18922`.

[Sax+23a] Raghuvansh R Saxena et al. "Streaming complexity of CSPs with randomly ordered constraints". In: *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2023, pp. 4083–4103.

[Sax+23b] Raghuvansh R. Saxena et al. "Improved Streaming Algorithms for Maximum Directed Cut via Smoothed Snapshots". In: *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*. 2023, pp. 855–870. DOI: `10.1109/FOCS57990.2023.00055`.