(Quantum) Streaming Complexity of $k$-clique Counting

Connor Clayton, Yingkang Cao, Amin Shiraz Gilani

December 10, 2022

1 Introduction

Streaming algorithms are designed to process large datasets that arrive one bit at a time and the goal is to compute a function of these datasets while minimizing the amount of information stored at any step. For graph problems, usually the data is a set of edges encoding a certain graph and the goal is to compute a property of the input graph. The streaming complexity of a problem $\mathcal{P}$ is the minimum memory required in the streaming model to solve $\mathcal{P}$.

A complete graph on $k$ vertices is known as a $k$-clique. Counting $k$-cliques is an important graph problem with numerous applications in dense subgraph mining. In particular, it is extensively used in bioinformatics, community detection, social network analysis and spam detection.

In this article, we will primarily consider the problem of lower bounding the streaming complexity of approximating the number of $k$-cliques in the input graph given as a stream of edges. In Section 2, we set the platform for the next sections by defining Triangle Counting and other relevant problems. We introduce the problems of INDEX and BOOLEAN HIDDEN MATCHING in Section 3 and revisit the reductions that provide lower bounds on the streaming complexity of Triangle Counting. In Section 4, we shed light on the results that we presented in Section 3 in the quantum setting and discuss a major open problem in quantum streaming complexity. Then, we state many communication complexity problems relevant for showing quantum streaming hardness via reductions and some well-known quantum streaming complexity results in Section 5. We proceed by formally defining the $k$-CLIQUE COUNTING and BOOLEAN HIDDEN MATCHING problem in Section 6. In Section 7, we generalize the reduction we presented in Section 3.2 by reducing BOOLEAN HIDDEN HYPERMATCHING to $k$-CLIQUE COUNTING for constant $k$ while in Section 8, we quadratically improve the dependence on $k$ for this reduction by making some observations which may be of independent interest. Finally, we conclude by summarizing our results and stating interesting open questions in Section 9.

2 Triangle Counting

In this section, we formally define the Triangle Counting and Triangle Distinguishing problems, and the parameterizations that we will consider. We begin with the definition of Triangle Counting.

**Triangle Counting**

**Input:** Graph $G = (V, E)$

**Question:** Approximate the number of triangles in $G$.

A related problem that is usually considered in the literature is that of Triangle Distinguishing, which is defined as follows.
**Triangle Distinguishing**

**Input:** Graph $G = (V, E)$ and an integer $T$. It is promised that $G$ has either 0 triangles or $T$ triangles.

**Question:** Does $G$ have 0 triangles or at least $T$ triangles?

Notice that one can use one instance of Triangle Counting to solve an instance of Triangle Distinguishing. Therefore, a lower bound on Triangle Distinguishing implies a lower bound on Triangle Counting. On the other hand, when proving lower bounds on Triangle Counting, it is usually easier and insightful to consider lower bounds for Triangle Distinguishing as the latter is a decision problem and the known algorithms for Triangle Distinguishing usually involve sampling enough edges such that there is one triangle in expectation, which imply an algorithm for Triangle Counting.

Let $n, m$ and $T$ denote the number of vertices, edges and triangles in the input graph $G$. We define $\Delta_V$ (respectively $\Delta_E$) to be the maximum number of triangles in $G$ that share a vertex (respectively an edge). Whenever we present lower bounds for Triangle Counting, we can assume that all these parameters are given to us since the lower bounds for this case will be valid lower bounds for the case when we only have partial information about these parameters. Therefore, our lower bounds will be a function of these parameters.

### 3 Streaming complexity of Triangle Counting

There is a known $\tilde{O}\left(\frac{m\Delta_E}{T} + \frac{m\sqrt{\Delta_V}}{T}\right)$ algorithm for Triangle Counting [5]. In this section, we show a matching lower bound presented in [7] to show that this algorithm is optimal.

It is also possible to show a (trivial) lower bound of $\Omega(m)$ via a reduction from Index. However, as the above upper bound shows, it is possible to give better than $O(m)$ algorithms for some values of $T, \Delta_E,$ and $\Delta_V$ (when $\Delta_E = o(T)$ or when $\Delta_V = o(T)$, which is the case for most graphs). Thus we use this parameterization in our lower bounds.

#### 3.1 Lower bound via Index

In this section, we show a lower bound on the space complexity of single-pass streaming algorithms for Triangle Distinguishing parameterized by $\Delta_E$ via a reduction from the communication problem Index. Recall the definition of Index:

**Index**

**Input:** Alice gets a string $x \in \{0, 1\}^n$; Bob gets an index $i \in [n]$.

**Question:** Compute $x_i$, using one-way communication from Alice to Bob.

It is well-known that Alice needs to send $\Omega(n)$ bits to Bob for solving Index. We use this to show the following result:

**Theorem 1.** Any (quantum) single-pass streaming algorithm for Triangle Distinguishing needs $\Omega\left(\frac{m\Delta_E}{T}\right)$ space.

**Proof.** Assume there is a $o\left(\frac{m\Delta_E}{T}\right)$ space streaming algorithm ALG for Triangle Distinguishing. Note that it works for any arrival order of edges. We will use ALG in the following protocol for Index. The protocol will take $o(n)$ space, which contradicts the communication complexity of Index. Hence no such ALG exists.

1. Alice has $x_1 \cdots x_n \in \{0, 1\}^n$ and Bob has $i \in [n]$. 

---

1.
2. Alice and Bob construct different parts of a graph. The vertices are $V = A \cup B \cup C$ where $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_n\}$ and $C = \{c_1, \ldots, c_n\}$.

- Alice constructs the graph on vertices $A \cup B$ by letting

$$E_A = \{(a_j, b_j) : x_j = 1\}$$

(1)

- Bob constructs the graph on $\{a_i, b_i\} \cup C$ by letting

$$E_B = \{(a_i, c_j) : j \in [n]\} \cup \{(b_i, c_j) : j \in [n]\}$$

(2)

3. (Comment, not part of the algorithm.) See Figure 1 for an example where $x = 10 \cdots 1$. The dashed line denotes that there is either a solid edge or no edge, depending on whether $x_i = 1$ or 0, respectively.

![Figure 1: Reduction from Index to Triangle Distinguishing](image)

4. (Comment, not part of the algorithm.) Note that the resulting graph contains $m = \Theta(n)$ edges. If $x_i = 1$ then there will be $n$ triangles and all of them share a single edge; if $x_i = 0$ then there will be no triangles.

5. Alice runs $E_A$ through the streaming algorithm ALG with parameters $\Delta_E = n$ and $T = n$. Since it uses only $o\left(\frac{m \Delta_E}{T}\right) = o(n)$ space, when she is done there is $o(n)$ bits in memory, which she then sends to Bob.

6. Bob continues ALG on $E_B$. If ALG returns Yes, then he knows that $x_i = 1$; if ALG returns No, then he knows that $x_i = 0$. The total one-way communication is $o(n)$ bits.

Note that the above lower bound derived from the Index problem is not very useful when $\Delta_E$ is small. What happens when $\Delta_E = O(1)$, can we still show that Triangle Distinguishing is hard? Next, we will parameterize the graph using $\Delta_V$ and show another lower bound even when $\Delta_E = O(1)$.
3.2 Lower bound via Boolean Hidden Matching

Now we show a lower bound on the space complexity of streaming algorithms for Triangle Distinguishing parameterized by $\Delta$ via a reduction from the communication problem Boolean Hidden Matching (BHM). We define BHM now:

**Boolean Hidden Matching**

**Input:** Alice gets a string $x \in \{0, 1\}^{2n}$; Bob gets a perfect matching $M$ over $[2n]$ and a string $w \in \{0, 1\}^n$. $M$ is interpreted as a $n \times 2n$ matrix where each row represents an edge in the matching and $w$ is promised to satisfy either $Mx = w$ or $Mx = \overline{w}$ (where $\overline{w}$ is $w$ with every bit flipped).

**Question:** Determine which is the case: $Mx = w$ or $Mx = \overline{w}$.

It is known that Alice must send $\Omega(\sqrt{n})$ bits to Bob in order to solve the Boolean Hidden Matching problem [8, 2]. We will use this to show a lower bound on Triangle Distinguishing.

**Theorem 2.** Any single-pass streaming algorithm for Triangle Distinguishing requires $\Omega \left(\frac{m\sqrt{n}}{T}\right)$ space.

**Proof.** Assume there is a $o \left(\frac{m\sqrt{n}}{T}\right)$ space streaming algorithm ALG for Triangle Distinguishing. Note that it works for any arrival order of edges. We use ALG in the following protocol for Boolean Hidden Matching. The protocol will take $o(\sqrt{n})$ space, which contradicts the communication complexity of Boolean Hidden Matching. Hence no such ALG exists.

1. Alice has $x_1 \cdots x_{2n} \in \{0, 1\}^{2n}$, and Bob has $M \in \{0, 1\}^{n \times 2n}$ and $w_1 \cdots w_n \in \{0, 1\}^n$.

2. Alice and Bob construct different parts of a graph. The vertices are $V = \{a\} \cup B \cup C$ where $B = \{b_1, \cdots, b_{2n}\}$ and $C = \{c_1, \cdots, c_{2n}\}$.

   - Alice constructs the graph on vertices $V$ by letting
     
     $E_A = \{(a, b_j) : x_j = 0\} \cup \{(a, c_j) : x_j = 1\}$ \tag{3}

   - Bob constructs the graph on $B \cup C$ by letting
     
     $E_B = \{(b_i, b_j), (c_i, c_j) : (i, j) \in M \land w_k = 0\} \cup \{(b_i, c_j), (c_i, b_j) : (i, j) \in M \land w_k = 1\}$ \tag{4}

3. (Comment, not part of the algorithm.) See Figure 2 for an example where $x = 0110$ and $M = \begin{bmatrix} 1010 \\ 0101 \end{bmatrix}$.

4. (Comment, not part of the algorithm.) Note that the resulting graph contains $m = \Theta(n)$ edges. If $Mx = w$ then there will be $n$ triangles and all of them share a single vertex; if $Mx = \overline{w}$ then there will be no triangles.

5. Alice runs $E_A$ through the streaming algorithm ALG with parameters $\Delta_E = 1$, $\Delta_V = n$, and $T = n$. Since it uses only $o \left(\frac{m\sqrt{n}}{T}\right) = o(\sqrt{n})$ space, when she is done there is $o(\sqrt{n})$ bits in memory, which she then sends to Bob.

6. Bob continues ALG on $E_B$. If ALG returns Yes, then he knows that $Mx = w$; if ALG returns No, then he knows that $Mx = \overline{w}$. The total one-way communication is $o(\sqrt{n})$ bits.
Figure 2: Example reduction from **Boolean Hidden Matching** to **Triangle Distinguishing** where $x = 0110$ and $M = \begin{bmatrix} 1010 \\ 0101 \end{bmatrix}$. (left) The graph Alice constructs. (center) The graph after Bob draws his edges when $w = 11$, so $Mx = w$. We rearrange this graph on the bottom to emphasize that it contains 2 triangles. (right) The graph after Bob draws his edges when $w = 0$, so $Mx = \overline{w}$. We rearrange this graph on the bottom to emphasize that it contains 0 triangles.

Combining this lower bound with the lower bound obtained from index gives us an overall bound of $\Omega \left( m \Delta E_T + m \sqrt{\Delta V_T} \right)$ bits for Triangle Counting in the streaming model. This matches the best known upper bound up to a poly-logarithmic factor.

### 4 Quantum streaming complexity of Triangle Counting

In this section, we state known results about the quantum streaming complexity of **Triangle Counting**. It is known that the **index** problem has quantum communication complexity $\Omega(n)$ [1]. Therefore, the lower bound of $\Omega \left( \frac{m \Delta E}{T} \right)$ from Theorem 1 follows for single-pass quantum streaming protocols. However, the **Boolean Hidden Matching** problem has logarithmic quantum communication complexity as shown by the following theorem.

**Theorem 3.** The quantum communication complexity of **Boolean Hidden Matching** is $O(\log n)$.

**Proof.** We explicitly provide the quantum communication protocol for **Boolean Hidden Matching** as follows [8, 2].

1. Alice prepares the state $|\psi\rangle = \frac{1}{\sqrt{2n}} \sum_{i \in [2n]} (-1)^{x_i} |i\rangle$ and sends it to Bob.

2. Bob performs a measurement on $|\psi\rangle$ in the orthonormal basis $\{|\phi_{e_i}^+\rangle, |\phi_{e_i}^-\rangle : i \in [n]\}$ where $|\phi_{e_i}^\pm\rangle = \frac{1}{\sqrt{2}} (|u\rangle \pm |v\rangle)$ for $e_i = (u, v)$.

3. If the measurement outcome is $|\phi_{e_i}^{(-1)^{x_i}}\rangle$ for some $i \in [n]$, Bob outputs $Mx = w$; otherwise, he outputs $Mx = \overline{w}$.
Notice that each $|i\rangle$ can be represented using only $O(\log n)$ qubits, which means that representing $|\psi\rangle$ needs $O(\log n)$ qubits. Therefore, Alice only needs to send $O(\log n)$ qubits to Bob.

Now, we show the correctness of this protocol. We compute the probability $p_{e_i, w_i}$ of measuring the outcome $|\phi_{e_i}^{(-1)w_i}\rangle$ as follows.

$$p_{e_i, w_i} = |\langle\phi_{e_i}^{(-1)w_i}|\psi\rangle|^2$$

$$= \frac{1}{\sqrt{2}}(|\langle u|\psi\rangle + (-1)^{w_i}\langle v|\psi\rangle|)^2$$

$$= \frac{1}{2\sqrt{n}}((-1)^{x_u} + (-1)^{w_i}(-1)^{x_v})^2$$

$$= \begin{cases} 
1/n & x_v \oplus x_u = w_i \\
0 & \text{otherwise}
\end{cases}$$

It follows that the probability $p_{e_i, w_i}$ of measuring the outcome $|\phi_{e_i}^{(-1)w_i}\rangle$ is non-zero iff $Mx = w$ since $x_v \oplus x_u = w_i$ for each $e_i = (u, v) \in M$ iff $Mx = w$. Hence, Bob will always correctly output whether $Mx = w$ or not.

Theorem 3 implies that lower bound of $\Omega \left( \frac{m\sqrt{\Delta V}}{T} \right)$ from Theorem 2 does not follow for quantum streaming protocols. This means that the reduction from BOOLEAN HIDDEN MATCHING can only give us an $\Omega(\log n)$ lower bound on the quantum streaming complexity of TRAINGLE COUNTING when $\Delta_E = O(1)$ and $T = \Omega(m)$. Recently, Kallaugher [6] showed a $O(n^{2/5})$ quantum streaming algorithm for this case. In fact, for these sets of parameters, it is unknown whether there exist a $O(\log n)$ quantum streaming protocol for TRAINGLE COUNTING or for any quantum streaming protocol, $\Omega(n^c)$ memory is required for some $c > 0$. It is a major open problem in quantum streaming complexity.

5 Other results in quantum streaming complexity

Here we list some communication problems that are hard even on quantum computers, which may be helpful in proving the space complexity for quantum streaming problems.

1. Any protocol for the following problem needs $\Omega(n)$ qubits of communication [9].

   **INNERPRODUCT**
   
   **Input:** Alice gets a string $x \in \{0, 1\}^n$; Bob gets another string $y \in \{0, 1\}^n$.
   
   **Question:** Compute $\sum_{i=1}^{n} x_i y_i \mod 2$.

2. Any protocol for the following problem needs $\Omega(\sqrt{n})$ qubits of communication [12].

   **DISJOINTNESS**
   
   **Input:** Alice gets a string $x \in \{0, 1\}^n$; Bob gets another string $y \in \{0, 1\}^n$.
   
   **Question:** Compute $\bigvee_{i=1}^{n} (x_i \land y_i)$.

3. Any protocol for the following problem needs $\Omega \left( n^2 \log p \right)$ qubits of communication [14].

   **SINGULARITY($p$)**
   
   **Input:** Alice gets an $n \times n$ matrix $X$ over $\mathbb{F}_p$; Bob gets another $n \times n$ matrix $Y$ over $\mathbb{F}_p$.
   
   **Question:** Determine whether $X + Y$ is singular over $\mathbb{F}_p$. 6
In the following, we briefly review some hardness results for quantum streaming problems. We begin with the known results about the DYCK(2) problem, defined as follows.

**DYCK(2)**

**Input:** A string $s$ over the alphabet $\{a, \overline{a}, b, \overline{b}\}$. The alphabet can be interpreted as encoding two types of parenthesis where $\overline{a}$ is the closing parenthesis for $a$ and $\overline{b}$ is the closing parenthesis for $b$.

**Question:** Is $s$ well-parenthesized?

Jain and Nayak [4] showed that recognizing DYCK(2) is hard if the stream of parentheses is unidirectional:

**Theorem 4.** For any $p \geq 1$, any unidirectional bounded-error randomized $p$-pass streaming algorithm for DYCK(2) needs space $\Omega\left(\sqrt{\frac{n}{p}}\right)$, where $n$ is the length of the string.


**Theorem 5.** For any $p \geq 1$, any unidirectional bounded-error quantum $p$-pass streaming algorithm for DYCK(2) needs space $\Omega\left(\sqrt{\frac{n}{p^3}}\right)$, where $n$ is the length of the string.

Theorems 4 and 5 are based on the communication complexity problem known as **Augmented Index**, introduced in [1] and defined as follows.

**Augmented Index**

**Input:** Alice gets a string $x \in \{0, 1\}^n$; Bob gets an index $i \in [n]$ and the sub-string of $x$ comprising of the first $i - 1$ characters.

**Question:** Compute $x_i$, using one-way communication from Alice to Bob.

For constant number of passes $p$, Theorem 5 is a generalization of Theorem 4. However, it is not known whether one can achieve a quantum protocol that could bypass the randomized lower bound result of Theorem 4 for some $p = \omega(1)$.

Now, consider the **Frequency Moment** ($k$) problem.

**Frequency Moment ($k$)**

**Input:** A data stream of numbers $y_1, \ldots, y_n$ from $[m]$. Let $n_j = |\{\ell : y_\ell = 1\}|$.

**Question:** Compute $F_k = \begin{cases} \max_j n_j & k = \infty \\ \sum_j n_j^k & \text{otherwise}. \end{cases}$

Montanaro [10] showed quantum streaming algorithms for the Frequency Moment ($k$) problem with $k \in \{0, 1, 2, \infty\}$ that uses $\omega(1)$ passes to the input stream and beats the classical lower bound given in [15]. Hamoudi and Magniez [3] generalized this result to all $k$. We formally state the aforementioned classical lower bound [15] and the quantum upper bound [3] as follows.

**Theorem 6.** For any $k \in \mathbb{N} \cup \{\infty\}$, any $p$-pass randomized streaming algorithm that approximates $F_k$ uses $\Omega(n^{1-2/k}/p)$ bits of memory.

**Theorem 7.** For any $k \in \mathbb{N} \cup \{\infty\}$, there is a $p$-pass quantum streaming algorithm that approximates $F_k$ using $\tilde{O}(n^{1-2/k}/p^2)$ qubits of memory.

Notice that for constant $p$, the quantum algorithm of Theorem 7 does not have any advantage for the Frequency Moment ($k$) by Theorem 6. However, for any $p = \omega(1)$ and for any $k \in \mathbb{N} \cup \{\infty\}$, this algorithm requires lower memory than the best-known classical algorithm for the Frequency Moment ($k$) problem with $k \in \mathbb{N} \setminus \{0\} \cup \{\infty\}$. 7
6 k-Clique Counting and Boolean Hidden Hypermatching

In this section, we define two problems for k-clique finding which are analogous to Triangle Counting and Triangle Distinguishing. We also introduce an important problem (Boolean Hidden Hypermatching) that we will use to show a lower bound on these problems and we discuss the motivation for this problem.

**K-Clique Counting**
- **Input:** Graph $G = (V, E)$
- **Question:** Approximate the number of cliques of size $k$ in $G$.

**K-Clique Distinguishing**
- **Input:** Graph $G = (V, E)$ and an integer $C$. It is promised that $G$ has either 0 k-cliques or $C$ k-cliques.
- **Question:** Does $G$ have 0 k-cliques or at least $C$ k-cliques?

Like for triangles, one can use an instance of K-Clique Counting to solve an instance of K-Clique Distinguishing. Therefore, a lower bound on K-Clique Distinguishing implies a lower bound on K-Clique Counting. It is usually easier and sufficient to consider lower bounds for K-Clique Distinguishing.

In Theorem 1, we showed a $\Omega \left( \frac{m \Delta_E}{T} \right)$ space lower bound Triangle Distinguishing. It is not hard to show that a similar reduction gives the same lower bound for K-Clique Distinguishing; however this gives a trivial lower bound on most graphs, since $\Delta_E$ is usually small. We want to show a stronger lower bound for more general graphs. Additionally, since the quantum streaming complexity of triangle counting in the parameter setting $\Delta_E = O(1)$ and $T = \Omega(m)$ is an open problem (see section 4), it might be instructive to look for lower bounds on k-clique counting for $k > 3$ in this parameter setting to understand if the difficulty of this problem is unique for triangle counting. To do this we will introduce a new problem:

**Boolean Hidden Hypermatching**
- **Input:** Alice gets a string $x \in \{0, 1\}^{kn}$; Bob gets a perfect hyper-matching $M$ over $[kn]$ and a string $w \in \{0, 1\}^n$. $M$ is interpreted as a $n \times kn$ matrix where each row represents a hyper-edge in the matching and $w$ is promised to satisfy either $Mx = w$ or $Mx = \overline{w}$ (where $\overline{w}$ is $w$ with every bit flipped).
- **Question:** Determine which is the case: $Mx = w$ or $Mx = \overline{w}$.

Boolean Hidden Hypermatching (BHHM) is a generalization of BHM. Importantly, this problem has a polynomial lower bound both classically and quantumly. The classical lower bound is $\Omega(n^{1-1/k})$ and the quantum lower bound is $\Omega(n^{1-2/k})$ [13].

7 Streaming complexity of k-clique Counting

**Theorem 8.** Any classical single-pass streaming algorithm for K-Clique Distinguishing requires $\Omega \left( m^{1-1/k} \right)$ bits of space. If the algorithm is allowed to be quantum, it requires $\Omega \left( m^{1-2/k} \right)$ qubits of space.

**Proof.** Assume there is a $o \left( m^{1-1/k} \right)$ space classical streaming algorithm $\text{ALG}_C$ and a $o \left( m^{1-2/k} \right)$ space quantum streaming algorithm $\text{ALG}_Q$ for K-Clique Distinguishing. Note that these work for any arrival order of edges. We use $\text{ALG}_C$ or $\text{ALG}_Q$ in the following protocol for Boolean Hidden Hypermatching. The protocol will take $o(n^{1-1/k})$ space in the classical setting and
\( o(n^{1-2/k}) \) space in the quantum setting, both of which contradict the communication complexity of Boolean Hidden Hypermatching. Hence ALGC and ALGQ cannot exist. The reduction is the same for both the classical and quantum settings, so we only present it once.

1. Alice has \( x_1 \cdots x_{kn} \in \{0,1\}^{kn} \), and Bob has \( M \in \{0,1\}^{n \times kn} \) and \( w_1 \cdots w_n \in \{0,1\}^n \).

2. Alice and Bob construct different parts of a graph. The vertices are \( V = \bigcup_{r=1}^{2^{k-1}} V_r \), where \( V_r = \{a^r\} \cup \{b_1^r, \ldots, b_{kn}^r\} \cup \{c_1^r, \ldots, c_{kn}^r\} \). We can think of \( V \) as \( 2^{k-1} \) copies of the graph \( V_r \).

   - Alice constructs the graph on vertices \( V \) by letting
     \[
     E_A = \bigcup_{r=1}^{2^{k-1}} \left[ \{(a^r, b_j^r) : x_j = 0\} \cup \{(a^r, c_j^r) : x_j = 1\} \right] \tag{9}
     \]
     Notice that Alice does the same thing on each copy \( V_r \) (see Figure 3a).

   - Bob constructs the graph on vertices \( V \). Let \( e_l = (i_1, \ldots, i_k) \) be the \( l \)th hyper-edge in the matching \( M \). He does the following for each \( l \in [n] \):
     
     - For every \( r \in [2^{k-1}] \), he assigns \( V_r \) a unique string \( y^{(r)} \) of length \( k \) with Hamming weight the same parity as \( w_l \). So if, e.g., \( k = 4 \) and \( w_l = 0 \), he might assign the strings \( y^{(1)} = 0000, y^{(2)} = 0011, y^{(3)} = 0101, \ldots \)  
     
     - For every \( r \in [2^{k-1}] \), he fully connects the vertices which Alice would connect to if the bits of \( x \) corresponding to \( e_l \) were the same as \( y^{(r)} \), i.e., the vertices:
     \[
     V_B^r = \left\{ b^r_{ij} : i_j \in e_l \land y^{(r)}_j = 0 \right\} \cup \left\{ c^r_{ij} : i_j \in e_l \land y^{(r)}_j = 1 \right\} \tag{10}
     \]
     This amounts to Bob drawing \( n \cdot 2^{k-1} \) \( k \)-cliques (see Figure 3b).

3. (Comment, not part of the algorithm.) See Figure 3 for an example where \( k = 4, x = 0110, \ldots, w = 0 \ldots \) and \( M = \begin{bmatrix} 11110 & \cdots & \vdots \\ 0000 & \cdots & \vdots \end{bmatrix} \).

4. (Comment, not part of the algorithm.) Note: for each hyperedge \( e \), let \( x[e] \) be \( x \) restricted to the indices in \( e \). If \( y^{(i)} = x[e] \), then we have a \( (k+1) \)-clique because Bob draws a \( k \)-clique and Alice connects vertex \( a \) to the same \( k \) vertices. On the other hand, if \( y^{(i)} \neq x[e] \), then we have no \( (k+1) \)-cliques because the \( k \)-clique Bob draws is not fully connected with any other vertices. When \( Mx = w \), there will be an \( i \) such that \( y^{(i)} = x[e] \) so the resulting graph will contain \( n \) \( (k+1) \)-cliques. When \( Mx = \overline{w} \), there will be no \( i \) such that \( y^{(i)} = x[e] \) so the resulting graph will contain no \( (k+1) \)-cliques.

5. Alice runs \( E_A \) through a streaming algorithm ALG (either ALGC or ALGQ). Since it uses only \( o(m^{1-1/k}) = o(n^{1-1/k}) \) bits or \( o(m^{1-2/k}) = o(n^{1-2/k}) \) qubits, when she is done there are \( o(n^{1-1/k}) \) bits or \( o(n^{1-2/k}) \) qubits in memory, which she then sends to Bob.

6. Bob continues ALG on his edge set. If ALG returns Yes, then he knows that \( Mx = w \); if ALG returns No, then he knows that \( Mx = \overline{w} \). The total one-way communication is \( o(n^{1-1/k}) \) bits or \( o(n^{1-2/k}) \) qubits.
The graph after Alice draws her edges (in red).

(b) The graph after Bob draws his edges (in blue). Notice there is a 5-clique in the lower-left copy, but not in any other copy.

Figure 3: Example of a reduction from **Boolean Hidden Hypermatching** to **k-Clique Distinguishing** where \( k = 4 \), \( x = 0110... \), \( w = 0... \), and \( M \) contains the hyper-edge \((1, 2, 3, 4)\).

8 Quadratic improvement on the dependence on \( k \)

The previous reduction from \( k\)-BHMM to \((k + 1)\)-Clique Distinguishing outputs graphs which contain \( |V| = \Theta(2^k n) \) number of vertices. Can we improve the dependence of the graph size on the parameter \( k \)? The answer is yes. We found that we can quadratically reduce the graph size. This is stated as follows.

**Theorem 9.** There is a reduction from \( k\)-BHMM to \((k + 1)\)-Clique Distinguishing on graphs of size \( |V| = \Theta(2^{k/2} n) \).

The main idea is to superimpose different “copies” that Bob drew before into a smaller number of “groups”. We will first show how the “copies” are grouped together, and then describe the reduction algorithm.

In the following, we assume \( k \) is even for simplicity. For each \( y \in \{0, 1\}^k \), we compute the parity of every two consecutive bits of \( y \) to obtain \( \alpha \in \{0, 1\}^{k/2} \):

\[
\alpha_i := y_{2i-1} \oplus y_{2i}, \quad i \in [k/2],
\]

and we say that \( \alpha \) is the name of the group that \( y \) belongs to. Note that there are \( 2^{k/2} \) different groups and each group contains \( 2^{k/2} \) elements. The following observation is key to our reduction algorithm:

**Lemma 10.** For any \( y \in \{0, 1\}^k \) and any \( \beta \in \{0, 1\}^{k/2} \), if \( y \) and \( \beta \) have different parity in their Hamming weights, then there is an index \( i \in [k/2] \) such that for any \( z \in \{0, 1\}^k \) belonging to the group \( \beta \), the following holds:

\[
y_{2i-1} \oplus y_{2i} \neq z_{2i-1} \oplus z_{2i}
\]

**Proof.** By way of contradiction, suppose there is no such index \( i \). Fix any index \( j \in [k/2] \). Then, by assumption, we know that there exists a \( z \in \{0, 1\}^k \) in the group \( \beta \) with \( y_{2j-1} \oplus y_{2j} = z_{2j-1} \oplus z_{2j} \). But this by definition implies that \( y_{2j-1} \oplus y_{2j} = \beta_j \).

Since the above holds for any \( j \), we immediately have \( \bigoplus_{j=1}^k y_j = \bigoplus_{j=1}^{k/2} (y_{2j-1} \oplus y_{2j}) = \bigoplus_{j=1}^{k/2} \beta_j \), which contradicts the assumption that the Hamming weights of \( y \) and \( \beta \) have different parity. \( \square \)
With the above grouping construction, we are now in a position to prove Theorem 9.

**Proof of Theorem 9.** We describe the reduction algorithm below:

1. Alice has $x_1 \cdots x_{kn} \in \{0,1\}^{kn}$, and Bob has $M \in \{0,1\}^{n \times kn}$ and $w_1 \cdots w_n \in \{0,1\}^n$.

2. Alice and Bob construct different parts of a graph. The vertices are $V = \bigcup_{r=1}^{2^{k/2}-1} V_r$, where $V_r = \{a^r\} \cup \{b^r_1, \cdots, b^r_{kn}\} \cup \{c^r_1, \cdots, c^r_n\}$. We can think of $V$ as $2^{k/2-1}$ copies of the graph $V_r$.

   - Alice constructs the graph on vertices $V$ by letting
     \[ E_A = \bigcup_{r=1}^{2^{k/2-1}} \left[ \{ (a^r, b^r_j) : x_j = 0 \} \cup \{ (a^r, c^r_j) : x_j = 1 \} \right] \]
     (13)

     Notice that Alice does the same thing on each copy $V_r$.

   - Bob constructs the graph on vertices $V$. Let $e_l = (i_1, \ldots, i_k)$ be the $l$’th hyper-edge in the matching $M$. He does the following for each $l \in [n]$:
     - For every $r \in [2^{k/2-1}]$, he assigns $V_r$ a unique string $\alpha^{(r)}$ of length $k/2$ with Hamming weight the same parity as the bit $w_l$. So if, e.g., $k = 4$ and $w_l = 0$, he might assign the strings $\alpha^{(1)} = 00, \alpha^{(2)} = 11$.
     - For every $r \in [2^{k/2-1}]$ and every $z \in \{0,1\}^k$ in the group $\alpha^{(r)}$, he fully connects the vertices which Alice would connect to if the $k$ bits in $x$ that correspond to the hyper-edge $e_l$ equal $z$, i.e., the vertices:
     \[ V^{r, z}_B = \left\{ b^r_{i_j} : i_j \in e_l \land z_j = 0 \right\} \cup \left\{ c^r_{i_j} : i_j \in e_l \land z_j = 1 \right\} \]
     (14)

     This amounts to Bob drawing $n \cdot 2^{k-1}$ $k$-cliques.

3. (Comment, not part of the algorithm.) See Figure 4 for an example of how we can reduce the number of copies from $2^{k-1}$ to $2^{k/2-1}$.

4. (Comment, not part of the algorithm.) Notice that in the “yes” case ($Mx = w$, see Figure 4a) where we already had a $(k+1)$-clique, we still draw the same clique, so the resulting graph will contain $\geq n$ $(k+1)$-cliques.

The “no” case ($Mx = w$, see Figure 4b) is more difficult to analyze. First, we notice that the “hub” $a^r$ must be involved in any $(k+1)$-clique, since any $(k+1)$-clique which does not contain $a^r$ must contain two vertices $b^r_i$ and $c^r_i$ (for some $i$) by the pigeonhole principle, but $b^r_i$ and $c^r_i$ are not connected by construction.

Pick any hyper-edge $e_l = (j_1, \cdots, j_k)$, denote $y = x|_{j_1} \cdots j_k$ to be the corresponding bits from $x$. Fix any “group” $\alpha^{(r)}$. From Lemma 10, there is an index $i$ such that for any $z \in \{0,1\}^k$ belonging to the “group” $\alpha^{(r)}$, $y_{2i-1} \oplus y_{2i} \neq z_{2i-1} \oplus z_{2i}$. Without loss of generality, assume $y_{2i-1} \oplus y_{2i} = 0$. Then, Alice either drew the edges $(a^r, b^r_{j_{2i-1}}), (a^r, b^r_{j_{2i}})$ or she drew the edges $(a^r, c^r_{j_{2i-1}}), (a^r, c^r_{j_{2i}})$. By construction of $V^{r, z}_B$ in Eq (14), it is easy to see that $V^{r, z}_B$ contains neither $b^r_{j_{2i-1}}, b^r_{j_{2i}}$ nor $c^r_{j_{2i-1}}, c^r_{j_{2i}}$. Since this is true for all $z$ and $r$, there are no $(k+1)$-cliques.

\[ \square \]
(a) $w_l = 0$: We are in the “yes” case. The 5-clique is preserved by the grouping.

(b) $w_l = 1$: We are in the “no” case. The grouping ensures that there are still no 5-cliques after combining edges.

Figure 4: Example of reducing the number of copies where $k = 4$, $x = 0110...$, and $M$ contains the hyper-edge $(1, 2, 3, 4)$. The graphs on the left result from the reduction in Section 6 (see Figure 3) We are able to reduce the number of copies from 8 to 2 while maintaining the same number of 5-cliques.

9 Conclusion and open questions

We have shown that $k$-clique counting is hard in both the classical and quantum streaming models, proving lower bounds of $\Omega(m^{1-1/k})$ and $\Omega(m^{1-2/k})$ respectively. Moreover, we showed that the reduction for this bound can be performed with graphs of size $|V| = \Theta(2^{k/2}n)$.

An important open question is to decide whether there is a polynomial lower bound or a sub-polynomial upper bound for triangle counting in the quantum streaming model for the case when $\Delta_E = O(1)$ and $T = \Omega(m)$. Reducing from Boolean Hidden Hypermatching does not seem to work: simply assigning $k = 2$ in our reduction above gives a vacuous lower bound of $\Omega(1)$ for triangle counting.

One natural extension of our result would be to determine for what classes of sub-graphs, the quantum streaming complexity of $k$-subgraph counting is polynomial in $n$. In particular, it would be interesting to categorize the quantum streaming complexity of $k$-cycle counting and $k$-star counting.
References


