Best-Order Streaming model

Atish Das Sarma[1,3], Richard J. Lipton[3], Danupon Nanongkai[2,3]

Georgia Institute of Technology, Atlanta, GA.
{atish, rjl, danupon}@cc.gatech.edu

Abstract

We study a new model of computation, called *best-order stream*, on graph problems. Roughly, it is a proof system where a space-limited verifier has to verify a proof sequentially (i.e., it reads the proof as a stream). Moreover, the proof itself is just a specific ordering of the input data. This model is closely related to many models of computation in other areas such as data streams, communication complexity, and proof checking, and could be used in applications such as cloud computing.

In this paper we focus on graph problems where the input is a sequence of edges. We show that even under this model, checking some basic graph properties deterministically requires linear space in the number of nodes. To contrast this, we show that randomized verifiers are powerful enough to check many graph properties in poly-logarithmic space.

Key words: Data Stream, Model of Computation, Communication Complexity

1. Introduction

This paper is motivated by three fundamental questions that arise in three widely studied areas in theoretical computer science - streaming algorithms, communication complexity, and proof checking. The first question is how efficient can space restricted streaming algorithms be. The second question, is whether the lower bound of a communication problem holds for

[1]Supported in part by ACO fellowship and ARC fellowship, Georgia Institute of Technology.

[2]Supported in part by ACO fellowship, Georgia Institute of Technology.

[3]Supported in part by NSF grant 0902717.

every partition of the input. Finally, in proof checking, the question is how many (extra) bits are needed for the verifier to establish a proof in a restricted manner. Before elaborating on these questions, we first describe an application that motivates our model.

Many big companies such as Amazon [1] and `salesforce.com` are currently offering *cloud computing* services. These services allow their users to use the companies' powerful resources for a short period of time, over the Internet. They also provide some softwares that help the users who may not have knowledge of, expertise in, or control over the technology infrastructure ("in the cloud") that supports them.[4] These services are very helpful, for example, when a user wants a massive computation over a short period of time.

Now, let's say that you want the cloud computer to do a simple task such as checking if a massive graph is strongly connected. Suppose that the cloud computer gets back to you with an answer "Yes" suggesting that the graph is strongly connected. What do you make of this? What if there is a bug in the code, or what if there was some communication error? Ideally one would like a way for the cloud to *prove* to you that the answer is correct. This proof might be long due to the massive input data; hence, it is impossible to keep everything in your laptop's main memory. Therefore, it is more practical to read the proof as a *stream* with a small working memory. Moreover, the proof should not be too long – one ideal case is when the proof is the input itself (in a specific order). This is the model considered in this paper. Related models motivated by similar applications have also been studied by Li et al. [2, 3], Papadopoulos et al. [4], Goldwasser et al. [5], Chakrabarti et al. [6], and Cormode et al. [7].

We describe previous models studied specifically in the stream, computational complexity, and proof checking domains and contrast them with our model.

Data Streams: The basic premise of streaming algorithms is that one is dealing with a humongous data set, too large to process in main memory. The algorithm has only sequential access to the input data; this is called a *stream*. In certain settings, it is acceptable to allow the algorithm to perform multiple

[4] `http://www.ebizq.net/blogs/saasweek/2008/03/distinguishing_cloud_computing/`.

passes over the stream. The general streaming algorithms framework has been studied extensively since the seminal work of Alon, Matias, Szegedy [8].

Models diverge in the assumptions made about what order the algorithm can access the input elements in. In the classic Finite State Automata model [9], the order of the data is set by the definition of the problem. Streaming models allow a richer class of order types. The most stringent restriction on the algorithm is to assume that the input sequence is presented to the algorithm in an adversarial order. A slightly more relaxed setting, that has also been widely studied is where the input is assumed to be presented in randomized order [10, 11, 12]. However, even a simple problem like finding median (which was considered in the earliest paper in the area by Munro and Patterson [13]) was shown recently [10] to require $\Omega(\log \log n)$ passes in both input orders if the space is bounded by $O(\text{polylog}\, n)$. In [14], one of the earliest paper in this area, it was shown that many graph problems require prohibitively large amount of space to solve. It is confirmed by the more recent result [15] that a huge class of graph problems cannot be solved efficiently in a few passes. Since then, new models have been proposed to overcome this obstruction. Feigenbaum et. al. [16] proposed a relaxation of the memory restriction in what is called the semi-stream model. Another input order suggested by Aggarwal et. al. [17] is that of receiving the input in some sorted order. In the classic Binary Decision Diagram [18] the order used is of *best oblivious*; i.e., the input is presented in the best manner for the problem but not necessarily for the problem instance.

Another model that has been considered is the W-Stream (write-stream) model [19, 20]. While the algorithm processes the input, it may also *write* a new stream to be read in the next pass.

We ask the following fundamental question:

> *If the input is presented in the best order possible, can we solve problems efficiently?*

A precise explanation is reserved for the models in Section 2; however, intuitively, this means that the algorithm processing the stream can decide on a *rule* on the order in which the stream is presented. We call this the best-order stream model. For an example, if the rule adopted by the algorithm is to read the input in sorted order, then this is equivalent to the single pass sort stream model. Another example of a rule, for graphs presented as edge streams could be that the algorithm requires all edges incident on a vertex to be presented together. This is again equivalent to a graph stream model

studied earlier called the incidence model (and corresponds to reading the rows of the adjacency matrix one after the other). A stronger rule could be that the algorithm asks for edges in some perfect matching followed by other edges. As we show in this paper, this rule leads to checking if the graph has a perfect matching and as a consequence shows the difference between our model and the sort-stream model.

Communication Complexity: Another closely related model is the communication complexity model [21, 22]. In the basic form of this model, two players, Alice and Bob, receive some input data and they want to compute some function together. The question is how much communication they have to make to accomplish the task. There are many variations of how the input is partitioned. The worst-case [23] and the best-case [24] partition models are two extreme cases that are widely studied over decades. The worst case asks for the partition that makes Alice and Bob communicate the most while the best case asks for the partition that makes the communication smallest. Moreover, even very recently, another variation where the input is partitioned according to some known distribution (see, e.g., [25]) was proposed. The main question is whether the lower bound of a communication problem holds for almost every partition of the input, as opposed to holding for perhaps just a few atypical partitions.

The communication complexity version of our model (described in Section 2) asks the following similar question: Does the lower bound of a communication problem hold for *every* partition of the input? Moreover, our model can be thought of as a more extreme version of the best-case partition communication complexity. We explain this in more details in Section 2.

Proof Checking: From a complexity theoretic standpoint, our model can be thought of as the case of proof checking where a polylog-space verifier is allowed to read the proof as a stream; additionally, the proof must be the input itself in a different order.

The field of probabilistically checkable proofs (PCPs) [26, 27, 28] deals with a verifier querying the proof at very few points (even if the data set is large and thus the proof) and using this to guarantee the proof with high probability. While several variants of proof checking have been considered, we only state the most relevant ones. A result most related to our setting is by Lipton [29] where it was shown that membership proofs for NP can be checked by probabilistic logspace verifiers that have one-way access to the

proof and use $O(\log n)$ random bits. This result almost answers our question except that the proof is not the reordered input and, more importantly, its size is not linear (but polynomial) in the size of the input which might be too large for many applications.

Another related result that compares streaming model with other models is by Feigenbaum et. al. [30] where the problem of testing and spot-checking on data streams is considered. They define sampling-tester and streaming-tester. A sampling-tester is allowed to sample some (but not all) of the input points, looking at them in any order. A streaming-tester, on the other hand is allowed to look at the entire input but only in a specific order. They show that some problems can be solved in a streaming-tester but not by a sampling-tester, while the reverse holds for other problems. Finally, we note that our model (when we focus on massive graphs) might remind some readers of the problem of property testing in massive graphs [31]. Chakrabarti et al. [6] consider an annotation model for streaming proofs, again motivated by cloud computing services. Their model allows a helper to add additional bits to the stream to generate a proof to be presented to the verifier. In this model, the helper observes the stream concurrently with the algorithm. In follow up work to Chakrabarti et al. [6] and this paper, Cormode et al. [7] consider a similar annotation model where the cloud and the verifier look at the stream input. Subsequently, the cloud service needs to provide a proof to the verifier about the specific problem, which may include the reordered stream and may include additional helper bits as well. The verifier still needs to work with small space though since the proof itself may be long.

Notice that in all of the work above, there are two common themes. The first is verification using *small space*. The second is some form of *limited access* to the input. The limited access is either in the form of sampling from the input, limited communication, or some restricted streaming approach. Our model captures both these aspects.

Our Results

In this paper, we partially answer whether there are efficient streaming algorithms when the input is in the best order possible. We give a negative answer to this question for the deterministic case and show evidence of a positive answer for the randomized case. Our positive results are similar in spirit to those for the W-stream and Sort-stream models [17, 20, 19].

For the negative answer, we show that the space requirement is too large even for the simple problem of checking if a given graph has a perfect match-

ing deterministically. In contrast, this problem, as well as the connectivity problem, can be solved efficiently by randomized algorithms. We show similar results for other graph properties.

Organization: The rest of the paper is organized as follow. In Section 2 we describe our best-order streaming model formally and also define some of the other communication complexity models that are well-studied. The problem of checking for distinctness in a stream of elements is discussed in Section 3. This is a building block for most of our algorithms. The following section, Section 4, talks about how perfect matchings can be checked in our model. We discuss the problem of stream checking graph connectivity in Section 5. Our techniques can be extended to a wide class of graph problems such as checking for regular bipartiteness, non-bipartiteness, Hamiltonian cycles etc. We describe the key ideas for these problems in Section 6. Finally, we conclude in Section 7 by stating some insights drawn from this paper, mention open problems and describe possible future directions.

2. Models

In this section we explain our main model and other related models that will be useful in the subsequent sections.

2.1. Best-Order Streaming Model

Recall the following classical streaming model which will be called the *worst-order* stream in this paper, to contrast with the proposed best-order stream. In this model, an input is in some order $e_1, e_2, ..., e_m$, where m is the size of the input. The input $e_1, e_2, \ldots, e_m$ could be numbers, edges, or any other items. In this paper, we are interested in the case where they are edges. We will assume this implicitly throughout. Moreover, we assume that the input element is indivisible (e.g., vertices in e_i must appear consecutively). In the case of graph problems considered in this paper, we also assume that the number of vertices is known to the algorithm before reading the stream. (We note that the algorithms presented in this paper also work even when we assume that the number of vertices are known only approximately.)

Consider any function f that maps the input stream to $\{0, 1\}$. The goal of the typical one-pass streaming model is to develop an algorithm that uses small space to read the input in order $e_1, e_2, \ldots, e_m$ and calculate $f(e_1, e_2, \ldots, e_m)$.

In the *best-order streaming* model, we consider any function f that is *order-independent*. That is, for any permutation π,

$$f(e_1, e_2, \ldots, e_m) = f(e_{\pi(1)}, e_{\pi(2)}, \ldots, e_{\pi(m)}) .$$

Note that many graph properties (including those considered in this paper) satisfy the above property. Our main question is how much space a one-pass streaming algorithm needs in order to compute f if the input is provided in the best order possible. Formally, for any function $s(m)$ and any function f, we say that a language L determined by f is in the STREAM-PROOF($s(m)$) class if there exists a streaming algorithm $\mathcal{A}$ with space $s(m)$ such that

- if $f(e_1, e_2, ..., e_m) = 1$ then there exists a permutation π such that $\mathcal{A}(e_{\pi(1)}, e_{\pi(2)}, \ldots, e_{\pi(m)})$ answers 1;
- otherwise, $\mathcal{A}(e_{\pi(1)}, e_{\pi(2)}, ..., e_{\pi(m)})$ answers 0 for *every* permutation π.

The other way to interpret this model is to consider the situation where there are two players in the setting, the *prover* and the *verifier*. The job of the prover is to provide the stream in some order so that the verifier can compute f using the smallest amount of memory possible. We assume that the prover has unlimited power but restrict the verifier to read the input in a streaming manner (with a limited memory).

The model above can be generalized to the following models.

- STREAM(p, s): A class of problems that, when presented in the best-order, can be checked by a deterministic streaming algorithm $\mathcal{A}$ using p passes and $O(s)$ space.
- RSTREAM(p, s): A class of problems that, when presented in the best-order, can be checked by a randomized streaming algorithm $\mathcal{A}$ using p passes and $O(s)$ space. The output is correct with probability at least 2/3.

It is important to point out that when the input is presented in a specified order, we still need to check that the oracle is not *cheating*. That is, we indeed need a way to verify that we receive the input based on the rule we asked for. This often turns out to be the difficult step.

To contrast this model with the well-studied communication complexity models, we first define a new communication complexity model called *magic-partition* communication complexity. We later show a relationship between this model and the best-order streaming model.

2.2. Magic-Partition Communication Complexity

Recall the following standard 2-player communication complexity which we call *worst-partition* communication complexity. In this model, an input S, which is the set of elements, is partitioned into two sets X and Y, which are given to Alice and Bob, respectively. Alice and Bob want to together compute $f(S)$, for some order-independent function f. In the *worst-partition* case, we consider the case when the input is partitioned in an adversarial way, i.e., we partition the input into X and Y in such a way that Alice and Bob have to communicate as many bits as possible.

For the *magic-partition* communication complexity, we instead consider the case when the input is partitioned in the *best* way possible. Formally, the magic-partition communication complexity consists of three players, the oracle, and Alice and Bob. An algorithm on this model consists of a function $\mathcal{O}$ (owned by the oracle) that partitions the input set $S = \{e_1, e_2, \ldots, e_m\}$ to two sets $X = \{e_{\pi(1)}, e_{\pi(2)}, \ldots, e_{\pi(\lfloor m/2 \rfloor)}\}$ and $Y = \{e_{\pi(\lfloor m/2 \rfloor+1)}, e_{\pi(\lfloor m/2 \rfloor+2)}, \ldots, e_{\pi(m)}\}$ for some permutation π and a protocol $\mathcal{P}$ used to communicate between Alice and Bob. We say that an algorithm consisting of $\mathcal{O}$ and $\mathcal{P}$ has communication complexity $c(m)$, for some function c, if

- for an input S such that $f(S) = 1$, the protocol $\mathcal{P}$ uses $c(m)$ bits of communication and outputs 1 when it is run on the sets X and Y partitioned according to $\mathcal{O}$, and
- for an input S such that $f(S) = 0$, the protocol $\mathcal{P}$ uses $c(m)$ bits of communication and outputs 0 when it is run on *any* sets X and Y coming from any partition.

One way to think of this protocol is to imagine that there is an oracle who looks at the input and then decides how to divide the data between Alice and Bob so that they can compute f using the smallest number of communicated bits and Alice and Bob have to also check if the oracle is lying. We restrict that the input data must be divided equally between Alice and Bob.

Example. Suppose that the input is a graph G. Alice and Bob might decide that the graph be broken down in a topological order, i.e., they traverse the vertices in topological order and order the edges by the time they first visit vertices incident to them. It is important to note the distinction that Alice and Bob actually have not seen the input; but they specify a *rule* by which

to partition the input, when actually presented.

Note that this type of communication complexity should not be confused with the best-partition communication complexity (defined in the next section).

The magic-partition communication complexity will be the main tool to prove the lower bounds of the best-order streaming model. The following lemma is the key to prove our lower bound results.

Lemma 2.1. *For any function f, if the deterministic magic-partition communication complexity of f is at least s, for some s, then for any p and t such that $(2p-1)t < s$, $f \notin \textsc{Stream}(p,t)$.*

Proof. Suppose that the lemma is not true; i.e., f has a magic-partition communication complexity at least s, for some s, but there is a best-order streaming algorithm $\mathcal{A}$ that computes f using p passes and t space such that $(2p-1)t < s$. Consider any input $e_1, e_2, ..., e_n$. Let π be a permutation such that $e_{\pi(1)}, e_{\pi(2)}, ..., e_{\pi(n)}$ is the best ordering of the input for $\mathcal{A}$. Then, define the partition of the magic-partition communication complexity by allocating $e_{\pi(1)}, e_{\pi(2)}, ..., e_{\pi(\lfloor n/2 \rfloor)}$ to Alice and the rest to Bob.

Alice and Bob can simulate $\mathcal{A}$ as follows. First, Alice simulates $\mathcal{A}$ on $e_{\pi(1)}, e_{\pi(2)}, ..., e_{\pi(\lfloor n/2 \rfloor)}$. Then, she sends the data on her memory to Bob. Then, Bob continues simulating $\mathcal{A}$ using the data given by Alice (as if he simulates $\mathcal{A}$ on $e_{\pi(1)}, e_{\pi(2)}, ..., e_{\pi(\lfloor n/2 \rfloor)}$ by himself). He then sends the data back to Alice and the simulation of the second pass of $\mathcal{A}$ begins. Observe that this simulations needs $2p-1$ rounds of communication and each round requires at most t bits. Therefore, Alice and Bob can compute f using $(2p-1)t < s$ bits, contradicting the original assumption. □

Similarly, if the randomized magic-partition communication complexity of f is at least s, for some s, then for any p and t such that $(2p-1)t < s$, $f \notin \textsc{RStream}(p,t)$. Note also that the converse of the above lemma clearly does not hold.

2.3. Related models

We now describe some previously studied communication complexity models that resemble ours.

2.3.1. Best-Partition Communication Complexity

The best-partition communication complexity model was introduced by Papadimitriou and Sipser [24] and heavily used for proving the lower bounds for many applications including VLSI (see [32, 33, 22] and references therein). (In fact, many early communication complexity results are in this model.)

In this model, Alice and Bob can pick how to divide the data into two parts of roughly equal size among them *before* they see the input. This means that they can decide that if an element e appears in the stream, who will get this element. After this decision, the adversary, knowing this partitioning rule, gives an input that makes them communicate the most.

We note the following distinction between this model and the magic-partition model. In this model the players have to pick how data will be divided before they see the input data. For example, if the data is the graph of n vertices then, for any edge (i, j), Alice and Bob have to decide who will get this edge if (i, j) is actually in the input data. However, in the magic-partition model, Alice and Bob can make a more complicated partitioning rule such as giving $(1, 2)$ to Alice *if* the graph is connected. (In other words, in the magic-partition model, Alice and Bob have an oracle that helps them decide how to divide an input *after* he sees it).

Similar to the magic-partition communication complexity, this model makes many problems easier to solve than the traditional worst-partition model where the worst partitioning is assumed. However, the magic-partition model adds more power to the algorithms. In fact, the best-partition makes some problems strictly easier than the worst-partition model and the magic-partition model makes some problems strictly easier than the best-partition model, as shown in the following two examples.

Example. Consider the set disjointness problem. In this problem, two n-bit vectors x and y that are characteristic vectors of two sets X and Y are given. Alice and Bob have to determine if $X \cap Y = \emptyset$. In other words, they want to know if there is a position i such that the i-th bits of x and y are both one.

In the randomized worst case communication complexity, it has been proved that Alice has to send roughly n bits to Bob when x is given to Alice and y is given to Bob. However, for the best-partition case, they can divide the input in the following way: Alice receives the first $n/2$ bits of x and y and Bob receives the rest. This way, each of them can check the disjointness separately and Alice only has to send one bit to Bob (to indicate whether her strings are disjoint or not). Therefore, this problem in the best-partition

model is strictly easier than in the worst-partition model.

Example. Consider the connectivity problem. Hajnal et al. [34] show that the best-partition communication complexity of connectivity is $\Theta(n \log n)$. In contrast, we show that $O((\log n)^2)$ is possible in our model in this paper. Therefore, this problem in the magic-partition model is strictly easier than in the best-partition model.

2.3.2. Nondeterministic Communication Complexity

In this model, Alice and Bob receive x and y respectively. An oracle, who sees x and y, wants to convince them that "$f(x, y) = 1$". He does so by giving them a proof. Alice and Bob should be able to verify the proof with a small amount of communication. This model is different from the magic-partition model in that additional information (the proof) is provided by the oracle.

Example. Let x and y be n-bit strings. Consider the function $f(x, y)$ which is 1 if and only if $x \neq y$. If a proof is allowed, it can simply be the number i where $x_i \neq y_i$. Then, Alice and Bob can check the proof by exchanging one bit (x_i and y_i). If $x = y$ then there is no proof and Alice and Bob can always detect the fake proof.

3. Detecting a Duplicate and Checking for Distinctness

In this section, we consider the following problem which is denoted by DISTINCT. Given a stream of n numbers $a_1, a_2, ..., a_n$ where $a_i \in \{1, 2, ..., n\}$, we want to check if every number appears exactly once (i.e., no duplicate). We are interested in solving this problem in the worst-order streaming model. This problem (in the worst-order model) appears to be a crucial component in solving all the problems we consider in the best-order streaming model and we believe that it will be useful in every problem.

Our goal in this section is to find a one-pass worst-order streaming algorithm for this problem. The algorithm for this problem will be an important ingredient of all algorithms we consider in this paper. In this section, we show that

1. any deterministic algorithm for this problem needs $\Omega(n)$ space, and
2. there is a randomized algorithm that solves this problem in $O(\log n)$ space with an error probability at most $\frac{1}{n}$.

3.1. Space lower bound of the deterministic algorithms

Since checking for distinctness is equivalent to checking if there is a duplicate, a natural problem to consider as a lower bound is the *set disjointness problem*. We define a variation of this problem called *full set disjointness problem*, denoted by F-DISJ.

In this problem, a set $X \subseteq [n]$ is given to Alice and a set $Y \subseteq [n]$ is given to Bob where $[n] = \{1, 2, 3, ..., n\}$ and $|X| + |Y| = n$. Alice and Bob want to together check whether $X \cap Y = \emptyset$.

Note that this problem is different from the well-known set disjointness problem in that we require $|X| + |Y| = n$. Although the two problems are very similar, they are different in that the set disjointness problem has an $\Omega(n)$ lower bound for the randomized protocol in the worst-partition communication complexity model while the F-DISJ has a $O(\log n)$-communication randomized protocol (shown in the next section). We also note that the lower bound of another related problem called k-disjointness problem (see, e.g., [22, example 2.12] and [35]) does not imply the lower bound of F-DISJ shown here.

Now we show that F-DISJ is hard in the deterministic case. The proof is essentially the same as the proof of the lower bound of the set disjointness problem.

Theorem 3.1. *The communication complexity of* F-DISJ *is* $\Omega(n)$.

PROOF. We use a standard technique called the fooling set technique. A fooling set is a set $F\{(A_1, B_1), (A_2, B_2), \ldots, (A_k, B_k)\}$ of size k such that $f(A_i, B_i) = 1$ for all i and $f(A_i, B_j) = 0$ for all $i \neq j$. Once this is shown, it will follow that the deterministic communication complexity is $\Omega(\log(|F|))$. (See the proof in, e.g., [22]).

Now, consider the fooling set $F = \{(A, N \setminus A) \ : \ \forall A \subseteq N\}$. It is easy to check that the property above holds. Since $|F| = 2^n$, the number of bits needed to sent between Alice and Bob is at least $\log(|F|) = \Omega(n)$. □

We note that the theorem also follows from the lower bound of the variation of EQUALITY (checking whether $X = Y$) where we let $Y = [n]$. The theorem implies the space lower bound of DISTINCT.

Corollary 3.2. *Any deterministic worst-order streaming algorithm for* DISTINCT *needs* $\Omega(n)$ *space.*

This lower bound is for the worst-order input. The reason we mention this here is because this seems to be an inherent difficulty in the algorithms in the best-order streaming model. As shown later, all algorithms developed in this paper need to solve DISTINCT as a subroutine. In fact, for all these algorithms, solving DISTINCT is the only part that needs the randomness.

3.2. Randomized algorithm

In this subsection we present a randomized one-pass worst-order streaming algorithm that solves DISTINCT using $O(\log n)$ space. This algorithm is based on the *Fingerprinting Sets* technique introduced by Lipton [36, 29]. Roughly speaking, given a multi-set $\{x_1, x_2, ..., x_k\}$, its *fingerprint* is defined to be

$$\Pi_{i=1}^{k}(x_i + r) \mod p$$

where p is a random prime and $r \in \{0, 1, ..., p-1\}$. We use the following property of the fingerprints.

Theorem 3.3. *[29] Let $\{x_1, x_2, ..., x_k\}$ and $\{y_1, y_2, ..., y_l\}$ be two multi-sets. If the two sets are equal then their fingerprints are always the same. Moreover, if they are unequal, the probability that they get the same fingerprints is at most*

$$O(\frac{\log b + \log m}{bm} + \frac{1}{b^2 m})$$

where all numbers are b-bit numbers and $m = \max(k, l)$ provided that the prime p is selected randomly from interval

$$[(bm)^2, 2(bm)^2].$$

Now, to check if $a_1, a_2, ..., a_n$ are all distinct, we simply check if the fingerprints of $\{a_1, a_2, ..., a_n\}$ and $\{1, 2, ..., n\}$ are the same. Here, $b = \log n$ and $m = n$. Therefore, the error probability is at most $1/n$.

Remark. We note that the fingerprinting sets technique can also be used in our motivating application of cloud computing above. That is, when the cloud sends back a graph as a proof, we have to check whether this "proof" graph is the same as the input graph we sent. This can be done by checking if the fingerprints of both graphs are the same. This enables us to concentrate on checking the stream without worrying about this issue in the rest of this paper.

We also note that the recent result by Gopalan et al. [37] can be modified to solve DISTINCT as well. Finally, note that we need to know n, or its upper bound, before we run the algorithm.

4. Perfect Matching

We exhibit the ideas of developing algorithms and lower bounds in the best-order streaming model through the perfect matching problem.

Problem. Let G be an input graph of n vertices where the vertices are labeled $1, 2, \ldots, n$. Given the edges of G in a streaming manner $e_1, e_2, ..., e_m$, we want to compute $f(e_1, ..., e_m)$ which is 1 if and only if G has a perfect matching. Let n be the number of vertices.

4.1. Upper Bound

Theorem 4.1. *The problem of determining if there exists a perfect matching can be solved by a randomized best-order streaming algorithm using $O(\log n)$ space with a success probability at least $1 - 1/n$.*

PROOF. Consider the following algorithm.

Algorithm. The prover sends $n/2$ edges of a perfect matching to the verifier first and then send the rest of the edges. The verifier then check the followings.

1. Check if the first $n/2$ edges form a perfect matching. This can be done by checking whether the fingerprint of the set $\bigcup_{i=1}^{n/2} e_i$ (where $e_1, e_2, \ldots, e_{n/2}$ are the first $n/2$ edges in the stream) is equal to the fingerprint of the set $\{1, 2, \ldots, n\}$.
2. Check if there are at most n vertices. This is done by checking that the maximum vertex label is at most n.

Finally, the verifier outputs 1 if the input passes all the above tests.

The correctness of this algorithm is quite straightforward, as follows. First, if the edges $e_1, e_2, \ldots e_{n/2}$ form a perfect matching then $e_1, e_2, \ldots e_{n/2}$ have no vertex in common and, therefore, $\bigcup_{i=1}^{n/2} e_i = \{1, 2, \ldots, n\}$. This means that the fingerprints of $\bigcup_{i=1}^{n/2} e_i$ and $\{1, 2, \ldots, n\}$ are always the same. Thus,

the first condition holds. The second condition can be also easily checked. Therefore, the algorithm will output 1 in this case.

For the case that the edges $e_1, e_2, \ldots e_{n/2}$ do not form a perfect matching, observe that $\bigcup_{i=1}^{n/2} \neq \{1, 2, \ldots, n\}$ and therefore the fingerprints of the two sets will be different with probability at least $1 - 1/n$. Consequently, the algorithm will successfully output 0 with probability at least $1 - 1/n$. □

4.2. Lower Bound

We show that the deterministic best-order streaming algorithms for the perfect matching problem have $\Omega(n)$ lower bound *if the input is ordered in an explicit way*; i.e., each edge cannot be split. This means that an edge is either represented in the form (a, b) or (b, a). The proof follows from a reduction from the magic-partition communication complexity (cf. Section 2) of the same problem by using Lemma 2.1.

Theorem 4.2. *If the input can be reordered only in an explicit way then any deterministic algorithm solving the perfect matching problem needs* $\Omega(n)$ *space, where* n *is the number of vertices.*

Proof. Let n be any even integer divisible by four. We show that the above theorem is true even when the input always contains exactly $n/2$ edges. In this case, checking whether these $n/2$ edges form a perfect matching is equivalent to checking whether every vertex appears as an end vertex of exactly one edge. We note that the input is allowed to contain multiple edges. (However, such inputs clearly do not form perfect matchings over n vertices.)

We now show that the magic-partition communication complexity of the perfect matching problem is $\Omega(n)$. Once this is done, the theorem follows immediately by Lemma 2.1.

Consider any magic-partition communication complexity protocol which consists of a partition function $\mathcal{O}$ owned by an oracle and a communication protocol $\mathcal{P}$ between Alice and Bob. That is, a function $\mathcal{O}$ partitions the input into two sets of edges, A and B where $|A| = n/4$ and $|B| = n/4$. Then, A and B are sent to Alice and Bob, respectively. Alice and Bob, upon receiving A and B, communicate to each other using a protocol $\mathcal{P}$ and one of them outputs whether the input edges form a perfect matching or not ("YES" or "NO"). The main goal is to show that for any partition function $\mathcal{O}$, there is some input that forces $\mathcal{P}$ to incur $\Omega(n)$ bits of communication.

Recall that $\mathcal{P}$ has to deal with the following cases: 1) If the input is a perfect matching, $\mathcal{P}$ has to output YES when the input is partitioned according to $\mathcal{O}$. 2) Otherwise, $\mathcal{P}$ has to output NO for *any* partition of the input. We now show the communication complexity of $\mathcal{P}$.

First, let us consider the inputs that are perfect matchings. Let $g(n)$ denote the number of distinct perfect matchings in the complete graph K_n. Observe that

$$g(n) = \frac{n!}{(n/2)!2^{n/2}}.$$

Denote these matchings by $M_1, M_2, \ldots, M_{g(n)}$. For any integer i, let A_i and B_i be the partition of M_i according to $\mathcal{O}$. We now partition $M_1, \ldots, M_{g(n)}$ into clusters in such a way that the matchings whose vertices are partitioned in the same way are in the same cluster. That is, any two inputs M_i and M_j are in the same cluster if and only if $\bigcup_{e \in M_i} e = \bigcup_{e \in M_j} e$.

We claim that there are at least $\binom{n/2}{n/4}$ clusters. To see this, observe that for any matching M_i, there are at most $g(n/2)^2$ matchings that vertices *could be* partitioned the same way as M_i. (I.e., if we define $V(A_i) = \{v \in V \ : \exists e \in A_i \ s.t. \ v \in e\}$ then for any i, $|\{j \ : \ V(A_i) = V(A_j)\}| \leq g(n/2)^2$.) This is because $n/2$ vertices on each side of the partition can make $g(n/2)$ different matchings. This implies that the size of each cluster is at most $g(n/2)^2$. Therefore, the number of matchings such that the vertices are divided differently is at least

$$\frac{g(n)}{g(n/2)^2} = \frac{n!}{(n/2)!2^{n/2}} \left(\frac{(n/4)!2^{n/4}}{(n/2)!} \right)^2 = \binom{n}{n/2} / \binom{n/2}{n/4} \geq \binom{n/2}{n/4}$$

where the last inequality follows from the fact that $\binom{n}{n/2}$ is the number of subsets of $\{1, 2, ..., n\}$ of size $n/2$ and $\binom{n/2}{n/4}^2$ is the number of parts of these subsets.

Let t be the number of clusters (so $t \geq \binom{n/2}{n/4}$) and let $M_{i_1}, M_{i_2}, \ldots, M_{i_t}$ be the inputs from different clusters and let $(A_{i_1}, B_{i_1}) \ldots, (A_{i_t}, B_{i_t})$ be the corresponding partitions according to $\mathcal{O}$. Observe that for any $t' \neq t''$, an input consisting of edges in $M_{i_{t'}}$ and $M_{i_{t''}}$ is not a perfect matching. Moreover, observe that for any t' and t'', any pair $(A_{i_{t'}}, B_{i_{t''}})$ could be an input to the protocol $\mathcal{P}$ (since the oracle can partition the input in anyway when the input is not a perfect matching). In other words, the communication complexity of $\mathcal{P}$ is the *worst case* (in term of communication bits) among all pairs $(A_{i_{t'}}, B_{i_{t''}})$.

For readers who are familiar with the standard fooling set argument, it follows almost immediately that the communication complexity of $\mathcal{P}$ is $\Omega(\log t) = \Omega(n)$ and the theorem is thus proved. For those who are not familiar with this argument, we offer the following alternative argument.

Let $t' = \lfloor \log t \rfloor$. (Note that $t' = \Omega(n)$.) Consider the problem $\text{EQ}_{t'}$ where Alice and Bob each gets a t'-bit vector x and y, respectively. They have to output YES if $x = y$ and NO otherwise. It is well known (see, e.g., [22, Example 1.21]) that the deterministic worst-partition communication complexity of $\text{EQ}_{t'}$ is at least $t' + 1 = \Omega(n)$.

Now we reduce $\text{EQ}_{t'}$ to our problem using the following protocol $\mathcal{P}'$: Upon receiving x and y, Alice and Bob locally map x to A_{i_x} and y to B_{i_y}, respectively and then simulate $\mathcal{P}$. Since $x = y$ if and only if $A_{i_x} \cup B_{i_y}$ is a perfect matching, $\mathcal{P}'$ outputs YES if and only if $x = y$. Therefore, Alice and Bob can use the protocol $\mathcal{P}'$ to solve $\text{EQ}_{t'}$. Since, the deterministic worst-partition communication complexity of $\text{EQ}_{t'}$ is $\Omega(n)$, so is the communication complexity of $\mathcal{P}$. This shows that the deterministic magic-partition communication complexity of the matching problem is $\Omega(n)$. □

Note that the above lower bound is asymptotically tight since we can check if there is a perfect matching using $O(n)$ space in the best-order streams: The oracle simply puts edges in the perfect matching first in the stream. Then, the algorithm checks whether the first $n/2$ edges in the stream form a matching by checking whether all vertices appear (using an array of n bits).

Also note that the following argument might lead to a wrong conclusion that the magic-partition communication complexity of DISTINCT is also $\Omega(n)$: If DISTINCT can be done in $o(n)$ bits by a magic-partition protocol, then we can put it in the protocol in Theorem 4.1 to solve the perfect matching problem using $o(n)$ bits. This will contradict Theorem 4.2.

However, the problem of the above argument is that the protocol in Theorem 4.1 needs the *worst-partition* communication complexity of DISTINCT. In fact, DISTINCT can be easily solved in 1 bit using the following magic-partition communication complexity protocol: The oracle sends the first $n/2$ smallest numbers to Alice and sends the rest to Bob. Alice sends 1 to Bob if her numbers are $1, 2, ..., n/2$ and Bob outputs YES if he receives 1 from Alice and his numbers are $n/2 + 1, n/2 + 2..., n$.

5. Graph Connectivity

Graph connectivity is perhaps the most basic property that one would like to check. However, even graph connectivity does not admit space-efficient algorithms in the traditional worst-order streaming model as there is an $\Omega(n)$ lower bound for randomized algorithms. To contrast this, we show that allowing the algorithm the additional power of requesting the input in a specific order allows for a very efficient, $O((\log n)^2)$-space algorithm for testing connectivity.

Problem. We consider a function where the input is a set of edges and $f(e_1, e_2, ..., e_m) = 1$ if and only if G is connected. As usual, let n be the number of vertices of G. As before, we assume that vertices are labeled $1, 2, 3, ..., n$.

5.1. Upper Bound

We will prove the following theorem.

Theorem 5.1. *Graph connectivity can be solved by a randomized algorithm using $O((\log n)^2)$ space in the best-order streaming model.*

Proof. We use the following lemma constructively.

Lemma 5.2. *For any graph G of $n-1$ edges, where $n \geq 3$, G is connected if and only if there exists a vertex v and trees $T_1, T_2, ..., T_q$ such that for all i,*

- $\bigcup_{i=1}^{q} V(T_i) = V(G)$ *and for any* $i \neq j$, $V(T_i) \cap V(T_j) = \{v\}$, *and*
- *there exists a unique vertex* $u_i \in V(T_i)$ *such that* $u_i v \in E(T_i)$, *and*
- $|V(T_i)| \leq \lceil 2n/3 \rceil$ *for all* i.

Proof. To see this proof, notice that G is a spanning tree since it is connected and has exactly $n-1$ edges. Consider any vertex in the spanning tree, which on deleting, disconnects the graph in to two or more pieces such that each piece has at most $2n/3$ vertices. The existence of such a vertex can be proven by induction. The base case where $n = 3$ can be proved simply by picking a vertex in the middle of the path of length 3. Suppose such a vertex exists for all $n' < n$. Consider a tree on n vertices. Now, remove a leaf node, say z, and, by induction, such a vertex v exists on the tree on remaining

n vertices. Now add z back and let C be the component (on deleting v) that contains z. If C has size at most $\lceil 2n/3 \rceil - 1$, the same v works on the larger tree. If C has size at least $\lceil 2n/3 \rceil$ then consider the unique vertex in this component that connects to v, say u. Observed that u serves as the new vertex for the lemma. This is because the complement of C together with u has size at most $n - (\lceil 2n/3 \rceil) + 1 \leq 2n/3$. Since this can be done, u can be chosen as a vertex for the lemma. Whenever a vertex in a tree is disconnected, the new components also form trees. Call these $T_1, T_2, \ldots T_q$. Notice that there can be at most one vertex adjacent to v in each component T_i, since G has no cycles. Call this vertex in T_i by u_i. Therefore each $u_i \in T_i$ is adjacent to u and each T_i has at most $\lceil 2n/3 \rceil$ nodes. □

It is sufficient to consider graphs of $n-1$ edges, as these $n-1$ edges that form a connected spanning component are sufficient to verify connectivity. Suppose that G is connected, i.e., G is a tree. Let v and $T_1, T_2, ..., T_q$ be as in the lemma. Define the order of G to be

$$Order(G) = vu_1, Order(T_1'), vu_2, Order(T_2'), ..., vu_q, Order(T_q')$$

where $T_i' = T_i \setminus \{vu_i\}$. Note that T_i' is a connected tree and so we present edges of T_i' recursively. The recursion step ends when the eventual subtree is a star, i.e., edges presented are $vu_1, vu_2, \ldots$. At this point, the verifier just checks that all consecutive edges are adjacent to the same vertex and form a star. This depth of recursion can be checked directly.

Now, when edges are presented in this order, the checker can check if the graph is connected as follows. First, the checker reads vu_1. The checker remembers the vertex v, which takes $O(\log n)$ bits. Then the edges in T_1' are presented. He checks if T_1' is connected by running the algorithm recursively. Note that he stops checking T_1' once he sees vu_2. Notice that this step is consistent since the vertex v does not appear in any T_i'. Once an edge with a vertex v is received, the checker knows that the tree has been verified and the next tree is to be presented. So the checker repeats with vu_2 and T_2' and so on. Here again, v does not appear in T_2' but u_2 does. Therefore, the checker now again needs to check that T_2' is connected. Further, it is automatically checked that T_2' connects to v due to the edge vu_2. The checker proceeds in this manner checking the connectivity of each T_i' up to $i = q$. If each tree is connected (which is checked recursively), and all the edges vu_i appear separating the trees, then all the trees are connected to v. Therefore, the entire set of edges presented is connected. However, this does not guarantee

that n distinct vertices, or n distinct edges have been received. Therefore, it only remains to be checked that n distinct edges have been presented.

He does so by applying the result in Section 3 once to each vertex v used as a root (as in above) and all leaf nodes of the tree. If all n distinct vertices have appeared at least once, and the set of first n edges form a connected component, then G is a connected graph. Also note that if G is not connected then such ordering cannot be made and the algorithm above will detect this fact.

The space needed is for vu_i and for checking T_i'. I.e., $space(|G|) = space(\max_i |T_i|) + O(\log n)$. That is, $space(n) \leq space(\lceil 2n/3 \rceil) + O(\log n)$. This gives the claimed space bound. □

5.2. Lower Bound

Recall that we say that the input is ordered in an explicit way if each edge is presented in the form (a, b) where a and b are the labels of its end vertices.

Theorem 5.3. *If the input can be ordered only in an explicit way, any deterministic algorithm solving the connectivity problem on the best-order stream needs $\Omega(n \log n)$ space, where n is the number of vertices.*

PROOF. Let n be an odd number. We show that the theorem holds even when the input always consists of exactly $n - 1$ edges. (Therefore, the task is only to check whether the input edges form a spanning tree over n nodes.) We show this via the magic-partition communication complexity. Since the argument is essentially the same as that in the the proof of Theorem 4.2, we only give the essential parts here.

Assume that n is an odd number more than two. Let $g(n)$ be the number of spanning trees of the complete graph K_n. By Cayley's formula (see, e.g., [38]),

$$g(n) = n^{n-2}.$$

Let $T_1, T_2, \ldots, T_{g(n)}$ denote such trees. Consider any best-partition communication complexity protocol which consists of a partition function $\mathcal{O}$ owned by the oracle and a protocol $\mathcal{P}$ used by Alice and Bob. For $i = 1, 2, \ldots, g(n)$, let (A_i, B_i) be the partition of the input edges of T_i to Alice and Bob, respectively, according to $\mathcal{O}$.

Now, draw a graph H consisting of $g(n)$ vertices, $v_1, v_2, \ldots, v_{g(n)}$. Draw an edge between vertices v_i and v_j if $A_i \cup B_j$ or $A_j \cup B_i$ is a spanning tree.

We claim that each vertex in H has degree at most $2g((n+1)/2)$. To see this, observe that for any set A of $(n-1)/2$ edges, there are at most $g((n+1)/2)$ sets B of $(n-1)/2$ edges such that $A \cup B$ is a spanning tree. This is because when we contract edges in A, there are $(n+1)/2$ vertices left and these vertices must form a spanning tree on the contracted graph. This observation is also true when we look at the set B. The claim thus follows.

Now pick an independent set from H using the following algorithm: Pick any vertex, delete such vertex and its neighbors and then repeat. Observe that this algorithm gives an independent set of size at least $\frac{g(n)}{2g((n+1)/2)+1}$ since there are $g(n)$ vertices in H and each vertex has degree at most $2g((n+1)/2)$. Let t be the size of the independent set. Note that $t = \frac{g(n)}{2g((n+1)/2)+1} = n^{\Omega(n)}$. Let $v_{i_1}, v_{i_2}, \ldots, v_{i_t}$ be the vertices in the independent set.

Consider the trees $T_{i_1}, T_{i_2}, \ldots, T_{i_t}$ corresponding to the independent set picked by the above algorithm. Since there is no edge between any $v_{i_{t'}}$ and $v_{i_{t''}}$, $(A_{i_{t'}}, B_{i_{t''}})$ and $(A_{i_{t''}}, B_{i_{t'}})$ do not form a spanning tree. As argued in the proof of Theorem 4.2, the protocol $\mathcal{P}$ must be able to receive any pair of the form $(A_{i_{t'}}, B_{i_{t''}})$ and answer YES if and only if $t' = t''$. By the fooling set argument or the reduction from $\text{EQ}_{\log t}$, it follows that $\mathcal{P}$ needs $\Omega(\log t) = \Omega(n \log n)$ bits, as desired. □

We note that the lower bound above is asymptotically tight since we can solve connectivity problem using the following $O(n \log n)$-space deterministic algorithm: The oracle present edges in a spanning tree first in the stream. Then the algorithm reads and checks whether these edges form a spanning tree using $O(n \log n)$ space.

6. Further Results

The previous sections give us a flavor of the results that can be obtained in the best-order streaming model. We describe a few more and mention the intuition behind the protocol without going into details since the techniques are essentially the same.

6.1. Bipartite k-Regular graph

The problem is to check if the graph is bipartite k-regular. First, note that since this problem is the generalization of the perfect matching problem (cf. Section 4), the $\Omega(n)$ lower bound of the deterministic algorithms holds here.

Now we show that this problem can be solved by a randomized algorithm with $O(\log n)$ space.

The point of the algorithm is that a k-regular bipartite graph can be decomposed into k disjoint perfect matchings. So the oracle can do this and present each of the perfect matchings one after the other. However, as it will be clear soon, the oracle has to send each edge in the form (a, b) where a is the "left" vertex and b is the "right" one. This forces the algorithm to find another way to find out the value of n (instead of looking for a "flip" edge as used by the perfect matching algorithm).

The algorithm can find out n in the following way: While reading the first perfect matching, it remembers the maximum vertex label it saw so far, denoted by n'. Once the number of edges it read so far equals $n'/2$ (for the current value of n'), it looks one more edge further. If this edge consists of vertices with labels at most n' then it concludes that this value of n' is the value of n. The correctness of this method can be seen by observing that if $n' < n$ and no vertex appears twice in the first $n'/2$ edges then the labels of vertices in the next edge must be both more than n'.

Now, we describe the last part of the algorithm. It has to verify the following.

1. Each set of $n/2$ edges form a perfect matching. This can be verified separately for each set of $n/2$ edges.
2. In each matching, it sees the same set of "left" vertices and "right" vertices. This can be done by computing the finger prints of the sets of left and right vertices.

Note that the reason that the oracle has to present the edges in the form of left and right vertices is to allow the algorithm to check the second condition.

6.2. Hamiltonian Cycle

The problem is to check whether the input graph has a Hamiltonian cycle. We claim that this problem is in RSTREAM$(1, \log n)$. The intuition is for the oracle to provide the Hamiltonian cycle first (everything else is ignored). The algorithm then checks if the first n edges indeed form a cycle; this requires two main facts. First that every two consecutive edges share a vertex, and the n-th edge shares a specific vertex with the first. This fact can be easily checked. The second key step is to check that these edges indeed span all n vertices (and not go through same vertex more than once). This can be done by the fingerprinting technique.

We also claim that there is an $\Omega(n)$ lower bound for the deterministic algorithms if the edges can be ordered only in an explicit way. The proof is essentially similar to the proof of other lower bounds shown earlier and we only sketch it here. We consider the inputs that have exactly n edges. Let $g(n)$ be the number of the Hamiltonian cycles covering n vertices. Clearly $g(n) = (n-1)!$. Let $C_1, \ldots C_{g(n)}$ be these cycles and $(A_1, B_1), \ldots, (A_{g(n)}, B_{g(n)})$ be the corresponding partitions. Since after we see $n/2$ edges, there could be only $g(n/2)$ possible Hamiltonian cycles containing these edges, we can pick $\frac{g(n)}{2g(n/2)+1}$ that are "independent" in the same sense as in Theorem 5.3. The communication complexity is thus

$$\Omega\left(\log\left(\frac{g(n)}{2g(n/2)+1}\right)\right) = \Omega(n)\,.$$

6.3. Non-Bipartiteness

The problem is to check if the graph is not bipartite. This problem can be solved by a *deterministic* algorithm with $O(\log n)$ space by having an oracle present an odd length (not necessarily simple) cycle. Verifying that this is indeed a cycle and that it is of odd length can be done easily.

In contrast to the Non-bipartiteness problem, we do not have an algorithm for checking the *bipartiteness* of graphs. We conjecture that this problem has a super-logarithmic randomized lower bound. We note, however, that if we relax the model by allowing the input to be presented twice then there is an efficient randomized algorithm: Let $U = \{u_1, u_2, \ldots, u_{n'}\}$ and $V = \{v_1, v_2, \ldots, v_{n''}$ be the two partitions. In the first rounds, present edges incident to u_1 first then present edges incident to u_2, and so on (i.e., edges are presented in the form (u_1, v_{i_1}), (u_1, v_{i_2}), ..., $(u_2, v_{i'_1})$, $(u_2, v_{i'_2})$, ..., $(u_{n'}, v_{i''_1})$, ...) Similarly, in the next round present edges incident to v_1, v_2, ..., $v_{n''}$, respectively, with vertices in V appearing first (i.e., in the form $(v_1, u_{i_1}), (v_1, u_{i_2}), \ldots$). We can use the fingerprinting technique to check if $u_1, \ldots, u_{n'}, v_1, \ldots, v_{n''}$ are all distinct.

7. Conclusions

This paper describes a new model of stream checking that lies at the intersection of several extremely well-studied and foundational fields of computer science. Specifically, the model connects several settings related to proof checking, communication complexity, and streaming algorithms. The

motivation of this paper, however, arises from the recent growth in the data sizes and the advent of the powerful cloud computing architectures and services. The question we ask is, can the verification of certain properties (on any input) be accompanied with a streaming proof of the fact? The checker should be able to verify that the prover is not cheating. We show that if the checker (or the algorithm in the best-order streaming setting) is given the power of choosing a specific rule for the prover to send the input, then many problems can be solved much more efficiently in this model than in the previous models.

While non-obvious, our algorithms and proofs are fairly simple. However, the nice aspect is that it uses several interesting techniques from many areas such as fingerprinting and covert channels. Fingerprinting is used in a crucial way to randomly test for the distinctness of a set of elements presented as a stream. The protocol between the prover and the checker also allows for a covert communication (which gives the covert channels a positive spin as opposed to the previous studies in security and cryptography). While the prover is only allowed to send the re-ordered input, the prover is able to encode some extra bits of information with the special ordering requested by the checker. The difficulty in most of our proof techniques is in how the checker or algorithm verifies that the prover or oracle is sending the input order as requested.

We have given randomized $O(\text{polylog}\, n)$-space algorithms for problems that previously, in the streaming model, had no sub-linear space algorithms. We note that in all protocols presented in this paper, the prover can construct the best-order proofs in polynomial time. There are still a lot of problems in graph theory that remain to be investigated. A nice direction is to consider testing for graph minors, which could in turn yield efficient methods for testing planarity and other properties that exclude specific minors. It is also interesting to see whether all graph problems in the complexity class P can be solved in our model with $O(\text{polylog}\, n)$ space. Such a result would be a huge improvement over the result in [29] (which needs a proof of size near-linear in the number of steps for the computation) in terms of the proof size for graph problems. (One good starting point are problems of checking bipartiteness, non-connectivity, and non-existence of perfect matching.) Moreover, it is interesting to see whether additional passes would be of much help. Additionally, is the "flipping trick" necessary? That is, if we present each edge as a set $\{u, v\}$ instead of an ordered pair (u, v), do efficient protocols for the problems presented here still exist?

Apart from the study of our specific model, we believe that the results and ideas presented in this paper could lead to improved algorithms in the previously studied settings as well as yield new insights to the complexity of the problems.

Acknowledgment: We would like to thank Justin Thaler and anonymous reviewers for useful comments.

References

[1] Amazon elastic compute cloud (amazon ec2).

[2] F. Li, K. Yi, M. Hadjieleftheriou, G. Kollios, Proof-infused streams: Enabling authentication of sliding window queries on streams, in: VLDB, 2007, pp. 147–158.

[3] K. Yi, F. Li, M. Hadjieleftheriou, G. Kollios, D. Srivastava, Randomized synopses for query assurance on data streams, in: ICDE, 2008, pp. 416–425.

[4] S. Papadopoulos, Y. Yang, D. Papadias, Cads: Continuous authentication on data streams, in: VLDB, 2007, pp. 135–146.

[5] S. Goldwasser, Y. T. Kalai, G. N. Rothblum, Delegating computation: interactive proofs for muggles, in: STOC, 2008, pp. 113–122.

[6] A. Chakrabarti, G. Cormode, A. McGregor, Annotations in data streams, in: ICALP (1), 2009, pp. 222–234.

[7] G. Cormode, M. Mitzenmacher, J. Thaler, Streaming graph computations with a helpful advisor, CoRR abs/1004.2899.

[8] N. Alon, Y. Matias, M. Szegedy, The space complexity of approximating the frequency moments, J. Comput. Syst. Sci. 58 (1) (1999) 137–147, also appeared in STOC'96.

[9] M. Sipser, Introduction to the Theory of Computation, International Thomson Publishing, 1996.

[10] A. Chakrabarti, T. S. Jayram, M. Patrascu, Tight lower bounds for selection in randomly ordered streams, in: SODA, 2008, pp. 720–729.

[11] S. Guha, A. McGregor, Approximate quantiles and the order of the stream, in: PODS, 2006, pp. 273–279.

[12] S. Guha, A. Mcgregor, Lower bounds for quantile estimation in random-order and multi-pass streaming, in: ICALP, 2007, pp. 704–715.

[13] J. I. Munro, M. Paterson, Selection and sorting with limited storage, Theor. Comput. Sci. 12 (1980) 315–323, also appeared in FOCS'78.

[14] M. R. Henzinger, P. Raghavan, S. Rajagopalan, Computing on data streams (1999) 107–118.

[15] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, J. Zhang, Graph distances in the streaming model: the value of space, in: SODA, 2005, pp. 745–754.

[16] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, J. Zhang, On graph problems in a semi-streaming model, Theor. Comput. Sci. 348 (2) (2005) 207–216.

[17] G. Aggarwal, M. Datar, S. Rajagopalan, M. Ruhl, On the streaming model augmented with a sorting primitive, FOCS (2004) 540–549.

[18] D. Knuth, The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams.

[19] J. M. Ruhl, Efficient algorithms for new computational models, Ph.D. thesis, supervisor-David R. Karger (2003).

[20] C. Demetrescu, I. Finocchi, A. Ribichini, Trading off space for passes in graph streaming problems, in: SODA, 2006, pp. 714–723.

[21] A. C.-C. Yao, Some complexity questions related to distributive computing (preliminary report), in: STOC, 1979, pp. 209–213.

[22] E. Kushilevitz, N. Nisan, Communication complexity, Cambridge University Press, New York, NY, USA, 1997.

[23] T. W. Lam, W. L. Ruzzo, Results on communication complexity classes, J. Comput. Syst. Sci. 44 (2) (1992) 324–342, also appeared in Structure in Complexity Theory Conference 1989.

[24] C. H. Papadimitriou, M. Sipser, Communication complexity, J. Comput. Syst. Sci. 28 (2) (1984) 260–269, also appeared in STOC'82.

[25] A. Chakrabarti, G. Cormode, A. McGregor, Robust lower bounds for communication and stream computation, in: STOC, 2008, pp. 641–650.

[26] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy, Proof verification and the hardness of approximation problems, J. ACM 45 (3) (1998) 501–555.

[27] S. Arora, S. Safra, Probabilistic checking of proofs: A new characterization of NP, J. ACM 45 (1) (1998) 70–122, also appeared in FOCS'92.

[28] I. Dinur, The PCP theorem by gap amplification, J. ACM 54 (3) (2007) 12, also appeared in STOC'06.

[29] R. J. Lipton, Efficient checking of computations, in: STACS, 1990, pp. 207–215.

[30] J. Feigenbaum, S. Kannan, M. Strauss, M. Viswanathan, Testing and spot-checking of data streams, in: SODA, 2000, pp. 165–174.

[31] O. Goldreich, Property testing in massive graphs (2002) 123–147.

[32] T. Lengauer, VLSI theory, in: Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A), 1990, pp. 835–868.

[33] J. I. Chu, G. Schnitger, The communication complexity of several problems in matrix computation, J. Complexity 7 (4) (1991) 395–407.

[34] A. Hajnal, W. Maass, G. Turán, On the communication complexity of graph properties, in: STOC, 1988, pp. 186–191.

[35] J. Håstad, A. Wigderson, The randomized communication complexity of set disjointness, Theory of Computing 3 (1) (2007) 211–219.

[36] R. J. Lipton, Fingerprinting sets, Cs-tr-212-89, Princton University (1989).

[37] P. Gopalan, J. Radhakrishnan, Finding duplicates in a data stream, in: SODA, 2009, pp. 402–411.

[38] M. Aigner, G. M. Ziegler, Proofs from THE BOOK, 3rd Edition, Springer, 2003.