

Best-Order Streaming model

Atish Das Sarma, Richard J. Lipton, and Danupon Nanongkai

Georgia Institute of Technology, atish,rjl,danupon@cc.gatech.edu

Abstract. We study a new model of computation called *stream checking* on graph problems 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 nothing but a reordering of the input data. This model has a close relationship 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 checking if a graph has a perfect matching is impossible to do deterministically using small space. To contrast this, we show that randomized verifiers are powerful enough to check whether a graph has a perfect matching or is connected.

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 hardness of a communication problem holds for 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 these questions, we first describe one 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.¹ 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 different order). This is the model considered in this paper.

Coincidentally, this model has connections with many previously studied models in many areas. We now continue with describing 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 called a *stream*. In certain settings, it is acceptable to allow the algorithm to perform multiple passes over the stream. However, for many applications, it is not feasible to perform more than a single pass. The general streaming algorithms framework has been studied extensively since the seminal work of Alon, Matias, Szegedy [3].

Models diverge in the assumptions made about what order the algorithm can access the input elements. The most stringent restriction on the algorithm is to assume that the input sequence is presented to the

¹ http://www.ebizq.net/blogs/saasweek/2008/03/distinguishing_cloud_computing/.

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 [7, 15, 16]. However, even a simple problem like finding median, which was considered in the earliest paper in the area by Munro and Patterson [24], in both input orders, was shown recently [7] to require many passes even when the input is in a random order (to be precise, any $O(\text{polylog } n)$ algorithm requires $\Omega(\log \log n)$ passes). This might be undesirable.

More bad news: Graph problems are extremely hard when presented in an adversarial order. In [19], 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 [11] that most graph problems cannot be solved efficiently in a few passes. Since then, new models have been proposed to overcome this obstruction. Feigenbaum et. al. [12] proposed a relaxation of the memory restriction in what is called the semi-stream model. Aggarwal et. al. [2] proposed that if the algorithm has a power to sort the stream in one pass then it is easier to solve some graph problems (although not in one or constant passes). Another model that has been considered is the W-Stream (write-stream) model [26, 8]. 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 opted 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 an 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.

It would be nice to obtain a characterization of problems that can be solved by a poly-log space, single pass, best-order stream algorithm. Studying this model, like all other related streaming models, is likely to yield new insights and might lead to an improvement of worst case analysis and an adjustment of models.

Communication Complexity: Another closely related model is the communication complexity model [27, 20]. This model was extensively studied and found many applications in many areas. 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 [21] and the best-case [25] 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, there is a study for another variation where the input is partitioned according to some known distribution (see, e.g., [6]). The main question is whether the hardness 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 hardness 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.

We briefly describe some work in the field of proof checking and its relation to our setting. The field of probabilistically checkable proofs (PCPs) [4, 5, 9] deals with 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 [23] where it showed 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. In other words, this result almost answers our question except that the proof is not the reordered input.

Another related result that compares streaming model with other models is by Feigenbaum et. al. [10] 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 [13].

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 factors.

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 an evidence of a positive answer for the randomized case. Our positive results are similar in spirit to W-stream and Sort-stream papers [2, 8, 26].

For the negative answer, we show that the space requirement is too large even for a simple answer of checking if a given graph has a perfect matching deterministically. In contrast, this problem, as well as the connectivity problem, can be solved efficiently by randomized algorithms.

Organization: The rest of the paper is organized as follow. In Section 2 we describe our stream proof checking 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. While we are unable to mention all details in this paper due to space limitations, 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 subsequent sections.

2.1 Stream Proof Model

Recall the streaming model where an input is in some order e_1, e_2, \dots, e_m where m is the size of the input. Consider any function f that maps these input stream to $\{0, 1\}$. The goal of the typical one-pass streaming model is to calculate f using the smallest amount of memory possible.

In the *stream proof* model, we consider any function f that is order-independent. Our main question is how much space a one-pass streaming algorithm needs to compute f if the input is provided in the best order. Formally, for any function s of m and any function f , we say that a language L determined by f is in the class $\text{STREAM-PROOF}(s(m))$ if there exists an algorithm \mathcal{A} using space at most $s(m)$ such that if

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

The other way to see this model is to consider the situation where there are two players in the setting, *prover* and *verifier*. The job of the prover is to provide the stream in some order so that the verifier can compute f using 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.

The model above can be generalized to the following.

- $\text{STREAM}(p, s)$: A class of problems that, when presented with best-order, can be checked by a deterministic streaming algorithm \mathcal{A} using p passes $O(s)$ space.
- $\text{RSTREAM}(p, s)$: A class of problems that, when presented with best-order, can be checked by a randomized streaming algorithm \mathcal{A} using p passes $O(s)$ space and with correct probability more than $1/2$.

It is important to point out that when the input is presented in a specified order, we still need to check that the adversary 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 well-studied communication complexity models, we first define a new communication complexity model, magic-partition, that closely relates to our proof checking model.

2.2 Magic-Partition Communication Complexity

In this subsection, we define magic-partition communication complexity which will be the main tool to prove the lower bound of the best-order streaming model.

Recall that in the standard 2-player communication complexity, Alice and Bob gets input x and y and want to compute $f(x, y)$. We usually consider when the input is partitioned in an adversarial order, i.e., we partition 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 consider the case when x and y are partitioned in the best way possible. 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 smallest number of communicated bits. We restrict that the input data must be divided equally between Alice and Bob.

Let us consider an example. Suppose the input is a graph G . Alice and Bob might decide that the graph be broken down in topological sort order, and Alice receives the first half of the total edges, starting with edges incident on the vertices (traversing them in topological order). 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.

The following lemma is the key to prove our lower bound results.

Lemma 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 $(2p - 1)t < s$, $f \notin \text{STREAM}(p, t)$.*

Proof. Suppose that the lemma is not true; i.e., f has 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, \dots, e_n . Let π be a permutation such that $e_{\pi(1)}, e_{\pi(2)}, \dots, e_{\pi(n)}$ is the best ordering of the input for \mathcal{A} . Then, define the partition of the magic-partition communication complexity by allocating $e_{\pi(1)}, e_{\pi(2)}, \dots, e_{\pi(\lfloor n/2 \rfloor)}$ to Alice and the rest to Bob.

Alice and Bob can simulate \mathcal{A} as follows. First, Alice simulates \mathcal{A} on $e_{\pi(1)}, e_{\pi(2)}, \dots, e_{\pi(\lfloor n/2 \rfloor)}$. Then, she sends the data on memory to Bob. Then, Bob continues simulating \mathcal{A} using data given by Alice (as if he simulates \mathcal{A} on $e_{\pi(1)}, e_{\pi(2)}, \dots, e_{\pi(\lfloor n/2 \rfloor)}$ 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 need $2p - 1$ 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. \square

Note that this type of communication complexity should not be confused with the best-partition communication complexity (defined below). Also, the converse of the above lemma clearly does not hold.

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

2.3 Related models

Best-case partition Communication Complexity For this model, Alice and Bob can pick how to divide the data among them (must be half-half) *before* they see the input. Then, the adversary gives an input that makes them communicate the most.

This model was introduced by Papadimitriou and Sipser [25] and heavily used for proving lower bounds for many applications (see [20] and references therein).

Similar to the best-partition communication complexity, this model makes many problems easier to solve than the traditional worst case communication complexity where the worst case input is assumed. One example is the set disjointness problem. In this problem, two n -bit vectors x and y that is a characteristic vector of two sets X and Y are given. Alice and Bob have to determine if $X \cap Y = \emptyset$.

In the worst case communication complexity, it is 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 this way: $x_1, y_1, x_2, y_2, \dots, x_{n/2}, y_{n/2}$ go to Alice and the rest go to Bob. This way, each of them can check the disjointness separately.

We note that this model is different from the magic-partition model in that, 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 decide how to divide an input *after* he sees it).

One problem that separates these model is the connectivity problem. Hajnal et al. [17] showed that the best-case 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.

Nondeterministic Communication Complexity Alice and Bob receives 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 small amount of communication.

Example: $f(x, y) = 1$ if $x \neq y$ where x and y are n -bit strings. The proof is simply the number i where $x_i \neq y_i$. Alice and Bob can check the proof by exchanging x_i and y_i . If $x = y$ then there is no proof and Alice and Bob can always detect the fake proof.

This model is different from our model because our model has no proof but the oracle's job is to help Alice and Bob find the answer (whether $f(x, y)$ is 0 or 1) by appropriately partitioning the input.

3 Detecting duplicate and Checking distinctness

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

Our goal in this section is to find a one-pass algorithm for this problem. An algorithm for this problem will be an important ingredient of all algorithm 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 error probability $\frac{1}{n}$.

3.1 Space lower bound of deterministic algorithms

Since checking for distinctness is equivalent to checking if there is a duplicate, a natural problem to use 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.

For this problem, a set $X \subseteq N$ is given to Alice and $Y \subseteq N$ is given to Bob where $N = \{1, 2, 3, \dots, n\}$ and $|X| + |Y| = n$.²

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

Theorem 1. *The communication complexity of F-DISJ is $\Omega(n)$.*

Proof. Consider the fooling set $F = \{(A, \bar{A}) : \forall A \subseteq N\}$. Since $|F| = 2^n$, the number of bits needed to sent between Alice and Bob is at least $\log |F| = \Omega(n)$. \square

The communication complexity lower bound of F-DISJ implies the space lower bound of DISTINCT.

Corollary 1. *Any deterministic algorithm for DISTINCT needs $\Omega(n)$ space.*

This lower bound is for worst-case input. The reason we mention this here is because this is an inherent difficulty in our algorithms. Our randomized algorithms use randomness only to get around this step of checking distinctness.

3.2 Randomized algorithm

In this subsection we present a randomized one-pass algorithm that solves this problem using $O(\log n)$ space. This algorithm is based on the *Fingerprinting Sets* technique introduced by Lipton [22, 23]. Roughly speaking, given a multi-set $\{x_1, x_2, \dots, x_k\}$, its *fingerprint* is defined to be

$$\prod_{i=1}^k (x_i + r) \pmod{p}$$

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

Theorem 2. [23] *Let $\{x_1, x_2, \dots, x_k\}$ and $\{y_1, y_2, \dots, y_k\}$ 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*

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

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, \dots, a_n are all distinct, we simply check if the fingerprints of $\{a_1, a_2, \dots, a_n\}$ and $\{1, 2, \dots, n\}$ 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 can be used in our motivating application above. That is, when the cloud compute 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 using the fingerprinting set. This enables us to concentrate on checking the stream without worrying about this issue in the rest of the paper.

We also note that the recent result by Gopalan et al. [14] can be modified to solve DISTINCT as well.

² 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 $\Omega(n)$ randomized algorithm while the F-DISJ has an $O(\log n)$ randomized protocol (shown in the next section). We also note that the lower bound of another related problem called k -disjointness problem ([20, example 2.12] and [18]) does not imply our result neither.

4 Perfect Matching

This section is devoted to the study of perfect matchings. We discuss lower bounds as well as upper bounds.

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

We now present the main upper bound of this section and follow it up with the checking protocol in the proof.

Theorem 3. *The problem of determining if there exists a perfect matching can be solved by a randomized algorithm in $O(\log n)$ space best-order stream checking.*

Proof. Protocol: The prover sends $n/2$ edges of a perfect matching to the verifier, followed by the “sign” which can be implemented by flipping the order of vertices in the last edge. Then the prover sends the rest edges. The verifier has to check three things.

1. Find out n by counting the number of edges before the “sign” is given.
2. Check if the first $n/2$ edges form a perfect matching. This can be done by checking if the sum of the labels of vertices in the first $n/2$ edges equals $1 + 2 + 3 + \dots + n$.
3. Check if there are n vertices. This is done by checking that the maximum vertex label is at most n .

The verifier outputs 1 if the input passes all the above tests. The correctness of this protocol is straightforward. \square

In the next subsection, we present a lower bound.

4.1 Hardness

We show that deterministic algorithms have $\Omega(n)$ lower bound if the input is reordered 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 by a reduction from the magic-partition communication complexity (cf. Section 2) of the same problem by using Lemma 1.

Theorem 4. *If the input can be reordered only in an explicit way then any deterministic algorithm solving the perfect matching problem needs $\Omega(n)$ space.*

Proof. Let n be an even integer. Let $g(n)$ denote the number of matchings in the complete graph K_n . Observe that $g(n) = \frac{n!}{(n/2)!2^{n/2}}$. Denote these matchings by $M_1, M_2, \dots, M_{g(n)}$. Let \mathcal{P} be any magic-partition protocol. For any integer i , let A_i and B_i be the best partition of M_i according to \mathcal{P} (A_i and B_i are sent to Alice and Bob respectively). Observe that for any i , there are at most $g(n/2)^2$ matchings that vertices are partitioned the same way as M_i . (I.e., if we define $C_i = \{v \in V \mid \exists e \in A_i \text{ s.t. } v \in e\}$ then for any $i, |\{j \mid C_i = C_j\}| \leq g(n/2)^2$.) This is because $n/2$ vertices on each side of the partition can make $g(n/2)$ different matchings.

Therefore, the number of matchings such that the vertices are divided differently is at least

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

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

In particular, if we let M_{i_1}, \dots, M_{i_t} , where $t = \binom{n/2}{n/4}$, be such matchings then for any $j \neq k$, (A_{i_j}, B_{i_k}) is not a perfect matching. Now, let $t' = \log t$. 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 1 if $x = y$ and 0 otherwise. By [20, example 1.21], $D(\text{EQ}_{t'}) \geq t' + 1 = \Omega(n)$.

Now we reduce $\text{EQ}_{t'}$ to our problem: Map x to M_{i_x} and y to M_{i_y} . Now, $x = y$ if and only if (M_{i_x}, M_{i_y}) is a perfect matching. This shows that the magic-partition communication complexity of the matching problem is $\Omega(n)$. \square

Note the the above lower bound is asymptotically tight since there is an obvious protocol where Alice sends Bob all vertices she has (using $O(n)$ bits of communication).

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 traditional streaming models. 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 very efficient, $O((\log n)^2)$ space algorithms for testing connectivity.

Problem: We consider a function where the input is a set of edges and $f(e_1, e_2, \dots, e_m) = 1$ 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 $1, 2, 3, \dots, n$.

We will prove the following theorem.

Theorem 5. *Graph connectivity can be solved by a randomized algorithm in $O((\log n)^2)$ space best-order stream checking.*

Proof. We use the following lemma which is easy to prove.

Lemma 2. *For any graph G of n edges, G is connected if and only if there exists a vertex v and trees T_1, T_2, \dots, T_q such that for all i ,*

- *there exists a unique vertex $u_i \in V(T_i)$ such that $vu_i \in E(T_i)$, and*
- *$|V(T_i)| \leq 2n/3$ for all i .*

Suppose G is connected, i.e., G is a tree. Let v and T_1, T_2, \dots, T_q be as in the lemma. Define the order of G to be

$$\text{Order}(G) = vu_1, \text{Order}(T'_1), vu_2, \text{Order}(T'_2), \dots, vu_q, \text{Order}(T'_q)$$

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.

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 . He checks if T'_1 is connected by running the algorithm recursively. Note that he stops checking T'_1 once he sees vu_2 . Next, he repeats with vu_2 and T'_2 and so on.

The space needed is for vu_i and for checking T'_i . I.e., $\text{space}(|G|) = \text{space}(\max_i |T_i|) + O(\log n)$. That is, $\text{space}(n) \leq \text{space}(2n/3) + O(\log n)$. This gives the claimed space bound.

Note that the checker has to make sure every vertex appears in the graph. He does so by applying result in Section 3 once to each vertex v used as a root (as in above) and all leaf nodes of the tree. Also note that if G is not connected then such ordering cannot be made and the algorithm above will detect this fact. \square

6 Further Results

The previous sections give us a flavor of the results that can be obtained in this model. We describe a few more and mention the intuition behind the protocol (without giving details, due to space constraints).

6.1 Bipartite k -Regular graph

The point is that a k -regular bipartite graph can be decomposed into k disjoint sets of perfect matchings. So the adversary can do this and present each of the perfect matchings one after the other. Now our previously described algorithm can be used to verify each perfect matching. In addition, a fairly simple algorithm can take care of verifying that we indeed receive k different sets (and to also know when one perfect matching ends and the new one is presented).

6.2 Hamiltonian cycle

It can be shown that $\text{HAMILTONIAN-CYCLE} \in \text{RSTREAM}(1, \log n)$. The intuition is for the protocol to request the hamiltonian cycle first (everything else is ignored). The checker then checks if the first n edges presented 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 is easy. 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 using the set distinctness approach.

6.3 Non-bipartiteness

Non-bipartiteness of graphs can again be checked in our model by requesting the adversary to present an odd length cycle. Verifying that this is indeed a cycle and that it is of odd length is again done in a manner very similar to verifying hamiltonian cycle.

We do not have an algorithm to verify general *bipartiteness* of graphs and leave it as an open question.

7 Conclusions

This paper describes a new model of stream checking, that lies at the intersection of extremely well-studied and foundational fields of computer science. Specifically, the model connects several settings relating to proof checking, communication complexity, and streaming algorithms. The motivation for this paper, however, draws from recent growth in data sizes and the advent of powerful cloud computing architectures and services. The question we ask is, can 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 algorithm in the streaming algorithms 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 and areas such as fingerprinting, and covert channels. Fingerprinting is used in a crucial way to randomly test for distinctness of a set of elements presented as a stream. The protocol between the prover and check also allows for covert communication (which gives covert channels a positive spin as opposed to previous studies in security and cryptography). While the prover is only allowed to send the input, re-ordered, the prover is able to encode 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 adversary is sending the input order as requested.

We have given $O(\text{polylog } n)$ space algorithms for problems that previously, in the streaming model, had no sub-linear algorithms. 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. We have some work in progress in this direction. 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. 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 previously studied settings as well as yield new insights to the complexity of the problems.

References

1. Amazon elastic compute cloud (amazon ec2).
2. Gagan Aggarwal, Mayur Datar, Sridhar Rajagopalan, and Matthias Ruhl. On the streaming model augmented with a sorting primitive. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:540–549, 2004.
3. Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.
4. Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.
5. Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of np. *J. ACM*, 45(1):70–122, 1998.
6. Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Robust lower bounds for communication and stream computation. In *STOC*, pages 641–650, 2008.
7. Amit Chakrabarti, T.S. Jayram, and Mihai Pătraşcu. Tight lower bounds for selection in randomly ordered streams. In *Proc. 19th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 720–729, 2008.
8. Camil Demetrescu, Irene Finocchi, and Andrea Ribichini. Trading off space for passes in graph streaming problems. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 714–723, New York, NY, USA, 2006. ACM.
9. Irit Dinur. The pcg theorem by gap amplification. *J. ACM*, 54(3):12, 2007.
10. J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. Testing and spot-checking of data streams (extended abstract). In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 165–174, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.
11. Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the streaming model: the value of space. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 745–754, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.
12. Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2):207–216, 2005.
13. Oded Goldreich. Property testing in massive graphs. pages 123–147, 2002.
14. Parikshit Gopalan and Jaikumar Radhakrishnan. Finding duplicates in a data stream. In *SODA '09: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, page To appear, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.
15. Sudipto Guha and Andrew McGregor. Approximate quantiles and the order of the stream. In *PODS '06: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 273–279, New York, NY, USA, 2006. ACM.
16. Sudipto Guha and Andrew McGregor. Lower bounds for quantile estimation in random-order and multi-pass streaming. In *in International Colloquium on Automata, Languages and Programming*, pages 704–715, 2007.
17. András Hajnal, Wolfgang Maass, and György Turán. On the communication complexity of graph properties. In *STOC*, pages 186–191, 1988.
18. Johan Håstad and Avi Wigderson. The randomized communication complexity of set disjointness. *Theory of Computing*, 3(1):211–219, 2007.
19. Monika R. Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. pages 107–118, 1999.
20. Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, New York, NY, USA, 1997.
21. Tak Wah Lam and Walter L. Ruzzo. Results on communication complexity classes. *J. Comput. Syst. Sci.*, 44(2):324–342, 1992.
22. Richard J. Lipton. Fingerprinting sets. Cs-tr-212-89, Princeton University, 1989.
23. Richard J. Lipton. Efficient checking of computations. In *STACS*, pages 207–215, 1990.
24. J. Ian Munro and Mike Paterson. Selection and sorting with limited storage. In *FOCS*, pages 253–258, 1978.
25. Christos H. Papadimitriou and Michael Sipser. Communication complexity. *J. Comput. Syst. Sci.*, 28(2):260–269, 1984.
26. Jan Matthias Ruhl. *Efficient algorithms for new computational models*. PhD thesis, 2003. Supervisor-David R. Karger.
27. Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *STOC*, pages 209–213, 1979.