Acyclicity Tests in Classes of Dense Digraphs in Streaming Model

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
in
Computer Science

by

Madhumita Kundu

[Roll No: CS1823]

under the guidance of

Prof. Sourav Chakraborty & Prof. Saket Saurabh
Professors
Advanced Computing and Microelectronics Unit

Indian Statistical Institute
Kolkata-700108, India

July 2020

Dedicated to my family and my supervisor

Declaration

I hereby declare that the dissertation report entitled **Acyclicity Tests in Classes of Dense Digraphs in Streaming Model** submitted to Indian Statistical Institute, Kolkata, is a bonafide record of work carried out in partial fulfillment for the award of the degree of Master of Technology in Computer Science. The work has been carried out under the guidance of Prof. Sourav Chakraborty, Associate Professor and Prof. Saket Saurabh, Adjunct Professor, ACMU, Indian Statistical Institute, Kolkata.

I further declare that this work is original, composed by myself. The work contained herein is my own except where stated otherwise by reference or acknowledgement, and that this work has not been submitted to any other institution for award of any other degree or professional qualification.

Place: Kolkata
Date: July 10, 2020

Madhumita Kundu
Roll No: CS1823
Indian Statistical Institute
Kolkata - 700108, Kolkata

Acknowledgments

I would like to show my highest gratitude to my advisor, *Prof. Saket Saurabh*, Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, for his guidance and continuous support and encouragement. He has literally taught me how to do good research, and motivated me with great insights and innovative ideas.

I would also like to thank *Prof. Sourav Chakraborty*, Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, for his valuable suggestions and discussions.

It is a pleasure to acknowledge that I got the positive influence in various aspects of my life from *Prof. Arijit Ghosh*.

My deepest thanks to all the teachers of Indian Statistical Institute, for their valuable suggestions and discussions which added an important dimension to my research work.

I am very much thankful to *Fahad Panolan* and *Pranabendu Mishra* for actively participating in the project.

Finally, I am very much thankful to my parents and family for their everlasting supports.

Last but not the least, I would like to thank all of my friends for their help and support. I also thank all those, whom I have missed out from the above list.

Madhumita Kundu
Indian Statistical Institute
Kolkata - 700108 , India.

CERTIFICATE

This is to certify that the dissertation entitled **"Acyclicity Tests in Classes of Dense Digraphs in Streaming Model"** submitted by **Madhumita Kundu** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by her under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Sourav Chakraborty
Associate Professor
Advanced Computing
and Microelectronics Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA.

Saket Saurabh
Adjunct Professor
Advanced Computing
and Microelectronics Unit
Indian Statistical Institute
Kolkata-700108, INDIA.

Abstract

Graph is a popular model to represent highly structured data which involves entities who have pairwise relations between them. In many applications, computing graph theoretic properties after modelling the entire dataset as graph, provides us interesting informations which gives us insights about the whole dataset. However, in case of application, the datasets in question can be so large that it's difficult to store in the main memory and the dataset can even be dynamic(can change with time). These days in so many applications, the algorithm that requires to solve the problem which takes massive dataset as input, has limitations on time as well as space taken to store the information. These constraints leads us for the development of new techniques. Streaming model of computation takes all these challenges into account and provides us solutions with limited resources in cost of accuracy. Graph stream is a sequence of imcoming edges and we are only allowed to insert(insertion only model) or both insert and delete(dynamic model) into an initially empty graph. Finally our objective is to find out certain properties of the graph at the end of the stream which minimizes the amount of space the algorithm uses. Sometimes this algorithm needs to provide the trade of between the space usage and the time taken.

There is a large volume work on undirected graphs in streaming model but the area of directed graph stream is a pretty unexplored. In this project, we study the problem of testing acyclicity in *dense digraphs* in semi-streaming model. Here the graph on n vertices is presented as a stream of edges and using $\mathcal{O}(n \operatorname{polylog}(n))$-space, we must determine if it is acyclic or not.

Keywords: *semi-streaming algorithm, digraphs, acyclicity.*

Contents

Chapter 1

Introduction

1.1 Introdution to Small Space Algorithm and it's Desire

Small space algorithms are algorithms that takes limited amount of space to calculate some function of a massively long input stream σ. Formally, a sequence $\sigma = \langle x_1, x_2, \cdots, x_m \rangle$ comes as a "stream"(which is the input) in the given order where the elements of the sequence are picked from the universe $[n] := \{1, 2, 3, \cdots, n\}$. Here, the stream length, m, and the universe size, n are two important parameters.

The main ***goal*** is to process the input stream to calculate the desired function using a small amount of space s , i.e., to use s bits of memory. Since it's believed that m and n are "huge", we want to make s to be as small as possible. Specifically, we want s to be sublinear in both m and n. In symbols, we want $s = o(\min\{m, n\})$. To be precise, we want s to be $\mathcal{O}(\log m + \log n)$.

This amount of usage of space is possible only when we store a constant number of elements from the stream and a constant number of counters that can count up to the length of the stream. Sometimes we can only come close and achieve a space bound of the form $s = \text{polylog}(min\{m, n\})$, where $f(n) = \text{polylog}(g(n))$ means that there exists a constant $c > 0$ such that $f(n) = \mathcal{O}((\log g(n))^c)$.

The input is called a ***stream*** because we are allowed to access the input in "streaming fashion" i.e. we do not have random access to the elements and we can only scan the sequence in the given order. We are happy with the algorithms that make p passes over the stream, for some "small" integer p, with the target of achieving $p = 1$.

Sometimes we can only compute an estimated value or approximated value of the function $\phi(\sigma)$(the function that we wish to calculate) because for many functions, we can not compute the exact value using sublinear space in which case, we often allow randomized algorithms that err with some small, but controllable probability.

Chapter 2

Graph Streaming

2.1 Motivation

A large body of research has been developed to process and analyze massive dataset. We specifically concentrate on the dataset which can be modeled as graph and then we try to find algorithms to compute it's properties using limited amount of space.

Graph is a popular model to represent highly structured data which involves entities who have pairwise relations between them. In many applications, computing graph theoretic properties of the entire datasets provides us interesting informations which gives us insights about the whole dataset. However, in case of application, the datasets in question can be so large that it's difficult to store in the main memory and the datasets can even be dynamic(can change with time). These days in so many applications, the algorithm that requires to solve the problem which takes massive dataset as input, has limitations on time as well as space taken to store the information. These constraints leads us for the development of new techniques. Streaming model of computation takes all these challenges into account and provides us solutions with limited resources in cost of accuracy. Graph stream is a sequence of incoming edges and we are allowed to only insert(insertion only model) or both insert and delete(dynamic model) into an initially empty graph. Finally our objective is to find out certain properties of the graph at the end of the stream which minimizes the amount of space the algorithm uses. Sometimes this algorithm needs to provide the trade of between the space usage and the time taken.

There is a large volume work on undirected graphs in streaming model but the area of directed graph stream is a pretty unexplored. In this project, we study the problem of testing acyclicity in *dense digraphs* in semi-streaming model. Here the graph on n vertices is presented as a stream of edges and using $\mathcal{O}(n \operatorname{polylog}(n))$-space we must determine if it is acyclic or not.

2.2 Difficulties with Classical Model

Classical graph algorithms use random access to the entire input graph which is stored in the memory. However, in practice the datasets in question can be so large that it is difficult to store in the main memory. These days in so many applications, the algorithm that requires to solve the problem and which takes massive dataset as input, has limitations on time as well as space taken to store the information. These constraints leads us for the development of new techniques.

2.3 Graph Streaming

A graph stream is defined by a sequence of edge insertions (and sometimes insertions and deletions both) into an initially empty graph and after the stream finishes, we compute certain properties of the graph.

2.3.1 Objective

The objective is to compute a certain property of the graph at the end of the stream while minimizing the amount of space the algorithm uses. The space used by the streaming algorithm is called as streaming complexity of the algorithm.

2.3.2 Models of Computation

There several models of computation in streaming listed below.

1. *Edge Insertion Model*: The stream consists of edges of G and we are allowed to insert the edge or to let it pass in the stream.

2. *Dynamic Edge Model*: The stream consists of edges of G and we are allowed to insert the edge or to delete the edge(if it already exists in the graph) or let it pass in the stream.

2.4 Notations

Symbols	**Meaning**
$G(V, E)$	**An undirected graph** G
$V(G)$	**Set of vertices of** G**.**
$E(G)$	**Set of vertices of** G**.**
$D(V, A)$	**A directed graph** D

$V(D)$	**Set of vertices of D.**
$A(D)$	**Set of edges of D.**
$xy \in A(D)$	**edge xy from the edge set $A(D)$**
For $U \subseteq V(G), G[U]$	**A subgraph of G induced on U.**
$V(G) = A \uplus B$	**Graph G with bipartition on A & B**

Chapter 3

Previous Works and Overview of Results

There are lots of fundamental graph-theoretic problems that serve as important primitives in massive graph analysis. There are several problems such as finding large matching in a graph in *Dynamic Edge Model*, approximating the number of fixed length cycles in a graph, topological sorting of a directed acyclic graph (DAG), checking whether the input graph is indeed a DAG etc which have been already solved. Most streaming algorithms are based on the surprising efficacy of using random linear projections, aka *linear sketching*, for solving combinatorial problems. Problems admitting the use of this technique include testing edge connectivity[1] and vertex connectivity[12], constructing sparsifiers[2, 3, 16], approximating the densest subgraph[6, 10, 19], correlation clustering[4], and estimating the number of triangles[19].

3.1 Dynamic Graph Stream

1. **Dynamic Graph Stream[2]:**. A stream $S = a_1, \cdots, a_t$ where $a_k \in [n] \times [n] \times \{-1, 1\}$ defines a multi-graph graph $G = (V, E)$ where $V = [n]$ and the multiplicity of an edge (i, j) equals

$$A(i, j) = |k : a_k = (i, j, +)| - |k : a_k = (i, j, -)|$$

   We assume that the edge multiplicity is non-negative and that the graph has no self-loops.

2. **Linear Measurements and Sketches[2]:** A linear measurement of a graph is defined by a set of coefficients $c(i, j)$ for $1 \leq i < j \leq n$. Given a multi-graph $G = (V, E)$ where edge (i, j) has multiplicity $A(i, j)$, the evaluation of this measurement is P, $\sum_{1 \leq i < j \leq n} c(i, j)A(i, j)$. A *sketch* is a collection of linear measurements.

3. Consider a turnstile stream $S = a_1, \cdots, a_t$ where each $s_i \in (u_i, \Delta_i) \in [n] \times R$ and the aggregate vector $x \in \mathcal{R}^n$ defined by this stream, i.e., $x_i = \sum_{j:u_j=i} \Delta_i$. A δ-error ℓ_0-sampler for $x \neq 0$ returns FAIL with probability at most δ and otherwise returns (i, x_i) where i is drawn uniformly at random from

$$support(x) = i : x_i \neq 0.$$

ℓ_0 **Sampling[2]:** There exists a sketch-based algorithm that performs ℓ_0 Sampling using $\mathcal{O}(\log^2 n \log \delta^{-1})$ space assuming access to a fully independent random hash function.

4. **Sparse Recovery[2]:** There exists a sketchbased algorithm, k-RECOVERY, that recovers x exactly with high probability if x has at most k non-zero entries and outputs FAIL otherwise. The algorithm uses $\mathcal{O}(k \log n)$ space assuming access to a fully independent random hash function.

3.2 Lower Bound Methods

If we want to compare amongst algorithms and try to find out an algorithm which performs better than whatever we have in our hand, then lower bound comes in picture. Lower bound of a problem says that the problem is hard instead of saying that we can not come up with a better algorithm.

3.2.1 Difficulty with lower bound

The difficulty with lower bound of a problem is that it is very hard to write a lower bound proof which cover all the algorithms including the ones no one has thought of yet.

Communication Complexity is a very useful area which helps to prove space lower bound for data streaming algorithms which are most often proved via reduction from standard problem of communication complexity.

3.2.2 Communication Complexity

Let $f : \{0,1\}^a \times \{0,1\}^b \to \{0,1\}$ be a Boolean function.In communication complexity, there are two parties, Alice and Bob. In the one-way communication model Alice receives an input $x \in \{0,1\}^a$ and Bob receives an input $y \in \{0,1\}^b$. Neither one has any idea about other's input. Alice is only allowed to send one message to Bob and no message is allowed to be sent from Bob to Alice. The goal is for Bob to compute $f(x, y)$. The communication cost is measured by the number of bits Alice sends in the

worst case. The communication protocol can be both deterministic and randomized with error probability say almost δ.

- One-way deterministic communication complexity of a boolean function f,denoted by $D^{\rightarrow}(f)$ is the minimum worst-case number of bits used by any one-way protocol that correctly calculates f.
- The one-way (bounded-error) randomized communication complexity of a boolean function f,denoted by $R^{\rightarrow}(f)$ is the minimum worst-case number of bits used by any one-way protocol that correctly decides the function with probability at least ϵ. For a randomized protocol R, R has error probability at most ϵ if $\Pr(R(x,y) = f(x,y)) \geq 1 - \epsilon$.

3.2.3 Connection to Streaming Algorithm

Communication lower bounds on $D^{\rightarrow}(f)$ and $R^{\rightarrow}(f)$ provide lower bounds on the memory required for deterministic and randomized data streaming algorithms, respectively, via reduction from the standard problems of Communication Complexity. Let's consider a problem that can be solved using a streaming algorithm S that uses space s. The idea for Alice and Bob is to treat their input as a stream (x, y) with all of $x \in \{0,1\}^a$ arriving before all of $y \in \{0,1\}^b$ i.e. Alice creates a stream $\sigma(x)$ from her input $x \in \{0,1\}^a$, and runs the streaming algorithm S on $\sigma(x)$. Then Alice passes the state of the algorithm to Bob. Bob creates a stream $\sigma(y)$ from his input $y \in \{0,1\}^b$, and continues the execution of the streaming algorithm S on $\sigma(y)$. Now If the output of the streaming algorithm on the concatenated stream $\sigma(x)\ o\ \sigma(y)$ can be used to solve the problem f, either deterministically or with constant error probability, then the space complexity of the streaming algorithm must be at least $D^{\rightarrow}(f)$ or $R^{\rightarrow}(f)$, respectively.

There are several communication complexity problems which have been studied before and their lower bounds and further we will look at some existing lower bounds for some undirected graph problems.

3.2.4 Some problems in Communication Complexity and their results:

INDEX$_n$ Problem

In INDEX$_n$ problem, Alice holds a n-bit string $x \in \{0,1\}^n$ and Bob holds an integer $i \in \{0, 1, 2 \cdots, n\}$. The goal for Bob is to calculate the i^{th} bit of Alice's input i.e. x_i.

In case of deterministic communication complexity protocol, the intuition is as Alice is unaware of Bob's input which makes her to send entire input of it's own for Bob to answer the question.

The one-way randomized communication complexity of Index problem is $\Omega(n)$.

DISJOINTNESS$_n$ Problem

In DISJOINTNESS$_n$ Problem, Alice holds a n-bit string $x \in \{0,1\}^n$ and Bob holds a n-bit string $x \in \{0,1\}^n$. The goal for Bob is to output 0 if there is an index i for which $x_i = y_i = 1$, otherwise output 1.

In case of deterministic communication complexity protocol, $D^{\rightarrow}DISJOINTNESS_n \geq n$.

EQUIVALENCE$_n$ Problem

In EQUIVALENCE$_n$ Problem, Alice holds a n-bit string $x \in \{0,1\}^n$ and Bob holds a n-bit string $x \in \{0,1\}^n$. The goal for Bob is to output 1 if there is an index i for which $x_i \neq y_i$, otherwise output 0.

In case of deterministic communication complexity protocol, $D^{\rightarrow}EQUIVALENCE_n \geq n$.

PERM$_n$ Problem

In PERM$_n$ problem, Alice holds a permutation σ of $\{1, 2, 3, \cdots, n\}$, represented as an ordered list $\sigma(1), \sigma(2), \cdots, \sigma(n)$ which has $n \log n$ bits. Bob holds an index $i \in [n \log n]$ and the goal for Bob is to determine the i^{th} bit of σ.

In case of randomized communication complexity protocol, $R^{\rightarrow}PERM_n \geq n \log n$.

3.3 Lower Bounds for undirected Graph Problems

There are several undirected graph problems whose lower bounds has been proved via reduction from previous Communication Complexity problems published by Sun et.al [20]

3.3.1 Connectivity

In Connectivity problem, Alice holds a subset E_A of edges of an undirected graph G with the set V of n vertices, while Bob has a disjoint subset E_B of the edges of G. Alice sends a randomized message $M(E_A)$ to Bob. The goal for Bob is to decide if the graph $(V, E_A \cup E_B)$ is connected. Bob should succeed with probability at least 9/10.

Theorem 1. $R^{\rightarrow}$*(Connectivity)* $= \Omega(n \log n)$

3.3.2 Diameter

In diameter problem, for a given $k \in [n-1]$ and a given sparse graph with $\mathcal{O}(n)$ edges, the goal for Bob is to decide if the diameter d of $(V, E_A \cup E_B)$ is at most k.

Theorem 2. *For any* $k \geq 4$, $R^{\rightarrow}(diameter - k) = \Omega(n \log n)$

3.3.3 Eulerian Testing

In Eulerian Testing problem, a graph $G(V, E)$ is given and the goal for Bob is to decide if $(V, E_A \cup E_B)$ is an Eulerian graph.

Theorem 3. $R^{\rightarrow}(Eulerian) = \Omega(n \log n)$

3.3.4 Bipartiteness

In Bipartiteness problem, Alice holds a subset E_A of edges of an undirected graph G with the set V of n vertices and Bob holds a disjoint subset E_B of the edges of G. Alice sends a randomized message $M(E_A)$ to Bob. The goal for Bob is to decide if the graph $(V, E_A \cup E_B)$ is bipartite.

Theorem 4. $R^{\rightarrow}(bipartite) = \Omega(n \log n)$

3.3.5 Cycle-free

In Cycle-free problem, Alice holds a subset E_A of edges of an undirected graph G with the set V of n vertices and Bob holds a disjoint subset E_B of the edges of G. Alice sends a randomized message $M(E_A)$ to Bob. The goal for Bob is to decide if the graph $(V, E_A \cup E_B)$ is cycle-free.

Theorem 5. $R^{\rightarrow}(Cyclefree) = \Omega(n \log n)$

Chapter 4

Cycle Detection Problem and Previous Work

4.1 Introduction to cycle detection problem

In this project we study the problem which is defined as: for a given dense digraph D on n vertices as a stream of edges, can we test if it is acyclic using $\mathcal{O}(n \operatorname{polylog}(n))$ space. *Semi-streaming* model of computation has been used here, where the stream of edges can be either *insertion-only*, or *dynamic* which means both insertion and deletion of edges are allowed. This model of computation is very well suited to study massive data-sets with an underlying graph structure.

4.2 Previous Work

Undirected graphs are quite well studied in this model [18], but problems on directed graphs are not yet well explored. Some results are known on space lower-bounds for finding sinks in digraphs [15] and reachability [11], ruling out constant-pass algorithms for directed reachability [13] and a few others. In a recent work, Chakrabarti et.al [7] considered several problems related to vertex ordering problems in (acyclic) digraphs, e.g. the space complexity of computing a topological ordering*. They obtain upper and lower-bounds on testing acyclicity and computing topological orders in tournaments and arbitrary digraphs. In particular, they show that $\Omega(n^2)$ space is required for testing acyclicity in arbitrary digraphs in constant-passes.

Ahn et.al. [1] initiated the study of *graph sketching*, by giving algorithms for several problems on undirected graphs using bounded-space, using a limited number of linear

*This problem was explicitly raised in an open problems session at the Shonan Workshop "Processing Big Data Streams" (June 5-8, 2017).

measurements. These include testing connectivity, bipartiteness, spanning trees etc in the semi-streaming model. While there are a number of results on undirected graphs in this model [18], not much is known on digraphs.

This work is motivated by the above results, in particular, whether we can do better than the $\Omega(n^2)$ lower-bound when the input digraph is more structured.

Other than the works mentioned earlier, some results are known about computing DFS-trees [17] and shortest-path trees [9] in $\mathcal{O}(n/\operatorname{polylog}(n))$ passes. These results remark upon the "implicit hardness" of problems in digraphs. Some super-linear(in n) lower-bounds are known for solving certain decision problems, such as perfect matching and short $s-t$ path [13].

Some graph problems and their lower bounds of Chakrabarti et.al. [7] are listed below.

- Solving topological sort in one pass requires $\Omega(n^2)$ space in streaming model.
- Solving strongly connected DAG in one pass requires $\Omega(n^2)$ space in streaming model.
- Solving acyclicity requires $\Omega(n^2)$ space in one pass and $n^{1+\Omega(1/p)}/p^{O(1)}$ space in p passes.

Chapter 5

Our Work

To recall the problem, for a given dense digraph D on n vertices as a stream of edges, whether we can test if it is acyclic using $\mathcal{O}(n \operatorname{polylog}(n))$ space. *Semi-streaming* model of computation has been used here, where the stream of edges can be either *insertion-only*, or *dynamic* which means both insertion and deletion of edges are allowed.

5.0.1 Model of Computation:

We design algorithms in the *graph semi-streaming model*, where an n-vertex digraph G is presented as a stream of edges. This stream could be *insertion-only* where edges are added one by one, or it could be *dynamic* where edges can be both added and deleted. In the case of dynamic stream we assume that the length of the stream is $\mathcal{O}(n^c)$ where c is a constant. Our algorithms are allowed to use $\mathcal{O}(n \operatorname{polylog}(n))$ space and output a solution at the end of the stream. When the digraph G has $\Omega(n^c)$ edges for any $c > 1$, we call it a *dense digraph*, and note that the space allowed in the semi-streaming model is sub-linear in the input-size for such graphs.

5.1 Notations and Definitions

We use the term "digraph" for a simple digraph without self-loops, labels, and parallel edges. That is, for a digraph D, there do not exist two vertices x and y such that both xy and yx are arcs in D. For a digraph D, $V(D)$ and $A(D)$ denote the set of vertices and arcs of D, respectively. For a digraph D and $v \in V(D)$, the *in-degree* and *out-degree* of v are $|\{uv \in A(D) : u \in V(D)\}|$ and $|\{vu \in A(D) : u \in V(D)\}|$, respectively. The *underlying (undirected) graph* of a digraph D is a undirected graph G on the vertex set $V(D)$ that contains an edge (u, v) whenever one of the arc (u, v) or (u, v) are present in D.

We study this question of testing acyclicity on several classes of *dense digraphs*. These

are defined in terms their underlying undirected graph, corresponding to some very well studied dense graph classes, namely tournaments and digraphs k-close to tournaments, split digraphs, co-bipartite digraphs and cluster digraphs. For each of these classes, we give algorithms that test the acyclicity of these class of digraphs. We use the term "graph" for a simple undirected graph without self-loops, labels, and parallel arcs. For a graph G, $V(G)$ and $E(G)$ are the set of vertices and edges of G, respectively. For a graph G and $U \subseteq V(G)$, $G[U]$ denote the subgraph of G induced on U. A graph G is a bipartite graph if there is a bipartition of the vertex set $V(G) = A \uplus B$ such that A and B are independent sets in G. The partition $A \uplus B$ is called a bipartition of G.

A cycle of length ℓ, where $\ell \geq 3$, in a directed graph D, is a sequence of vertices $v_1, \ldots, v_\ell, v_1$ such that $v_i \neq v_j$ for all $1 \leq i < j \leq \ell$, $v_i v_{i+1} \in A(D)$ for all $i \in [\ell - 1]$, and $v_\ell v_1 \in A(D)$. We use C_ℓ to denote a cycle of length ℓ and C_3 is also called a triangle. For a cycle C in a digraph, we use $V(C)$ to denote the set of vertices in the cycle. For a digraph D and $U \subseteq V(D)$, $D[U]$ denote the subgraph of D induced on U. For a digraph D, the underlying undirected graph of D, is the graph G with vertex set $V(G) = V(D)$ and the edge set $E(G) = \{\{x, y\} : xy \in A(D)\}$. For an acyclic digraph D, there is an ordering $<$ of the vertices $V(D)$, called the *topological order*, where for any arc $uv \in A(D)$, $u < v$. Moreover, if a digraph has a topological order, then it is acyclic. A tournament is a digraph D such that for any two distinct vertices $u, v \in V(D)$, either $uv \in A(D)$ or $vu \in A(D)$ and not both. Formally we prove the following.

Theorem 6. *Let D be a digraph on n vertices that is presented as an edge stream. Then in $\mathcal{O}(n \operatorname{polylog}(n))$ space and constant-passes we can recognize if D is an acyclic digraph that belongs to one of the following graph classes:*

- *tournaments or digraphs k-close to a tournament for a dynamic edge stream,*
- *split digraphs for a stream of edge insertions,*
- *co-bipartite digraphs for a stream of edge insertions,*
- *or cluster digraphs for a dynamic edge stream.*

We remark that, our algorithms not only need to test for acyclicity, but also determine if the input digraph actually lies in the digraph class. Indeed, while the idea behind our algorithms is to exploit the structure of the underlying undirected graph, as a first step we must obtain a "structured decomposition" of the graph. For example, for digraphs k-close to tournaments, we must first partition it into a tournament and a sub-graph on k vertices. In the next step, we test for acyclicity using this decomposition.

Definition 1 (k-sparse-subgraph)**.** *Let G be a graph and k be an integer. Then a subgraph H of G is called a* k-sparse-subgraph *if for every vertex $v \in V(G)$, we have $2k \geq d_H(v) \geq \min\{k, d_G(v)\}$.*

To compute k-sparse subgraph, we require the following result of Barkay et.al. [5].

Proposition 1. *Let U be a universe on n elements, and let k be an integer. There is a randomized data-structure that accepts insertion and deletion of elements from U such that, at any moment it can provide a (uniformly at random) sample of size $\min\{k, \ell\}$ from the currently stored elements with probability at least $1 - 1/n^{c_0}$. Here, c_0 is some constant and ℓ denotes the number of currently stored elements in the data structure. This data-structure requires $\mathcal{O}(k \operatorname{polylog}(n))$ space.*

Lemma 1. *Let G be a graph on n vertices that is given as a dynamic stream, and let k be an integer. Then, there is a randomized algorithm uses $\mathcal{O}(nk \operatorname{polylog}(n))$ space and in one-pass outputs a k-sparse subgraph of the graph G with probability at least $1 - 1/n^c$ for some constant c.*

Proof. Let H denote the k-sparse subgraph of G that we will compute. Initially, H has no edges. And For each vertex $v \in V(G)$, we have an instance of the data-structure given by Proposition 1 that stores the edges incident on v in the edge stream. Now we consider the stream of edges, and for each $v \in V(G)$ whenever an edge incident to v is inserted or deleted, we update the data-structure for v. At the end of the stream, for every $v \in V(G)$, we compute a $\min\{k, d_G(v)\}$ sample of edges incident to v, and add them to H. Note that this succeeds with probability at least $1 - 1/n^c$ over all n vertices, where $c = c_0 - 1$. Further, the $2k \geq d_H(v) \geq \min\{k, d_G(v)\}$ for every vertex v, i.e. H is a k-sparse subgraph of G. Finally obeserve that, we require $\mathcal{O}(nk \operatorname{polylog}(n))$ space in total. □

We require the following result of Ahn et.al. [1] on dynamic connectivity in the semi-streaming model.

Proposition 2. *Let G be a graph on n vertices that is presented as a dynamic edge stream. There is a single-pass algorithm for computing a spanning forest of G using $\mathcal{O}(n \operatorname{polylog}(n))$ space.*

5.2 Some Results

5.2.1 Tournaments and digraphs k-close to Tournaments

In this section we describe an algorithm for testing acyclicity in tournaments and in digraphs that are close to tournaments. Let us begin with the simple case of tournaments, and let us recall that in a tournament there is an arc between each pair of vertices. We also require the following well known properties of tournaments [8].

Proposition 3. *Let D be a tournament. Then D contains a cycle if and only if D contains a C_3.*

Proposition 4. *Let D be a tournament on n vertices. Then D is acyclic if and only if D has a unique topological order, and hence the in-degree of every vertex is distinct and belongs to $\{0, 1, 2, \cdots, n-1\}$.*

We now describe a one-pass streaming algorithm for testing the acylicity of a tournament in the semi-streaming model.

Theorem 7. *Let D be a digraph on n vertices that is given as an dynamic edge stream. Then in one-pass and using $\mathcal{O}(n \log n)$ space we can test if D is an acyclic tournament, and further output it's unique topological order.*

Proof. We maintain n counters $\{\mathsf{count}_{\mathsf{In\text{-}Deg}}(v)\}|_{v \in V(D)}$, each of $\mathcal{O}(\log n)$ bits, that maintain the in-degree of every vertex. When an arc $e = (u, v)$ is added, we increment the counter $\mathsf{count}_{\mathsf{In\text{-}Deg}}(v)$, and when it is deleted we decrement the counter by one. Observe that at the end of the stream, $\mathsf{count}_{\mathsf{In\text{-}Deg}}(v)$ is the in-degree of $v \in D$. When the stream ends, we check if the following two conditions are satisfied: (i) $\sum_{v \in V(G)} \mathsf{count}_{\mathsf{In\text{-}Deg}}(v) = \binom{n}{2}$, and (ii) for every vertex $v \in V(D)$ it's in-degree $\mathsf{count}_{\mathsf{In\text{-}Deg}}(v)$ is a distinct number in $\{0, 1, \cdots, n-1\}$. If so, we output that the graph is an acyclic tournament. Further, when D is acyclic, we sort the vertices by their in-degree and output it as the topological order of D. The correctness of this algorithm follows directly from Proposition 4. Further, it is clear that it requires $\mathcal{O}(n \log n)$. □

An n vertex digraph D is *k-close to a tournament* if there exists $X \subseteq V(D)$ such that $D - X$ is a tournament and $|X| \leq k$. Let us now consider the more challenging problem of testing acyclicity in digraphs that are k-close to a tournament. We require the following property of this class of digraphs.

Observation 1. *Let D be a digraph that is k-close to a tournament. Let G be the underlying graph of D. Then the complement graph $\overline{G}$ admits a vertex cover of size k.*

We also require the following subroutine in our algorithm.

Lemma 2. *Let G be an undirected graph that is presented as a dynamic stream. Suppose that G admits a vertex-cover of size k. Then in one-pass and $\mathcal{O}(nk \log n)$ space we can compute a 2-approximate vertex cover of G.*

Proof. Our algorithm computes a $(2k+1)$-sparse subgraph H of G, by applying Lemma 1 to the dynamic stream of G in a single-pass, and using $\mathcal{O}(nk \operatorname{polylog}(n)\}$ space. Note that for every vertex $v \in V(G)$, in the graph H, $4k + 2 \geq d_H(v) \geq$

$\min\{2k+1, d_G(v)\}$. Observe that, as H is a subgraph of G, H also admits a vertex cover of size k.

In the reverse direction, we claim that if X is a vertex cover of H of size $2k$, then it is also a vertex cover of G. Suppose not, and consider an edge $(u,v) \in E(G)$ such that $u, v \notin X$. This means $(u,v) \notin E(H)$, and since $d_H(v) \geq \min\{2k+1, d_G(v)\}$, this means $d_H(u),\ d_H(v) \geq 2k+1$. Now observe that as $d_H(v) \geq 2k+1$, it must be in X, otherwise X has to contain all its neighbors (i.e at least $2k+1$ vertices). But this is a contradiction. Hence X is a vertex cover of G.

To compute a vertex cover X of H of size $2k$, we simply compute a maximal matching M of H. Since H admits a vertex cover of size H, M contains at most k-edges. Hence $X = V(M)$ is a vertex cover of G of size at most $2k$. □

Theorem 8. *Let D be a digraph on n vertices that is given as an dynamic edge stream. Then in two-passes and using $\mathcal{O}(nk \log n)$ space we can test if D is an acyclic digraph that is k-close to a tournament.*

Proof. Let G denote the underlying undirected graph of D. By Observation 1, the complement graph $\overline{G}$ has a vertex cover of size k. In the first pass of the graph stream, we compute a 2-approximate vertex cover X of $\overline{G}$, by applying Lemma 2 as follows. Initially, insert every edge to $\overline{G}$ which corresponds to the state when G is empty. Then whenever an edge (u,v) is added to G, we delete it from $\overline{G}$ and vice-versa. At the end of the stream we obtain the vertex cover X of $\overline{G}$ and by Lemma 2 this requires $\mathcal{O}(nk \log n)$ space. For the next stage of the algorithm, we consider the partition of $V(D)$ into X and $Y = V(D) \setminus X$. Observe that $D[Y]$ is a tournament and $|X| \leq 2k$.

In the second pass of the graph stream, we collect the following data. We store every arc that is incident on a vertex in X. Note that, as $|X| \leq 2k$, this requires $\mathcal{O}(nk \log n)$ space. Further, we also apply Theorem 7 to the tournament $D[Y]$ to test it's acyclicity. At the end of the stream, we either obtain that $D[Y]$ contains a cycle, or it is an acyclic tournament with the unique topological order $<^Y_{top}$.

Finally it remains to check if there are any cycles in D that contain a vertex from X. Towards this, we construct a digraph H from $D[X]$ by introducing some additional arcs to it. Note that H may contain anti-parallel arcs (e.g. $(u,v),(v,u)$) and self-loops (v,v). For each ordered pair of vertices $(u,v) \in X \times X$, we test if there is a path in D starting from u and ending at v with all internal vertices in Y. Observe that such a path exists whenever u has an out neighbor $p \in Y$ and v has an in-neighbor q such that $p <^Y_{top} q$. If such a path exists, we add an arc (u,v) to H, if it is not already present. Further, it is easy to see that every cycle in D maps to a cycle in H, and any cycle in H maps to a closed walk in D. Therefore, H is acyclic if and only if D is acyclic. This fact can be easily checked, and we output an answer accordingly. It is also clear that the algorithm requires $\mathcal{O}(nk \log n)$ space. □

5.2.2 Split digraph

In this section we give a streaming algorithm for testing the acyclicity of *directed split graphs*. Recall that an undirected graph G is called a *split graph* if $V(G)$ can be partitioned into $K \uplus I$ such that K is a clique and I is an independent set in G. This partition is called a *split partition* of the graph G.

Definition 2. *A digraph D is called a directed split graph if the underlying undirected graph of D is a split graph. In other words, if D is a directed split graph, then $V(D)$ can be partitioned into $K \uplus I$, called the split partition, such that $D[K]$ is a tournament and I is an independent set in D.*

We establish some basic properties of cycles in directed split graphs.

Lemma 3. *Let D be a directed split graph, with a split-partition $K \uplus I$, where the tournament $D[K]$ is acyclic. Let $<_{top}$ denote the unique topological order of $D[K]$. Then D contains a C_3 if and only if there is a vertex $v \in V(I)$ such that it has an in-neighbor $u \in K$ and an out-neighbor $w \in K$ such that $w <_{top} u$.*

Proof. If such a vertex v exists, then clearly D contains a C_3. In the reverse direction, let u, v, w, u be a C_3 in D . As I is an independent set and $D[K]$ is an acyclic tournament, the cycle u, v, w, u has exactly one vertex, say v, from I and exactly two vertices, say u, w, from K. Notice that u is an in-neighbor and w is an out-neighbor of v. Moreover, as $wu \in A(D)$ and $w, u \in K$, we have that $w <_{top} u$. This completes the proof of the lemma. □

Lemma 4. *A directed split graph contains a cycle if and only if it contains a C_3.*

Proof. Let D be a directed split graph that contains a cycle. Let $K \uplus I$ be the split partition of D. If the tournament $D[K]$ contains a cycle, then by Proposition 3 it contains a C_3, as required. Otherwise, consider the shortest cycle C in D, and consider a vertex $v \in V(C) \cap I$. Let u and w be the in-neighbor and the out-neighbor, respectively, of v in C. Observe that $u, w \in K$ and hence there is an arc between them in D. If we have the arc $wu \in A(D)$, then u, v, w, u is a C_3 in D. Otherwise, we have an arc uw and $D[V(C) \setminus v]$ contains cycle C' shorter than C obtained by replacing the sub-path u, v, w in C with u, w. This is a contradiction to the choice of C. The reverse direction of the proof is trivial. □

We also require a well-known characterization of split graphs via their degree-sequences.

Proposition 5 ([14])**.** *Let G be an undirected graph on n vertices and let the degree sequence of G be $d_1 \geq d_2 \geq ... \geq d_n$. Let m be the largest index i such that $d_i \geq i-1$. Then G is a split graph if and only if the following holds.*

$$\sum_{i=1}^{m} d_i = m(m-1) + \sum_{i=m+1}^{n} d_i$$

Further, there is a split partition of G where the first $n - m$ vertices of the degree-sequence form an independent set and the remaining m vertices form a clique.

Theorem 9. *Let D be a digraph on n vertices that is given as a stream of edge insertions. Then in $\mathcal{O}(n \log n)$ space and 3-passes we can determine if D is a acyclic split digraph.*

Proof. Our algorithm is implemented in 3-passes. We will describe each pass along with the data-structures required for it. Let G denote the underlying undirected graph of D. In the first pass, we maintain n degree-counters $\{\mathsf{count}_{\mathsf{Deg}}(v)\}|_{v \in V(G)}$ for the vertices in G. At the end of the first pass, we check if G is a split graph by applying Proposition 5, and if so, we also obtain a split partition $K \uplus I$ of G (and D). In the second pass, we apply Theorem 7 to the subgraph $D[K]$ to test if it is acyclic and if so, also obtain a topological order $<_{top}$ of $D[K]$. If $D[K]$ is not acyclic, then we declare that D is not acyclic.

Finally, in the third pass, we test if there is a cycle in D. Here, we assume that $D[K]$ is an acyclic tournament. By Lemma 3, it is enough to test for a C_3 in D. As $D[K]$ is an acyclic tournament, by Lemma 4, it is enough to test if there is a vertex $v \in I$ such that it has an in-neighbor $u \in K$ and an out-neighbor $w \in K$ such that $w <_{top} u$. Towards this, for each vertex $v \in I$, we maintain two vertices $u_v, w_v \in K \cap N(v)$ that denote the *largest in-neighbor* and the *smallest out-neighbor* of v as per the topological order $<_{top}$ of $D[K]$. Whenever we see an arc that is incident on v, we update u_v and w_v accordingly. At the end of the stream, we check if there exists a vertex $v \in I$ such that $w_v <_{top} u_v$. If so, we output that D contains a cycle, otherwise we output that D is acyclic. To conclude the proof, observe that we only require $\mathcal{O}(n \log n)$ space in total during the algorithm. □

5.2.3 Complement of Bipartite Graphs

In this section we describe a streaming algorithm to test acyclicity of digraphs whose underlying undirected graph is the complement of a bipartite graph. We require the following result of Ahn et.al. [1].

Proposition 6. *There exists a single-pass dynamic streaming algorithm for testing whether a graph G is bipartite using $\mathcal{O}(n \operatorname{polylog}(n))$ space. Furthermore, the algorithm outputs a bipartition of G if G is bipartite.*

We require the following properties for our algorithm.

Lemma 5. *Let D be a digraph whose underlying undirected graph is the complement of a bipartite graph. Then, D contains a cycle if and only if D contains a C_3 or a C_4 with $|V(C_4) \cap A| = 2$ and $|V(C_4) \cap B| = 2$.*

Proof. Let G be the underlying undirected graph of D. By assumption we know that G is a complement of a bipartite graph. That is, $\overline{G}$ is a bipartite graph. Let $A \uplus B$ be a bipartition of $V(\overline{G})$. Observe that $D[A]$ and $D[B]$ are both tournaments. If $D[A]$ or $D[B]$ contains a cycle, then by Proposition 3, there is a C_3. Otherwise both $D[A]$ and $D[B]$ are acyclic, and let $<^A_{top}$ and $<^B_{top}$ denote the unique topological order of $D[A]$ and $D[B]$, respectively. Now consider, a shortest cycle C in D, and note that it must intersect both A and B. If the length of C is at most 3, then we are done. Otherwise, we claim that C intersects A in at most two vertices and B in at most two vertices. This will imply that C is a cycle of length 4 and $|V(C) \cap A| = |V(C) \cap B| = 2$. Next we prove that $|V(C) \cap A| \leq 2$. The proof for the statement $|V(C) \cap B| \leq 2$ is symmetric.

Now we prove that $|V(C) \cap A| \leq 2$. Let $v_1, v_2, \ldots, v_\ell, v_1$ be the cycle C, where $\ell \geq 4$. Suppose $|V(C) \cap A| > 2$. Then there exist $1 \leq i < j < k \leq \ell$ such that $v_i, v_j, v_k \in V(C)$. Suppose $j \neq i+1$. That is $j > i+1$. As $v_i, v_j \in A$, and $D[A]$ is a tournament, either there is an arc v_iv_j or there is an arc v_jv_i in D. If v_iv_j is an arc, then $v_1, \ldots v_i, v_j, v_{j+1}, \ldots, v_k, \ldots, v_\ell, v_1$ is a cycle of length less than ℓ– a contradiction to the assumption that C is a shortest cycle. If v_jv_i is an arc in D, then $v_i, v_{i+1} \ldots, v_j, v_i$ is a cycle of length less than ℓ in D which is a contradiction to the assumption that C is a shortest cycle in D. Thus, it should be the case that $j = i+1$, By using similar arguments as above, one can prove that $k = j+1 = i+2$. That is, $v_i, v_{i+1}, v_{i+2} \in A$. Since v_i and v_{i+2} are two distinct vertices in the tournament $D[A]$, either $v_iv_{i+2} \in A(D)$ or $v_{i+2}v_i \in A(D)$. If $v_{i+2}v_i \in A(D)$, then $v_i, v_{i+1}, v_{i+2}, v_i$ is a shorter cycle that C which is a contradiction. If $v_iv_{i+2} \in A(D)$, then $v_1, \ldots, v_i, v_{i+2}, \ldots, v_\ell, v_1$ is a cycle of length less than ℓ which is a contradiction. Thus we have proved that $|V(C) \cap A| \leq 2$ and this completes the proof of the lemma.

□

Lemma 6. *Let D be a digraph whose underlying undirected graph G is the complement of a bipartite graph. Let $A \uplus B$ be a bipartition of the bipartite graph $\overline{G}$. Furthermore, let $D[A]$ and $D[B]$ be acyclic tournaments, and let $<^A_{top}$ and $<^B_{top}$ be their topological orders, respectively. Then D contains a cycle if and only if at least one of the following holds.*

(i) There exist vertices $v \in A$, $u, w \in B$ such that $uv \in A(D)$, $vw \in A(D)$, and $w <^B_{top} u$.

(ii) There exist vertices $v \in B$, $u, w \in A$ such that $uv \in A(D)$, $vw \in A(D)$, and $w <^A_{top} u$.

(iii) There exist vertices $u, v \in A$ and $x, y \in B$ such that $yu \in A(D)$, $vx \in A(D)$, $u <^A_{top} v$, and $x <^B_{top} y$,

Proof. It is easy to verify that if at least one of the condition $(i) - (iii)$ holds, then there is a cycle in D. Now we prove the forward direction. By Lemma 5, if D contains a cycle, then it contains a C_3 or a C_4. Suppose there is a triangle u, v, w, u in D. Suppose $|\{u, v, w\} \cap A| = 1$ and $|\{u, v, w\} \cap B| = 2$. Without loss of generality $v \in A$ and $u, w \in B$. Then, $uv, vw \in A(D)$ and $w <^B_{top} u$ because $wu \in A(D)$. This implies that condition (i) of the lemma holds. Now, if $|\{u, v, w\} \cap A| = 2$ and $|\{u, v, w\} \cap B| = 1$, by using arguments similar to above, one can show that condition (ii) of the lemma holds.

Now suppose that there is a no C_3 in D and there is a cycle C of length 4 in D such that $|V(C) \cap A| = |V(C) \cap B| = 2$. Let u, v, x, y, u be the cycle C. We claim that two consecutive vertices in C are in A and two consecutive vertices in C are in B. Suppose not. Then, without loss of generality let $u, x \in A$ and $v, y \in B$. As u and x are two distinct vertices in the tournament $D[A]$, either $ux \in A(D)$ of $xu \in A(D)$. If $ux \in A(D)$, then u, x, y, u is a C_3 in D which is a contradiction to the assumption that there is no C_3 in D. If $xu \in A(D)$, then x, u, v, x is a C_3 in D which is a contradiction to the assumption that there is no C_3 in D. Thus, we have proved that two consecutive vertices in C are in A and two consecutive vertices in C are in B. Without loss of generality $u, v \in A$ and $x, y \in B$. As $uv \in A(D)$ and $u, v \in A$, we have that $u <^A_{top} v$. Similarly, as $xy \in A(D)$ and $x, y \in A$, we have that $x <^B_{top} y$. Also note that $vx, yu \in A(D)$. This implies that condition (iii) of the lemma holds.

□

Theorem 10. *Let D be a digraph on n vertices such that the underlying undirected graph G of D is the complement of a bipartite graph. Then, there is 3-pass insertion only streaming algorithm using $\mathcal{O}(n \operatorname{polylog} n)$ space to test whether D is acyclic or not.*

Proof. First we check whether $\overline{G}$ is bipartite and if yes, obtain a bipartition $A \uplus B$ of $\overline{G}$ using one pass.

Towards this, we apply Proposition 6 in the following manner. Initially, we insert all $\binom{n}{2}$ pairs of $V(\overline{G})$ as edges, then when the edge stream of G comes, we delete those edges. At the end the graph will $\overline{G}$, and by Proposition 6, the output is Yes if $\overline{G}$ is bipartite. Moreover, if $\overline{G}$ is bipartite, the algorithm will output a bipartition $A \uplus B$ of $\overline{G}$. The space used for this computation is $\mathcal{O}(n \operatorname{polylog}(n))$.

In the second pass, we test if the tournaments $D[A]$ and $D[B]$ are acyclic, and if so we also obtain their respective topological orders $<^A_{top}$ and $<^B_{top}$. Towards this we apply Theorem 7 simultaneously for $D[A]$ and $D[B]$, which requires $\mathcal{O}(n \log n)$ space. If $D[A]$ and $D[B]$ are both acyclic tournaments, then we proceed to the next step. Otherwise we declare that D is not acyclic.

Finally, in the third pass, we test if there is a cycle in D by using Lemma 6.

(a) Observe that to test whether there exist vertices $v \in A$, $u, w \in B$ such that

$uv \in A(D)$, $vw \in A(D)$, and $w <^B_{top} u$ (i.e., condition (i) of Lemma 6), it is enough to store the following information for each vertex $z \in A$: the *largest* vertex $l_z \in B$ and the *smallest* vertex $s_z \in B$ in the order $<^B_{top}$ such that $l_z z, z s_z \in A(D)$. At the end if there is a vertex $z \in A$ such that $s_z <^B_{top} l_z$ for any $z \in A$, then condition (i) of the Lemma 6.

(b) Similarly, to test whether condition (ii) of Lemma 6 holds, it is enough to store the following information for each vertex $z \in B$: the *largest* vertex $l_z \in A$ and the *smallest* vertex $s_z \in A$ in the order $<^A_{top}$ such that $l_z z, z s_z \in A(D)$. And at the end if there is a vertex $zinB$ such that $s_z <^A_{top} l_z$, then condition (ii) of the Lemma 6.

(c) To test whether condition (iii) of Lemma 6 holds, it is enough to store the following information for each vertex $z \in B$: the *largest* vertex $l_z \in A$ and the *smallest* vertex $s_z \in A$ in the order $<^A_{top}$ such that $l_z z, z s_z \in A(D)$. And at the end if there exist two vertices x and y such that $x <^B_{top} y$ and $s_y <^A_{top} l_x$, then condition (iii) of the Lemma 6 holds.

Now, in the third stream, for each vertex z we store two values l_z and s_z. Initially, for each vertex z, we set l_z and s_z to be NULL. Then, when an arc wx comes where $w \in A$ and $x \in B$, if $x <^B_{top} s_w$, then we set $s_w := x$. Also if $l_x <^A_{top} w$, then we set $l_x := w$. Similarly, when an arc xw comes, where $x \in B$ and $w \in A$, if $w <^B_{top} s_x$, then we set $s_x := w$. Also if $l_w <^A_{top} x$, then we set $l_w := x$. When an arc e in $D[A]$ or $D[B]$ comes, we ignore it. At the end of the arc stream we test the cases (a), (b) and (c) and if at least one of them holds, then there is a cycle in D, by Lemma 6. Otherwise D is acyclic.

Notice that in the third stream, for each vertex z, we keep two information l_z and s_z. This implies that the space required in the third stream is $\mathcal{O}(n \log n)$.

This completes the proof of the theorem. □

5.2.4 Cluster digraphs

A *cluster digraph* is a digraph D that is a disjoint union of tournaments. Observe that the underlying graph of D is a cluster graph, i.e. a disjoint union of complete graphs.

In this section we present a streaming algorithm for testing acyclicity of cluster digraphs in the dynamic streaming model.

Theorem 11. *Let D be a digraph on n vertices that is presented as a dynamic edge stream. Then in $\mathcal{O}(n \operatorname{polylog}(n))$ space and one-pass we can determine if G is an acyclic cluster digraph and also outputs a partition of $V(G)$ such that each part induces a tournament.*

Proof. Let G denotes the underlying undirected graph of D. We require the following data-structures for our algorithm. We maintain a dynamic connectivity sketch of G, using Proposition 2. Recall that this requires $\mathcal{O}(n \operatorname{polylog}(n))$ space for a single-pass. We also store a collection of counters storing $\mathsf{count}_{\mathsf{In\text{-}Deg}}(u)$ for each vertex $u \in V(D)$. When an arc uv is added, we increment $\mathsf{count}_{\mathsf{In\text{-}Deg}}(u)$ by one and conversely when an arc uv is deleted, we decrement it. Observe that at the end of the stream, $\mathsf{count}_{\mathsf{In\text{-}Deg}}(u)$ is the in-degree of the vertex $u \in V(D)$.

To determine if D is a cluster digraph, we obtain a spanning forest F of G using Proposition 2. Let T be a tree in F, let $n_T = |V(T)|$ denotes the number of vertices in this tree. By Proposition 4, to test if $D[V(T)]$ is an acyclic tournament, it is enough to check if for each $v \in V(T)$, $\mathsf{count}_{\mathsf{In\text{-}Deg}}(v)$ is a distinct integer in $\{0, 1, \cdots, n_T - 1\}$. We perform this test for every tree in the forest F. If they are all acyclic tournaments, then we output that D is an acyclic cluster digraph.

Notice that the dynamic connectivity sketch uses $\mathcal{O}(n \operatorname{polylog}(n))$ space (see Proposition 4). Moreover, for each vertex u, we use a counter $\mathsf{count}_{\mathsf{In\text{-}Deg}}(u)$. Thus, the total space used by our algorithm is $\mathcal{O}(n \operatorname{polylog}(n))$. □

Chapter 6

Conclusion and future work

6.1 Conclusion

In this project, we obtained several algorithms for the problem of *Acyclicity Tests in Classes of Dense Digraphs in Streaming Model* for some special class of dense digraphs whose underlying undirected graph belongs to the class of graphs such as complement of bipartite graphs, split graphs,cluster graphs, k vertices away from tournament. In a recent work, Chakrabarti et.al. [7] considered several problems related to vertex ordering problems in (acyclic) digraphs, the space complexity of computing a topological ordering. They obtain upper and lower-bounds on testing acyclicity and computing topological orders in tournaments and arbitrary digraphs. In particular, they show that $\Omega(n^2)$ space is required for testing acyclicity in arbitrary digraphs in constant-passes. Solving this problem for tournament is very easy. However, the problem becomes maximally hard without the promise of a tournament. So we picked special classes of dense digraphs and addressed the problem.

6.2 Future Work

We are concentrating on few more dense digraphs and graphs with k vertices away from several classes of dense digraph to get some good upper bound for the same problem. We are also trying to prove lower bounds for the same problem of the some classes of dense digraphs for which we have come up with upper bounds.

Bibliography

[1] Ahn, K.J., Guha, S., McGregor, A.: Analyzing graph structure via linear measurements. In: Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms. pp. 459–467. SIAM (2012)

[2] Ahn, K.J., Guha, S., McGregor, A.: Graph sketches: sparsification, spanners, and subgraphs. In: Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems. pp. 5–14 (2012)

[3] Ahn, K.J., Guha, S., McGregor, A.: Spectral sparsification in dynamic graph streams. In: Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, pp. 1–10. Springer (2013)

[4] Ahn, K., Cormode, G., Guha, S., McGregor, A., Wirth, A.: Correlation clustering in data streams. In: International Conference on Machine Learning. pp. 2237–2246 (2015)

[5] Barkay, N., Porat, E., Shalem, B.: Efficient sampling of non-strict turnstile data streams. Theoretical Computer Science 590, 106–117 (2015)

[6] Bhattacharya, S., Henzinger, M., Nanongkai, D., Tsourakakis, C.: Space-and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In: Proceedings of the forty-seventh annual ACM symposium on Theory of computing. pp. 173–182 (2015)

[7] Chakrabarti, A., Ghosh, P., McGregor, A., Vorotnikova, S.: Vertex ordering problems in directed graph streams. In: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 1786–1802. SIAM (2020)

[8] Diestel, R.: Graduate texts in mathematics. Graph theory 173 (2000)

[9] Elkin, M.: Distributed exact shortest paths in sublinear time. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing. pp. 757–770 (2017)

[10] Esfandiari, H., Hajiaghayi, M., Woodruff, D.P.: Applications of uniform sampling: Densest subgraph and beyond. arXiv preprint arXiv:1506.04505 (2015)

[11] Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., Zhang, J.: On graph problems in a semi-streaming model. In: International Colloquium on Automata, Languages, and Programming. pp. 531–543. Springer (2004)

[12] Guha, S., McGregor, A., Tench, D.: Vertex and hyperedge connectivity in dynamic graph streams. In: Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems. pp. 241–247 (2015)

[13] Guruswami, V., Onak, K.: Superlinear lower bounds for multipass graph processing. Algorithmica 76(3), 654–683 (2016)

[14] Hammer, P.L., Simeone, B.: The splittance of a graph. Combinatorica 1(3), 275–284 (1981)

[15] Henzinger, M.R., Raghavan, P., Rajagopalan, S.: Computing on data streams. External memory algorithms 50, 107–118 (1998)

[16] Kapralov, M., Lee, Y.T., Musco, C., Musco, C.P., Sidford, A.: Single pass spectral sparsification in dynamic streams. SIAM Journal on Computing 46(1), 456–477 (2017)

[17] Khan, S., Mehta, S.K.: Depth first search in the semi-streaming model. arXiv preprint arXiv:1901.03689 (2019)

[18] McGregor, A.: Graph stream algorithms: a survey. SIGMOD Rec. 43(1), 9–20 (2014), `https://doi.org/10.1145/2627692.2627694`

[19] McGregor, A., Tench, D., Vorotnikova, S., Vu, H.T.: Densest subgraph in dynamic graph streams. In: International Symposium on Mathematical Foundations of Computer Science. pp. 472–482. Springer (2015)

[20] Sun, X., Woodruff, D.P.: Tight bounds for graph problems in insertion streams. In: Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2015)