UNIVERSITÉ PARIS-SUD

ÉCOLE DOCTORALE: INFORMATIQUE
Laboratoire de Recherche en Informatique (LRI)

***Discipline:** INFORMATIQUE*

THÈSE DE DOCTORAT SUR TRAVAUX

soutenue le 05/07/2013
par

Christian KONRAD

Computations on Massive Data Sets: Streaming Algorithms and Two-Party Communication

Directeur de thèse:	Frédéric Magniez	Directeur de recherche (CNRS, Université Paris Diderot)
Composition du jury		
Rapporteurs:	Christoph Dürr	Directeur de recherche (CNRS, Université P. et M. Curie)
	Andrew McGregor	Assistant Professor (University of Massachusetts, Amherst)
Examinateurs:	Magnús Halldórsson	Professor (Reykjavik University)
	Claire Mathieu	Directrice de recherche (CNRS, ENS Paris)
	Alain Denise	Professeur (Université Paris-Sud)

Abstract

The treatment of massive data sets is a major challenge in computer science nowadays. In this PhD thesis, we consider two computational models that address problems that arise when processing massive data sets.

The first model is the *Data Streaming Model*. When processing massive data sets, random access to the input data is very costly. Therefore, streaming algorithms only have restricted access to the input data: They sequentially scan the input data once or only a few times. In addition, streaming algorithms use a random access memory of sublinear size in the length of the input. Sequential input access and sublinear memory are drastic limitations when designing algorithms. The major goal of this PhD thesis is to explore the limitations and the strengths of the streaming model.

The second model is the *Communication Model*. When data is processed by multiple computational units at different locations, which are connected through a slow interconnection network such as the Internet, then the message exchange of the participating parties for synchronizing their calculations is often a bottleneck. The amount of communication should hence be as little as possible. A particular setting is the one-way two-party communication setting. Here, two parties collectively compute a function of the input data that is split among the two parties, and the whole message exchange reduces to a single message from one party to the other one. This model has strong connections to streaming algorithms: A one-pass streaming algorithm with space s serves as a one-way two-party communication protocol with message size s. Furthermore, a lower bound on the message size of a one-way two-party protocol is also a lower bound on the space requirements of a one-pass streaming algorithm. Studying streaming algorithms from the communication point of view therefore provides valuable insight.

In this work, we study the following four problems in the context of streaming algorithms and one-way two-party communication:

1. *Matchings in the Streaming Model.* We are given a data stream of edges of a graph $G = (V, E)$ with $n = |V|$, and the goal is to design a streaming algorithm that computes a matching using a random access memory of size $O(n \operatorname{polylog} n)$. The Greedy matching algorithm fits into this setting and computes a matching of size at least $1/2$ times the size of a maximum matching. A long standing open question is whether it is possible to compute a matching in one pass of size strictly larger than $1/2$ times the size of

a maximum matching if no assumption about the order of the input stream is made. We show that it is possible to obtain an approximation ratio strictly larger than $1/2$ in one pass if the input stream is in uniform random order. Furthermore, we show that with two passes an approximation ratio strictly larger than $1/2$ can be obtained if no assumption on the order of the input stream is made.

2. *Semi-Matchings in Streaming and in Two-Party Communication.* A semi-matching in a bipartite graph $G = (A, B, E)$ is a subset of edges that matches *all* A vertices exactly once to B vertices, not necessarily in an injective way. The goal is to minimize the maximal number of A vertices that are matched to the same B vertex. The problem is equivalent to scheduling a set of unit length jobs on identical machines with assignment conditions expressed through the edges of a bipartite graph. For this problem, we provide one of the very few streaming algorithms for matching problems that allows a trade off between space usage and approximation factor: for any $0 \leq \epsilon \leq 1$, with space $\tilde{O}(n^{1+\epsilon})$ our one-pass streaming algorithm computes an $O(n^{(1-\epsilon)/2})$-approximation. Furthermore, we provide upper and lower bounds on the two-party communication complexity of this problem, as well as new results on the structure of semi-matchings.
3. *Validity of XML Documents in the Streaming Model.* An XML document (eXtended Markup Language) is a sequence of opening and closing tags. A DTD (Document Type Definition) is a set of local validity constraints of an XML document. We study streaming algorithms for checking whether an XML document fulfills the validity constraints of a given DTD. Our main result is a $O(\log n)$-pass streaming algorithm with three auxiliary streams and $O(\log^2 n)$ space for this problem, where n is the length of the input XML document. Furthermore, for checking validity of XML documents that encode binary trees, we present a one-pass streaming algorithm with space $\tilde{O}(\sqrt{n})$, and a two-pass algorithm with space $O(\log^2 n)$ that makes one pass from left to right, and one pass from right to left.
4. *Budget-Error-Correcting under Earth-Mover-Distance.* We study the following one-way two-party communication problem. Alice and Bob have sets of n points on a d-dimensional grid $[\Delta]^d$ for an integer Δ. Alice sends a small sketch of her points to Bob and Bob adjusts his point set *towards* Alice's point set so that the Earth-Mover-Distance of Bob's points and Alice's points decreases. The Earth-Mover-Distance of two point sets of the same cardinality is the weight of a minimum weight perfect matching between the two point sets.

   For any $k > 0$, we show that there is a randomized protocol with communication cost $\tilde{O}(k \cdot d)$ such that Bob's adjustments lead to an $O(d)$-approximation compared to the k best possible adjustments that Bob could make. Furthermore, our upper bound is complemented by an almost matching lower bound.

Acknowledgements

First of all, I would like to thank my PhD advisor Frédéric Magniez. Frédéric made it possible for me to go down the path towards becoming a researcher. Amongst many other things, he aroused my interest in streaming algorithms, gave me the oppurtunity to travel, and acquainted me with many great researchers. Through his working attitude, I learned the importance of the mathematical precision that is necessary for a theoretical computer scientist. Thank you very much for the great support.

I would also like to explicitely thank Claire Mathieu for hosting me at Brown University. She found the time to work with me every day over the course of my one month visit and I am very grateful for the valuable experience that I obtained from this collaboration. Furthermore, I would like to thank Christoph Dürr for reviewing my thesis and for his continuous support in many respects throughout the last years. I would also like to thank Andrew McGregor for reviewing my thesis and for the continuous interest that he has taken in my research. I would like to thank the jury members Alain Denise, for the enjoyable collaboration in teaching the compiler class, and Magnús Halldórsson, for offering me the opportunity to collaborate in the future.

I was fortunate to discuss and collaborate with many great researchers over the last years. Many thanks go to Adi Rosén, Qin Zhang, Wei Yu, Ashwin Nayak, and Michel de Rougemont.

Thanks to my fellow PhD students (some of whom graduated years before me) Xavier, Marc, Jamie, Carola, André, Loïck, Mathieu, Antoine, Mathieu, Djamal, Nans, Virginie, Nathanaël, Laure and Charles, and all other members of the Algorithms and Complexity group at LIAFA.

Finally, I wish to thank my loved ones for the most precious support that one may receive.

Contents

List of Figures

List of Algorithms

LIST OF ALGORITHMS

Chapter 1

Introduction

In 2010, the amount of information created or replicated was more than one zettabyte (one billion terabytes) [GR11]. Nowadays, we are experiencing an immense data explosion. Experiments in particle physics at Cern generate up to one petabyte (one thousand terabytes) of data each second, and even though most of it is discarded immediately, up to 25 petabytes of data is stored every day[1]. The German Climate Computing Centre (Deutsches Klimarechenzentrum, DKRZ) has a database of 60 petabytes of climate data[2]. Google, Youtube, Facebook and many other web companies have databases of petabyte size in which heterogeneous data such as text, videos, music, pictures, and user statistics is stored[3]. The big data phenomenon is enabled, amongst other things, by inexpensive storage devices [GR11].

Extracting useful information from this data is a major challenge nowadays, and a task that makes high demands on modern computer science. The fact that data is no longer stored locally, but may be scattered all over the Internet or might even be discarded soon after its creation, causes many difficulties for algorithms that process it. In this PhD thesis we address two models of computation that tackle these difficulties. We consider the *data streaming model* and the *communication model* of computation. Data streaming is a way of coping with the *data access problem* for algorithms that process massive data sets. In the communication model, the amount of *communication* that is necessary when multiple parties process massive data in a distributed manner is studied. Data access and communication are fundamental problems in the area of massive data set algorithms and are bottlenecks in many applications. These problems are illustrated in Figure 1.1.

Algorithms that process massive data sets must address the problem of how to access the input data. Nowadays, standard computers are equipped with a *RAM (Random Access Memory)* of up to 16 gigabytes. Data that is stored in this memory can be addressed by algorithms in an arbitrary order with reasonable latencies. However, when processing data sets that exceed by far the size of a computer's RAM, random access to the input is an unrealistic assumption: Massive data sets are by far too large to fit into a computer's random access memory, and the possibility of firstly transferring the entire data and then running an algorithm on the transfered

[1]http://www.v3.co.uk/v3-uk/news/2081263/cern-experiments-generating-petabyte

[2]http://www.treehugger.com/clean-technology/meet-the-worlds-most-powerful-weather-supercomputer.html

[3]http://www.comparebusinessproducts.com/fyi/10-largest-databases-in-the-world

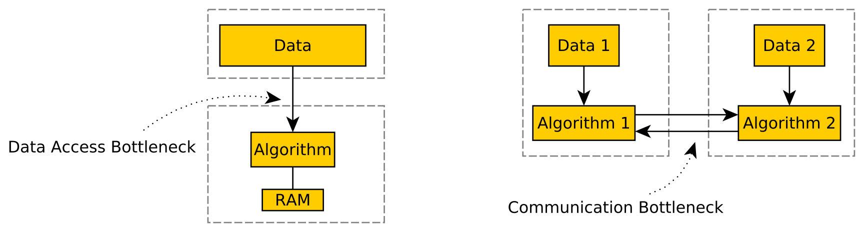


Figure 1.1: Left: An Algorithm that processes massive data sets must deal with the problem of accessing the input data. Right: If massive data sets are processed in a distributed manner, the amount of communication should be as little as possible.

data with random access is hence excluded. Furthermore, if the data is stored on an external disk or on a computer connected through a slow network, such as the Internet, then random access to the input is costly and different data access alternatives must be considered. The alternative that we consider in this thesis is the data streaming model. Streaming algorithms receive the input data as a stream. While the data is arriving, the algorithm is already processing it, and it uses a small random access memory, which is often only polylogarithmic in the size of the input. In Section 1.1 we give a more detailed introduction to streaming algorithms.

Another problem arises when big data is collaboratively processed by different parties at different locations. In this setting, the parties exchange messages in order to coordinate their calculations. Note that if the input data was small, the parties could simply send their input to one central processing unit, which could then process the entire data set locally. However, data transmission is a bottleneck in many applications, since interconnection networks are usually slow, and hence the message exchange should be as little as possible. *Communication Complexity* is the area in theoretical computer science that focuses on the amount of communication that is necessary to collectively compute functions on distributed input data. We provide a brief introduction to communication complexity in Section 1.2.

1.1 Streaming Algorithms

One objective of this PhD thesis is to explore the strengths and the limitations of the *streaming model* of computation. *Streaming algorithms* fall into the category of massive data set algorithms. They address the problem that in many applications, the data that an algorithm is called upon to process is too large to fit into the computer's random access memory. A streaming algorithm uses memory space of sublinear size in the length of its input.

Besides sublinear memory, another characteristic feature of the streaming model is the restriction of the input access method to only sequential access. Whereas algorithms have random access to input data in the RAM model, a *streaming algorithm* reads its input in passes scanning the input from *left* to *right*. We illustrate this point in Figure 1.2.

The memory space restriction is necessary for a meaningful definition of the streaming model: Random access memory of size linear in the input length would allow the copying of the whole input into memory by reading the input in one pass, and this copy of the data could

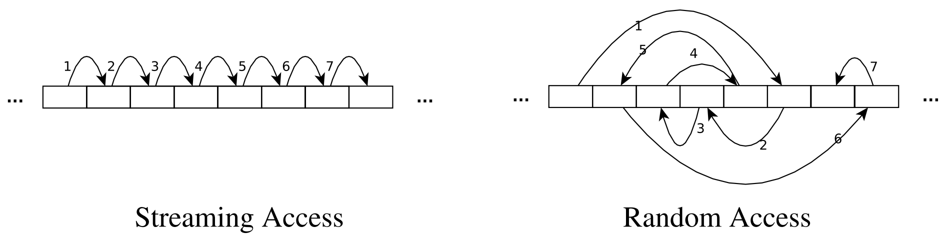


Figure 1.2: Random Access versus Streaming Access. Streaming Algorithms have only sequential access to input data.

then be processed by any linear space algorithm, now with random access.

Streaming videos from the Internet is an example of a streaming algorithm. Video files can be huge, however, at any moment during the visualization of the video, the video player application only requires access to the small portion of the video file that encodes the images and the sound data of that particular moment. The video is hence being downloaded and arrives at the video player application as a data stream. The player visualizes the data as it arrives and discards it immediately afterwards. This procedure allows the visualization of huge video files with a small amount of random access memory.

The main goal in the design of streaming algorithms is to minimize the size of the random access memory of the algorithm. We illustrate this with the following more concrete example:

Problem 1 (Counting the number of distinct elements)**.** *Let $A = A[1], \dots, A[n]$ be a sequence of n elements each taken from a universe $\mathcal{U}$. How many distinct elements appear in A?*

We denote by $F_0(A)$ the number of distinct elements in A. The notation F_0 comes from the fact that the number of distinct elements is the 0th *frequency moment* of a set of frequencies.The frequency moments are defined as follows. For each $u \in \mathcal{U}$ denote by $f_A(u)$ the number of occurrences of u in A. Then the ith frequency moment F_i is defined as $F_i(A) := \sum_{u \in \mathcal{U}} f_A(u)^i$.

To obtain a space efficient algorithm for counting the number of distinct elements in the RAM model, an algorithm could simply keep track of a counter and check for each $u \in \mathcal{U}$ whether u appears in A by sequentially scanning A. This strategy requires only $O(\log |\mathcal{U}|)$ space. It can also be implemented as a streaming algorithm that makes $|\mathcal{U}|$ passes over the input. In the streaming setting, one usually aims for algorithms that perform only a constant number of passes over the input, and the one-pass model is of particular interest: How much memory does a streaming algorithm require if it performs a single pass over the input?

A one-pass streaming algorithm could store one bit for each $u \in \mathcal{U}$ and keep track if u appears in the stream. This algorithm requires $O(|\mathcal{U}|)$ space, and it turns out that this algorithm is optimal among deterministic algorithms: In [AMS96] it is shown that any deterministic one-pass streaming algorithm that outputs a number Y such that $|Y - F_0(A)| \leq 0.1 F_0(A)$ requires $\Omega(|\mathcal{U}|)$ space.

This space lower bound highlights two important properties that are often encountered in the streaming model. Firstly, for many problems, linear space lower bounds exist for deterministic streaming algorithms, and *randomization* is crucial for obtaining streaming algorithms with sublinear space. Secondly, the previous space lower bound uses the notion of *approximation*. In streaming, it is rarely possible to solve problems exactly with sublinear space, and, as a consequence, approximation algorithms are designed. In the case of counting the number of distinct elements, there is an optimal randomized one-pass streaming algorithm that uses $O(1/\epsilon^2 + \log |\mathcal{U}|)$ space and computes a $(1 \pm \epsilon)$-approximation, meaning that it outputs a value Y such that $|Y - F_0(A)| \leq \epsilon F_0(A)$ for any $\epsilon > 0$. This is a work of Kane et al. [KNW10b] and concludes a long line of research on this problem that was initiated by Flajolet and Martin in 1983 [FM83].

1.2 Communication Complexity

Communication Complexity is an area of computer science that has strong connections to streaming algorithms. In the communication setting, the input data is split among multiple parties, and the parties communicate with each other by exchanging messages in order to collectively compute a function of the entire input. In communication complexity the goal is both to design communication protocols that use as little communication as possible, and to prove lower bounds on the amount of communication that is necessary for computing a function. The *determinstic/randomized k-party communication complexity* of a function f is the minimal amount of communication necessary for a deterministic/randomized protocol that is executed by the k parties to compute f. In the following, we will refer to the parties that execute a communication protocol as players.

As an example, consider the two-party communication problem of determining whether two n-bit strings are equal. This problem is illustrated in Figure 1.3 and is defined as follows.

Problem 2 (Equality of two n-bit strings)**.** *Alice holds a binary string $X \in \{0,1\}^n$ and Bob holds a binary string $Y \in \{0,1\}^n$. Alice and Bob exchange messages in order to determine whether $X = Y$. How much communication is necessary to achieve this task?*

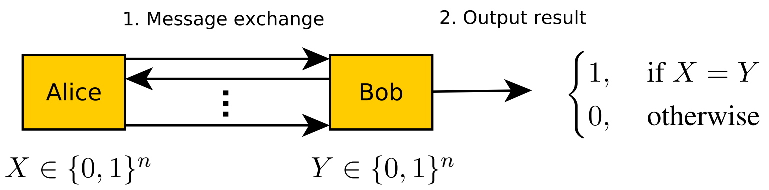


Figure 1.3: Two-Party Communication Setting. Here, Alice and Bob both hold n-bit strings and they want to determine whether their strings are equal or not. Alice and Bob communicate in order to achieve this task. In this illustration, Bob outputs the result.

Consider the following deterministic protocol for the equality problem. Alice sends the bit $X[1]$ to Bob and upon reception, Bob verifies if it matches $Y[1]$. If there is a mismatch, Bob outputs 0. In case of equality, Bob sends bit $Y[2]$ to Alice and she compares it to $X[2]$ and outputs 0 in case of a mismatch. This procedure continues until a mismatch is found or until all bits have been compared. Clearly, if $X = Y$, then this protocol has a communication cost of

$\Omega(n)$. If $X \neq Y$, the worst case communication cost of this protocol is still $\Omega(n)$ since it might be that $X[i] = Y[i]$ for all $i < n$ and $X[n] \neq Y[n]$. It is known that the deterministic two-party communication complexity of the equality function is $\Omega(n)$ (see for instance [KN97]), and this protocol is hence optimal.

Here randomization helps to substantially decrease the amount of communication. Given an appropriate hash function h, Alice computes the hash value $h(X)$ and sends it to Bob. Bob then computes the hash value $h(Y)$ and checks whether $h(X) = h(Y)$. There are hash functions h such that $|h(X)| \in O(\log n)$, the probability that $h(X) = h(Y)$ for $X \neq Y$ is at most $1/n$, and if $X = Y$ then $h(X) = h(Y)$. This implies a randomized two-party communication protocol with communication cost $O(\log(n))$. See [KN97] for details. This example shows that there are functions for which there is an exponential gap between the deterministic and the randomized communication complexity.

In this thesis, we are mainly interested in *one-way communication*. In this setup, the input is split among k players, and the communication is one-way: Player one sends a message to player two, who upon reception of the message sends a message to player three. This procedure continues until player k receives a message from player $k-1$, and player k outputs the result. Consider for now the case $k = 2$ and denote the two players by *Alice* and *Bob*. Then Alice computes a message from her part of the input and sends this message to Bob. Upon reception of this message, Bob computes the output as a function of his part of the input and the message of Alice. The randomized communication protocol for the equality function as discussed above is a one-way two-party protocol.

There is an inherent link between one-way communication and one-pass streaming algorithms: A one-pass streaming algorithm with space s for problem P serves also as a one-way two-party communication protocol with message size $O(s)$ for problem P if the input stream A is split among the two players such that Alice gets a prefix and Bob gets the remaining suffix. To see this, Alice runs the streaming algorithm on her input and sends the resulting memory state to Bob. Then, Bob initializes his memory with Alice's message and continues the streaming algorithm on his input. This connection implies that a lower bound on the size of the message in the one-way two-party communication setting is also a lower bound on the memory requirements of a streaming algorithm. Approaching streaming algorithms from the communication perspective therefore provides valuable insight.

We illustrate this link with the equality function discussed above. Suppose that a one-pass streaming algorithm with space s can decide whether the first half of an input stream A of length $2n$ equals the second half, that is $\forall 1 \leq i \leq n : A[i] = A[i+n]$. Such a streaming algorithm then gives rise to a one-way two-party communication protocol with communication cost s for the equality function. As previously, denote Alice's input by X and Bob's input by Y. Then Alice runs the streaming algorithm on X and sends the memory state after processing $X[n]$ to Bob. Bob then initializes his memory with the message he received from Alice and continues the streaming algorithm on Y. See Figure 1.4 for an illustration.

As we already pointed out, the deterministic two-party communication complexity of the equality function is $\Omega(n)$, and since one-way two-party communication is a special case of two-party communication, the deterministic one-way two-party communication complexity of the equality function is also $\Omega(n)$. A deterministic streaming algorithm for the equality problem

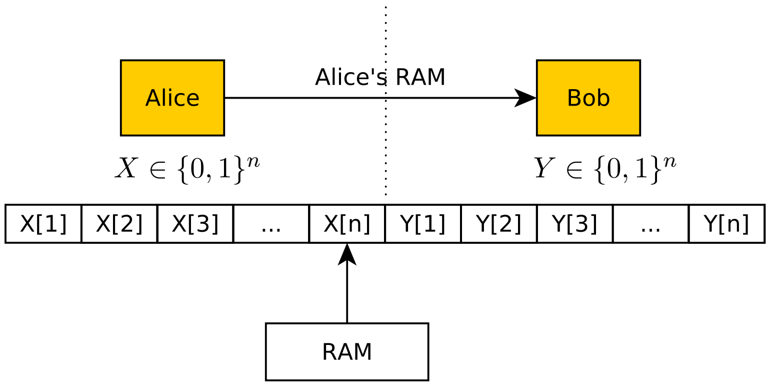


Figure 1.4: Connection between Streaming Algorithms and Two-Party Communication. A one-pass streaming algorithm with space s also serves as a one-way two-party communication protocol with message size $O(s)$ if the input stream A is split among the two players such that Alice gets the first half and Bob gets the second half as input. Alice runs the streaming algorithm on her input and sends the resulting memory state to Bob. Bob then initializes his memory with Alice's message and continues the streaming algorithm on his input.

hence also requires $\Omega(n)$ space. We repeat the previously discussed argumentation: If there was a deterministic streaming algorithm with space $o(n)$ for the equality problem, then this algorithm would give rise to a deterministic one-way two-party communication protocol for the equality function with message size $o(n)$. However, this contradicts the $\Omega(n)$ lower bound on the deterministic one-way two-party communication complexity of the equality function.

In this thesis, we study different problems in the streaming and the one-way two-party communication setting. In the following section we briefly and informally introduce these problems in order to provide the reader with an overview of the research areas that we consider in this document. Then, in Chapter 2 we formally introduce the streaming setting and the communication setting. This then allows us to fully discuss our contributions in detail. In Chapter 3, we provide the contexts for each of the four problems in this thesis, and we give an overview of our results. Then, in Chapters 4 to 7 all technical details of these works are discussed.

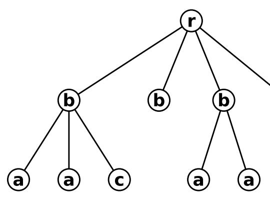
1.3 Considered Problems

Maximum Matching in the Streaming Model [KMM12]. Given an unweighted graph $G = (V, E)$, a *graph stream* is a sequence of the edges of G. A *matching* in a graph is a vertex disjoint subset of the edges $M \subseteq E$. How large a matching can be computed by a streaming algorithm that receives a graph stream as its input with sublinear space?

Semi-Matchings in the Streaming Model and in Two-Party Communication [KR13].

A semi-matching in a bipartite graph $G = (A, B, E)$ is a subset of edges $S \subseteq E$ such that *all* A vertices are matched in S. Here, multiple A vertices may be matched to the same B vertex. The goal is to find a semi-matching with a small maximal degree of the B vertices. Given a graph stream of a bipartite graph as input, how well can this be done in the streaming setting and how much memory is required? Given a constraint on the message size, how well can this be done in the one-way two-party communication model?

Validating XML Documents in the Streaming Model [KM12]. A *well-formed XML document* (eXtended Markup Language) is a well-parenthesized sequence of opening and closing tags. Such documents describe rooted, unranked, vertex labeled trees. A DTD (Document Type Definition) is a set of local validity constraints: For each label in the tree, the DTD specifies a regular expression. Then a document is valid against a DTD, if for each node the sequence of labels of its children fulfills the regular expression specified in the DTD for the label of the node. Figure 1.5 illustrates this setup.



$$\begin{array}{lll} r & \rightarrow & b^* c^+ \\ a & \rightarrow & \epsilon \\ b & \rightarrow & a^* c^* \\ c & \rightarrow & \epsilon \end{array}$$

Figure 1.5: This tree is valid against the DTD described on the right side of the figure: For each node of the tree, the sequence of labels of its children fulfills the regular expression attached to the label of the node.

Are there sublinear space streaming algorithms that, given an XML document and a DTD, can check whether the XML document is valid against the DTD?

Budget Error-Correcting under Earth-Mover-Distance. [KYZ13] The Earth-Mover-Distance between two point sets of cardinality n is defined as the weight of a minimum weight perfect matching between the two point sets, where the weight of an edge between two points is defined as the distance of the points. We study the following one-way two-party communication problem: Alice and Bob are each given a set of n points on a d-dimensional grid $[\Delta]^d$. Alice sends a message of size k ($k \in o(n)$) to Bob, and Bob uses this message to adjust his point set *towards* Alice's point set such that the Earth-Mover-Distance between the two sets decreases. How well can such an adjustment be done?

1. INTRODUCTION

Chapter 2

Models of Computation

2.1 The Streaming Model

A data stream of length n is a sequence of items $A = A[1] \dots A[n]$ each coming from a finite universe $\mathcal{U}$. The universe $\mathcal{U}$ depends on the application at hand; it may be numbers, letters, edges of a graph, XML tags or any other finite set. In this thesis, we assume that accessing a stream element $A[i]$ can be done in time $O(1)$. Furthermore, for ease of presentation, we assume that the length of the input stream n is known in advance to our streaming algorithms. This is not a real limitation: All of our algorithms can be adapted to work without the knowledge of n in advance. We now formally define a streaming algorithm.

Definition 1 (Streaming Algorithm)**.** *A $p(n)$-pass* streaming algorithm ***S** with space $s(n)$ and update time $t(n)$ is an algorithm such that, for every input stream $A = A[1] \dots A[n]$ with $A[i] \in \mathcal{U}$ for a universe $\mathcal{U}$:*

1. ***S** performs at most $p(n)$ passes on stream A,*
2. ***S** maintains a random access memory of size $s(n)$,*
3. ***S** has running time $O(t(n))$ between two consecutive read operations from the stream.*

Furthermore, preprocessing time (the time before the first read operation) and postprocessing time (the time after the last read operation and the output of the result) is $O(t(n))$. We assume that read operations on any stream require constant time.

2.1.1 A Brief History and Applications

Streaming algorithms find applications in settings where the input data is too large to fit into the computer's working memory. The most natural setting is the processing of *real* data streams. Consider a network router that is supposed to compute statistics on the data packages that run through the router such as the number of different IP addresses (distinct elements problem) or the most addressed target IP addresses (heavy hitters problem). In settings of this kind, the application imposes linear access to the input data and only one pass is possible.

Another field of application is that of processing massive data sets that are stored on external hard disks or on computers that are connected through a network or the Internet. This is a typical situation for databases. In these settings, random access to the input data is very costly, and accessing this data by a data stream is a good alternative that is pursued in practice. These situations also justify the study of multi-pass streaming algorithms, since the data can be read several times.

Streaming algorithms were already studied as early as in 1978. Munro and Paterson [MP78] studied the relationship of the number of passes and the required memory space for selecting the kth largest element of a stream. This work, as well as the previously mentioned work by Flajolet and Martin [FM83] on approximate counting of streaming data, are considered to be some of the first works on streaming algorithms. A milestone in the area of streaming algorithms is the paper "The space complexity of approximating the frequency moments" by Alon, Matias, and Szegedy [AMS96] who received the Gödel prize in 2005 for this work. We quote the award committee: " This concise work laid the foundations of the analysis of data streams using limited memory."[1]. This paper has since been cited more than 1000 times.

Today, there is a huge literature on streaming algorithms. Concrete problems from areas such as statistics, bioinformatics, text algorithms, computational geometry, graph problems, linear algebra, databases, and many others have been studied in the streaming model. For an overview about streaming algorithms, we refer the reader to [Mut05]. A list of open problems in the data streaming area can be found on the website http://sublinear.info/.

2.1.2 Graph Streams

Definition 2 (Graph Stream). *A* graph stream *is a sequence of the edges of a graph.*

For the sake of a clear presentation, we again suppose that the number of vertices and edges of a graph are known in advance to our streaming algorithms that process graph streams.

Graph streams were firstly studied by Henzinger et al. [HRR99] and form a very active research area today. Many graph problems, such as independent set [HHLS10], matching problems [McG05, GKK12, KMM12], counting triangles [JG05, BFL$^+$06] and arbitrary subgraphs [KMSS12], connectivity of nodes in directed graphs [GO13], sparsification [AGM12], and many others have since been studied in the streaming model.

Feigenbaum et al. showed in [FKM$^+$05] that testing many basic properties such as connectivity and bipartiteness requires $\Omega(n)$ space, where n is the number of vertices of a graph. This justifies the study of the so-called *semi-streaming model*, which was introduced by Muthukrishnan [Mut05]. In the semi-streaming model, a streaming algorithm is allowed to use space of size $O(n \operatorname{polylog} n)$. Throughout this document, we will write $\tilde{O}(n)$ for $O(n \operatorname{polylog} n)$.

The difficulty of graph problems in the streaming model depends heavily on the arrival order of the incoming edges. At least four edge arrival orders have been studied in the literature.

1. **Adversarial order.** An algorithm does not make any assumption on the order of the incoming edges. Most works in the literature consider this model.

[1]This quote is taken from: http://www.sigact.org/Prizes/Godel/2005.html

2. **Vertex arrival order.** Consider a bipartite graph $G = (A, B, E)$. Edges incident to the same A vertex arrive subsequently [GKK12, Kap13]. In [BYKS02] and [BFL$^+$06], a graph stream in this order is called an *incidence stream* and this notion is also used for general graphs.

3. **Random arrival order.** The edges arrive in uniform random order [DLOM02, GMV06, GM09, KMM12]. Studying this model is motivated by the fact that some algorithms behave very well in practice, but they fail on a few rather artificial arrival orders. The adversarial order is considered to be too pessimistic and too far from reality. The random arrival order is a way of conducting an average-case analysis. Note that a random-order arrival analysis still considers a worst-case input graph.

4. **Best order.** Edges arrive in the desired order of an algorithm. This model was studied in the context of *stream checking* [DSLN09]. [1] We are listing this setting for completeness, and will not further consider it.

2.1.3 Bidirectional Streaming Algorithms

For some problems, allowing a streaming algorithm to perform a pass in reverse direction saves a lot of space. Magniez et al. show in [MMN10] that checking whether a sequence of parentheses of length n is well-parenthesized can be checked by a streaming algorithm in one pass with space $\tilde{O}(\sqrt{n})$. Furthermore, it is shown in [CCKM10] and independently in [JN10] that a p-pass streaming algorithm requires $\tilde{O}(\sqrt{n}/p)$ space. Magniez et al. also present in [MMN10] a bidirectional two-pass algorithm that performs one pass from left to right, and one pass from right to left, and this algorithm uses memory space of $O(\log^2 n)$.

We observe a similar phenomenon when checking validity of XML documents that encode binary trees [KM12]. For details see Section 6.3. This phenomenon was also observed by François and Magniez [FM13] when checking priority queues with timestamps.

Definition 3 (Bidirectional Streaming Algorithm)**.** *A streaming algorithm is called* bidirectional *if it makes at least one pass from left to right and one pass from right to left.*

2.1.4 Streaming with Auxiliary Streams

Streaming with auxiliary streams is also known as streaming with external memory, and has been studied for instance in [GS05, GHS06, BJR07, BH12, KM12]. In this model, besides the input stream, the streaming algorithm has access to auxiliary streams onto which the algorithm can write, and from which the algorithm can read. This equips the streaming algorithm with a huge amount of external memory. Note, however, that the access method to external memory is sequential. Furthermore, we assume that the length of an auxiliary stream is $O(n)$, where n

[1] A computationally unbounded prover computes a graph property on a massive input graph and has to convince a space bounded verifier that this property holds. To this end, the prover sends a graph stream of the input graph in an order that makes the verification easier for the verifier. For instance, checking whether a graph has a perfect matching can be done in $O(\log n)$ space in this model [DSLN09] while it requires $\Omega(n^2)$ space in adversarial order [GO13].

denotes the length of the input stream. We define a streaming algorithm with auxiliary streams as follows:

Definition 4 (Streaming Algorithm with auxiliary Streams)**.** *A $p(n)$-pass* streaming algorithm ***S*** *with $k(n)$ auxiliary streams, space $s(n)$, and update time $t(n)$ is an algorithm such that, for every input stream $A = A[1] \dots A[n]$ with $A[i] \in \mathcal{U}$ for a universe $\mathcal{U}$:*

1. ***S** has read/write access to $k(n)$ auxiliary streams, the input stream is read-only,*
2. ***S** performs at most $p(n)$ passes in total on the input stream π and the k auxiliary streams,*
3. ***S** maintains a random access memory of size $s(n)$,*
4. ***S** has running time $t(n)$ between two consecutive read or write operations.*

Furthermore, preprocessing time (the time before the first read operation) and postprocessing time (the time after the last read operation and the output of the result) is $O(t(n))$. We assume that read and write operations on any stream require constant time.

Auxiliary streams allow a streaming algorithm to sort the input data. *Merge sort* can be implemented as a streaming algorithm with three auxiliary streams and $O(\log n)$ passes [GS05] (here n is the input length). For an illustration of the streaming model with auxiliary streams we provide this implementation of merge sort as an example in Algorithm 1. For the sake of simplicity, Algorithm 1 assumes that the input length is 2^l for some $l > 0$.

Algorithm 1 Merge Sort as a Streaming Algorithm with Auxiliary Streams

```
Require: unsorted data of length $2^l$ on auxiliary stream 1
1: for $i = 0 \dots l-1$ do
2:     copy data in blocks of length $2^i$ from stream 1 alternately onto stream 2 and stream 3
3:     for $j = 1 \dots 2^{l-i-1}$ do
4:         merge($2^i$)
5:     end for
6: end for
```

merge(b) reads simultaneously the next b elements from stream 2 and stream 3, and merges them onto stream 1. The for loop in Line 3 of Algorithm 1 requires one read pass on stream 2, one read pass on stream 3, and one write pass on stream 1. See Figure 2.1 for an illustration.

	line 2 (copy)	line 3 (merge)
str 1:	B_1 B_2 B_3 B_4 $\cdots B_{2^{l-i}}$	B_{12} $B_{34} \cdots B_{2^{l-i-1}2^{l-i}}$
str 2:	B_1 B_3 $\cdots B_{l-i-1}$	B_1 B_3 $\cdots B_{l-i-1}$
str 3:	B_2 B_4 $\cdots$ B_{l-i}	B_2 B_4 $\cdots$ B_{l-i}

Figure 2.1: Left: Illustration of the copy operation in Line 2 of Algorithm 1. Blocks from stream 1 are copied alternately onto stream 2 and stream 3. Right: Illustration of the merge operations executed within the for loop of Line 3 of Algorithm 1. The B_i are sorted blocks. All blocks B_i and B_{i+1} are merged into a sorted block $B_{i(i+1)}$.

Streaming with auxiliary streams is an interesting model for problems that do not allow sublinear space streaming algorithms without auxiliary streams. The auxiliary streams are implemented in external memory to which the model only assumes linear access. Since in practice linear access is less costly than random access, good solutions may be obtained.

2.2 Communication Complexity

Communication Complexity focuses on how much communication is necessary if multiple parties aim to collectively compute a function that depends on the data of all parties. The parties are allowed to exchange messages, and the communication cost of a communication protocol is then the amount of bits that are communicated in total. Besides the development of efficient protocols, the main goal is to prove lower bounds on the amount of communication that is necessary to fulfill a task.

The study of communication complexity was initiated by Andrew Yao in 1979 [Yao79]. Many different communication settings have since been studied in the literature (deterministic, randomized, non-deterministic, one-way, blackboard, number-on-forehead, . . .) and providing an overview exceeds by far the scope of this thesis. We refer the reader to the book of Kushilevitz and Nissan [KN97] for an introduction to Communication Complexity.

Communication Complexity is used to obtain results in many related research areas. As already discussed in Section 1.2, space lower bounds for streaming algorithms may be proven by communication lower bounds. Furthermore, communication complexity has already been successfully applied to prove results in decision tree complexity, data structures, boolean circuit and many others [KN97]. Besides the use of communication complexity as a tool for proving results in related fields, the study of the communication complexity of functions itself constitutes an important research area. We have already mentioned that communication is the dominant bottleneck in many applications. Consequently, communication protocols must be designed that do not exchange more data then absolutely necessary.

In this thesis, we focus on the one-way two-party communication setting. In this setting, we have two players, Alice and Bob, and the input is split amongst them. Alice sends a single message to Bob, and Bob then computes and outputs the result of the protocol.

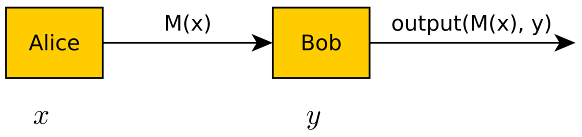


Figure 2.2: One-way Two-Party Communication. The input x, y is split among Alice and Bob such that Alice has x and Bob has y. Alice computes a message $M(x)$ of her input and sends it to Bob. Upon reception of the message, Bob computes the output of the protocol as a function of Alice's message and his input y.

Consider the *Augmented Indexing* problem.

Problem 3 (Augmented Indexing)**.** *Let $X = X[1], \dots, X[n]$ where $X \in \mathfrak{U}^n$ for some universe $\mathfrak{U}$. Let $I \in [n]$. Alice is given X, Bob is given I and $X[I+1], \dots, X[n]$. Alice sends message M_{AI} to Bob and upon reception Bob outputs $X[I]$.*

This problem is a hard problem in the one-way communication setting. Many versions of this problem have been studied in the literature [CCKM10, JN10], and we provide a proof in Lemma 50 that shows that any possibly randomized protocol with error probability less than $1/3$ for this problem requires a message of size $\Omega(n \log |\mathcal{U}|)$. Note that in the usual two-party communication setting where Alice and Bob can freely exchange messages, $O(\log n + \log |\mathcal{U}|)$ bits are sufficient: Bob simply sends I to Alice and Alice sends $X[I]$ back to Bob.

A particularity of the one-way two-party communication setting is that Alice does not obtain any information about Bob's input during the run of the protocol. Therefore, Alice can not adapt her message to Bob's needs. Instead, Alice has to send a sketch of her data from which Bob can recover the necessary information. We are faced with this situation multiple times in this thesis.

Chapter 3

Contributions

In this section, we discuss the contributions of this PhD thesis. We provide the concrete contexts of the problems at hand, and we detail related works. Our contributions are based on the following articles:

- [KMM12]: Christian Konrad, Frédéric Magniez and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Proceedings of the 24th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, 2012.

- [KR13]: Christian Konrad and Adi Rosén. Approximating Semi-Matchings in Streaming and in Two-Party Communication. In *Proceedings of 40th International Colloquium on Automata, Languages and Programming*, 2013.

- [KM12]: Christian Konrad and Frédéric Magniez. Validating XML Documents in the Streaming Model with External Memory. In *Proceedings of 15th International Conference on Database Theory*, 2012.

- [KYZ13]: Christian Konrad, Wei Yu and Qin Zhang. Budget Error-Correcting under Earth-Mover-Distance. Technical Report.

3.1 Matching in the Streaming Model

Our first contribution concerns the computation of *matchings* in the streaming model. Given a graph $G = (V, E)$, a matching is a vertex disjoint subset of edges $M \subseteq E$. Then a *maximum matching* is a matching of maximal size, and we write M^* to denote an arbitrary but fixed maximum matching. For $0 < c < 1$, we say that an algorithm computes a c-approximation to the maximum matching problem if it outputs a matching M' such that

$$|M'| \geq c|M^*|.$$

3.1.1 History of the Matching Problem in the Streaming Model

Computing matchings in the streaming model was firstly addressed by Feigenbaum et al. in 2004 [FKM$^+$04]. In this work, the authors assume that the edge stream is in adversarial order. They present an $O(\frac{\log(1/\epsilon)}{\epsilon})$-pass $(2/3 - \epsilon)$-approximation semi-streaming (meaning $\tilde{O}(n)$ space) algorithm for maximum bipartite matching for any $0 < \epsilon < 1/3$. This algorithm uses the fact that a *maximal matching* can be computed in one-pass with $O(n \log n)$ space by the following simple Greedy algorithm: start with an empty matching M and put an incoming edge e into M if $M \cup \{e\}$ is still a matching. It is well-known that a maximal matching is of size at least $1/2$ times the size of a maximum matching. Hence, the Greedy algorithm is a $1/2$-approximation algorithm to the maximum matching problem. Furthermore, they present a $1/6$-approximation algorithm for weighted maximum matching. In a weighted graph, the weight of a matching is the sum of the weights of the edges in the matching. A c-approximation algorithm to the weighted matching problem ($0 < c < 1$) outputs a matching of weight at least c times the weight of a matching of maximal weight.

Since then, a multitude of results on matchings in the streaming model have appeared. Most of these results are multi-pass algorithms for both, the unweighted and the weighted setting. We are mainly interested in the one-pass setting and the case of unweighted graphs. Before discussing related works to this setting, we provide an overview about the currently best algorithms in the one-pass, weighted graphs setting as well as in multi-pass settings. The following results are in the semi-streaming setting, meaning that algorithms are allowed to use $O(n \operatorname{polylog} n)$ space, and the edge stream is in adversarial order:

- **One-pass algorithms for weighted graphs:** In the one-pass setting, the already mentioned $1/6$-approximation for computing a weighted matching in general graphs of [FKM$^+$04] was improved by McGregor [McG05] to a factor of $1/5.82$. Subsequently, Zelke [Zel12] obtained a $1/5.58$-approximation. The currently best semi-streaming algorithm for weighted matching in one-pass is due to Epstein et al. [ELMS11] and has an approximation factor of $1/4.91$.

- **Multi-pass algorithms for unweighted graphs:** In the unweighted case, Ahn and Guha show that a $1 - \epsilon$ approximation for bipartite graph can be obtained using $O(\epsilon^{-2} \log\log \epsilon^{-1})$ passes [AG11]. For general unweighted graphs, McGregor gives in [McG05] a $1/(1 + \epsilon)$ approximation that makes $O((1/\epsilon)^{1/\epsilon})$ passes.

- **Multi-pass algorithms for weighted graphs:** In the weighted setting, Ahn and Guha describe a $1 - \epsilon$ approximation streaming algorithm for weighted bipartite graphs with $O(\epsilon^{-2} \log \epsilon^{-1})$ passes in [AG11]. For general weighted graphs, the currently best algorithm is also due to Ahn and Guha and computes a $(1 - \epsilon)$-approximation using $O(\epsilon^{-4} \log n)$ passes. To the authors best knowledge, there is no streaming algorithm for weighted matching in general graphs that computes a $1 - \epsilon$ approximation and the number of passes depends only on ϵ.

One-pass algorithms for unweighted graphs. The already mentioned Greedy matching algorithm that computes a $1/2$ approximation to the maximum matching problem was used in

2004 in the work of Feigenbaum et al. [FKM$^+$04] and is still the best semi-streaming algorithm for computing unweighted matchings in one-pass if the input stream is in adversarial order. The well-known Karp-Vazirani-Vazirani (KVV) online algorithm [KVV90] can be seen as a semi-streaming algorithm if the input stream is in vertex arrival order. The KVV algorithm is randomized and computes a $1-1/e \approx 0.63$ approximation to the maximum matching problem. A deterministic $1 - 1/e$ approximation algorithm for input that is in vertex arrival order was given by Goel et al. in 2012 [GKK12].

It is a major open question whether it is possible to beat the approximation guarantee of $1/2$ of the Greedy algorithm for streams in adversarial order. On the negative side, Goel et al. [GKK12] showed that for any $\epsilon > 0$, a streaming algorithm that computes a $2/3 + \epsilon$ approximation requires $n^{1+\Omega(1/(\log \log n))}$ space which proves that there is no semi-streaming algorithm with such an approximation ratio. Recently, Kapralov showed that semi-streaming algorithms for streams in vertex arrival can not obtain an approximation guarantee better than $1 - 1/e$ [Kap13]. This result renders the KVV algorithm and the deterministic counterpart of [GKK12] tight. Since the vertex arrival order is a special case of the adversarial order, the lower bound of $1 - 1/e$ also holds for algorithms for streams in adversarial order.

3.1.2 Our Contribution

In a joint work with Frédéric Magniez and Claire Mathieu [KMM12], we show that there is a one-pass semi-streaming algorithm with *expected approximation ratio* strictly larger than $1/2$ if the input stream is in uniform random order. In the random arrival order setting, we measure the quality of an algorithm by taking the expectation over all orderings of the graph edges. In the following, we will omit the term 'expected' since this is the only meaningful definition of the approximation ratio in the random arrival setting. We obtain a $1/2 + 0.005$ approximation algorithm for bipartite graphs (**Theorem 1**). We extend this result to non-bipartite graphs and we obtain a $1/2 + 0.003$ approximation algorithm (**Theorem 2**).

Furthermore, in the adversarial order model, we show that it is possible to improve on Greedy if an algorithm is allowed to perform two passes. We design a simple randomized algorithm for bipartite graphs (**Theorem 4**), and two deterministic algorithms for bipartite (**Theorem 5**) and non-bipartite (**Theorem 6**) graphs.

The starting point for our algorithms is a simple 3-pass semi-streaming algorithm for bipartite graphs on adversarial order that has an approximation ratio strictly better than $1/2$. Our one-pass algorithm on random order and our two-pass algorithms simulate this algorithm with fewer passes.

Details of our contribution can be found in Chapter 4.

3.2 Semi-Matchings in Streaming and in Two-Party Communication

A *semi-matching* $S \subseteq E$ in a bipartite graph $G = (A, B, E)$ can be seen as an extension of a matching in that it is required that *all* A vertices are matched to B vertices. This is generally not possible in an injective way, and therefore we now allow the matching of multiple A vertices to the same B vertex. Typical objectives here are to minimize the maximal number of A vertices that are matched to the same B vertex, or to optimize with respect to even stronger balancing constraints. The term 'semi-matching' was coined by [HLLT03] and also used in [FLN10, GKS11, CHSW12], however, the problem had already previously been intensely studied in the scheduling literature [ECS73, Hor73, ANR95, Abr03, LL04]. We stick to this term since it nicely reflects the structural property of entirely matching one bipartition of the graph.

The semi-matching problem captures the problem of assigning a set of unit-length jobs to a set of identical machines with respect to assignment conditions expressed through edges between the two sets. The objective of minimizing the maximal number of jobs that a machine receives then corresponds to minimizing the *makespan* of the scheduling problem. Optimizing the cost function $\sum_{b \in B} \deg_S(b)(\deg_S(b) + 1)/2$, where $\deg_S(b)$ denotes the number of jobs that a machine b receives in the semi-matching S, corresponds to minimizing the *total completion time* of the jobs (optimizing with respect to this cost function automatically minimizes the maximal degree as well).

3.2.1 Optimality of a Semi-Matching and the Notion of Approximation

It is well known that matchings are of maximal size if they do not admit *augmenting paths* [Ber57]. Augmenting paths for matchings correspond to *degree-minimizing paths* for semi-matchings. They first appeared in [HLLT03] under the name of *cost-reducing-paths*, and they were used for the computation of a semi-matching that minimizes a certain cost function. We use the term 'degree-minimizing-path' since it is more appropriate in our setting. A degree-minimizing path starts at a B node of high degree, then alternates between edges of the semi-matching and edges outside the semi-matching, and ends at another B node of smaller degree. Flipping the semi-matching and non-semi-matching edges of the path then generates a new semi-matching such that the large degree of the start node of the path is decreased by 1, and the small degree of the end node of the path is increased by 1. We define an *optimal semi-matching* as one that does not admit any degree-minimizing paths. It was shown in [HLLT03] that such a semi-matching is also optimal with respect to a large set of cost functions, including the minimization of the maximal degree as well as the minimization of the total completion time.

Since an optimal semi-matching minimizes many convex cost functions, there is not only one meaningful definition of what an approximation to the semi-matching problem should be. We will consider a notion that is already used in [ANR95]. We say that an algorithm is a c-approximation algorithm to the semi-matching problem if for any input graph, it outputs a semi-matching S such that $\deg\max S \le c \deg\max S^*$, where S^* is an optimal semi-matching. In [CHSW12], the semi-matching problem is studied in the distributed setting, and the cost function $\sum_{b \in B} \binom{\deg_S(b)+1}{2}$ is used. These notions are not comparable.

3.2.2 Related Results on the Semi-Matching Problem

The semi-matching problem was firstly studied by Horn [Hor73] and independently by Bruno et al. [BCS74], and both designed an $O(|V|^3)$ algorithm. At present, the best existing algorithm for computing an optimal semi-matching runs in time $O(\sqrt{|V|}|E| \log |V|)$ [FLN10, GKS11] where $V = A \cup B$. Furthermore, in [GKS11] a randomized algorithm with time complexity $O(|V|^{\omega} \log^{1+o(1)} |V|)$ is given, where ω is the exponent of the best known matrix multiplication algorithm. Since $\omega \leq 2.38$, this algorithm improves on the $O(\sqrt{|V|}|E| \log |V|)$ time algorithm for dense graphs. The semi-matching problem has not yet been studied in the streaming setting and the communication setting prior to our work. In the online setting, a $\lceil \log(n) + 1 \rceil$-approximation online algorithm is given in [ANR95], where the maximal degree is approximated. In this model, an A vertex comes in together with its incident edges, and the A vertex has to be matched immediately and irrevocably. Their algorithm can also be seen as a one-pass streaming algorithm if the input stream is in vertex arrival order. Recently, the semi-matching problem was studied in the distributed setting. They show that a 2-approximation to the semi-matching problem can be computed in $O(\Delta^5)$ time, where Δ is the maximal degree in the graph. They consider the notion of approximation with respect to the cost function $\sum_{b \in B} \binom{\deg_S(b)+1}{2}$. It can be shown that their algorithm is a $\lceil \log(n+1) \rceil$-approximation if the cost function $\deg \max S$ for a semi-matching S is considered.

3.2.3 Our Contribution

In a joint work with Adi Rosén [KR13], we initiate the study of the semi-matching problem in the streaming and the communication settings. We present a deterministic one-pass streaming algorithm that for any $0 \leq \epsilon \leq 1$ uses space $\tilde{O}(n^{1+\epsilon})$ and computes an $O(n^{(1-\epsilon)/2})$ approximation to the semi-matching problem (**Theorem 7**). Furthermore, we show that with $O(\log n)$ passes we can compute an $O(\log n)$ approximation with space $\tilde{O}(n)$ (**Theorem 8**).

In the one-way two-party communication setting, we show that for any $\epsilon > 0$, deterministic communication protocols that compute an $O(n^{\frac{1}{(1+\epsilon)c+1}})$ approximation to the semi-matching problem require a message of size at least cn bits (**Theorem 11**). We present two deterministic protocols communicating n and $2n$ edges that compute an $O(\sqrt{n})$ approximation and an $O(n^{1/3})$ approximation, respectively (**Theorem 9**).

While it was known that an optimal semi-matching necessarily contains a maximum matching [HLLT03], we show that there is a hierarchical decomposition of an optimal semi-matching into maximum matchings. Similarly, we show that semi-matchings that do not admit length-two degree-minimizing paths can be decomposed into maximal matchings. The latter result allows us to prove that the maximal degree of a semi-matching that does not admit a length-two degree-minimizing path is at most $\lceil \log(n+1) \rceil$ times the maximal degree of an optimal semi-matching (**Theorem 12**).

3.3 Validity of XML Documents

An *XML document* is a linear encoding of a finite unranked labeled tree using *tags*. In the usual XML notation an opening tag of a label x of the tree is denoted by $\langle x \rangle$ and a closing tag of x is denoted by $\langle / x \rangle$. We will use the more concise notation x for an opening tag and $\overline{x}$ for a closing tag. The XML encoding of a tree is obtained by a depth-first left-to-right traversal: When a node x is seen in a down-step in the traversal then an opening tag x is outputted, and if a node x is seen in an up-step in the traversal then the closing tag $\overline{x}$ is outputted. For a tree t we denote the corresponding XML document by $\mathrm{XML}(t)$.

An important property of an XML document is *well-formedness*. An XML document is well-formed if it is well-parenthesized meaning that an opening tag is correctly closed by its corresponding closing tag. In other words, an XML sequence X is well-formed if there is a tree t such that $X = \mathrm{XML}(t)$. Well-formedness is illustrated in Figure 3.1.

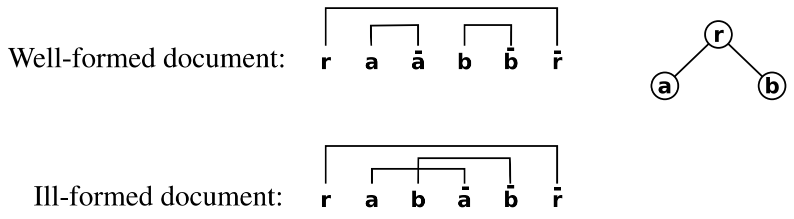


Figure 3.1: Well-formedness of an XML document. A well-formed XML document is well-parenthesized.

XML documents are an important encoding scheme for information. The well-known HTML language (HyperText Markup Language) is an XML language describing the structure and the content of Web pages. Many applications allow to export and import data in XML format. For such applications, it is necessary to restrict the layout of the XML documents that the application is processing. An *XML schema* is a set of constraints that a particular XML document has to fulfill. Nowadays, there are many XML schema languages that allow to express such constraints. The most well-known are DTD (Document Type Definition), EDTD (Extended DTD), XML Schema[1] and RELAX NG (REgular LAnguage for XML Next Generation).

In this document, we are mostly interested in DTDs. DTDs are sets of local validity constraints: For each label in the tree, the DTD specifies a regular expression. Then an XML document is valid against a DTD, if for each node, the sequence of labels of its children fulfills the regular expression specified in the DTD for the label of the node. We illustrated this setup already in Figure 1.5 in Section 1.3. Note that only well-formed XML documents correctly encode trees. For this reason, a requirement for an XML document to be valid is that the XML document is well-formed.

In the following, we study the DTD-validity problem in the streaming model. Given a DTD, does a data stream encode an XML document that is valid against the DTD? We will report now on related works to this problem, and we discuss then our results.

3.3.1 Related Work

Prior works on the DTD-validity problem in the streaming model essentially try to characterize those DTDs for which validity can be checked by a finite-state automaton [SS07]. Under the assumption that DTDs are of constant size, this implies a one-pass deterministic streaming algorithm with constant memory. Concerning arbitrary DTDs, two approaches have been

[1]Note the capital S here.

considered in [SV02]. The first one leads to an algorithm with memory space which is linear in the height of the XML document. The second one consists of constructing a refined DTD of at most quadratic size, which defines a similar family of tree documents as the original one, and against which validation can be done with constant space. Nonetheless, for an existing document and DTD, the latter requires that both, documents and DTD, are converted before validation.

One of the obstacles that prior works had to cope with was to verify well-formedness of XML documents, meaning that every opening tag matches its same-level closing tag. Due to the work [MMN10], such a verification can be performed now with a constant-pass randomized streaming algorithm with sublinear memory space and no auxiliary streams. In one pass the memory space is $O(\sqrt{N \log N})$, and collapses to $O(\log^2 N)$ with an additional pass in reverse direction.

3.3.2 Our Contribution

The starting point of this work is the fact that checking DTD-validity is hard without auxiliary streams. There are DTDs that admit ternary XML documents, and any p-pass bidirectional randomized streaming algorithm which validates those documents against those DTDs requires $\Omega(N/p)$ space. This lower bound comes from encoding a well-known communication complexity problem, Set-Disjointness, as an XML validity problem. This lower bound should be well-known, however we are not aware of a complete proof in the literature. In [GKS07], a similar approach using ternary trees with a reduction from Set-Disjointness is used for proving lower bounds for queries. For the sake of completeness we provide a proof in Section 6.2.1 (**Theorem 13**).

On the other hand, it is possible to validate XML documents in one pass and space $O(d)$, where d is the depth of the XML document (**Theorem 18**). This algorithm is straightforward and should as well be known. Furthermore, we mention that the previously discussed lower bound of $\Omega(N/p)$ can be modified to obtain a lower bound of $\Omega(d/p)$. Therefore, space $O(d)$ is best possible for one-pass algorithms. For completeness we discuss this algorithm in Section 6.4.1.

For the case of XML documents encoding binary trees, we present in Section 6.3 three deterministic streaming algorithms for checking validity with sublinear space. As a consequence, the presence of nodes of degree at least 3 is indeed a necessary condition for the linear space lower bound for general documents. We first present two one-pass algorithms with space $O(\sqrt{N \log N})$ (**Theorem 15** and **Theorem 16**). The first algorithm, Algorithm 14, processes the input XML document in blocks and is easy to analyze, however, it is not optimal in terms of processing time per letter. The second algorithm, Algorithm 15, uses a stack and has constant processing time per letter. We conjecture that there is a $\Omega(N^{1/2})$ lower bound for one-pass algorithms. With a second pass in reverse direction the memory collapses to $O(\log^2 N)$ (**Theorem 17**). These three algorithms make use of the simple but fundamental fact that in one pass over an XML document each node is seen twice by means of its opening and closing tag. Hence, it is not necessary to remember all opening tags in the stream since there is a second chance to get the same information from their closing tags. Our algorithms exploit this observation. We summarize our streaming algorithms for validating documents that encode binary trees in Figure 3.2.

Passes	Space	Time	Remark
1	$O(\sqrt{N \log N})$	$\Omega(\sqrt{N \log N})$	Block Algorithm (Theorem 15), simple analysis
1	$O(\sqrt{N \log N})$	$O(1)$	Stack Algorithm (Theorem 16)
2	$O(\log^2 N)$	$O(\log N)$	Bidirectional Algorithm (Theorem 17)

Figure 3.2: Overview about our streaming algorithms for checking DTD-validity of XML documents that encode binary trees. Time refers to the worst-case processing time between two consecutive read operations from the stream. We did not fully analyze the processing time of the block algorithm, however, since it processes the input in blocks of size $\Theta(\sqrt{N \log N})$, the processing time is $\Omega(\sqrt{N \log N})$.

Then, in Section 6.4 we present our main result. **Corollary 2** states that the validation of any XML document against any DTD can be checked in the streaming model with external memory with poly-logarithmic space, a constant number of auxiliary streams, and $O(\log N)$ passes over these streams. Validity of a node depends on its children, hence it is crucial to have easy access to the sequence of children of any node. We establish this by computing the FCNS encoding, which is an encoding of the XML document as a 2-ranked tree. In this encoding, the sequence of closing tags of the children of a node are consecutive. The computation of this encoding is the hard part of the validation process, and the resource requirements of our validation algorithm stem from this operation (**Theorem 19**). Since the FCNS encoding can be seen as a reordering of the tags of the original document, our strategy is to regard this problem as a sorting problem with a particular comparison function. Merge sort can be implemented as a streaming algorithm with auxiliary streams. We use a version that is customized with an adapted merge function. The same idea can be used for FCNS decoding with similar complexity (**Theorem 23**). Then, based on the FCNS encoding, verification can be completed either in one pass and $O(\sqrt{N \log N})$ space (**Theorem 21**), or in two bidirectional passes and $O(\log^2 N)$ space (**Theorem 20**). Figure 3.3 illustrates how our streaming algorithm of Corollary 2 for the validation of general XML documents is obtained.

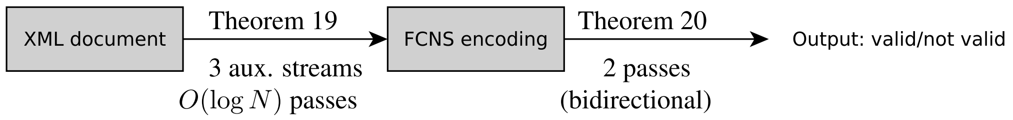


Figure 3.3: Schema of our Streaming Algorithm of Corollary 2 for checking DTD-validity of arbitrary XML documents. First, the First-Child-Next-Sibling encoding of the input XML document is computed with 3 auxiliary streams and $O(\log N)$ space. Then, validity of the document is checked with 2 bidirectional passes and $O(\log^2 N)$ space.

Concerning the computation of the FCNS encoding and the FCNS decoding, we show linear space lower bounds for algorithms that perform one pass over the input and one pass over the output. For decoding, we present an algorithm that uses $O(\sqrt{N \log N})$ space (**Theorem 22**) and performs one pass over the input, but two passes over the output. Furthermore, we show that with 3 auxiliary streams and $O(\log N)$ passes, both encoding and decoding can be done with $O(\log N)$ space. For encoding, we conjecture that if no access to auxiliary streams is granted the memory space remains $\Omega(N)$ after any constant number of passes. This would show that decoding is easier than encoding.

LB/UB	Passes	Space	Remark
FCNS encoding:			
Lower Bound	1 pass on input, 1 pass on output	$\Omega(N)$	(Fact 6)
Upper Bound	$O(\log N)$ passes on 3 auxiliary streams	$O(\log N)$	(Corollary 1)
FCNS decoding:			
Lower Bound	1 pass on input, 1 pass on output	$\Omega(N)$	(Theorem 24)
Upper Bound	1 pass on input, 2 passes on output	$O(\sqrt{N} \log N)$	(Theorem 22)
Upper Bound	$O(\log N)$ passes on 3 auxiliary streams	$O(\log N)$	(Theorem 23)

Figure 3.4: Overview about our results on computing the FCNS encoding and the FCNS decoding. For encoding, an XML document is on the input stream and the goal is to output the FCNS encoding of this document on an output stream. For decoding, the FCNS encoding of an XML document is on the input stream and the goal is to output the original document on an output stream.

3.4 Budget Error-Correcting under Earth-Mover-Distance

Consider the following one-way communication setting. Alice holds an object x and Bob holds an object y. Alice want to send Bob a message M of size $o(|x|)$ such that if y is close to x then Bob can learn x and if y is far apart from x then Bob can report that x and y are far apart. The *document exchange* problem [CPSV00, IMS05, Jow12] fits into this setting. In this problem Alice has a string x, Bob has a string y, and Alice wants to send a short message to Bob so that either Bob can learn x or he can output that the *Edit Distance* between x and y is at least k. The Edit Distance between two strings is the minimal number of character insertions, deletions or substitutions that are required to convert x into y.

Suppose now that Bob wants to adjust his object towards Alice's object in case their inputs x and y are far apart. We call this setting the *object synchronization* setting. Efficient protocols for this setting exist if for instance the Hamming Distance or the Edit Distance is used as a similarity measure of the objects. In the following, we are going to study this setting under the Earth-Mover-Distance (EMD).

3.4.1 Earth-Mover-Distance

Definition 5 (Earth-Mover-Distance)**.** *Let $\Delta > 0$ denote the side length of a grid. Let $x = \{x_1, \dots, x_n\}, y = \{y_1, \dots, y_n\} \subseteq [\Delta]^d$ be sets of n points on the d-dimensional grid $[\Delta]^d$. Then the* Earth-Mover-Distance *(EMD) between x and y is defined as the minimum weight perfect matching in the weighted complete bipartite graph with bipartitions x and y where the weight of an edge between x_i and y_j is defined as $\|x_i - y_j\|_2$. In other words:*

$$\mathrm{EMD}(x, y) = \min_{\pi:[n]\to[n]} \textstyle\sum_{1\le i\le n} \left\|x_i - y_{\pi(i)}\right\|_2 ,$$

where π is a permutation.

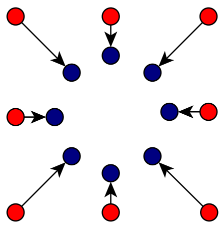


Figure 3.5: Illustration of the Earth-Mover-Distance. Here we see a set R of red points and a set B of blue points. The Earth-Mover-Distance $\mathrm{EMD}(R, B)$ is the weight of a minimum weight perfect matching between R and B. This matching is illustrated by the arrows and the value of $\mathrm{EMD}(R, B)$ corresponds to the sum of the lengths of these arrows.

EMD is a popular distance function for images in computer vision (see, e.g., [PBRT99, RTG00, HRT02, GD04, CB07]). EMD as a distance measure is more sensitive to noise than for example the Hamming Distance or the Edit Distance. If little noise is introduced into an image (for instance during data transmission), then this does not necessarily mean that the EMD between the original image and the noisy image is small. Noise in an image may lead to a slight shift or rotation of it. Often, this does not affect the usefulness of the image, however, the EMD is high between the original image and the one after the shift/rotation. For this reason, a document exchange setting for EMD where Bob only reports that x and y are far apart is not useful in practice since a small visual difference may lead already to a high EMD. This motivates the study of the object synchronization setting under Earth-Mover-Distance.

3.4.2 Measuring the Quality of an Adjustment

To measure the quality of Bob's adjustment, we use an idea from the compressive-sensing and sparse-recovery literature (see [GI10] for a survey). In sparse recovery, Alice has an n-dimensional vector x. She sends Bob a linear sketch Ax where A is an $m \times n$ $(m \ll n)$ matrix, and then Bob reconstructs a vector x^* such that

$$\|x - x^*\|_p \leq C \min_{k\text{-sparse } \tilde{x}} \|x - \tilde{x}\|_q ,$$

where p, q are norm parameters, $C > 0$ is the approximation factor, and we say a vector x is k-sparse if it has at most k non-zero coordinates. The goal is to minimize m, that is, the number of rows of the sketch matrix A since this number corresponds to a message size if the problem is seen as a one-way communication problem. Inspired by this setting, we define the EMD k-Budget Error-Correcting Problem.

Definition 6 (The EMD k-Budget Error-Correcting Problem). *Let Δ be an integer. Alice holds a set of n points $x = \{x_1, \ldots, x_n\} \subseteq [\Delta]^d$ on the d-dimensional grid $[\Delta]^d$, and Bob holds another set of n points $y = \{y_1, \ldots, y_n\} \subseteq [\Delta]^d$. Alice sends a message M to Bob, and Bob then relocates his points to $y^* = \{y_1^*, \ldots, y_n^*\} \subseteq [\Delta]^d$ such that*

$$\mathrm{EMD}(x, y^*) \leq C \min_{\tilde{y} \in \mathcal{N}_k(y)} \mathrm{EMD}(x, \tilde{y}),$$

where $\mathcal{N}_k(y)$ denotes all point sets of cardinality n that can be obtained by relocating k points in y, and C is a fixed approximation factor. The goal is to minimize the message size $|M|$.

A sparse-recovery scheme could potentially be used for our error-correcting setting: Alice computes Ax and sends the result to Bob. Bob then computes Ay and $(Ax - Ay) = A(x - y)$. Then, Bob tries to recover the heaviest coordinates of $(x - y)$ from $A(x - y)$. Indyk and Price [IP11] studied the sparse recovery setting with $p = q = \text{EMD}$, however, their algorithm only works for vectors with positive coordinates, and hence their scheme can not be used to recover the heavy coordinates of $x - y$.

3.4.3 Our Contribution

In a joint work with Qin Zhang and Wei Yu [KYZ13], we show the following results for EMD k-Budget Error-Correcting.

- We give an $O(d)$-approximation protocol with $\tilde{O}(k \log \Delta \log(n\Delta^d))$ bits[1] of communication **(Theorem 25)**. The algorithm is randomized and has success probability $2/3$.
- We complement our upper bound with a lower bound of $\Omega(k \log \Delta \log(\Delta^d/k)/\log d)$ bits of communication **(Theorem 26)**. The lower bound holds for randomized algorithms that compute an $O(d)$-approximation.

Note that for typical settings where $d = O(1), n = \Delta^{O(1)}$, the upper bound almost matches the lower bound.

[1]We use $\tilde{O}(f)$ to denote a function of the form $O(f \log f)$.

Chapter 4

Computing Matchings in the Streaming Model

In this section, we present our results on the computation of matchings in the semi-streaming model. We start our presentation with notations and definitions in Section 4.1. Then, we present in Section 4.2 a 3-pass algorithm for bipartite graphs on adversarial order that serves as a starting point for all our algorithms. This 3-pass algorithm computes a maximal matching in one-pass and searches for 3-augmenting paths in the second and the third pass. Then, in Section 4.3 we show how this 3-pass algorithm can be simulated in one pass if the input is in random order. This algorithm requires a new lemma about the convergence of the Greedy matching algorithm that we also discuss in this section. We obtain a deterministic one-pass algorithm on random order for bipartite graphs, and we also show that this idea generalizes to non-bipartite graphs. Furthermore, we show that in adversarial order, the 3-pass algorithm can be simulated with 2 passes. We present in Section 4.4 a randomized 2-pass algorithm for bipartite graphs. Finally, we present a deterministic 2-pass algorithm for bipartite and non-bipartite graphs in Section 4.5.

4.1 Preliminaries

Let $G = (V, E)$ be a graph with vertex set V and edge set E. If G is bipartite with bipartitions A and B then we write $G = (A, B, E)$ and we denote $V = A \cup B$. Let $n = |V|$ and $m = |E|$. For an edge $e \in E$ with end points $u, v \in V$, we denote e by uv. For a subset of edges $S \subseteq E$ and a vertex $v \in V$, we write $\deg_S(v)$ for the degree of v in S, meaning the number of edges in S that have v as one of its endpoints.

We define now matchings, maximum matchings and maximal matchings.

Definition 7 (Matching)**.** A matching *in a graph* $G = (V, E)$ *is a subset of edges* $M \subseteq E$ *such that* $\forall v \in V : \deg_M(v) \leq 1$. A maximum matching M^* *is a matching such that for any other matching* $M' : |M^*| \geq |M'|$. A maximal matching M *is a matching that is inclusion-wise maximal, a.e.* $\forall e \in E \setminus M : M \cup \{e\}$ *is not a matching.*

The MAXIMUM BIPARTITE MATCHING problem consists of computing a maximum matching in a bipartite graph and we abbreviate it by MBM.

The MAXIMUM MATCHING problem consists of computing a maximum matching in a general graph and we abbreviate it by MM.

For a subset of edges $F \subseteq E$, we denote by $\text{opt}(F)$ a maximum matching in the graph G restricted to edges F. We may write $\text{opt}(G)$ for $\text{opt}(E)$, and M^* for $\text{opt}(G)$. For a set of vertices S and a set of edges F, let $S(F)$ be the subset of vertices of S covered by F. Furthermore, we use the abbreviation $\overline{S(F)} := S \setminus S(F)$. For $S \subseteq V$, we write $\text{opt}(S)$ for $\text{opt}(G|_S)$, that is a maximum matching in the subgraph of G induced by vertices S. In case of bipartite graphs, for $S_A \subseteq A$ and $S_B \subseteq B$ we write $\text{opt}(S_A, S_B)$ for $\text{opt}(G|_{S_A \cup S_B})$. Moreover, for two sets S_1, S_2 we denote by $S_1 \oplus S_2$ the symmetric difference $(S_1 \setminus S_2) \cup (S_2 \setminus S_1)$ of the two sets.

A standard technique to increase the size of matchings is to search for *augmenting paths*. We define augmenting paths as follows.

Definition 8 (Augmenting Path). *Let $p \geq 3$ be an odd integer. Then a length p augmenting path with respect to a matching M in a graph $G = (V, E)$ is a path $P = (v_1, \dots, v_{p+1})$ such that $v_1, v_{p+1} \notin V(M)$ and for $i \leq 1/2(p-1) : v_{2i}v_{2i+1} \in M$, and $v_{2i-1}v_{2i} \notin M$.*

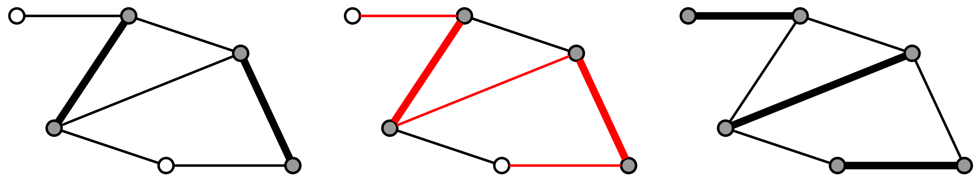


Figure 4.1: Left: A graph with a matching (in bold) of size 2. The solid vertices are matched vertices, the white vertices are free vertices. Middle: a length-5 augmenting path. Right: After augmentation, the size of the matching increased by 1 and all vertices are matched.

An augmenting path of length p ($p \geq 3, p$ odd) with respect to a matching M in a graph $G = (V, E)$ is a path that starts and ends at nodes that are not matched in M. We call such nodes *free* nodes. All internal nodes of the path are matched in M, and we call these node *matched* nodes. The path alternates between edges outside M and edges of M. Removing from M the edges of the augmenting path that are also in M and inserting into M the edges outside M increases the size of M by 1. This is illustrated in Figure 4.1.

The input graph G is given as a graph stream, i.e. as a sequence of edges arriving one by one in some order. Let $\Pi(G)$ be the set of all edge sequences of G. An input stream for our streaming algorithms is then an edge sequence $\pi \in \Pi(G)$. We write $\pi[i]$ for the i-th edge of π, and $\pi[i, j]$ for the subsequence $\pi[i]\pi[i+1] \dots \pi[j]$. In this notation, a round bracket excludes the smallest or respectively largest element: $\pi(i, j] = \pi[i+1, j]$, and $\pi[i, j) = \pi[i, j-1]$. If i, j are real, $\pi[i, j] := \pi[\lfloor i \rfloor, \lfloor j \rfloor]$, and $\pi[i] := \pi[\lfloor i \rfloor]$. Given a subset $S \subseteq V$, $\pi|_S$ is the largest subsequence of π such that all edges in $\pi|_S$ are among vertices in S.

We say that an algorithm $\mathbf{A}$ computes a *c-approximation to the maximum matching problem* if $\mathbf{A}$ outputs a matching M such that $|M| \geq c \cdot |\text{opt}(G)|$. We consider two potential sources of randomness: from the algorithm and from the arrival order. Nevertheless, we will always consider worst case against the graph. For each situation, we relax the notion of c-approximation so that the expected approximation ratio is c, that is $\mathbb{E}\,|M| \geq c \cdot |\mathrm{opt}(G)|$ where the expectation can be taken either over the internal random coins of the algorithm, or over all possible arrival orders.

4.2 Three-pass Semi-Streaming Algorithm for Bipartite Graphs on Adversarial Order

Let us firstly discuss the greedy matching algorithm. Formally, the *greedy matching algorithm* Greedy *on graph stream* π is defined as follows: Starting with an empty matching M, upon arrival of an edge $\pi[i]$, Greedy inserts $\pi[i]$ into M if $\pi[i]$ does not intersect any edges in M, see Algorithm 2. Denote by $\mathrm{Greedy}(\pi)$ the matching M after the stream π has been fully processed. By maximality, $|\mathrm{Greedy}(\pi)| \geq \frac{1}{2}|\mathrm{opt}(G)|$. Greedy can be seen as a semi-streaming algorithm for MBM (and even also for MM) with approximation ratio $\frac{1}{2}$ and update time $O(1)$.

Algorithm 2 The Greedy Matching Algorithm

```
1: $M \leftarrow \varnothing$
2: while edge stream not empty do
3:     $e = v_1 v_2 \leftarrow$ next edge in stream
4:     if $\{v_1, v_2\} \cap V(M) = \varnothing$ then $M \leftarrow M \cup \{e\}$ end if
5: end while
6: return $M$
```

To improve on Greedy with three passes, a simple strategy is to, firstly, compute a maximal matching M_G in one pass, and then use the second and the third pass to search for 3-augmenting paths to augment M_G.

Suppose that M_G is close to a $1/2$-approximation.Then almost all edges of M_G are 3-*augmentable*. We say that an edge $e \in M_G$ is 3-augmentable if the removal of e from M allows the insertion of two edges $f, g \in M^* \setminus M$ into M. More formally, the following lemma holds.

Lemma 1. *Let $\epsilon \geq 0$. Let M be a maximal matching of G st. $|M| \leq (\frac{1}{2} + \epsilon)|M^*|$. Then M contains at least $(\frac{1}{2} - 3\epsilon)|M^*|$ 3-augmentable edges.*

Proof. The proof is folklore. Let k_i denote the number of paths of length i in $M \oplus M^*$. Since M^* is maximum, it has no augmenting path, so all odd length paths are augmenting paths of M. Since M is maximal, there are no augmenting paths of length 1, so $k_1 = 0$. Every even length path and every cycle has an equal number of edges from M and from M^*. A path of length $2i + 1$ has i edges from M and $i + 1$ edges from M^*.

$$|M^*| - |M| = \sum_{i \geq 1} k_{2i+1} \leq k_3 + \sum_{i \geq 2} \frac{1}{2} i k_{2i+1} = \frac{1}{2} k_3 + \frac{1}{2} \sum_{i \geq 1} i k_{2i+1} \leq \frac{1}{2} k_3 + \frac{1}{2}|M|.$$

Thus, using our assumption on $|M|$, $k_3 \geq 2|M^*| - 3|M| \geq 2|M^*| - (\frac{3}{2} + 3\epsilon)|M^*|$, implying the Lemma. $\square$

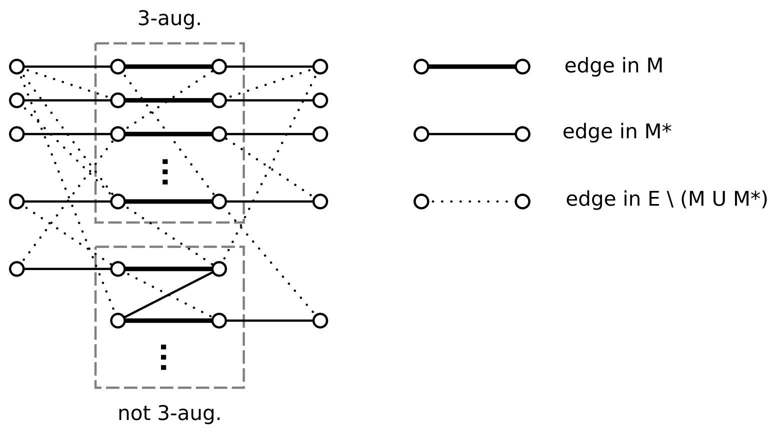


Figure 4.2: Illustration of Lemma 1. If $|M| \leq (1/2 + \epsilon)|M^*|$, then at least $(\frac{1}{2} - 3\epsilon)|M^*|$ edges of M are 3-augmentable.

We search for 3-augmenting paths as follows. Firstly, we compute a maximal matching M_L via the Greedy algorithm between the A vertices that are matched in M_G and the free B vertices. Under the assumption that M_G is close to a $1/2$ approximation, most of the edges of M_G are 3-augmentable. Hence, there exists a large matching, and since M_L is a maximal matching, M_L will be at least of size $1/2$ times the number of 3-augmentable edges. Edges from M_L will serve as the start of length 3-augmenting paths. Then in the third pass, we compute another maximal matching M_R in order to complete 3-augmenting paths with the edges of M_G and M_L. This algorithm is stated in Algorithm 3, and illustrated in Figure 4.3. This idea was already used in [FKM$^+$04]. The authors present there an $O((\log \frac{1}{\epsilon})/\epsilon)$-pass semi-streaming algorithm that computes a $2/3 - \epsilon$ approximation to the maximum bipartite matching problem. An analysis for Algorithm 3 can be derived from their work.

Algorithm 3 Three-pass Bipartite Matching Algorithm

```
Require: The input stream $\pi$ is an edge stream of a bipartite graph $G = (A, B, E)$
1: $M_G, M_L, M_R \leftarrow \varnothing$
2: 1st pass: $M_G \leftarrow$ Greedy($\pi$)
3: $G_L \leftarrow$ complete graph between $A(M_G)$ and $B \setminus B(M_G)$
4: 2nd pass: $M_L \leftarrow$ Greedy($\pi \cap G_L$)
5: $G_R \leftarrow$ complete graph between $\{b \in B(M_G) : A(M_G(b)) \in A(M_L)\}$ and $A \setminus A(M_G)$
6: 3rd pass: $M_R \leftarrow$ Greedy($\pi \cap G_R$)
7: return maximum matching in $M_G \cup M_L \cup M_R$
```

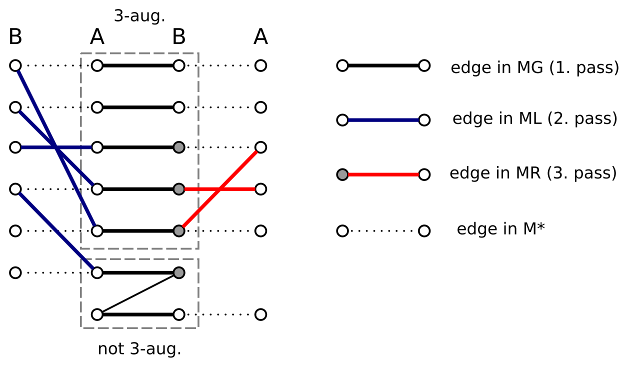


Figure 4.3: Illustration of Algorithm 3. The graph contains a perfect matching of size 13. In the first pass, M_G is computed and has size 7. This is close to a $1/2$ approximation and by Lemma 1, M has many (here 5) 3-augmentable edges. There exists hence a matching of size at least 5 between $A(M_G)$ and the free B vertices. Since M_L is maximal, it is of size at least $5/2$ (here 4). Then, a maximal matching is computed between the solid vertices, which are the B vertices of edges of M_G that can be potentially be completed to a 3-augmenting path, and the free A vertices. In this example, two length 3 augmenting paths were found.

4.3 One-pass Matching Algorithm on Random Order

We discuss now, how the 3-pass algorithm from the previous section, Section 4.2, can be simulated with a single pass if the input is in random order. We firstly present in Subsection 4.3.1 a Lemma about the convergence of the Greedy matching algorithm if the input is in random order. This lemma is the main ingredient for our one-pass algorithms. Then, in Subsection 4.3.2 we discuss our one-pass algorithm on random order for bipartite graphs, and we extend it to general graphs in Subsection 4.3.3.

4.3.1 A Lemma on the Convergence of the Greedy Algorithm

The Greedy matching algorithm (Algorithm 2) plays a central role in the design of our one-pass matching algorithm for streams that are in uniform random order. It is very easy to see that the Greedy algorithm is not better than a $1/2$-approximation when the stream is in adversarial order: Consider a line of length 3 and suppose that the edge in the middle arrives first. When the input stream is in uniform random order, Dyer and Frieze proved in [DF91] for general graphs that Greedy is not better than a $1/2$-approximation. Their hard instance can be adapted to bipartite graphs as shown in Figure 4.4.

The hard instance is a graph on $4n$ vertices. Each vertex of a complete bipartite graph $K_{n,n}$ on $2n$ vertices is duplicated and an edge connecting the vertex with its duplicate is inserted (this is the set E' in Figure 4.4). Then E' is also a perfect matching in this graph. The graph is constructed such that all edges of the $K_{n,n}$ are *bad* edges for the matching: Taking such an edge into the matching blocks two edges of E'. Since the $K_{n,n}$ has $\Theta(n^2)$ edges and the set E' consists only of n edges, Greedy picks almost only edges from the $K_{n,n}$ in the random arrival

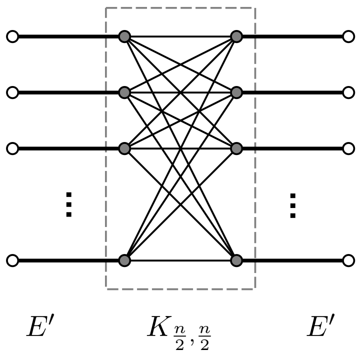


Figure 4.4: Hard instance for the Greedy matching algorithm in the random arrival order model. Each vertex of a complete bipartite graph $K_{n,n}$ on $2n$ vertices is duplicated and an edge connecting the vertex with its duplicate is inserted (the set E'). This graph has a perfect matching consisting of the edges E'. Since the $K_{n,n}$ has $\Theta(n^2)$ edges while the set E' consists only of $2n$ edges, in the random arrival order Greedy picks almost only edges from the $K_{n,n}$. Each of these edges, however, *blocks* two optimal edges from E'.

order. For a precise analysis, see the work of Dyer and Frieze [DF91].

This example illustrates that, if on random order Greedy produces a matching of size close to $1/2$ times the size of a maximum matching M^*, then the graph contains many bad edges blocking two edges of M^*. Note that since the edges of M^* arrive in uniform random order, in average an α fraction of these edges arrives in the first α fraction of the stream, and as we already pointed out, most of these edges are not taken into the Greedy matching. This implies that if Greedy produces a matching close to a $1/2$ approximation on random order, then Greedy must take many bad edges *early on*.

We make this intuition rigorous in Lemma 2. Lemma 2 allows us to conclude that if Greedy on the entire graph is no better than a $1/2 + \epsilon$ approximation, then after seeing a mere one third of the edges of the graph, Greedy is already a $1/2 - \epsilon$ approximation, so it is already close to maximal.

Lemma 2. *Let* $G = (V, E)$ *be a graph with* $m = |E|$. *Then, if* $\mathbb{E}_{\pi} |\text{Greedy}(\pi)| \leq (\frac{1}{2} + \epsilon)|M^*|$ *for some* $0 < \epsilon < 1/2$*, then for any* $0 < \alpha \leq 1$,

$$\mathop{\mathbb{E}}_{\pi} |\text{Greedy}(\pi[1, \alpha m])| \geq |M^*|(\frac{1}{2} - (\frac{1}{\alpha} - 2)\epsilon).$$

Proof. Let $M_0 = \text{Greedy}(\pi[1, \alpha m])$. Rather than directly analyzing the number of edges $|M_0|$, we analyze the number of vertices matched by M_0, which is equivalent since $|V(M_0)| = 2(|M_0|)$.

Fix an edge $e = ab$ of M^*. Either $e \in M_0$, or at least one of a, b is matched by M_0, or neither a nor b are matched. Summing over all $e \in M^*$ gives

$$|V(M_0)| \geq 2|M^* \cap M_0| + |M^* \setminus M_0| - \sum_{e=ab \in M^*} \chi[a \text{ and } b \notin V(M_0)],$$

where $\chi[X] = 1$ if the event X happens, otherwise $\chi[X] = 0$. We show in Lemma 3 that

$$\Pr[a \text{ and } b \notin V(M_0)] \leq (\frac{1}{\alpha} - 1)\Pr[e \in M_0]. \tag{4.1}$$

Taking expectations and using Inequality 4.1,

$$\begin{aligned} \mathbb{E}_{\pi}(|V(M_0)|) &\geq 2\mathbb{E}_{\pi}|M^* \cap M_0| + \mathbb{E}_{\pi}|M^* \setminus M_0| - (\frac{1}{\alpha} - 1)\mathbb{E}_{\pi}|M^* \cap M_0| \\ &= |M^*| - (\frac{1}{\alpha} - 2)\mathbb{E}_{\pi}|M^* \cap M_0|. \end{aligned}$$

We will show in Lemma 4 that for a maximum matching M^* and any maximal matching M_G, we have $|M_G \cap M^*| \leq 2(|M_G| - 1/2|M^*|)$. Using this, and since M_0 is just a subset of the edges of M_G, we obtain by linearity of expectation

$$\mathbb{E}_{\pi}|M^* \cap M_0| \leq \mathbb{E}_{\pi}|M^* \cap M_G| \leq 2(\mathbb{E}_{\pi}|M_G| - \frac{1}{2}|M^*|) \leq 2\epsilon|M^*|.$$

Combining gives the Lemma. □

We now prove Lemma 3 that was used in the proof of Lemma 2.

Lemma 3. *Let $G = (V, E)$ be a graph with $m = |E|$. Furthermore, let $\mathbb{E}_\pi|\text{Greedy}(\pi)| \leq (\frac{1}{2} + \epsilon)|M^*|$ for some $0 < \epsilon < 1/2$. Let $M_0 = \text{Greedy}(\pi[1, \alpha m])$ for some $0 < \alpha \leq 1/2$. Then:*

$$\forall e = ab \in E : \Pr[a \textit{ and } b \notin V(M_0)] \leq (\frac{1}{\alpha} - 1)\Pr[e \in M_0].$$

Proof. Observe: $\Pr[a \text{ and } b \notin V(M_0)] + \Pr[e \in M_0] = \Pr[a \text{ and } b \notin V(M_0 \setminus \{e\})]$, because the two events on the left hand side are disjoint and their union is the event on the right hand side.

Consider the following probabilistic argument. Take the execution for a particular ordering π. Assume that a and $b \notin V(M_0 \setminus \{e\})$ and let t be the arrival time of e. If we modify the ordering by changing the arrival time of e to some time $t' \leq t$, then we still have a and $b \notin V(M_0 \setminus \{e\})$. More formally, we define a map f from the uniform distribution on all orderings to the uniform distribution on all orderings such that $e \in \pi[1, \alpha m]$: if $e \in \pi[1, \alpha m]$ then $f(\pi) = \pi$ and otherwise $f(\pi)$ is the permutation obtained from π by removing e and re-inserting it at a position picked uniformly at random in $[1, \alpha m]$. Thus,

$$\Pr[a \text{ and } b \notin V(M_0 \setminus \{e\})] \leq \Pr[a \text{ and } b \notin V(M_0 \setminus \{e\})|e \in \pi[1, \alpha m]].$$

Now, the right-hand side equals $\Pr[e \in M_0 | e \in \pi[1, \alpha m]]$, which simplifies into $\Pr[e \in M_0]/\Pr[e \in \pi[1, \alpha m]]$ since e can only be in M_0 if it is one of the first αm arrivals. Then we conclude the Lemma by the random order assumption $\Pr[e \in \pi[1, \alpha m]] = \alpha$. □

Lemma 4 shows that an optimal matching and a maximal matching that is far from this optimal matching in size do not have many edges in common.

Lemma 4. *Let M be a maximal matching of a graph G. Then*

$$|M \cap M^*| \leq 2(|M| - \frac{1}{2}|M^*|).$$

Proof. This is a piece of elementary combinatorics. Since M is a maximal matching, for every edge e of $M^* \setminus M$, at least one of the two endpoints of e is matched in $M \setminus M^*$, and so $|M \setminus M^*| \geq (1/2)|M^* \setminus M|$. We have $|M^* \setminus M| = |M^*| - |M^* \cap M|$. Combining gives

$$|M \cap M^*| = |M| - |M \setminus M^*| \leq |M| - \frac{1}{2}|M^* \setminus M| = |M| - \frac{1}{2}(|M^*| - |M^* \cap M|)$$

which implies the Lemma. □

4.3.2 Bipartite Graphs

Algorithm. We simulate the 3-pass algorithm, Algorithm 3, in one pass as follows. We split the input graph stream $\pi \in \Pi(G)$ into three phases $\pi[1, \alpha m]$, $\pi(\alpha m, \beta m]$, and $\pi(\beta m, m]$ and in each phase, we build a matching. M_0 is built during the first phase and corresponds to matching M_G of our 3-pass algorithm. M_1 is built in the second phase and M_2 in the third, and they correspond to M_L and M_R of our 3-pass algorithm, respectively. Assume that Greedy performs badly on the input graph G. Lemma 1 tells us that almost all of the edges of M_0 are 3-augmentable. To find 3-augmenting paths, in the next part of the stream we run Greedy to compute a matching M_1 between $B(M_0)$ and $\overline{A(M_0)}$. The edges in M_1 serve as one of the edges of 3-augmenting paths (from the B-side of M_0). In Lemma 5, we show that we find a constant fraction of those. In the last part of the stream, again by the help of Greedy, we compute a matching M_2 that completes the 3-augmenting paths. Lemma 8 shows that by this strategy we find many 3-augmenting paths. Then, either a simple Greedy matching performs well on G, or else we can find many 3-augmenting paths and use them to improve M_0: see the main theorem, Theorem 1 whose proof is deferred to the end of this section. An illustration is provided in Figure 4.5.

Algorithm 4 One-pass Bipartite Matching on Random Order

```
1: $\alpha \leftarrow 0.4312, \beta \leftarrow 0.7595$
2: $M_G \leftarrow$ Greedy($\pi$)
3: $M_0 \leftarrow$ Greedy($\pi[1, \alpha m]$), matching obtained by Greedy on the first $\lfloor \alpha m \rfloor$ edges
4: $F_1 \leftarrow$ complete bipartite graph between $B(M_0)$ and $\overline{A(M_0)}$
5: $M_1 \leftarrow$ Greedy($F_1 \cap \pi(\alpha m, \beta m]$), matching obtained by Greedy on edges $\lfloor \alpha m \rfloor + 1$
   through $\beta m$ that intersect $F_1$
6: $A' \leftarrow \{a \in A \mid \exists b \in B(M_1) : ab \in M_0\}$
7: $F_2 \leftarrow$ complete bipartite graph between $A'$ and $\overline{B(M_0)}$
8: $M_2 \leftarrow$ Greedy($F_2 \cap \pi(\beta m, m]$), matching obtained by Greedy on edges $\lfloor \beta m \rfloor + 1$ through
   $m$ that intersect $F_2$
9: $M \leftarrow$ matching obtained from $M_0$ augmented by $M_1 \cup M_2$
10: return larger of the two matchings $M_G$ and $M$
```

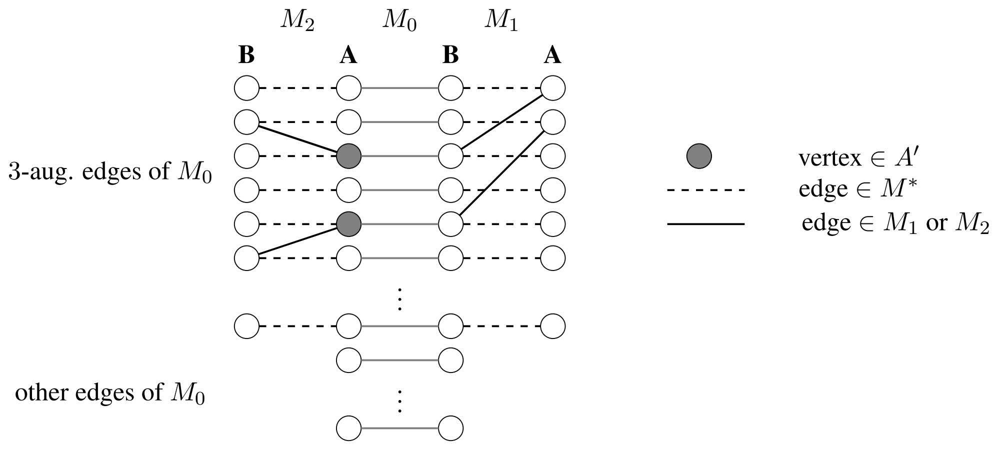


Figure 4.5: Illustration of Algorithm 4. Note that every edge of M_2 completes a 3-augmenting path consisting of one edge of M_1 (on the right hand side of the picture) followed by one edge of M_0 (center) followed by one edge of M_2 (on the left hand side of the picture).

Observe that our algorithm only uses memory space $O(n \log n)$. Indeed, the subsets F_1 and F_2 can be compactly represented by two n-bit arrays, and checking if an edge of π belongs to one of them can be done within time $O(1)$ from that compact representation.

Analysis. We use the notations of Algorithm 4. Consider α and β as variables with $0 \le \alpha \le \frac{1}{2} < \beta < 1$.

Lemma 5. *Assume that $\mathbb{E}_\pi |M_G| \le (\frac{1}{2} + \epsilon)|M^*|$. Then the expected size of a maximum matching between the vertices of A left unmatched by M_0 and the vertices of B matched by M_0 can be bounded below as follows:*

$$\mathop{\mathbb{E}}_{\pi} |\mathrm{opt}(\overline{A(M_0)}, B(M_0))| \ge |M^*|(\frac{1}{2} - (\frac{1}{\alpha} + 2)\epsilon).$$

Proof. The size of a maximum matching between $\overline{A(M_0)}$ and $B(M_0)$ is at least the number of augmenting paths of length 3 in $M_0 \oplus M^*$. By Lemma 1, in expectation, the number of augmenting paths of length 3 in $M_G \oplus M^*$ is at least $(\frac{1}{2} - 3\epsilon)|M^*|$. All of those are augmenting paths of length 3 in $M_0 \oplus M^*$, except for at most $|M_G| - |M_0|$. Hence, in expectation, M_0 contains $(\frac{1}{2} - 3\epsilon)|M^*| - (\mathbb{E}_\pi |M_G| - \mathbb{E}_\pi |M_0|)$ 3-augmentable edges. Lemma 2 applied to M_0 concludes the proof. □

Lemma 6. $\mathbb{E}_\pi |M_1| \ge \frac{1}{2}(\beta - \alpha)(\mathbb{E}_\pi |\mathrm{opt}(\overline{A(M_0)}, B(M_0))| - \frac{1}{1-\alpha})$.

Proof. Since Greedy computes a maximal matching which is at least half the size of a maximum matching, $\mathbb{E}_\pi |M_1| \ge \frac{1}{2} \mathbb{E}_\pi |\mathrm{opt}(\overline{A(M_0)}, B(M_0)) \cap \pi(\alpha m, \beta m]|$.

By independence between M_0 and the ordering within $(\alpha m, m]$, we see that even if we condition on M_0, we still have that $\pi(\alpha m, \beta m]$ is a random uniform subset of $\pi(\alpha m, m]$. Thus:

$$\mathop{\mathbb{E}}_{\pi} |\mathrm{opt}(\overline{A(M_0)}, B(M_0)) \cap \pi(\alpha m, \beta m]| = \frac{\beta-\alpha}{1-\alpha} \mathop{\mathbb{E}}_{\pi} |\mathrm{opt}(\overline{A(M_0)}, B(M_0)) \cap \pi(\alpha m, m]|.$$

We use a probabilistic argument similar to but slightly more complicated than the proof of Lemma 3. We define a map f from the uniform distribution on all orderings to the uniform distribution on all orderings such that $e \in \pi(\alpha m, m]$: if $e \in \pi(\alpha m, m]$ then $f(\pi) = \pi$ and otherwise $f(\pi)$ is the permutation obtained from π by removing e and re-inserting it at a position picked uniformly at random in $(\alpha m, m]$; in the latter case, if this causes an edge $f = a'b'$, previously arriving at time $\lfloor \alpha m \rfloor + 1$, to now arrive at time $\lfloor \alpha m \rfloor$ and to be added to M_0, we define $M_0' = M_0 \setminus \{f\}$; in all other cases we define $M_0' = M_0$. Thus, if in π we have $e \in \text{opt}(\overline{A(M_0)}, \overline{B(M_0)})$, then in $f(\pi)$ we have $e \in \text{opt}(\overline{A(M_0')}, \overline{B(M_0')})$. Since the distribution of $f(\pi)$ is uniform conditioned on $e \in \pi(\alpha m, m]$:

$$\frac{\Pr[e \in \text{opt}(\overline{A(M_0')}, \overline{B(M_0')}) \text{ and } e \in \pi(\alpha m, m]]}{\Pr[e \in \pi(\alpha m, m]]} \geq \Pr[e \in \text{opt}(\overline{A(M_0)}, \overline{B(M_0)})],$$

Using $\Pr[e \in \pi(\alpha m, m]] = 1 - \alpha$ and summing over e:

$$\mathbb{E}_\pi \, |\text{opt}(\overline{A(M_0')}, \overline{B(M_0')}) \cap \pi(\alpha m, m]| \geq (1-\alpha)\, \mathbb{E}_\pi \, |\text{opt}(\overline{A(M_0)}, \overline{B(M_0)})|.$$

Since M_0' and M_0 differ by at most one edge, $|\text{opt}(\overline{A(M_0)}, \overline{B(M_0)})| \geq |\text{opt}(\overline{A(M_0')}, \overline{B(M_0')})| - 1$, and the Lemma follows. □

Lemma 7. *Assume that* $\mathbb{E}_\pi \, |M_G| \leq (\frac{1}{2} + \epsilon)|M^*|$. *Then:*

$$\mathop{\mathbb{E}}_{\pi} |\text{opt}(A', \overline{B(M_0)}| \geq \mathop{\mathbb{E}}_{\pi} |M_1| - 4\epsilon |M^*|.$$

Proof. $|\text{opt}(A', \overline{B(M_0)}|$ is at least $|M_1|$ minus the number of edges of M_0 that are not 3-augmentable. Since M_0 is a subset of M_G, the latter term is bounded by the number of edges of M_G that are not 3-augmentable, which by Lemma 1 is in expectation at most $(\frac{1}{2} + \epsilon)|M^*| - (\frac{1}{2} - 3\epsilon)|M^*| = 4\epsilon|M^*|$. □

Lemma 8. $\mathop{\mathbb{E}}_{\pi} |M_2| \geq \dfrac{1}{2}((1-\beta) \mathop{\mathbb{E}}_{\pi} |\text{opt}(A', \overline{B(M_0)})| - 1)$.

Proof. Since Greedy computes a maximal matching which is at least half the size of a maximum matching,

$$\mathop{\mathbb{E}}_{\pi} |M_2| \geq \frac{1}{2} \mathop{\mathbb{E}}_{\pi} |\text{opt}(A', \overline{B(M_0)}) \cap \pi(\beta m, m]|.$$

Formally, we define a map f from the uniform distribution on all orderings to the uniform distribution on all orderings such that $e \in \pi(\beta m, m]$: if $e \in \pi(\beta m, m]$ then $f(\pi) = \pi$ and otherwise $f(\pi)$ is the permutation obtained from π by removing e and re-inserting it at a position picked uniformly at random in $(\beta m, m]$; in the latter case, if this causes an edge $e' = a'b'$, previously arriving at time $\lfloor \beta m \rfloor + 1$, to now arrive at time $\lfloor \beta m \rfloor$ and to be added to M_1, we define $A'' = A' \setminus \{M_0(b')\}$; in all other cases we define $A'' = A'$. Thus, if in π we have $e \in \text{opt}(A', \overline{B(M_0)})$, then in $f(\pi)$ we have $e \in \text{opt}(A'', \overline{B(M_0)})$. Since the distribution of $f(\pi)$ is uniform conditioned on $e \in \pi(\beta m, m]$:

$$\frac{\Pr[e \in \text{opt}(A'', \overline{B(M_0)}) \text{ and } e \in \pi(\beta m, m]]}{\Pr[e \in \pi(\beta m, m]]} \geq \Pr[e \in \text{opt}(A', \overline{B(M_0)})],$$

Using $\Pr[e \in \pi(\beta m, m]] = 1 - \beta$ and summing over e:

$$\mathop{\mathbb{E}}_{\pi} |\text{opt}(A'', \overline{B(M_0)}) \cap \pi(\beta m, m]| \geq (1-\beta) \mathop{\mathbb{E}}_{\pi} |\text{opt}(A', \overline{B(M_0)})|.$$

Since A' and A'' differ by at most one vertex,
$|\mathrm{opt}(A'', \overline{B(M_0)})| \geq |\mathrm{opt}(A', \overline{B(M_0)})| - 1$, and the Lemma follows. □

We now present the proof of the main theorem, Theorem 1.

Theorem 1. *Algorithm 4 is a deterministic one-pass semi-streaming algorithm for* MBM *with expected approximation ratio* $\frac{1}{2} + 0.005$ *against (uniform) random order for any graph, and can be implemented with* $O(1)$ *processing time per letter.*

Proof. Assume that $\mathbb{E}_\pi |M_G| \leq (\frac{1}{2} + \epsilon)|M^*|$. By construction, every $e \in M_2$ completes a 3−augmenting path, hence $|M| \geq |M_0| + |M_2|$. In Lemma 2 we show that $\mathbb{E}_\pi |M_0| \geq |M^*|(\frac{1}{2} - (\frac{1}{\alpha} - 2)\epsilon)$. By Lemmas 8 and 7, $|M_2|$ can be related to $|M_1|$:

$$\mathop{\mathbb{E}}_{\pi} |M_2| \geq \frac{1}{2}(1-\beta) \mathop{\mathbb{E}}_{\pi} |\mathrm{opt}(A', \overline{B(M_0)})| - \frac{1}{2} \geq \frac{1}{2}(1-\beta)(\mathop{\mathbb{E}}_{\pi} |M_1| - 4\epsilon|M^*|) - \frac{1}{2}.$$

By Lemmas 6 and 5, $|M_1|$ can be related to $|M^*|$:

$$\begin{aligned} \mathop{\mathbb{E}}_{\pi} |M_1| &\geq \tfrac{1}{2}(\beta - \alpha) \mathop{\mathbb{E}}_{\pi} |\mathrm{opt}(\overline{A(M_0)}, B(M_0)| - O(1) \\ &\geq \tfrac{1}{2}(\beta - \alpha)(|M^*|(\frac{1}{2} - (\frac{1}{\alpha} + 2)\epsilon)) - O(1). \end{aligned}$$

Combining,

$$\mathop{\mathbb{E}}_{\pi} |M| \geq |M^*|(\tfrac{1}{2} - (\tfrac{1}{\alpha} - 2)\epsilon + \tfrac{1}{2}(1-\beta)(\tfrac{1}{2}(\beta - \alpha)(\tfrac{1}{2} - (\tfrac{1}{\alpha} + 2)\epsilon) - 4\epsilon)) - O(1).$$

The expected value of the output of the Algorithm is at least $\min_\epsilon \max\{(\frac{1}{2} + \epsilon)|M^*|, \mathbb{E}_\pi |M|\}$. We set the right hand side of the above Equation equal to $(\frac{1}{2} + \epsilon)|M^*|$. By a numerical search we optimize parameters α, β. Setting $\alpha = 0.4312$ and $\beta = 0.7595$, we obtain $\epsilon \approx 0.005$ which proves the Theorem. □

4.3.3 Extension to General Graphs

In this section, we show how the one-pass algorithm of Section 4.3.2 can be adapted to general graphs $G = (V, E)$.

Algorithm. Algorithm 5 follows the same line as Algorithm 4 for the bipartite case. While in the bipartite case, edges from M_1 extend M_0 on only one bipartition, and those edges do not interfere with edges from M_2, this structure is no longer given in the general setting. Here, M_1 is a Greedy matching between the matched vertices in M_0 and all free vertices. This may already produce some 3-augmenting paths, however, it may also happen that by taking a *bad* edge into M_1, this rules out any possibility of finishing the 3-augmenting paths containing these edges. We call the edge of M_0 *blocked* if it can not be completed to a 3-augmenting path, see Definition 9.

We show in Lemma 11 that the probability that an edge of M_0 will become blocked is at most $1/2$. This guarantees that we can finalize many 3-augmenting paths by the Greedy matching M_2.

Aug is a set of length 3 paths. $|Aug|$ denotes the number of length 3 paths in Aug. For some vertex $a \in V$ (resp. some edge $e \in E$), we write $a \in Aug$ (resp. $e \in Aug$) if a (resp. e) is part of some length 3 path.

Analysis. We bound the size of a maximum matching between $V(M_0)$ and $\overline{V(M_0)}$.

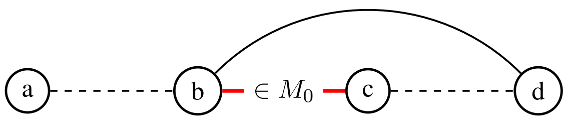


Figure 4.6: If edge bd is taken into M_1 and edge $ac \notin E$, this may block the 3-augmenting path ab, bc, cd. In that case we call bc blocked.

Algorithm 5 One-pass Matching on Random Order for General Graphs

```
1: $\alpha \leftarrow 0.413, \beta \leftarrow 0.708$
2: $M_G \leftarrow \text{Greedy}(\pi)$
3: $M_0 \leftarrow \text{Greedy}(\pi[1, \alpha m])$, matching obtained by Greedy on the first $\lfloor \alpha m \rfloor$ edges
4: $F_1 \leftarrow$ complete bipartite graph between $V(M_0)$ and $\overline{V(M_0)}$
5: $M_1 \leftarrow \text{Greedy}(F_1 \cap \pi(\alpha m, \beta m])$, matching obtained by Greedy on edges $\lfloor \alpha m \rfloor + 1$ through $\beta m$
   that intersect $F_1$
6: $Aug \leftarrow$ length 3 paths in $M_0 \oplus M_1$
7: $V_1 \leftarrow \underline{\{u \in V} \setminus V(Aug) \,|\, \exists v \in V(M_1) : uv \in M_0\}$
8: $V_2 \leftarrow \overline{V(M_0)} \setminus V(Aug)$
9: $F_2 \leftarrow$ maximal bipartite graph between $V_1$ and $V_2$ such that $\nexists m_0 \in M_0 \setminus Aug, m_1 \in M_1 \setminus$
   $Aug, f_2 \in F_2$ st. they form a triangle
10: $M_2 \leftarrow \text{Greedy}(F_2 \cap \pi(\beta m, m])$, matching obtained by Greedy on edges $\lfloor \beta m \rfloor + 1$ through $m$ that
   intersect $F_2$
11: $M \leftarrow$ matching obtained from $M_0$ augmented by $M_1 \cup M_2$
12: **return** larger of the two matchings $M_G$ and $M$
```

Lemma 9. *Assume that* $\mathbb{E}_\pi |M_G| \leq (\frac{1}{2} + \epsilon)|M^*|$. *Then:*

$$\mathbb{E}\,|\text{opt}(V(M_0), \overline{V(M_0)})| \geq |M^*|(1 - 2(\frac{1}{\alpha} + 2)\epsilon).$$

Proof. The size of a maximum matching between $\overline{V(M_0)}$ and $V(M_0)$ is at least twice the number of augmenting paths of length 3 in $M_0 \oplus M^*$. By Lemma 1, in expectation, the number of augmenting paths of length 3 in $M_G \oplus M^*$ is at least $(\frac{1}{2} - 3\epsilon)|M^*|$. All of those are augmenting paths of length 3 in $M_0 \oplus M^*$, except for at most $|M_G| - |M_0|$. Hence, in expectation, M_0 contains $(\frac{1}{2} - 3\epsilon)|M^*| - (\mathbb{E}\,|M_G| - \mathbb{E}\,|M_0|)$ edges that are 3-augmentable. Lemma 2 applied to M_0 concludes the proof. □

Lemma 10. *Assume that* $\mathbb{E}_\pi |M_G| \leq (\frac{1}{2} + \epsilon)|M^*|$. *Then:*

$$\mathbb{E}\,|M_1| \geq \frac{1}{2}(\beta - \alpha)(\mathbb{E}\,|\text{opt}(V(M_0), \overline{V(M_0)})| - \frac{1}{1-\alpha}).$$

Proof. The proof is identical to the proof of Lemma 6. □

Definition 9 (Blocked edge). *Let $e = uv \in M_0$ such that e is 3-augmentable by edges $o_1 = uu', o_2 = vv' \in M^*$. We call e blocked, if:*

1. *either $uv' \in E$ or $u'v \in E$ (not both of them), and*

2. *if* $uv' \in E$ *then* $uv' \in M_1$*, otherwise* $u'v \in M_1$.

Lemma 11.

$$\Pr[e \textit{ blocked } \mid e \in M_0] \leq \frac{1}{2}.$$

Proof. W.l.o.g. let $uv' \in E$ and $u'v \notin E$.

$$\begin{aligned} \Pr[e \text{ blocked } \mid e \in M_0] &= \Pr[e \notin \mathit{Aug} \text{ and } uv' \in M_1 \mid e \in M_0] \\ &\leq \Pr[uv' \in M_1 \mid e \in M_0 \setminus \mathit{Aug}]. \end{aligned}$$

Since $\Pr[uv' \in M_1 \mid e \in M_0 \setminus \mathit{Aug}] = \Pr[vv' \in M_1 \mid e \in M_0 \setminus \mathit{Aug}]$, and since the events $(uv' \in M_1 \mid e \in M_0 \setminus \mathit{Aug})$ and $(vv' \in M_1 \mid e \in M_0 \setminus \mathit{Aug})$ exclude each other, the result follows.

□

Lemma 12.

$$\mathbb{E}\,|\mathrm{opt}(F_2)| \geq \max\{\frac{1}{2}(\mathbb{E}\,|M_1| - 4|\mathit{Aug}| - 4\epsilon|M^*|), 0\}.$$

Proof. The size of a maximum matching in F_2 is at least the number of length 2 paths in $M_0 \oplus M_1$ that can be completed to a 3-augmenting path. Denote by k_2 the number of length two paths in $M_0 \oplus M_1$. Then, $|M_1| = 2|\mathit{Aug}| + k_2$. A length 3 path may block at most 2 other length 2 paths from being completed.

By Lemma 1, the number of edges of $|M_G|$ that are not 3-augmentable is in expectation at most $(\frac{1}{2} + \epsilon)|M^*| - (\frac{1}{2} - 3\epsilon)|M^*| = 4\epsilon|M^*|$. Since M_0 is a subset of M_G, it follows that at most $4\epsilon|M^*|$ edges from M_0 are not 3-augmentable. Hence, the number of M_0 edges for which a length two path was found and which is 3-augmentable is at least $(k_2 - 2|\mathit{Aug}| - 4\epsilon|M^*|)$. In expectation, by Lemma 11, at most half of these edges are blocked. The Lemma follows. □

Lemma 13.

$$\mathbb{E}\,|M_2| \geq \frac{1}{2}\left((1-\beta)\,\mathbb{E}\,|\mathrm{opt}(F_2)| - 1\right).$$

Proof. This proof is identical to the proof of Lemma 8. □

We now present the proof of the main theorem, Theorem 2.

Theorem 2. *Algorithm 5 is a deterministic one-pass semi-streaming algorithm for* MAXIMUM MATCHING *with approximation ratio* $\frac{1}{2} + 0.00363$ *in expectation over (uniform) random order for any graph, and can be implemented with* $O(1)$ *processing time per letter.*

Proof. The expected matching size is

$$\mathbb{E}\,|M| \geq \mathbb{E}\,|M_0| + |\mathit{Aug}| + \frac{1}{2}\,\mathbb{E}\,|M_2|, \tag{4.2}$$

since, by construction, at least half of the edges of M_2 can be used to complete a 3-augmenting path. Firstly, we bound $|M_2|$ by Lemma 13 and Lemma 12 and we obtain

$$\mathbb{E}\,|M_2| \geq \max\{0, (1-\beta)(\frac{1}{4}\,\mathbb{E}\,|M_1| - |Aug| - \epsilon|M^*|) - O(1)\}. \tag{4.3}$$

By Lemma 10 and Lemma 9, we bound the size of M_1 and we obtain

$$\mathbb{E}\,|M_1| \geq \frac{1}{2}|M^*|(\beta-\alpha)(1-2(\frac{1}{\alpha}+2)\epsilon) - O(1). \tag{4.4}$$

Using Inequality 4.4 in Inequality 4.3, we obtain

$$\mathbb{E}\,|M_2| \geq \max\{0, (1-\beta)(\frac{1}{8}|M^*|\left((\beta-\alpha)(1-2(\frac{1}{\alpha}+2)\epsilon) - \epsilon\right) - |Aug|) - O(1)\}. \tag{4.5}$$

Furthermore, in Lemma 2 we show that $\mathbb{E}_\pi\,|M_0| \geq |M^*|(\frac{1}{2} - (\frac{1}{\alpha} - 2)\epsilon)$. We use this and Inequality 4.5 in Inequality 4.2 and we obtain an Inequality for $\mathbb{E}\,|M|$ that depends on $\alpha, \beta, |Aug|$ and ϵ. It is easy to see that this Inequality is minimized if $|Aug| = 0$.

The expected value of the output of the Algorithm is at least $\min_\epsilon \max\{(\frac{1}{2} + \epsilon)|M^*|, \mathbb{E}_\pi\,|M|\}$. By a numerical search we optimize parameters α, β. Setting $\alpha = 0.413, \beta = 0.708$, we obtain $\epsilon \approx 0.00363$. which proves the Theorem.

□

4.4 Randomized Two-pass Algorithm on any Order

We present now a randomized two-pass semi-streaming algorithm for maximum matching for bipartite graphs with approximation ratio strictly greater than $\frac{1}{2}$. This algorithm simulates the three passes of the 3-pass algorithm of Section 4.2 in two passes. We require a new property of the Greedy algorithm that may be of independent interest. In Subsection 4.4.1, we discuss this new property. Then, we present in Subsection 4.4.2 our two-pass randomized algorithm for bipartite graphs.

4.4.1 Matching Many Vertices of a Random Vertex Subset

Consider a bipartite graph $G = (A, B, E)$. For a fixed parameter $0 < p \leq 1$, Algorithm 6 generates an independent random sample of vertices $A' \subseteq A$ such that $\Pr[a \in A'] = p$, for all $a \in A$, and runs then the Greedy algorithm on the subgraph $G|_{A' \times B}$.

Algorithm 6 Matching a Random Subset of Vertices (Bipartite Graphs)

```
1: Take independent random sample $A' \subseteq A$ st. $\Pr[a \in A'] = p$, for all $a \in A$
2: Let $F$ be the complete bipartite graph between $A'$ and $B$
3: return $M' = \mathrm{Greedy}(F \cap \pi)$
```

We prove in Theorem 3 that the greedy algorithm restricted to the edges with an endpoint in A' will output a matching of expected approximation ratio $p/(1+p)$, compared to a maximum matching $\mathrm{opt}(G)$ over the full graph G. Since, in expectation, the size of A' is $p|A|$, one can roughly say that a fraction of $1/(1+p)$ of vertices in $|A'|$ has been matched.

The proof of Theorem 3 will use Wald's equation for super-martingales, see [MU05], Wald's Equation, p.300, section 12.3.[1]

Lemma 14 (Wald's equation)**.** *Consider a process described by a sequence of random states $(S_i)_{i\geq 0}$ and let D be a random stopping time for the process, such that $\mathbb{E}\, D < \infty$. Let $(\Phi(S_i))_{i\geq 0}$ be a sequence of random variables for which there exist c, μ such that*

1. $\Phi(S_0) = 0$;
2. $\Phi(S_{i+1}) - \Phi(S_i) < c$ *for all* $i < D$; *and*
3. $\mathbb{E}[\Phi(S_{i+1}) - \Phi(S_i) \mid S_i] \leq \mu$ *for all* $i < D$.

Then:

$$\mathbb{E}\, \Phi(S_D) \leq \mu \, \mathbb{E}\, D.$$

Theorem 3. *Let $0 < p \leq 1$, let $G = (A, B, E)$ be a bipartite graph. Let A' be an independent random sample $A' \subset A$ such that $\Pr[a \in A'] = p$, for all $a \in A$. Let F be the complete bipartite graph between A' and B Then for any input stream $\pi \in \Pi(G)$:*

[1]The theorem cited in the book is actually weaker than the one we need, but our statement follows from the proof of that Theorem. Another source is available online at `http://greedyalgs.info/blog/stopping-times-walds/`

$$\mathop{\mathbb{E}}_{A'} |\text{Greedy}(F \cap \pi)| \geq \frac{p}{1+p} |\text{opt}(G)|.$$

Proof. Let $M' = \text{Greedy}(F \cap \pi)$. For $i \leq |M'|$, denote by M'_i the first i edges of M', in the order in which they were added to M' during the execution of Greedy.

Let M^* be a fixed maximum matching in G and let M_F denote the edges of M^* that are in F. Let $A'' = A(M_F)$ denote the vertices of A' matched by M_F. Consider a vertex $a \in A''$ and its match b in matching M_F. We say that a is *live* with respect to M'_i if both a and b are unmatched in M'_i. A vertex that is not live is *dead*. Furthermore, we say that an edge of $M'_{i+1} \setminus M'_i$ *kills* a vertex a if a is live with respect to M'_i and dead with respect to M'_{i+1}.

We use Lemma 14. Here, by "time", we mean the number of edges in M', so between time $i-1$ and time i, during the execution of Greedy, several edges arrive and all are rejected except the last one which is added to M'. We use a potential function $\phi(i)$ which we define as the number of dead vertices wrt. M'_i. We define the stopping time D as the first time when the event $\phi(i) = |A''|$ holds.

We only need to check that the three assumptions of the Stopping Lemma hold. First, initially all nodes of A'' are live, so $\phi(0) = 0$. Second, the potential function ϕ is non-decreasing and uniformly bounded: since adding an edge to M' can kill at most two vertices of A'', we always have $\Delta\phi(i) := \phi(i+1) - \phi(i) \leq 2$. Third, let S_i denote the state of the process at time i, namely the information about the entire sequence of edge arrivals up to that time, hence, in particular, the set of i edges currently in M'. Observe that, here, G and M^* are fixed. Then D is indeed a stopping time, since the event $D \geq i+1$ can be inferred from the knowledge of S_i.

We now claim that:

$$\mathbb{E}(\Delta\phi(i) \mid S_i) \leq 1 + p. \tag{4.6}$$

Indeed, since $\Delta\phi(i)$ only takes on values 0, 1 or 2, we can write that $\mathbb{E}(\Delta\phi(i)|S_i) \leq 1 + \Pr[\Delta\phi(i) = 2|S_i]$. To bound the latter probability, let $e = ab$ denote the edge of $M'_{i+1} \setminus M'_i$ and let t be such that $e = \pi[t]$. In order for e to change ϕ by 2, it must be that b is matched in M^* to a node a' that is also in A''. Furthermore, it is required that a' was unmatched before edge e arrived. Since a' was unmatched up to arrival t, no edge $a'b'$ had been seen among the first t edges of stream π, such that b' was free at arrival time (of $a'b'$). Thus

$$\begin{aligned}&\Pr[\Delta\phi(i) = 2|S_i] \leq \\ &\qquad \Pr[a' \in A' \text{ and } \nexists a'b' \in \pi[1,t] \text{ st. } b' \text{ was free when } a'b' \text{ arrived}|S_i].\end{aligned}$$

Now, given that no edge $f = a'b'$ arrived before t such that b' was free when $a'b'$ arrived, the outcome of the random coin determining whether $a' \in A'$ was never looked at, and could have been postponed until t. Thus

$$\begin{aligned}&\Pr[a' \in A' \mid (\nexists a'b' \in \pi[1,t] \text{ such that } b' \text{ was free when } a'b' \text{ arrived}, S_i)] = \\ &\qquad \Pr[a' \in A'] = p,\end{aligned}$$

implying Inequality 4.6. Applying Walds' Stopping Lemma, we obtain $\mathbb{E}\,\phi(D) \leq (1+p)\,\mathbb{E}\,D$. Finally, observe that $\mathbb{E}\,\phi(D) = \mathbb{E}\,|A''| = p \cdot |\text{opt}(G)|$ and that $D \leq |\text{Greedy}(F \cap \pi)|$, and the Theorem follows. □

4.4.2 A Randomized Two-pass Algorithm for Bipartite Graphs

Based on Theorem 3, we design our randomized two-pass algorithm for bipartite graphs $G = (A, B, E)$. Assume that $\mathrm{Greedy}(\pi)$ returns a matching that is close to a $\frac{1}{2}$-approximation. In order to apply Theorem 3, we pick an independent random sample $A' \subseteq A$ such that $\Pr[a \in A'] = p$ for all a. In a first pass, our algorithm computes a Greedy matching M_0 of G, and a Greedy matching M' between vertices of A' and B. M' then contains some edges that form parts of 3-augmenting paths for M_0: see Figure 4.7 and Figure 4.8 for an illustration. Let $M_1 \subset M'$ be the set of those edges. It remains to complete these length 2 paths $M_0 \cup M_1$ in a second pass by a further Greedy matching M_2. In the prove of Theorem 4, we show that if $\mathrm{Greedy}(\pi)$ is close to a $\frac{1}{2}$-approximation, then we find many 3-augmenting paths.

Algorithm 7 Two-pass Randomized Bipartite Matching Algorithm

```
1: Let $p \leftarrow \sqrt{2} - 1$.
2: Take an independent random sample $A' \subseteq A$ st. $\Pr[a \in A'] = p$, for all $a \in A$
3: Let $F_1$ be the set of edges with one endpoint in $A'$.
4: **First pass:** $M_0 \leftarrow \mathrm{Greedy}(\pi)$ and $M' \leftarrow \mathrm{Greedy}(F_1 \cap \pi)$
5: $M_1 \leftarrow \{e \in M' \mid e$ goes between $B(M_0)$ and $\overline{A(M_0)}\}$
6: $A_2 \leftarrow \{a \in A(M_0) : \exists b, c : ab \in M_0$ and $bc \in M_1\}$.
7: Let $F_2 \leftarrow \{da : d \in \overline{B(M_0)}$ and $a \in A(M_0)$ and $\exists b, c : ab \in M_0$ and $bc \in M_1\}$.
8: **Second pass:** $M_2 \leftarrow \mathrm{Greedy}(F_2 \cap \pi)$
9: Augment $M_0$ by edges in $M_1$ and $M_2$ and store it in $M$
10: **return** the resulting matching $M$
```

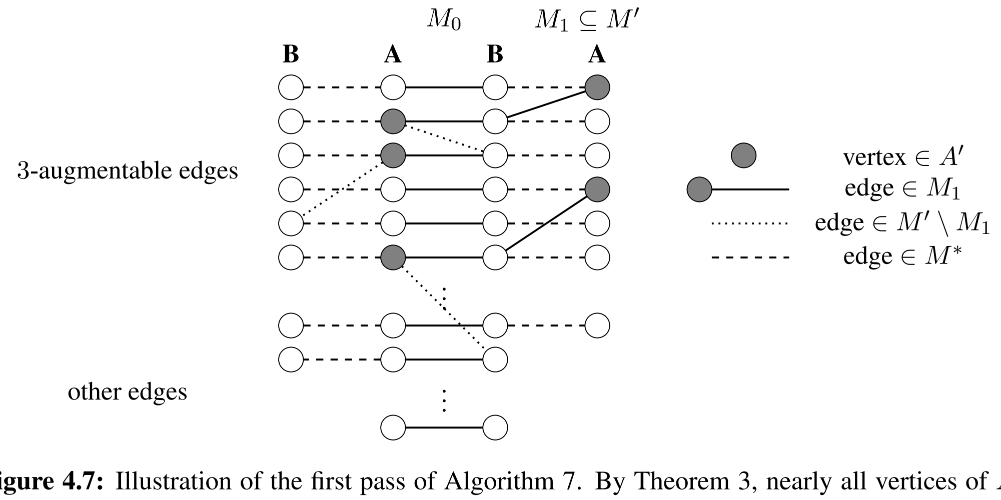


Figure 4.7: Illustration of the first pass of Algorithm 7. By Theorem 3, nearly all vertices of A' are matched in M', in particular those that are not matched in M_0.

Theorem 4. *Algorithm 7 is a randomized two-pass semi-streaming algorithm for* MBM *with expected approximation ratio* $\frac{1}{2} + 0.019$ *in expectation over its internal random coin flips for any graph and any arrival order, and can be implemented with* $O(1)$ *processing time per letter.*

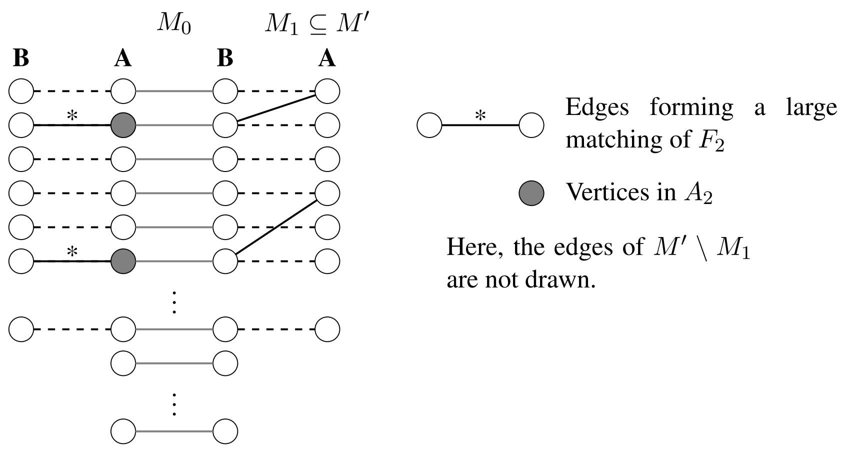

M_0
$M_1 \subseteq M'$
B
A
B
A
*
*
*
Edges forming a large matching of F_2
Vertices in A_2
Here, the edges of $M' \setminus M_1$ are not drawn.

Figure 4.8: Analysis of the second pass of Algorithm 7. Here, we see that $M_0 \oplus M_1$ has two paths of length 2, and that both of those paths can be extended into 3-augmenting paths using M^*: this illustrates $|\text{opt}(F_2)| \geq 2$. Matching M_2, being a $1/2$ approximation, will find at least one 3-augmenting path.

Proof. By construction, each edge in M_2 is part of a 3-augmenting path, hence the output has size: $|M| = |M_0| + |M_2|$.

Define ϵ to be such that $|M_0| = (\frac{1}{2} + \epsilon)|\text{opt}(G)|$. Since M_2 is a maximal matching of F_2, we have $|M_2| \geq \frac{1}{2}|\text{opt}(F_2)|$. Let M^* be a maximum matching of G. Then $|\text{opt}(F_2)|$ is greater than or equal to the number of edges ab of M_0 such that there exists an edge bc of M_1 and an edge da of M^* that altogether form a 3-augmenting path of M_0:

$$\begin{aligned} |\text{opt}(F_2)| &\geq |\{ab \in M_0 \mid \exists c : bc \in M_1 \text{ and } \exists d : da \in M^*\}| \\ &\geq |\{ab \in M_0 \mid \exists c : bc \in M_1\}| - |\{ab \in M_0 \mid ab \text{ not 3-augmentable}\}|. \end{aligned}$$

Lemma 1 gives $|\{ab \in M_0 \mid ab \text{ is not 3-augmentable with } M^*\}| \leq 4\epsilon|\text{opt}(G)|$. It remains to bound $|\{ab \in M_0 \mid \exists c : bc \in M_1\}|$ from below. By definition of M' and of $M_1 \subseteq M'$, and by maximality of M_0,

$$\begin{aligned} |\{ab \in M_0 \mid \exists c : bc \in M_1\}| &= |M'| - |\{ab \in M' \mid a \in A(M_0)\}| \\ &\geq |M'| - |A(M_0) \cap A'|. \end{aligned}$$

Taking expectations, by Theorem 3 and by independence of M_0 from A':

$$\mathop{\mathbb{E}}_{A'} |M'| - \mathop{\mathbb{E}}_{A'} |A(M_0) \cap A'| \geq \frac{p}{1+p}|\text{opt}(G)| - p(\frac{1}{2} + \epsilon)|\text{opt}(G)|.$$

Combining:

$$\mathop{\mathbb{E}}_{A'} |M| \geq (\frac{1}{2} + \epsilon)|\text{opt}(G)| + \frac{1}{2}\left(|\text{opt}(G)|p(\frac{1}{1+p} - \frac{1}{2} - \epsilon) - 4\epsilon|\text{opt}(G)|.\right)$$

For ϵ small, the right hand side is maximized for $p = \sqrt{2} - 1$. Then $\epsilon \approx 0.019$ minimizes $\max\{|M|, |M_0|\}$ which proves the theorem.

□

4.5 Deterministic Two-pass Algorithm on any Order

We discuss now deterministic two-pass streaming algorithms for MBM and MM for input streams in adversarial order. We start our presentation with an algorithm for bipartite graphs in Section 4.5.1. Then, we show how this idea can be extended to general graphs in Section 4.5.2.

4.5.1 Bipartite Graphs

Algorithm. The deterministic two-pass algorithm, Algorithm 9, follows the same line as its randomized version, Algorithm 7. In a first pass, we compute a Greedy matching M_0 and some additional edges S that we compute by Algorithm 8. If M_0 is close to a $\frac{1}{2}$-approximation then S contains edges that serve as parts of 3-augmenting paths. These are completed via a Greedy matching in the second pass.

The way we compute the edge set S is now different. In Algorithm 7, S was a matching M' between B and a random subset A' of A. Now, S is not a matching but a relaxation of matchings as follows. Given an integer $\lambda \geq 2$, an *incomplete λ-bounded semi-matching S* of a bipartite graph $G = (A, B, E)$ is a subset $S \subseteq E$ such that $\deg_S(a) \leq 1$ and $\deg_S(b) \leq \lambda$, for all $a \in A$ and $b \in B$. This notion is closely related to semi-matchings. A semi-matching matches all A vertices to B vertices without limitations on the degree of a B vertex. However, since we require that the B vertices have constant degree, we loosen the condition that all A vertices need to be matched.

In Lemma 15, we show that Algorithm 8, a straightforward greedy algorithm, computes an incomplete λ-bounded semi-matching that covers at least $\frac{\lambda}{\lambda+1}|M^*|$ vertices of A.

Algorithm 8 Incomplete λ-bounded Semi-matching iSEMI(λ)

```
$S \leftarrow \varnothing$
**while** $\exists$ edge $ab$ in stream
  **if** $\deg_S(a) = 0$ and $\deg_S(b) \leq \lambda - 1$ **then** $S \leftarrow S \cup \{ab\}$
**return** $S$
```

Now, assume that the greedy matching algorithm computes a M_0 close to a $\frac{1}{2}$-approximation. Then, for $\lambda \geq 2$ there are many A vertices that are not matched in M_0 but are matched in S. Edges incident to those in S are candidates for the construction of 3-augmenting paths. This argument can be made rigorous, leading to Algorithm 9 where λ is set to 3, see Theorem 5.

We show two figures illustrating the first pass (Figure 4.9) and the second pass (Figure 4.10) of Algorithm 9.

Analysis. We firstly present a lemma, Lemma 15, that analyses Algorithm 8. This lemma is then used in the proof of the main theorem, Theorem 5.

Lemma 15. *Let $S = \text{iSEMI}(\lambda)$ be the output of Algorithm 8 for some $\lambda \geq 2$. Then S is an incomplete λ-bounded semi-matching such that $|A(S)| \geq \frac{\lambda}{\lambda+1}|M^*|$.*

Proof. By construction, S is an incomplete degree λ bounded semi-matching. We bound $A(M^*) \setminus A(S)$ from below. Let $a \in A(M^*) \setminus A(S)$ and let b be its mate in M^*. The

Algorithm 9 Two-pass Deterministic Bipartite Matching Algorithm

```
**First pass:** $M_0 \leftarrow$ Greedy($\pi$) and $S \leftarrow$ iSEMI(3)
$S_1 \leftarrow \{e \in S \mid e = ab$ such that $a \in \overline{A(M_0)}$ and $b \in B(M_0)\}$
$A_2 \leftarrow \{a \in A(M_0) \mid \exists bc : ab \in M_0$ and $bc \in S_1\}$
$F \leftarrow \{e \mid e = ab$ such that $a \in A_2$ and $b \in \overline{B(M_0)}\}$
**Second pass:** $M_2 \leftarrow$ Greedy($\pi \cap F$)
Augment $M_0$ by edges in $S_1$ and $M_2$ and store it in $M$
**return** $M$
```

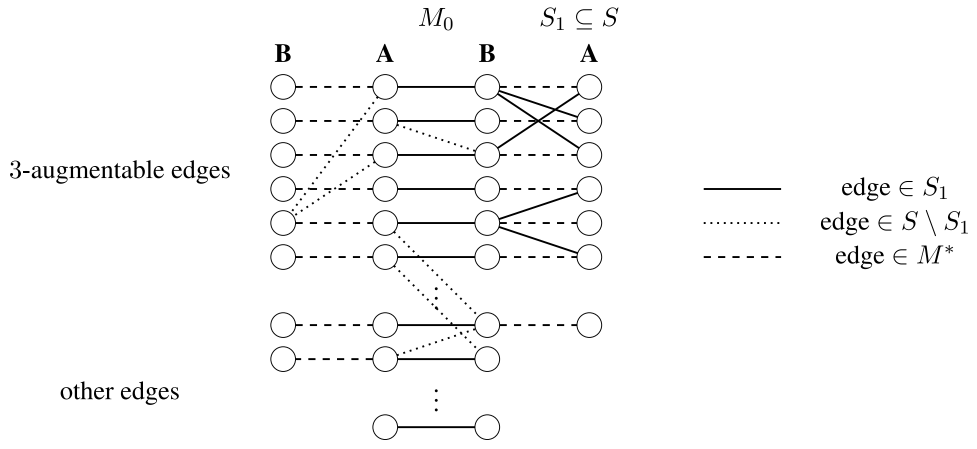


Figure 4.9: Illustration of the first pass of Algorithm 9. In this example we set $\lambda = 2$ and we compute an incomplete degree 2 limited semi-matching S. By Lemma 15, we match at least $\frac{2}{3}|M^*|$ A vertices. Since $|M| \approx \frac{1}{2}|M^*|$, some A vertices that are not matched in M_0 are matched in S. The edges incident to those define S_1.

algorithm did not add the optimal edge ab upon its arrival. This implies that b was already matched to λ other vertices. Hence, $|A(M^*) \setminus A(S)| \leq \frac{1}{\lambda}|A(S)|$. Then the result follows by combining this inequality with $|M^*| - |A(S)| \leq |A(M^*) \setminus A(S)|$.

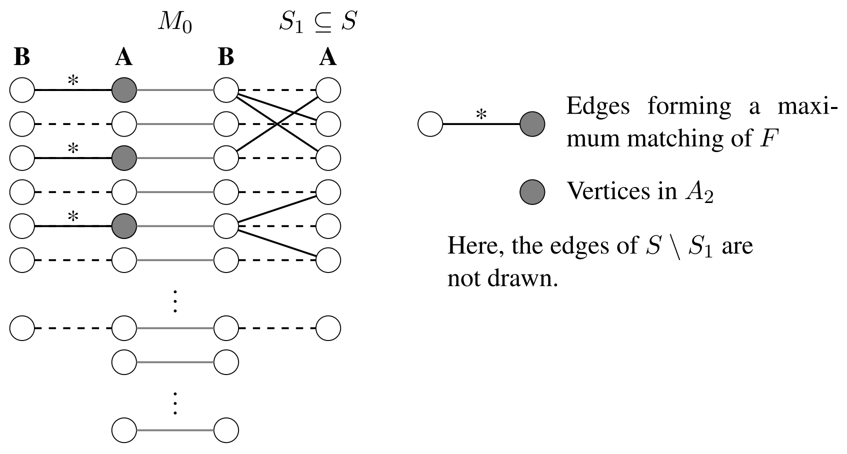
□

Theorem 5. *Algorithm 9 is a deterministic two-pass semi-streaming algorithm for* MBM *with approximation ratio* $\frac{1}{2} + 0.019$ *for any graph and any arrival order and can be implemented with* $O(1)$ *processing time per edge.*

Proof. The computed matching M is of size $|M_0| + |M_2|$ since, by construction, for each edge in M_2 there is at least one distinct edge in S_1 that allows the construction of a 3-augmenting path. Each 3-augmenting path increases the matching M_0 by 1. See also Figure 4.10. Since $|M_2|$ is a maximal matching of the graph induced by the edges F, we obtain

$$|M| \geq |M_0| + \frac{1}{2}|\text{opt}(F)|.$$

Let ϵ be such that $|M_0| = (\frac{1}{2} + \epsilon)|M^*|$. By Lemma 1, at most $4\epsilon|M^*|$ edges of M_0 are not



M_0
$S_1 \subseteq S$
B
A
B
A
*
Edges forming a maximum matching of F
Vertices in A_2
Here, the edges of $S \setminus S_1$ are not drawn.

Figure 4.10: Analysis of the second pass of Algorithm 9. In this example, we set $\lambda = 2$. Here, we see that $M_0 \oplus S_1$ has five paths of length 2. These paths are not disjoint, but since the maximal degree in S is 2, $M_0 \oplus S_1$ has at least $\frac{1}{2} \cdot 5$ disjoint paths, and hence $|A_2| = 3 \geq \frac{1}{2} \cdot 5$. A maximum matching in F is of size 3, and in the second pass, Greedy will find at least half of them leading to at least two 3-augmenting paths.

3-augmentable, hence

$$\text{opt}(F) \geq |A_2| - 4\epsilon|M^*|.$$

A_2 are those vertices matched also by M_0 such that there exists an edge in S_1 matching the mate of the A_2 vertex. Since the maximal degree in S_1 is λ, we can bound $|A_2|$ by

$$|A_2| \geq \frac{1}{\lambda}|S_1|.$$

Note that $|S_1| = |A(S) \setminus A(M_0)|$ since the degree of an A vertex matched by S in S is one, and S can be partitioned into $S_{M_0}, S_{\overline{M_0}}$ such that edges in S_{M_0} couple an A vertex also matched in M_0, and edges in $S_{\overline{M_0}}$ couple an A vertex that is not matched in M_0. Now, $|S_1| = |S_{\overline{M_0}}|$ since an edge of S is taken into S_1 if it is in $S_{\overline{M_0}}$.

Lemma 15 allows us to bound the size of the set $A(S) \setminus A(M_0)$ via

$$|A(S) \setminus A(M_0)| \geq |A(S)| - |A(M_0)| \geq (\frac{\lambda}{\lambda+1} - \frac{1}{2} - \epsilon)|M^*|.$$

Using the prior Inequalities, we obtain

$$|M| \geq (\frac{1}{2} - \epsilon + \frac{1}{2\lambda+2} - \frac{1}{4\lambda} - \frac{\epsilon}{2\lambda})|M^*|.$$

Since we have also $|M| \geq |M_0| = (\frac{1}{2} + \epsilon)|M^*|$, we set

$$\begin{aligned} \epsilon_0 &= \arg\min_{\epsilon} \max\{(\frac{1}{2} - \epsilon + \frac{1}{2\lambda+2} - \frac{1}{4\lambda} - \frac{\epsilon}{2\lambda})|M^*|, (\frac{1}{2} + \epsilon)|M^*|\} \\ &= \frac{\lambda - 1}{8\lambda^2 + 10\lambda + 2}, \end{aligned}$$

which is maximized for $\lambda = 3$ leading to an approximation factor of $\frac{1}{2} + \frac{1}{52} \approx \frac{1}{2} + 0.019$.

Concerning the processing time per edge, note that once an edge is added in the second pass, a corresponding 3-augmenting path can be determined in time $O(1)$.

□

4.5.2 Extension to General Graphs

Algorithm. The deterministic two-pass algorithm for general graphs follows the same line as the deterministic two-pass algorithm for bipartite graphs. In the first pass, Greedy matching M together with some additional edges F are computed. F forms an *incomplete b-bounded forest*.

Definition 10 (incomplete b-bounded forest)**.** *Given an integer b, an incomplete b-bounded forest F is a cycle free subset of the edges of a graph $G = (V, E)$ with maximal degree b.*

If $F \oplus M$ contains 3-augmenting paths, we augment M by a maximal set of disjoint 3-augmenting paths and store the result in M'. Those edges of F that were not used in the previous augmentation and that form length-2 paths with edges of M' are stored in M_R. In a second pass, length-2 paths of $M' \cup M_R$ are completed to 3-augmenting paths by computing a matching M_L. A maximal set of disjoint 3-augmenting paths of $M' \cup M_L \cup M_R$ is then used to augment M'.

Algorithm 10 b-bounded Forest: FOREST(b)

```
Require: $b$
1: $S \leftarrow \varnothing$
2: while stream not empty do
3:     $uv \leftarrow$ next edge in stream
4:     if ($\deg_S(u) = 0$ and $\deg_S(v) \leq b - 1$) or ($\deg_S(u) \leq b - 1$ and $\deg_S(v) = 0$) then
5:         $S \leftarrow S \cup \{uv\}$
6:     end if
7: end while
8: return $S$
```

Algorithm 10 is a greedy algorithm that constructs a forest F such that the maximal degree of a node in F is b, for some $b \geq 1$. For a large enough b, all but a small fraction of the vertices of the graph are also in the vertex set of F.

The situation of the algorithm after the first pass is illustrated in Figure 4.11. Note that M_R is an incomplete b-bounded semi-matching in the induced bipartite graph with vertex sets $V \setminus V(M')$ and $V(M')$.

Analysis. The analysis refers to the variables that are used in the algorithm. Furthermore, let M^* denote a maximum matching in the input graph and let ϵ be such that $|M| = (1/2 + \epsilon)|M^*|$. Let $\alpha = \frac{|M'|}{|M|} - 1$, or in other words, the set of disjoint 3-augmenting paths found in Line 3 is of size $\alpha|M|$.

The analysis of the algorithm requires a lemma concerning the structure of forests.

Algorithm 11 Two-pass Deterministic Matching Algorithm for General Graphs

```
Require: $b$
1: $Aug \leftarrow \varnothing$
2: first pass: $M \leftarrow$ Greedy() and $F \leftarrow$ FOREST($b$)
3: $M' \leftarrow M$ augmented by a maximal set of 3-augmenting paths in $M \oplus F$
4: $M_R \leftarrow$ maximal subset of $F$ such that $\forall uv \in M_R : u \in V(M')$ and $\deg_{M_R}(v) = 1$
5: $V' \leftarrow \{v \in V(M') : v' = M'(v)$ and $\exists v'u \in M_R : u \notin V(M')\}$
6: while stream not empty do {second pass}
7:    $vw \leftarrow$ next edge in stream
8:    if $v \in V'$ and $w \notin V(M')$ and $vw$ completes a 3-augmenting path with edges $uv \in M', tu \in M_R$ then
9:       $Aug \leftarrow Aug \cup \{vw, tu\}$, remove all edges from $M_R$ incident to $u$
10:   end if
11: end while
12: $M'' \leftarrow M'$ augmented with $Aug$
13: return $M''$
```

Lemma 16. *Let T be a forest with at least k nodes of degree at least d. Then:*

$$|T| \geq (d-1)k.$$

Proof. Consider the directed Graph D that is obtained from T by directing the edges from the roots of the trees of T towards the leaves. Let $v_1, \dots, v_k$ denote the nodes that have degree at least d. Then for all $i \neq j : \Gamma_D(v_i) \cap \Gamma_D(v_j) = \varnothing$. Furthermore, for each $i : |\Gamma_D(v_i)| \geq (d-1)$. The result follows. □

Lemma 17. *Let M^* denote a maximum matching in $G = (V, E)$. Consider the state of F after the first pass. Then:*

$$|F| \geq (b-1)|V(M^*) \setminus V(F)|. \tag{4.7}$$

Proof. By induction it is easy to see that F is a forest with maximal degree b. We argue that F has at least $|V(M^*) \setminus V(F)|$ nodes of degree b. The result then follows by applying Lemma 16. Let $u \in V(M^*) \setminus V(F)$ and denote by v the mate of u in M^*. Since uv is not taken, the degree of v was already b upon arrival of uv. Hence, for each node $u \in V(M^*) \setminus V(F)$ the partner $M^*(u)$ has degree b in F. □

Lemma 18. *Let $|M| = (\frac{1}{2} + \epsilon)|M^*|$. Consider the state of F after the first pass. Then*

$$|V(F) \setminus V(M)| \geq (1 - 2\epsilon - \frac{2}{b})|M^*|.$$

Proof. By Lemma 17, $|F| \geq (b-1)|V(M^*) \setminus V(F)|$. Then

$$\begin{aligned} |V(F) \setminus V(M)| &\geq |V(F)| - |V(M)| \geq (b-1)|V(M^*) \setminus V(F)| - 2|M| \\ &= (b-1)|V(M^*) \setminus V(F)| - (1+2\epsilon)|M^*|. \end{aligned} \tag{4.8}$$

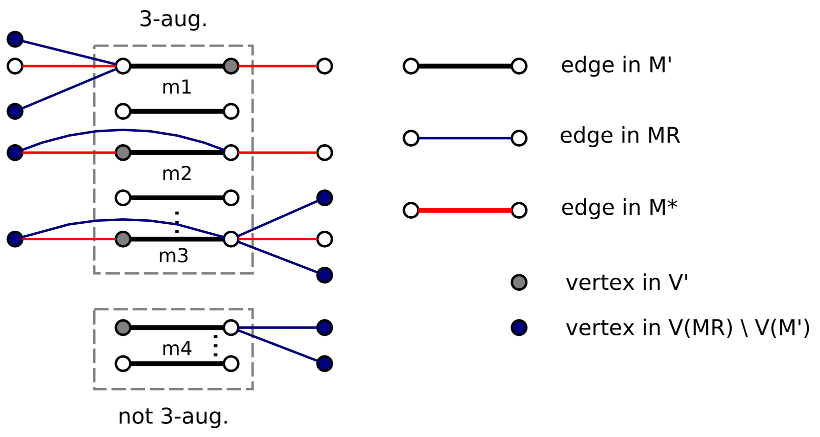


Figure 4.11: Analysis of Algorithm 11.

Furthermore,we also have $|V(F)| \geq 2|M^*| - |V(M^*) \setminus V(F)|$, and hence

$$\begin{aligned} |V(F) \setminus V(M)| &\geq |V(F)| - |V(M)| \geq 2|M^*| - |V(M^*) \setminus V(F)| - 2|M| \\ &= 2|M^*| - |V(M^*) \setminus V(F)| - (1+2\epsilon)|M^*| \\ &= (1-2\epsilon)|M^*| - |V(M^*) \setminus V(F)|. \qquad (4.9) \end{aligned}$$

Then $|V(M^*) \setminus V(F)| = \frac{2|M^*|}{b}$ minimizes $\max\{4.8, 4.9\}$ and we obtain $|V(F) \setminus V(M)| \geq \frac{2|M^*|}{b}$. □

Lemma 19. *Consider the state of the variables of the algorithm before the second pass. Let $M'_a \subseteq M'$ such that $\forall m \in M'_a$ there is an edge $m_R \in M_R$ and an edge $m_L \in E$ such that m_R, m, m_L forms a 3-augmenting path. Then:*

$$|M'_a| \geq \frac{1}{b}\left(|V(M_R) \setminus V(M')| - |M'|\right) - 4(\epsilon + \frac{1}{2}\alpha + \alpha\epsilon)|M^*|.$$

Proof. The set M'_a is precisely the subset of edges uv of M' that fulfill the following two conditions.

1. uv is 3-augmentable, and
2. uv has an edge of M_R incident that is not a *blocking* edge.

We say that an edge $m_R = u'v \in M_R$ is a blocking edge, if uv is the incident edge of M', uu', vv' are the edges incident to uv in $M' \oplus M^*$, and the edge $u'v$ is not in the graph G. See Figure 4.12 for an illustration. Note that there are at most $|M'|$ blocking edges in the graph.

We consider the vertices that are matched in M_R but are free in M'. Each vertex $v \in V(M_R) \setminus V(M')$ is connected by an edge of M_R to an edges of M'. We remove from $V(M_R) \setminus V(M')$ these vertices that have a blocking edge incident. There are at most $|M'|$ blocking edges. Since the maximal degree in M_R is b, there are at least $1/b(|V(M_R) \setminus V(M')| - |M'|)$

edges in M' that fulfill condition (2). By Lemma 1, there are at most $4(\epsilon + \frac{1}{2}\alpha + \alpha\epsilon)|M^*|$ edges in M' that are not 3-augmentable, and the result follows.

□

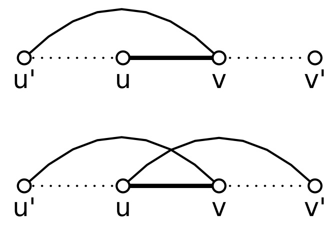


Figure 4.12: Illustration of a blocking edge. In the first setting, the edge $u'v$ is a blocking edge, since the edge uv' is not in the graph. The edge $u'v$ blocks edge uu' from augmenting uv. In the second setting, neither $u'v$ nor uv' are blocking edges. $u'v$ blocks the edge vv', however, the edge $u'v$ is an alternative for the node v for being augmented. This alternative is not present in the first figure.

Theorem 6. *Algorithm 11 with $b = 8$ is a deterministic 2-pass semi-streaming algorithm for* Maximum Matching *with approximation ratio $1/2 + 1/140 \approx 1/2 + 0.007142$ for any graph and any arrival order.*

Proof. By construction, the computed matching M'' is of size $|M'| + |Aug|$. Since $|M'| = (1+\alpha)|M|$ and $|M| = (\frac{1}{2} + \epsilon)|M^*|$, we obtain

$$|M''| = (1+\alpha)(\frac{1}{2} + \epsilon)|M^*| + |Aug|. \tag{4.10}$$

It remains to lower bound $|Aug|$.

In Lemma 19, we show that there is a subset $M'_a \subseteq M'$ such that

$$|M'_a| \geq \frac{1}{b}\left(|V(M_R) \setminus V(M')| - |M'|\right) - 4(\epsilon + \frac{1}{2}\alpha + \alpha\epsilon)|M^*|,$$

and for each edge of M'_a there is a 3-augmenting path with an edge from M_R and another edge from the stream. Any 3-augmenting path that is added in Line 9 of Algorithm 11 to Aug may block at most 2 further edges of M'_a from being augmented, see Figure 4.13. We will find hence at least $\frac{1}{3}|M'_a|$ 3-augmenting paths, and we obtain

$$|Aug| \geq 1/3|M'_a| \geq \frac{1}{3}\left(\frac{1}{b}\left(|V(M_R) \setminus V(M')| - |M'|\right) - 4(\epsilon + \frac{1}{2}\alpha + \alpha\epsilon)|M^*|\right). \tag{4.11}$$

Note that by construction, $|V(M_R) \setminus V(M')| = |V(F) \setminus V(M')|$. We bound now $|V(F) \setminus V(M')|$. By Lemma 18, $|V(F) \setminus V(M)| \geq (1 - 2\epsilon - \frac{2}{b})|M^*|$. Note that M' is the matching that is obtained by augmenting M with edges from F. Each augmented edge of M has two edges incident from F that are used for the augmentation. Hence,

$$|V(F) \setminus V(M')| \geq (1 - 2\epsilon - \frac{2}{b})|M^*| - 2\alpha|M|. \tag{4.12}$$

Using Inequality 4.12 and Inequality 4.11 in Inequality 4.10, we obtain

$$|M''| \geq \left(\frac{1}{2} + \frac{1}{6b} - \frac{1}{3}(\alpha\epsilon + \epsilon + \frac{\alpha}{2}) - \frac{1}{b}(\alpha\epsilon + \epsilon + \frac{\alpha}{2} + \frac{2}{3b})\right)|M^*|. \tag{4.13}$$

Note that we also have

$$|M''| \geq |M'| \geq |M^*|(\frac{1}{2} + \epsilon + \frac{\alpha}{2} + \alpha\epsilon). \tag{4.14}$$

We determine ϵ_0 as a function of α and b that minimizes the maximum of the right sides of Inequality 4.13 and Inequality 4.14. For any α and ϵ_0, M'' is maximized by setting $b = 8$. This leads to an approximation factor $1/2 + 1/140 \approx 1/2 + 0.007142$. $\square$

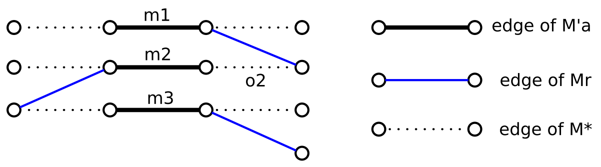


Figure 4.13: m_1, m_2, m_3 have each an edge of M_R incident and can be augmented with this edge and an incident edge from M^*. If m_2 is augmented with its incident edge from M_R and o_2, then this may prevent m_1 and m_3 from being augmented.

Chapter 5

Semi-Matchings in Streaming and in Two-Party Communication

We present now our results concerning the computation of semi-matchings in streaming algorithms and in the one-way two-party communication setting. In Section 5.1, we give notations and definitions the are required for the presentation of our results. Then, in Section 5.2 we present our streaming algorithms for the semi-matching problem. We present then upper and lower bounds for the semi-matching problem in the one-way two-party communication setting in Section 5.3. Finally, in Section 5.4 we give a new structure theorem for semi-matchings.

5.1 Preliminaries

Let $G = (A, B, E)$ be a bipartite graph and let $n = |A|$. We assume that the graph does not have isolated A-vertices in order to guarantee that the graph has a semi-matching. Furthermore, we assume that $|B| = \mathrm{poly}(n)$. Let $e \in E$ be an edge connecting nodes $a \in A$ and $b \in B$. Then, we write $A(e)$ to denote the vertex a, $B(e)$ to denote the vertex b, and ab to denote e. Furthermore, for a subset $E' \subseteq E$, we define $A(E') = \bigcup_{e \in E'} A(e)$ (respectively $B(E')$). For subsets $A' \subseteq A$ and $B' \subseteq B$ we write $E'|_{A' \times B'}$ to denote the subset of edges of E' whose endpoints are all in $A' \cup B'$. We denote by $E'(a)$ the set of edges of $E' \subseteq E$ that have an endpoint in vertex a, and $E'(A')$ the set of edges that have endpoints in vertices of A', where $A' \subseteq A$ (similarly we define $E'(B')$ for $B' \subseteq B$).

For a node $v \in A \cup B$, the *neighborhood* of v is the set of nodes that are adjacent to v and we denote it by $\Gamma(v)$. For a subset $E' \subseteq E$, we write $\Gamma_{E'}(v)$ to denote the neighborhood of v in the graph induced by E'. Note that by this definition $\Gamma(v) = \Gamma_E(v)$. For a subset $E' \subseteq E$, we denote by $\deg_{E'}(v)$ the *degree* in E' of a node $v \in V$, which is the number of edges of E' with an endpoint in v. We define $\deg \max E' := \max_{v \in A \cup B} \deg_{E'}(v)$.

Definition 11 (Semi-matching)**.** *A* semi-matching *in a bipartite graph $G = (A, B, E)$ is a subset $S \subseteq E$ such that $\forall a \in A : \deg_S(a) = 1$.*

An important notion for the computation of semi-matchings are *degree-minimizing-paths*.

Definition 12 (Degree-minimizing-path)**.** *A* degree-minimizing path P *with respect to a semi-matching S is a path $P = (b_1, a_1, \dots, b_{k-1}, a_{k-1}, b_k)$ of length $2k$ ($k \geq 1$) such that for all $i \leq k : (a_i, b_i) \in S$, for all $i \leq k-1 : (a_i, b_{i+1}) \notin S$, and $\deg_S(b_1) > \deg_S(b_2) \geq \deg_S(b_3) \geq \dots \geq \deg(b_{k-1}) > \deg(b_k)$.*

We define optimality of a semi-matching by means of degree-minimizing-paths.

Definition 13 (Optimal Semi-matching). *An* optimal semi-matching $S^* \subseteq E$ *is a semi-matching that does not admit any degree-minimizing-paths.*

The SEMI-MATCHING problem consists of computing an optimal semi-matching in a bipartite graph and we abbreviate it by SM.

For subsets $A' \subseteq A, B' \subseteq B, E' \subseteq E$, we denote by $\mathrm{semi}(A', B', E')$ an optimal semi-matching in the graph $G' = (A', B', E')$, and we denote by $\mathrm{semi}_2(A', B', E')$ a semi-matching that does not admit degree-minimizing paths of length 2 in G'.

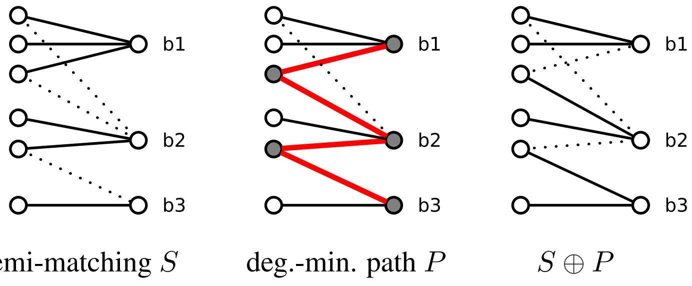


Figure 5.1: Illustration of a semi-matching S. P is a degree-minimizing path of length 4 starting at node b_1 and ending at node b_3. Initially, the degree of b_1 in S is 3 and the degree of b_3 in S is 1. Removing the edges $P \cup S$ from S and inserting the edges $P \setminus S$ into S decreases the degree of b_1 by 1 and increases the degree of b_3 by 1. Here, $S \oplus P$ is an optimal semi-matching.

Our algorithms for semi-matchings require the notion of *incomplete d-bounded semi-matchings*. These are semi-matchings that do not match all A-vertices and have a bounded maximal degree.

Definition 14 (Incomplete d-bounded Semi-Matching)**.** *Let d be an integer. Then an* incomplete d-bounded semi-matching *of G is a subset $S \subseteq E$ such that $\forall a \in A : \deg_S(a) \leq 1$ and $\forall b \in B : \deg_S(b) \leq d$.*

For subsets $A' \subseteq A, B' \subseteq B, E' \subseteq E$, we write $\mathrm{isemi}_d(A', B', E')$ to denote an incomplete d-bounded semi-matching of maximal size in the graph $G' = (A', B', E')$.

We say that an algorithm (or communication protocol) is a c-approximation algorithm (resp. communication protocol) to SM if it outputs a semi-matching S such that $\deg\max S \leq c \cdot \deg\max S^*$, where S^* denotes an optimal semi-matching. We note that this measure was previously used for approximating semi-matching, e.g, in [ANR95].

5.2 Streaming Algorithms

In this section, we describe an algorithm, $\textsc{asemi}(G, s, d, p)$ (Algorithm 12), that computes an incomplete $2dp$-bounded semi-matching in the graph G using space $\tilde{O}(s)$, and makes at most $p \geq 1$ passes over the input stream. If appropriate parameters are chosen, then the output is not only an incomplete semi-matching, but also a semi-matching. We run multiple copies of this algorithm with different parameters in parallel in order to obtain a one-pass algorithm for the semi-matching problem (Theorem 7). Using other parameters, we also obtain a $\log(n)$-pass algorithm, as stated in Theorem 8.

Algorithm 12 Skeleton for Approximating Semi-Matchings: $\textsc{asemi}(G, s, d, p)$

```
Require: $G = (A, B, E)$ is a bipartite graph
  $S \leftarrow \varnothing$
  repeat at most $p$ times or until $|A(S)| = |A|$
    $S \leftarrow S \cup \textsc{incomplete}(G|_{(A \setminus A(S)) \times B}, s, d)$ {requires one pass}
  end repeat
  return $S$
```

Algorithm 13 Computing Incomplete Semi-Matchings: $\textsc{incomplete}(G, s, d)$

```
Require: $G = (A, B, E)$ is a bipartite graph
  $k \leftarrow s/|A|$, $S_1 \leftarrow \varnothing$, $E' \leftarrow \varnothing$
  while $\exists$ an edge $ab$ in stream do
    if $ab \notin A \times B$ then continue
    if $\deg_{S_1}(a) = 0$ and $\deg_{S_1}(b) < d$ then $S_1 \leftarrow S_1 \cup \{ab\}$
    if $\deg_{E'}(a) < k$ then $E' \leftarrow E' \cup \{ab\}$
  end while
  $S_2 \leftarrow \text{isemi}_d(E'|_{(A \setminus A(S_1)) \times B})$
  $S \leftarrow S_1 \cup S_2$
  return $S$
```

$\textsc{asemi}(G, s, d, p)$ starts with an empty incomplete semi-matching S and adds edges to S by invoking $\textsc{incomplete}(G, s, d)$ (Algorithm 13) on the subgraph of the as yet unmatched A vertices in S and all B vertices. Each invocation of $\textsc{incomplete}(G, s, d)$ makes one pass over the input stream and returns a $2d$-bounded incomplete semi-matching while using space $\tilde{O}(s)$. Since we make at most p passes, the resulting incomplete semi-matching has a maximal degree of at most $2dp$.

$\textsc{incomplete}(G, s, d)$ collects edges greedily from graph G and puts them into an incomplete d-bounded semi-matching S_1 and a set E'. An edge e from the input stream is put into S_1 if $S_1 \cup \{e\}$ is still an incomplete d-bounded semi-matching. An edge $e = ab$ is added to E' if the degree of a in $E' \cup \{e\}$ is less or equal to a parameter k which is chosen to be $s/|A|$ in order to ensure that the algorithm does not exceed space $\tilde{O}(s)$. The algorithm returns an incomplete $2d$-bounded semi-matching that consists of S_1 and S_2, where S_2 is an optimal incomplete d-bounded semi-matching between the A vertices that are not matched in S_1 and all B vertices, using only edges in E'.

We lower-bound the size of S_2 in Lemma 20. We prove that for any bipartite graph $G = (A, B, E)$ and any $k > 0$, if we store for each $a \in A$ any $\max\{k, \deg_G(a)\}$ incident edges to a, then we can compute an incomplete d-bounded semi-matching of size at least $\min\{kd, |A|\}$ using only those edges, where d is an upper-bound on the maximal degree of an optimal semi-matching between A and B in G.

Lemma 20 is then used in the proof of Lemma 21, where we show a lower bound on the size of the output $S_1 \cup S_2$ of INCOMPLETE(G, s, d).

Lemma 20. *Let $G = (A, B, E)$ be a bipartite graph, let $d \geq \deg\max\mathrm{semi}(A, B, E)$ and let $k > 0$. Furthermore, let $E' \subseteq E$ be a subset of edges such that for all $a \in A : \deg_{E'}(a) = \min\{k, \deg_E(a)\}$. Then there is an incomplete d-bounded semi-matching $S \subseteq E'$ such that $|S| \geq \min\{kd, |A|\}$.*

Proof. Let $d^* = \deg\max\mathrm{semi}(A, B, E)$. We explicitly construct an incomplete semi-matching S. Let $A_0 \subseteq A$ such that for all $a \in A_0 : \deg_{E'}(a) = \deg_E(a)$, and let $A_1 = A \setminus A_0$. Let $S_0 = \mathrm{semi}(A_0, B, E)$. Clearly, $\deg\max S_0 \leq d^*$. We construct now S as follows.

Start with $S = S_0$, and then add greedily edges in any order from $E'|_{A_1 \times B}$ to S such that S remains an incomplete semi-matching with maximal degree d. Stop as soon as there is no further edge that can be added to S.

We prove that S contains at least $\min\{kd, |A|\}$ edges. To see this, either all nodes of A are matched in S, or there is at least one node $a \in A_1$ that is not matched in S (note that all nodes in A_0 are matched in S). Since $\deg_{E'}(a) = k$, all nodes $b \in \Gamma_{E'}(a)$ have degree d since otherwise a would have been added to S. This implies that there are at least $k \cdot d$ nodes matched in S which proves the lemma. □

Lemma 21. *Let $G = (A, B, E)$ be a bipartite graph, let $d \geq \deg\max\mathrm{semi}(A, B, E)$ and let $s \geq |A|$. Then INCOMPLETE(G, s, d) (Algorithm 13) uses $\tilde{O}(s)$ space and outputs an incomplete $2d$-bounded semi-matching S such that $|S| \geq \min\{|A|\frac{d}{d+d^*} + \frac{ds}{|A|}, |A|\}$.*

Proof. The proof refers to the variables of Algorithm 13 and the values they take at the end of the algorithm. Furthermore, let $S^* = \mathrm{semi}(A, B, E)$, $d^* = \deg\max S^*$, and let $A' = A \setminus A(S_1)$.

Firstly, we lower-bound $|S_1|$. Let $a \in A'$ and $b = S^*(a)$. Then $\deg_{S_1}(b) = d$ since otherwise a would have been matched in S_1. Hence, we obtain $|A(S_1)| \geq d|B(S^*(A'))| \geq d|A'|/d^*$, where the second inequality holds since the maximal degree in S^* is d^*. Furthermore, since $A' = A \setminus A(S_1)$ and $|S_1| = |A(S_1)|$, we obtain $|S_1| \geq |A|\frac{d}{d+d^*}$. We apply Lemma 20 on the graph induced by the edge set $E'|_{A' \times B}$. We obtain that $|S_2| \geq \min\{ds/|A|, |A'|\}$ and consequently $|S| = |S_1| + |S_2| \geq \min\{|A|\frac{d}{d+d^*} + \frac{ds}{|A|}, |A|\}$.

Concerning space, the dominating factor is the storage space for the at most $k + 1$ edges per A vertex, and hence space is bounded by $\tilde{O}(k|A|) = \tilde{O}(s)$. □

In the proof of Theorem 7, for $0 \leq \epsilon \leq 1$ we show that ASEMI$(G, n^{1+\epsilon}, n^{(1-\epsilon)/2}d', 1)$ returns a semi-matching if d' is at least the maximal degree of an optimal semi-matching. Using a standard technique, we run $\log(n) + 1$ copies of ASEMI for all $d' = 2^i$ with $0 \leq i \leq \log(n)$ and we return the best semi-matching, obtaining a 1-pass algorithm. We use the same idea in Theorem 8, where we obtain a $4\log(n)$ approximation algorithm that makes $\log(n)$ passes and uses space $\tilde{O}(n)$.

Theorem 7. *Let $G = (A, B, E)$ be a bipartite graph with $n = |A|$. For any $0 \leq \epsilon \leq 1$ there is a one-pass streaming algorithm using $\tilde{O}(n^{1+\epsilon})$ space that computes a $4n^{(1-\epsilon)/2}$ approximation to* SM.

Proof. We run $\log(n) + 1$ copies of Algorithm 12 in parallel as follows. For $0 \leq i \leq \lceil\log(n)\rceil$ let $S_i = \text{ASEMI}(G, n^{1+\epsilon}, n^{(1-\epsilon)/2}2^i, 1)$ and choose among the S_i a semi-matching S_k such that $|S_k| = n$ and for any other S_l with $|S_l| = n : \deg\max S_k \leq \deg\max S_l$.

We show now that there is a S_j which is a semi-matching that fulfills the desired approximation guarantee. Let $S^* = \text{semi}(A, B, E)$ and $d^* = \deg\max(S^*)$. Then define j to be such that $d^* \leq 2^j < 2d^*$ and let $d = n^{(1-\epsilon)/2}2^j$. S_j is the output of a call to $\text{INCOMPLETE}(G, n^{1+\epsilon}, d)$. By Lemma 21, S_j is of size at least $\min\{n\frac{d}{d+d^*} + dn^\epsilon, |A|\}$ which equals $|A|$ for our choice of d. This proves that all $a \in A$ are matched in S_j. By Lemma 21, $\deg\max S_j \leq 2d$ which is less or equal to $4n^{(1-\epsilon)/2}d^*$. Hence, S_j is a $4n^{(1-\epsilon)/2}$ approximation.

The space requirement is $\log n$ times the space requirement for the computation of a single S_i which is dominated by the space requirements of Algorithm 13. By Lemma 21, this is $\tilde{O}(n^{1+\epsilon})$, and hence the algorithm requires $\tilde{O}(n^{1+\epsilon}\log n) = \tilde{O}(n^{1+\epsilon})$ space. □

Theorem 8. *Let $G = (A, B, E)$ be a bipartite graph with $n = |A|$. There is a $\log(n)$-pass streaming algorithm using space $\tilde{O}(n)$ that computes a $4\log(n)$ approximation to* SM.

Proof. As in the proof of Theorem 7, we run $\log(n) + 1$ copies of Algorithm 12 in parallel. For $0 \leq i \leq \lceil\log(n)\rceil$ let $S_i = \text{ASEMI}(G, n, 2^i, \log(n))$ and choose among the S_i a semi-matching S_k such that $|S_k| = n$ and for any other S_l with $|S_l| = n : \deg\max S_k \leq \deg\max S_l$.

We show now that there is a S_j which is a semi-matching that fulfills the desired approximation guarantee. Let $S^* = \text{semi}(A, B, E)$ and $d^* = \deg\max(S^*)$. Then define j to be such that $d^* \leq 2^j < 2d^*$ and let $d = 2^j$. S_j is the output of a call to $\text{ASEMI}(G, n, d, \log(n))$. In each iteration, the algorithm calls $\text{INCOMPLETE}(G', n, d)$, where G' is the subgraph of G of the not yet matched A vertices and the B vertices. By Lemma 21, at least a $\frac{d}{d+d^*} \geq 1/2$ fraction of the unmatched A vertices is matched since $d \geq d^*$, and the maximal degree of the incomplete semi-matching returned by $\text{INCOMPLETE}(G', n, d)$ is at most $2d$. Hence, after $\log(n)$ iterations, all A vertices are matched. Since $d < 2d^*$ and the algorithm performs at most $\log(n)$ iterations, the algorithm returns a $4\log(n)$ approximation.

Each copy of Algorithm 12 uses space $\tilde{O}(n)$ and since we run $O(\log n)$ the required space is $\tilde{O}(n)$. □

5.3 One-Way Two-Party Communication

We now consider deterministic one-way two-party protocols which are given a bipartite graph $G = (A, B, E)$ as input, such that $E_1 \subseteq E$ is given to Alice and $E_2 \subseteq E$ is given to Bob. Alice sends a single message to Bob, and Bob outputs a valid semi-matching S for G. A central idea for our upper and lower bounds is what we call a *c-semi-matching skeleton* (or *c-skeleton*). Given a bipartite graph $G = (A, B, E)$, we define a c-semi-matching skeleton to be a subset of edges $S \subseteq E$ such that $\forall A' \subseteq A : \deg\max\mathrm{semi}(A', B, S) \leq c \cdot \deg\max\mathrm{semi}(A', B, E)$. We show how to construct an $O(\sqrt{n})$-skeleton of size n, and an $O(n^{1/3})$-skeleton of size $2n$. We show that if Alice sends a c-skeleton of her subgraph $G = (A, B, E_1)$ to Bob, then Bob can output a $c + 1$-approximation to the semi-matching problem. Using our skeletons, we thus obtain one-way two-party communication protocols for the semi-matching problem with approximation factors $O(\sqrt{n})$ and $O(n^{1/3})$, respectively (Theorem 9). Then we show that for any $\epsilon > 0$, an $O(n^{\frac{1}{(1+\epsilon)c+1}})$-skeleton requires at least cn edges. This renders our $O(\sqrt{n})$-skeleton and our $O(n^{1/3})$-skeleton tight up to a constant.

5.3.1 Upper Bounds

Firstly, we discuss the construction of two skeletons. In Lemma 24, we show that an optimal semi-matching is an $O(\sqrt{n})$-skeleton, and in Lemma 27, we show how to obtain a $O(n^{1/3})$-skeleton. In these constructions, we use the following key observation: Given a bipartite graph $G = (A, B, E)$, let $A' \subseteq A$ be such that A' has minimal expansion, meaning that $A' = \arg\min_{A'' \subseteq A} \frac{|\Gamma(A'')|}{|A''|}$. The maximal degree in a semi-matching is then clearly at least $\lceil \frac{|A'|}{|\Gamma(A')|} \rceil$ since all vertices of A' have to be matched to its neighborhood. However, it is also true that the maximal degree of a semi-matching *equals* $\lceil \frac{|A'|}{|\Gamma(A')|} \rceil$. This is a known fact that was used for instance in [GKK12] without proof, and also in [KR99]. For completeness, we are going to prove this fact in Lemma 23. This proof requires the following technical lemma, Lemma 22.

Lemma 22. *Let $G = (A, B, E)$ be a bipartite graph and let $A' \subseteq A$ such that $|\Gamma(A')| \leq |A'|$. Then:*

$$\forall A'' \subseteq A' : \frac{|\Gamma(A'')|}{|A''|} \geq \frac{|\Gamma(A')|}{|A'|} \Rightarrow \deg\max\mathrm{semi}(A', B, E) \leq \lceil \frac{|A'|}{|\Gamma(A')|} \rceil.$$

Proof. The proof is by contradiction. Let $d = \lceil \frac{|A'|}{|\Gamma(A')|} \rceil$, $S = \mathrm{semi}(A', B, E)$ and suppose that $\deg\max S \geq d + 1$. We construct now a set $\tilde{A} \subset A'$ such that $\frac{|\Gamma(\tilde{A})|}{|\tilde{A}|} < \frac{|\Gamma(A')|}{|A'|}$ contradicting the premise of the lemma.

To this end, we define two sequences $(A_i)_i$ with $A_i \subseteq A'$ and $(B_i)_i$ with $B_i \subseteq \Gamma(A')$. Let $b \in \Gamma(A')$ be a node with $\deg_S(b) \geq d + 1$ and let $B_1 = \{b\}$. We define

$$\begin{aligned} A_i &= \Gamma_S(B_i), \\ B_{i+1} &= \Gamma(A_i) \setminus \cup_{j \leq i} B_j. \end{aligned} \tag{5.1}$$

This setting is illustrated in Figure 5.2. Note that all A_i and all B_i are disjoint. Let k be such that $|A_k| > 0$ and $|A_{k+1}| = 0$. Then we set $\tilde{A} = \bigcup_{i=1}^{k} A_i$.

By construction of the sequence $(B_i)_i$, it is clear that for any $b' \in \cup B_i : \deg_S(b') \geq \deg_S(b) - 1$, since otherwise there is a degree-minimizing path from b to b' contradicting the definition of S. Then, by Equation 5.1, we obtain for all i that $|A_i| \geq |B_i|(\deg_S(b) - 1)$ which implies that $|A_i| \geq d|B_i|$ since $\deg_S(b) \geq d + 1$. Remind that $|A_1| \geq d + 1$. We compute

$$\frac{|\Gamma(\tilde{A})|}{|\tilde{A}|} = \frac{|B_1| + \sum_{2\leq i\leq k}|B_i|}{|A_1| + \sum_{2\leq i\leq k}|A_i|} \leq \frac{1 + \sum_{2\leq i\leq k}|B_i|}{(d+1) + \sum_{2\leq i\leq k}|B_i|d} < \frac{1}{d} \leq \frac{|\Gamma(A')|}{|A'|},$$

and we obtain a contradiction to the premise of the lemma. □

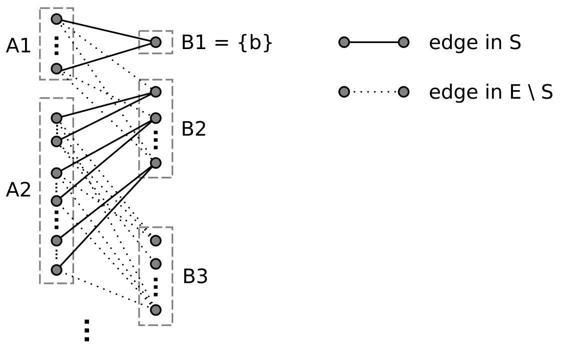


Figure 5.2: Illustration of the proof of Lemma 22. All nodes $b' \in \bigcup_{i\geq 2} B_i$ have $\deg_S(b') \geq \deg_S(b) - 1$ since otherwise there is a degree-minimizing path. To keep the figure simple, only those edges of $E\backslash S$ are drawn that connect the A_i to B_{i+1}. Note that in general there are also edges outside S from A_i to $\bigcup_{j<i} B_j$. However, there are no edges in the graph from A_i to $\bigcup_{j\geq i+2} B_j$.

Lemma 23. *Let $G = (A, B, E)$ with $|A| = n$, and let $d = \deg\max\mathrm{semi}(A, B, E)$. Let A' be a subset of A with minimal expansion α, that is*

$$A' = \arg\min_{A''\subseteq A} \frac{|\Gamma(A'')|}{|A''|},$$

and let $\alpha = \frac{|\Gamma(A')|}{|A'|}$. Then:

$$d = \lceil \alpha^{-1} \rceil.$$

Proof. We show that $d \geq \lceil \alpha^{-1} \rceil$ and $d \leq \lceil \alpha^{-1} \rceil$ separately.

1. $d \geq \lceil \alpha^{-1} \rceil$**:** The set A' has to be matched entirely to vertices in its neighborhood. Therefore, there is a node $b \in \Gamma(A')$ with degree at least $\lceil \frac{|A'|}{|\Gamma(A')|} \rceil = \lceil \alpha^{-1} \rceil$.

2. $d \leq \lceil \alpha^{-1} \rceil$**:** We construct a semi-matching explicitly with maximal degree d. Since an optimal semi-matching has at most this degree, the claim follows.

   Consider a decomposition of A into sets $A_1, A_2, \ldots$ as follows. $A_1 \subseteq A$ is a set with minimal expansion, and for $i > 1$, $A_i \subseteq A\backslash(\bigcup_{j<i} A_j)$ is the set with minimal expansion in $G|_{(A\backslash\bigcup_{j<i} A_j)\times(B\backslash\Gamma(\bigcup_{j<i} A_j))}$.

We construct a semi-matching $\tilde{S} = S_1 \cup S_2 \ldots$ as follows. Firstly, match A_1 to $\Gamma(A_1)$ in S_1. By Lemma 22, the maximal degree in S_1 is at most $\lceil \frac{|A_1|}{|\Gamma(A_1)|} \rceil = \lceil \alpha^{-1} \rceil$.

For a general S_i, we match A_i to vertices in $\Gamma(A_i) \setminus \Gamma(\bigcup_{j<i} A_j)$. By Lemma 22, the maximal degree in S_i is at most $\lceil \frac{|A_i|}{|\Gamma(A_i) \setminus \Gamma(\bigcup_{j<i} A_j)|} \rceil$.

This decomposition is illustrated in Figure 5.3.

Furthermore, it holds

$$\frac{|A_i|}{\Gamma(A_i) \setminus \Gamma(\bigcup_{j<i} A_j)|} \leq \frac{|A_{i+1}|}{\Gamma(A_{i+1}) \setminus \Gamma(\bigcup_{j<i+1} A_j)|},$$

since if this was not true, then the set $A_i \cup A_{i+1}$ would have smaller expansion in the graph $G|_{(A \setminus \bigcup_{j<i} A_j) \times (B \setminus \Gamma(\bigcup_{j<i} A_j))}$ than A_i. This implies that $\deg\max \tilde{S} = \deg\max S_1$ which in turn is $\lceil \alpha^{-1} \rceil$.

□

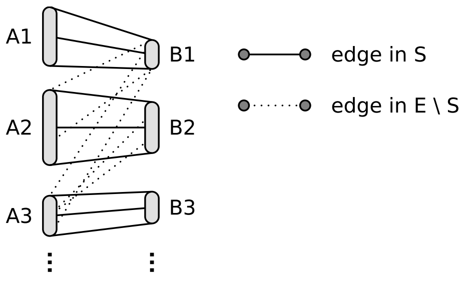


Figure 5.3: Illustration of the graph decomposition used in the proof of Lemma 23. Here, B_i is the set $\Gamma(A_i) \setminus \Gamma(\bigcup_{j<i} A_j)$. The neighborhood of A_i in G is a subset of $\bigcup_{j \leq i} B_i$. In S, however, A_i is matched entirely to vertices in B_i.

We prove now that an optimal semi-matching is a $O(\sqrt{n})$-skeleton.

Lemma 24. *Let $G = (A, B, E)$ with $n = |A|$, and let $S = \text{semi}(A, B, E)$. Then:*

$$\forall A' \subseteq A : \deg\max \text{semi}(A', B, S) < \sqrt{n}\,(\deg\max \text{semi}(A', B, E))^{1/2} + 1.$$

Proof. Let $A' \subseteq A$ be an arbitrary subset. Let $A'' = \arg\min_{A''' \subseteq A'} \frac{|\Gamma_S(A''')|}{|A'''|}$, and let $k = |\Gamma_S(A'')|$. Let $d = \deg\max \text{semi}(A', B, S)$. Then by Lemma 23, $d = \lceil \frac{|A''|}{k} \rceil$. Furthermore, since A'' is the set of minimal expansion in S, for all $b \in \Gamma_S(A'') : \deg_S(b) = d$, and hence $|A''| = kd$.

Let $d^* = \deg\max \text{semi}(A'', B, E)$. Then $d^* \leq \deg\max \text{semi}(A', B, E)$, since $A'' \subseteq A'$. It holds that $\forall x \in \Gamma_E(A'') \setminus \Gamma_S(A'') : \deg_S(x) \geq d - 1$ since otherwise there was a degree-minimizing path of length 2 in S. Figure 5.4 illustrates this setting. The sum of the degrees of the vertices in $\Gamma_E(A'')$ is upper-bounded by the number of A nodes. We obtain hence $(|\Gamma_E(A'')| - k)(d-1) + kd \leq n$, and this implies that $|\Gamma_E(A'')| \leq \frac{n-k}{d-1}$. Clearly, $d^* \geq |A''|/|\Gamma_E(A'')|$, and using the prior upper bound on $|\Gamma_E(A'')|$ and the equality $|A''| = kd$, we obtain $d^* \geq \frac{kd(d-1)}{n-k}$ which implies that $d < \sqrt{n}\sqrt{d^*} + 1$ for any $k \geq 1$.

□

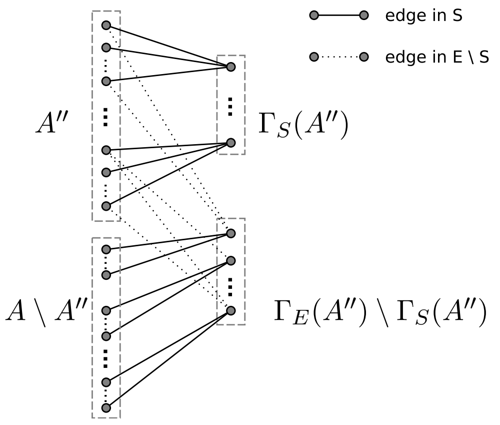


Figure 5.4: Illustration of the proof of Lemma 24. All nodes $b \in \Gamma_S(A'')$ have $\deg_S(b) = d$, and all nodes $b' \in \Gamma_E(A'') \setminus \Gamma_S(A'')$ have $\deg_S(b) \geq d - 1$.

In order to obtain an $O(n^{1/3})$-skeleton, for each $a \in A$ we add one edge to the $O(\sqrt{n})$-skeleton. Let $S = \text{semi}(A, B, E)$ be the $O(\sqrt{n})$-skeleton, let $B' = B(S)$ be the B nodes that are matched in the skeleton, and for all $b \in B'$ let $A_b = \Gamma_S(b)$ be the set of A nodes that are matched to b in S. Intuitively, in order to obtain a better skeleton, we have to increase the size of the neighborhood in the skeleton of all subsets of A, and in particular of the subsets A_b for $b \in B'$. We achieve this by adding additional optimal semi-matchings $S_b = \text{semi}(A_b, B, E)$ for all subsets A_b with $b \in B'$ to S, see Lemma 27. We firstly prove a technical lemma, Lemma 25, that points out an important property of the interplay between the matchings S and the matchings S_b for $b \in B'$. Then, we state in Lemma 26 an inequality that is an immediate consequence of Hölder's inequality. Lemma 26 is then used in the proof of Lemma 27, which proves that our construction is an $O(n^{1/3})$-skeleton.

Lemma 25. *Let $G = (A, B, E)$, $A' \subseteq A$, $A'' \subseteq A'$, and let $S = \text{semi}(A', B, E)$. Furthermore, let $\Gamma_S(A') = \{b_1, \dots, b_k\}$, and $\forall b_i \in \Gamma_S(A')$: let $A'_i = \Gamma_S(b_i) \cap A'$, and $A''_i = \Gamma_S(b_i) \cap A''$. Then:*

$$\deg \max \text{semi}(A'', B, E)^{-1} \sum_{i: b_i \in \Gamma_S(A'')} |A''_i|(|A'_i| - 1) \leq |A'|.$$

Proof. Let $S'' = \mathrm{semi}(A'', B, E)$, and denote $d = \deg\max S''$. Clearly,

$$\sum_{b'' \in B(S'')} \deg_S(b'') \leq |A'|. \tag{5.2}$$

Consider any $b'' \in B(S'')$. We bound $\deg_S(b'')$ from above as follows

$$\deg_S(b'') \quad \geq \quad \max\{|A'_i| - 1 \,:\, \exists a \in A''_i \text{ with } b'' \in \Gamma_E(a)\}. \tag{5.3}$$

Let j be such that $|A'_j| - 1$ poses the maximum of the set in the right hand side of Inequality 5.3. Note that if Inequality 5.3 was not true, then there would be a length two degree minimizing path in S connecting b'' and b_j. The setup up visualized in Figure 5.5. We bound now the right hand side of Inequality 5.3 as follows

$$\begin{aligned}(|A'_j| - 1) = \max\{|A'_i| - 1 \quad &: \quad \exists a \in A''_i \text{ with } b'' \in \Gamma_E(a)\} \\ &\geq \quad \sum_{a \in \Gamma_{S''}(b'')} \frac{1}{\deg_{S''}(b'')}(|A'_{B(S(a))}| - 1).\end{aligned} \tag{5.4}$$

We used here that $|A'_{B(S(a))}| \leq |A'_j|$ for any $a \in \Gamma_{S''}(b'')$, and $|a \in \Gamma_{S''}(b'')| = \deg_{S''}(b'')$. Since $d = \deg\max S''$, and using Inequalities 5.3 and 5.4 we obtain

$$\deg_S(b'') \geq \sum_{a \in \Gamma_{S''}(b'')} \frac{1}{d}(|A'_{B(S(a))}| - 1). \tag{5.5}$$

We combine Inequalities 5.2 and 5.5, and the result follows

$$\begin{aligned}|A'| \geq \sum_{b'' \in B(S'')} \deg_S(b'') \quad &\geq \quad \sum_{b'' \in B(S'')} \sum_{a \in \Gamma_{S''}(b'')} \frac{1}{d}(|A'_{B(S(a))}| - 1) \\ &= \quad \frac{1}{d} \sum_{A''_i} |A''_i||A'_i - 1|.\end{aligned}$$

□

In the proof of Lemma 27, we also need the following inequality.

Lemma 26. *Let $x_1, \ldots, x_k \geq 0$, and let $p > 0$ be an integer. Then:*

$$\frac{(\sum_{i=1}^k x_i)^p}{k^{p-1}} \leq \sum_{i=1}^k x_i^p.$$

Proof. This is an immediate consequence of Hölder's inequality:

$$\sum_{i=1}^k x_i \leq (\sum_{i=1}^k x_i^p)^{1/p} k^{\frac{p-1}{p}}.$$

□

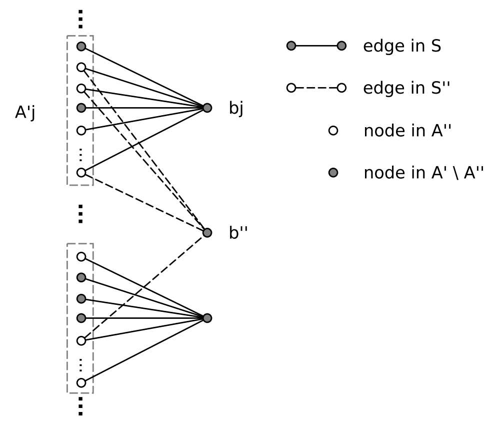


Figure 5.5: Illustration of the proof of Lemma 25. The degree of b'' in S is at least $|A'_j| - 1$. Otherwise there would be a length two degree-minimizing path between b'' and b_j.

Lemma 27. *Let $G = (A, B, E)$ be a bipartite graph with $n = |A|$. Let $S = \mathrm{semi}(A, B, E)$, and for all $b \in B(S) : S_b = \mathrm{semi}(\Gamma_S(b), B, E)$. Then:*

$$\forall A' \subseteq A : \deg\max\mathrm{semi}(A', B, S \cup \bigcup_{b \in B(S)} S_b) \leq \lceil 2n^{1/3} \deg\max\mathrm{semi}(A', B, E)\rceil.$$

Proof. Let $A' \subseteq A$. Let $\tilde{S} = S \cup \bigcup_{b \in B(S)} S_b$. Let $A'' = \arg\min_{A''' \subseteq A'} \frac{|\Gamma_{\tilde{S}}(A''')|}{|A'''|}$ and let $k = |\Gamma_{\tilde{S}}(A'')|$. From Lemma 23 it follows that $\deg\max\mathrm{semi}(A', B, \tilde{S}) = \lceil \frac{|A''|}{k} \rceil$. Furthermore, let $d = \deg\max\mathrm{semi}(A'', B, E)$. For a node $b \in \Gamma_{\tilde{S}}(A'')$, let $A''_b = \{a \in A : \tilde{S}(a) = b\}$. For two nodes $b_i, b_j \in \Gamma_{\tilde{S}}(A'')$, let $A''_{b_i,b_j} = \{a \in A'' : S(a) = b_i, S_{b_i}(a) = b_j\}$.

We consider the cases $k \geq n^{1/3}$ and $k < n^{1/3}$ separately.

1. $k \geq n^{1/3}$. Consider the semi-matching S. From Lemma 25 we obtain the condition

$$1/d \sum_{i=1}^{k} |A''_i|(A_i - 1) \leq n,$$

and since $A''_i \leq A_i$ we obtain from the prior Inequality that

$$1/d \sum_{i=1}^{k} (|A''_i| - 1)^2 < n.$$

Using $\sum_{i=1}^{k} |A''_i| = |A''|$ and Lemma 26, we obtain

$$\begin{aligned} \frac{1}{d}\frac{1}{k}(|A''| - k)^2 &< n, \quad \Rightarrow \\ |A''| &< \sqrt{ndk} + k. \end{aligned} \tag{5.6}$$

Then, since $\deg \max \mathrm{semi}(A'', B, \tilde{S}) = \lceil \frac{|A''|}{k} \rceil$, we obtain from Inequality 5.6

$$\deg \max \mathrm{semi}(A'', B, \tilde{S}) \leq \lceil \frac{\sqrt{nd}}{\sqrt{k}} \rceil + 1.$$

Since $k \geq n^{1/3}$, we conclude that

$$\deg \max \mathrm{semi}(A'', B, \tilde{S}) \leq n^{1/3}\sqrt{d} + 2.$$

2. $k < n^{1/3}$. We consider here the two subcases $|A''| < 2dk^2$ and $|A''| \geq 2dk^2$.

   (a) $|A''| < 2dk^2$. Then since $\deg \max \mathrm{semi}(A'', B, \tilde{S}) = \lceil \frac{|A''|}{k} \rceil$, we conclude that

   $$\deg \max \mathrm{semi}(A'', B, \tilde{S}) \leq \lceil 2dk \rceil < \lceil 2dn^{1/3} \rceil.$$

   (b) $|A''| \geq 2dk^2$. Let $b \in B(S)$ and consider the semi-matching S_b matching A''_b to B. From Lemma 25 and the fact that $A''_{b,b_i} \subseteq A'_{b,b_i}$ we obtain

   $$\begin{aligned} \frac{1}{d}\sum_{i=1}^{k} |A''_{b,b_i}|(|A''_{b,b_i}| - 1) &\leq |A_b|, \\ \left(\frac{1}{d}\sum_{i=1}^{k} |A''_{b,b_i}|^2\right) - \frac{1}{d}|A''_b| &\leq |A_b|. \end{aligned}$$

   By Lemma 26, we obtain

   $$\frac{1}{dk}|A''_b|^2 - \frac{1}{d}|A''_b| \leq |A_b|. \tag{5.7}$$

   Consider now the semi-matching S. From Lemma 25 we obtain the condition

   $$\frac{1}{d}\sum_{i=1}^{k} |A''_i|(|A_i| - 1) \leq n. \tag{5.8}$$

   Using Inequality 5.7 in Inequality 5.8 and simplifying, we obtain

   $$\begin{aligned} \frac{1}{d}\sum_{i=1}^{k} |A''_i| \left((\frac{1}{dk}|A''_i|^2 - \frac{1}{d}|A''_i|) - 1 \right) &\leq n, \\ \frac{1}{d^2k}\sum_{i=1}^{k} |A''_i|^3 - \sum_{i=1}^{k} \frac{1}{d^2}|A''_i|^2 - \sum_{i=1}^{k} \frac{1}{d}|A''_i| &\leq n, \\ \frac{1}{d^2k^3}|A''|^3 - \underbrace{\frac{1}{d^2k}|A''|^2}_{I} - \underbrace{\frac{1}{d}|A''|}_{II} &\leq n. \end{aligned} \tag{5.9}$$

Since $|A''| \geq 2dk^2$, we can upper bound the terms I and II from Inequality 5.9 as follows

$$\frac{1}{2d^3k^3}|A''|^3 \geq I, \text{ and} \tag{5.10}$$

$$\frac{1}{4d^3k^4}|A''|^3 \geq II. \tag{5.11}$$

Using bounds 5.10 and 5.11 in Inequality 5.9 and simplifying, we obtain

$$\frac{1}{4d^2k^3}|A''|^3 < n, \Rightarrow$$

$$|A''| < 2^{2/3}n^{1/3}d^{2/3}k. \tag{5.12}$$

Since $\deg\max\text{semi}(A'', B, \tilde{S}) = \lceil \frac{|A''|}{k} \rceil$, and using Inequality 5.12, we conclude that

$$\deg\max\text{semi}(A'', B, \tilde{S}) \leq \lceil 2^{2/3}n^{1/3}d^{2/3} \rceil.$$

Combining the bounds from cases 1, 2a and 2b, the result follows. □

We mention that straightforwardly extending this idea does not lead to better skeletons. There are graphs for which adding further semi-matchings $S_{b_1b_2} = \text{semi}(A_{b_1b_2}, B, E)$ to our $O(n^{1/3})$-skeleton, where $A_{b_1b_2}$ is the set of A vertices whose neighborhood in our $O(n^{1/3})$-skeleton is the set $\{b_1, b_2\}$, does not help to improve the quality of the skeleton. Before stating our main theorem, Theorem 9, we show in Lemma 28 that if Alice sends a c-matching skeleton, then Bob can compute a $c+1$ approximation. Then, we state our main theorem.

Lemma 28. *Let $G = (A, B, E)$ be a bipartite graph and let E_1, E_2 be a partition of the edge set E. Furthermore, let $E_1' \subseteq E_1$ such that for any $A' \subseteq A(E_1)$:*

$$\deg\max\text{semi}(A(E_1), B, E_1') \leq c \deg\max\text{semi}(A(E_1), B, E_1').$$

Then:

$$\deg\max\text{semi}(A, B, E_1' \cup E_2) \leq (c+1)\deg\max\text{semi}(A, B, E).$$

Proof. We construct a semi-matching S between A and B with edges from $E_1' \cup E_2$ explicitly and we show that $\deg\max S \leq (c+1)\deg\max\text{semi}(A, B, E)$. Since $\deg\max\text{semi}(A, B, E_1' \cup E_2) \leq \deg\max S$, the result then follows.

Let $S_2 = \text{semi}(A, B, E) \cap E_2$, and let $S_1 = \text{semi}(A \setminus A(S_2), B, E_1)$. Then $S = S_1 \cup S_2$. Clearly, $\deg\max S_2 \leq \deg\max\text{semi}(A, B, E)$. Furthermore, by the premise of the lemma we obtain $\deg\max S_1 \leq c\deg\max\text{semi}(A, B, E)$. Since $\deg\max S \leq \deg\max S_1 + \deg\max S_2$ and $\deg\max S_1 + \deg\max S_2 \leq (c+1)\deg\max(A, B, E)$ the result follows. □

Theorem 9. *Let $G = (A, B, E)$ with $n = |A|$ and $m = |B|$. Then there are one-way two party deterministic communication protocols for the semi-matching problem, one with*

1. *message size $cn\log m$ and approximation factor $n^{1/2} + 2$, and another one with*

2. *message size* $2cn \log m$ *and approximation factor* $2n^{1/3} + 2$.

Proof. Alice computes the skeletons as in Lemma 24 or in Lemma 27 and sends them to Bob. Bob computes an optimal semi-matching considering his edges and the edges received from Alice. By Lemma 28 the results follow. □

5.3.2 Lower Bounds

5.3.2.1 Lower Bounds for Semi-matching-skeletons

We present now a lower bound that shows that the skeletons of the previous subsection are essentially optimal. For an integer c, we consider the complete bipartite graph $K_{n,m}$ where m is a carefully chosen value depending on c and n. We show in Lemma 29 that for any subset of edges E' of $K_{n,m}$ such that for all $a \in A : \deg_{E'}(a) \le c$, there is a subset $A' \subseteq A$ with $|A'| \le m$ such that an optimal semi-matching that matches A' using edges in E' has a maximal degree of $\Omega(n^{\frac{1}{c+1}})$. Note that since $|A'| \le m$, there is a matching in $K_{n,m}$ that matches all A' vertices. This implies that such an E' is only an $\Omega(n^{\frac{1}{c+1}})$-skeleton.

Lemma 29. *Let* $G = (A, B, E)$ *be the complete bipartite graph with* $|A| = n$ *and* $|B| = (c!)^{\frac{1}{c+1}} n^{\frac{1}{c+1}}$ *for an integer* c. *Let* $E' \subseteq E$ *be an arbitrary subset such that* $\forall a \in A : \deg_{E'}(a) \le c$. *Then there exists an* $A' \subseteq A$ *with* $|A'| \le |B|$ *and*

$$\deg \max \mathrm{semi}(A', B, E') \ge \frac{(c!)^{\frac{1}{c+1}}}{c} n^{\frac{1}{c+1}} > e^{-1.3} n^{\frac{1}{c+1}}. \tag{5.13}$$

Proof. Let $E' \subseteq E$ be as in the statement of the lemma. Let E'' be an arbitrary superset of E' such that $\forall a \in A : \deg_{E''}(a) = c$. Since $\deg \max \mathrm{semi}(A', B, E'') \le \deg \max \mathrm{semi}(A', B, E')$ it is enough to show the lemma for E''. Denote by $A_{\{i_1,\dots,i_c\}}$ the subset of A such that $\forall a \in A_{\{i_1,\dots,i_c\}} : \Gamma_{E''}(a) = \{b_{i_1}, \dots, b_{i_c}\}$. Then

$$|A| = \sum_{\substack{A_i : i = \{i_1,\dots,i_c\} \text{ and} \\ \{b_{i_1},\dots,b_{i_c}\} \text{ is a } c\text{-subset of } B}} |A_i|, \tag{5.14}$$

since $\forall a \in A : \deg_{E''}(a) = c$. Suppose for the sake of a contradiction that Inequality 5.13 is not true. Then for all A_i on the right side of Inequality 5.14 we have $|A_i| < (c!)^{\frac{1}{c+1}} n^{\frac{1}{c+1}}$. There are at most $\binom{|B|}{c}$ such sets. This implies that:

$$|A| \le \binom{|B|}{c} \cdot (c!)^{\frac{1}{c+1}} n^{\frac{1}{c+1}} < \frac{|B|^c}{c!} (c!)^{\frac{1}{c+1}} n^{\frac{1}{c+1}} < \frac{(c!)^{\frac{c}{c+1}} n^{\frac{c}{c+1}}}{c!} (c!)^{\frac{1}{c+1}} n^{\frac{1}{c+1}} = n.$$

This is a contradiction to the fact that $|A| \ge n$ and proves the first inequality in Inequality 5.13. To proof the second, we apply Stirling's formula, and we obtain

$$\frac{(c!)^{\frac{1}{c+1}}}{c} > \frac{(\sqrt{2\pi} c^{c+1/2} e^{-c})^{\frac{1}{c+1}}}{c} = e^{\frac{1/2 \ln(2\pi) - 1/2 \ln(c) - c}{c+1}}.$$

It can be shown that for any $c > 0$, $\frac{1/2\ln(2\pi)-1/2\ln(c)-c}{c+1} > -1.3$ which proves the result. $\square$

We extend Lemma 29 now to edge sets of bounded cardinality without restriction on the maximal degree of an A node, and we state then our lower-bound result in Theorem 10.

Lemma 30. *Let $c > 0$ be an integer, let $\epsilon > 0$ be a constant, and let $c' = (1+\epsilon)c$. Let $G = (A, B, E)$ be the complete bipartite graph with $|A| = n$ and $|B| = (c'!)^{\frac{1}{c'+1}}(\frac{\epsilon}{1+\epsilon} \cdot n)^{\frac{1}{c'+1}}$. Let $E' \subseteq E$ be an arbitrary subset of size at most $c \cdot n$. Then there exists an $A' \subseteq A$ with $|A'| \leq |B|$ and*

$$\deg\max\mathrm{semi}(A', B, E') > e^{-1.3}(\frac{\epsilon}{1+\epsilon}n)^{\frac{1}{c'+1}}. \tag{5.15}$$

Proof. Split A into $A_>$ and $A_\leq$ such that for all $a \in A_>$: $\deg_{S'}(a) > c'$, and for all $a \in A_\leq$: $\deg_{S'}(a) \leq c'$. Then $|A_>|c' + |A_\leq| \leq cn$ which implies that $|A_\leq| \geq \frac{\epsilon}{1+\epsilon}n$. Let $G' = G|_{A_\leq \times B}$. Then by Lemma 29 applied on G' there is a subset $A' \subseteq A_\leq$ with $|A'| \leq |B|$ such that

$$\deg\max\mathrm{semi}(A', B, E'|_{A_\leq \times B}) > e^{-1.3}|A_\leq|^{\frac{1}{c'+1}},$$

and since $\deg\max\mathrm{semi}(A', B, E'|_{A_\leq \times B}) = \deg\max\mathrm{semi}(A', B, E')$, the result follows. $\square$

Theorem 10. *Let $c > 0$ be an integer. Then for all $\epsilon > 0$, an $O(n^{\frac{1}{(1+\epsilon)c+1}})$-semi-matching skeleton requires at least cn edges.*

5.3.2.2 One-way Two-party Communication Lower Bound

To prove a lower bound on the deterministic communication complexity we define a family of bipartite graphs. For given integers n and m, let $\mathcal{G}_1 = \{G_1(x) | x \in \{0,1\}^{n \times m}\}$ be defined as follows. Let $B_0 = \{b_1^0, \dots, b_m^0\}$, $B_1 = \{b_1^1, \dots, b_m^1\}$ and $A = \{a_1, \dots, a_n\}$. Given $x \in \{0,1\}^{n\times m}$, let $E_x = \{(a_i, b_j^{x_{i,j}}) \,|\, 1 \leq i \leq n, 1 \leq j \leq m\}$ (i.e, the entries of the matrix x determine if there is an edge (a_i, b_j^0) or an edge (a_i, b_j^1) for all i, j). Then, we define $G_1(x) = (A, B_0 \cup B_1, E_x)$. From Lemma 30 we immediately obtain the following lemma.

Lemma 31. *Let $c > 0$ be an integer, let $\epsilon > 0$ be a constant, and let $c' = (1+\epsilon)c$. Let n be a sufficiently large integer, and let $m = (c'!)^{\frac{1}{c'+1}}(\frac{\epsilon}{1+\epsilon} \cdot n)^{\frac{1}{c'+1}}$. Let $G = (A, B_0 \cup B_1, E)$ be a graph $G \in \mathcal{G}_1$, and let $E' \subseteq E$ be such that $|E'| \leq cn$. Then there exists a set of nodes $A' \subseteq A$ with $|A'| \leq m$ and* $\deg\max\mathrm{semi}(A', B_0 \cup B_1, E') > 1/2e^{-1.3}(\frac{\epsilon}{1+\epsilon}n)^{\frac{1}{c'+1}}$.

We further define a second family of bipartite graphs $\mathcal{G}_2$ on the sets of nodes A and C, $|A| = |C| = n$. For a set $A' \subseteq A$ we define the graph $G_2(A')$ to be an arbitrary matching from all the nodes of A' to nodes of C. The family of graphs $\mathcal{G}_2$ is defined as $\mathcal{G}_2 = \{G_2(A') | A' \subseteq A\}$.

Our lower bound will be proved using a family of graphs $\mathcal{G}$. Slightly abusing notation, the family of graphs $\mathcal{G}$ is defined as $\mathcal{G} = \mathcal{G}_1 \times \mathcal{G}_2$. That is, the graphs in $\mathcal{G}$ are all graphs

$G = (A, B_0 \cup B_1 \cup C, E_1 \cup E_2)$ built from a graph $G_1 = (A, B_0 \cup B_1, E_1) \in \mathcal{G}_1$ and a graph $G_2 = (A, C, E_1) \in \mathcal{G}_2$ where the set of nodes A is the same for G_1 and G_2. We now prove our lower bound.

Theorem 11. *Let $c > 0$ be an integer and let $\epsilon > 0$ be an arbitrarily small constant. Let $\mathcal{P}$ be a β-approximation one-way two-party protocol for semi matching that has communication complexity at most α. If $\beta \leq \gamma = 1/2\frac{1}{e^{1.3}}(\frac{\epsilon}{\epsilon+1}n)^{\frac{1}{(1+\epsilon)c+1}}$, then $\alpha > cn$, where n is the number of nodes to be matched.*

Proof. Take n sufficiently large. Let $c' = (1+\epsilon)c$ and let $m = (c'!)^{\frac{1}{c'+1}}(\frac{\epsilon}{1+\epsilon} \cdot n)^{\frac{1}{c'+1}}$. We consider as possible inputs the graphs in $\mathcal{G}$ (for n and m). Given an input graph, Alice will get as input all edges between A and $B_0 \cup B_1$ (i.e., a graph in $\mathcal{G}_1$) and Bob will get all edges between A and C (i.e., a graph in $\mathcal{G}_2$)

Assume towards a contradiction that the communication complexity of $\mathcal{P}$ is at most cn. Then there is a set of graphs $\mathcal{G}^* \subseteq \mathcal{G}_1$, $|\mathcal{G}^*| \geq 2^{nm-cn}$, such that on all graphs in $\mathcal{G}^*$ Alice sends the same message to Bob. Consider the set $X^* \subseteq \{0,1\}^{n \times m}$ such that $\mathcal{G}^* = \{G_1(x) \mid x \in X^*\}$, Since there is a one-to-one correspondence between $\mathcal{G}^*$ and X^*, $|X^*| \geq 2^{nm-cn}$, and there are at most cn entries which are constant over all matrices in X^*, otherwise $|X^*| < 2^{nm-cn}$. This means that there are at most cn edges that exist in all graphs in $\mathcal{G}^*$. Let E' be the set of all these edges.

Consider now the graph $G = (A, B_0 \cup B_1, E')$. Since $|E'| \leq cn$, by Lemma 31 there exists a set $A' \subseteq A$ with $|A'| \leq m$ and $\deg \max \text{semi}(A', B_0 \cup B_1, E') > \gamma$. We now define $G_2^* \in \mathcal{G}_2$ to be $G_2^* = G_2(A \setminus A')$.

Now observe that on any of $G \in \mathcal{G}^* \times \{G_2^*\} \subseteq \mathcal{G}$, $\mathcal{P}$ gives the same output semi-matching S. S can include, as edges matching the nodes in A', only edges from E', since for any other edge there exists an input in $\mathcal{G}^* \times \{G_2^*\}$ in which that edge does not exist and $\mathcal{P}$ would not be correct on that input. It follows (by Lemma 31) that the maximum degree of S is greater than γ. On the other hand, since $|A'| \leq m$, there is a perfect matching in any graph in $\mathcal{G}^* \times \{G_2^*\}$. The approximation ratio of $\mathcal{P}$ is therefore greater than γ. A contradiction. □

5.4 Structure Theorem

We now present our results concerning the structure of semi-matchings. Firstly, we show in Lemma 32 that a semi-matching that does not admit length 2 degree-minimizing paths can be decomposed into maximal matchings. In Lemma 33, we show that if a semi-matching does not admit *any* degree-minimizing paths, then there is a similar decomposition into maximum matchings.

Lemma 32 is then used to prove that semi-matchings that do not admit length 2 degree-minimizing paths approximate optimal semi-matchings within a factor $\lceil \log(n+1) \rceil$. To this end, we firstly show in Lemma 34 that the first d^* maximal matchings of the decomposition of such a semi-matching match at least $1/2$ of the A vertices, where d^* is the maximal degree of an optimal semi-matching. In Theorem 12, we then apply this result $\lceil \log(n+1) \rceil$ times, showing that the maximal degree of a semi-matching that does not admit length 2 degree-minimizing paths is at most $\lceil \log(n+1) \rceil$ times the maximal degree of an optimal semi-matching.

Lemma 32. *Let $S = \mathrm{semi}_2(A, B, E)$ be a semi-matching in G that does not admit a length 2 degree-minimizing path, and let $d = \deg\max S$. Then S can be partitioned into d matchings $M_1, \ldots, M_d$ such that*

$$\forall i : M_i \textit{ is a maximal matching in } G|_{A_i \times B_i},$$

where $A_1 = A$, $B_1 = B$, and for $i > 1 : A_i = A \setminus \bigcup_{1 \le j < i} A(M_j)$ and $B_i = B(M_{j-1})$.

Proof. The matchings $M_1, \ldots, M_d$ can be obtained as follows. For each $b \in B(S)$, label its incident edges in S by $1, 2, \ldots, \deg_S(b)$ arbitrarily. Matching M_i is then the subset of edges of S that are labeled by i.

We prove the statement by contradiction. Let i be the smallest index such that M_i is not maximal in $G|_{A_i \times B_i}$. Then there exists an edge $e = ab \in E$ with $a \in A_i$ and $b \in B_i$ such that $M_i \cup \{e\}$ is a matching in $G|_{A_i \times B_i}$. Note that $\deg_S(b) < i$ since b is not matched in M_i. Consider now the edge $e' \in S$ matching the node a to b' in S. Since $a \in A_i$ and a is not matched in M_i, e' is in a matching M_j with $j > i$ and hence $\deg_S(b') \ge j > i$. Then $P = (b', a, b)$ is a length 2 degree-minimizing path since $\deg_S(b') > i$ and $\deg_S(b) < i$ contradicting our assumption. □

Lemma 33. *Let $S^* = \mathrm{semi}(A, B, E)$ be a semi-matching in G that does not admit degree-minimizing paths of any length, and let $d^* = \deg\max S^*$. Then S^* can be partitioned into d^* matchings $M_1, \ldots, M_{d^*}$ such that*

$$\forall i : M_i \textit{ is a maximum matching in } G|_{A_i \times B_i},$$

where $A_1 = A$, $B_1 = B$, and for $i > 1 : A_i = A \setminus \bigcup_{1 \le j < i} A(M_j)$ and $B_i = B(M_{j-1})$.

Proof. The proof is similar to the proof of Lemma 32. The matchings $M_1, \ldots, M_{d^*}$ can be obtained as follows. For each $b \in B(S)$, label its incident edges in S by $1, 2, \ldots, \deg_{S^*}(b)$ arbitrarily. Matching M_i is then the subset of edges of S that are labeled by i.

We prove the statement by contradiction. Let i be the smallest index such that M_i is not a maximum matching in $G|_{A_i \times B_i}$. Then there exists an augmenting path $A = (a_1, b_1, \ldots a_l, b_l)$ such that for all $j < l : (a_{j+1}, b_j) \in M_i$ and $\forall i : (a_i, b_i) \notin M_i$. Let b' be the match of a_1 in S^*. Since $a_1 \in A_l$, $\deg_{S^*}(b') > i$. Since $b_l \in B_i$ and b_l is not matched in M_i^*, $\deg_{S^*}(b_l) < i$. Then $P = (b', a_1, b_1, \ldots, a_l, b_l)$ is a degree-minimizing path contradicting our assumption. □

We firstly prove a lemma that is required in the proof of Theorem 12.

Lemma 34. *Let $A' \subseteq A$, let $S = \mathrm{semi}_2(A', B, E)$ be a semi-matching in $G|_{A' \times B}$ that does not admit length 2 degree-minimizing paths and let $S^* = \mathrm{semi}(A', B, E)$ be an optimal semi-matching in $G|_{A' \times B}$. Then $\exists A'' \subseteq A'$ with $|A''| \ge 1/2|A'|$ such that*

1. *$\deg\max S|_{A'' \times B} \le \deg\max S^*$,*
2. *$S|_{A' \setminus A'' \times B}$ is a semi-matching of $G|_{A' \setminus A'' \times B}$ and it does not admit length 2 degree-minimizing paths.*

Proof. Let $d = \deg\max S$ and let $d^* = \deg\max S^*$. Partition S into matchings $M_1, \dots, M_d$ as in Lemma 32. We will show that $A'' = \bigcup_{i \le d^*} A(M_i)$ fulfills Item 1 and Item 2 of the Lemma.

We have to show that $|A''| \ge 1/2|A'|$. Let $A''' = A' \setminus A''$ and let $(a, b) \in S^*$ be an edge such that $a \in A'''$. We argue now, that $\deg_S(b) \ge d^*$.

Suppose for the sake of a contradiction that $\deg_S(b) < d^*$. Then (a, b) could have been added to some matching M_j with $j \le d^*$. Since by Lemma 32 all M_i are maximal, we obtain a contradiction and this proves that $\deg_S(b) \ge d^*$.

This implies further that $|A''| \ge d^* \cdot |B(S^*|_{A''' \times B})| \ge d^* \cdot |A'''|/d^* = |A'''|$, where the last inequality comes from the fact that a node $b \in B(S^*|_{A''' \times B})$ has at most d^* edges incident in S^*. Since A''' and A'' form a partition of A', we obtain $|A''| \ge 1/2|A'|$.

Since $A'' = A(S|_{A'' \times B})$ and $S|_{A'' \times B}$ is a set of d^* matchings, Item 1 is trivially true. Concerning Item 2, note that if $S|_{A' \setminus A'' \times B}$ admitted a length 2 degree-minimizing path, then that path would also be a degree-minimizing path in S contradicting the premise that S does not admit a length 2 degree-minimizing path. □

Theorem 12. *Let $S = \mathrm{semi}_2(A, B, E)$ be a semi-matching of G that does not admit a length 2 degree-minimizing path. Let S^* be an optimal semi-matching in G. Then:*

$$\deg\max S \le \lceil \log(n+1) \rceil \deg\max S^*.$$

Proof. We construct a sequence of vertex sets (A_i) and a sequence of semi-matchings (S_i) as follows. Let $A_1 = A$, and let $S_1 = S$. For any i, S_i will be a semi-matching in the graph $G|_{A_i \times B}$ and it will not admit length 2 degree-minimizing paths.

We construct A_{i+1} and S_{i+1} from A_i and S_i as follows. By Item 1 of Lemma 34, there is a subset $A'_i \subseteq A_i$ of size at least $1/2|A_i|$ such that $S_i|_{A'_i \times B}$ has maximal degree d^*. Let $A_{i+1} = A_i \setminus A'_i$, and let $S_{i+1} = S_i|_{A_{i+1} \times B}$. By Item 2 of Lemma 34, S_{i+1} does not comprise length 2 degree-minimizing paths in the graph $G|_{A_{i+1} \times B}$. We stop this construction at iteration l when $A'_l = A_l$ occurs.

Note that $S = \bigcup_i S_i|_{A'_i \times B}$ and hence $\deg\max S \le \sum_{i=1}^{l} \deg\max S_i|_{A'_i \times B} \le l \cdot d^*$. It remains to argue that $l \le \log(n) + 1$. Since $|A'_i| \ge 1/2|A_i|$ and $A_{i+1} = A_i \setminus A'_i$, we have $|A_{i+1}| \le 1/2|A_i|$. Since $|A_1| = n$, we have $|A_i| \le (\frac{1}{2})^{i-1} n$. Then, $|A_{\lceil \log(n+1) \rceil}| < 1$ which implies that $|A_{\lceil \log(n+1) \rceil}| = 0$. We obtain hence $l \le \lceil \log(n+1) \rceil$, which proves the theorem. □

Chapter 6

Validating XML Documents in the Streaming Model

In this chapter, we present our results on the problem of validating XML documents. Let N denote the length of an XML document. We start our presentation with preliminaries in Section 6.1 where we define XML documents and DTDs, and we also state the DTD-validity problem of XML documents formally. Furthermore, we discuss the First-Child-Next-Sibling (FCNS) encoding, an encoding of unranked trees into binary trees, that we use in Section 6.4 for the validation of arbitrary XML documents.

Then, in Section 6.2 we show that there are DTDs and XML documents that encode ternary trees such that any p-pass streaming algorithm requires $\Omega(N/p)$ space for the validation. We also show that validating XML documents against XML schemas that are more expressive than DTDs is even harder: for EDTDs (extended DTDs), validating XML documents that encode a binary trees in the streaming model with p passes requires $\Omega(N/p)$ space.

Then, in Section 6.3 we discuss three streaming algorithms for the validation of XML documents that encode binary trees. We present two one-pass algorithms with space $\tilde{O}(\sqrt{N})$ and one bidirectional two-pass algorithm with space $O(\log^2 N)$.

In Section 6.4 we present our main result. We show that validity of any XML document against any DTD can be decided by a streaming algorithm with space $O(\log^2 N)$, $O(\log N)$ passes and a constant number of auxiliary streams.

Finally, we conclude with the presentation of linear space lower bounds for FCNS encoding and decoding for streaming algorithms without auxiliary streams in Section 6.5.

6.1 Preliminaries

Let Σ be a finite alphabet. The k-th letter of $X \in \Sigma^N$ is denoted by $X[k]$, for $1 \leq k \leq N$, and the consecutive letters of X between positions i and j by $X[i, j]$. A *subsequence* of X is any string $X[i_1]X[i_2] \dots X[i_k]$, where $1 \leq i_1 < i_2 < \dots < i_k \leq N$.

6.1.1 XML Documents

We consider finite unranked ordered labeled trees t, where each tree node has a label in Σ. From now on, we omit the terms ordered, labeled, and finite. Moreover, the children of every non-leaf node are ordered. k-ranked trees are a special case where each node has at most k children. Binary trees are a special type of 2-ranked tree, where each node is either a leaf or has exactly 2 children. We use the following notations to access the nodes of a tree:

- $\mathrm{root}(t)$: root node of tree t,
- $\mathrm{children}(x)$: ordered sequence of children nodes of node x, if x is a leaf then this sequence is empty,
- $\mathrm{fc}(x)$: first child of node x, if x is a leaf then $\mathrm{fc}(x) = \perp$,
- $\mathrm{ns}(x)$: next sibling of node x, if x is a right most (last) child then $\mathrm{ns}(x) = \perp$.

For each label $a \in \Sigma$, we associate its corresponding *opening tag* a and *closing tag* $\overline{a}$, standing for $\langle a \rangle$ and $\langle /a \rangle$ in the usual XML notations. An *XML sequence* is a sequence over the alphabet $\Sigma' = \{a, \overline{a} : a \in \Sigma\}$. The *XML sequence of a tree* t is the sequence of *opening tags* and *closing tags* in the order of a depth-first left-to-right traversal of t (Figure 6.1): when at step i we visit a node with label a top-down (respectively bottom-up), we let $X[i] = a$ (respectively $X[i] = \overline{a}$). Hence X is a word over $\Sigma' = \{a, \overline{a} : a \in \Sigma\}$ of size twice the number of nodes of t. The XML file describing t is unique, and we denote it as $\mathrm{XML}(t)$. We define $\mathrm{XML}(t)$ as a recursive function in Definition 15. For a node $x \in t$, we write (ambiguously) x and $\overline{x}$ to denote its opening and closing tag. x is also used to denote its label.

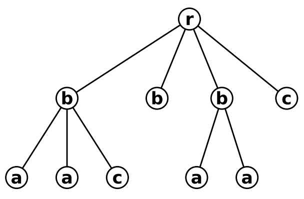


Figure 6.1: Let $\Sigma = \{a, b, c\}$, and let t be the tree as above. Then $\mathrm{XML}(t) = rba\overline{a}a\overline{a}c\overline{c}\overline{b}b\overline{b}ba\overline{a}a\overline{a}\overline{b}c\overline{c}\overline{r}$.

Definition 15. *Let t be an unranked tree, let* $x, x_1, \dots, x_n \in t$ *be nodes. Then:*

$$\begin{aligned} \mathrm{XML}(x) &= x\,\mathrm{XML}(\mathrm{children}(x))\,\overline{x}, \\ \mathrm{XML}(x_1, \dots, x_n) &= \mathrm{XML}(x_1) \dots \mathrm{XML}(x_n), \\ \mathrm{XML}(\perp) &= \epsilon, \end{aligned}$$

and we write $\mathrm{XML}(t)$ *for* $\mathrm{XML}(\mathrm{root}(t))$.

We assume that the input XML sequences X are *well-formed*, namely $X = \mathrm{XML}(t)$, for some tree t. The work [MMN10] legitimates this assumption, since checking well-formedness is at least as easy as any of our algorithms for checking validity. Hence, we could run an algorithm for well-formedness in parallel without increasing the resource requirements. Note that randomness is necessary for checking well-formedness with sublinear space, whereas we will show that randomness is not needed for validation.

Since the length of a well-formed XML sequence is known in advance, we will denote it by $2N$ instead of N. Each opening tag $X[i]$ and matching closing tag $X[j]$ in $X = \mathrm{XML}(t)$ corresponds to a unique tree node v of t. We sometimes denote v either by $X[i]$ or $X[j]$. Then, the *position* of v in X is $\mathrm{pos}(v) = i$. Similarly, $\mathrm{pos}(\overline{v}) = j$.

6.1.2 FCNS Encoding and Decoding

The *FCNS encoding* (see for instance [Nev02]) is an encoding of unranked trees as *extended* 2-ranked trees, where we distinguish left child from right child. This is an extension of ordered 2-ranked trees, since a node may have a left child but no right child, and vice versa. We therefore duplicate the labels $a \in \Sigma$ to a_{L} and a_{R}, in order to denote the *left* and *right* opening/closing tags of a. The FCNS tree is obtained by keeping the same set of tree nodes. The root node of the unranked tree remains the root in the FCNS tree, and we annotate it by default left. The left child of any internal node in the FCNS tree is the first child of this node in the unranked tree if it exists, otherwise it does not have a left child. The right child of a node in the FCNS tree is the next sibling of this node in the unranked tree if it exists, otherwise it does not have a right child. For a tree t, we denote $\mathrm{FCNS}(t)$ the FCNS tree, and $\mathrm{XML}(\mathrm{FCNS}(t))$ the XML sequence of the FCNS encoding of t. Figure 6.2 illustrates the construction of the FCNS encoding, and we define $\mathrm{XML}(\mathrm{FCNS}(t))$ by means of a recursive function $\mathrm{XML}^{\mathrm{F}}$ in Definition 16.

Definition 16. *Let t be an unranked tree, and let $x \in t$ be some node. Let $D \in \{\mathrm{L}, \mathrm{R}\}$. Then* $\mathrm{XML}^{\mathrm{F}}$ *is defined as follows:*

$$\begin{aligned} \mathrm{XML}^{\mathrm{F}}(x, D) &= x_D\, \mathrm{XML}^{\mathrm{F}}(\mathrm{fc}(x), \mathrm{L})\, \mathrm{XML}^{\mathrm{F}}(\mathrm{ns}(x), \mathrm{R})\, \overline{x_D}, \\ \mathrm{XML}^{\mathrm{F}}(\perp, D) &= \epsilon, \end{aligned}$$

and we write $\mathrm{XML}(\mathrm{FCNS}(t))$ *for* $\mathrm{XML}^{\mathrm{F}}(\mathrm{root}(t), L)$.

Instead of annotating by left/right, another way to uniquely identify a node as left or right is to insert dummy leaves with a new label $\perp$, and we assume that $\perp \notin \Sigma$. For a tree t, we denote the binary version without annotations and insertion of $\perp$ leaves by $\mathrm{FCNS}^{\perp}(t)$, and the XML sequence of $\mathrm{FCNS}^{\perp}(t)$ by $\mathrm{XML}(\mathrm{FCNS}^{\perp}(t))$. This is illustrated in Figure 6.3. The two representations can be easily transformed into each other. Depending on the application, we will use the more convenient version.

We call the transformation of $\mathrm{XML}(t)$ into $\mathrm{XML}(\mathrm{FCNS}(t))$ *FCNS encoding*, and the transformation of $\mathrm{XML}(\mathrm{FCNS}(t))$ into $\mathrm{XML}(t)$ *FCNS decoding*.

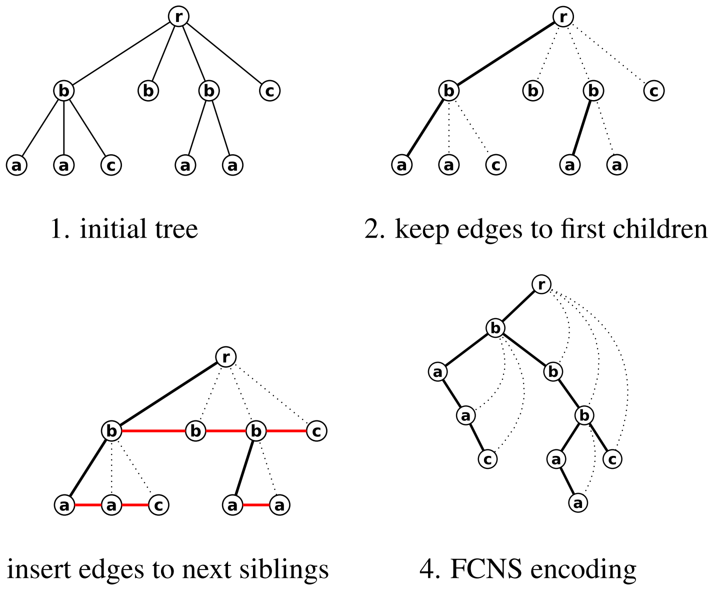

r
b
b
b
c
a
a
c
a
a
1. initial tree
2. keep edges to first children
3. insert edges to next siblings
4. FCNS encoding

Figure 6.2: Construction of the First-Child-Next-Sibling Encoding. 1: introductory example tree t already shown in Figure 6.1. 2: removal of all edges except edges to first children. 3: Insertion of edges to next siblings. 4: FCNS encoding of tree t. $\mathrm{XML}^F(t) = r_\mathrm{L} b_\mathrm{L} a_\mathrm{L} a_\mathrm{R} c_\mathrm{R} \overline{c_\mathrm{R} a_\mathrm{R} a_\mathrm{L}} b_\mathrm{R} b_\mathrm{R} a_\mathrm{L} a_\mathrm{R} \overline{a_\mathrm{R} a_\mathrm{L}} c_\mathrm{R} \overline{c_\mathrm{R}} \overline{b_\mathrm{R} b_\mathrm{R} b_\mathrm{L}} \overline{r_\mathrm{L}}$.

6.1.3 Validity and DTDs

We consider XML validity against DTDs.

Definition 17 (DTD)**.** *A DTD is a triple (Σ, d, s_d) where Σ is a finite alphabet, d is a function that maps Σ-symbols to regular expressions over Σ and $s_d \in \Sigma$ is the start symbol. A tree t satisfies (Σ, d, s_d) if its root is labeled by s_d, and for every node with label a, the sequence $a_1 \dots a_n$ of labels of its children is in the language defined by $d(a)$.*

Throughout the document we assume that DTDs are considerably small and our algorithms have full access to them without accounting this to their space requirements.

Definition 18 (VALIDITY)**.** *Let D be a DTD. The problem* VALIDITY *consists of deciding whether an input tree t given by its XML sequence* $\mathrm{XML}(t)$ *on an input stream is valid against D.*

We denote by VALIDITY(2) the problem VALIDITY restricted to input XML sequences describing binary trees.

6.2 Hardness of XML schemas

In this section, we discuss some lower bounds for checking different notions of validity of XML files. In section 6.2.1, we show that there are DTDs that admit ternary trees and checking those

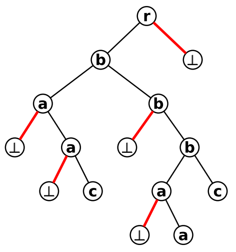


Figure 6.3: FCNS$^{\perp}$ encoding of the example tree already shown in Figure 6.1. $\text{XML}^{\text{F}\perp}(t) = rba\perp\overline{\perp}a\perp\overline{\perp}c\overline{c}\overline{a}\overline{a}b\perp\overline{\perp}ba\perp\overline{\perp}a\overline{a}\overline{a}c\overline{c}\overline{b}\overline{b}\overline{b}\perp\overline{\perp}\overline{r}$.

requires linear space. In section 6.3 we show that checking DTD validity of XML documents encoding binary trees can be done with sublinear space. We conclude that ternary trees are necessary for obtaining a linear space lower bound.

In section 6.2.2, we consider validity notions that allow to express a node's validity as a function of its children and its grandchildren. These validity notions are harder to check than DTD validity in the sense that even checking XML documents encoding binary trees requires linear space.

6.2.1 A Linear Space Lower Bound for VALIDITY Using Ternary Trees

We provide now a proof showing that p-pass algorithms require $\Omega(N/p)$ space for checking validity of arbitrary XML files against arbitrary DTDs. Many space lower bound proofs for streaming algorithms are reductions from problems in communication complexity [AMS99, BYJKS04, MMN10]. For an introduction to communication complexity we refer the reader to [KN97].

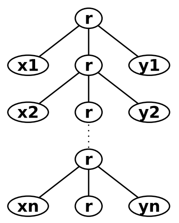
Consider a player Alice holding an N bit string $x = x_1 \dots x_N$, and a player Bob holding an N bit string $y = y_1 \dots y_N$ both taken from the uniform distribution over $\{0,1\}^N$. Their common goal is to compute the function $f(x,y) = \bigvee_i x[i] \wedge y[i]$ by exchanging messages. This communication problem is the widely studied problem Set-Disjointness (DISJ).

It is well known that the randomized communication complexity with bounded two-sided error of the Set Disjointness function $R(\text{DISJ}) = \Theta(N)$. In this model, the players Alice and Bob have access to a common string of independent, unbiased coin tosses. The answer is required to be correct with probability at least $2/3$.

We make use of this fact by encoding this problem into an XML validity problem. Consider $\Sigma^{\text{DISJ}} = \{r, 0, 1\}$, the DTD $D^{\text{DISJ}} = (\Sigma^{\text{DISJ}}, d^{\text{DISJ}}, r)$ such that $d^{\text{DISJ}}(r) = 0r0 \mid 0r1 \mid 1r0 \mid \epsilon$, $d^{\text{DISJ}}(0) = \epsilon$, and $d^{\text{DISJ}}(1) = \epsilon$. Given an input x, y as above, we construct an input tree $t(x,y)$ as in Figure 6.4.

Clearly, $\text{DISJ}(x,y) = 0$ if and only if $\text{XML}(t(x,y))$ is valid with respect to D^{DISJ}.

Theorem 13. *Every p-pass randomized streaming algorithm for* VALIDITY *with bounded error uses $\Omega(N/p)$ space, where N is the input length.*



$$d^{\mathrm{DISJ}}(r) = 0r0 \,|\, 0r1 \,|\, 1r0 \,|\, \epsilon$$
$$d^{\mathrm{DISJ}}(0) = d^{\mathrm{DISJ}}(1) = \epsilon$$

Figure 6.4: $t(x,y)$ is a hard instance for VALIDITY.

Proof. Given an instance $x \in \{0,1\}^N$, $y \in \{0,1\}^N$ of DISJ, we construct an instance for VALIDITY. Then, we show that if there is a p-pass randomized algorithm for VALIDITY using space s with bounded error, then there is a communication protocol for DISJ with the same error and communication $O(s \cdot p)$. This implies that any p-pass algorithm for VALIDITY requires space $\Omega(N/p)$ since $R(\mathrm{DISJ}) = \Theta(N)$.

Assume that A is a randomized streaming algorithm deciding validity with space s and p passes. Alice generates the first half of XML$(t(x,y))$, that is $rx_1\overline{x_1}rx_2\overline{x_2}\dots rx_N\overline{x_N}r$ of length $3N+1$ and executes algorithm A on this sequence using a memory of size $O(s)$. Alice sends the content of the memory to Bob via message M_A^1. Bob initializes his memory with M_A^1, and continues algorithm A on the second half of XML$(t(x,y))$, that is $\overline{r}y_N\overline{y_N}\overline{r}\dots\overline{r}y_2\overline{y_2}\overline{r}y_1\overline{y_1}\overline{r}$ of length $3N+1$. After execution, Bob sends the content of the memory back to Alice via M_B^1. This procedure is repeated at most p times.

This protocol has a total length of $O(s \cdot p)$ since the size of each message is at most s. Since $R(\mathrm{DISJ}) \in \Theta(N)$, we obtain that $s \cdot p \in \Omega(N)$. The claim follows. □

6.2.2 Lower Bounds for More Expressive XML Schemas than DTDs

Suppose that a validity schema allows to express a node's validity not only through the labels of its children but also of its grandchildren. Note that this is not the case for DTDs since DTD validity only considers the direct descendants of a node for checking its validity. We show that checking validity against such schemas requires linear space even if the XML document encodes a binary tree, see Theorem 14 below.

As in the prior subsection, we encode the communication problem Set-Disjointness into an XML document. Let $x = x_1 \dots x_n \in \{0,1\}^n$ denote the input of Alice, and let $y = y_1 \dots y_n \in \{0,1\}^n$ denote the input of Bob. They construct a binary tree $t'(x,y)$ as on the left side of Figure 6.5.

$t'(x,y)$ is valid if the subtrees below a node with label r are as in the right side of Figure 6.5. Only if this is true then $\mathrm{DISJ}(x,y) = 0$. EDTD as well as XML schema allow to express validity constraints of that kind. EDTDs were introduced in [PV00] under the name *specialized DTDs*. They are defined as follows.

Definition 19. *An extended DTD (EDTD) is a tuple $D = (\Sigma, \Delta, d, s_d, \mu)$, where Δ is a finite set of types, μ is a mapping from Δ to Σ, and (Δ, d, s_d) is a DTD. A tree t satisfies D if $t = \mu(t')$ for some t satisfying the DTD (Δ, d, s_d).*

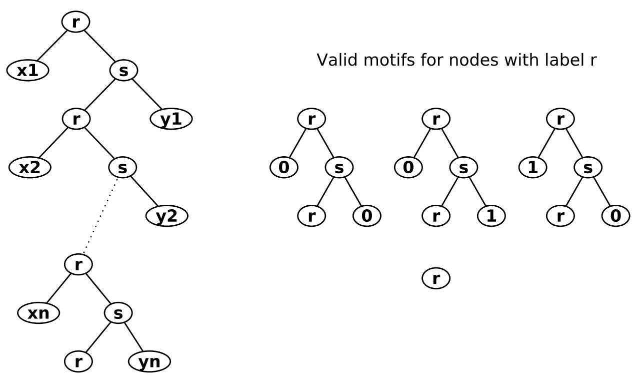


Figure 6.5: Left: hard instance $t'(x, y)$ for validity schemas that allow to relate nodes to its grandchildren. Right: the validity constraints for nodes labeled r.

EDTD validity of the trees $t'(x, y)$ of Figure 6.5 can be checked by the EDTD $D = (\Sigma, \Delta, d, s_d, \mu)$ where

$$\begin{aligned}
\Sigma &= \{r, s, 0, 1\}, \Delta = \{r, 0, 1, s_0, s_1\}, s_d = r, \\
\mu(r) &= r, \mu(0) = 0, \mu(1) = 1, \mu(s_0) = s, \mu(s_1) = s, \text{ and} \\
d(0) &= \epsilon, d(1) = \epsilon, d(s_0) = r\,0, d(s_1) = r\,1, d(r) = 0s_0|0s_1|1s_0|\epsilon.
\end{aligned}$$

Theorem 14. *Every p-pass randomized streaming algorithm validating XML documents encoding binary trees against a validity schema that allows to express a node's validity as a function of its children and its grandchildren with bounded error uses $\Omega(N/p)$ space, where N is the input length.*

Proof. The proof is identical to the proof of Theorem 13, except that the encoding is slightly different. Let $x \in \{0, 1\}^N$, $y \in \{0, 1\}^N$ be an instance of DISJ. Alice generates the left half of the tree $t'(x, y)$ as follows: $rx_1\overline{x_1}srx_2\overline{x_2}s \dots rx_n\overline{x_n}sr\overline{r}$. Bob generates the right half of the tree $t'(x, y)$ as follows: $y_n\overline{y_n}\overline{s}\overline{r}y_{n-1}\overline{y_{n-1}}\overline{s}\overline{r} \dots y_1\overline{y_1}\overline{s}\overline{r}$. A p-pass streaming algorithm with space s checking the two-level validity constraints as on the right side of Figure 6.5 of $t'(x, y)$ solves hence DISJ with a protocol of length $O(s \cdot p)$. Since $R(\text{DISJ}) = \Theta(N)$, the result follows. □

6.3 Validity of Binary Trees

For simplicity, we only consider binary trees in this section. A *left opening/closing tag* (respectively *right opening/closing tag*) of an XML sequence X is a tag whose corresponding node is the first child of its parent (respectively second child).

Our algorithms for binary trees can be extended to 2-ranked trees. This requires few changes in the one-pass Algorithms 14 and 15, and the two-pass Algorithm 16 (indeed in the subroutine Algorithm 17) that we do not describe here since they only complicate the presentation and do not affect the essence of the algorithms.

We fix now a DTD $D = (\Sigma, d, s_d)$, and assume that in our algorithms we have access to a procedure $\mathrm{check}(v, v_1, v_2)$ that signalizes invalidity and aborts if $v_1 v_2$ is not valid against the regular expression $d(v)$. Otherwise it returns without any action.

In order to validate an XML document, we ensure validity of all tree nodes. For checking validity of a node v with two children v_1, v_2, we have to relate the labels v_1, v_2 to v. In a *top-down* verification we use the opening tag v of the parent node v for verification, in a *bottom-up* verification we use the closing tag $\overline{v}$ of the parent node v.

6.3.1 One-pass Block Algorithm

Algorithms 14 reads the XML document in blocks of size K (we optimize by setting $K = \sqrt{N \log N}$) into memory. Such a block corresponds to a subtree, and the algorithm performs all verifications that are possible within this block. We guarantee that all nodes are verified by ensuring that all substrings $\overline{v_1} v_2$ that correspond to the children of a node v are used for verification. We show in Lemma 35 that within a block of any size there is at most one node v with children v_1, v_2 such that $\overline{v_1}$ is in that block but neither the opening tag v nor the closing tag $\overline{v}$ is in that block. Hence, per block all necessary verifications but at most one can be performed. If a pair of tags $\overline{v_1} v_2$ can not be related to their parent node within a block, we store $\overline{v_1} v_2$ and we perform a bottom-up verification upon arrival of the parent node's closing tag $\overline{v}$, see Algorithm 14.

In order to compute the depth of tags (as it is required for instance in Line 13), throughout the algorithm we keep track of the current depth with the help of an integer with initial value 0. We increase its value when we encounter an opening tag in the stream and we decrease it when we encounter a closing tag. The depth of a tag is the number of opening tags minus the number of closing tags that preceed the tag in the input stream.

The condition in line 10 can be checked as follows. Starting from index i such that $X[i] = \overline{v_1}$, we firstly traverse X to the left. The first encountered opening tag that has a depth $\mathrm{depth}(v_1) - 1$ (if any) is the opening tag of the parent node v. If the parent node's opening tag is not in X, we traverse then X to the right starting at index i. The first encountered closing tag at level $\mathrm{depth}(v_1) - 1$ (if any) is the closing tag of the parent node v. If X does not comprise any tags at depth $\mathrm{depth}(v_1) - 1$ then the condition evaluates to false. Similarly, the condition in line 19 can be checked. For implementing the condition in line 8 a lookahead of one on the stream might be required if the last tag of X is an opening tag.

Lemma 35. *Let $X[i, j]$ be a block. Then there is at most one left closing tag $\overline{a}$ with parent node p such that:*

$$\mathrm{pos}(p) < i \leq \mathrm{pos}(\overline{a}) \leq j < \mathrm{pos}(\overline{p}). \tag{6.1}$$

Proof. For the sake of a contradiction, assume that there are 2 left closing tags $\overline{a}, \overline{b}$ with p being the parent node of a, and q being the parent node of b, for which Inequality 6.1 holds. Without loss of generality, we assume that $\mathrm{pos}(p) < \mathrm{pos}(q)$. Since $\mathrm{pos}(p) < \mathrm{pos}(q) < \mathrm{pos}(\overline{a})$, q is contained in the subtree of a or $q = a$. This, however, implies that $\mathrm{pos}(\overline{q}) \leq \mathrm{pos}(\overline{a}) < j$ contradicting $\mathrm{pos}(\overline{q}) > j$. □

Algorithm 14 Validity of Binary Trees in one Pass, Block Algorithm

```
Require: input stream is a well-formed XML document
 1: $K \leftarrow \sqrt{N \log N}$
 2: $X \leftarrow$ array of size $K+1$, $S \leftarrow$ empty stack
 3: while stream not empty do
 4:    $X \leftarrow$ next $K$ tags on stream
 5:    if $X[K]$ is a closing tag and next tag on stream is an opening tag then
 6:       $X[K+1] \leftarrow$ next tag on stream
 7:    end if
 8:    for all leaves $v$ in $X$ do check($v, \epsilon, \epsilon$) end for
 9:    for all substrings $\overline{v_1}v_2$ of $X$ do {denote the parent node of $v_1, v_2$ by $v$}
10:       if $v \in X$ or $\overline{v} \in X$ then
11:          check($v, v_1, v_2$)
12:       else
13:          push($(v_1, v_2, \mathrm{depth}(v_1)), S$)
14:       end if
15:    end for
16:    if stack $S$ not empty then
17:       repeat
18:          $(v_1, v_2, d_1) \leftarrow$ topmost item on stack $S$ {denote the parent node of $v_1, v_2$ by $v$}
19:          if $v \in X$ or $\overline{v} \in X$ then
20:             check($v, v_1, v_2$)
21:             pop $S$
22:          end if
23:       until $v \notin X$ and $\overline{v} \notin X$ or $S$ empty
24:    end if
25: end while
```

Theorem 15. *Algorithm 14 is a one-pass streaming algorithm for* VALIDITY(2) *with space* $O(\sqrt{N \log N})$.

Proof. To prove correctness, we have to ensure validity of all nodes. Leaves are validated in line 8. Concerning non-leaf nodes, note that all substrings $\overline{v_1}v_2$ are used for validation. Either a node v is validated in line 11 if its opening tag v or its closing tag $\overline{v}$ is in the same block as $\overline{v_1}v_2$, or the node is validated in line 20 if v, $\overline{v_1}v_2$ and $\overline{v}$ are all in different blocks. In this case, the children are pushed on the stack S and the verification is done upon arrival of $\overline{v}$.

Concerning the space, X is of size at most $K+1$. By Lemma 35, the stack S grows at most by one element per iteration of the while loop. A stack element requires $O(\log N)$ storage space since we require to store the depth of the tags which is a number in $[N]$. Since there are $O(N/K)$ iterations, the total memory requirements are $O(K + N/K \log(N))$ which is minimized for $K = \sqrt{N \log N}$. □

6.3.2 One-pass Stack Algorithm

We present now a second one-pass streaming algorithm, Algorithm 15, for checking validity of XML documents that encode binary trees. This algorithm has the same space complexity as the block algorithm, Algorithm 14, of the previous section, however, it has optimal (constant) processing time per letter.

Algorithm 15 Validity of Binary Trees in one Pass, Stack Algorithm

```
Require: input stream is a well-formed XML document
 1: $d \leftarrow 0, S \leftarrow$ empty stack
 2: $K \leftarrow \sqrt{N \log N}$
 3: while stream not empty do
 4:     $x \leftarrow$ next tag on stream
 5:     if $x$ is an opening tag $c$ then
 6:         if $x$ is a leaf then check$(c, \epsilon, \epsilon)$ end if
 7:         if $S$ has on top $(a, -1), (\overline{b}, d)$ then
 8:             check($a, b, c$); pop $S$ {Top-down verification}
 9:         end if
10:         if $|\{(a, -1) \in S \mid a$ opening $\}| \geq K$ then
11:             remove bottom-most $(a, -1)$ in $S$, where $a$ is an opening tag
12:         end if
13:         $d \leftarrow d + 1$
14:         push $((x, -1), S)$
15:     else if $x$ is a closing tag $\overline{c}$ then
16:         $d \leftarrow d - 1$
17:         if S has on top $(\overline{a}, d + 1), (\overline{b}, d + 1)$ then
18:             check $(c, a, b)$ {Bottom-up verification}
19:             pop $S$, pop $S$
20:         else if $S$ has on top $(\overline{b}, d + 1)$ then
21:             pop $S$
22:         end if
23:         if $S$ has on top $(c, -1)$ then pop $S$ end if
24:         push $((x, d), S)$
25:     end if
26: end while
```

Algorithms 15 performs top-down and bottom-up verifications. It uses a stack onto which it pushes all opening tags in order to perform top-down verifications once the information of the children nodes arrives on the stream. $\overline{v_1}v_2$ forms a substring of the input, hence top-down verification requires only the storage of the opening tag v since the labels of the children arrive in a block. The algorithm's space requirement depends on a parameter K (we optimize by setting $K = \sqrt{N \log N}$). Once the number of opening tags on the stack is about to exceed K, we remove the bottom-most opening tag. The corresponding node will then be verified bottom-up. Note that $\overline{v_2 v}$ forms a substring of the input. Hence, for bottom-up verifications it is enough to store the label of the left child v_1 on the stack since the label of the right child arrives in form of a closing tag right before the closing tag of the parent node. See Algorithm 15 for details.

For the unique identification of closing tags on the stack, we have to store them with their depth in the tree. A stack item corresponding to a closing tag requires hence $O(\log N)$ space. Opening tags don't require the storage of their depth (we store the default depth -1).

The query in line 6 can be implemented by a lookahead of 1 on the stream. The opening tag x corresponds to a leaf only if the subsequent tag in the stream is the corresponding closing tag $\overline{x}$.

Figure 6.6 visualizes the different cases with their stack modifications appearing in Algorithm 15.

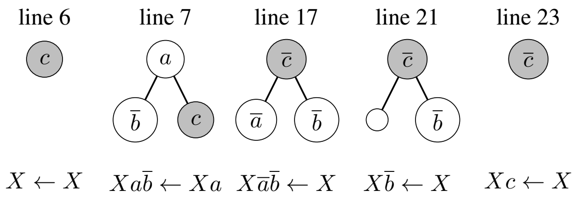


Figure 6.6: Visualization of the different conditions in Algorithm 15 with the applied stack modifications. X represents the bottom part of the stack. Note that Algorithm 15 pushes the currently treated tag c or $\overline{c}$ on the stack in Line 14 or Line 24. c or $\overline{c}$ corresponds to the highlighted node.

Fact 1 (which can be easily proved by induction) and Lemma 36 concern the structure of the stack S used in Algorithm 15.

Fact 1. *Let $S = (x_1, d_1), \dots (x_k, d_k)$ be the stack at the beginning of the while loop in line 3. Then:*

1. $\text{pos}(x_1) < \text{pos}(x_2) \cdots < \text{pos}(x_k)$,
2. $\text{depth}(x_1) \leq \text{depth}(x_2) \cdots \leq \text{depth}(x_k) \leq d$*. Moreover, if* $\text{depth}(x_i) = \text{depth}(x_{i+1})$ *then x_i is the left sibling of x_{i+1},*
3. *The sequence $x_1 \dots x_k$ satisfies the regular expression $\overline{a}^* b^* (\epsilon \,|\, \overline{c} \,|\, \overline{d}\overline{e})$, where $\overline{a}^*$ are left closing tags, b^* are opening tags, $\overline{c}$ is a closing tag, $\overline{d}$ is a left closing tag, and $\overline{e}$ is a right closing tag.*
4. *A left closing tag $\overline{a}$ is removed from S just after its parent node is verified.*

Lemma 36. *Let $S = (x_1, d_1), \dots (x_k, d_k)$ be the stack at the beginning of the while loop in line 3. Let $(\overline{c_i}, d_i), (\overline{c_{i+1}}, d_{i+1})$ be two consecutive left closing tags in S such that $(\overline{c_{i+1}}, d_{i+1})$ is not the topmost left closing tag. Then* $\text{pos}(\overline{c_{i+1}}) \geq \text{pos}(\overline{c_i}) + 2K$.

Proof. Denote by $X = X[1]X[2]\dots X[2N]$ the input stream. Since $\overline{c_{i+1}}$ is not the topmost left closing tag in S, the algorithm has already processed the right sibling opening tag $X[\text{pos}(\overline{c_{i+1}}) + 1]$ of $\overline{c_{i+1}}$. By Item 4 of Fact 1, no verification has been done of the parent of $\overline{c_{i+1}}$, since $\overline{c_{i+1}}$ is still in S. Therefore, the parent's opening tag $X[k]$ of $\overline{c_{i+1}}$ has been deleted from S, where $\text{pos}(\overline{c_i}) < k < \text{pos}(\overline{c_{i+1}})$. This can only happen if at least K opening tags have been pushed on S between $X[k]$ and $\overline{c_{i+1}}$. Since these K opening tags must have been closed between $X[k]$ and $\overline{c_{i+1}}$ we obtain $\text{pos}(\overline{c_{i+1}}) \geq \text{pos}(\overline{c_i}) + 2K$. □

Fact 1 and Lemma 36 provide more insight in the stack structure and are used in the proof of Theorem 16. Item 3 of Fact 1 states that the stack basically consists of a sequence of left closing tags which are the left children that are needed for bottom-up verifications of nodes

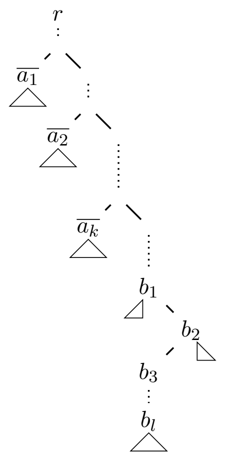


Figure 6.7: Visualization of the structure of the stack used in Algorithm 15. The stack fulfills the regular expression $\overline{a}^* b^* (\epsilon \,|\, \overline{c} \,|\, \overline{d}\overline{e})$, compare Item 3 of Fact 1. The $(\overline{a_i})_{i=1\dots k}$ are closing tags whose parents' nodes were not verified top-down. For $j > i$, a_j is connected to a_i by the right sibling of a_i. The $(b_i)_{i=1\dots l}$ form a sequence of opening tags such that b_i is the parent node of b_{i+1}. On top of the stack might be one or two closing tags depending on the current state of the verification process.

that could not be verified top-down. This sequence is followed by a sequence of opening tags for which we still aim a top-down verification. The proof of Lemma 36 explains the fact that the two sequences are strictly separated: a left-closing tag $\overline{v_1}$ only remains on the stack if at the moment of insertion there are no opening tags on the stack.

Theorem 16. *Algorithm 15 is a one-pass streaming algorithm for* VALIDITY(2) *with space* $O(\sqrt{N \log N})$ *and* $O(1)$ *processing time per letter.*

Proof. To prove correctness, we have to ensure validity of all nodes. Leaves are correctly validated upon arrival of its opening tag in line 6. Concerning non-leaf nodes, firstly, note that all closing tags are pushed on S in line 24, in particular all closing tags of left children appear on the stack. The algorithm removes left closing tags only after validation of its parent node, no matter whether the verification was done top-down or bottom-up, compare Item 4 of Fact 1. Emptiness of the stack after the execution of the algorithm follows from Item 2 of Fact 1 and implies hence the validation of all non-leaf nodes.

For the space bound, Line 10 guarantees that the number of opening tags in S is always at most K. We bound the number of closing tags on the stack by $\frac{N}{K} + 2$. Item 3 of Fact 36 states that the stack contains at most one right closing tag. From Item 4 of Fact 36 we deduce that S comprises at most $\frac{N}{K} + 1$ left closing tags, since the stream is of length $2N$, and the distance in the stream of two consecutive left closing tags that reside on S except the top-most one is at least $2K$. A closing tag with depth $(a, d) \in \Sigma' \times [N]$ requires $O(\log N)$ space, an opening tag requires only constant space. Hence the total space requirements are $O((\frac{N}{K} + 2) \log N + K)$ which is minimized for $K = \sqrt{N \log N}$.

Concerning the processing time per letter, the algorithm only performs a constant number of local stack operations in one iteration of the while loop. □

Remark Algorithm 15 can be turned into an algorithm with space complexity $O(\sqrt{D \log D})$, where D is the depth of the XML document. If D is known beforehand, it is enough to set $K = \sqrt{D \log D}$ in line 2. If D is not known in advance, we make use of an auxiliary variable D' storing a guess for the document depth. Initially we set $D' = C$, $C > 0$ some constant, we set $K = \sqrt{D' \log D'}$, and we run Algorithm 15. Each time d exceeds D', we double D', and we update K accordingly.

This guarantees that the number of opening tags on the stack is limited by $O(\sqrt{D \log D})$. Since we started with a too small guess for the document depth, we may have removed opening tags that would have remained on the stack if we had chosen the depth correctly. This leads to further bottom-up verifications, but no more than $O(\sqrt{D/\log D})$ guaranteeing $O(\sqrt{D \log D})$ space.

6.3.3 Bidirectional Two-pass Algorithm

Algorithm 16 Two-pass Algorithm Validating Binary Trees

```
run Algorithm 17 reading the stream from left to right
run Algorithm 17 reading the stream from right to left, where opening tags are interpreted as closing
tags, and vice versa.
```

Algorithm 17 Validating Nodes with size(Left Subtree) $\geq$ size(Right Subtree)

```
1: $l \leftarrow 0$; $n \leftarrow 0$; $S \leftarrow$ empty stack
2: while stream not empty do
3:     $x \leftarrow$ next tag on stream (and move stream to next tag)
4:     $y \leftarrow$ next tag on stream, without consuming it yet
5:     $n \leftarrow n + 1$
6:     if $x$ is an opening tag $c$ then
7:         $l \leftarrow l + 1$
8:         if $y = \bar{c}$ then check($c, \epsilon, \epsilon$) end if
9:     else {$x$ is a closing tag $\bar{c}$}
10:        $l \leftarrow l - 1$
11:        if $S$ has on top $(\cdot, \cdot, l+1, \cdot)$ then
12:            $(\bar{a}, b, \cdot, \cdot) \leftarrow$ pop from $S$; check($c, a, b$)
13:        end if
14:        if $y$ is an opening tag $d$ then
15:            push ($\bar{c}$, $d$, $l$, $n$) to $S$
16:        end if
17:    end if
18:    while there is $s_1 = (\cdot, \cdot, \cdot, n_1)$ just below $s_2 = (\cdot, \cdot, \cdot, n_2)$ in $S$ with $n - n_2 > n_2 - n_1$ do
19:        delete $s_2$ from $S$
20:    end while
21: end while
```

The bidirectional two-pass algorithm, Algorithm 16, uses a subroutine that checks in one-pass validity of all nodes whose left subtree is at least as large as its right subtree. Feeding into this subroutine the XML document read in reverse direction and interpreting opening tags as closing tags and vice versa, it checks validity of all nodes whose right subtree is at least as large as its left subtree. In this way all tree nodes get verified.

The subroutine performs only checks in a bottom-up fashion, that is, the verification of a node v with children c_1, c_2 makes use of the tags $\overline{c_1}$ and c_2 (which are adjacent in the XML document and hence easy to recognize) and the closing tag of $\overline{v}$. When $\overline{c_1}, c_2$ appear in the stream, a 4-tuple consisting of $\overline{c_1}, c_2, \mathrm{depth}(c_1)$ and $\mathrm{pos}(\overline{c_1})$ is pushed on the stack. Upon arrival of $\overline{v}$, $\mathrm{depth}(c_1)$ is needed to identify c_1, c_2 as the children of v. $\mathrm{pos}(\overline{c_1})$ is needed for cleaning the stack: with the help of the pos values of the stack items, we identify stack items whose parents' nodes have larger right subtrees than left subtrees, and these stack items get removed from the stack. In so doing, we guarantee that the stack size does not exceed $\log(N)$ elements which is an exponential improvement over the one-pass algorithm.

Note that the reverse pass can be done independently of the first one, for instance in parallel to the first pass.

Figure 6.8 visualizes the different cases in Algorithm 17.

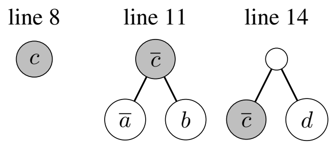


Figure 6.8: Visualization of the different conditions in Algorithm 17. The incoming tag x corresponds to the highlighted node.

We highlight some properties concerning the stack used in Algorithm 17.

Fact 2. *S in Algorithm 17 satisfies the following:*

1. *If $(\overline{a_2}, b_2, \mathrm{depth}(\overline{a_2}), \mathrm{pos}(a_2))$ is below $(\overline{a_1}, b_1, \mathrm{depth}(\overline{a_1}), \mathrm{pos}(a_1))$ in S, then $\mathrm{pos}(\overline{a_2}) < \mathrm{pos}(\overline{a_1})$, $\mathrm{depth}(\overline{a_2}) < \mathrm{depth}(\overline{a_1})$, and a_1, b_1 are in the subtree of b_2.*
2. *Consider l at the end of the while loop in line 20. Then there are no stack elements $(\cdot, \cdot, l', \cdot)$ with $l' > l$.*

Figure 6.9 illustrates the relationship between two consecutive stack elements as discussed in Item 1 of Fact 2.

Lemma 37. *Algorithm 17 verifies all nodes v whose left subtree is at least as large as its right subtree.*

Proof. Let q be such a node. Let a_1, b_1 be the children of q. Then it holds that

$$\mathrm{pos}(\overline{a_1}) - \mathrm{pos}(a_1) \geq \mathrm{pos}(\overline{b_1}) - \mathrm{pos}(b_1), \tag{6.2}$$

since the size of the left subtree of q is at least as large as the size of the right subtree.

Upon arrival of $\overline{a_1}$ Algorithm 17 pushes the 4-tuple $t = (\overline{a_1}, b_1, \mathrm{pos}(\overline{a_1}), \mathrm{depth}(a_1))$ onto the stack S. We have to show that t remains on the stack until the arrival of $\overline{q}$. More precisely,

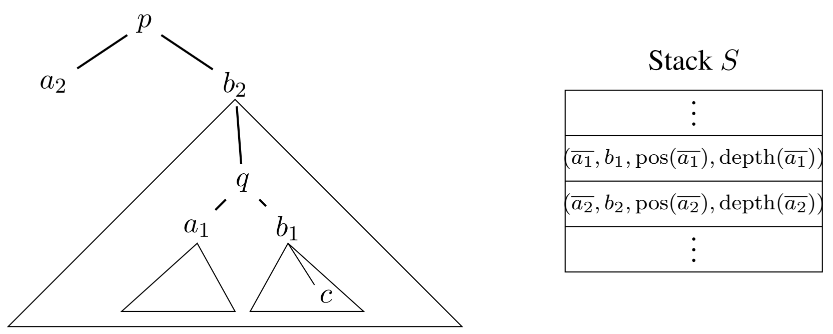

p
a_2
b_2
q
a_1
b_1
c
Stack S
$\vdots$
$(\overline{a_1}, b_1, \text{pos}(\overline{a_1}), \text{depth}(\overline{a_1}))$
$(\overline{a_2}, b_2, \text{pos}(\overline{a_2}), \text{depth}(\overline{a_2}))$
$\vdots$

Figure 6.9: Visualization of two consecutive stack items. c is the current element under consideration in Algorithm 17. a_1, b_1 is in the subtree of b_2, compare Item 1 of Fact 2.

we have to show that the condition in line 18 is never satisfied for $s_2 = t$. Since the algorithm never deletes the bottom-most stack item, we consider the case where there is a stack item $(\overline{a_2}, b_2, \text{pos}(\overline{a_2}), \text{depth}(a_2))$ just below t. Item 1 of Fact 2 tells us that a_1, b_1 are in the subtree of b_2. Let c be the current tag under consideration such that $\text{pos}(b_1) < \text{pos}(c) < \text{pos}(\overline{q})$. The situation is visualized in Figure 6.9.

According to the condition of line 18, t gets removed from the stack if

$$\text{pos}(c) - \text{pos}(\overline{a_1}) > \text{pos}(\overline{a_1}) - \text{pos}(\overline{a_2}). \tag{6.3}$$

Note that the left side of Inequality 6.3 is a lower bound on the size of the right subtree of q. Furthermore, the right side of Inequality 6.3 is an upper bound for the size of the left subtree of q.

Using $\text{pos}(c) - \text{pos}(\overline{a_1}) \leq \text{pos}(\overline{b_1}) - \text{pos}(b_1) + 1$ and $\text{pos}(\overline{a_1}) - \text{pos}(\overline{a_2}) > \text{pos}(\overline{a_1}) - \text{pos}(a_1)$, Inequality 6.3 contradicts Inequality 6.2 which shows that t remains on the stack until the arrival of $\overline{q}$. Item 2 of Fact 2 guarantees that there is no other stack element on top of t upon arrival of $\overline{q}$. This guarantees the verification of node q and proves the lemma. □

Theorem 17. *Algorithm 16 is a bidirectional two-pass streaming algorithm for* VALIDITY(2) *with space $O(\log^2 N)$ and $O(\log N)$ processing time per letter.*

Proof. To prove correctness of Algorithm 16, we ensure that all nodes get verified. By Lemma 37, in the first pass, all nodes with a left subtree being at least as large as its right subtree get verified. The second pass ensures then verification of nodes with a right subtree that is at least as large as its left subtree.

Next, we prove by contradiction that for any current value of variable n in Algorithm 17, the stack contains at most $\log(n)$ elements. Assume that there is a stack configuration of size $t \geq \log(n) + 1$. Let $(n_1, n_2 \dots, n_t)$ be the sequence of the fourth parameters of the stack elements. Since these elements are not yet removed, due to line 18 of Algorithm 17, it holds that $n - n_i \leq n_i - n_{i-1}$, or equivalently $n_i \geq 1/2(n + n_{i-1})$, for all $1 < i \leq t$. Since $n_1 \geq 1$, we obtain that $n_i \geq \frac{2^i - 1}{2^i} n + \frac{1}{2^i}$, and, in particular, $n_{t-1} \geq (n-1) + \frac{1}{n}$. Since all n_i are integers, it holds that $n_{t-1} \geq n$. Furthermore, since $n_t > n_{t-1}$, we obtain $n_{\log n + 1} \geq n + 1$ which is a contradiction, since the element at position $n + 1$ has not yet been seen.

Since $n \leq 2N$ and the size of a stack element is in $O(\log n)$, Algorithm 17 uses space $O(\log^2 N)$. This also implies that the while-loop at line 18 of Algorithm 17 can only be iterated $O(\log n)$ times during the processing of a tag on the stream. The processing time per letter is then $O(\log N)$, since we assume that operations on the stack run in constant time. $\square$

6.4 Validity of General Trees

First, we present in Subsection 6.4.1 a trivial one-pass algorithm that uses space $O(d)$ where d is the depth of the input XML document. Then, in Subsection 6.4.2 we design a streaming algorithm that uses space $O(\log^2 N)$ and 3 auxiliary streams, and makes $O(\log N)$ passes.

6.4.1 One-pass Algorithm with Space Linear in the Depth of the Document

We discuss now a straight-forward one-pass streaming algorithm, Algorithm 18, that uses $O(d)$ space to validate a well-formed XML document of depth d. Let (Σ, e, s_e) denote the input DTD, and for all $c \in \Sigma$ let A_c be a deterministic finite automaton with initial state q_e^0 and transition function δ_e that accepts words ω iff $e(c)$ accepts ω. Note that since we assume in this work that the size of the input DTD is $O(1)$, the size of $(A_e)_{e \in \Sigma}$ and the time complexity to compute it is $O(1)$.

In order to validate the input stream, we check for all nodes v that the sequence of labels of its children fulfills the regular expression $e(v)$ (remember: we write ambiguously v to denote the node as well as the label of v). We do this by feeding the sequence of labels of its children into automaton A_v, and we reject if the automaton does not accept this sequence.

Consider an internal node v at depth $\mathrm{depth}(v)$ with children $c_1, \dots, c_k$. Then $vc_1 \dots \overline{c_1}c_2 \dots \overline{c_{k-1}}c_k \dots \overline{c_k}\overline{v}$ is a substring of the input stream. As soon as we encounter the opening tag v, we store the initial state of A_v in an array S at index $\mathrm{depth}(v)$. As soon as an opening tag of a children node of v is encountered, we compute the follow-up state of $S[\mathrm{depth}(v)]$ by feeding the children node's label into A_v on state $S[\mathrm{depth}(v)]$. When $\overline{v}$ is reached and $S[\mathrm{depth}(v)]$ is not an accepting state of A_v, we report invalidity.

Since the depth of the document is d, there are at most d nodes whose validity has to be checked at the same time. These nodes are the nodes on the path from the current node to the root node. Therefore, we need to store at most d states of the automata $(A_\sigma)_{\sigma \in \Sigma}$ leading to a space complexity $O(d)$. See Algorithm 18 for details.

Algorithm 18 One-pass Streaming Algorithm for VALIDITY with space $O(d)$

```
Require: input stream is a well-formed XML document of depth $d$
 1: $l \leftarrow -1$, $L, S \leftarrow$ array of size $d$
 2: while stream not empty do
 3:   $x \leftarrow$ next tag on stream
 4:   if $x$ is an opening tag $c$ then
 5:     if $l = -1$ then {Root node}
 6:       if $c \neq s_e$ then report error and abort end if
 7:     else {Node different from the root node}
 8:       $S[l] \leftarrow \delta_{L[l]}(S[l], c)$
 9:     end if
10:     $l \leftarrow l + 1$
11:     $L[l] \leftarrow c$
12:     $S[l] \leftarrow q_c^0$
13:   else {$x$ is a closing tag $\overline{c}$}
14:     if $l \neq -1$ then
15:       if $S[l]$ is not an accepting state of $A_c$ then report error and abort end if
16:     end if
17:     $l \leftarrow l - 1$
18:   end if
19: end while
```

Theorem 18. *Algorithm 18 is a deterministic one-pass streaming algorithm for* VALIDITY *with space $O(d)$ and $O(1)$ processing time per letter where d is the depth of the input XML document.*

Proof. Correctness of the algorithm follows by construction. Concerning space, the arrays L and S are of size $O(d)$ and since l does not exceed d, the space requirements for storing l are $O(\log d)$. □

6.4.2 Streaming Algorithm with 3 Auxiliary Streams

In the following subsections we provide streaming algorithms for FCNS encoding which is the transformation of $\mathrm{XML}(t)$ to $\mathrm{XML}(\mathrm{FCNS}(t))$, and FCNS decoding which is the transformation of $\mathrm{XML}(\mathrm{FCNS}(t))$ to $\mathrm{XML}(t)$, see the definition in Section 6.1.2.

We are interested in computing the transformation $\mathrm{XML}(t) \rightarrow \mathrm{XML}(\mathrm{FCNS}(t))$. Our strategy is to compute the subsequence of opening tags of $\mathrm{XML}(\mathrm{FCNS}(t))$ (discussed in Subsection 6.4.2.1) and the subsequence of closing tags (discussed in Subsection 6.4.2.2) of $\mathrm{XML}(\mathrm{FCNS}(t))$ independently, and merge them afterwards (discussed in Subsection 6.4.2.3).

6.4.2.1 Computing the sequence of opening tags

First, we provide a lemma that shows that the sequence of opening tags in $\mathrm{XML}(t)$ and $\mathrm{XML}(\mathrm{FCNS}(t))$ coincide.

Lemma 38. *The opening tags in $\mathrm{XML}(t)$ are in the same order as the opening tags in $\mathrm{XML}(\mathrm{FCNS}(t))$.*

Proof. Recall Definition 15 of XML and Definition 16 of $\mathrm{XML^F}$. We will show that the following two functions XML' and $\mathrm{XML^F}'$ which, applied to the root of a tree t, generate the sequences of opening tags of $\mathrm{XML}(t)$ and $\mathrm{XML}(\mathrm{FCNS}(t))$ (without left/right annotations) are equivalent. For a tree t and nodes $x, x_1, \dots, x_n$ we define

$$\begin{aligned}
\mathrm{XML}'(x) &= x\,\mathrm{XML}'(\mathrm{children}(x)),\\
\mathrm{XML}'(x_1,\dots,x_n) &= \mathrm{XML}'(x_1)\dots\mathrm{XML}'(x_n),\\
\mathrm{XML}'(\perp) &= \epsilon,
\end{aligned}$$

and

$$\begin{aligned}
\mathrm{XML^F}'(x) &= x\,\mathrm{XML^F}'(\mathrm{fc}(x))\,\mathrm{XML^F}'(\mathrm{ns}(x)),\\
\mathrm{XML^F}'(\perp) &= \epsilon.
\end{aligned}$$

Clearly, $\mathrm{XML}'(\mathrm{root}(t))$ and $\mathrm{XML^F}'(\mathrm{root}(t))$ construct the sequences of opening tags of $\mathrm{XML}(t)$ and $\mathrm{XML}(\mathrm{FCNS}(t))$. Let $x \in t$ be any node. We prove the following statement by induction on the size of the subtree below x:

$$\mathrm{XML}'(x) = x\,\mathrm{XML^F}'(\mathrm{fc}(x)). \tag{6.4}$$

The statement is trivially true if x is a leaf, that is a tree of size 1. Let x be a non-leaf node with children $x_1, \dots, x_n$. Then

$$\begin{aligned}
x\mathrm{XML^F}'(\mathrm{fc}(x)) &= x\,x_1\,\mathrm{XML^F}'(\mathrm{fc}(x_1))\,\mathrm{XML^F}'(\mathrm{ns}(x_1)) && (6.5)\\
&= x\,\mathrm{XML}'(x_1)\,\mathrm{XML^F}'(x_2) && (6.6)\\
&= x\,\mathrm{XML}'(x_1)\,x_2\,\mathrm{XML^F}'(\mathrm{fc}(x_2))\,\mathrm{XML^F}'(ns(x_2)) && (6.7)\\
&= x\,\mathrm{XML}'(x_1)\,\mathrm{XML}'(x_2)\,\mathrm{XML^F}'(x_3) && (6.8)\\
&\dots\\
&= x\,\mathrm{XML}'(x_1)\,\dots\,\mathrm{XML}'(x_n) = x\,\mathrm{XML}'(\mathrm{children}(x)) = \mathrm{XML}'(x).
\end{aligned}$$

We used the induction hypothesis in Equation 6.5 to obtain Equation 6.6 and in Equation 6.7 to Equation 6.8. Let r denote the root of t. Then using Equation 6.4 the result follows

$$\begin{aligned}
\mathrm{XML^F}'(r) &= r\mathrm{XML^F}'(\mathrm{fc}(r))\mathrm{XML^F}'(\mathrm{ns}(r))\\
&= \mathrm{XML}'(r)\mathrm{XML^F}'(\perp) = \mathrm{XML}'(r).
\end{aligned}$$

□

Since due to Lemma 38 the subsequences of opening tags in $\mathrm{XML}(t)$ and $\mathrm{XML}(\mathrm{FCNS}(t))$ coincide, we extract the subsequence of opening tag of $\mathrm{XML}(t)$, and we annotate them with left or right as they should be in $\mathrm{XML}(\mathrm{FCNS}(t))$. Recall that an opening tag is left if it is the opening tag of a first child, otherwise it is right. Furthermore, for later use we annotate each opening tag c with $\mathrm{depth}(c)$ in t and the position in the stream $\mathrm{pos}(c)$, see Algorithm 19.

Algorithm 19 Extracting the opening tags of $\mathrm{XML}(t)$

```
Require: input stream is a well-formed XML document
 1: $d \leftarrow 0, p \leftarrow 0$
 2: $D \leftarrow L$
 3: while stream not empty do
 4:     $x \leftarrow$ next tag on stream
 5:     $p \leftarrow p + 1$
 6:     if $x$ is an opening tag $c$ then
 7:         $d \leftarrow d + 1$
 8:         write on output stream $(c_D, d, p)$
 9:         $D \leftarrow L$
10:     else {$x$ is a closing tag $\overline{c}$}
11:         $d \leftarrow d - 1$
12:         $D \leftarrow R$
13:     end if
14: end while
```

Fact 3. *Algorithm 19 is a streaming algorithm with space $O(\log N)$ that, given $\mathrm{XML}(t)$ as input, outputs on an auxiliary stream the sequence of opening tags of $\mathrm{XML}(\mathrm{FCNS}(t))$ with left/right annotations, and furthermore, annotates each tag c with $\mathrm{depth}(c)$ and $\mathrm{pos}(c)$. It performs one read pass on the input stream and one write pass on the auxiliary stream.*

6.4.2.2 Computing the sequence of closing tags

For a node v of some tree t, let $\mathrm{pos}'(v)$ and $\mathrm{pos}'(\bar{v})$ be the respective positions of the opening and closing tags of v in $\mathrm{XML}(\mathrm{FCNS}(t))$. Lemma 39 refers to the structure of the subsequence of closing tags in $\mathrm{XML}(\mathrm{FCNS}(t))$.

Lemma 39. *Let v_1, v_2 be nodes of t with $\mathrm{pos}(v_1) < \mathrm{pos}(v_2)$. Then $\mathrm{pos}'(\overline{v_2}) < \mathrm{pos}'(\overline{v_1})$ iff:*

1. *v_2 is in the subtree of v_1 in t;*
2. *or v_2 is in the subtree of a right sibling of v_2 in t.*

Proof. Suppose that either Item 1 or Item 2 is true. Note that for a node x, $\mathrm{XML}^{\mathrm{F}}(x)$ generates opening and closing tags for the entire subtree of x, and for all right siblings of x. Disregarding the annotations, we have $\mathrm{XML}^{\mathrm{F}}(v_2) = v_2 \mathrm{XML}^{\mathrm{F}}(\mathrm{fc}(v_2))\mathrm{XML}^{\mathrm{F}}(\mathrm{ns}(v_2))\overline{v_2}$, and hence $\overline{v_2}$ is preceded by all closing tags that are in the subtree of v_2 (Item 1) and all closing tags that are right siblings of v_2 or in the subtrees of right siblings of v_2 (Item 2).

We prove now that if Item 1 and Item 2 are false then $\mathrm{pos}'(\overline{v_1}) < \mathrm{pos}'(\overline{v_2})$. Suppose now that Item 1 and Item 2 are false. Let $p = \mathrm{lca}(v_1, v_2)$ where $\mathrm{lca}(x, y)$ denotes the lowest common ancestor of nodes x and y. Then $\mathrm{depth}(p) \leq \mathrm{depth}(v_1) - 2$ since otherwise Item 1 or Item 2 would be true.

Consider now $\text{XML}^{\text{F}}(p) = p\text{XML}^{\text{F}}(\text{fc}(p))\text{XML}^{\text{F}}(\text{ns}(p))\overline{p}$. If v_2 equals p then the lemma follows immediately since $\overline{v_1}$ is generated by $\text{XML}^{\text{F}}(\text{fc}(p))$ and $\overline{p}$ is generated after $\text{XML}^{\text{F}}(\text{fc}(p))$. Otherwise, let p' be the node at depth $\text{depth}(p) + 1$ that is on the path from v_1 to p. Then v_2 is a right sibling of p' or v_2 is in a subtree of a right sibling of p'. Consider $\text{XML}^{\text{F}}(p') = p'\text{XML}^{\text{F}}(\text{fc}(p'))\text{XML}^{\text{F}}(\text{ns}(p'))\overline{p'}$. Then v_1 is generated by $\text{XML}^{\text{F}}(\text{fc}(p'))$ and v_2 is generated by $\text{XML}^{\text{F}}(\text{ns}(p'))$ and this proves that $\text{pos}'(\overline{v_1}) < \text{pos}'(\overline{v_2})$. □

For computing the sequence of closing tags, we start with the sequence of opening tags of $\text{XML}(\text{FCNS}(t))$ as produced by the output of the Algorithm 19, that is, correctly annotated with left/right and with depth and position annotations. To obtain the correct subsequence of closing tags as in $\text{XML}(\text{FCNS}(t))$, we interpret the opening tags as closing tags and we sort them with a merge sort algorithm. Merge sort can be implemented as a streaming algorithm with $O(\log(N))$ passes and 3 auxiliary streams [GHS09]. For the sake of simplicity, Algorithm 20 assumes an input of length 2^l for some $l > 0$.

Algorithm 20 Merge sort

```
Require: unsorted data of length $2^l$ on stream 1
1: for $i = 0 \dots l - 1$ do
2:     copy data in blocks of length $2^i$ from stream 1 alternately onto stream 2 and stream 3
3:     for $j = 1 \dots 2^{l-i-1}$ do
4:         merge($2^i$)
5:     end for
6: end for
```

The function merge(b) reads simultaneously the next b values from stream 2 and stream 3, and merges them onto stream 1. The for loop in Line 3 of Algorithm 20 requires one read pass on stream 2, one read pass on stream 3, and one write pass on stream 1. See Figure 6.10 for an illustration.

	line 2 (copy)	line 3 (merge)
str 1:	B_1 B_2 B_3 B_4 $\cdots B_{2^{l-i}}$	B_{12} $B_{34} \cdots B_{2^{l-i-1}2^{l-i}}$
str 2:	B_1 B_3 $\cdots B_{l-i-1}$	B_1 B_3 $\cdots B_{l-i-1}$
str 3:	B_2 B_4 $\cdots$ B_{l-i}	B_2 B_4 $\cdots$ B_{l-i}

Figure 6.10: Left: Illustration of the copy operation in Line 2 of Algorithm 20. Blocks from stream 1 are copied alternately onto stream 2 and stream 3. Right: Illustration of the merge operations executed within the for loop of Line 3 of Algorithm 20. The B_i are sorted blocks. All blocks B_i and B_{i+1} are merged into a sorted block $B_{i(i+1)}$.

In order to use merge sort, we have to define a comparator function that, given two closing tags $\overline{c_1}, \overline{c_2}$ with $\text{pos}(c_1) < \text{pos}(c_2)$, decides whether $\text{pos}'(\overline{c_1}) < \text{pos}'(\overline{c_2})$. Lemma 39 states that if c_2 is in the subtree of c_1 or c_2 is in the subtree of a right sibling of c_1 then $\text{pos}'(\overline{c_2}) < \text{pos}'(\overline{c_1})$, otherwise $\text{pos}'(\overline{c_2}) > \text{pos}'(\overline{c_1})$. Therefore, a comparator has to be able to distinguish between these two situations. This, however, seems difficult in the streaming model.

To overcome this problem, instead of only defining a comparison function, we design a complete merge function in Lemma 40 that, by construction, never encounters the situation that nodes v_1, v_2 with $\mathrm{pos}(v_1) < \mathrm{pos}(v_2)$ that do not fulfill Item 1 and Item 2 of Lemma 39 are compared. The key idea is to introduce *separator* tags which we denote by new tags outside of Σ. They are initially inserted right after each closing tag of a last child u. We denote by $\overline{\overline{u}}$ the separator we introduce when seeing the last child u, and we define $\mathrm{depth}(\overline{\overline{u}}) = \mathrm{depth}(u)$.

Algorithm 21 Unsorted sequence of closing tags of $\mathrm{XML}(\mathrm{FCNS}(t))$ with separators

```
Require: input stream is a well-formed XML document
 1: $d \leftarrow 0, p \leftarrow 0$
 2: $D \leftarrow L$
 3: while stream not empty do
 4:     $x \leftarrow$ next tag on stream
 5:     $p \leftarrow p + 1$
 6:     if $x$ is an opening tag $c$ then
 7:         $d \leftarrow d + 1$
 8:         write on output stream $(\overline{c_D}, d, p)$
 9:         $D \leftarrow L$
10:     else $\{x$ is a closing tag $\overline{c}\}$
11:         if next item on stream is a closing tag then
12:             write on output stream $(\overline{\overline{c}}, d, p)$
13:         end if
14:         $d \leftarrow d - 1$
15:         $D \leftarrow R$
16:     end if
17: end while
```

Fact 4. *Algorithm 21 is a streaming algorithm with space $O(\log N)$ that, given a sequence* $\mathrm{XML}(t)$ *on a stream, computes on an auxiliary stream the sequence of closing tags* $\mathrm{XML}(\mathrm{FCNS}(t))$ *together with their separators and annotates the tags with* depth, pos, *and left/right. It performs one read pass on the input stream and one write pass on the auxiliary stream.*

We have to define the way we integrate the separators into our sorting. Let $v_1, v_2, \ldots, v_k$ be the ordered sequence of the children of some node. For the separator $\overline{\overline{v_k}}$ we ask their position among the closing tags to satisfy for each node v:

$$\mathrm{pos}'(\overline{v}) < \mathrm{pos}'(\overline{\overline{v_k}}) \quad \text{iff} \quad \mathrm{pos}'(\overline{v}) \leq \mathrm{pos}'(\overline{v_1}); \tag{6.9}$$

and for any other separator $\overline{\overline{w_k}}$:

$$\mathrm{pos}'(\overline{\overline{v_k}}) < \mathrm{pos}'(\overline{\overline{w_k}}) \quad \text{iff} \quad \mathrm{pos}'(v_k) < \mathrm{pos}'(w_k). \tag{6.10}$$

Blocks appearing in merge sort fulfill a property that we call *well-sorted*. A block B of closing tags is *well-sorted* if the corresponding tags in $\mathrm{XML}(\mathrm{FCNS}(t))$ appear in the same order, and for all $\overline{v_1}, \overline{v_2} \in B$ with $\mathrm{pos}(v_1) < \mathrm{pos}(v_2)$, all closing tags $\overline{v}$ of nodes v with $\mathrm{pos}(v_1) < \mathrm{pos}(v) < \mathrm{pos}(v_2)$ are in B as well.

In addition, for two blocks B_1, B_2 of closing tags, we say that (B_1, B_2) is a *well-sorted adjacent pair*, if B_1 and B_2 are well-sorted, for each closing tag $\overline{v_1} \in B_1$ and each closing tag

$\overline{v_2} \in B_2$ $\mathrm{pos}(v_1) < \mathrm{pos}(v_2)$ is satisfied, and furthermore, all closing tags $\overline{v}$ of nodes v with $\mathrm{pos}(v_1) < \mathrm{pos}(v) < \mathrm{pos}(v_2)$ are either in B_1 or B_2.

The following lemma shows that we can merge a well-sorted adjacent pair correctly.

Lemma 40. *Let* (B_1, B_2) *be a well-sorted adjacent pair, and let* $b_1 = B_1[p_1]$ *and* $b_2 = B_2[p_2]$ *for some* p_1, p_2*. Assume that* $\mathrm{pos}'(b) < \mathrm{pos}'(b_1)$ *and* $\mathrm{pos}'(b) < \mathrm{pos}'(b_2)$*, for all* $b \in B_1[1, p_1 - 1] \cup B_2[1, p_2 - 1]$*. Then:*

1. *If* b_1 *is a separator, or there is a separator in* B_1 *after* b_1*, then* $\mathrm{pos}'(b_1) < \mathrm{pos}'(b_2)$*;*
2. *Else if* b_2 *is a separator then:*
   (a) *if* $\mathrm{depth}(b_1) < \mathrm{depth}(b_2)$ *then* $\mathrm{pos}'(b_2) < \mathrm{pos}'(b_1)$*,*
   (b) *else* $\mathrm{depth}(b_1) = \mathrm{depth}(b_2)$ *and* $\mathrm{pos}'(b_1) < \mathrm{pos}'(b_2)$*;*
3. *Else (neither* b_1 *nor* b_2 *are separators and there is no separator in* B_1 *after* b_1*):* $\mathrm{pos}'(b_2) < \mathrm{pos}'(b_1)$.

Proof. Let (B_1, B_2) be a well-sorted adjacent pair. Let $l = \max\{i : B_1[i] \text{ is a separator}\}$. If there are no separators in B_1, let $l = 0$.

Item 1. Since B_1 is well-sorted, we only need to check that $\mathrm{pos}'(B_1[l]) < \mathrm{pos}'(B_2[1])$. Denote by u the last child that was responsible for the insertion of the separator tag $B_1[l]$. Let u' be the left-most sibling of u. Due to Equation (6.9) it suffices to show that $\mathrm{pos}'(\overline{u'}) < \mathrm{pos}'(B_2[1])$. Since the separator $B_1[l]$ indicates that the last child u has been seen, $B_2[1]$ is not in the subtree of u' or in a subtree of a right sibling of u'. Therefore, by Lemma 39 we get $\mathrm{pos}'(\overline{u'}) < \mathrm{pos}'(B_2[1])$.

Item 2. Let v denote a node in the tree with children $v_1, \dots, v_k$. First, note that the separator $\overline{\overline{v_k}}$ is initially inserted after v_k. Furthermore, between the initial position of any v_i and $\overline{\overline{v_k}}$ there are no other separators with a depth smaller than $\mathrm{depth}(\overline{\overline{v_k}})$. Therefore, it can not happen that the node b_1 is compared to a separator tag with depth smaller than $\mathrm{depth}(b_1)$.

If $\mathrm{depth}(b_2) = \mathrm{depth}(b_1)$ then b_2 is the seperator tag that was inserted after the right-most sibling of b_1. Let l be the left-most sibling of b_1. Then $\mathrm{pos}'(b_1) < \mathrm{pos}'(l)$ and therefore by Equation 6.9 we have $\mathrm{pos}'(b_1) < \mathrm{pos}'(b_2)$. If $\mathrm{depth}(b_2) > \mathrm{depth}(b_1)$ then b_2 is the separator that was introduced after a node that is either in the subtree of b_1 or in the subtree of a right sibling of b_1. Let l' denote the left-most sibling of that node. By Lemma 39 we have $\mathrm{pos}'(l') < \mathrm{pos}'(b_1)$ and hence by Equation 6.9 we have $\mathrm{pos}'(b_2) < \mathrm{pos}'(b_1)$.

Item 3. We argue that b_2 is in the subtree of b_1 or b_2 is in the subtree of a right sibling of b_1. Then, by Lemma 39, we have $\mathrm{pos}'(b_2) < \mathrm{pos}'(b_1)$. Suppose for the sake of contradiction that this is not the case. Then the separator that was introduced after the right-most sibling of b_1 must be in $B_1[p_1 + 1, k] \cup B_2[1, p_2 - 1]$, where $k = |B_1|$. Suppose that this separator was in $B_1[p_1 + 1, k]$. Then this is a contradiction since this case is treated in Item 1 of this lemma. Suppose that this separator was in $B_2[1, p_2 - 1]$. Then this is a contradiction to the assumption of the lemma that $\mathrm{pos}'(B_2[j]) < \mathrm{pos}'(b_1)$ for all $j < p_1$. □

Lemma 41. *There is a* $O(\log N)$*-pass streaming algorithm with space* $O(\log N)$ *and* 3 *auxiliary streams that computes the subsequence of closing tags of the FCNS encoding of any XML document given in the input stream.*

Proof. Using Algorithm 21, we compute on the first auxiliary stream the sequence of opening tags interpreted as closing tags with corresponding annotations, together with separators.

We show that we can do a merge sort algorithm with a merge function inspired by Lemma 40 on the first three auxiliary streams with $O(\log N)$ space and passes. For that assume that the first stream contains a sequence $(B_1, B_2, \ldots, B_M)$ of blocks of size 2^i. For simplicity we assume that M is even, otherwise we add an empty block. We alternately copy odd blocks on the second stream, and even blocks on the third stream. For a block B_{2i} that we write on the third stream, we write before each of them, the number of separators that occur in the block B_{2i-1} that was copied on the second stream.

Then we merge sequentially all pairs of blocks (B_{2k-1}, B_{2k}) for $1 \leq k \leq M/2$ using Lemma 40. Note that $(B_{2k-1}, B_{2k})_k$ are all well-sorted pairs. Let $l = \max\{i : B_{2k-1}[i]$ is a separator$\}$. Firstly, we copy elements $B_{2k-1}[1, l]$ onto auxiliary stream 1. Knowing the number of separators in B_{2k-1} allows us to perform this operation. The correctness of this step follows from Item 1 of Lemma 40. Then, we merge blocks $B_{2k-1}[l+1, 2^i]$ and B_{2k} by using the comparison function defined in Items 2 and 3 of Lemma 40. □

6.4.2.3 Merging opening and closing tags

Merging the subsequence of opening tags of $\mathrm{XML}(\mathrm{FCNS}(t))$ and the subsequence of closing tags of $\mathrm{XML}(\mathrm{FCNS}(t))$ can be done by simultaneously reading the two subsequences and performing one write pass over an auxiliary stream.

Algorithm 22 Merging the sequence of opening and closing tags

Require:

- stream 1: annotated opening tags as output by Algorithm 19
- stream 2: annotated closing tags as output by the algorithm of Lemma 21

```
1: while stream 2 not empty do
2:     $(c_1,p_1,d_1),\ldots,(c_k,p_k,d_k)(c_{k+1},p_{k+1},d_{k+1}) \leftarrow$ next $k{+}1$ annotated opening tags from stream
       1 such that $d_1 \leq d_2 \leq \cdots \leq d_k$ and $d_k > d_{k+1}$ and $(c_{k+1},p_{k+1},d_{k+1})$ is not discarded from
       the stream
3:     output $c_1 \ldots c_k$ on output stream
4:     for $i = 1 \ldots d_k - d_{k+1}$ do
5:         $(\overline{c_1},p_1,d_1),\ldots,(\overline{c_l},p_l,d_l)(\overline{\overline{c_{l+1}}},p_{l+1},d_{l+1}) \leftarrow$ next $l$ annotated closing tags from stream 2
           such that $(\overline{c_i})_{1\leq i\leq l}$ are closing tags and $\overline{\overline{c_{l+1}}}$ is a separator
6:         output $\overline{c_1} \ldots \overline{c_l}$ on output stream
7:     end for
8: end while
```

When merging the sequence of opening tags and closing tags, we have to write closing tags only between two opening tags a_m, b_{i+1} (see Figure 6.11) if $\mathrm{depth}(a_m) > \mathrm{depth}(b_{i+1})$ in t. Figure 6.11 shows the closing tags that have to be written at that moment. The sequences $\overline{a_m} \ldots \overline{a_1}$, $\overline{e}$, $\overline{d_l} \ldots \overline{d_1}$ and $\overline{c_j} \ldots \overline{c_1}$ are all separated by a separator in the sequence of closing tags. Therefore, it is enough to write the next $\mathrm{depth}(a_m) - \mathrm{depth}(b_{i+1})$ blocks of closing tags that are separated by a separator between a_m and b_{i+1}.

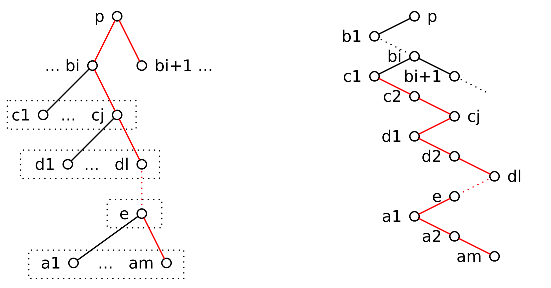


Figure 6.11: A part of a tree and its FCNS encoding. Consider nodes a_m and b_{i+1}. In the FCNS encoding, the sequence of closing tags $\overline{a_m} \dots \overline{a_1}\overline{e}\overline{d_l} \dots \overline{d_1}\overline{c_j} \dots \overline{c_1}$ has to be written in between the opening tag a_m and b_{i+1}.

Lemma 42. *Algorithm 22 merges correctly the sequence of opening tags and closing tags using space $O(\log N)$.*

Proof. First, we argue that closing tags only have to be written between consecutive opening tags a and b in the sequence of opening tags such that $\mathrm{depth}(a) > \mathrm{depth}(b)$. We have $\mathrm{XML}^F(a) = a\mathrm{XML}^F(\mathrm{fc}(a))\mathrm{XML}^F(\mathrm{ns}(a))\overline{a}$, and therefore if $\mathrm{depth}(a) < \mathrm{depth}(b)$ then b is either the first child of a or if a is a leaf then b is the next sibling of a. In both cases, there are no closing tags between a and b.

Figure 6.11 illustrates the closing tags that have to be written beween two consecutive opening tags a and b with $\mathrm{depth}(a) > \mathrm{depth}(b)$. Since closing tags at different levels are separated by a separator, it is enough to write the next $\mathrm{depth}(a) - \mathrm{depth}(b)$ blocks of closing tags that are separated by a separator between the tags a and b. □

From Fact 3, Lemma 41 and Lemma 42 we obtain Theorem 19.

Theorem 19. *There is a $O(\log N)$-pass streaming algorithm with space $O(\log N)$ and 3 auxiliary streams and $O(1)$ processing time per letter that computes on the third auxiliary stream the FCNS transformation of any XML document given in the input stream.*

Proof. Firstly, we compute according to Lemma 41 the sequence of closing tags and we store them on auxiliary stream 1. Then, by Fact 3 we extract the sequence of opening tags, and we store them on auxiliary stream 2. By Lemma 42 we can merge the tags of auxiliary stream 1 and auxiliary stream 2 correctly onto stream 3.

The space requirements of these operations do not exceed $O(\log N)$. The processing time per letter of these operations is constant. □

The algorithm described in the proof of Theorem 19 can be easily modified such that it outputs $\mathrm{XML}(\mathrm{FCNS}^{\perp}(t))$ instead of $\mathrm{XML}(\mathrm{FCNS}(t))$. We state this fact in the following.

Corollary 1. *There is a $O(\log N)$-pass streaming algorithm with space $O(\log N)$ and 3 auxiliary streams and $O(1)$ processing time per letter that computes on the third auxiliary stream the $\mathrm{FCNS}^{\perp}$ encoding of any XML document given in the input stream.*

Proof. Firstly, we use the algorithm described in Theorem 19 to compute the transformation $\mathrm{XML}(t)$ into $\mathrm{XML}(\mathrm{FCNS}(t))$. Then, with an additional read pass and an additional write pass we transform $\mathrm{XML}(\mathrm{FCNS}(t))$ into $\mathrm{XML}(\mathrm{FCNS}^{\perp}(t))$. To perform this transformation, we read the tags of $\mathrm{XML}(\mathrm{FCNS}(t))$ and output them on another stream without left/right annotations, and at the same time we insert leaves labeled with $\perp$. Such a leaf has to be inserted below internal nodes that have only a single child. The left/right annotations of the input stream allow us to recognize those nodes. Note that the transformation $\mathrm{XML}(\mathrm{FCNS}(t))$ into $\mathrm{XML}(\mathrm{FCNS}^{\perp}(t))$ requires only constant space. □

6.4.2.4 Checking Validity on the encoded form

The problem of validating trees given in their encoded form and the problem of validating binary trees are similar. We will provide intuition that basically *any* streaming algorithm that decides validity of binary trees by calling a check function upon all triplets (v, v_1, v_2) of internal nodes v with children v_1, v_2 ((v, ϵ, ϵ) for leaves) can be transformed into an algorithm that decides validity of trees given in their encoded form. We will explicitly show how to use the bidirectional 2-pass algorithm, Algorithm 16, and the one-pass algorithm, Algorithm 15, to perform this task.

To validate a node v with children $v_1, \dots, v_k$, an algorithm has to ensure that the sequence $v_1 \dots v_k$ is valid with respect to the regular expression $d(v)$. To perform such a check, an algorithm has to gather the relevant information, which is the label of v and the label of its children $v_1, \dots, v_k$, from the stream. Figure 6.12 illustrates the fact that $\overline{v_k} \dots \overline{v_1}$ forms a substring in $\mathrm{XML}(\mathrm{FCNS}^{\perp}(t))$. Suppose that the information about the labels of the children $v_1, \dots, v_k$ was available at node v_1 in $\mathrm{FCNS}^{\perp}(t)$ (in a compressed form since the number of children of a node can be large). Then we could use any algorithm validating binary trees which uses a check function as described above for our purpose: since such an algorithm relates a node to its two children, we can use this algorithm on $\mathrm{FCNS}^{\perp}(t)$ to relate a node to its left child.

Granting access to all children labels when it is required is established with the help of a finite automaton that we discuss later. Consider a left-to-right pass over $\mathrm{FCNS}^{\perp}(t)$. When seeing the sequence $\overline{v_k} \dots \overline{v_1}$, we feed it into a finite automaton. The resulting state is a compressed version of this sequence. A binary tree validity algorithm will then relate this state to the parent node. The details follow.

For a non-leaf node v, we gather the information of the children nodes $v_1, \dots, v_k$ with the help of finite automata $\mathcal{A}_1$ (for left-to-right passes) and $\mathcal{A}_2$ (for right-to-left passes).

We denote by $(\Sigma, Q, q_0, \delta, F)$ a deterministic finite automaton where Σ is its input alphabet, Q is the state set, q_0 is its initial state, $\delta : Q \times \Sigma \to Q$ is the transition function, and F is a set of final states. Furthermore, for a word $\omega = \omega_1 \dots \omega_n$ of length n, we define ω^{rev} to be ω read from right to left, that is $\omega^{\mathrm{rev}} = \omega_n \dots \omega_1$.

Lemma 43. *Let $D = (\Sigma, d, s_d)$ denote a DTD. Then there is a deterministic finite automaton $\mathcal{A}_1 = (\Sigma, Q_1, q_0^1, \delta_1, F_1)$ that for any $v \in \Sigma$ and any $v_1 \dots v_k$ in Σ^k accepts the word $v_k \dots v_1 v$ only if $v_1 \dots v_k$ fulfills the regular expression $d(v)$.*

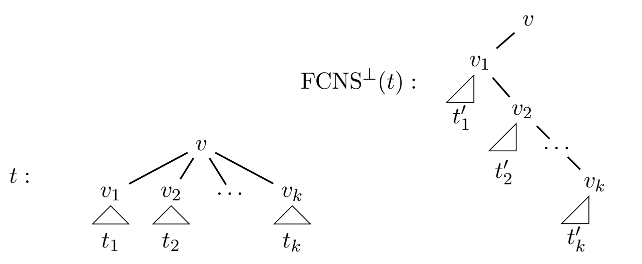


Figure 6.12: A tree t and its $\mathrm{FCNS}^\perp$ encoding. While the opening and closing tags of the children of a node v are separated by the subtrees $t_1, \dots t_k$ in $\mathrm{XML}(t)$, the closing tags of the children of v are consecutive in $\mathrm{XML}^{\mathrm{F}\perp}(t)$ in reverse order, that is $\overline{v_k}\,\overline{v_{k-1}} \dots \overline{v_2}\,\overline{v_1}$ is a substring of $\mathrm{XML}^{\mathrm{F}\perp}(t)$.

Proof. For $a \in \Sigma$, denote by A_a a deterministic finite automaton that accepts the regular expression $d(a)$. We compose the A_a as in the left illustration of Figure 6.13 to an automaton A that accepts words ω' such that $\omega' = a\omega$, $a \in \Sigma, \omega \in \Sigma^*$ if $\omega \in d(a)$. $\mathcal{A}_1$ is a deterministic finite automaton that accepts a word ω, iff ω^{rev} is accepted by A. □

Lemma 44. *Let $D = (\Sigma, d, s_d)$ denote a DTD. Then there is a deterministic finite automaton $\mathcal{A}_2 = (\Sigma, Q_2, q_0^2, \delta_2, F_2)$ that for any $v \in \Sigma$ and any $v_1 \dots v_k$ in Σ^k accepts the word $v_1 \dots v_k v$ only if $v_1 \dots v_k$ fulfills the regular expression $d(v)$.*

Proof. For $a \in \Sigma$, denote by A_a a deterministic finite automaton that accepts the regular expression $d(a)$. We compose the A_a as in the right illustration of Figure 6.13 to an automaton A that accepts words ω' such that $\omega' = \omega a$, $a \in \Sigma, \omega \in \Sigma^*$ if $\omega \in d(a)$. Then $\mathcal{A}_2$ is a deterministic version of A without ϵ transitions. □

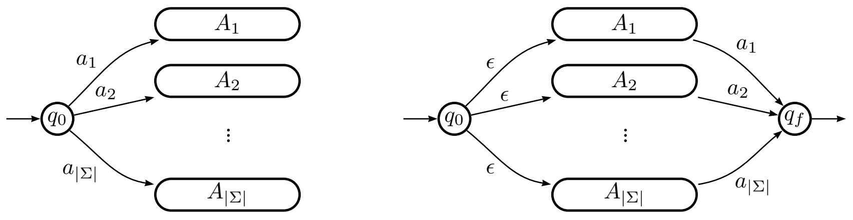


Figure 6.13: Left: Automaton A. $\mathcal{A}_1$ accepts words ω if A accepts ω^{rev}. Right: Automaton $\mathcal{A}_2$ is a version of the illustrated automaton without ϵ transitions.

We show now that by the help of automata $\mathcal{A}_1$ and $\mathcal{A}_2$, Algorithm 16 can be reused for the validation of trees given in their encoded form.

Theorem 20. *There is a bidirectional two-pass deterministic algorithm for* VALIDITY *with space $O(\log^2 N)$ and $O(\log N)$ processing time per letter when the input is given in its FCNS encoding.*

Proof. We run a modified version of Algorithm 16 on $\mathrm{XML}(\mathrm{FCNS}^{\perp}(t))$. The modifications concern the subroutine described in Algorithm 17. The modifications are different for the left-to-right pass and the right-to-left pass.

Firstly, we consider the left-to-right pass. We will annotate the closing tags of left children on the fly by states of the automaton $\mathcal{A}_1$ as described in Lemma 43. Let $v_1, \dots, v_k$ denote the children of a node v. Then the annotation of $\overline{v_1}$ is a state that we denote by $q_1(v)$. $q_1(v)$ is the resulting state of $\mathcal{A}_1$ when feeding the sequence $v_k, \dots, v_1$ into it. We describe later how to compute it on the fly. Given this annotation, we use a different implementation of the check function. For internal nodes v with first child v_1 and annotation $q_1(v)$, the check function simply computes the state $\delta_1(q_1(v), v)$ and stops if the prior state is not an accepting state. Note that by the definition of $\mathcal{A}_1$, v is valid if $\delta_1(q_1(v), v)$ is an accepting state.

We discuss now how to compute this annotation. As discussed before and illustrated in Figure 6.12, the closing tags $\overline{v_k} \dots \overline{v_1}$ of children $v_1, \dots, v_k$ of a node v form a substring. Hence, as soon as we see $\overline{v_k}$ which we can easily identify since it is a right leaf, we run the automaton $\mathcal{A}_1$ on the labels of the upcoming closing tags. We stop this procedure after $\overline{v_1}$ is read which we can identify since $\overline{v_1}$ is followed by an opening tag. Hence, when $\overline{v_1}$ is pushed on the stack (in Algorithm 17 it is actually pushed on the stack together with the opening tag of the right child of v), we can annotate it with $q_1(\overline{v_1})$.

Consider now a right-to-left pass. Note that in a right-to-left pass, closing tags are interpreted as opening tags and vice versa. This implies that a left child becomes a right child and a right child becomes a left child. Let $v_1, \dots, v_k$ denote the children of a node v. Then in a right-to-left pass, we see the sequence of opening tags $v_1, \dots, v_k$ as a substring, where v_1 is a right opening tag and $v_2, \dots, v_k$ are left opening tags. We will annotate the closing tag of the left child of v. Note that due to the exchange of the role of left and right, the left closing tag is the next sibling of v and not the first child. Since our input tree $\mathrm{FCNS}^{\perp}(t)$ is a binary tree, it is guaranteed that this node exists. The annotation is the state $q_2(v)$. $q_2(v)$ is obtained by feeding the sequence $v_1, \dots, v_k$ into the automaton $\mathcal{A}_2$, who is described in Lemma 44. The check function then computes $\delta_2(q_2(v), v)$ and stops if the resulting state is not an accepting state.

We discuss now that this annotation can be computed on the fly and it can be added correctly to the closing tag of the left child of v. The main difference to the left-to-right pass is that we compute the annotation after having pushed the children of v onto the stack and we add the annotation afterwards. Denote by v_l the left child of v in $\mathrm{FCNS}^{\perp}(t)$. Then in the right-to-left pass we see the substring $\overline{v_l} v_1 v_2 \dots v_k$. Algorithm 17 pushes $\overline{v_l}, v_1$ on the stack as soon as v_1 is seen. We then feed the sequence $v_1 v_2 \dots v_k$ into $\mathcal{A}_2$. As soon as v_k is read which can be easily identified since v_k is either a leaf of followed by a right opening tag, we annotate the left closing tag of the topmost stack item by the state $q_2(v)$.

Correctness, that is the validation of all nodes, follows then from the correctness of Algorithm 16. The automata $\mathcal{A}_1, \mathcal{A}_2$ are of constant size since we assumed that the input DTD is of constant size. Hence, the described algorithm has the same space complexity as Algorithm 16. □

By modifying the one-pass algorithm Algorithm 15 in a similar way, the following theorem can be obtained.

Theorem 21. *There is a one-pass deterministic algorithm for* VALIDITY *with space* $O(\sqrt{N}\log N)$ *and* $O(1)$ *processing time per letter when the input is given in its* $\mathrm{FCNS}^{\perp}$ *encoding.*

Proof. We reuse Algorithm 15. Concerning the modifications, the idea is the same as for the left-to-right pass of the algorithm described in the proof of Theorem 20. For all internal nodes v with children $v_1, \dots, v_k$, we compress the sequence $v_1, \dots, v_k$ into a state $q_1(v)$ of the finite automaton $\mathcal{A}_1$ who is described in Lemma 43. $q_1(v)$ is obtained by feeding $v_k \dots v_1$ into $\mathcal{A}_1$ which can be done since $\overline{v_k} \dots \overline{v_1}$ forms a substring of the input XML sequence. We annotate the closing tag of v_1 with this state. The check routine is modified in the same way as in the proof of Theorem 20: only if $\delta_1(q_1(v), v)$ is an accepting state then v is valid, otherwise the check routine aborts and the algorithm reports an invalid node. The correctness of Algorithm 15 ensures the validation of all nodes. $\square$

Applying the bidirectional algorithm of Theorem 20 on the encoded form $\mathrm{XML}(\mathrm{FCNS}^{\perp}(t))$, we obtain that validity of general trees can be decided memory efficiently in the streaming model with auxiliary streams.

Corollary 2. *There is a bidirectional* $O(\log N)$*-pass deterministic streaming algorithm for* VALIDITY *with space* $O(\log^2 N)$*,* $O(\log N)$ *processing time per letter, and* 3 *auxiliary streams.*

Proof. We perform the transformation $\mathrm{XML}(t)$ into $\mathrm{XML}(\mathrm{FCNS}^{\perp}(t))$ with the algorithm stated in Corollary 1. Then, we run the two-pass bidirectional algorithm of Theorem 20 on $\mathrm{XML}(\mathrm{FCNS}^{\perp}(t))$ and the result follows. $\square$

Note that this result only holds for the validation of DTDs. Nothing is known about the validation of more powerful validity schemas such as extended DTDs or XML Schema if access to auxiliary streams is granted.

6.4.3 Decoding

In the following, we present streaming algorithms for FCNS decoding, that is, given $\mathrm{XML}(\mathrm{FCNS}(t))$ of some tree t, output $\mathrm{XML}(t)$. These results complement our results on the computation of the FCNS encoding and are mainly of theoretical interest to us. There are, however, potential applications: It may have advantages to store the FCNS encoding of an XML file instead of the XML file itself. Then validity could be efficiently ensured by Algorithm 20 with two bidirectional passes and space $O(\log^2 N)$. The original document could then be *exported* by means of the algorithms that we present in this section. The applicability of this approach is left open.

We start with a non-streaming algorithm, Algorithm 23 performing this task.

We first discuss the correctness of Algorithm 23. We show that the algorithm run on $\mathrm{XML}(\mathrm{FCNS}(t))$ computes the function $\mathrm{dec}(\mathrm{root}(t))$ which we define in the following. Let t be a tree and let $x \in t$ be a node, then

$$\begin{aligned} \mathrm{dec}(x) &= x\,\mathrm{dec}(\mathrm{fc}(x))\,\overline{x}\,\mathrm{dec}(\mathrm{ns}(x)), \\ \mathrm{dec}(\perp) &= \epsilon. \end{aligned}$$

Algorithm 23 Offline algorithm for FCNS decoding

```
1:  for $i = 1 \to 2N$ do
2:      if $X[i]$ is an opening tag then
3:          write $X[i]$
4:          if $X[i]$ does not have a left subtree then {$X[i]$ is a leaf}
5:              write $\overline{X[i]}$
6:          end if
7:      else if $X[i]$ is a left closing tag then {See Figure 6.14}
8:          let $p$ be the parent node of $X[i]$
9:          write $\overline{p}$
10:     end if
11: end for
```

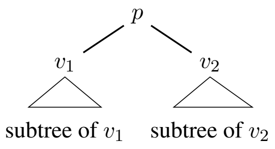


Figure 6.14: The main difficulty of the FCNS decoding is to write the closing tag of a node p when the closing tag of its left child is seen. This is difficult when the subtrees of v_1 and v_2 are large.

The only difference between dec and $\mathrm{XML}^{\mathrm{F}}$ is that for some non-leaf node x, $\mathrm{dec}(x)$ outputs $\overline{x}$ between the recursive calls to $\mathrm{dec}(\mathrm{fc}(x))$ and $\mathrm{dec}(\mathrm{ns}(x))$ while $\mathrm{XML}^{\mathrm{F}}$ outputs $\overline{x}$ at the very end. Algorithm 23 computes dec since it ignores the closing tags of the FCNS encoding and it inserts closing tags when we do a transition from the left child to a right child, that is between the recursive calls to $\mathrm{dec}(\mathrm{fc}(x))$ and $\mathrm{dec}(\mathrm{ns}(x))$. We show in Lemma 45 that $\mathrm{dec}(\mathrm{root}(t))$ produces the same output as $\mathrm{XML}(\mathrm{root}(t))$.

Lemma 45. $\mathrm{dec}(\mathrm{root}(t)) = \mathrm{XML}(\mathrm{root}(t))$.

Proof. We will prove that for a node $x \in t$ the following is true

$$\mathrm{XML}(x) = x\,\mathrm{dec}(\mathrm{fc}(x))\,\overline{x}. \tag{6.11}$$

The proof is by induction on the height of the subtree below x and is similar to the proof of Lemma 38. The claim is obvious for leaves. Let x be a node and let $v_1, \dots, v_n$ denote the children of x. Then

$$\begin{aligned}
x\,\mathrm{dec}(v_1)\,\overline{x} &= x\,v_1\,\mathrm{dec}(\mathrm{fc}(v_1))\,\overline{v_1}\,\mathrm{dec}(v_2)\,\overline{x} && (6.12)\\
&= x\mathrm{XML}(v_1)\,v_2\,\mathrm{dec}(\mathrm{fc}(v_2))\,\overline{v_2}\,\mathrm{dec}(v_3)\,\overline{x} && (6.13)\\
&= x\mathrm{XML}(v_1)\,\mathrm{XML}(v_2)\,v_3\,\mathrm{dec}(\mathrm{fc}(v_3))\,\overline{v_3}\,\mathrm{dec}(v_4)\,\overline{x} && (6.14)\\
&\dots\\
&= x\,\mathrm{XML}(\mathrm{children}(x))\,\overline{x} = \mathrm{XML}(x),
\end{aligned}$$

where we used the induction hypothesis in Equation 6.12 to obtain Equation 6.13, and in Equation 6.13 to obtain Equation 6.14. Since the root node r of the tree t does not have a next sibling, the result follows using Equation 6.11

$$\operatorname{dec}(r) \quad = \quad r \operatorname{dec}(\operatorname{fc}(r))\, \overline{r} \operatorname{dec}(\operatorname{ns}(r)) = \operatorname{XML}(r).$$

□

Corollary 3. *Algorithm 23 is an offline algorithm that computes* $\operatorname{XML}(t)$ *given* $\operatorname{XML}(\operatorname{FCNS}(t))$.

We describe how this algorithm can be converted into a streaming algorithm. For an opening tag $X[i]$, checking the condition in Line 4 can easily be done by investigating $X[i+1]$. If $X[i+1]$ is a right opening tag or equals $\overline{X[i]}$, $X[i]$ does not have a left subtree. The difficulty in converting this algorithm into a streaming algorithm is in Line 8, it is difficult to keep track of opening tags until the respective closing tags of their left children are seen, and indeed, this cannot be done with sublinear space in one pass, see Theorem 24.

In the following, we present a streaming algorithm that performs one pass over the input, but two passes over the output, and uses $O(\sqrt{N \log N})$ space, and a streaming algorithm that performs $O(\log N)$ passes over the input and 3 auxiliary streams using $O(\log^2(N))$ space.

6.4.3.1 One read-pass and two write-passes

We read blocks of size $\sqrt{N \log N}$ and execute Algorithm 23 on each block. In Lemma 35 we showed that in any block there is at most one left closing tag for which the parent's opening and closing tag are not in that block. Hence per block there is at most one left closing tag for which we can not obtain the label of the parent node. We call this closing tag *critical*. In this case we write a *dummy symbol* on the output stream that will be overwritten by the parent's closing tag in the second pass. The closing tag of the parent node will arrive in a subsequent block, and it can easily be identified as this since it is the next closing tag arriving at a depth -1 of the critical closing tag. We store it upon its arrival in our random access memory. Since there is at most one critical closing tag per block and we have a block size of $\sqrt{N \log N}$, we have to recover at most $O(\sqrt{N / \log N})$ parent nodes. At the end of the pass over the input stream we have recovered all closing tags of parent nodes for which we wrote dummy symbols on the output stream. In a second pass over the output stream we overwrite the dummy symbols by the correct closing tags.

The space complexity uses Lemma 35 that was already applied in Section 6.3.1.

Theorem 22. *There is a streaming algorithm using* $O(\sqrt{N \log N})$ *space and* $O(1)$ *processing time per letter which performs one pass over the input stream containing* $\operatorname{XML}(t)$ *and two passes over the output stream onto which it outputs* $\operatorname{XML}(\operatorname{FCNS}(t))$.

6.4.3.2 Logarithmic number of passes

Again, we use the offline Algorithm 23 as a starting point for the algorithm we design now. For coping with the problem that it is hard to remember all opening parent tags when their corresponding closing tag ought to be written on the output, we always write *dummy symbols* on the output stream for all parent closing tags. The crux then is the following observation:

Fact 5. *Let* $\overline{c_1}_{\mathrm{L}} \dots \overline{c_N}_{\mathrm{L}}$ *be the subsequence of closing tags of left children of* $\mathrm{XML}(\mathrm{FCNS}(t))$. *Then the sequence* $\overline{p_1} \dots \overline{p_N}$ *is a subsequence of* $\mathrm{XML}(t)$ *where* p_i *is the parent node of* c_i *in* $\mathrm{FCNS}(t)$.

We apply a modified version of our bidirectional two-pass Algorithm 16 to recover the missing tags. Instead of checking validity, once the check function is called in Algorithm 17 with variables (a, b, c), we output the parent label a onto an auxiliary stream, annotated with $\mathrm{pos}(b)$. We do the same in a reverse pass over the input stream counting positions from $2N$ downwards to 1. In so doing, the auxiliary stream contains all parent labels for which dummy symbols are written on the output stream.

Fact 5 shows that it is enough to sort by means of two further auxiliary streams the auxiliary stream with respect to the annotated position of the closing tags of the left children of these nodes. In a last pass we insert the parent closing tags into the output stream.

Theorem 23. *There is a* $O(\log N)$*-pass streaming algorithm with space* $O(\log^2 N)$ *and* $O(\log N)$ *processing time per letter and* 3 *auxiliary streams that computes on the third auxiliary stream the FCNS decoding of any FCNS encoded document given in the input stream.*

6.5 Lower Bounds for FCNS Encoding and Decoding

6.5.1 Lower bound for FCNS encoding

Let $x \in \Sigma^n$. We define a family of hard instances $t(x)$ of length $N = \Theta(n)$ for the computation of $\mathrm{XML}(\mathrm{FCNS}(t(x)))$ given $\mathrm{XML}(t(x))$ as in Figure 6.15.

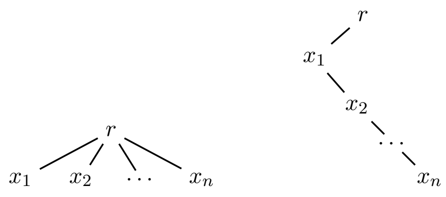


Figure 6.15: Left: hard instance. Right: its FCNS encoded form.

It is easy to see that computing the sequence of closing tags in the FCNS encoding requires to reverse a stream. Let t be a hard instance. Then $\mathrm{XML}(t) = r x_1 \overline{x_1} x_2 \overline{x_2} \dots x_n \overline{x_n}\,\overline{r}$, and $\mathrm{XML}(\mathrm{FCNS}(t)) = r_{\mathrm{L}} x_{1\mathrm{L}} x_{2\mathrm{R}} \dots x_{n\mathrm{R}} \overline{x_{n\mathrm{R}}}\,\overline{x_{n-1\mathrm{R}}} \dots \overline{x_{2\mathrm{R}}}\,\overline{x_{1\mathrm{L}}}\,\overline{r_{\mathrm{L}}}$. Since writing the closing tags on the output stream can only start after reading x_n, we deduce that memory space $\Omega(n)$ is required in order to store all previous tags $x_1, \dots, x_{n-1}$.

Fact 6. *Every randomized streaming algorithm for FCNS encoding that performs one pass on the input stream and one pass on the output stream with bounded error requires* $\Omega(N)$ *space.*

6.5.2 Lower bound for FCNS decoding

We define now a family of hard instances of length $N = \Theta(n)$ for decoding a FCNS encoded tree. Let $X \in \{0,1\}^n, Y \in \{0,1\}^n$ and $K \in [n]$ be uniformly distributed random variables.

Let $x \leftarrow X, y \leftarrow Y$ and $k \leftarrow K$. Denote by $t'(y)$ an arbitrary but fixed two-ranked tree with n nodes that are labeled by $y_1, \dots, y_n$, and let $s'(y)$ be the decoded form of $t'(y)$. We define then the hard instance $t(x, y, k)$ and its decoded form $s(x, y, k)$ as in Figure 6.16.

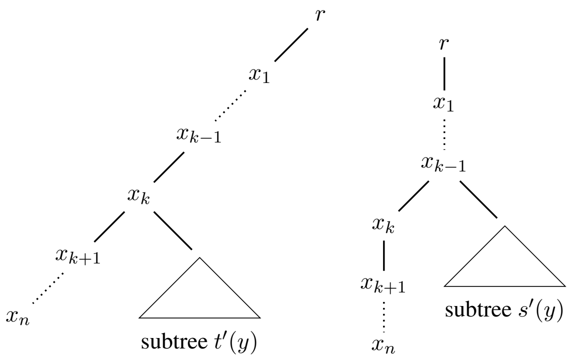


Figure 6.16: Left: hard instance $t(x, y, k)$ in FCNS form. Right: its decoded form $s(x, y, k)$. $s'(y)$ is the decoded form of subtree $t'(y)$.

Then we have

$$\begin{aligned} \text{XML}(t(x,y,k)) &= r x_{1\text{L}} \dots x_{n\text{L}} \overline{x_{n\text{L}}} \dots \overline{x_{k+1\text{L}}} \text{XML}(t'(y)) \overline{x_{k\text{L}}} \dots \overline{x_{1\text{L}}} \overline{r_{\text{L}}}, \text{ and} \\ \text{XML}(s(x,y,k)) &= r x_1 \dots x_n \overline{x_n} \dots \overline{x_k} \text{XML}(s'(y)) \overline{x_{k-1}} \dots \overline{x_1} \overline{r}. \end{aligned}$$

The crucial difference between $t(x, y, k)$ and $s(x, y, k)$ is that the subtree $t'(y)$ is attached to node x_k in $t(x, y, k)$ while the subtree $s'(y)$ is attached to node x_{k-1} in $s(x, y, k)$. As a consequence, $\text{XML}(t'(y))\overline{x_{k\text{L}}}$ is a substring of $\text{XML}(t(x, y, k))$ while $\overline{x_k}\text{XML}(s'(y))$ is a substring of $\text{XML}(s(x, y, k))$. We will see that it is difficult to write the substring $\overline{x_k}\text{XML}(s'(y))$ with sublinear space.

Note that $\overline{x_k}$ has to be written before $s'(y)$ (which is the decoded version of $t'(y)$). However, $\overline{x_k}$ appears after the subtree $t'(y)$ in $\text{XML}(t(x, y, k))$. Hence, we either have to store the entire subtree $t'(y)$ in memory using $\Theta(n)$ space, or we have to infer $\overline{x_k}$ from the opening tag x_{kL} in $t(x, y, k)$. Since, however, k is not known to the algorithm before seeing the subtree $t'(y)$, this can not be done with sublinear memory.

We start with the definition of a one-way three-party communication game that we denote by IndexCopy. Let the input be uniformly distributed random variables $X \in \{0, 1\}^n, Y \in \{0, 1\}^n$ and $K \in [n]$. They are given to the three parties Alice, Bob and Charlie as follows:

$$\begin{array}{ccccc} \text{Alice} & \xrightarrow{M_A} & \text{Bob} & \xrightarrow{M_B} & \text{Charlie} \\ X & & K, X[K+1, n], Y & & X[1, K] \end{array}$$

The common goal of the parties is to write the sequence $X[K]Y$ on a shared output stream. Firstly, Alice is allowed to write, followed by Bob and then Charlie. The communication is one-way: Alice sends message M_A to Bob, and then Bob sends message M_B to Charlie.

From the presentation of the family of hard instances for FCNS decoding, it is easy to see that an algorithm for FCNS decoding that makes one pass over the input stream and one pass over the output stream can be used to obtain a communication protocol for INDEXCOPY. We state this as a fact.

Fact 7. *A streaming algorithm for FCNS decoding that makes one pass over the input stream and one pass over the output stream with space s serves as a communication protocol for* INDEXCOPY *with communication cost $O(s)$.*

We will prove now that a communication protocol for INDEXCOPY has communication cost $\Omega(N)$.

Lemma 46. *Every possibly randomized communication protocol for* INDEXCOPY *with error $O(1/N)$ has communication cost $\Omega(N)$.*

Proof. Let P be a (possibly randomized) communication protocol such that the parties output $X[K]Y$ on the shared output stream with error $\epsilon = \frac{1}{32n}$ on any input. We will prove now that P has communication cost $\Omega(n)$.

By Yao's minimax principle, there is a deterministic communication protocol P_d with distributional error at most ϵ that has the same communication complexity as P. Suppose that Alice's message in P_d is at most of length $n/100$ bits. We will show that under this assumption, for a particular input, Bob has to send a message of length $\Omega(n)$ bits which proves the theorem.

Since P_d has distributional error $\epsilon = \frac{1}{32n}$, we obtain by the Markov inequality:

$$\Pr_{x \leftarrow X}[\text{error} \geq 1/(16n) \,|\, X = x] \leq \frac{\epsilon}{1/(16n)} = \frac{1}{2}.$$

Therefore, there are at least $(1/2)2^n = 2^{n-1}$ values x for which the protocol errs with probability at most $1/(16n)$. Denote this set of x values by U. Furthermore, again by the Markov inequality:

$$\forall x \in U : \Pr_{k \leftarrow K}[\text{error} \geq 1/4 \,|\, X = x, K = k] \leq \frac{\frac{1}{16n}}{1/4} = \frac{1}{4n}.$$

Therefore, for any $x \in U$ and any $k \in [n]$, the protocol errs with probability less than $1/4$.

Consider an input of Alice x coming from U that is different from $x^0 = 0 \dots 0$ and $x^1 = 1 \dots 1$. Then Alice cannot write the bit $X[K]$ on the output stream. If on input x Alice writes deterministically 0 (or 1) then there is a value of K such that Alice wrote the wrong bit. The error on x would then be greater than $1/(16n)$ contradicting the fact that $x \in U$. Therefore, for all $x \in U \setminus \{x^0, x^1\}$ it is either Bob or Charlie who writes the bit $X[K]$.

We will argue now that there is at least one $x \in U$ and $k \in [n]$ such that Charlie has to output the bit $X[K]$.

Since the maximal message length of Alice is $n/100$ bits, there is a subset $U' \subseteq U$ with $|U'| \geq \frac{|U|}{2^{n/100}} = 2^{n-1-n/100}$ such that for all $x \in U'$, the message M_A sent by Alice is the same. Denote this message by m_A.

Using a technique of [MMN10], we will show now that there are $x_1, x_2 \in U'$ and $k \in [n]$ such that $x_1[k] \neq x_2[k]$ and $x_1[k+1, n] = x_2[k+1, n]$. This can be seen by building the following two-ranked tree: For each $x \in U'$ the tree has exactly one leaf at depth n that is labeled by x. All edges of the tree are labeled by bits 0 or 1. The sequence of labels of the edges of a path from a leaf to the root then equals the label of the leaf. An example for such a tree is provided in Figure 6.17.

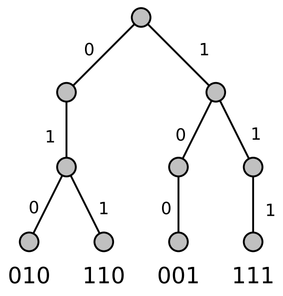


Figure 6.17: Organizing the set $\{010, 110, 001, 111\}$ in a two-ranked tree.

By construction of the tree, the labels of leaves that have a common ancestor at depth i have the same suffix of length i. Consider now an inner node v of this tree with two children nodes. Such an inner node exists since the two-ranked tree has depth n and contains $|U'| \geq 2^{n-1-n/100}$ leaves. Let k be its depth. Furthermore, let x_1 be the label of an arbitrary leaf connected to the left child of v, and let x_2 be the label of an arbitrary leaf connected to the right child of v. Then $x_1[k+1,n] = x_2[k+1,n]$ and $x_1[k] \neq x_2[k]$ as desired.

Since $x_1, x_2 \in U'$, the protocols errs with probability at most $1/4$ if $X \in \{x_1, x_2\}$ and $K = k$. Note that on both inputs x_1 and x_2, Bob has the same suffix $X[K+1,n]$ since $x_1[k+1,n] = x_2[k+1,n]$. Furthermore, Bob receives the same message m_A of Alice. Hence, Bob can not distinguish between the two events $X = x_1$ and $X = x_2$ if $K = k$. Since $x_1[k] \neq x_2[k]$, Bob can not output the bit $X[K]$ since this would lead to an error larger than $1/4$.

Therefore, in this setting Charlie has to output the bit $X[K]$. This also requires that subsequently Charlie outputs Y. Since Y is independent of the conditioning $X = x_1$ (or x_2) and $K = k$ and Charlie has no information about Y, we deduce that Charlie can learn Y from Bob's message M_B with error at most $3/4$.

Fix the input distribution $X \in \{x_1, x_2\}$, $K = k$ and Y is chosen uniformly at random. Then

$$\mathrm{H}(M_B) \geq \mathrm{H}(M_B) - \mathrm{H}(M_B \,|\, Y) = \mathrm{I}(M_B : Y),$$

where H is the entropy function and $\mathrm{I}(M_B : Y)$ denotes the mutual information between M_B and Y. Furthermore,

$$\mathrm{I}(M_B : Y) = \mathrm{H}(Y) - \mathrm{H}(Y \,|\, M_B) = n - \mathrm{H}(Y \,|\, M_B).$$

By the Fano Inequality, we obtain

$$\mathrm{H}(Y \,|\, M_B) \leq 1 + 1/4n.$$

This implies that $\mathrm{I}(M_B : Y) \geq 3/4n - 1$ and $\mathrm{H}(M_B) \in \Omega(n)$ which in turn implies that the average message length is $\Omega(n)$. □

Finally, we state our space lower bound for streaming algorithm for FCNS decoding that make one pass over the input stream and one pass over the output stream.

Theorem 24. *Every randomized streaming algorithm for FCNS decoding that makes one pass over the input stream and one pass over the output stream with error probability $O(1/N)$ requires space $\Omega(N)$.*

Proof. The proof is by contradiction. Suppose that there is such a streaming algorithm with space $o(N)$. Then, by Fact 7 there is a communication protocol for INDEXCOPY with communication cost $o(N)$. This, however, is a contradiction to Lemma 46 that states that such a communication protocol has communication cost at least $\Omega(N)$. □

Chapter 7

Budget Error-Correcting under Earth-Mover-Distance

Before presenting our results on the EMD k-Budget Error-Correcting problem, we recapitulate briefly the problem. For a given grid length $\Delta > 0$, Alice has n points $x = \{x_1, \ldots, x_n\} \in [\Delta]^d$ and Bob has n points $y = \{y_1, \ldots, y_n\} \in [\Delta]^d$ on the d-dimensional grid $[\Delta]^d$. We consider the one-way communication setting: Alice sends a message M to Bob, and using this message, Bob changes his points y to y^* such that the Earth-Mover-Distance $\mathrm{EMD}(x, y^*)$ is minimized. The quality of the adjustment is measured as follows. For a given parameter k that Alice and Bob are aware of, Bob aims to find a y^* such that

$$\mathrm{EMD}(x, y^*) \leq C \min_{\tilde{y} \in \mathcal{N}_k(y)} \mathrm{EMD}(x, \tilde{y}),$$

where $\mathcal{N}_k(y)$ is the set of point sets that can be obtained from y by moving at most k points to a different location. We are hence comparing our adjustment to the best adjustment that can be obtained by moving at most k points. For a given approximation factor C, the goal is to minimize the message length $|M|$.

In Section 7.1, we discuss a protocol that achieves an $O(d)$-approximation, and in Section 7.2, we show an almost matching lower bound for protocols that achieve an $O(d)$-approximation.

7.1 Upper Bound

Intuition. We illustrate our protocol in the one dimensional case. Let $\Delta > 0$ be an integer. Given Alice's input $x = \{x_1, \ldots, x_n\} \in [\Delta]^n$ and Bob's input $y = \{y_1, \ldots, y_n\} \in [\Delta]^n$ on the one-dimensional grid $[\Delta]$, the optimal solution will return a set of k pairs of points $\{(u_1, v_1), \ldots, (u_k, v_k)\}$ $(u_i \in x, v_i \in y)$ so that if Bob moves point v_i to u_i for all $i \in [k]$, the EMD between Alice's point set x and Bob's modified point set $y_{\mathtt{OPT}}$ is minimized. Intuitively, we can view the k pairs of points as the top k edges of a perfect matching between x and y. Our objective is to report those edges.

For the sake of a simple presentation, we assume that $\Delta = 2^L$ for some integer L. Firstly, we employ an idea that was already used in [IN03]. Alice builds a hierarchical partition (we later call it *pyramid arrays*) $\mathrm{PA}(x) = \{\mathrm{PA}_0(x), \mathrm{PA}_1(x), \ldots, \mathrm{PA}_L(x)\}$ for x, where $\mathrm{PA}_i(x)$ is an array containing 2^i elements, and the j-th element of $\mathrm{PA}_i(x)$ contains the number of points in x that fall into the interval $\big((j-1)\cdot 2^{L-i}, j\cdot 2^{L-i}\big]$. This partitioning is illustrated in Figure 7.1. Bob builds a similar hierarchical partition $\mathrm{PA}(y)$ for y.

Consider two points u, v $(u \in x, v \in y)$ such that $2^{L-r} \leq \|u-v\|_2 < 2^{L-r+1}$. Then u and v lie in cells with different indices in $\mathrm{PA}_\ell(x)$ and $\mathrm{PA}_\ell(y)$ for all $r \leq \ell \leq L$, and hence (u, v) contributes with 2 to $\|\mathrm{PA}_\ell(x) - \mathrm{PA}_\ell(y)\|_1$ for $r \leq \ell \leq L$. On the other hand, it is very likely that u and v lie in cells with the same index in $\mathrm{PA}_\ell(x)$ and $\mathrm{PA}_\ell(y)$ for all $0 \leq \ell < r$. Hence, (u, v) does not contribute to $\|\mathrm{PA}_\ell(x) - \mathrm{PA}_\ell(y)\|_1$ for $0 \leq \ell < r$. This observation leads to the following quite natural approach to our problem: Firstly, find the largest ℓ such that $\|\mathrm{PA}_\ell(x) - \mathrm{PA}_\ell(y)\|_1 \leq 2\alpha k$ for some small constant $\alpha > 1$. Then, Alice sends an encoding of $\mathrm{PA}_\ell(x)$ to Bob, from which Bob can compute a set of edges that approximate the k longest ones. There are two difficulties to consider.

1. For a pair of points (u, v) with $\|u-v\|_2 < 2^{L-r}$, u and v may lie in cells with different indices in $\mathrm{PA}_\ell(x)$ and $\mathrm{PA}_\ell(y)$ for an $\ell \leq r$. In this case we may introduce a *false positive* into the relocation.

2. For a pair (u, v) $(u \in x, v \in y)$, Bob can only learn from $\mathrm{PA}_\ell(x)$ that u lies in the interval $\big((j-1)\cdot 2^{L-\ell}, j\cdot 2^{L-\ell}\big]$. Hence, Bob does not know the exact location of u to where he should relocate his corresponding point v.

To handle the first problem, we perform a random shift of all points in x and y. In doing so, we guarantee with that there are not many false positives with good probability. This is a standard technique and was used before, e.g., in [Ind07].

To handle the second problem, we introduce a constant redundancy factor $\alpha > 1$ in the algorithm. That is, Alice sends a message to Bob so that Bob is able to relocate αk $(> k)$ points (if needed). Now, since Bob cannot decide the exact location that he should relocate a point to, he simply moves the point to an arbitrary location in the corresponding interval. Such an operation may introduce an additional error, but since the optimal solution can only relocate k points, we can charge the errors that we make when relocating each point to the error that the optimal solution has to make on the (at least) $(\alpha-1)k$ points that it is unable to relocate. Some extra difficulties come from the interplay of these two problems: It is for instance possible that the relocation of a false positive will again introduces some error. We hence have to carefully design a charging scheme such that those errors can also be charged to the error that the optimal solution makes on the $(\alpha-1)k$ points that it fails to recover.

Communication Protocol. In order to present our communication protocol, we require a formal definition of the pyramid arrays.

Definition 20 (d-Dimensional Pyramid Arrays)**.** *Let $\Delta > 0$ be an integer. Let $x = \{x_1, \ldots, x_n\} \subseteq [\Delta]^n$ be a set of n points on the d-dimensional grid $[\Delta]^d$, and assume that $\Delta = 2^L$. We define the* d-dimensional pyramid arrays *of x to be a set of arrays*

$\mathrm{PA}(x) = \{\mathrm{PA}_0(x), \dots, \mathrm{PA}_L(x)\}$, *where* $\mathrm{PA}_l(x)$*'s are constructed as follows.*

$$\begin{aligned} \textit{for } 0 \le l < L : \mathrm{PA}_l(x) &= (\mathrm{PA}_l^i)_{i \in [2^l]^d} \textit{ with } \mathrm{PA}_l^i = \sum_{s \in \{0,1\}^d} \mathrm{PA}_{l+1}^{2i+s}, \\ \mathrm{PA}_L(x) &= (\mathrm{PA}_L^i)_{i \in [\Delta]^d} \textit{ with } \mathrm{PA}_L^i = |\{j \mid x_j = i\}|. \end{aligned}$$

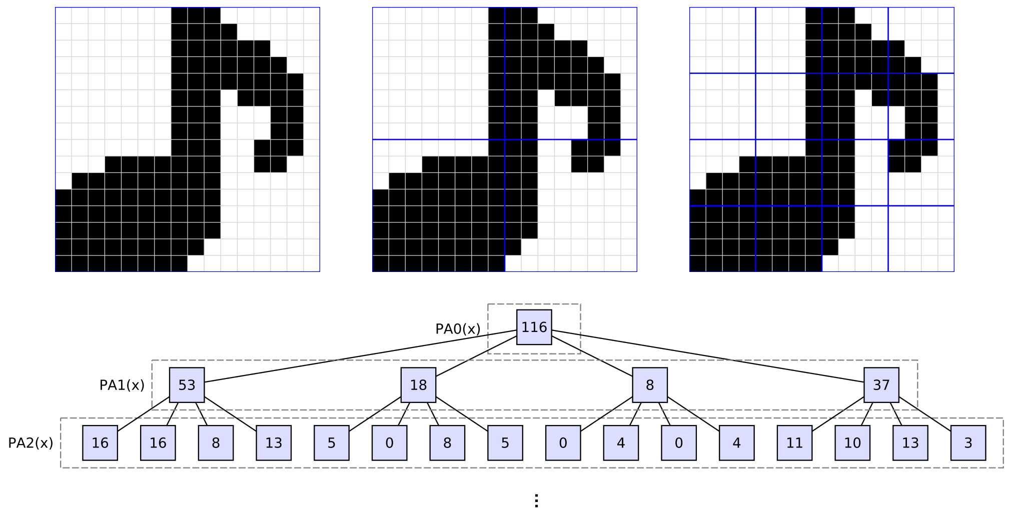


Figure 7.1: Here, we can see a point set x of 116 points in the domain $[16]^2$, and the first 3 levels of the pyramid array of x. $\mathrm{PA}(x)$ has tree structure where the sum of the labels of the children of an internal node is the label of the node.

The d-dimensional pyramid arrays can naturally be seen as a tree with the coordinates of $\mathrm{PA}_L(x)$ as leaves. Each coordinate r of $\mathrm{PA}_\ell(x)$ $(0 \le \ell \le L-1)$ corresponds to an internal node of the tree, and the value of coordinate r is the sum of the values of all leaves in the subtree rooted at r. For a leaf u, let $r_\ell(u)$ be the internal node at level ℓ such that u is in the subtree rooted at node $r_\ell(u)$.

Given two arrays A, B of the same length, we define $\|A - B\|_1 = \sum_i |A[i] - B[i]|$. For the upper bound we need an encoding scheme for arrays with certain properties. This encoding is stated in Lemma 47.

Lemma 47. *Let n, N, t be positive integers and let $A, B \in [n]^N$. Alice has A and Bob has B. Then there is a one-way two-party communication protocol with a message of length $O(\log(nN)\log(\frac{1}{\epsilon}) + t\log(n))$ and error probability at most ϵ such that*

- *if $\|A - B\|_1 \le t$ then Bob outputs A, and*
- *if $\|A - B\|_1 > t$ then Bob outputs $\perp$.*

The computation time of Alice and Bob is $O(N \operatorname{polylog}(nN)\log(\frac{1}{\epsilon}) + N^2\log(n)^2)$.

Proof. Alice sends a message $M = (M_1, M_2)$ composed of two messages M_1 and M_2 to Bob.

- Message M_1: M_1 is a sketch of A that allows Bob to estimate the l_1 distance $\|A - B\|_1$. To this end, we borrow an idea from [FKSV02].

  Lemma 48 ([FKSV02]). *Let $A, B \in \{0, \dots, n\}^N$. For some small constants $\epsilon, \delta > 0$ there is a sketch $s(A, \epsilon, \delta)$ of A such that $|s(A, \epsilon, \delta)| \in O(\log(Nn)\log(1/\epsilon)/\delta^2)$ bits, and using $s(A, \epsilon, \delta)$ and B an approximation a to the l_1 distance of x and y can be computed such that*

  $$(1 - \delta)\|A - B\|_1 \leq a \leq (1 + \delta)\|A - B\|_1$$

  with probability at least $1 - \epsilon$. The time complexity for encoding and decoding is $O(N \operatorname{polylog}(nN)\log(\frac{1}{\epsilon})\frac{1}{\delta^2})$.

  Alice sets $M_1 = s(A, \epsilon, 1/2)$. Then $|M_1| \in O(\log(nN)\log(\frac{1}{\epsilon}))$.

- Message M_2: M_2 is the error-correcting part of a Reed-Solomon code.

  Lemma 49 (Reed-Solomon Code [HP03, WB86]). *For a message Q of l elements from $GF(2^r)$, there is a systematic erasure code of $l + 2d + 1 < 2^r$ elements which can be used to recover Q if there are at most d lost elements in the coding. Moreover, the code is of the form (Q, G), where G is the "error-correcting" part consisting of $2d$ elements. The encoding and decoding time of the code is $O((l + d)^2)$.*

  Let $\bar{x} = \{\bar{x}_1, \dots, \bar{x}_N\}$ and $\bar{y} = \{\bar{y}_1, \dots, \bar{y}_N\}$ be the $N(\log(n)+1)$ binary representation of x and y, respectively, such that for each $i : \bar{x}_i, \bar{y}_i \in \{0,1\}^{\log(n)+1}$. Note that if $\|x - y\|_1 \leq t$ then $\sum_{i=1}^{N} \|\bar{x}_i - \bar{y}_i\|_1 \leq t(\log(n) + 1)$.

  We set $M_2 = G$, where $(\bar{x}, G)$ is the message from Lemma 49 with the following parameters: $r = 1, d = 3t(\log(n) + 1)$. Then $|M_2| \in O(t \log n)$.

Bob receives the message $M = (M_1, M_2)$. Firstly, using M_1 and according to Lemma 48, Bob computes a value A such that $1/2\|x - y\|_1 \leq A \leq 3/2\|x - y\|_1$. If $A > 3/2t$ then Bob outputs $\perp$ since this implies that $\|x - y\|_1 > t$ with probability $1 - \epsilon$.

If $A \leq 3/2t$ then with probability $1 - \epsilon$ it holds that $\|x - y\|_1 \leq 3t$. Using M_2 and according to Lemma 49, Bob can recover x with probability $1 - \epsilon$. Then, Bob computes $\|x - y\|_1$ and outputs x if $\|x - y\|_1 \leq t$ and $\perp$ if $\|x - y\|_1 > t$.

The communication cost of the protocol is $O(|M_1| + |M_2|) = O(\log(nN)\log(\frac{1}{\epsilon}) + t\log(n))$. The computation time of Alice and Bob is $O(N \operatorname{polylog}(nN)\log(\frac{1}{\epsilon}) + N^2\log(n)^2)$. □

Theorem 25. *The protocol depicted in Algorithm 24 is a randomized protocol for EMD k-Budget Error-Correcting with approximation ratio $O(d)$ on the d-dimensional grid $[\Delta]^d$ with communication cost $\tilde{O}(k \log \Delta \log(n\Delta^d))$ bits and success probability $2/3$.*

Proof. Set $\alpha = 6$ and $\beta = 2$. Let the approximation ratio be $C = 10d$ [1]. The communication protocol is shown in Algorithm 24. We show now its correctness and analyze its communication complexity.

[1]We are not trying to optimize constants here.

Algorithm 24 Communication protocol for d-dimensional EMD k-Budget Error-Correcting

```
Alice and Bob use $\alpha = 6$ and $\epsilon = 1/(12\log(2\Delta))$. Denote by $\mathrm{enc}(A, t, \epsilon)$ (encoding) the message of Alice as in Lemma 47, and denote by $\mathrm{dec}(M, B, t, \epsilon)$ (decoding) the output of Bob as in Lemma 47.
Alice:
  1. Pick vector $\delta \in [\Delta]^d$ uniformly at random and let $x' = x + \delta$ (note that $x' \in [2\Delta]^d$)
  2. Construct $d$-dimensional Pyramid-Arrays $\mathrm{PA}(x')$
  3. Encoding of $\mathrm{PA}(x')$. For all $0 \le i \le L+1 : M_i = \mathrm{enc}(\mathrm{PA}_i(x'), 2\alpha k, \epsilon)$
  4. Send Message $M = (\delta, M_0, \ldots, M_{L+1})$ to Bob
Bob:
  1. Receive message $M = (\delta, M_0, \ldots, M_{L+1})$ from Alice
  2. Let $y' = y + \delta$ (note that $y' \in [2\Delta]^d$)
  3. Construct $d$-dimensional Pyramid-Arrays $\mathrm{PA}(y')$
  4. Decoding of $\mathrm{PA}(x')$. For all $0 \le i \le L+1 : PA'_i = \mathrm{dec}(M_i, \mathrm{PA}_i(y'), \alpha k, \epsilon)$
  5. Let $l^*$ be the largest value such that $\|\mathrm{PA}'_{l^*} - \mathrm{PA}_{l^*}(y')\|_1 \le 2\alpha k$ and $\mathrm{PA}'_{l^*+1} = \perp$
  6. Relocate at most $2\alpha k$ points of $y'$ to $y'^*$ such that $\mathrm{PA}'_{l^*} = \mathrm{PA}_{l^*}(y'^*)$
  7. Let $y^* = y'^* - \delta$
  8. output $y^*$
```

Correctness. Let $y_{\mathtt{OPT}} \in \mathcal{N}_k(y)$ be such that $\mathrm{EMD}(x, y_{\mathtt{OPT}})$ is minimized. Our goal is to show that $\mathrm{EMD}(x, y^*) \le C \cdot \mathrm{EMD}(x, y_{\mathtt{OPT}})$ with probability 2/3.

Let $\{(u_1, v_1), \ldots, (u_k, v_k)\}$ be a set of pairs such that in the optimal solution Bob moves one of his points at location v_i to u_i (that is, to match one of Alice's point at location u_i) for each $i \in [k]$. Let $(u_{k+1}, v_{k+1}), \ldots, (u_n, v_n)$ be a minimum weight perfect matching between the remaining $(n-k)$ of Bob's points with the remaining $(n-k)$ of Alice's points. If there are more than one such minimum perfect matchings, the choice can be made arbitrarily. W.l.o.g., we assume that $\|u_1 - v_1\|_2 \ge \|u_2 - v_2\|_2 \ge \ldots \ge \|u_n - v_n\|_2$.

Consider level ℓ^* computed by Bob. ℓ^* is the largest level $\ell \in [L] \cup \{0\}$ such that $\|\mathrm{PA}_\ell(x') - \mathrm{PA}_\ell(y')\|_1 \le 2\alpha k$ (assuming that no error occurred). Let $A = \sqrt{d} \cdot 2^{L-\ell^*}$ be the diagonal distance of a grid cell in level ℓ^*. Then, for all those pairs (u_i, v_i) $(i \in [n])$ with $\|u_i - v_i\|_2 \ge A$, we have $r_{\ell^*}(u_i) \ne r_{\ell^*}(v_i)$ at level ℓ^* in the corresponding tree. Thus each such pair will contribute 2 to $\|\mathrm{PA}_{\ell^*}(x') - \mathrm{PA}_{\ell^*}(y')\|_1$. However, there may also be pairs (u_i, v_i) with $\|u_i - v_i\|_2 < A$, $r_{\ell^*}(u_i) \ne r_{\ell^*}(v_i)$, and they also contribute 2 to $\|\mathrm{PA}_{\ell^*}(x') - \mathrm{PA}_{\ell^*}(y')\|_1$. In this case we say such a pair (u_i, v_i) is *misclassified*. We will bound now $\mathrm{EMD}(x, y_{\mathtt{OPT}})$ and $\mathrm{EMD}(x, y^*)$ separately.

- **Lower bound on** $\mathrm{EMD}(x, y_{\texttt{OPT}})$: In $y_{\texttt{OPT}}$, only the first k pairs are corrected. Hence:

$$\mathrm{EMD}(x, y_{\texttt{OPT}}) \geq \sum_{i=k+1}^{n} \|u_i - v_i\|_2 \geq \sum_{i=\beta k}^{n} \|u_i - v_i\|_2 . \tag{7.1}$$

- **Upper bound on** $\mathrm{EMD}(x, y^*)$: In our protocol, the first αk pairs are recovered so that the distance between each such pair is at most A after the relocation. Let n_1 be the largest number such that $\|u_{n_1} - v_{n_1}\|_2 \geq A$. The first n_1 pairs always get recovered since the original distance between each such pair is at least A. Therefore,

$$\begin{aligned}\mathrm{EMD}(x, y^*) &\leq \alpha k \cdot A + \sum_{i=n_1+1}^{n} \|u_i - v_i\|_2 \\ &\leq \alpha k \cdot A + \max\{\beta k - n_1, 0\} \cdot A + \sum_{i=\beta k}^{n} \|u_i - v_i\|_2 . \end{aligned} \tag{7.2}$$

Using Inequality 7.1 and Inequality 7.2, we obtain:

$$\begin{aligned}\frac{\mathrm{EMD}(x, y^*)}{\mathrm{EMD}(x, y_{\texttt{OPT}})} &\leq \frac{\alpha k \cdot A + \max\{\beta k - n_1, 0\} \cdot A + \sum_{i=\beta k}^{n} \|u_i - v_i\|_2}{\sum_{i=\beta k}^{n} \|u_i - v_i\|_2} \\ &\leq 1 + \frac{(\alpha + \beta) A k}{\sum_{i=\beta k}^{n} \|u_i - v_i\|_2}. \end{aligned} \tag{7.3}$$

We will show later that with probability at least $3/4$, it holds:

$$A \leq \frac{4d}{(\alpha - \beta)k} \sum_{i=\beta k}^{n} \|u_i - v_i\|_2 . \tag{7.4}$$

Using Inequality 7.4 and Inequality 7.3, we obtain

$$\begin{aligned}\frac{\mathrm{EMD}(x, y^*)}{\mathrm{EMD}(x, y_{\texttt{OPT}})} &\leq 1 + \frac{4d(\alpha + \beta)}{\alpha - \beta} \\ &= 1 + 8d < C. \end{aligned}$$

Therefore with probability $(3/4 - \varepsilon \cdot \log(2\Delta)) \geq 2/3$ our algorithm achieves a C-approximation. The first term $3/4$ is the probability that Equation 7.4 holds, and the second error term is introduced by applying Lemma 47 (choose $\epsilon = 1/(12 \log(2\Delta))$) to each of the $\log(2\Delta)$ levels of the pyramid arrays.

It remains to prove that Inequality 7.4 holds with probability at least $3/4$. To this end, we require the following observation (the proof of Proposition 1 is deferred to after the current proof).

Proposition 1. *Let l be any line of length x in d dimensions. Then the probability that a random grid G with side length K intersects l is at most $\frac{\sqrt{d}x}{K}$.*

Set $\eta = \frac{\alpha-\beta}{4\sqrt{d}}$. We focus on a level ℓ such that $2^{L-\ell} = \sum_{i=\beta k}^{n} \|u_i - v_i\|_2 / (\eta k)$ [1], and we will show that with probability $3/4$, $\|\mathrm{PA}_\ell(x') - \mathrm{PA}_\ell(y')\|_1 \le 2\alpha k$. If this is the case, then according to the definition of ℓ^*, it holds that $A = \sqrt{d} \cdot 2^{L-\ell^*} \le \sqrt{d} \cdot 2^{L-\ell}$.

For $i = \beta k, \dots, n$, let T_i be the indicator variable of the event that (u_i, v_i) is misclassified at level ℓ. Then by Proposition 1, we have that $\Pr[T_i = 1] \le \frac{\sqrt{d}\|u_i - v_i\|_2}{2^{L-\ell}}$. Let $T = \sum_{\beta k}^{n} T_i$. We have

$$\mathbb{E}[T] \le \sum_{i=\beta k}^{n} \frac{\sqrt{d}\,\|u_i - v_i\|_2}{2^{L-\ell}} = \frac{\sum_{i=\beta k}^{n} \sqrt{d}\,\|u_i - v_i\|_2}{\sum_{i=\beta k}^{n} \|u_i - v_i\|_2 / (\eta k)} = \sqrt{d}\eta k.$$

By the Markov inequality, we have $T \le 4\sqrt{d}\eta k = (\alpha - \beta)k$ with probability $3/4$. Therefore with probability $3/4$, it holds that

$$\left\|\mathrm{PA}_\ell(x') - \mathrm{PA}_\ell(y')\right\|_1 \le 2(T + \beta k) \le 2\alpha k.$$

Consequently, with probability $3/4$,

$$A \le \sqrt{d} \cdot \sum_{i=\beta k}^{n} \|u_i - v_i\|_2 / (\eta k) = \frac{4d}{(\alpha - \beta)k} \sum_{i=\beta k}^{n} \|u_i - v_i\|_2 .$$

Communication complexity. Note that $M = (\delta, M_0, \dots, M_{L+1})$ with $L = O(\log \Delta)$. For all $0 \le i \le L+1$ we have $|M_i| \in O(\log(n\Delta^d)\log(\log \Delta) + k \log n)$, and δ can be encoded with $O(d \log \Delta)$ bits. Hence, the total communication is bounded by

$$O\left(\log(n\Delta^d)\log(\Delta)\log(\log \Delta) + k \log n \log \Delta\right).$$

$\square$

Proposition 2. *Let l be any line of length x in d dimensions. Then the probability that a random grid G with side length K intersects l is at most $\frac{\sqrt{d}x}{K}$.*

Proof. Let $(x_1, \dots, x_d)$ be the direction vector of l. Then clearly $\sum_i x_i^2 = x^2$. We bound $\Pr[l \text{ intersects } G]$ as follows:

$$\Pr[l \text{ intersects } G] \le \sum_i \Pr[x_i e_i \text{ intersects } G],$$

where e_i denotes the d-dimensional standard basis vector pointing into dimension i. Furthermore,

$$\sum_i \Pr[x_i e_i \text{ intersects } G] = \sum_i \frac{x_i}{K} = \frac{1}{K}\sum_i x_i \le \frac{1}{K}\sqrt{d}\sum_i x_i^2 = \frac{\sqrt{d}x}{K},$$

where we used the fact that for any d-dimenional vector v with non-negative coordinates the inequality $\sum_i v_i \le \sqrt{d}\sum_i v_i^2$ holds. This fact is an immediate consequence of Hölder's inequality. $\square$

[1]For convenience, we assume that $\sum_{i=\beta k}^{n} \|u_i - v_i\|_2 / (\eta k)$ is a power of 2. Such an assumption will not change the approximation ratio by more than a factor of 2.

7.2 Lower Bound

We again illustrate the idea in the one dimensional case. We first describe a family of hard instances for EMD k-Budget Error-Correcting on the one dimensional grid $[\Delta]$. Alice and Bob hold sets of n points x and respectively y on grid $[\Delta]$. The construction is performed in two steps. In the first step, we choose p *point center* locations $1, \Delta/p+1, 2\Delta/p+1, \ldots, (p-1)\Delta/p+1$, and in both x and y we assign n/p points to each point center. In the second step, we move points from these point centers in x and y to the right. At this step we make the point sets x and y different. We pick L $(= \Theta(\log \Delta))$ subsets $X_1, \ldots, X_L \subseteq [p]$ such that $|X_i| = k$ for all $i \in [L]$. In x, for all $i \in [L]$, for all $j \in X_i$, we move one point in the j-th point center by a distance of 2^{Bi} where B is a technical parameter. In y we perform similar operations: we first pick a random $I \in [L]$, and then for all $i = \{I+1, \ldots, L\}$, for all $j \in X_i$, we move one point from the j-th point center by a distance of 2^{Bi}. Note that x and y differ by those points that are moved in x indicated by $X_1, \ldots, X_I$. These points remain in point centers in y. The k most significant differences in point set x and y are the k points that Alice moved by distance 2^{BI}, that is, those points indicated by X_I. Intuitively, if Bob wants to correct most of these points, Bob has to learn X_I approximately.

The technical implementation of this idea is a reduction from the well-known two-party one-way communication problem called Augmented Indexing. Augmented Indexing has been used to prove lower bounds in streaming and sparse-recovery literature [CW09, KNW10a, DBIPW10]. In Augmented Indexing, Alice has $(X_1, \ldots, X_L)$ and Bob has $(X_{I+1}, \ldots, X_L)$ for some index $I \in [L]$. Alice sends a single message to Bob and upon reception Bob outputs X_I. In our application, each X_i $(i \in [L])$ is a subset of $[p]$ of cardinality k. The main difficulty lies in the fact that we aim to solve Augmented Indexing given a protocol for EMD k-Budget Error-Correcting that only computes a constant factor approximation. The key of our argument is that on our hard instances a constant factor approximation to EMD k-Budget Error-Correcting must identify *many* of the k points indicated by X_i. We use a family of k-subsets with bounded intersection which is similar to a binary constant weight code such that those identified points are enough to recover the correct k-subset, that is X_i. We comment that similar ideas have also been used in [DBIPW10] for proving lower bounds for sparse-recovery problems.

In this section, we show that any randomized communication protocol that computes a C-approximation for d-dimensional EMD k-Budget Error-Correcting has communication complexity $\Omega(k\frac{\log \Delta}{\log C}(d \log \Delta - \log k)$. The proof is a reduction from the two-party one-way communication problem Augmented Indexing.

Definition 21 (Augmented Indexing)**.** *Let $X = (X_1, \ldots, X_n)$ where $X \in \mathcal{U}^n$ for some universe $\mathcal{U}$. Let $I \in [n]$. Alice is given X, Bob is given I and $(X_{I+1}, \ldots, X_n)$. Alice sends message M_{AI} to Bob and upon reception Bob outputs X_I.*

In [JW11] it is shown that the uniform distribution on X is a hard distribution for a version of Augmented Indexing where Bob also holds some $Y \in \mathcal{U}$ and the goal is to output 1 if $X_I = Y$ and 0 otherwise. They show that $\Omega(n \log |\mathcal{U}|)$ communication is necessary for protocols with error at most $\frac{1}{4|\mathcal{U}|}$. In our version of Augmented Indexing , Bob has to learn X_I. This allows us to modify (actually simplify) the proof in [JW11] to obtain the same communication bound for constant error.

Lemma 50. *If X and I are chosen uniformly at random and the failure probability of the protocol is at most $1/3$, then $\mathbb{E}_X |M_{\mathrm{AI}}| = \Omega(n \log |\mathcal{U}|)$.*

Proof. The proof follows [JW11], and uses the standard tools from information complexity. We refer the reader to [BYJKS04] for an introduction of information complexity. Let $X = (X_1, \ldots, X_n)$ where X_i is chosen uniformly and independently of $(X_j)_{j \neq i}$ from $\mathcal{U}$. Since $\mathbb{E}_X |M_{\mathrm{AI}}| \geq \mathrm{H}(M_{\mathrm{AI}}) \geq \mathrm{I}(X : M_{\mathrm{AI}})$, it is enough to bound $\mathrm{I}(X : M_{\mathrm{AI}})$. Then, by the chain rule for mutual information, and the definition of mutual information we obtain

$$\begin{aligned} \mathrm{I}(X : M_{\mathrm{AI}}) &= \sum_{i=1}^{n} \mathrm{I}(X_i : M_{\mathrm{AI}} \,|\, X_{i+1}, \ldots, X_n) \\ &= \sum_{i=1}^{n} \mathrm{H}(X_i \,|\, X_{i+1}, \ldots, X_n) - \sum_{i=1}^{n} \mathrm{H}(X_i \,|\, M_{\mathrm{AI}}, X_{i+1}, \ldots, X_n). \end{aligned}$$

By independence, we can simplify for all $i \in \{1, \ldots, n\}$ as follows

$$\mathrm{H}(X_i \,|\, X_{i+1}, \ldots, X_n) = \mathrm{H}(X_i) = \log(|\mathcal{U}|).$$

It remains to upper bound $\mathrm{H}(X_i \,|\, M_{\mathrm{AI}}, X_{i+1}, \ldots, X_n)$ for all $i \in \{1, \ldots, n\}$. Note that $\{M_{\mathrm{AI}}, X_{i+1}, \ldots, X_n\}$ is exactly Bob's input for Augmented Indexing. Bob outputs X_i with error ϵ, hence $M_{\mathrm{AI}}, X_{i+1}, \ldots, X_n$ is a predictor for X_i with error probability ϵ. We apply Fano's Inequality and obtain

$$\mathrm{H}(X_i \,|\, M_{\mathrm{AI}}, X_{i+1}, \ldots, X_n) \leq \mathrm{H}(\epsilon) + \epsilon \cdot \log(|\mathcal{U}| - 1),$$

where $\mathrm{H}(\epsilon)$ denotes the binary entropy of ϵ. Combining and setting $\epsilon = 1/3$, we obtain $\mathrm{I}(X : M_{\mathrm{AI}}) = \Omega(n \log |\mathcal{U}|)$. □

In the following, we will show how to solve Augmented Indexing with a protocol for d-dimensional EMD k-Budget Error-Correcting. In our application, the universe $\mathcal{U}$ from which the elements of X are chosen is a large family of k-subsets of a set $[p]$ ($p > k$) with bounded intersection. For $\epsilon > 0$, we define $\mathcal{C}_{k,p}^{\epsilon k}$ to be a family of k-subsets of $[p]$ such that any two subsets have at most $k(1-\epsilon)$ elements in common. Then we will use $\mathcal{U} = \mathcal{C}_{k,p}^{k/100}$. Such a family is equivalent to a binary constant weight code of length p, weight k, and distance $2\epsilon k$. In Lemma 52 we show that there is a large set of k-subsets with bounded intersection.

Lemma 51 (Gilbert-Varshamov Bound [HP03]). *Let $A_q(M, d)$ be the maximum possible size of a q-ary code with length M and Hamming distance at least d. Then,*

$$A_q(M, d) \geq \frac{q^M}{\sum_{j=0}^{d-1} \binom{M}{j} (q-1)^j}.$$

Lemma 52. *Let k, p be integers such that $k < p/2$, and let $\epsilon < 1 - 1/(\lfloor p/k \rfloor)$. Then there is a family $\mathcal{C}_{k,p}^{\epsilon k}$ of k-subsets of $[p]$ such that for $c_1, c_2 \in \mathcal{C}_{k,p}^{\epsilon k}, c_1 \neq c_2 : |c_1 \cap c_2| \leq k(1-\epsilon)$ and*

$$|\mathcal{C}_{k,p}^{\epsilon k}| \geq (\lfloor p/k \rfloor)^{k(1-H_{\lfloor p/k \rfloor}(\epsilon))},$$

where H_q is the q-ary entropy function $H_q(x) = -x \log_q \frac{x}{q-1} - (1-x)\log_q(1-x)$.

Proof. We follow the proof of Lemma 3.1 of [DBIPW10]. Let T be a code of block length k, alphabet $\{1, \ldots, \lfloor p/k \rfloor\}$ and Hamming distance ϵk. From T we obtain a binary code T' with block length p and Hamming distance $2\epsilon k$ by replacing each character i with the $\lfloor p/k \rfloor$-long standard basis vector e_i. Note that T' has exactly k ones. The set $\mathcal{C}_{k,p}^{\epsilon k}$ is obtained by interpreting the code words of T' as the characteristic vectors of the subsets. Then if code words $t'_1, t'_2 \in T'$ have Hamming distance $2\epsilon k$ then the corresponding k-subsets c_1, c_2 obtained from t'_1, t'_2 are such that $|c_1 \cap c_2| = k(1-\epsilon)$. By the Gilbert-Varshamov bound (Lemma 51) we obtain

$$|\mathcal{C}_{k,p}^{\epsilon k}| = |T| \geq \frac{(\lfloor p/k \rfloor)^k}{\sum_{i=0}^{\epsilon k-1} \binom{k}{i}(\lfloor p/k \rfloor - 1)^i}.$$

Following [DBIPW10], for $\epsilon < 1 - 1/(\lfloor p/k \rfloor)$ we can use $\sum_{i=0}^{\epsilon k-1} \binom{k}{i}(\lfloor p/k \rfloor - 1)^i < (\lfloor p/k \rfloor)^{H_{\lfloor p/k \rfloor}(\epsilon)k}$, and the result follows. □

Suppose that the EMD k-Budget Error-Correcting protocol outputs a C-approximation. We take three integer parameters L, p, k such that $p > k$ and $L = \lceil \frac{\log(p^{1/d}/10)}{\log(200C)+2} \rceil$. Let $X = (X_1, \ldots, X_L)$ where $X_i \in \mathcal{U} = \mathcal{C}_{k,p}^{k/100}$. X_i is a k-subset of $[p]$ and we write $X_i = (X_i^1, \ldots, X_i^k)$.

Consider the Augmented Indexing problem where Alice has X, Bob has $I \in [L]$ and $(X_{I+1}, \ldots, X_L)$. Then by applying Lemma 50 and Lemma 52, the communication complexity of this problem is

$$\begin{aligned}
\Omega(L \cdot \log|\mathcal{U}|) &= \Omega\left(L \cdot \log(|\mathcal{C}_{k,p}^{k/100}|)\right) \\
&= \Omega\left(\frac{\log(p^{1/d})}{\log C} \cdot \log\left((\lfloor p/k \rfloor)^{k\left(1-H_{\lfloor p/k \rfloor}(1/100)\right)}\right)\right) \\
&= \Omega\left(\frac{k}{d}\frac{\log p}{\log C}\log\left(\frac{p}{k}\right)\right), \qquad (7.5)
\end{aligned}$$

where we used the fact that for any $q \geq 1, H_q(1/100) < 0.35$.

Reduction. Given such an Augmented Indexing instance, Alice and Bob construct a d-dimensional instance for EMD k-Budget Error-Correcting with grid $[\Delta]^d$ ($\Delta = p^{2/d}$) and $n = 10kpL$ points. The construction requires a parameter B which we set to be $\log(200C)+2$. Furthermore, we make use of the set of coordinates $Z_p = \{1, p^{1/d}+1, 2p^{1/d}+1, \ldots (p^{1/d}-1)p^{1/d}+1\}^d$ that we call *point centers* since in the reduction Alice and Bob place many points onto these coordinates. Note that $|Z_p| = p$.

The reduction consists of 3 steps.

Step 1. Alice and Bob use an arbitrary but fixed bijection $f : [p] \to Z_p$. They proceed as follows to set up the EMD k-Budget Error-Correcting instance:

1. Alice and Bob place $\frac{n}{p}$ points to each point center in Z_p.

2. For each X_i^j ($i \in [L], j \in [k]$), Alice moves one point from point center $f(X_i^j)$ by a distance of 2^{Bi}, resulting a new point at location $f(X_i^j) + 2^{Bi}e_1$, where e_1 is the d-dimensional standard basis unit vector pointing to dimension 1. Bob does the same for each X_i^j with $i > I$. Denote Alice's points set by x and Bob's points set by y. Since $n = 10kpL$, there will be $n/p = 10kL$ points on each point center. Thus Alice and Bob can ensure that there are always enough points to move.

Here, the effect of parameter B becomes clear: Alice and Bob displace points from the point centers by distances 2^{Bi}. B is hence responsible for increasing the distance of points that correspond to different values of i. Note that we set $B = \Theta(\log C)$, hence the distances increase as the approximation factor increases.

Step 2. Alice and Bob run the protocol for EMD k-Budget Error-Correcting. Let y^* denote the points of Bob after the relocation outputted by the protocol.

Step 3. Bob rounds the points y^* to the closest positions in $\{Z_p + 2^{Bi}e_1 \mid i \in [L]\}$. Then, he computes an estimate $\tilde{X}_I'$ of X_I as follows: if there is a point in y^* at position $x + 2^{BI}e_1$ for some $x \in Z_p$, then $f^{-1}(x) \in \tilde{X}_I'$. Next, Bob selects $\tilde{X}_I \in \mathcal{C}_{k,p}^{k/100}$ such that $|\tilde{X}_I \cap \tilde{X}_I'|$ is maximized.

Theorem 26. *Any randomized communication protocol that computes a C-approximation to EMD k-Budget Error-Correcting on d-dimensional grid $[\Delta]^d$ with probability $2/3$ requires a message of size $\Omega(k\frac{\log\Delta}{\log C}(d\log\Delta - \log k))$, assuming that $k < \Delta^{d/2}$.*

Proof. We use the prior reduction from Augmented Indexing to EMD k-Budget Error-Correcting. Recall that the setup for the EMD k-Budget Error-Correcting instance uses $p = \Delta^{d/2}$, $B = \log(200C) + 2$, $L = \lceil \log(p^{1/d}/10)/B \rceil$, and $n = 10kpL$.

Firstly, note that the distance between any two point centers is larger than or equal to $\sqrt{\Delta}$. The maximal distance that a point is displaced from its point center is $2^{BL} = 0.1p^{1/d} = 0.1\sqrt{\Delta}$. Under this condition, the EMD between Alice's and Bob's points is the sum of the distances of the points that only Alice moved, that is, $\mathrm{EMD}(x, y) = k\sum_{i=1}^{I} 2^{Bi}$. Furthermore, we have $\min_{\tilde{y}\in\mathcal{N}_k(y)} \mathrm{EMD}(x, \tilde{y}) = k\sum_{i=1}^{I-1} 2^{Bi}$, which can be obtained by correcting the k points that Alice moved by a distance of 2^{BI}. Since our protocol approximates the EMD within a factor C, we obtain $\mathrm{EMD}(x, y^*) \le C \cdot k\sum_{i=1}^{I-1} 2^{Bi}$. Let $\mathrm{err} = |X_I \setminus \tilde{X}_I|$ be the number of points that Bob failed to recover. Then each of these points contributes to the EMD by at least $(2^{BI} - 2^{B(I-1)})/2$, since these points got rounded to some index other than I. We obtain

$$\mathrm{err} \cdot \left(2^{BI} - 2^{B(I-1)}\right)/\,2 \le C \cdot k\sum_{i=1}^{I-1} 2^{Bi},$$

Therefore we conclude that err $< \frac{Ck}{2^{B-2}} = \frac{k}{200}$. Since the k-subsets of $\mathcal{C}_{k,p}^{k/100}$ differ by at least $\frac{k}{100}$ elements, we can recover $\tilde{X}_I^i = X_I$. The lower bound for EMD k-Budget Error-Correcting follows by plugging $p = \Delta^{d/2}$ into Equation (7.5). $\square$

References

[Abr03] D. Abraham. Algorithmics of two-sided matching problems. Master's thesis, University of Glasgow, 2003. 18

[AG11] Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. In *Proceedings of the 38th international conference on Automata, languages and programming - Volume Part II*, ICALP'11, pages 526–538, Berlin, Heidelberg, 2011. Springer-Verlag. 16

[AGM12] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st symposium on Principles of Database Systems*, PODS '12, pages 5–14, New York, NY, USA, 2012. ACM. 10

[AMS96] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, STOC '96, pages 20–29, New York, NY, USA, 1996. ACM. 3, 10

[AMS99] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999. 75

[ANR95] Yossi Azar, Joseph Naor, and Raphael Rom. The competitiveness of on-line assignments. *J. Algorithms*, 18(2):221–237, 1995. 18, 19, 54

[BCS74] J. Bruno, E. G. Coffman, Jr., and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Commun. ACM*, 17(7):382–387, July 1974. 18

[Ber57] C. Berge. Two Theorems in Graph Theory. *Proceedings of the National Academy of Sciences of the United States of America*, 43(9):842–844, 1957. 18

[BFL$^+$06] Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '06, pages 253–262, New York, NY, USA, 2006. ACM. 10, 11

[BH12] Paul Beame and Trinh Huynh. The value of multiple read/write streams for approximating frequency moments. *ACM Trans. Comput. Theory*, 3(2):6:1–6:22, January 2012. 11

[BJR07] Paul Beame, T. S. Jayram, and Atri Rudra. Lower bounds for randomized read/write stream algorithms. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, STOC '07, pages 689–698, New York, NY, USA, 2007. ACM. 11

REFERENCES

[BYJKS04] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of Computer and System Sciences*, 68(4):702–732, 2004. 75, 115

[BYKS02] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '02, pages 623–632, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics. 11

[CB07] Christophe Chefd'hotel and Guillaume Bousquet. Intensity-based image registration using earth mover's distance. In *SPIE*, 2007. 24

[CCKM10] Amit Chakrabarti, Graham Cormode, Ranganath Kondapally, and Andrew McGregor. Information cost tradeoffs for augmented index and streaming language recognition. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 387–396, Washington, DC, USA, 2010. IEEE Computer Society. 11, 14

[CHSW12] Andrzej Czygrinow, Michal Hanckowiak, Edyta Szymanska, and Wojciech Wawrzyniak. Distributed 2-approximation algorithm for the semi-matching problem. In *DISC*, pages 210–222, 2012. 18

[CPSV00] Graham Cormode, Mike Paterson, Süleyman Cenk Sahinalp, and Uzi Vishkin. Communication complexity of document exchange. In *SODA '00*, pages 197–206. SIAM, 2000. 23

[CW09] Kenneth L. Clarkson and David P. Woodruff. Numerical linear algebra in the streaming model. In *STOC '09*, pages 205–214. ACM, 2009. 114

[DBIPW10] Khanh Do Ba, Piotr Indyk, Eric Price, and David P. Woodruff. Lower bounds for sparse recovery. In *SODA '10*, pages 1190–1197. SIAM, 2010. 114, 116

[DF91] M. Dyer and Al. Frieze. Randomized greedy matching. *Random Structures & Algorithms*, 2(1):29–46, 1991. 31, 32

[DLOM02] Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Frequency estimation of internet packet streams with limited space. In *Proceedings of the 10th Annual European Symposium on Algorithms*, ESA '02, pages 348–360, London, UK, UK, 2002. Springer-Verlag. 11

[DSLN09] Atish Das Sarma, Richard J. Lipton, and Danupon Nanongkai. Best-order streaming model. In *Proceedings of the 6th Annual Conference on Theory and Applications of Models of Computation*, TAMC '09, pages 178–191, Berlin, Heidelberg, 2009. Springer-Verlag. 11

[ECS73] J. Eruno, E. G. Coffman, Jr., and R. Sethi. Scheduling independent tasks to reduce mean finishing-time (extended abstract). In *Proceedings of the fourth ACM symposium on Operating system principles*, SOSP '73, pages 102–103, New York, NY, USA, 1973. ACM. 18

[ELMS11] L. Epstein, A. Levin, J. Mestre, and D. Segev. Improved approximation guarantees for weighted matching in the semi-streaming model. *SIAM J. on Discrete Mathematics*, 25(3):1251–1265, 2011. 16

[FKM$^+$04] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *ICALP*, volume 3142 of *Lecture Notes in Computer Science*, pages 531–543. Springer, 2004. 16, 17, 30

[FKM$^+$05] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the streaming model: the value of space. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '05, pages 745–754, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics. 10

[FKSV02] Joan Feigenbaum, Sampath Kannan, Martin J Strauss, and Mahesh Viswanathan. An approximate l 1-difference algorithm for massive data streams. *SIAM Journal on Computing*, 32(1):131–151, 2002. 110

[FLN10] Jittat Fakcharoenphol, Bundit Laekhanukit, and Danupon Nanongkai. Faster algorithms for semi-matching problems (extended abstract). In *ICALP (1)*, pages 176–187, 2010. 18

[FM83] Philippe Flajolet and G. Nigel Martin. Probabilistic counting. In *FOCS*, pages 76–82, 1983. 4, 10

[FM13] N. François and F. Magniez. Streaming complexity of checking priority queues. In *Proceedings of 30th Symposium on Theoretical Aspects of Computer Science*, 2013. To appear. 11

[GD04] Kristen Grauman and Trevor Darrell. Fast contour matching using approximate earth mover's distance. In *CVPR*, pages 220–227, 2004. 24

[GHS06] Martin Grohe, André Hernich, and Nicole Schweikardt. Randomized computations on large data sets: tight lower bounds. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '06, pages 243–252, New York, NY, USA, 2006. ACM. 11

[GHS09] M. Grohe, A. Hernich, and N. Schweikardt. Lower bounds for processing data with few random accesses to external memory. *Journal of the ACM*, 56(3):1–16, 2009. 90

[GI10] A. Gilbert and P. Indyk. Sparse recovery using sparse matrices. *Proceedings of the IEEE*, 98(6):937–947, 2010. 24

[GKK12] A. Goel, M. Kapralov, and S. Khanna. On the communication and streaming complexity of maximum bipartite matching. In *SODA*, 2012. To appear. 10, 11, 17, 58

[GKS07] Martin Grohe, Christoph Koch, and Nicole Schweikardt. Tight lower bounds for query processing on streaming and external memory data. *Theor. Comput. Sci.*, 380:199–217, July 2007. 21

[GKS11] Frantisek Galcík, Ján Katrenic, and Gabriel Semanisin. On computing an optimal semi-matching. In *WG*, pages 250–261, 2011. 18, 19

[GM09] Sudipto Guha and Andrew McGregor. Stream order and order statistics: Quantile estimation in random-order streams. *SIAM J. Comput.*, 38(5):2044–2059, January 2009. 11

[GMV06] Sudipto Guha, Andrew McGregor, and Suresh Venkatasubramanian. Streaming and sublinear approximation of entropy and information distances. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, SODA '06, pages 733–742, New York, NY, USA, 2006. ACM. 11

[GO13] V. Guruswami and K. Onak. Superlinear lower bounds for multipass graph processing. *28th IEEE Conference on Computational Complexity*, 2013. 10, 11

[GR11] John Gantz and David Reinsel. Extracting value from chaos. 2011. 1

[GS05] Martin Grohe and Nicole Schweikardt. Lower bounds for sorting with few random accesses to external memory. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '05, pages 238–249, New York, NY, USA, 2005. ACM. 11, 12

[HHLS10] Bjarni V. Halldórsson, Magnús M. Halldórsson, Elena Losievskaja, and Mario Szegedy. Streaming algorithms for independent sets. In *Proceedings of the 37th international colloquium conference on Automata, languages and programming*, ICALP'10, pages 641–652, Berlin, Heidelberg, 2010. Springer-Verlag. 10

[HLLT03] N. J. A. Harvey, R. E. Ladner, L. Lovász, and T. Tamir. Semi-matchings for bipartite graphs and load balancing. In *WADS*, pages 294–308, 2003. 18, 19

[Hor73] W. A. Horn. Minimizing average flow time with parallel machines. *Operations Research*, pages 846–847, 1973. 18

[HP03] W Cary Huffman and Vera Pless. *Fundamentals of error-correcting codes*. Cambridge university press, 2003. 110, 115

[HRR99] Monika R. Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. External memory algorithms. chapter Computing on data streams, pages 107–118. American Mathematical Society, Boston, MA, USA, 1999. 10

[HRT02] Anthony S. Holmes, C. J. Rose, and Christopher J. Taylor. Transforming pixel signatures into an improved metric space. *Image Vision Comput.*, 20(9-10):701–707, 2002. 24

[IMS05] U. Irmak, S. Mihaylov, and T. Suel. Improved single-round protocols for remote file synchronization. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 1665–1676. IEEE, 2005. 23

[IN03] P. Indyk and N.Thaper. Fast color image retrieval via embeddings. *Workshop on Statistical and Computational Theories of Vision (at ICCV)*, 2003. 107

[Ind07] Piotr Indyk. A near linear time constant factor approximation for euclidean bichromatic matching (cost). In *SODA '07*, pages 39–42. SIAM, 2007. 108

[IP11] Piotr Indyk and Eric Price. K-median clustering, model-based compressive sensing, and sparse recovery for earth mover distance. In *STOC*, pages 627–636, 2011. 25

[JG05] Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In *In COCOON*, pages 710–716, 2005. 10

[JN10] Rahul Jain and Ashwin Nayak. The space complexity of recognizing well-parenthesized expressions. Technical Report TR10-071, Electronic Colloquium on Computational Complexity, `http://eccc.hpi-web.de/`, April 19 2010. Revised July 20, 2011. 11, 14

[Jow12] Hossein Jowhari. Efficient communication protocols for deciding edit distance. In *ESA*, 2012. 23

[JW11] T. S. Jayram and David Woodruff. Optimal bounds for johnson-lindenstrauss transforms and streaming problems with sub-constant error. In *SODA '11*, pages 1–10. SIAM, 2011. 114, 115

[Kap13] Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of 24th ACM-SIAM Symposium on Discrete Algorithms*, 2013. To appear. 11, 17

[KM12] Christian Konrad and Frédéric Magniez. Validating xml documents in the streaming model with external memory. In *Proceedings of 15th International Conference on Database Theory*, 2012. 7, 11, 15

[KMM12] Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Proceedings of 15th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, 2012. 6, 10, 11, 15, 17

[KMSS12] Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *ICALP (2)*, volume 7392 of *Lecture Notes in Computer Science*, pages 598–609. Springer, 2012. 10

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997. 5, 13, 75

[KNW10a] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. On the exact space complexity of sketching and streaming small norms. In *SODA '10*, pages 1161–1178. SIAM, 2010. 114

[KNW10b] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '10, pages 41–52, New York, NY, USA, 2010. ACM. 4

[KR13] Christian Konrad and Adi Rosén. Approximation Semi-Matchings in Streaming and in Two-Party Communication. In *Proceedings of the 40th international conference on Automata, languages and programming*, ICALP'13, 2013. 6, 15, 19

[KR99] Jon Kleinberg, Yuval Rabani, and Éva Tardos. Fairness in routing and load balancing. In *J. Comput. Syst. Sci*, pages 568–578, 1999. 58

[KVV90] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, STOC '90, pages 352–358, New York, NY, USA, 1990. ACM. 17

[KYZ13] Christian Konrad, Wei Yu, and Qin Zhang. Budget Error-Correcting under Earth-Mover Distance. Research report, 2013. submitted. 7, 15, 25

[LL04] Y. Lin and W. Li. Parallel machine scheduling of machine-dependent jobs with unit-length. *European Journal of Operational Research*, 2004. 18

[McG05] Andrew McGregor. Finding graph matchings in data streams. In *Proceedings of the 8th international workshop on Approximation, Randomization and Combinatorial Optimization Problems, and Proceedings of the 9th international conference on Randamization and Computation: algorithms and techniques*, APPROX'05/RANDOM'05, pages 170–181, Berlin, Heidelberg, 2005. Springer-Verlag. 10, 16

[MMN10] F. Magniez, C. Mathieu, and A. Nayak. Recognizing well-parenthesized expressions in the streaming model. In *Proceedings of 42nd ACM Symposium on Theory of Computing*, pages 261–270, 2010. 11, 21, 73, 75, 103

[MP78] J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, SFCS '78, pages 253–258, Washington, DC, USA, 1978. IEEE Computer Society. 10

[MU05] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. 41

[Mut05] S. Muthukrishnan. Data streams: algorithms and applications. *Found. Trends Theor. Comput. Sci.*, 1(2):117–236, August 2005. 10

[Nev02] Frank Neven. Automata theory for xml researchers. *Sigmod Record*, 31:2002, 2002. 73

[PBRT99] Jan Puzicha, Joachim M. Buhmann, Yossi Rubner, and Carlo Tomasi. Empirical evaluation of dissimilarity measures for color and texture. In *ICCV*, pages 1165–1173, 1999. 24

[PV00] Yannis Papakonstantinou and Victor Vianu. Dtd inference for views of xml data. In *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '00, pages 35–46, New York, NY, USA, 2000. ACM. 76

[RTG00] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The earth mover's distance as a metric for image retrieval. *Int. J. Comput. Vision*, 40(2):99–121, November 2000. 24

[SS07] L. Segoufin and C. Sirangelo. Constant-memory validation of streaming XML documents against DTDs. In *ICDT*, pages 299–313, 2007. 20

[SV02] L. Segoufin and V. Vianu. Validating streaming XML documents. In *ACM PODS*, pages 53–64, 2002. 21

[WB86] Lloyd R Welch and Elwyn R Berlekamp. Error correction for algebraic block codes, December 30 1986. US Patent 4,633,470. 110

[Yao79] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing(preliminary report). In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, STOC '79, pages 209–213, New York, NY, USA, 1979. ACM. 13

[Zel12] M. Zelke. Weighted matching in the semi-streaming model. *Algorithmica*, 62(1-2):1–20, 2012. 16