Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 51–62
www.stacs-conf.org

LOWER BOUNDS FOR MULTI-PASS PROCESSING OF MULTIPLE DATA STREAMS

NICOLE SCHWEIKARDT

Institut für Informatik, Goethe-Universität Frankfurt am Main,
Robert-Mayer-Str. 11–15, D-60325 Frankfurt am Main, Germany
E-mail address: schweika@informatik.uni-frankfurt.de
URL: http://www.informatik.uni-frankfurt.de/~schweika

ABSTRACT. This paper gives a brief overview of computation models for data stream processing, and it introduces a new model for multi-pass processing of multiple streams, the so-called *mp2s-automata*. Two algorithms for solving the set disjointness problem with these automata are presented. The main technical contribution of this paper is the proof of a lower bound on the size of memory and the number of heads that are required for solving the set disjointness problem with mp2s-automata.

1. Introduction

In the basic data stream model, the input consists of a stream of data items which can be read only sequentially, one after the other. For processing these data items, a memory buffer of limited size is available. When designing data stream algorithms, one aims at algorithms whose memory size is far smaller than the size of the input.

Typical application areas for which data stream processing is relevant are, e.g., IP network traffic analysis, mining text message streams, or processing meteorological data generated by sensor networks. Data stream algorithms are also used to support query optimization in relational database systems. In fact, virtually all query optimization methods in relational database systems rely on information about the number of distinct values of an attribute or the self-join size of a relation — and these pieces of information have to be maintained while the database is updated. Data stream algorithms for accomplishing this task have been introduced in the seminal paper [2].

Most parts of the data stream literature deal with the task of performing **one pass over a single stream**. For a detailed overview on algorithmic techniques for this scenario we refer to [23]. *Lower bounds* on the size of memory needed for solving a problem by a one-pass algorithm are usually obtained by applying methods from *communication complexity* (see, e.g., [2, 20]). In fact, for many concrete problems it is known that the memory needed for solving the problem by a deterministic one-pass algorithm is at least linear in the size n of the input. For some of these problems, however, *randomized* one-pass algorithms can still compute good *approximate* answers while using memory

1998 ACM Subject Classification: F.1.1 (Computation by Abstract Devices: Models of Computation); F.2.2 (Analysis of Algorithms and Problem Complexity: Nonnumerical Algorithms and Problems); F.2.3 (Analysis of Algorithms and Problem Complexity: Tradeoffs between Complexity Measures) .

Key words and phrases: data streams, lower bounds, machine models, automata, the set disjointness problem.

© Nicole Schweikardt
Creative Commons Attribution-NoDerivs License

STACS 2009
Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 51-62
http://drops.dagstuhl.de/opus/volltexte/2009/1857

of size sublinear in n. Typically, such algorithms are based on *sampling*, i.e., only a "representative" portion of the data is taken into account, and *random projections*, i.e., only a rough "sketch" of the data is stored in memory. See [23, 10] for a comprehensive survey of according algorithmic techniques and for pointers to the literature.

Also the generalization where **multiple passes over a single stream** are performed, has received considerable attention in the literature. Techniques for proving lower bounds in this scenario can be found, e.g., in [20, 18, 9, 12, 22].

A few articles also deal with the task of **processing several streams in parallel**. For example, the authors of [28] consider algorithms which perform one pass over several streams. They introduce a new model of multi-party communication complexity that is suitable for proving lower bounds on the amount of memory necessary for one-pass algorithms on multiple streams. In [28], these results are used for determining the exact space complexity of processing particular XML twig queries.
In recent years, the database community has also addressed the issue of designing general-purpose *data stream management systems* and query languages that are suitable for new application areas where multiple data streams have to be processed in parallel. To get an overview of this research area, [3] is a good starting point. Foundations for a theory of *stream queries* have been laid in [19]. Stream-based approaches have also been examined in detail in connection with *XML query processing and validation*, see, e.g. the papers [27, 26, 13, 8, 4, 5, 16].

The *finite cursor machines* (FCMs, for short) of [14] are a computation model for performing **multiple passes over multiple streams**. FCMs were introduced as an abstract model of database query processing. Formally, they are defined in the framework of *abstract state machines*. Informally, they can be described as follows: The input for an FCM is a relational database, each relation of which is represented by a *table*, i.e., an ordered list of rows, where each row corresponds to a tuple in the relation. Data elements are viewed as "indivisible" objects that can be manipulated by a number of "built-in" operations. This feature is very convenient to model standard operations on data types like integers, floating point numbers, or strings, which may all be part of the universe of data elements. FCMs can operate in a finite number of *modes* using an *internal memory* in which they can store bitstrings. They access each relation through a finite number of *cursors*, each of which can read one row of a table at any time. The model incorporates certain *streaming* or *sequential processing* aspects by imposing a restriction on the movement of the cursors: They can move on the tables only sequentially in one direction. Thus, once the last cursor has left a row of a table, this row can never be accessed again during the computation. Note, however, that several cursors can be moved asynchronously over the same table at the same time, and thus, entries in different, possibly far apart, regions of the table can be read and processed simultaneously.

A common feature of the computation models mentioned so far in this paper is that the input streams are *read-only* streams that cannot be modified during a pass. Recently, also **stream-based models for external memory processing** have been proposed, among them the *StrSort model* [1, 24], the *W-Stream* model [11], and the model of *read/write streams* [17, 16, 15, 7, 6]. In these models, several passes may be performed over a single stream or over several streams in parallel, and during a pass, the content of the stream may be modified.

A detailed introduction to *algorithms on data streams*, respectively, to the related area of *sublinear algorithms* can be found in [23, 10]. A survey of *stream-based models for external memory processing* and of methods for proving *lower bounds* in these models is given in [25]. A database systems oriented overview of so-called *data stream systems* can be found in [3]. For a list of *open problems* in the area of data streams we refer to [21].

In the remainder of this article, a new computation model for multi-pass processing of multiple streams is introduced: the *mp2s-automata*. In this model, (read-only) streams can be processed by forward scans as well as backward scans, and several "heads" can be used to perform several passes over the streams in parallel. After fixing the basic notation in Section 2, the computation model of mp2s-automata is introduced in Section 3. In Section 4, we consider the *set disjointness problem* and prove upper bounds as well as lower bounds on the size of memory and the number of heads that are necessary for solving this problem with an mp2s-automaton. Section 5 concludes the paper by pointing out some directions for future research.

2. Basic notation

If f is a function from the set of non-negative integers to the set of reals, we shortly write $f(n)$ instead of $\lceil f(n) \rceil$ (where $\lceil x \rceil$ denotes the smallest integer $\geqslant x$). We write $\lg n$ to denote the logarithm of n with respect to base 2. For a set $\mathbb{D}$ we write $\mathbb{D}^*$ to denote the set of all finite strings over alphabet $\mathbb{D}$. We view $\mathbb{D}^*$ as the set of all finite *data streams* that can be built from elements in $\mathbb{D}$. For a stream $\vec{S} \in \mathbb{D}^*$ write $|\vec{S}|$ to denote the length of $\vec{S}$, and we write s_i to denote the element in $\mathbb{D}$ that occurs at the i-th position in $\vec{S}$, i.e., $\vec{S} = s_1 s_2 \cdots s_{|\vec{S}|}$.

3. A computation model for multi-pass processing of multiple streams

In this section, we fix a computation model for multi-pass processing of multiple streams. The model is quite powerful: Streams can be processed by forward scans as well as backward scans, and several "heads" can be used to perform several passes over the stream in parallel. For simplicity, we restrict attention to the case where just *two* streams are processed in parallel. Note, however, that it is straightforward to generalize the model to an arbitrary number of streams.

The computation model, called *mp2s-automata*[1], can be described as follows: Let $\mathbb{D}$ be a set, and let m, k_f, k_b be integers with $m \geqslant 1$ and $k_f, k_b \geqslant 0$. An

mp2s-automaton $\mathcal{A}$ *with parameters* $(\mathbb{D}, m, k_f, k_b)$

receives as input two streams $\vec{S} \in \mathbb{D}^*$ and $\vec{T} \in \mathbb{D}^*$. The automaton's memory consists of m different states (note that this corresponds to a memory buffer consisting of $\lg m$ bits). The automaton's state space is denoted by Q. We assume that Q contains a designated *start state* and that there is a designated subset F of Q of so-called *accepting states*.

On each of the input streams $\vec{S}$ and $\vec{T}$, the automaton has k_f heads that process the stream from left to right (so-called *forward heads*) and k_b heads that process the stream from right to left (so-called *backward heads*). The heads are allowed to move asynchronously. We use k to denote the total number of heads, i.e., $k = 2k_f + 2k_b$.

In the *initial configuration* of $\mathcal{A}$ on input $(\vec{S}, \vec{T})$, the automaton is in the *start state*, all *forward* heads on $\vec{S}$ and $\vec{T}$ are placed on the leftmost element in the stream, i.e., s_1 resp. t_1, and all *backward* heads are placed on the rightmost element in the stream, i.e., $s_{|\vec{S}|}$ resp. $t_{|\vec{T}|}$.

During each computation step, depending on (a) the current state (i.e., the current content of the automaton's memory) and (b) the elements of $\vec{S}$ and $\vec{T}$ at the current head positions, a deterministic transition function determines (1) the next state (i.e., the new content of the automaton's memory) and (2) which of the k heads should be advanced to the next position (where forward heads are

[1] "*mp2s*" stands for multi-pass processing of 2 streams

$\mathbb{D}$	:	set of *data items* of which input streams $\vec{S}$ and $\vec{T}$ are composed
m	:	size of the automaton's *state space* Q (this corresponds to $\lg m$ bits of memory)
k_f	:	number of *forward heads* available on each input stream
k_b	:	number of *backward heads* available on each input stream
k	:	$2k_f + 2k_b$ (total number of heads)

Figure 1: The meaning of the parameters $(\mathbb{D}, m, k_f, k_b)$ of an mp2s-automaton.

advanced one step to the right, and backward heads are advanced one step to the left). Formally, the transition function can be specified in a straightforward way by a function

$$\delta \ : \ Q \times (\mathbb{D} \cup \{\mathit{end}\})^k \ \longrightarrow \ Q \times \{\mathit{advance}, \mathit{stay}\}^k$$

where Q denotes the automaton's state space, and *end* is a special symbol (not belonging to $\mathbb{D}$) which indicates that a head has reached the end of the stream (for a forward head this means that the head has been advanced beyond the rightmost element of the stream, and for a backward head this means that the head has been advanced beyond the leftmost element of the stream).

The automaton's computation on input $(\vec{S}, \vec{T})$ ends as soon as each head has passed the entire stream. The input is *accepted* if the automaton's state then belongs to the set F of accepting states, and it is *rejected* otherwise.

The computation model of mp2s-automata is closely related to the *finite cursor machines* of [14]. In both models, several streams can be processed in parallel, and several heads (or, "cursors") may be used to perform several "asynchronous" passes over the same stream in parallel. In contrast to the mp2s-automata of the present paper, finite cursor machines were introduced as an abstract model for database query processing, and their formal definition in [14] is presented in the framework of *abstract state machines*.

Note that mp2s-automata can be viewed as a generalization of other models for one-pass or multi-pass processing of streams. For example, the scenario of [28], where a single pass over two streams is performed, is captured by an mp2s-automaton where 1 forward head and no backward heads are available on each stream. Also, the scenario where p consecutive passes of each input stream are available (cf., e.g., [20]), can be implemented by an mp2s-automaton: just use p forward heads and 0 backward heads, and let the i-th head wait at the first position of the stream until the $(i-1)$-th head has reached the end of the stream.

4. The set disjointness problem

Throughout Section 4 we consider a particular version of the *set disjointness problem* where, for each integer $n \geqslant 1$, $\mathbb{D}_n := \{ a_1, b_1, \ \ldots, \ a_n, b_n \}$ is a fixed set of $2n$ data items. We write Disj_n to denote the following decision problem: The input consists of two streams $\vec{S}$ and $\vec{T}$ over $\mathbb{D}_n$ with $|\vec{S}| = |\vec{T}| = n$. The goal is to decide whether the sets $\{s_1, \ldots, s_n\}$ and $\{t_1, \ldots, t_n\}$ are disjoint.

An mp2s-automaton *solves* the problem Disj_n if, for all valid inputs to Disj_n (i.e., all $\vec{S}, \vec{T} \in \mathbb{D}^*$ with $|\vec{S}| = |\vec{T}| = n$), it accepts the input if, and only if, the corresponding sets are disjoint.

4.1. Two upper bounds for the set disjointness problem

It is straightforward to see that the problem *Disj*$_n$ can be solved by an mp2s-automaton with 2^{2n} states and a single forward head on each of the two input streams: During a first phase, the head on $\vec{S}$ processes $\vec{S}$ and stores, in the automaton's current state, the subset of $\mathbb{D}_n$ that has been seen while processing $\vec{S}$. Afterwards, the head on $\vec{T}$ processes $\vec{T}$ and checks whether the element currently seen by this head belongs to the subset of $\mathbb{D}_n$ that is stored in the automaton's state. Clearly, 2^{2n} states suffice for this task, since $|\mathbb{D}_n| = 2n$. We thus obtain the following trivial upper bound:

Proposition 4.1. *Disj*$_n$ *can be solved by an mp2s-automaton with parameters* $(\mathbb{D}_n, 2^{2n}, 1, 0)$.

The following result shows that, at the expense of increasing the number of forward heads on each stream to $\sqrt{n}$, the memory consumption can be reduced exponentially:

Proposition 4.2. *Disj*$_n$ *can be solved by an mp2s-automaton with parameters* $(\mathbb{D}_n, n{+}2, \sqrt{n}, 0)$.[2]

Proof. The automaton proceeds in two phases.

The goal in *Phase 1* is to move, for each $i \in \{1, \ldots, \sqrt{n}\}$, the i-th head on $\vec{S}$ onto the $\big((i{-}1)\sqrt{n} + 1\big)$-th position in $\vec{S}$. This way, after having finished *Phase 1*, the heads partition $\vec{S}$ into $\sqrt{n}$ sub-streams, each of which has length $\sqrt{n}$. Note that $n + 1 - \sqrt{n}$ states suffice for accomplishing this: The automaton simply stores, in its state, the current position of the rightmost head(s) on $\vec{S}$. It starts by leaving head 1 at position 1 and moving the remaining heads on $\vec{S}$ to the right until position $\sqrt{n} + 1$ is reached. Then, it leaves head 2 at position $\sqrt{n} + 1$ and proceeds by moving the remaining heads to the right until position $2\sqrt{n} + 1$ is reached, etc.

During *Phase 2*, the automaton checks whether the two sets are disjoint. This is done in $\sqrt{n}$ sub-phases. During the j-th sub-phase, the j-th head on $\vec{T}$ processes $\vec{T}$ from left to right and compares each element in $\vec{T}$ with the elements on the current positions of the $\sqrt{n}$ heads on $\vec{S}$. When the j-th head on $\vec{T}$ has reached the end of the stream, each of the heads on $\vec{S}$ is moved one step to the right. This finishes the j-th sub-phase. Note that *Phase 2* can be accomplished by using just 2 states: By looking at the combination of heads on $\vec{T}$ that have already passed the entire stream, the automaton can tell which sub-phase it is currently performing. Thus, for *Phase 2* we just need one state for indicating that the automaton is in *Phase 2*, and an additional state for storing that the automaton has discovered already that the two sets are *not* disjoint. ∎

4.2. Two lower bounds for the set disjointness problem

We first show a lower bound for mp2s-automata where only forward heads are available:

Theorem 4.3. *For all integers* n, m, k_f, *such that, for* $k = 2k_f$ *and* $v = k_f^2 + 1$,

$$k^2 \cdot v \cdot \lg(n{+}1) \;+\; k \cdot v \cdot \lg m \;+\; v \cdot (1 + \lg v) \;\leqslant\; n\,,$$

the problem Disj$_n$ *cannot be solved by any mp2s-automaton with parameters* $(\mathbb{D}_n, m, k_f, 0)$.

Proof. Let n, m, and k_f be chosen such that they meet the theorem's assumption. For contradiction, let us assume that $\mathcal{A}$ is an mp2s-automaton with parameters $(\mathbb{D}_n, m, k_f, 0)$ that solves the problem *Disj*$_n$.

[2] To be precise, the proof shows that already $n + 2 - \sqrt{n}$ states suffice.

Recall that $\mathbb{D}_n = \{ a_1, b_1, \ldots, a_n, b_n \}$ is a fixed set of $2n$ data items. Throughout the proof we will restrict attention to input streams $\vec{S}$ and $\vec{T}$ which are enumerations of the elements in a set

$$A^I \quad := \quad \{a_i : i \in I\} \cup \{b_i : i \in \overline{I}\}$$

for arbitrary $I \subseteq \{1, .., n\}$ and its complement $\overline{I} := \{1, .., n\} \setminus I$.
Note that for all $I_1, I_2 \subseteq \{1, .., n\}$ we have

$$A^{I_1} \text{ and } A^{I_2} \text{ are disjoint} \iff I_2 = \overline{I_1}. \tag{4.1}$$

For each $I \subseteq \{1, .., n\}$ we let $\vec{S}^I$ be the stream of length n which is defined as follows: For each $i \in I$, it carries data item a_i at position i; and for each $i \notin I$, it carries data item b_i at position i. The stream $\vec{T}^I$ contains the same data items as $\vec{S}^I$, but in the opposite order: For each $i \in I$, it carries data item a_i at position $n - i + 1$; and for each $i \notin I$, it carries data item b_i at position $n - i + 1$.

For sets $I_1, I_2 \subseteq \{1, .., n\}$, we write $D(I_1, I_2)$ to denote the input instance $\vec{S}^{I_1}$ and $\vec{T}^{I_2}$ for the problem *Disj*$_n$. From (4.1) and our assumption that the mp2s-automaton $\mathcal{A}$ solves *Disj*$_n$, we obtain that

$$\mathcal{A} \text{ accepts } D(I_1, I_2) \iff I_2 = \overline{I_1}. \tag{4.2}$$

Throughout the remainder of this proof, our goal is to find two sets $I, I' \subseteq \{1, .., n\}$ such that

(1) $I \neq I'$, and
(2) the accepting run of $\mathcal{A}$ on $D(I, \overline{I})$ is "similar" to the accepting run of $\mathcal{A}$ on $D(I', \overline{I'})$, so that the two runs can be combined into an accepting run of $\mathcal{A}$ on $D(I, \overline{I'})$ (later on in the proof, we will see what "similar" precisely means).

Then, however, the fact that $\mathcal{A}$ accepts input $D(I, \overline{I'})$ contradicts (4.2) and thus finishes the proof of Theorem 4.3.

For accomplishing this goal, we let

$$v \quad := \quad k_f^2 + 1 \tag{4.3}$$

be 1 plus the number of pairs of heads on the two streams. We subdivide the set $\{1, .., n\}$ into v consecutive blocks $B_1, \ldots, B_v$ of equal size $\frac{n}{v}$. I.e., for each $j \in \{1, .., v\}$, block B_j consists of the indices in $\{ (j-1)\frac{n}{v} + 1, \ldots, j\frac{n}{v} \}$.

We say that a pair (h_S, h_T) of heads of $\mathcal{A}$ *checks block* B_j during the run on input $D(I_1, I_2)$ if, and only if, at some point in time during the run, there exist $i, i' \in B_j$ such that head h_S is on element a_i or b_i in $\vec{S}^{I_1}$ and head h_T is on element $a_{i'}$ or $b_{i'}$ in $\vec{T}^{I_2}$.

Note that each pair of heads can check at most one block, since only forward heads are available and the data items in $\vec{T}^{I_2}$ are arranged in the reverse order (with respect to the indices i of elements a_i and b_i) than in $\vec{S}^{I_1}$. Since there are v blocks, but only $v - 1$ pairs (h_S, h_T) of heads on the two streams, we know that for each $I_1, I_2 \subseteq \{1, .., n\}$ there exists a block B_j that is *not checked* during $\mathcal{A}$'s run on $D(I_1, I_2)$.

In the following, we determine a set $X \subseteq \{I : I \subseteq \{1, .., n\}\}$ with $|X| \geqslant 2$ such that for all $I, I' \in X$, item (2) of our goal is satisfied. We start by using a simple averaging argument to find a $j_0 \in \{1, .., v\}$ and a set $X_0 \subseteq \{I : I \subseteq \{1, .., n\}\}$ such that

- for each $I \in X_0$, block B_{j_0} is not checked during $\mathcal{A}$'s run on input $D(I, \overline{I})$, and
- $|X_0| \geqslant \frac{2^n}{v}$.

For the remainder of the proof we fix $\hat{B} := B_{j_0}$.
We next choose a sufficiently large set $X_1 \subseteq X_0$ in which everything outside block $\hat{B}$ is fixed:
A simple averaging argument shows that there is a $X_1 \subseteq X_0$ and a $\hat{I} \subseteq \{1, \ldots, n\} \setminus \hat{B}$ such that

- for each $I \in X_1$, $I \setminus \hat{B} = \hat{I}$, and
- $|X_1| \geqslant \frac{|X_0|}{2^{n-\frac{n}{v}}} \geqslant 2^{\frac{n}{v} - \lg v}$.

We next identify a set $X_2 \subseteq X_1$ such that for all $I, I' \in X_2$ the runs of $\mathcal{A}$ on $D(I, \overline{I})$ and $D(I', \overline{I'})$ are "similar" in a sense suitable for item (2) of our goal. To this end, for each head h of $\mathcal{A}$ we let config_h^I be the *configuration* (i.e., the current state and the absolute positions of all the heads) in the run of $\mathcal{A}$ on input $D(I, \overline{I})$ at the particular point in time where head h has just left block $\hat{B}$ (i.e., head h has just left the last element a_i or b_i with $i \in \hat{B}$ that it can access). We let config^I be the ordered tuple of the configurations config_h^I for all heads h of $\mathcal{A}$. Note that the number of possible configurations config_h^I is $\leqslant m \cdot (n{+}1)^k$, since $\mathcal{A}$ has m states and since each of the $k = 2k_f$ heads can be at one out of $n{+}1$ possible positions in its input stream. Consequently, the number of possible k-tuples config^I of configurations is $\leqslant \big(m \cdot (n{+}1)^k\big)^k$.
A simple averaging argument thus yields a tuple c of configurations and a set $X_2 \subseteq X_1$ such that

- for all $I \in X_2$, $\mathit{config}^I = c$, and
- $|X_2| \geqslant \frac{|X_1|}{(m \cdot (n+1)^k)^k} \geqslant 2^{\frac{n}{v} - \lg v \,-\, k \lg m \,-\, k^2 \lg(n+1)}$.

Using the theorem's assumption on the numbers n, m, and k_f, one obtains that $|X_2| \geqslant 2$. Therefore, we can find two sets $I, I' \in X_2$ with $I \neq I'$.

To finish the proof of Theorem 4.3, it remains to show that the runs of $\mathcal{A}$ on $D(I, \overline{I})$ and on $D(I', \overline{I'})$ can be combined into a run of $\mathcal{A}$ on $D(I, \overline{I'})$ such that $\mathcal{A}$ (falsely) accepts input $D(I, \overline{I'})$. To this end let us summarize what we know about I and I' in X_2:

(a) I and I' only differ in block $\hat{B}$.

(b) Block $\hat{B}$ is not checked during $\mathcal{A}$'s runs on $D(I, \overline{I})$ and on $D(I', \overline{I'})$. I.e., while any head on $\vec{S}^I$ (resp. $\vec{S}^{I'}$) is at an element a_i or b_i with $i \in \hat{B}$, no head on $\vec{T}^{\overline{I}}$ (resp. $\vec{T}^{\overline{I'}}$) is on an element $a_{i'}$ or $b_{i'}$ with $i' \in \hat{B}$.

(c) Considering $\mathcal{A}$'s runs on $D(I, \overline{I})$ and on $D(I', \overline{I'})$, each time a head leaves the last position in $\hat{B}$ that it can access, both runs are are in exactly the same configuration. I.e., they are in the same state, and all heads are at the same absolute positions in their input streams.

Due to item (a), $\mathcal{A}$'s run on input $D(I, \overline{I'})$ starts in the same way as the runs on $D(I, \overline{I})$ and $D(I', \overline{I'})$: As long as no head has reached an element in block $\hat{B}$, the automaton has not yet seen any difference between $D(I, \overline{I'})$ on the one hand and $D(I, \overline{I})$ and $D(I', \overline{I'})$ on the other hand.

At some point in time, however, some head h will enter block $\hat{B}$, i.e., it will enter the first element a_i or b_i with $i \in \hat{B}$ that it can access. The situation then is as follows:

- If h is a head on $\vec{S}^I$, then, due to item (b), no head on $\vec{T}^{\overline{I'}}$ is at an element in $\hat{B}$. Therefore, until head h leaves block $\hat{B}$, $\mathcal{A}$ will go through the same sequence of configurations as in its run on input $D(I, \overline{I})$. Item (c) ensures that when h leaves block $\hat{B}$, $\mathcal{A}$ is in the same configuration as in its runs on $D(I, \overline{I})$ and on $D(I', \overline{I'})$.

- Similarly, if h is a head on $\vec{T}^{\overline{I'}}$, then, due to item (b), no head on $\vec{S}^I$ is at an element in $\hat{B}$. Therefore, until head h leaves block $\hat{B}$, $\mathcal{A}$ will go through the same sequence of configurations as in its run on input $D(I', \overline{I'})$. Item (c) ensures that when h leaves block $\hat{B}$, $\mathcal{A}$ is in the same configuration as in its runs on $D(I', \overline{I'})$ and on $D(I, \overline{I})$.

In summary, in $\mathcal{A}$'s run on $D(I, \overline{I'})$, each time a head h has just left the last element in block $\hat{B}$ that it can access, it is in exactly the same configuration as in $\mathcal{A}$'s runs on $D(I, \overline{I})$ and on $D(I', \overline{I'})$ at the points in time where head h has just left the last element in block $\hat{B}$ that it can access. After the last head has left block $\hat{B}$, $\mathcal{A}$'s run on $D(I, \overline{I'})$ finishes in exactly the same way as $\mathcal{A}$'s runs on $D(I, \overline{I})$ and $D(I', \overline{I'})$. In particular, it accepts $D(I, \overline{I'})$ (since it accepts $D(I, \overline{I})$ and $D(I', \overline{I'})$). This, however, is a contradiction to (4.2). Thus, the proof of Theorem 4.3 is complete. ■

Remark 4.4. Let us compare the lower bound from Theorem 4.3 with the upper bound of Proposition 4.2: The upper bound tells us that *Disj*$_n$ can be solved by an mp2s-automaton with $n+2$ states and $\sqrt{n}$ forward heads on each input stream. The lower bound implies (for large enough n) that if just $\sqrt[5]{n}$ forward heads are available on each stream, not even $2^{\sqrt[3]{n}}$ states suffice for solving the problem *Disj*$_n$ with an mp2s-automaton.

Remark 4.5. A straightforward calculation shows that the assumptions of Theorem 4.3 are satisfied, for example, for all sufficiently large integers n and all integers m and k_f with $4k_f \leqslant \sqrt[4]{\frac{n}{\lg n}}$ and $\lg m \leqslant \frac{n}{4k_f \cdot (k_f^2+1)}$.

Theorem 4.3 can be generalized to the following lower bound for mp2s-automata where also backward heads are available:

Theorem 4.6. *For all n, m, k_f, k_b such that, for $k = 2k_f + 2k_b$ and $v = (k_f^2 + k_b^2 + 1) \cdot (2k_f k_b + 1)$,*

$$k^2 \cdot v \cdot \lg(n+1) \;+\; k \cdot v \cdot \lg m \;+\; v \cdot (1 + \lg v) \;\leqslant\; n\,,$$

the problem Disj$_n$ cannot be solved by any mp2s-automaton with parameters $(\mathbb{D}_n, m, k_f, k_b)$.

Proof. The overall structure of the proof is the same as in the proof of Theorem 4.3. We consider the same sets A^I, for all $I \subseteq \{1, .., n\}$. The stream $\vec{S}^I$ is chosen in the same way as in the proof of Theorem 4.3, i.e., for each $i \in I$, the stream $\vec{S}^I$ carries data item a_i at position i; and for each $i \notin I$, it carries data item b_i at position i.

Similarly as in the proof of Theorem 4.3, the stream $\vec{T}^I$ contains the same data items as $\vec{S}^I$. Now, however, the order in which the elements occur in $\vec{T}^I$ is a bit more elaborate. For fixing this order, we choose the following parameters:

$$v_1 \;:=\; k_f^2 + k_b^2 + 1\,, \qquad v_2 \;:=\; 2k_f k_b + 1\,, \qquad v \;:=\; v_1 \cdot v_2\,. \tag{4.4}$$

We subdivide the set $\{1, .., n\}$ into v_1 consecutive blocks $B_1, \ldots, B_v$ of equal size $\frac{n}{v_1}$. I.e., for each $j \in \{1, .., v_1\}$, block B_j consists of the indices in $\{\,(j-1)\frac{n}{v_1} + 1,\, \ldots,\, j\frac{n}{v_1}\,\}$.
Afterwards, we further subdivide each block B_j into v_2 consecutive subblocks of equal size $\frac{n}{v}$. These subblocks are denoted $B_j^1, \ldots, B_j^{v_2}$. Thus, each subblock $B_j^{j'}$ consists of the indices in $\{\,(j-1)\frac{n}{v_1} + (j'-1)\frac{n}{v} + 1,\, \ldots,\, (j-1)\frac{n}{v_1} + j'\frac{n}{v}\,\}$.

Now let π be the permutation of $\{1, .., n\}$ which maps, for all j, r with $1 \leqslant j \leqslant v_1$ and $1 \leqslant r \leqslant \frac{n}{v_1}$, element $(j-1)\frac{n}{v_1} + s$ onto element $(v_1 - j)\frac{n}{v_1} + s$. Thus, π maps elements in block B_j onto elements in block B_{v_1-j+1}, and inside these two blocks, π maps the elements of subblock

$B_j^{j'}$ onto elements in subblock $B_{v_1-1+1}^{j'}$. Note that π reverses the blocks B_j in order, but it does *not* reverse the order of the subblocks $B_j^{j'}$.

Finally, we are ready to fix the order in which the elements in A^I occur in the stream $\vec{T}^I$: For each $i \in I$, the stream $\vec{T}^I$ carries data item a_i at position $\pi(i)$; and for each $i \notin I$, it carries data item b_i at position $\pi(i)$.

In the same way as in the proof of Theorem 4.3, we write $D(I_1, I_2)$ to denote the input instance $\vec{S}^{I_1}$ and $\vec{T}^{I_2}$.

A pair of heads (h_S, h_T) is called *mixed* if one of the heads is a forward head and the other is a backward head. Since π reverses the order of the blocks $B_1, .., B_{v_1}$, it is straightforward to see that every *non-mixed* pair of heads can check at most one of the blocks $B_1, .., B_{v_1}$. Since there are v_1 blocks, but only $(v_1 - 1)$ non-mixed pairs of heads, we know that for all $I_1, I_2 \subseteq \{1, .., n\}$ there exists a block B_j that is *not checked* by any non-mixed pair of heads during $\mathcal{A}$'s run on input $D(I_1, I_2)$.

The same averaging argument as in the proof of Theorem 4.3 thus tells us that there is a $j_1 \in \{1, .., v_1\}$ and a set $X_0' \subseteq \{I : I \subseteq \{1, .., n\}\}$ such that

- for each $I \in X_0'$, block B_{j_1} is not checked by any non-mixed pair of heads during $\mathcal{A}$'s run on input $D(I, \overline{I})$, and
- $|X_0'| \geqslant \frac{2^n}{v_1}$.

From our particular choice of π, it is straightforward to see that every *mixed* pair of heads can check at most one of the subblocks $B_{j_1}^1, \ldots, B_{j_1}^{v_2}$. Since there are v_2 such subblocks, but only $(v_2 - 1)$ mixed pairs of heads, there must be a $j_2 \in \{1, .., v_2\}$ and a set $X_0 \subseteq X_0'$ such that

- for each $I \in X_0$, subblock $B_{j_1}^{j_2}$ is not checked by any pair of heads during $\mathcal{A}$'s run on input $D(I, \overline{I})$, and
- $|X_0| \geqslant \frac{|X_0'|}{v_2} \geqslant \frac{2^n}{v}$.

For the remainder of the proof we fix $\hat{B} := B_{j_1}^{j_2}$, and we let $k := 2k_f + 2k_b$ denote the total number of heads. Using these notations, the rest of the proof can be taken vertatim from the proof of Theorem 4.3. ■

The proof of Theorem 4.6 is implicit in [14] (see Theorem 5.11 in [14]). There, however, the proof is formulated in the terminology of a different machine model, the so-called *finite cursor machines*.

5. Final remarks

Several questions concerning the computational power of mp2s-automata occur naturally. On a technical level, it would be nice to determine the exact complexity of the set disjointness problem with respect to mp2s-automata. In particular: Is the upper bound provided by Proposition 4.2 optimal? Can backward scans significantly help for solving the set disjointness problem? Are $\sqrt{n}$ heads really necessary for solving the set disjointness problem when only a sub-exponential number of states are available?

A more important task, however, is to consider also randomized versions of mp2s-automata, to design efficient randomized approximation algorithms for particular problems, and to develop techniques for proving lower bounds in the randomized model.

Acknowledgement. I would like to thank Georg Schnitger for helpful comments on an earlier version of this paper.

References

[1] G. Aggarwal, M. Datar, S. Rajagopalan, and M. Ruhl. On the streaming model augmented with a sorting primitive. In *Proc. FOCS'04*, pages 540–549, 2004.
[2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58:137–147, 1999.
[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. PODS'02*, pages 1–16, 2002.
[4] Z. Bar-Yossef, M. Fontoura, and V. Josifovski. On the memory requirements of XPath evaluation over XML streams. In *Proc. PODS'04*, pages 177–188, 2004.
[5] Z. Bar-Yossef, M. Fontoura, and V. Josifovski. Buffering in query evaluation over XML streams. In *Proc. PODS'05*, pages 216–227, 2005.
[6] P. Beame and D.-T. Huynh-Ngoc. On the value of multiple read/write streams for approximating frequency moments. In *Proc. FOCS'08*, 2008.
[7] P. Beame, T. S. Jayram, and A. Rudra. Lower bounds for randomized read/write stream algorithms. In *Proc. STOC'07*, pages 689–698, 2007.
[8] C. Y. Chan, P. Felber, M. Garofalakis, and R. Rastogi. Efficient filtering of XML documents with XPath expressions. *VLDB Journal*, 11(4):354–379, 2002.
[9] T. M. Chan and E. Y. Chen. Multi-pass geometric algorithms. *Discrete & Computational Geometry*, 37(1):79–102, 2007.
[10] A. Czumaj and C. Sohler. Subliner-time algorithms. *Bulletin of the EATCS*, 89:23–47, 2006.
[11] C. Demetrescu, I. Finocchi, and A. Ribichini. Trading off space for passes in graph streaming problems. In *Proc. SODA'06*, pages 714–723, 2006.
[12] A. Gál and P. Gopalan. Lower bounds on streaming algorithms for approximating the length of the longest increasing subsequence. In *Proc. FOCS'07*, pages 294–304, 2007.
[13] T. Green, A. Gupta, G. Miklau, M. Onizuka, and D. Suciu. Processing XML streams with deterministic automata and stream indexes. *ACM Transactions on Database Systems*, 29(4):752–788, 2004.
[14] M. Grohe, Y. Gurevich, D. Leinders, N. Schweikardt, J. Tyszkiewicz, and J. Van den Bussche. Database query processing using finite cursor machines. *Theory of Computing Systems*, 2009. To appear. A preliminary version can be found in *Proc. ICDT'07*, pages 284–298.
[15] M. Grohe, A. Hernich, and N. Schweikardt. Randomized computations on large data sets: Tight lower bounds. In *Proc. PODS'06*, pages 243–252, 2006. Full version available as CoRR Report, arXiv:cs.DB/0703081.
[16] M. Grohe, C. Koch, and N. Schweikardt. Tight lower bounds for query processing on streaming and external memory data. Accepted at *Theoretical Computer Science*, special issue for selected papers from ICALP'05.
[17] M. Grohe and N. Schweikardt. Lower bounds for sorting with few random accesses to external memory. In *Proc. PODS'05*, pages 238–249, 2005.
[18] S. Guha and A. McGregor. Tight lower bounds for multi-pass stream computation via pass elimination. In *Proc. ICALP'08*, pages 760–772, 2008.
[19] Y. Gurevich, D. Leinders, and J. Van den Bussche. A theory of stream queries. In *Proc. DBPL*, pages 153–168, 2007.
[20] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. In *External memory algorithms*, volume 50, pages 107–118. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1999.
[21] A. McGregor et al. Open problems in data streams and related topics, December 2006. IITK Workshop on Algorithms for Data Streams. `http://www.cse.iitk.ac.in/users/sganguly/workshop.html`.
[22] J. Munro and M. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.
[23] S. Muthukrishnan. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.
[24] M. Ruhl. *Efficient Algorithms for New Computational Models.* PhD thesis, Massachusetts Institute of Technology, 2003.
[25] N. Schweikardt. Machine models and lower bounds for query processing. In *Proc. PODS'07*, pages 41–52, 2007.

[26] L. Segoufin and C. Sirangelo. Constant-memory validation of streaming XML documents against DTDs. In *Proc. ICDT'07*, pages 299–313, 2007.
[27] L. Segoufin and V. Vianu. Validating streaming XML documents. In *Proc. PODS'02*, pages 53–64, 2002.
[28] M. Shalem and Z. Bar-Yossef. The space complexity of processing XML twig queries over indexed documents. In *Proc. ICDE'08*, pages 824–832, 2008.

This work is licensed under the Creative Commons Attribution-NoDerivs License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nd/3.0/.