

Machine Models and Lower Bounds for Query Processing

Nicole Schweikardt

Institut für Informatik, Humboldt-Universität zu Berlin, Germany
schweika@informatik.hu-berlin.de

ABSTRACT

This paper gives an overview of recent work on machine models for processing massive amounts of data. The main focus is on generalizations of the classical data stream model where, apart from an “internal memory” of limited size, also a number of (potentially huge) streams may be used as “external memory devices”.

Categories and Subject Descriptors

F.1.1 [Computation by Abstract Devices]: Models of Computation; F.1.3 [Computation by Abstract Devices]: Complexity Measures and Classes; H.2.3 [Database Management]: Query Languages

General Terms

Theory, Languages, Algorithms

Keywords

machine models, data streams, external memory, complexity, lower bounds, query processing, XML, survey

1. INTRODUCTION

The massive data sets that have to be processed in many application areas are often far too large to fit completely into a computer’s internal memory. For such applications on massive amounts of data, two different scenarios have been extensively studied in recent years:

(1) *External memory processing:*

This scenario considers data that is stored in external memory. When processing such data, the resulting input/output communication between fast internal memory and slower external memory is a major performance bottleneck. There has been a wealth of research on the design and analysis of so-called external memory algorithms which aim at optimizing the costs produced by external memory accesses.

An overview of external memory algorithms was given in an invited tutorial at PODS’98 [35]; more details can be found in the survey [36] and the books [1, 27].

(2) *Data stream processing:*

This scenario considers data that is not stored but, instead, arrives as a stream and has to be processed on-the-fly by using only a limited amount of memory. 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.

A systems-oriented overview of models and issues concerning data streams was given in an invited talk at PODS’02 [4]; an excellent overview of efficient algorithms for data stream processing can be found in [28].

The present paper intends to give an overview of some recent work concerning the theoretical foundations of these two scenarios. It begins with Section 2 by introducing a number of basic computational problems that will serve as running examples throughout the paper. Section 3 gives a short introduction to the classical *data stream model* and to the area of communication complexity. Section 4 concentrates on the computational model of *read/write-streams*, which can be viewed as a model for stream-based external memory processing. Section 5 briefly refers to three related models of computation, namely the *finite cursor machines*, the *StrSort model*, and the *Parallel Disk Model*. Finally, Section 6 concludes the paper by pointing out some directions for future research.

2. PRELIMINARIES

We write $\mathbb{N}$ to denote the set of natural numbers, i.e., the non-negative integers. For $i, j \in \mathbb{N}$ with $i \leq j$ we use $[i, j]$ to denote the interval $\{k \in \mathbb{N} : i \leq k \leq j\}$.

As running examples throughout the entire article, we will take a closer look at the problem of *sorting* a given set of data items and at the problem of deciding whether two sets of data items are *disjoint*. Note that these two problems are of fundamental importance for query processing: Efficient query evaluation often relies on intermediate sorting steps; and already the easiest kind of *join* (i.e., the join of two *unary* relations) corresponds to computing the intersection of two sets. Formally, we consider the problem of sorting a list of bitstrings, represented as follows:

Sorting:

Instance: $v_1\# \dots v_m\#$, where $m \geq 1$ and $v_1, \dots, v_m \in \{0, 1\}^*$

Output: $v_{\psi(1)}\# \dots v_{\psi(m)}\#$, where ψ is a permutation of $\{1, \dots, m\}$ such that $v_{\psi(1)} \leq \dots \leq v_{\psi(m)}$ (with $\leq$ denoting the lexicographic order).

The input instances for the *disjointness* problem, and also for a number of related decision problems, are encoded as follows:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PODS’07, June 11–13, 2007, Beijing, China.

Copyright 2007 ACM 978-1-59593-685-1/07/0006 ...\$5.00.

Instance: $v_1 \# \dots \# v_m \# v'_1 \# \dots \# v'_m \#$,
where $m \geq 1$ and $v_1, \dots, v_m, v'_1, \dots, v'_m \in \{0, 1\}^*$.

The tasks are:

Set-Disjointness: Decide if $\{v_1, \dots, v_m\} \cap \{v'_1, \dots, v'_m\} = \emptyset$.
(We speak of a “yes”-instance if the two sets are disjoint,
and of a “no”-instance otherwise.)

Check-Sort: Decide if $v'_1, \dots, v'_m$ is the lexicographically sorted
(in ascending order) version of $v_1, \dots, v_m$.

Multiset-Equality: Decide if the multisets $\{v_1, \dots, v_m\}$ and
 $\{v'_1, \dots, v'_m\}$ are equal (i.e., they contain the same ele-
ments with the same multiplicities).

Set-Equality: Decide if $\{v_1, \dots, v_m\} = \{v'_1, \dots, v'_m\}$ (with re-
spect to ordinary set semantics).

For all kinds of problems we usually let

N denote the total length of the input,

i.e. $N = m + \sum_{i=1}^m |v_i|$ for the *Sorting* problem, and $N = 2m + \sum_{i=1}^m (|v_i| + |v'_i|)$ for the related decision problems. Furthermore, in our proofs we will mainly consider instances where all the v_i and v'_i have the same length n , so that $N = m \cdot (n+1)$ for the *Sorting* problem, and $N = 2m \cdot (n+1)$ for the related decision problems.

3. DATA STREAMS

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.

This section gives a brief introduction to this computation model. We start with two examples in Subsection 3.1. Afterwards, Subsection 3.2 gives a quick introduction to the area of *communication complexity*, which provides powerful tools for proving lower bounds on the memory requirement of data stream algorithms. Finally, Subsection 3.3 gives pointers to further topics and literature related with data streams.

3.1 Two Examples

Let us start with an easy example, the *missing number puzzle*, that is basically taken from [28], and that deals with the following problem: Given a fixed natural number n , the input consists of a stream of values $v_1, v_2, \dots, v_{n-1}$ such that $\{v_1, \dots, v_{n-1}\}$ is a subset of $[1, n]$ of size $n-1$. The task is to find out which of the numbers from $[1, n]$ is missing.

The straightforward idea for solving this problem is to store a bitstring B of length n which is initialized to 0^n . When receiving the value v_i in the i -th computation step, B is updated by setting the v_i -th bit to 1. Then, after $n-1$ steps, the missing number obviously is the (uniquely defined) value j for which the j -th bit of B is 0. Of course, for storing B , n bits of memory are needed.

This naive approach, however, is far from being optimal. An alternative, more (space-)efficient way of solving the missing number puzzle is to maintain a running sum S which, after the i -th computation step, holds the sum $v_1 + \dots + v_i$ of all the values seen so far. Since the sum of *all* values from $[1, n]$ is $\frac{n(n+1)}{2}$, we know that after $n-1$ computation steps the missing number is simply the value $j := \frac{n(n+1)}{2} - S$. For storing S , obviously $2 \cdot \lg n$ bits of memory suffice.

This is almost optimal, as the missing number puzzle cannot be solved with less than $\lg n$ bits of memory. To see this, it suffices to note that there are n candidates for the missing number. If the memory buffer consists of less than $\lg n$ bits, then there must exist two different input streams $\bar{v}$ and $\bar{v}'$ with two different missing numbers j and j' , respectively, such that the memory content after having read $\bar{v}$ is the same as the memory content after having read $\bar{v}'$. Then, however, it is impossible to decide whether j or j' is the actually missing number.

Let us now turn to a more elaborate example concerning the *Multiset-Equality* problem (defined in Section 2). By using standard fingerprinting techniques it is not difficult to obtain the following

THEOREM 3.1. *There is a randomized data stream algorithm which, when given the parameters m and n , solves the Multiset-Equality problem with internal memory of size $O(\lg N)$ such that each*

- “yes”-instance is accepted with probability 1,
- “no”-instance is rejected with probability $\geq 2/3$.

PROOF (SKETCH). The basic idea is to identify the input strings v_i, v'_i with natural numbers and to associate two polynomials

$$f(x) := \sum_{i=1}^m x^{v_i} \quad \text{and} \quad g(x) := \sum_{i=1}^m x^{v'_i}$$

with the input $v_1 \# \dots \# v_m \# v'_1 \# \dots \# v'_m \#$. Obviously, the two polynomials are equal if, and only if, the input is a “yes”-instance of the *Multiset-Equality* problem.

The basic idea of the algorithm now is to choose a random number r , to evaluate $f(r)$ and $g(r)$, and to accept if, and only if, $f(r) = g(r)$. This algorithm will accept with probability 1 if the input is a “yes”-instance. However, if the input is a “no”-instance, the two polynomials f and g only have few common points, and thus the algorithm will reject with high probability.

A closer look shows that this procedure can be implemented as a data stream algorithm which computes $f(r)$ and $g(r)$ on-the-fly and which uses arithmetic modulo a prime number of moderate size such that $O(\lg N)$ bits of memory suffice. Details can be found in [19]. $\square$

It is a natural question to ask if *randomization* is really necessary here — and in fact it turns out that without randomization, exponentially larger internal memory is inevitable:

PROPOSITION 3.2. *Every deterministic data stream algorithm which, when given the parameters m and n , solves the Multiset-Equality problem, requires internal memory of size $\Omega(N)$.*

Proving this lower bound is fairly easy when employing results on *communication complexity*, introduced in the next subsection.

3.2 Communication Complexity

This subsection gives a quick introduction to some basic notions of communication complexity. A detailed treatment of this subject can be found, e.g., in the textbook [26].

Let X, Y be finite sets and $F : X \times Y \rightarrow \{\text{“yes”}, \text{“no”}\}$ a function. In Yao’s basic model of communication, two players, Alice and Bob, jointly want to evaluate the value $F(A, B)$ for input values $A \in X$ and $B \in Y$, where Alice only knows A and Bob only knows B . The two players can exchange messages according to some fixed protocol $\mathcal{P}$ that depends on F . The exchange of messages starts with Alice sending a message to Bob and ends as soon as one of

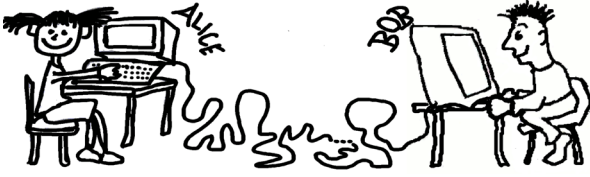


Figure 1: Basic Model of Communication

the players has enough information on A and B to compute $F(A, B)$. The *cost* of $\mathcal{P}$ on input (A, B) is the number of bits communicated by $\mathcal{P}$ on input (A, B) ; and the *cost* of $\mathcal{P}$ is the maximal cost of $\mathcal{P}$ over all inputs $(A, B) \in X \times Y$. The *communication complexity* of F is defined as the minimum cost of $\mathcal{P}$ over all protocols $\mathcal{P}$ that compute F .

For some problems it is straightforward to obtain lower bounds on their communication complexity, for example:

PROPOSITION 3.3. *Every deterministic communication protocol for the problem*

*given two m -element subsets A, B of an n -element set,
decide if $A = B$*

requires at least $\lg \binom{n}{m}$ bits of communication.

PROOF (SKETCH). Obviously, there are $\binom{n}{m}$ “yes”-instances of the problem. Now, if $\mathcal{P}$ is a protocol with less than $\lg \binom{n}{m}$ bits of communication, there must be two distinct “yes”-instances (C, C) and (D, D) with exactly the same communication. Then, however, it is straightforward to see that also on input (C, D) the protocol $\mathcal{P}$ produces the same communication (and the same output) as on inputs (C, C) and (D, D) . Thus, on input (C, D) the protocol leads to the answer “yes”, although the correct answer would be “no”. $\square$

Note that the lower bound of Proposition 3.3 is optimal, as the following protocol solves the problem by using $\lceil \lg \binom{n}{m} \rceil$ bits of communication: Let $\ell := \binom{n}{m}$ and let $C_1, \dots, C_\ell$ be a list of all m -element subsets of the n -element set. If, on input $A = C_i$, Alice just sends the binary representation of i to Bob, Bob can reconstruct A and check whether $A = B$.

By using Proposition 3.3, one easily obtains a

PROOF OF PROPOSITION 3.2.

For contradiction let us assume that there is a deterministic data stream algorithm M that solves the *Multiset-Equality* problem with $o(N)$ bits of memory. This algorithm can be used to obtain a communication protocol which, given two m -element subsets A, B of $\{0, 1\}^n$ decides if $A = B$ as follows: Alice represents her input set $A = \{v_1, \dots, v_m\}$ by the string $\bar{a} := v_1\# \dots v_m\#$, whereas Bob represents his input set $B = \{v'_1, \dots, v'_m\}$ by the string $\bar{b} = v'_1\# \dots v'_m\#$.

Alice starts the communication protocol by starting the data stream algorithm M on input m, n , and the “partial input” $\bar{a} \dots$. She lets M run until the moment it tries to access the first symbol to the right of $\bar{a}$. Then she sends the current content of the internal memory to Bob. Now Bob has all the information needed to continue the execution of M on input $\dots \bar{b}$ until the algorithm has consumed the entire stream “ $\bar{a}\bar{b}$ ” and stops, deciding whether $\bar{a}\bar{b}$ is a “yes”-instance of the *Multiset-Equality* problem and thus deciding whether $A = B$.

During this protocol, only $o(N)$ bits are communicated between Alice and Bob (namely, the content of the internal memory after having consumed the prefix $\bar{a}$ of the input stream), where N denotes the length of the string $\bar{a}\bar{b}$.

Now, for the specific case where m is a power of 2 and $n = 2 \cdot \lg m$, we have $N = \theta(m \cdot \lg m)$ and thus we have got a deterministic communication protocol which uses $o(m \cdot \lg m)$ bits of communication to decide, given two m -element subsets A, B of a 2^n -element set, whether $A = B$. This, however, is a contradiction to Proposition 3.3 which tells us that any such protocol requires at least

$$\lg \binom{2^n}{m} = \lg \binom{m^2}{m} \geq \lg(m^m) = m \cdot \lg m$$

bits of communication. $\square$

Another, much more involved, communication complexity result that will be used in later sections of this paper, is Razborov’s following theorem (cf. Example 1.23 of [26] for an elegant proof):

THEOREM 3.4 (RAZBOROV). *Every deterministic communication protocol for the problem*

*given two m -element subsets A, B of an n -element set,
decide if A and B are disjoint*

requires at least $\Omega(\lg \binom{n}{m})$ bits of communication.

3.3 Some Pointers to the Literature

Communication complexity has been used for showing lower bounds on the memory requirement of data stream algorithms in many places, see e.g. [23, 3, 7, 6], sometimes also in connection with algorithms that may perform *multiple* passes over the data, e.g., [23, 20, 12].

Detailed overviews of algorithms on data streams, respectively of the related area of sublinear algorithms are given in [28, 11]. I would like to emphasize that many of the sophisticated data stream algorithms achieve a surprisingly good performance. For example, it is not at all obvious how to maintain information on the number of *distinct* elements that occurred in a stream, without storing a list of all those elements (since, intuitively, for each element that arrives one has to check whether this is a *new* element or just the repetition of an element that had already occurred in the stream). Here, randomized algorithms are known which give good approximate solutions to this problem while using just a logarithmic number of bits (see [3] for details).

Note that also from the point of view of database systems, efficient data stream algorithms are very desirable. For example, virtually all query optimization methods in relational database systems rely on information about, e.g., 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 when the database is updated. The seminal paper [3] provides efficient data stream algorithms for these tasks.

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 massive amounts of transient data have to be processed. To get an overview of this research area, [4] is a good starting point.

In the context of XML query processing and validation, stream-based approaches have been examined in detail in the last few years, see e.g. the papers [34, 33, 16, 9, 13, 30, 7, 6, 20, 25].

4. READ/WRITE STREAMS

It is generally assumed that databases have to reside in *external*, inexpensive storage because of their sheer size. Current technology for external storage systems (disks and tapes) presents us with a situation where a small number of *sequential scans* of the external memory devices is strictly preferable over *random accesses* to

external memory: A random access to a hard disk requires to move the read/write head to a certain position of the disk — and this is a comparably slow mechanical operation. Indeed, during the time required for a single such *random* access, a considerable amount of data stored *in sequence*, starting at the current position of the disk’s read/write head could have been processed. Modern software and database technology uses clever heuristics to minimize the costs produced by external memory accesses.

While the previous section’s *data stream* model restricted attention to *internal* memory, the present section concentrates on a machine model for *external memory processing* which was introduced in [20, 22] and which has available

- *internal memory* that can be accessed very fast, but that is limited in size, and, additionally,
- *external memory* that can store huge amounts of data, which can easily be accessed (for reading as well as for writing) in a sequential way, but for which *random accesses* are expensive.

Formally, this model is based on a standard multi-tape Turing machine. Some of the tapes of the machine, among them the input tape, represent the external memory (each of these tapes can be viewed as representing an external memory device such as, e.g., a hard disk). They are unrestricted in size, but access to these tapes is restricted by allowing only a certain number $r(N)$ of reversals of the head directions (where N denotes the size of the input). This may be seen as a way of (a) restricting the number of sequential scans and (b) restricting random accesses to these tapes (because each random access can be simulated by moving the head to the desired position on a tape, and this involves at most two head reversals). The remaining tapes of the Turing machine represent the internal memory. Access to these internal memory tapes is unrestricted, but their size is bounded by a parameter $s(N)$. Such Turing machines will be called (r, s, t) -bounded, if t is the total number of external memory tapes of the machine.

It will often be convenient to adopt Beame, Jayram, and Rudra’s [8] informal view of this as being a computation model where, in addition to some internal memory, a constant number of *read/write-streams* are available — each such read/write-stream corresponds to an external memory tape of the Turing machine. The resources of interest are

- (1) the number $r(N)$ of scans of the read/write-streams (where N denotes the size of the input),
- (2) the size $s(N)$ of internal memory, and
- (3) the total number t of read/write-streams that are available.

Adopting the terminology of [8], we will henceforth call this the *computational model of read/write-streams*, and we will sometimes informally say that there is an

(r, s, t) -bounded algorithm on read/write-streams

for a specific problem, iff this problem can be solved by an (r, s, t) -bounded Turing machine.

REMARK 4.1. Note that this model allows *forward* scans as well as *backward* scans of the external memory tapes and, even more, it allows the read/write heads to reverse direction in the middle of a tape. While it is realistic to assume that in current “real-world” hard disks, sequential *forward* scans can be performed very efficiently, this is not the case for *backward* scans. Thus, when aiming at designing efficient algorithms for read/write-streams, one should make sure that the algorithms essentially use only forward

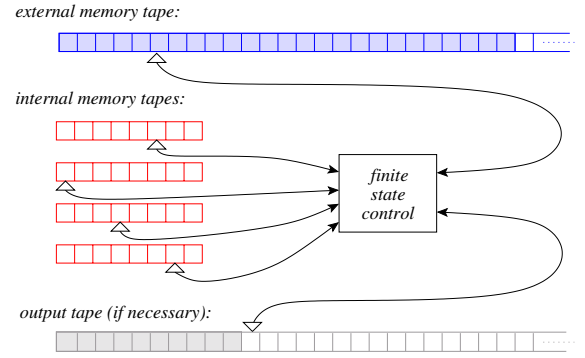


Figure 2: Turing machine for (r, s) -bounded algorithms on a read/write-stream

scans of the read/write-streams. On the other hand, when aiming at *lower bounds*, i.e., showing that some problem can *not* be solved by any (r, s, t) -bounded algorithm on read/write-streams, allowing for forward scans as well as backward scans makes our lower bound results only stronger.

Having in mind the motivation that (r, s, t) -bounded algorithms on read/write-streams serve as a computation model for external memory processing, we are mainly interested in cases where the size $s(N)$ of the internal memory is considerably smaller than the input size N , and the number $r(N)$ of sequential scans of (or, correspondingly, the number of random accesses to) external memory is, preferably, as small as possible.

The computation model of read/write-streams has been investigated in the articles [20, 22, 19, 18, 24, 8]; a preliminary survey was given in [21]. The next two subsections summarize what is known about this model, first concentrating on the case where only *one* read/write-stream is available, and then moving over to the general case where an arbitrary number of read/write-streams may be used.

4.1 One Read/Write Stream

This subsection concentrates on the variant of the read/write-streams model where only *one* read/write stream is available.

4.1.1 The Model

Formally, this model relies on Turing machines with

- an input tape, which is a read/write tape and will henceforth be called *external memory tape*,
- an arbitrary number u of work tapes, which will henceforth be called *internal memory tapes*, and
- if needed, an additional write-only output tape,

see Figure 2. Let M be such a Turing machine and let ρ be a run of M . By $rev_{ext}(\rho)$ we denote the number of times the *external memory tape*’s head changes its direction during the run ρ . Furthermore, we let $space_{int}(\rho)$ denote the total number of cells on the *internal memory tapes* that are used by ρ .

M is called (r, s) -bounded for some functions $r, s : \mathbb{N} \rightarrow \mathbb{N}$, if for every $N \in \mathbb{N}$, every input w for M of length N , and every run ρ of M with input w , the following is true:

- (1) ρ is finite,
- (2) $1 + rev_{ext}(\rho) \leq r(N)$, and¹

¹We add 1 to the number $rev_{ext}(\rho)$ of head reversals, because then $r(N)$ really bounds the number of sequential (forward or backward) scans of the external memory tape.

(3) $\text{space}_{\text{int}}(\rho) \leq s(N)$.

Often we will informally say that a problem

belongs to $\text{ST}(r, s)$

or, equivalently, that it is solved by an (r, s) -bounded algorithm on a read/write-stream iff there is an (r, s) -bounded Turing machine which solves that problem.

Note that a priori there is no restriction on the running time or the size of the external memory tape of an (r, s) -bounded Turing machine. However, an easy induction on $r(N)$ shows that implicitly these two parameters are bounded in terms of the head reversals, the internal memory space, and the input size:

PROPOSITION 4.2 ([21]). *Let $r, s : \mathbb{N} \rightarrow \mathbb{N}$, and let M be an (r, s) -bounded Turing machine. Then the length of every run of M on an input of length N , and the space used on M 's external memory tape during such a run, are*

$$\leq (N + r(N)) \cdot r(N) \cdot 2^{O(s(N))}.$$

For the particular case where $r(N) = 1$ for all $N \in \mathbb{N}$, $(1, s)$ -bounded algorithms on a read/write-stream precisely correspond to conventional data stream algorithms described in Section 3, where only a single forward scan over the input data is available.

Obviously, a multi-pass data stream algorithm which performs p passes over the input stream and uses internal memory of size s can, in particular, be viewed as a $(2p, s)$ -bounded algorithm on a read/write-stream (which never *writes* to the external memory tape and which uses backward scans just for moving the head back to the first position of the input).

4.1.2 Lower Bounds

Similarly to lower bounds for the data stream model, lower bounds for (r, s) -bounded algorithms on a read/write-stream can be obtained fairly easily by employing known results from *communication complexity*. The basic idea for such proofs is to divide the external memory tape into two parts. Obviously, during an (r, s) -bounded computation on an input of size N , the border between the two parts of the external memory tape can be crossed at most $r(N)$ times. Each time we cross this border, we can only “transport” the amount of information that is currently stored in internal memory, and this consists of at most $O(s(N))$ bits. Consequently, during an entire (r, s) -bounded computation, only $O(r(N) \cdot s(N))$ bits of information can be communicated between the two parts of the external memory tape. Thus, in a similar way as in the proof of Proposition 3.2 (see Section 3.2), lower bounds known from communication complexity almost immediately imply corresponding lower bounds for (r, s) -bounded algorithms on a read/write-stream.

For example, the communication complexity lower bounds from Proposition 3.3 and Theorem 3.4 easily lead to

THEOREM 4.3 ([20]). *For all functions r, s with $r(N) \cdot s(N) = o(N)$, none of the problems *Set-Disjointness*, *Multiset-Equality*, *Set-Equality*, *Check-Sort* belongs to $\text{ST}(r, s)$.*

This result deals with *decision* problems. When considering problems like *Sorting* that produce an output other than just the answer “yes” or “no”, we have to deal with Turing machines that, apart from the external memory tape and the internal memory tapes, have an additional *write-only* output tape, i.e., the output is produced as a data stream. In this case, it is still possible to obtain lower bound proofs for (r, s) -bounded Turing machines by using communication complexity lower bounds (although the reduction to communication complexity is not quite as immediate as in the proof of Theorem 4.3). An example of such a bound is

THEOREM 4.4 ([20]). *If r, s are functions with $r(N) \cdot s(N) = o(N)$ and $s(N) \geq 2 \cdot \lg N$, then *Sorting* $\notin \text{ST}(r, s)$.*

4.1.3 Consequences for Query Processing

By using the lower bound of Theorem 4.3 for the *Set-Disjointness* problem, one immediately obtains that evaluating *relational algebra queries* is hard for algorithms on a read/write-stream: Already the basic task of checking whether the join of two relations A and B is empty cannot be performed by an (r, s) -bounded algorithm on a read/write-stream if $r(N) \cdot s(N)$ is of size $o(N)$, where N denotes the number of bits used for storing the two input relations A and B . — To see this, it suffices to note that if A and B are unary relations, then $A \bowtie B = A \cap B$.

In a similar way one also obtains lower bounds for evaluating queries against XML data. For example, two unary relations A and B can be encoded by an XML document in a straightforward way such that the query which asks if the join of A and B is empty can be formalized in the language *XQuery*. This immediately leads to the result stating that there exists an *XQuery* query Q such that, for all functions r, s with $r(N) \cdot s(N) = o(N)$, there is no (r, s) -bounded algorithm on a read/write-stream which, when getting as input an XML document D of length N , checks whether the result of Q on D is empty.

When considering the node-selecting XML query language *Core XPath* (see [15]), let us write $Q(D)$ to denote the set of nodes that Q selects in an XML document D . We obtain the following tight bound on the complexity of processing *Core XPath* queries.

THEOREM 4.5 ([20]).

- (a) *For every Core XPath query Q there is a data stream algorithm (i.e., it performs a single pass over the input) which, when given as input an XML document D , decides whether $Q(D) \neq \emptyset$. This algorithm uses internal memory space $O(h(D))$, where $h(D)$ denotes the height of the tree representation of D .*
- (b) *There is a Core XPath query Q such that for all functions r, s with $r(H) \cdot s(H) = o(H)$, there exists no algorithm on a read/write-stream which, when given as input an XML document D , decides whether $Q(D) \neq \emptyset$ and uses at most $r(h(D))$ head reversals and internal memory space of size at most $s(h(D))$.*

The upper bound (a) is proved by standard automata theoretic techniques. For the lower bound (b), we can again use the lower bound for the *Set-Disjointness* problem from Theorem 4.3. More details on this topic can be found in the article [20] and in the survey [21].

4.1.4 A Hierarchy Based on the Number of Scans

Of course it is natural to expect that increasing the allowed number of head reversals strictly increases the computational power of algorithms on a read/write-stream. In fact it even turns out that allowing a single extra head reversal may be more powerful than significantly increasing the internal memory space:

THEOREM 4.6 ([20, 24]). *For every logspace-computable function r with $r(N) = o(N/(\lg N)^2)$ there exists a decision problem that*

- *can be solved by an $(r(N)+1, O(\lg N))$ -bounded algorithm on a read/write-stream, but that*
- *cannot be solved by any $(r(N), o(\frac{N}{r(N) \cdot \lg N}))$ -bounded algorithm on a read/write-stream.*

The proof relies on a sophisticated result by Nisan and Wigderson [29] on the so-called k -round communication complexity of a particular “pointer jumping” problem.

4.2 Several Read/Write Streams

Let us now turn to the computation model where an arbitrary number t of read/write-streams are available.

4.2.1 The Model

Formally, this model is based on multi-tape Turing machines in the same way as the model in Section 4.1, with the exception that

- instead of a single external memory tape, now the machine has t external memory tapes,
- at the beginning of the computation, the input is given on the first external memory tape,
- there is no extra output tape, but instead, if needed, the output is given by the content of the last external memory tape at the end of the computation.

Now, if M is such a Turing machine and ρ is a run of M , then $rev_{ext}(\rho)$ denotes the sum $R_1 + \dots + R_t$, where R_i is the number of times the head on the i -th external memory tape changes its direction during the run ρ . In the same way as in Section 4.1.1, $space_{int}(\rho)$ denotes the internal memory space used by ρ .

M is called (r, s, t) -bounded for some functions $r, s : \mathbb{N} \rightarrow \mathbb{N}$, if M has t external memory tapes and for every $N \in \mathbb{N}$, every input word w for M of length N , and every run ρ of M with input w , the following is true: ρ is finite, $1 + rev_{ext}(\rho) \leq r(N)$, and $space_{int}(\rho) \leq s(N)$. We will informally say that a problem is solved by an

(r, s, t) -bounded algorithm on read/write-streams

iff there is an (r, s, t) -bounded Turing machine which solves that problem.

In addition to *deterministic* algorithms we will in this subsection also consider *nondeterministic* and *randomized* (r, s, t) -bounded algorithms on read/write-streams. The *nondeterministic* algorithms are based on nondeterministic Turing machines in the obvious way. The *randomized* algorithms, intuitively, have the possibility that in each computation step a coin may be tossed to decide what to do in this step. This can be formalized in a straightforward way: A *randomized Turing machine* (RTM, for short) is simply a nondeterministic Turing machine M to which “acceptance probabilities” are associated as follows: For a configuration γ of M , we write $Next_M(\gamma)$ to denote the set of all configurations γ' that can be reached from γ in a single step. Each such configuration $\gamma' \in Next_M(\gamma)$ is chosen with uniform probability, i.e., $\Pr(\gamma \rightarrow_M \gamma') = 1/|Next_M(\gamma)|$. For a run $\rho = (\gamma_1, \dots, \gamma_\ell)$, the probability that M performs run ρ is the product of the probabilities $\Pr(\gamma_i \rightarrow_M \gamma_{i+1})$, for all $i < \ell$. For an input word w , the probability

$$\Pr(M \text{ accepts } w)$$

that M accepts w is defined as the sum of $\Pr(\rho)$ over all *accepting* runs ρ of M on input w . Based on varying error probabilities allowed for randomized read/write-stream algorithms, we consider the following classes which are the (r, s, t) -bounded analogues of the classical complexity classes (cf. e.g. [31, 5]) P, NP, RP, co-RP, and BPP.

DEFINITION 4.7 (ST, NST, RST, co-RST, BPST).

Let $r, s : \mathbb{N} \rightarrow \mathbb{N}$ and $t \in \mathbb{N}$.

- $ST(r, s, t)$ is the class of all decision problems solvable by an (r, s, t) -bounded deterministic algorithm on read/write-streams.
- $NST(r, s, t)$ is the class of all decision problems that can be solved by an (r, s, t) -bounded nondeterministic algorithm on read/write-streams.

- $RST(r, s, t)$ is the class of all decision problems solvable by an (r, s, t) -bounded randomized algorithm on read/write-streams for which there exists a δ with $0 \leq \delta < 1/2$ such that each

- “yes”-instance is accepted with probability $\geq 1 - \delta$
- “no”-instance is rejected with probability 1.

(I.e., the algorithm has 1-sided bounded error such that it produces no “false positives”, and it produces “false negatives” with probability $\leq \delta$.)

- $co-RST(r, s, t)$ is the class of all decision problems whose complements belong to $RST(r, s, t)$.

I.e., a problem belongs to $co-RST(r, s, t)$ iff it can be solved by an (r, s, t) -bounded randomized algorithm on read/write-streams for which there exists a δ with $0 \leq \delta < 1/2$ such that each

- “yes”-instance is accepted with probability 1
- “no”-instance is rejected with probability $\geq 1 - \delta$.

(I.e., the algorithm has 1-sided bounded error such that it produces no “false negatives”, and it produces “false positives” with probability $\leq \delta$.)

- $BPST(r, s, t)$ is the class of all decision problems solvable by an (r, s, t) -bounded randomized algorithm on read/write-streams for which there exists a δ with $0 \leq \delta < 1/2$ such each

- “yes”-instance is accepted with probability $\geq 1 - \delta$
- “no”-instance is rejected with probability $\geq 1 - \delta$.

(I.e., the algorithm has 2-sided bounded error such that it produces a wrong answer with probability $\leq \delta$.)

The following inclusions of classes are obvious:

$$ST(r, s, t) \subseteq RST(r, s, t) \subseteq NST(r, s, t)$$

$$RST(r, s, t), co-RST(r, s, t) \subseteq BPST(r, s, t).$$

When considering a problem F which, on input w , produces an output $F(w)$ other than just the answer “yes” or “no” (such as e.g. the *Sorting* problem), we will sometimes still say that “ F belongs to $ST(r, s, t)$ ” to indicate that there is an (r, s, t) -bounded deterministic algorithm on read/write-streams that solves F . Concerning randomized algorithms, we will say that “ F is solved by an (r, s, t) -bounded LasVegas-algorithm on read/write-streams” and write

$$F \in \text{LasVegas-RST}(r, s, t)$$

to indicate that there is an (r, s, t) -bounded randomized algorithm M on read/write-streams such that every run of M on every input instance w for F outputs either $F(w)$ or “I don’t know”, and there is a δ with $0 \leq \delta < 1/2$ such that for each input w the output $F(w)$ is produced with probability $\geq 1 - \delta$.

For classes R and S of functions we define

$$ST(R, S, t) := \bigcup_{r \in R, s \in S} ST(r, s, t),$$

$$ST(R, S, O(1)) := \bigcup_{t \in \mathbb{N}} ST(R, S, t).$$

Analogous notations are used for the classes NST, RST, co-RST, BPST and LasVegas-RST.

REMARK 4.8. The classes RST, co-RST, and *LasVegas*-RST were introduced in [19] where, instead of allowing arbitrary error probabilities $\delta < 1/2$ (as in Definition 4.7 above), a *fixed* error probability of $1/2$ was considered. Note that this leads to essentially the same classes, because by performing k independent runs of a randomized (r, s, t) -bounded algorithm, one can reduce the error probability from δ to δ^k . This does not change the size of the internal memory of the resulting algorithm, and it increases the number of head reversals by just a factor of k .

Let us now look at a couple of examples of problems that can be solved by algorithms on read/write-streams.

EXAMPLES 4.9.

- (a) Recall from Theorem 3.1 that there is a randomized data stream algorithm which, when given the parameters m and n , solves the *Multiset-Equality* problem with a single scan of the input and internal memory of size $O(\lg N)$ such that each
 - “yes”-instance is accepted with probability 1,
 - “no”-instance is rejected with probability $\geq 2/3$.

In particular (cf. Definition 4.7), this implies that

$$\begin{aligned} \text{Multiset-Equality} &\in \text{co-RST}(O(1), O(\lg N), 1) \\ &\subseteq \text{BPST}(O(1), O(\lg N), 1). \end{aligned}$$

- (b) A straightforward implementation of the *merge sort* algorithm shows that the problem of sorting m input strings each of which has length at most n can be solved efficiently by an algorithm that uses 3 read/write-streams, $O(\lg m)$ sequential scans (where, essentially, only *forward* passes are needed), and internal memory of size $O(n)$ (this is used for storing and comparing two of the input strings). Thus, e.g., the problem *Short-Sorting* of sorting (for arbitrary m) m strings of length at most $2 \cdot \lg m$ each, belongs to the class $\text{ST}(O(\lg N), O(\lg N), 3)$.

From results of Chen and Yap [10] it follows that already 2 read/write-streams suffice and no internal memory is necessary, and that thus even the general *Sorting* problem belongs to $\text{ST}(O(\lg N), O(1), 2)$.²

Since each of the problems *Check-Sort*, *(Multi)Set-Equality*, and *Set-Disjointness* can easily be solved by separately sorting the two halves of the input, storing them on two external memory tapes, and then comparing them in a single scan of the two tapes, each of these problems belongs to $\text{ST}(O(\lg N), O(1), 2)$.

Similarly as in the case with just one read/write-stream, although there is a priori no restriction on the running time or the size of the read/write-streams, it is not difficult to show that these parameters cannot get too large during an (r, s, t) -bounded computation:

PROPOSITION 4.10 ([18]). *Let $r, s : \mathbb{N} \rightarrow \mathbb{N}$, let $t \in \mathbb{N}$, and let M be an (r, s, t) -bounded (deterministic, randomized, or nondeterministic) Turing machine. Then, the length of every run of M on an input of length N , and the space used on each of M 's external memory tapes during such a run, are*

$$\leq N \cdot 2^{O(r(N) \cdot (s(N) + t))}.$$

²This is of no practical use, since the resulting algorithm produces *huge* intermediate results, but it is of major theoretical interest, because it implies that the lower bounds proved in Section 4.2.2 are tight.

4.2.2 Lower Bounds

To prove lower bounds for algorithms on $t \geq 2$ read/write-streams, *communication complexity* based arguments as used in Section 3 and Section 4.1 utterly fail. The reason is that we can easily communicate arbitrarily many bits from one part of the input to any other part by just copying the first part to a second external memory tape and then reading it in parallel with the second part of the input. This requires no internal memory and just two head reversals on the external memory tapes.

But still, although the use of several read/write-streams allows to copy large consecutive segments of the input from one place to another, there seems no easy way of significantly *permuting* the input without using too many head reversals. This intuition was confirmed in [22, 19]. A key lemma of [19] reduces the problem of proving lower bounds for algorithms on read/write-streams to a purely *combinatorial* problem. All currently known lower bound proofs for algorithms operating on more than one read/write-stream rely on this reduction. Before giving the precise statement of this reduction in Lemma 4.13 below, we need to introduce some further notation.

DEFINITION 4.11 (SORTEDNESS).

- (a) A sequence $(s_1, \dots, s_\lambda)$ is a subsequence of a sequence $(s'_1, \dots, s'_{\lambda'})$, if there exist indices $j_1 < \dots < j_\lambda$ such that $s_1 = s'_{j_1}, s_2 = s'_{j_2}, \dots, s_\lambda = s'_{j_\lambda}$.
- (b) Let $m \in \mathbb{N}$ and let φ be a permutation of $\{1, \dots, m\}$. We define *sortedness*(φ) to be the length of the longest subsequence of $(\varphi(1), \dots, \varphi(m))$ that is sorted in either ascending or descending order (i.e., that is a subsequence of $(1, \dots, m)$ or of $(m, \dots, 1)$).

EXAMPLES 4.12. We consider permutations of $\{1, \dots, m\}$.

- (a) The *identity* on $[1, m]$ as well as the permutation ψ_m with $\psi_m(i) := m - i$, for all $i \in [1, m]$, both have sortedness m .
- (b) The particular permutation χ_m for which $\chi_m(1), \dots, \chi_m(m)$ are the numbers $1, \dots, m$, sorted lexicographically by their reverse binary representation, has *sortedness*(χ_m) $\leq 2\sqrt{m}$.
- (c) If $m = \ell^2$ is a square number, then the permutation φ_m with $\varphi_m((i-1) \cdot \ell + j) = (\ell - j) \cdot \ell + i$, for all $i, j \in \{1, \dots, \ell\}$, has *sortedness*(φ_m) $= \ell = \sqrt{m}$.

This is the smallest possible sortedness, since the well-known Erdős-Szekeres Theorem [14] implies that the sortedness of every permutation of $\{1, \dots, m\}$ is at least $\lfloor \sqrt{m} \rfloor$.

As further notation, we use π to denote the ordinary projection operator from relational algebra. I.e., if $\bar{v} = (v_1, \dots, v_k)$ is a k -tuple and $J = \{j_1, \dots, j_\ell\}$ is a set of indices with $1 \leq j_1 < \dots < j_\ell \leq k$, then $\pi_J(\bar{v}) = (v_{j_1}, \dots, v_{j_\ell})$. If $\mathcal{J}$ is a set of k -tuples, then $\pi_J(\mathcal{J}) = \{\pi_J(\bar{v}) : \bar{v} \in \mathcal{J}\}$.

The following Lemma 4.13 reduces the problem of proving lower bounds for read/write-streams to a purely combinatorial problem. It uses the notion of an (i, j) -rectangle, which is defined as follows: Let $1 \leq i < j \leq k$ and let $\mathcal{J}$ be a set of k -tuples. Then $\mathcal{J}$ is called (i, j) -rectangle iff

1. all tuples $\bar{u}$ and $\bar{v}$ in $\mathcal{J}$ satisfy

$$\pi_{[1, k] \setminus \{i, j\}}(\bar{u}) = \pi_{[1, k] \setminus \{i, j\}}(\bar{v}),$$

i.e., they coincide in all components except for, potentially, i and j , and

2. there exist sets X and Y such that $\pi_{i,j}(\mathcal{J}) = X \times Y$.

I.e., if $\bar{u} = (u_1, \dots, u_k)$ and $\bar{v} = (v_1, \dots, v_k)$ are tuples in $\mathcal{J}$, then also the tuple $\bar{u}'$ obtained from $\bar{u}$ by replacing u_j with v_j belongs to $\mathcal{J}$.

LEMMA 4.13 (IMPLICIT IN [19]). *Let $r, s : \mathbb{N} \rightarrow \mathbb{N}$, let $t \in \mathbb{N}$, and let M be an (r, s, t) -bounded randomized Turing machine. Let Σ be a finite alphabet such that M works on input strings from $(\Sigma \cup \{\#\})^*$ (where $\# \notin \Sigma$).*

Then there exists a number $\lambda \in \mathbb{N}$ such that the following is true for all $m, n \in \mathbb{N}$: Letting $I := \Sigma^n$ and $N := 2m \cdot (n+1)$, there exist

- *a finite set³ C whose elements are called “random choices”*
- *a finite set S of size $|S| \leq 2^{2m \cdot (r(N) \cdot s(N) + \lg N) \cdot 2^{\lambda \cdot r(N)}}$ whose elements are called “skeletons”*
- *a set $S_{acc} \subseteq S$ whose elements are called “accepting skeletons”*
- *a function $\sigma : I^{2m} \times C \rightarrow S$*

such that the following two statements are true:

1. *for every $\bar{v} = (v_1, \dots, v_{2m}) \in I^{2m}$ and the string $\bar{v} := v_1 \# \dots \# v_{2m} \#$,*

$$\Pr(M \text{ accepts } \bar{v}) = \frac{|\{c \in C : \sigma(\bar{v}, c) \in S_{acc}\}|}{|C|}$$

2. *for every permutation ϕ of $\{1, \dots, m\}$ and for every $\zeta \in S$ there exists a set $J \subseteq \{1, \dots, m\}$ of size*

$$|J| \geq m - \text{sortedness}(\phi) \cdot t^{2 \cdot r(N)}$$

such that for every $c \in C$, every $i \in J$, and all $\bar{a} \in I^{2m-2}$ the set

$$\mathcal{J} := \left\{ \bar{v} \in I^{2m} : \sigma(\bar{v}, c) = \zeta \text{ and } \bar{a} = \pi_{[1, 2m] \setminus \{i, m+\phi(i)\}}(\bar{v}) \right\}$$

is an $(i, m+\phi(i))$ -rectangle.

This lemma follows from a long series of technical lemmas in [19] which, in particular, use an intermediate machine model called *list machines* and a notion of *skeletons* of runs of such machines. The proof proceeds in two overall steps by (1) showing a “simulation lemma” which states that (r, s, t) -bounded Turing machines can be simulated by list machines, and by (2) carefully analyzing the possible flows of information during a list machine’s computation. Step (1) can be achieved because list machines are non-uniform and have a large number of tape symbols and states. In this sense, they are much stronger than Turing machines. Step (2) can be achieved because list machines can only compare and move around input strings as a whole and in this sense are much weaker than Turing machines. We exploit this weakness in our proof that (appropriately restricted) list machines cannot *significantly* permute the relative order of its input strings (the essence of this statement is reflected in item 2 of Lemma 4.13).⁴

Note that Lemma 4.13 enables us to prove lower bounds for (r, s, t) -bounded randomized algorithms on read/write-streams in a *purely combinatorial* way. This may still be a bit tricky — but at least it has the advantage that we can now prove lower bounds without having to refer to particular Turing machines (or algorithms).

Let us next look at an example of how to use Lemma 4.13 for proving a lower bound.

³The size of C won’t be important for our proofs, but for completeness let me mention that $|C| = 2^{2m \cdot 2^{O(r(N) \cdot s(N) + \lg N)}}$, and if T is deterministic, then we even have $|C| = 1$.

⁴Since Lemma 4.13 does not explicitly occur in [19], let me indicate which lemmas from the full version of [19] need to be combined to immediately obtain a proof: Lemma 16, Lemma 31(a), Lemma 32, Lemma 38, and Lemma 34.

THEOREM 4.14 ([18]). *For any ε with $0 < \varepsilon < 1$,*

$$\text{Multiset-Equality} \notin \text{RST}(o(\lg N), N^{1-\varepsilon}, O(1)).$$

PROOF. For contradiction let us assume that there is an ε with $0 < \varepsilon < 1$ such that *Multiset-Equality* $\in \text{RST}(o(\lg N), N^{1-\varepsilon}, O(1))$. Then, there are $r, s : \mathbb{N} \rightarrow \mathbb{N}$ with $r(N) = o(\lg N)$, $s(N) \leq N^{1-\varepsilon}$, a $t \in \mathbb{N}$, an (r, s, t) -bounded RTM M , and a number δ with $0 \leq \delta < 1/2$ such that for each input instance $\tilde{w}$ of the *Multiset-Equality* problem we have

- $\Pr(M \text{ accepts } \tilde{w}) \geq 1 - \delta \geq \frac{1}{2}$, if $\tilde{w}$ is a “yes”-instance
- $\Pr(M \text{ accepts } \tilde{w}) = 0$, if $\tilde{w}$ is a “no”-instance.

Let $\lambda \in \mathbb{N}$ be the constant obtained from Lemma 4.13.

Step 1: We first choose suitable numbers m and n for which, further on, the statement of Lemma 4.13 will be used. Right now, the choice of these numbers will look somewhat arbitrary, but later in the proof we will see that the chosen numbers have just the right properties needed for finishing the proof of Theorem 4.14. We fix

$$d := \lceil 5/\varepsilon \rceil \quad (1)$$

i.e., d is the smallest natural number such that $d \cdot \varepsilon \geq 5$. Next, we choose m to be a power of 2 that meets the following requirements:

$$m - 2 \cdot m^{\frac{3}{4}} \geq 1 \quad (2)$$

$$t^{2 \cdot r(m^d)} \leq m^{\frac{1}{4}} \quad (3)$$

$$2m \cdot (r(m^d) \cdot s(m^d) + \lg(m^d)) \cdot 2^{\lambda \cdot r(m^d)} \leq m^{d-3} \quad (4)$$

By using that $r(m^d) = o(\lg(m^d))$ (for (3) and (4)) and $s(m^d) \leq m^{d \cdot (1-\varepsilon)} \leq m^{d-5}$ (for (4)) it is straightforward to see that these requirements are met by all sufficiently large m . Further, we choose

$$n := \frac{m^{d-1}}{2} - 1 \quad (5)$$

and note that n is a natural number (since m is even) with

$$n \geq m^{d-2} \quad \text{and} \quad N := 2m \cdot (n+1) = m^d \quad (6)$$

From now on, we will restrict attention to these numbers m, n, N .

Step 2: Let $I := \{0, 1\}^n$ and let C, S, S_{acc}, σ be chosen according to Lemma 4.13. Note for later use that, due to equation (4),

$$\lg(|S|) \leq m^{d-3}. \quad (7)$$

Let us fix a permutation ϕ on $\{1, \dots, m\}$ with $\text{sortedness}(\phi) \leq 2\sqrt{m}$ (from Example 4.12 we know that such permutations exist). We note for later use that, due to equations (2) and (3),

$$m - \text{sortedness}(\phi) \cdot t^{2 \cdot r(N)} \geq 1. \quad (8)$$

We identify I with the set $\{0, \dots, 2^n - 1\}$ of natural numbers. We divide this set into m consecutive intervals of equal length and, for every $i \in [1, m]$ we let I_i denote the i -th such interval. From now on we will restrict attention to tuples $\bar{v} \in I^{2m}$ from the set

$$\mathcal{J}_{eq} := \left\{ (v_1, \dots, v_m, v'_1, \dots, v'_m) : (v_1, \dots, v_m) = (v'_{\phi(1)}, \dots, v'_{\phi(m)}) \in I_{\phi(1)} \times \dots \times I_{\phi(m)} \right\}$$

Clearly,

$$|\mathcal{J}_{eq}| = (2^n/m)^m = 2^{m \cdot n - m \cdot \lg(m)}. \quad (9)$$

Of course, for each $\bar{v} = (v_1, \dots, v_m, v'_1, \dots, v'_m) \in \mathcal{J}_{eq}$, the corresponding string $\bar{v} = v_1 \# \dots v_m \# v'_1 \# \dots v'_m \#$ is a “yes”-instance of the *Multiset-Equality* problem, and thus $\Pr(M \text{ accepts } \bar{v}) \geq 1/2$. Using this, we obtain from item 1 of Lemma 4.13 that, for each $\bar{v} \in \mathcal{J}_{eq}$,

$$\frac{|\{c \in C : \sigma(\bar{v}, c) \in S_{acc}\}|}{|C|} \geq \frac{1}{2}.$$

Now, a straightforward double counting argument shows that

$$\begin{aligned} & \sum_{c \in C} |\{\bar{v} \in \mathcal{J}_{eq} : \sigma(\bar{v}, c) \in S_{acc}\}| \\ &= |\{(\bar{v}, c) \in \mathcal{J}_{eq} \times C : \sigma(\bar{v}, c) \in S_{acc}\}| \\ &= \sum_{\bar{v} \in \mathcal{J}_{eq}} |\{c \in C : \sigma(\bar{v}, c) \in S_{acc}\}| \geq |\mathcal{J}_{eq}| \cdot \frac{1}{2} \cdot |C|. \end{aligned}$$

Thus there must exist a particular $c \in C$ such that

$$|\{\bar{v} \in \mathcal{J}_{eq} : \sigma(\bar{v}, c) \in S_{acc}\}| \geq |\mathcal{J}_{eq}| \cdot \frac{1}{2} \stackrel{(9)}{\geq} 2^{m \cdot n - m \cdot \lg(m) - 1}.$$

From now on, we restrict attention to this particular c and consider the set $\mathcal{J}_{eq,c} := \{\bar{v} \in \mathcal{J}_{eq} : \sigma(\bar{v}, c) \in S_{acc}\}$.

Of course, there must be a particular $\zeta \in S$ (more precisely, $\zeta \in S_{acc}$) such that $\sigma(\bar{v}, c) = \zeta$ for at least $\frac{|\mathcal{J}_{eq,c}|}{|S|}$ different elements $\bar{v}$ from $\mathcal{J}_{eq,c}$. From now on, we restrict attention to this particular ζ , consider the set $\mathcal{J}_{eq,c,\zeta} := \{\bar{v} \in \mathcal{J}_{eq,c} : \sigma(\bar{v}, c) = \zeta\}$, and note that

$$|\mathcal{J}_{eq,c,\zeta}| \geq \frac{|\mathcal{J}_{eq,c}|}{|S|} \geq 2^{m \cdot n - m \cdot \lg(m) - 1 - \lg(|S|)}. \quad (10)$$

Let J be the subset of $\{1, \dots, m\}$ obtained from Lemma 4.13 with

$$|J| \geq m - \text{sortedness}(\varphi) \cdot t^{2 \cdot r(N)} \stackrel{(8)}{\geq} 1.$$

I.e., there exists at least one element i in J . For a fixed such i let us next choose a tuple $\bar{a} \in I^{2m-2}$ such that the set

$$\mathcal{J}_{eq,c,\zeta,i,\bar{a}} := \left\{ \bar{v} \in \mathcal{J}_{eq,c,\zeta} : \bar{a} = \pi_{[1,2m] \setminus \{i, m+\varphi(i)\}}(\bar{v}) \right\}$$

is as large as possible. It is straightforward to see that

$$|\mathcal{J}_{eq,c,\zeta,i,\bar{a}}| \geq \frac{|\mathcal{J}_{eq,c,\zeta}|}{|\pi_{[1,2m] \setminus \{i, m+\varphi(i)\}}(\mathcal{J}_{eq,c,\zeta})|}. \quad (11)$$

Furthermore, from the choice of $\mathcal{J}_{eq}$ it is clear that

$$\begin{aligned} |\pi_{[1,2m] \setminus \{i, m+\varphi(i)\}}(\mathcal{J}_{eq,c,\zeta})| &\leq \\ |\pi_{[1,2m] \setminus \{i, m+\varphi(i)\}}(\mathcal{J}_{eq})| &\leq \left(\frac{2^n}{m}\right)^{m-1} = 2^{m \cdot n - n \cdot (m-1) \cdot \lg(m)} \end{aligned}$$

Using this and (10), we obtain from (11) that

$$|\mathcal{J}_{eq,c,\zeta,i,\bar{a}}| \geq 2^{n - \lg(m) - 1 - \lg(|S|)}.$$

Since

$$n - \lg(m) - 1 - \lg(|S|) \stackrel{(6),(7)}{\geq} m^{d-2} - \lg(m) - 1 - m^{d-3} \geq 1,$$

we know that $|\mathcal{J}_{eq,c,\zeta,i,\bar{a}}| \geq 2$. Thus there exist two different tuples $\bar{u} = (u_1, \dots, u_{2m})$ and $\bar{v} = (v_1, \dots, v_{2m})$ in $\mathcal{J}_{eq,c,\zeta,i,\bar{a}}$.

Since, obviously, $\mathcal{J}_{eq,c,\zeta,i,\bar{a}}$ is a subset of the set $\mathcal{J}$ from item 2 of Lemma 4.13, $\bar{u}$ and $\bar{v}$ are two different elements in the set $\mathcal{J}$ which, due to item 2 of Lemma 4.13, is an $(i, m+\varphi(i))$ -rectangle. Thus, also the tuple $\bar{w} = (w_1, \dots, w_{2m})$ obtained from $\bar{u}$ by replacing $u_{m+\varphi(i)}$ with $v_{m+\varphi(i)}$ belongs to $\mathcal{J}$. In particular, this tells us that $\sigma(\bar{w}, c) = \zeta \in S_{acc}$. Hence, due to item 1 of Lemma 4.13 we know that $\Pr(M \text{ accepts } \bar{w}) > 0$.

However, $w_i \neq w_{m+\varphi(i)}$ (since $\bar{u} \neq \bar{v}$), and thus $\bar{w}$ is a “no”-instance for *Multiset-Equality*. This contradicts the assumption on M that $\Pr(M \text{ accepts } \bar{w}) = 0$ for “no”-instances $\bar{w}$. Finally, the proof of Theorem 4.14 is complete. $\square$

With essentially the same proof, one also obtains the following results, where *Check* $_{\varphi,\varepsilon}$ is the following problem where, for each m , φ_m denotes the permutation of sortedness $\sqrt{m}$ from Example 4.12(c).

Check $_{\varphi,\varepsilon}$

Instance: $v_1 \# \dots v_m \# v'_1 \# \dots v'_m \#$,

where $m \geq 1$ is an even power of 2
(i.e. $m = 2^{2q}$ for some $q \in \mathbb{N}$), and

$(v_1, \dots, v_m, v'_1, \dots, v'_m) \in I_{\varphi_m(1)} \times \dots \times I_{\varphi_m(m)} \times I_1 \times \dots \times I_m$

Here, the sets $I_1, \dots, I_m$ are obtained as the partition of the set $I := \{0, 1\}^n$ into m consecutive subsets, each of size $2^n/m$, where $n := \frac{m^{d-1}}{2} - 1$ with $d := \lceil 5/\varepsilon \rceil$.

Problem: Decide if $(v_1, \dots, v_m) = (v'_{\varphi_m(1)}, \dots, v'_{\varphi_m(m)})$.

COROLLARY 4.15 ([18]).

(a) For any ε with $0 < \varepsilon < 1$,

- *Check* $_{\varphi,\varepsilon}$, *Check-Sort*, (*Multi*)*Set-Equality*
 $\notin \text{RST}(o(\lg N), N^{1-\varepsilon}, O(1))$,
- *Sorting* $\notin \text{LasVegas-RST}(o(\lg N), N^{1-\varepsilon}, O(1))$.

(b) Let $r, s: \mathbb{N} \rightarrow \mathbb{N}$ such that $s(N) = o(N)$ and $r(N) = o(\lg \frac{N}{s(N)})$. Then, none of the problems *Multiset-Equality*, *Set-Equality*, *Check-Sort* belongs to $\text{RST}(r(N), s(N), O(1))$, and the problem *Sorting* does not belong to $\text{LasVegas-RST}(r(N), s(N), O(1))$.

Note that the lower bounds from Corollary 4.15(a) precisely match (in terms of the number of head reversals) the upper bounds from Example 4.9(b) which tell us that each of the considered problems belongs to $\text{ST}(O(\lg N), O(1), O(1))$. Also the lower bound from Corollary 4.15(b) can be matched by an upper bound telling us that all the considered problems belong to $\text{ST}(r(N), s(N), O(1))$ if $r(N) = \Omega(\lg \frac{N}{s(N)})$ (see [18]).

The lower bound for *Check* $_{\varphi,\varepsilon}$, in particular, tells us that also the following problem is difficult for algorithms on read/write-streams:

Arrange $_{\varphi,\varepsilon}$

Instance: $v_1 \# \dots v_m \#$,

where $m \geq 0$ is an even power of 2, and $v_1, \dots, v_m \in \{0, 1\}^n$
where $n := \frac{m^{d-1}}{2} - 1$ with $d := \lceil 5/\varepsilon \rceil$.

Output: $v_{\varphi_m(1)} \# \dots v_{\varphi_m(m)} \#$.

I.e., the task in problem *Arrange* $_{\varphi,\varepsilon}$ is to rearrange the input strings according to the fixed permutation φ_m . Of course, a read/write-streams algorithm solving this problem could be used as a subroutine for solving the problem *Check* $_{\varphi,\varepsilon}$ by

1. rearranging the sequence $v'_1 \# \dots v'_m \#$ into the sequence $v'_{\varphi_m(1)} \# \dots v'_{\varphi_m(m)} \#$, and
2. reading and comparing $v_1 \# \dots v_m \#$ and $v'_{\varphi_m(1)} \# \dots v'_{\varphi_m(m)} \#$ by a simultaneous scan of two streams.

We thus obtain

COROLLARY 4.16. *For any ε with $0 < \varepsilon < 1$,*

$$\text{Arrange}_{\varphi, \varepsilon} \notin \text{LasVegas-RST}(o(\lg N), N^{1-\varepsilon}, O(1)).$$

Let us have a quick look at the intuitive meaning of this. Consider the case where $\varepsilon \leq 1/2$. Then, $d \geq 10$ and the size of internal memory is $N^{1-\varepsilon} \geq \sqrt{N} = m^{d/2} \geq m^5$. Thus, with a single pass over the input, the binary representation of m can easily be computed and stored in internal memory. Further, a look at the particular permutation φ_m (see Example 4.12(c)) shows that, given m , also the list of (suitable representations of) $\varphi_m(1), \dots, \varphi_m(m)$ can easily be computed and stored in internal memory. Thus, the algorithm perfectly knows which input string has to be moved onto which position of the output stream. — But Corollary 4.16 tells us that physically moving the data cannot be achieved by a read/write-stream algorithm with $o(\lg N)$ scans, even if internal memory of size up to $N^{1-\varepsilon}$ and an arbitrary (constant) number of streams are available.

In light of the fact that $\text{Arrange}_{\varphi, \varepsilon}$ deals with the problem of rearranging a small number (m) of data items of huge length ($n = \theta(m^{d-1})$), one might suspect that the reason for the lower bounds is mainly the *length* of the data items — and one might wonder if the considered problems get easier when restricting the inputs to a more realistic scenario where the total number m of data items is considerably larger than the size n of each single input item. This is, however, not the case: We can prove that the problems *Sorting*, *Check-Sort*, and *(Multi)Set-Equality* remain hard even if restricted to inputs where the length n of the strings v_i, v'_i is logarithmically bounded in m such that $|v_i|, |v'_i| \leq 2 \cdot \lg m$, for all $i \in \{1, \dots, m\}$. Formally, we use *Short-Sorting* (resp. *Short-Check-Sort*, *Short-Set-Equality*, and *Short-Multiset-Equality*) to denote the restrictions of the respective problems to inputs consisting of such “short” data items. A simple reduction from the lower bounds for $\text{Arrange}_{\varphi, \varepsilon}$ and $\text{Check}_{\varphi, \varepsilon}$ leads to the following:

THEOREM 4.17 ([22, 18]). *For any ε with $0 < \varepsilon < 1$,*

- *Short-Sorting* $\notin \text{LasVegas-RST}(o(\lg N), N^{1-\varepsilon}, O(1))$.
- *Short-Check-Sort*, *Short-Multiset-Equality*, *Short-Set-Equality* $\notin \text{RST}(o(\lg N), N^{1-\varepsilon}, O(1))$.

Recall from Example 4.9(a) that

$$\begin{aligned} \text{Multiset-Equality} &\in \text{co-RST}(O(1), O(\lg N), 1) \\ &\subseteq \text{BPST}(O(1), O(\lg N), 1), \end{aligned}$$

i.e., the *Multiset-Equality* problem can easily be solved by a randomized algorithm on read/write-streams with 2-sided bounded error. Recently, Beame, Jayram, and Rudra [8] obtained a strong *lower bound* for randomized algorithms on read/write-streams with 2-sided bounded error, resolving the main open problem of [19].

THEOREM 4.18 ([8]). *For any ε with $0 < \varepsilon < 1$,*

$$\text{Set-Disjointness} \notin \text{BPST}\left(o\left(\frac{\lg N}{\lg \lg N}\right), N^{1-\varepsilon}, O(1)\right).$$

The proof is based on Lemma 4.13 and uses an additional sophisticated combinatorial argument (much more elaborate than the argument used in the above proof of Theorem 4.14). Note that the lower bound of Theorem 4.18 almost matches (in terms of the number of head reversals) the upper bound from Example 4.9(b) which tells us that $\text{Set-Disjointness} \in \text{ST}(O(\lg N), O(1), O(1))$.

4.2.3 Consequences for Query Processing

Let us first consider the data complexity of *relational algebra* queries. By using Example 4.9(b) (for the upper bound) and Corollary 4.15 and Theorem 4.18 (for the lower bounds), one easily obtains the following

THEOREM 4.19 ([19, 8]).

- (a) *For every relational algebra query Q , the problem of evaluating Q on a stream consisting of the tuples of the input database relations, can be solved by an $(O(\lg N), O(1), O(1))$ -bounded deterministic algorithm on read/write-streams.*
- (b) *There exists a relational algebra query Q' such that the problem of evaluating Q' on a stream of the tuples of the input database relations cannot be solved by any $(o(\lg N), N^{1-\varepsilon}, O(1))$ -bounded Las Vegas-algorithm on read/write-streams.*
- (c) *The task of checking whether the join of two relations A and B is empty cannot be performed by any $(o(\frac{\lg N}{\lg \lg N}), N^{1-\varepsilon}, O(1))$ -bounded randomized algorithm on read/write-streams with a two-sided bounded error of at most $\delta < 1/2$.*

Similarly as in Section 4.1.3, it is now also straightforward to conclude lower bounds for evaluating queries against XML data:

THEOREM 4.20 ([19, 8]).

- (a) *There is an XQuery query Q_1 such that the problem of evaluating Q_1 on an input XML document of length N cannot be solved by any $(o(\lg N), N^{1-\varepsilon}, O(1))$ -bounded Las Vegas-algorithm on read/write-streams.*
- (b) *There is an XPath query Q_2 such that the problem of checking, for an input XML document D of length N , whether $Q_2(D) \neq \emptyset$, cannot be solved by any $(o(\lg N), N^{1-\varepsilon}, O(1))$ -bounded randomized algorithm on read/write-streams with 1-sided bounded error that accepts “yes”-instances with probability 1 and that rejects “no”-instances with probability $\geq 1 - \delta$ for a $\delta < 1/2$.*
- (c) *There is an XQuery query Q_3 such that the problem of checking whether the result of Q_3 on an input XML document of length N is empty cannot be solved by any $(o(\frac{\lg N}{\lg \lg N}), N^{1-\varepsilon}, O(1))$ -bounded randomized algorithm on read/write-streams with a two-sided bounded error of at most $\delta < 1/2$.*

4.2.4 A Complexity Theoretic Point of View

In [24], the relations between “classical” complexity classes and classes based on the computation model of read/write-streams were investigated, and it turned out that there is a close correspondence between nondeterministic algorithms on read/write streams and nondeterministic time-bounded complexity classes, as well as between deterministic algorithms on read/write streams with $\Omega(\lg N)$ scans and space-bounded complexity classes:

THEOREM 4.21 ([24]).

- (a) $\text{NP} = \text{NST}(3, O(\lg N), 2) = \text{NST}(O(1), O(\lg N), O(1))$
 $= \text{NST}(O(\lg N), O(1), O(1)).$
- (b) $\text{ST}(O(1), O(\lg N), O(1)) \subsetneq$
 $\text{LOGSPACE} \subseteq \text{ST}(O(\lg N), O(1), O(1)) \subseteq$
 $\text{DSPACE}(O((\lg N)^2)).$

(c) In general, for all functions r and s with $r(N) \cdot s(N) = \Omega(\lg N)$,

$$\text{ST}(r, s, O(1)) \subseteq \text{DSpace}(r^2 \cdot s) \subseteq \text{ST}(O(r^2 \cdot s), O(1), O(1)).$$

(The latter inclusion is a consequence of a result by Chen and Yap [10].)

This close correspondence to classical complexity classes indicates that it can be expected to be rather difficult to prove lower bounds for particular problems in connection with

- nondeterministic algorithms on read/write-streams or
- deterministic (or randomized) algorithms on read/write-streams with $\Omega(\lg N)$ scans,

because showing such a lower bound for a particular problem would imply that this problem does not belong to NP, respectively, LOGSPACE.

On the other hand, the classical *space hierarchy theorem* (cf., e.g. the textbooks [31, 5]) implies that there is a strict hierarchy of the ST-classes based on the number of scans (as long as this number is of size $\Omega(\lg N)$) so that, for example, for each $k \in \mathbb{N}$, we obtain that

$$\text{ST}(O((\lg N)^k), O(\lg N), O(1)) \subsetneq \text{ST}(O((\lg N)^{2k+2}), O(1), O(1)).$$

In [24], a similar hierarchy is also obtained for the case where only $o(\lg N)$ scans are available:

THEOREM 4.22 ([24]). *For every $k \geq 2$,*

$$\begin{aligned} \text{ST}(O(\sqrt[k+1]{\lg N}), O(\lg N), O(1)) &\subsetneq \\ &\text{ST}(O(\sqrt[k]{\lg N}), O(\lg N), O(1)), \text{ and} \\ \text{RST}(O(\sqrt[k+1]{\lg N}), O(\lg N), O(1)) &\subsetneq \\ &\text{RST}(O(\sqrt[k]{\lg N}), O(\lg N), O(1)). \end{aligned}$$

The proof is by considering suitably padded versions of the *Multiset-Equality* problem in connection with the lower bound from Theorem 4.14.

From Theorem 3.1, Theorem 4.14, and Theorem 4.21(a) one immediately obtains the following separation between the deterministic, the randomized (with 1-sided bounded error), the nondeterministic, and the randomized (with 2-sided bounded error) computation models for read/write-streams.

COROLLARY 4.23 ([19]). *For every ε with $0 < \varepsilon < 1$ and all $r, s : \mathbb{N} \rightarrow \mathbb{N}$ with $r(N) = o(\lg N)$ and $s(N) \in O(N^{1-\varepsilon}) \cap \Omega(\lg N)$ we have*

- (a) $\text{ST}(O(r), O(s), O(1)) \subsetneq \text{RST}(O(r), O(s), O(1)) \subsetneq \text{NST}(O(r), O(s), O(1)).$
- (b) $\text{RST}(O(r), O(s), O(1)) \neq \text{co-RST}(O(r), O(s), O(1)).$
- (c) $\text{RST}(O(r), O(s), O(1)), \text{co-RST}(O(r), O(s), O(1)) \subsetneq \text{BPST}(O(r), O(s), O(1)).$

We conclude the section on read/write-stream algorithms by noting that there is a trade-off between the size of internal memory and the number of scans of the read/write-streams, stating that internal memory can be compressed from size $s(N)$ to $O(1)$ at the expense of adding an extra factor $s(N)$ to the number of scans of the read/write-streams:

THEOREM 4.24 ([24]). *For all $t \in \mathbb{N}$ and all functions r, s with $r(N) \cdot s(N) = \Omega(\lg N)$ we have*

$$(a) \text{ST}(r, s, t) \subseteq \text{ST}(O(r \cdot s), O(1), t+2).$$

$$(b) \text{NST}(r, s, t) \subseteq \text{NST}(O(r \cdot s), O(1), t+2).$$

- (c) *For every (r, s, t) -bounded randomized algorithm A on read/write-streams there is an $(O(r \cdot s), O(1), t+2)$ -bounded randomized algorithm B on read/write-streams which has the same acceptance probabilities as A , i.e., for every input instance w , $\Pr(B \text{ accepts } w) = \Pr(A \text{ accepts } w)$.*

5. OTHER MACHINE MODELS

Finite Cursor Machines. Finite cursor machines (FCMs, for short) were introduced in [17] as an abstract model of database query processing. Formally, they are defined in the framework of *abstract state machines* (as opposed to the Turing machine based model of Section 4). 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.

FCMs model quite faithfully what happens in database query processing. For example, every *semijoin algebra*⁵ query can be computed in a straightforward way by a query plan composed of FCMs and sorting operations. And it can be shown that intermediate sorting operations sometimes are inevitable. In fact, even if the machine gets *sorted* input tables, already the composition of two semijoins is not computable by an FCM with internal memory of size $o(n)$ (where n denotes the total number of tuples in the input database). Details can be found in [17].

The StrSort Model. A related computation model based on read/write-streams and intermediate sorting steps is the *StrSort* model of [2] that was further considered in [32].

In [12], the *W-Stream* model, a restriction of the StrSort model in which intermediate sorting steps are prohibited, was introduced. This model can also be viewed as a restriction of the present paper’s computation model with one read/write-stream (see Section 4.1).

The Parallel Disk Model. Finally, the *Parallel Disk Model* (see [35, 36, 27] and the references therein) is the probably most widely used model for designing and analyzing efficient external memory algorithms. In this model, external memory can be accessed in units of *blocks* (each containing a fixed number of data items), and the performance of algorithms is measured in terms of the total number of disk accesses (for reading as well as for writing). Note, however, that there is no distinction between sequential access and random access to external memory in this model.

⁵The *semijoin algebra* is the restriction of relational algebra where, instead of joins, only semijoins are allowed.

6. SOME OPEN QUESTIONS

To conclude this paper let me point out some directions for future research.

Concerning the computation model of read/write-streams, the following two tasks should be attacked: 1. Develop methods for proving lower bounds in the presence of $\Omega(\lg N)$ head reversals. (Recall, however, from Section 4.2.4 that this can be expected to be rather difficult, since showing a lower bound for a problem in the presence of $\Omega(\lg N)$ head reversals on several read/write-streams implies that this problem does not belong to the complexity class LOGSPACE.) 2. It would certainly be interesting to study the related model with multiple read/write-streams and intermediate sorting steps. First steps in this direction were already taken in [2, 32].

Concerning the finite cursor machine model, it would be nice to settle the main open question from [17]: Is there a Boolean relational algebra query that cannot be computed by a composition of FCMs and sorting operations? (The conjectured answer is “yes” since this can in fact be proved modulo a plausible complexity theoretic assumption.)

Concerning the Parallel Disk Model, a number of lower bound results are known, among them lower bounds for the *Sorting* problem (see [36]). But all these lower bounds rely on the assumption that the input data items (e.g., the strings that are to be sorted) are *indivisible* and that at any point in time, the external memory consists, in some sense, of a permutation of the input items. As pointed out already in [35, 36], it is a challenging future task to develop methods for proving lower bounds for the Parallel Disk Model without relying on such an indivisibility assumption.⁶

Finally, turning to the area of complexity theory, it would certainly be nice to settle the long standing open question whether the *Sorting* problem can be solved by a linear-time multi-tape Turing machine.

Acknowledgments. I would like to thank Martin Grohe, André Hernich, Christoph Koch, and Leonid Libkin for helpful comments on an earlier version of this paper.

7. REFERENCES

- [1] J. Abello and J. Vitter, editors. *External Memory Algorithms*, volume 50. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1999.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*. Springer-Verlag, 2nd edition, 1995.
- [6] Z. Bar-Yossef, M. Fontoura, and V. Josifovski. Buffering in query evaluation over XML streams. In *Proc. PODS'05*, pages 216–227.
- [7] 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.
- [8] P. Beame, T. Jayram, and A. Rudra. Lower bounds for randomized read/write stream algorithms. In *Proc. STOC'07*, 2007.
- [9] C. Y. Chan, P. Felber, M. Garofalakis, and R. Rastogi. Efficient filtering of XML documents with XPath expressions. *VLDB J.*, 11(4):354–379, 2002.
- [10] J. Chen and C.-K. Yap. Reversal complexity. *SIAM Journal on Computing*, 20(4):622–638, 1991.
- [11] A. Czumaj and C. Sohler. Sublinear-time algorithms. *Bulletin of the EATCS*, 89:23–47, 2006.
- [12] C. Demetrescu, I. Finocchi, and A. Ribichini. Trading off space for passes in graph streaming problems. In *Proc. SODA'06*, pages 714–723, 2006.
- [13] Y. Diao, M. Altinel, M. Franklin, H. Zhang, and P. Fischer. Path sharing and predicate evaluation for high-performance XML filtering. *ACM TODS*, 28(4):467–516, 2003.
- [14] P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.
- [15] G. Gottlob, C. Koch, and R. Pichler. The complexity of XPath query evaluation. In *Proc. PODS'03*, pages 179–190, 2003.
- [16] T. Green, A. Gupta, G. Miklau, M. Onizuka, and D. Suciu. Processing XML streams with deterministic automata and stream indexes. *ACM TODS*, 29(4):752–788, 2004.
- [17] M. Grohe, Y. Gurevich, D. Leinders, N. Schweikardt, J. Tyszkiewicz, and J. Van den Bussche. Database query processing using finite cursor machines. In *Proc. ICDT'07*, volume 4353 of *Springer LNCS*, pages 284–298, 2007.
- [18] M. Grohe, A. Hernich, and N. Schweikardt. Lower bounds for processing data with few random accesses to external memory. Journal version of [22] and [19], submitted in 2006.
- [19] 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.
- [20] 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.
- [21] M. Grohe, C. Koch, and N. Schweikardt. The complexity of querying external memory and streaming data. In *Proc. FCT'05*, volume 3623 of *Springer LNCS*, pages 1–16, 2005.
- [22] M. Grohe and N. Schweikardt. Lower bounds for sorting with few random accesses to external memory. In *Proc. PODS'05*, pages 238–249, 2005.
- [23] 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.
- [24] A. Hernich and N. Schweikardt. Reversal complexity revisited. CoRR Report, arXiv:cs.CC/0608036, August 2006.
- [25] C. Koch. Efficient processing of expressive node-selecting queries on XML data in secondary storage: A tree automata-based approach. In *Proc. VLDB'03*, pages 249–260, 2003.
- [26] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.
- [27] U. Meyer, P. Sanders, and J. Sibeyn, editors. *Algorithms for Memory Hierarchies*, volume 2625 of *Springer LNCS*. 2003.
- [28] S. Muthukrishnan. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.
- [29] N. Nisan and A. Wigderson. Rounds in communication complexity revisited. *SIAM Journal on Computing*, 22(1):211–219, 1993.
- [30] D. Olteanu. SPEX: Streamed and progressive evaluation of XPath. To appear in *Trans. Know. and Data Eng. (TKDE)*.
- [31] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [32] M. Ruhl. *Efficient Algorithms for New Computational Models*. PhD thesis, Massachusetts Institute of Technology, 2003.
- [33] L. Segoufin and C. Sirangelo. Constant-memory validation of streaming XML documents against DTDs. In *Proc. ICDT'07*, volume 4353 of *Springer LNCS*, pages 299–313, 2007.
- [34] L. Segoufin and V. Vianu. Validating streaming XML documents. In *Proc. PODS'02*, pages 53–64, 2002.
- [35] J. Vitter. External memory algorithms. In *Proc. PODS'98*, pages 119–128, 1998.
- [36] J. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33:209–271, 2001.

⁶Note that the lower bounds for read/write-streams presented in Section 4 do not rely on such an indivisibility assumption.