

Tight Lower Bounds for Query Processing on Streaming and External Memory Data

Martin Grohe¹, Christoph Koch², and Nicole Schweikardt¹

¹ Institut für Informatik, Humboldt-Universität Berlin, Germany
`{grohe,schweika}@informatik.hu-berlin.de`

² Database Group, Universität des Saarlandes, Saarbrücken, Germany
`koch@cs.uni-sb.de`

Abstract. We study a clean machine model for external memory and stream processing. We show that the number of scans of the external data induces a strict hierarchy (as long as work space is sufficiently small, e.g., polylogarithmic in the size of the input). We also show that neither joins nor sorting are feasible if the product of the number $r(n)$ of scans of the external memory and the size $s(n)$ of the internal memory buffers is sufficiently small, e.g., of size $o(\sqrt[3]{n})$. We also establish tight bounds for the complexity of XPath evaluation and filtering.

1 Introduction

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 reality that performance-wise, a small number of *sequential scans* of the data is strictly preferable over random data accesses. Indeed, the combined latencies and access times of moving to a certain position in external storage are by orders of magnitude greater than actually reading a small amount of data once the read head has been placed on its starting position.

Database engines rely on main memory buffers for assuring acceptable performance. These are usually small compared to the size of the externally stored data. Database technology – in particular query processing technology – has developed around this notion of memory hierarchies with layers of greatly varying sizes and access times. There has been a wealth of research on query processing and optimization along these lines (cf. e.g. [27, 14, 32, 22]). It seems that the current technologies scale up to current user expectations, but on closer investigation it may appear that our theoretical understanding of the problems involved – and of optimal algorithms for these problems – is not quite as developed.

Recently, data stream processing has become an object of study by the data management community (e.g. [15]) but from the viewpoint of database theory, this is, in fact, a special case of the query processing problem on data in external storage where we are limited to a single scan of the input data.

In summary, it appears that there are a variety of data management and query processing problems in which a comparably small but efficiently accessible main memory buffer is available and where accessing external data is costly

and is best performed by sequential read/write scans. This calls for an appropriate formal model that captures the essence of external memory and stream processing. In this paper, we study such a model, which employs a Turing machine with one *external memory tape* (external tape for short) and a number *internal memory tapes* (internal tapes for short). The external tape initially holds the input; the internal tapes correspond to the main memory buffers of a database management system and are thus usually small compared to the input.

As computational resources for inputs of size n , we study the space $s(n)$ available on the internal tapes and the number $r(n)$ of scans of (or, random accesses to) the external tape, and we write $ST(r, s)$ to denote the class of all problems solvable by (r, s) -bounded Turing machines, i.e., Turing machines which comply to the resource bounds $r(n)$ and $s(n)$ on inputs of size n .

Formally, we model the number of scans, respectively the number of random accesses, by the number of reversals of the Turing machine’s read/write head on the external tape. The number of reversals of the read/write head on the internal tapes remains unbounded. The reversals done by a read/write head are a clean and fundamental notion [8], but of course real external storage technology based on disks does not allow to reverse their direction of rotation. On the other hand, we can of course simulate k forward scans by $2k$ reversals in our machine model — and allowing for forward as well as backward scans makes our *lower* bound results even stronger.

As we allow the external tape to be both read and written to, the external tape can be viewed, for example, as modeling a hard disk. By closely watching reversals of the external tape head, anything close to random I/O will result in a very considerable number of reversals, while a full sequential scan of the external data can be effected cheaply. We will obtain strong lower bounds in this paper that show that even if the external tape (whose size we do not put a bound on) may be written to and re-read, certain bounds cannot be improved upon. For our matching upper bounds, we will usually not write to the external tape. Whenever one of our results requires writing to the external tape, we will explicitly indicate this.

The model is similar in spirit to the frameworks used in [18, 19], but differs from the previously considered *reversal complexity* framework [8]. Reversal complexity is based on Turing machines with a single read/write tape and the overall number of reversals of the read/write head the main computational resource. In our notion, only the number of reversals on the external tape is bounded, while reversals on the internal tapes are free; however, the space on the internal tapes is considered to be a limited resource.³

³ The justification for this assumption is simply that accessing data on disks is currently about five to six orders of magnitude slower than accessing main memory. For that reason, processor cycles and main memory access times are often neglected when estimating query cost in relational query optimizers, where cost measures are often exclusively based on the amount of expected page I/O as well as disk latency and access times. Moreover, by taking buffer space rather than running time as a parameter, we obtain more robust complexity classes that rely less on details of the machine model (see also [31]).

Apart from formalizing the $ST(r, s)$ model, we study its properties and locate a number of data management problems in the hierarchy of $ST(\cdot, \cdot)$ classes. Our technical contributions are as follows:

- We prove a reduction lemma (Lemma 4.1) which allows easy lower bound proofs for certain problems.
- We prove a hierarchy (Corollary 4.11 and Theorem 4.10), stating for each fixed number k that $k + 1$ scans of the external memory tape are strictly more powerful than k scans of the external memory tape.
- We consider machines where the product of the number of scans of the external memory tape, $r(n)$, and internal memory tape size, $s(n)$, is of size $o\left(\frac{n}{\log n}\right)$, where n is the input size, and show that *joins* cannot be computed by (r, s) -bounded Turing machines (cf., Lemma 4.4).
- We show that the *sorting* problem cannot be solved with $(o(\sqrt[5]{n}), O(\sqrt[5]{n}))$ -bounded Turing machines that are not allowed to write intermediate results to the external memory tape (cf., Corollary 4.9).
- We show (cf., Theorem 4.5) that for some *XQuery* queries, filtering is impossible for machines with $r(T) \cdot s(T) \in o\left(\frac{n}{\log n}\right)$, where n is the size of the input XML document T .
- We show (cf., Corollary 5.5) that for some *Core XPath* [12] queries, filtering is impossible for machines with $r(T) \cdot s(T) \in o(d)$, where d denotes the *depth* of the input XML document T . Furthermore, we show that the lower bound on *Core XPath* is *tight* in that we give an algorithm that solves the Core XPath filtering problem with a single scan of the external data (zero reversals) and $O(d)$ buffer space.

The primary technical machinery that we use for obtaining lower bounds is that of *communication complexity* (cf. [21]). Techniques from communication complexity have been used previously to study queries on streams [4, 6, 2, 3, 5, 23, 24, 18]. The work reported on in [4] addresses the problem of determining whether a given relational query can be evaluated scalably on a data stream or not at all. In comparison, we ask for tight bounds on query evaluation problems, i.e. we give algorithms for query evaluation that are in a sense worst-case optimal. As we do, the authors of [6] study XPath evaluation; however, they focus on *instance data complexity* while we study worst-case bounds. This allows us to find strong and tight bounds for a greater variety of query evaluation problems. Many of our results apply beyond stream processing in a narrow sense to a more general framework of queries on data in external storage. Also, our worst-case bounds apply for *any* evaluation algorithm possible, that is, our bounds are not in terms of complexity classes closed under reductions that allow for nonlinear expansions of the input (such as LOGSPACE) as is the case for the work on the complexity of XPath in [12, 13, 28].

Lower bound results for a machine model with *multiple* external memory tapes (or harddisks) are presented in [17]. In the present paper, we only consider a single external memory tape, and are consequently able to show (sometimes exponentially) stronger lower bounds.

Due to space limitations we had to defer detailed proofs of our results to the full version of this paper [16] which extends the present paper by an appendix that contains proofs of all results presented here.

2 Preliminaries

In this section we fix some basic notation concerning trees, streams, and query languages. We write \mathbb{N} for the set of non-negative integers. If M is a set, then 2^M denotes the set of all subsets of M . Throughout this paper we make the following convention: Whenever the letters r and s denote functions from \mathbb{N} to \mathbb{N} , then these functions are *monotone*, i.e., we have $r(x) \leq r(y)$ and $s(x) \leq s(y)$ for all $x, y \in \mathbb{N}$ with $x \leq y$.

Trees and Streams. We use standard notation for trees and streamed trees (i.e. *documents*). In particular, we write $Doc(T)$ to denote the XML document associated with an XML document tree T . An example is given in Figure 1.

Query Languages. By $Eval(\cdot, \cdot)$ we denote the evaluation function that maps each tuple (Q, T) , consisting of a query Q and a tree T to the corresponding query result. Let \mathcal{Q} be a query language and let $\mathcal{T}_1 \subseteq Trees_\tau$ and $\mathcal{T}_2 \subseteq \mathcal{T}_1$. We say that \mathcal{T}_2 can be filtered from \mathcal{T}_1 by a \mathcal{Q} -query if, and only if, there is a query $Q \in \mathcal{Q}$ such that the following is true for all $T \in \mathcal{T}_1$: $T \in \mathcal{T}_2 \iff Eval(Q, T) \neq \emptyset$.

We assume that the reader is familiar with first-order logic (*FO*) and monadic second-order logic (*MSO*). An *FO*- or *MSO*-sentence (i.e., a formula without any free variable) specifies a Boolean query, whereas a formula with exactly one free first-order variable specifies a unary query, i.e., a query which selects a set of nodes from the underlying input tree.

It is well-known [9, 30] that the MSO-definable Boolean queries on binary trees are exactly the (Boolean) queries that can be defined by finite (deterministic or nondeterministic) bottom-up tree automata. An analogous statement is true about MSO on unranked trees and unranked tree automata [7].

Theorem 4.5 in section 4 gives a lower bound on the worst case complexity of the language *XQuery*. As we prove a lower bound for *one* particular XQuery query, we do not give a formal definition of the language but refer to [33].

Apart from *FO*, *MSO*, and XQuery, we also consider a fragment of the XPath language, Core XPath [12, 13]. As we will prove not only lower, but also upper bounds for Core XPath, we give a precise definition of this query language in [16]. An example of a Core XPath query is

$$/descendant::*[child::A and child::B]/child::*,$$

which selects all children of descendants of the root node that (i.e., the descendants) have a child node labeled A and a child node labeled B.

Core XPath is a strict fragment of XPath [12], both syntactically and semantically. It is known that Core XPath is in LOGSPACE w.r.t. data complexity and P-complete w.r.t. combined complexity [13]. In [12], it is shown that Core XPath can be evaluated in time $O(|Q| \cdot |D|)$, where $|Q|$ is the size of the query and $|D|$ is the size of the XML data. Furthermore, every Core XPath query is equivalent to a unary MSO query on trees (cf., e.g., [11]).

Communication complexity. To prove basic properties and lower bounds for our machine model, we use some notions and results from *communication complexity*, cf., e.g., [21].

Let A, B, C be sets and let $F : A \times B \rightarrow C$ be a function. In Yao's [34] basic model of communication two players, Alice and Bob, jointly want to evaluate $F(x, y)$, for input values $x \in A$ and $y \in B$, where Alice only knows x and Bob only knows y . The two players can exchange messages according to some fixed protocol \mathcal{P} that depends on F , but not on the particular input values x, y . The exchange of messages starts with Alice sending a message to Bob and ends as soon as one of the players has enough information on x and y to compute $F(x, y)$.

\mathcal{P} is called a *k-round protocol*, for some $k \in \mathbb{N}$, if the exchange of messages consists, for each input $(x, y) \in A \times B$, of at most k rounds. The *cost* of \mathcal{P} on input (x, y) is the number of bits communicated by \mathcal{P} on input (x, y) . The *cost* of \mathcal{P} is the *maximal* cost of \mathcal{P} over all inputs $(x, y) \in A \times B$. The *communication complexity* of F , $\text{comm-compl}(F)$, is defined as the minimum cost of \mathcal{P} , over all protocols \mathcal{P} that compute F . For $k \geq 1$, the *k-round communication complexity* of F , $\text{comm-compl}_k(F)$, is defined as the minimum cost of \mathcal{P} , over all k -round protocols \mathcal{P} that compute F .

Many powerful tools are known for proving lower bounds on communication complexity, cf., e.g., [21]. In the present paper we will use the following basic lower bound for the problem of deciding whether two sets are disjoint.

Definition 2.1. For $n \in \mathbb{N}$ let the function $\text{Disj}_n : 2^{\{1, \dots, n\}} \times 2^{\{1, \dots, n\}} \rightarrow \{0, 1\}$ be given via

$$\text{Disj}_n(X, Y) := \begin{cases} 1, & \text{if } X \cap Y = \emptyset \\ 0, & \text{otherwise.} \end{cases} \quad \square$$

Theorem 2.2 (cf., e.g., [21]). For every $n \in \mathbb{N}$, $\text{comm-compl}(\text{Disj}_n) \geq n$.

3 Machine Model

We consider Turing machines with (1) an input tape, which is a read/write tape and will henceforth be called “external memory tape” or “external tape”, for short, (2) an arbitrary number u of work tapes, which will henceforth be called “internal memory tapes” or “internal tapes”, for short, and, if needed, (3) an additional write-only output tape.

Let M be such a Turing machine and let ρ be a run of M . By $\text{rev}(\rho)$ we denote the number of times the external memory tape's head changes its direction in the run ρ . For $i \in \{1, \dots, u\}$ we let $\text{space}(\rho, i)$ be the number of cells of internal memory tape i that are used by ρ .

The class $ST(r, s)$ for strings.

Definition 3.1 ($ST(r, s)$ for strings). Let $r : \mathbb{N} \rightarrow \mathbb{N}$ and $s : \mathbb{N} \rightarrow \mathbb{N}$.

- (a) A Turing machine M is (r, s) -bounded, if every run ρ of M on an input of length n satisfies the following conditions:

- (1) ρ is finite, (2) $1 + \text{rev}(\rho) \leq r(n)$,⁴ and (3) $\sum_{i=1}^u \text{space}(\rho, i) \leq s(n)$, where u is the number of internal tapes of M .
- (b) A string-language $L \subseteq \Sigma^*$ belongs to the class $ST(r, s)$ (resp., $NST(r, s)$), if there is a deterministic (respectively, nondeterministic) (r, s) -bounded Turing machine which accepts exactly those $w \in \Sigma^*$ that belong to L .
- (c) A function $f : \Sigma^* \rightarrow \Sigma^*$ belongs to the class $ST(r, s)$, if there is a deterministic (r, s) -bounded Turing machine which produces, for each input string $w \in \Sigma^*$, the string $f(w)$ on its write-only output tape. \square

For classes R and S of functions, we let $ST(R, S) := \bigcup_{r \in R, s \in S} ST(r, s)$.

If $k \in \mathbb{N}$ is a constant, then we write $ST(k, s)$ instead of $ST(r, s)$, where r is the function with $r(x) = k$ for all $x \in \mathbb{N}$. We freely combine these notations and use them for $NST(\cdot, \cdot)$ instead of $ST(\cdot, \cdot)$, too.

If we think of the external memory tape of an (r, s) -bounded Turing machine as representing the incoming stream, stored on a hard disk, then admitting the external memory tape's head to reverse its direction might not be very realistic. But as we mainly use our model to prove lower bounds, it does not do any harm either, since the reversals can be used to simulate *random access*. Random access can be introduced explicitly into our model as follows: A *random access Turing machine* is a Turing machine M which has a special internal memory tape that is used as *random access address tape*, i.e., on which only binary strings can be written. Such a binary string is interpreted as a positive integer specifying an external memory address, that is, the position index number of a cell on the external tape (we think of the external tape cells being numbered by positive integers). The machine has a special state q_{ra} . If q_{ra} is entered, then in one step the external memory tape head is moved to the cell that is specified by the number on the random access address tape, and the content of the random access address tape is deleted.

Definition 3.2. Let $q, r, s : \mathbb{N} \rightarrow \mathbb{N}$. A random access Turing machine M is (q, r, s) -bounded, if it is (r, s) -bounded (in the sense of an ordinary Turing machine) and, in addition, every run ρ of M on an input of length n involves at most $q(n)$ random accesses. \square

Noting that a random access can be simulated with at most 2 changes of the direction of the external memory tape head, one immediately obtains:

Lemma 3.3. *Let $q, r, s : \mathbb{N} \rightarrow \mathbb{N}$. If a problem can be solved by a (q, r, s) -bounded random access Turing machine, then it can also be solved by an $(r + 2q, O(s))$ -bounded Turing machine.*

In the subsequent parts of this paper, we will concentrate on ordinary Turing machines (without random access). Via Lemma 3.3, all results can be transferred from ordinary Turing machines to random access Turing machines.

⁴ It is convenient for technical reasons to add 1 to the number $\text{rev}(\rho)$ of changes of the head direction. As defined here, $r(n)$ bounds the number of sequential scans of the external memory tape rather than the number of changes of head directions.

The class $ST(r, s)$ for trees. We make an analogous definition to $ST(r, s)$ on strings for trees. This definition is given in detail in [16].

4 Lower bounds for the ST model

A reduction lemma. The following lemma provides a convenient tool for showing that a problem L does not belong to $ST(r, s)$. The lemma's assumption can be viewed as a *reduction* from the problem $Disj_n(\cdot, \cdot)$ to the problem L .

Lemma 4.1. *Let Σ be an alphabet and let $\lambda : \mathbb{N} \rightarrow \mathbb{N}$ such that the following is true: For every $n_0 \in \mathbb{N}$ there is an $n \geq n_0$ and functions $f_n, g_n : 2^{\{1, \dots, n\}} \rightarrow \Sigma^*$ such that for all $X, Y \subseteq \{1, \dots, n\}$ the string $f_n(X)g_n(Y)$ has length $\leq \lambda(n)$.*

Then we have for all $r, s : \mathbb{N} \rightarrow \mathbb{N}$ with $r(\lambda(n)) \cdot s(\lambda(n)) \in o(n)$, that there is no (r, s) -bounded deterministic Turing machine which accepts a string of the form $f_n(X)g_n(Y)$ if, and only if, $X \cap Y = \emptyset$.

Disjointness. Every n -bit string $x = x_1 \cdots x_n \in \{0, 1\}^n$ specifies a set $S(x) := \{i : x_i = 1\} \subseteq \{1, \dots, n\}$. Let L_{Disj} consist of those strings $x\#y$ where x and y specify disjoint subsets of $\{1, \dots, n\}$, for some $n \geq 1$. That is,

$$L_{Disj} := \{ x\#y : \text{ex. } n \geq 1 \text{ with } x, y \in \{0, 1\}^n \text{ and } S(x) \cap S(y) = \emptyset \}.$$

From Lemma 4.1 one easily obtains

Proposition 4.2. *Let $r : \mathbb{N} \rightarrow \mathbb{N}$ and $s : \mathbb{N} \rightarrow \mathbb{N}$. If $r(n) \cdot s(n) \in o(n)$, then $L_{Disj} \notin ST(r, s)$.*

The bound given by Proposition 4.2 is tight, as it can be easily seen that $L_{Disj} \in ST(r, s)$ for all $r, s : \mathbb{N} \rightarrow \mathbb{N}$ with $r(n) \cdot s(n) \in \Omega(n)$.

Joins. Let τ be the set of tag names $\{\text{rels}, \text{rel1}, \text{rel2}, \text{tuple}, \text{no1}, \text{no2}, 0, 1\}$. We represent a pair (A, B) of finite relations $A, B \subseteq \mathbb{N}^2$ as a τ -tree $T(A, B)$ whose associated XML document $Doc(T(A, B))$ is a Σ_τ -string of the following form: For each number $i \in \mathbb{N}$ let $Bin(i) = b_{\ell_i}^{(i)} \cdots b_0^{(i)}$ be the binary representation of i . For each tuple $(i, j) \in \{1, \dots, n\}^2$ let $Doc(i, j) :=$

$$\langle \text{tuple} \rangle \langle \text{no1} \rangle \langle b_{\ell_i}^{(i)} \rangle / \cdots \langle b_0^{(i)} \rangle / \langle / \text{no1} \rangle \langle \text{no2} \rangle \langle b_{\ell_j}^{(j)} \rangle / \cdots \langle b_0^{(j)} \rangle / \langle / \text{no2} \rangle \langle / \text{tuple} \rangle.$$

For each finite relation $A \subseteq \mathbb{N}^2$ let $t_1, \dots, t_{|A|}$ be the lexicographically ordered list of all tuples in A . We let $Doc(A) := Doc(t_1) \cdots Doc(t_{|A|})$. Finally, we let

$$Doc(T(A, B)) := \langle \text{rels} \rangle \langle \text{rel1} \rangle Doc(A) \langle / \text{rel1} \rangle \langle \text{rel2} \rangle Doc(B) \langle / \text{rel2} \rangle \langle / \text{rels} \rangle.$$

It is straightforward to see that the string $Doc(T(A, B))$ has length $O((|A| + |B|) \cdot \log n)$, if $A, B \subseteq \{1, \dots, n\}^2$.

We write $A \bowtie_1 B$ to denote the join of A and B on their first component, i.e., $A \bowtie_1 B := \{(x, y) : \exists z A(z, x) \wedge B(z, y)\}$. We let

$$\begin{aligned} \mathcal{T}_{Rels} &:= \{ T(A, B) : A, B \subseteq \mathbb{N}^2, A, B \text{ finite} \} \\ \mathcal{T}_{EmptyJoin} &:= \{ T(A, B) \in \mathcal{T}_{Rels} : A \bowtie_1 B = \emptyset \} \\ \mathcal{T}_{NonEmptyJoin} &:= \{ T(A, B) \in \mathcal{T}_{Rels} : A \bowtie_1 B \neq \emptyset \}. \end{aligned}$$

Lemma 4.3. $\mathcal{T}_{NonEmptyJoin}$ can be filtered from \mathcal{T}_{RelS} by an XQuery query.

Lemma 4.4. Let $r, s : Trees_\tau \rightarrow \mathbb{N}$.

If $r(T) \cdot s(T) \in o\left(\frac{size(T)}{\log(size(T))}\right)$, then $\mathcal{T}_{EmptyJoin} \not\subseteq ST(r, s)$.

From Lemma 4.4 and Lemma 4.3 we immediately obtain a lower bound on the worst-case data complexity for filtering relative to an XQuery query:

Theorem 4.5. The tree-language $\mathcal{T}_{EmptyJoin}$

(a) can be filtered from \mathcal{T}_{RelS} by an XQuery query,

(b) does not belong to the class $ST(r, s)$, whenever $r, s : Trees_\tau \rightarrow \mathbb{N}$ with

$$r(T) \cdot s(T) \in o\left(\frac{size(T)}{\log(size(T))}\right).$$

Remark 4.6. Let us note that the above bound is “almost tight” in the following sense: The problem of deciding whether $A \bowtie_1 B = \emptyset$ and, in general, all FO-definable problems belong to $ST(1, n)$ – in its single scan of the external memory tape, the Turing machine simply copies the entire input on one of its internal memory tapes and then evaluates the FO-sentence by the straightforward LOGSPACE algorithm for FO-model-checking (cf. e.g. [1]). \square

Sorting. By KEYSORT, we denote the problem of sorting a set S of tuples $t = (K, V)$ consisting of a key K and a value V by their keys. Let $ST^-(r, s)$ denote the class of all problems in $ST(r, s)$ that can be solved without writing to the external memory tape. Then,

Theorem 4.7. Let $r, s : \mathbb{N} \rightarrow \mathbb{N}$. If KEYSORT is in $ST^-(r, s)$, then computing the natural join $A \bowtie B$ of two finite relations A, B is in

$$ST^-\left(r(n^2) + 2, s(n^2) + O(\log n) + O(\max_{t \in A \cup B} |t|)\right).$$

Remark 4.8. Given that the size of relations A and B is known (which is usually the case in practical database management systems DBMS), the algorithm given in the previous proof can do a merge-join without additional scans after the sort run and without a need to buffer more than one tuple. This is guaranteed even if both relations may contain many tuples with the same join key – in current implementations of the merge join in DBMS, this may lead to grass-roots swapping. The (substantial) *practical* drawback of the join algorithm of the proof of Theorem 4.7, however, is that much larger relations A', B' need to be sorted: indeed $|A'| = |A| * |B|$. \square

Corollary 4.9.

(a) Let $r, s : \mathbb{N} \rightarrow \mathbb{N}$ such that $r(n^2) \cdot (s(n^2) + \log n) \in o\left(\frac{n}{\log n}\right)$.

Then, KEYSORT $\notin ST^-(r, s)$.

(b) KEYSORT $\notin ST^-(o(\sqrt[3]{n}), O(\sqrt[3]{n}))$.

It is straightforward to see that by using MergeSort, the sorting problem can be solved using $O(\log n)$ scans of external memory provided that *three* external memory tapes are available. (In [17], this logarithmic bound is shown to be tight, for arbitrarily many external tapes.) Corollary 4.9 gives an exponentially stronger lower bound for the case of a single external tape.

A hierarchy based on the number of scans.

Theorem 4.10. For every fixed $k \geq 1$,

$$ST(k, O((\log k) + \log n)) \cap NST(1, O(k \cdot \log n)) \not\subseteq ST(k-1, o(\frac{\sqrt{n}}{k^5(\log n)^3})).$$

The proof of this theorem is based on a result due to Duris, Galil and Schnitger [10]. Theorem 4.10 directly implies

Corollary 4.11. For every fixed $k \in \mathbb{N}$ and all classes S of functions from \mathbb{N} to \mathbb{N} such that $O(\log n) \subseteq S \subseteq o(\frac{\sqrt{N}}{(\lg n)^3})$ we have $ST(k, S) \subsetneq ST(k+1, S)$.

Remark 4.12. On the other hand, of course, the hierarchy collapses if internal memory space is at least linear in the size of the input: For every $r : \mathbb{N} \rightarrow \mathbb{N}$ and for every $s : \mathbb{N} \rightarrow \mathbb{N}$ with $s(n) \in \Omega(n)$, we have

$$ST(r, s) \subseteq ST(1, n + s(n)) \quad \text{and} \quad ST(r, O(s(n))) = \text{DSPACE}(O(s(n))).$$

5 Tight bounds for filtering and query evaluation on trees

Lower bound. We need the following notation: We fix a set τ of tag names via $\tau := \{ \text{root, left, right, blank} \}$. Let T_1 be the τ -tree from Figure 1. Note that T_1 has a unique leaf v_1 labeled with the tag name “left”. For any arbitrary τ -tree T we let $T_1(T)$ be the τ -tree rooted at T_1 ’s root and obtained by identifying node v_1 with the root of T and giving the label “left” to this node. Now, for every $n \geq 2$ let T_n be the τ -tree inductively defined via $T_n := T_1(T_{n-1})$. It is straightforward to see that T_n has exactly $2n$ leaves labeled “blank”. Let $x_1, \dots, x_n, y_1, \dots, y_1$ denote these leaves, listed in *document order* (i.e., in the order obtained by a pre-order depth-first left-to-right traversal of T_n). For an illustration see Figure 2.

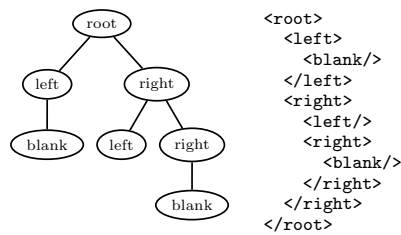


Fig. 1. A τ -tree T_1 and its XML document $Doc(T_1) \in \Sigma_\tau^*$ with tag names $\tau := \{ \text{root, left, right, blank} \}$.

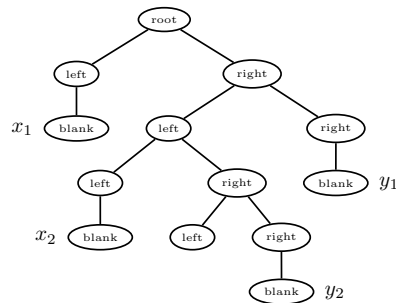


Fig. 2. Tree T_2 and nodes x_1, x_2, y_1, y_2 .

We let $\tau_{01} := \tau \cup \{0, 1\}$. For all sets $X, Y \subseteq \{1, \dots, n\}$ let $T_n(X, Y)$ be the τ_{01} -tree obtained from T_n by replacing, for each $i \in \{1, \dots, n\}$, (*) the label “blank” of leaf x_i by the label 1 if $i \in X$, and by the label 0 otherwise and (*) the label “blank” of leaf y_i by the label 1 if $i \in Y$, and by the label 0 otherwise.

We let

$$\begin{aligned}\mathcal{T}_{Sets} &:= \{ T_n(X, Y) : n \geq 1, X, Y \subseteq \{1, \dots, n\} \}, \\ \mathcal{T}_{Disj} &:= \{ T_n(X, Y) \in \mathcal{T}_{Sets} : X \cap Y = \emptyset \}, \\ \mathcal{T}_{NonDisj} &:= \{ T_n(X, Y) \in \mathcal{T}_{Sets} : X \cap Y \neq \emptyset \}.\end{aligned}$$

- Lemma 5.1.** (a) *There is a Core XPath query Q such that the following is true for all τ -trees $T \in \mathcal{T}_{Sets}$: $Eval(Q, T) \neq \emptyset \iff T \in \mathcal{T}_{NonDisj}$.*
(b) *There is a FO-sentence φ such that the following is true for all τ -trees T : $T \models \varphi \iff T \in \mathcal{T}_{NonDisj}$.*

Lemma 5.2. *Let $r, s : Trees_\tau \rightarrow \mathbb{N}$. If $r(T) \cdot s(T) \in o(depth(T))$, then $\mathcal{T}_{NonDisj} \notin ST(r, s)$.*

From Lemma 5.1 and Lemma 5.2 we directly obtain a lower bound on the worst-case data complexity of Core XPath filtering:

Theorem 5.3. *The tree-language $\mathcal{T}_{NonDisj}$*

- (a) *can be filtered from \mathcal{T}_{Sets} by a Core XPath query,*
- (b) *is definable by an FO-sentence (and therefore, also definable by a Boolean MSO query and recognizable by a tree automaton), and*
- (c) *does not belong to the class $ST(r, s)$, whenever $r, s : Trees_\tau \rightarrow \mathbb{N}$ with $r(T) \cdot s(T) \in o(depth(T))$.*

In the following subsection we match this lower bound with the corresponding upper bound.

Upper bounds. Recall that a tree-language $\mathcal{T} \subseteq Trees_\tau$ is definable by an MSO-sentence if, and only if, it is recognizable by an unranked tree automaton, respectively, if, and only if, the language $\{BinTree(T) : T \in \mathcal{T}\}$ of associated *binary* trees is recognizable by an ordinary (ranked) tree automaton (cf., e.g., [7, 9, 30]).

Theorem 5.4 (implicit in [25, 29]). *Let $\mathcal{T} \subseteq Trees_\tau$ be a tree-language. If \mathcal{T} is definable by an MSO-sentence (or, equivalently, recognizable by a ranked or an unranked finite tree automaton), then $\mathcal{T} \in ST(1, depth(\cdot) + 1)$.*

Recall that every Core XPath query is equivalent to a unary MSO query. Thus a Core XPath filter can be phrased as an MSO sentence on trees. From the Theorems 5.4 and 5.3 we therefore immediately obtain a tight bound for Core XPath filtering:

- Corollary 5.5.** (a) *Filtering from the set of unranked trees with respect to every fixed Core XPath query Q belongs to $ST(1, O(depth(\cdot)))$.*
(b) *There is a Core XPath query Q such that, for all $r, s : Trees_\tau \rightarrow \mathbb{N}$ with $r(T) \cdot s(T) \in o(depth(T))$, filtering w.r.t. Q does not belong to $ST(r, s)$.*

Next, we provide an upper bound for the problem of computing the set $Eval(Q, T)$ of nodes in an input tree T matching a unary MSO (or Core XPath) query Q . We first need to clarify what this means, because writing the subtree of each matching node onto the output tape requires a very large amount of internal memory (or a large number of head reversals on the external memory tape), and this gives us no appropriate characterization of the difficulty of the problem. We study the problem of computing, for each node matched by Q , its index in the tree, in the order in which they appear in the document $Doc(T)$. We distinguish between the case where these indexes are to be written to the output tape in ascending order and the case where they are to be output in descending (i.e., reverse) order.

Theorem 5.6 (implicit in [26, 20]). *For every unary MSO or Core XPath query Q , the problem of computing, for input trees T , the nodes in $Eval(Q, T)$*

- (a) *in ascending order belongs to $ST(3, O(\text{depth}(\cdot)))$.*
- (b) *in reverse order belongs to $ST(2, O(\text{depth}(\cdot)))$.*

Note that this bound is tight: From Corollary 5.5(c) we know that, for some Core XPath query Q , not even *filtering* (i.e., checking whether $Eval(Q, T)$ is empty) is possible in $ST(r, s)$ if $r(T) \cdot s(T) \in o(\text{depth}(T))$.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
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.
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. A. Arasu, B. Babcock, T. Green, A. Gupta, and J. Widom. “Characterizing Memory Requirements for Queries over Continuous Data Streams”. In *Proc. PODS'02*, pages 221–232, 2002.
5. 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.
6. 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.
7. A. Brüggemann-Klein, M. Murata, and D. Wood. “Regular Tree and Regular Hedge Languages over Non-ranked Alphabets: Version 1, April 3, 2001”. Technical Report HKUST-TCSC-2001-05, Hong Kong Univ. of Science and Technology, 2001.
8. J.-E. Chen and C.-K. Yap. “Reversal Complexity”. *SIAM J. Comput.*, **20**(4):622–638, Aug. 1991.
9. J. Doner. “Tree Acceptors and some of their Applications”. *Journal of Computer and System Sciences*, **4**:406–451, 1970.
10. P. Duris, Z. Galil, and G. Schnitger. Lower bounds on communication complexity. *Information and Computation*, 73:1–22, 1987. Journal version of STOC'84 paper.
11. G. Gottlob and C. Koch. “Monadic Datalog and the Expressive Power of Web Information Extraction Languages”. *Journal of the ACM*, **51**(1):74–113, 2004.
12. G. Gottlob, C. Koch, and R. Pichler. “Efficient Algorithms for Processing XPath Queries”. In *Proc. VLDB 2002*, pages 95–106, Hong Kong, China, 2002.

13. G. Gottlob, C. Koch, and R. Pichler. "The Complexity of XPath Query Evaluation". In *Proc. PODS'03*, pages 179–190, San Diego, California, 2003.
14. G. Graefe. "Query Evaluation Techniques for Large Databases". *ACM Computing Surveys*, **25**(2):73–170, June 1993.
15. T. J. Green, G. Miklau, M. Onizuka, and D. Suciu. "Processing XML Streams with Deterministic Automata". In *Proc. ICDT'03*, 2003.
16. M. Grohe, C. Koch, and N. Schweikardt. Tight lower bounds for query processing on streaming and external memory data. Technical report CoRR cs.DB/0505002, 2005. Full version of ICALP'05 paper.
17. M. Grohe and N. Schweikardt. Lower bounds for sorting with few random accesses to external memory. In *Proc. PODS*, 2005. To appear.
18. 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.
19. J. E. Hopcroft and J. D. Ullman. Some results on tape-bounded Turing machines. *Journal of the ACM*, **16**(1):168–177, 1969.
20. C. Koch. "Efficient Processing of Expressive Node-Selecting Queries on XML Data in Secondary Storage: A Tree Automata-based Approach". In *Proc. VLDB 2003*, pages 249–260, 2003.
21. E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge Univ. Press, 1997.
22. U. Meyer, P. Sanders, and J. Sibeyn, editors. *Algorithms for Memory Hierarchies*, volume 2832 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.
23. J. Munro and M. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.
24. S. Muthukrishnan. Data streams: algorithms and applications. In *Proc. 14th SODA*, pages 413–413, 2003.
25. A. Neumann and H. Seidl. "Locating Matches of Tree Patterns in Forests". In *Proc. 18th FSTTCS, LNCS 1530*, pages 134–145, 1998.
26. F. Neven and J. van den Bussche. "Expressiveness of Structured Document Query Languages Based on Attribute Grammars". *J. ACM*, **49**(1):56–100, Jan. 2002.
27. R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 2002.
28. L. Segoufin. "Typing and Querying XML Documents: Some Complexity Bounds". In *Proc. PODS'03*, pages 167–178, 2003.
29. L. Segoufin and V. Vianu. "Validating Streaming XML Documents". In *Proc. PODS'02*, 2002.
30. J. Thatcher and J. Wright. "Generalized Finite Automata Theory with an Application to a Decision Problem of Second-order Logic". *Math. Syst. Theory*, **2**(1):57–81, 1968.
31. P. van Emde Boas. "Machine Models and Simulations". In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 1, chapter 1, pages 1–66. Elsevier Science Publishers B.V., 1990.
32. J. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, **33**(2):209–271, June 2001.
33. World Wide Web Consortium. "XQuery 1.0 and XPath 2.0 Formal Semantics. W3C Working Draft (Aug. 16th 2002), 2002. <http://www.w3.org/XML/Query>.
34. A. Yao. Some complexity questions related to distributive computing. In *Proc. 11th STOC*, pages 209–213, 1979.