On the Optimality of Holistic Algorithms for Twig Queries

Byron Choi[1], Malika Mahoui[1], and Derick Wood[2]

[1] University of Pennsylvania
{kkchoi, mmahoui}@seas.upenn.edu
[2] HKUST
dwood@cs.ust.hk

Abstract. Streaming XML documents has many emerging applications. However, in this paper, we show that the restrictions imposed by data streaming are too restrictive for processing twig queries – the core operation for XML query processing. Previous proposed algorithm `TwigStack` is an optimal algorithm for processing twig queries with only descendent edges over streams of nodes. The cause of the suboptimality of the `TwigStack` algorithm is the structurally recursions appearing in XML documents. We show that without relaxing the data streaming model, it is not possible to develop an optimal holistic algorithm for twig queries. Also the computation of the twig queries is not memory bounded. This motivates us to study two variations of the data streaming model: (1) offline sorting is allowed and the algorithm is allowed to select the correct nodes to be streamed and (2) multiple scans on the data streams are allowed. We show the lower bounds of the two variations.

1 Introduction

As much database research has shifted its focus from relational systems to the extensions on semistructured data and XML [1], there is a growing need for processing XML document streams efficiently. Recently, streaming XML documents has many emerging applications, e.g. in monitoring stock markets and in managing the network traffic. In such applications, the systems do not know the size of the XML document streams and do not have the access to sort the items in the streams. Also, an item in a stream is either stored in the main memory or discarded once it has been processed. As a key step in developing a data stream management system, Arasu and his colleagues [2] study the theoretical issues of some important classes of relational queries over data streams. However, none of the previous work has studied some theoretical issues of the core operation – the twig queries – for XML query processing over data streams.

Recently, efficient algorithms [6, 10, 9, 8] have been proposed to evaluate twig queries in XML-enabled relational databases. These algorithms typically decompose the queries into subqueries, evaluate the subqueries and combine the results of the subqueries. One drawback of this approach is that intermediate results can be large even though the results can be relatively small. In contrast, holistic algorithms for twig queries evaluate a twig query *as a whole*. Hence, memory is not used unwisely to store irrelevant nodes. Algorithms as such are useful in data streaming since the sizes of the streams is usually not known in advance.

The holistic twig join algorithm called `TwigStack` was recently proposed by Bruno and his co-researchers [4]. It evaluates twig queries as a whole over streams of XML documents efficiently. Each node in the twig query is associated with a stream of nodes. The algorithm scans the streams only once and assigns constant memory only to the nodes that participate in at least one solution. Thus, the algorithm is asymptotically optimal among all sequential algorithms that read the entire input. However, the algorithm is suboptimal when the twig queries contain child edges.

We demonstrate that it is not possible to develop an optimal holistic algorithm for evaluating twig queries. This negative result implies that either the data streaming model [3] or our assumptions about the streams are too restrictive for processing twig queries. We consider two variations of the data streaming model. First, document preprocessing is allowed and the corresponding optimal holistic algorithm is allowed selecting the correct nodes to be streamed. Second, multiple scans of the data streams are allowed. We present some lower bounds for the two computation models.

1.1 Background and the Problem Statement

We describe the node representation and the assumptions about the streams of nodes that we use. We also briefly describe the syntax and the semantics of twig queries. Finally, we describe the technical problem that we investigate.

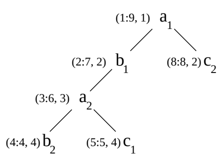

Figure. 1. An example XML document.

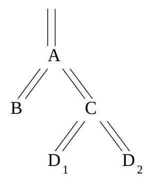

Figure. 2. Graphical representation of $//A\ (//B,\ //C\ (//D_1,\ //D_2))$.

Twig queries are applied to the XML documents represented as follows. A document is modeled as a label tree. Nodes are represented by (1) the preorder number, (2) the postorder number and (3) the depth of the node. An example is to be found in Fig. 1.

Assumptions. We assume that the preorder number, the postorder number, the depth and the label are the only accessible information of a node. These assumptions imply the followings:

1. Given any two nodes, one can compute the ancestor-descendent and parent-child relationship of the two nodes in constant time.

2. One can compute the depth of a node in constant time.

Definition 1. *The syntax of a twig query, $Twig$, is defined as follows in Backus-Naur Form:*
Step ::= / | //
$NodeTest$ ::= label
Path ::= Step $NodeTest$ | Step $NodeTest$ Path
$Twig$ ::= Path | Path ($Twig$, $Twig$, ..., $Twig$)

We first give the syntax of twig queries in Definition 1. The steps '/' and '//' denote advancing one step from a parent node to its children nodes and from an ancestor node to its descendent nodes, respectively. Given a set of nodes as input, a $NodeTest$ returns the subset of nodes that have the same label as the $NodeTest$. A path is a sequence of alternating steps and node tests. A twig is path possibly followed by branches, that are also twigs. Fig. 2 graphically represents the twig query $//A$ ($//B$, $//C$ ($//D_1$, $//D_2$)). The subscript of the D-node test denotes its occurrence in the query. The *solution* of a twig query is the set of all node combinations that satisfy the query.

Since a stream contains only nodes with the same label, the partial ordering of two nodes in a stream is only defined if they have the same label. $a_1 < a_2$ is interpreted as node a_1 precedes node a_2 in the A-stream.

Our computation model assumes that there is a stream of nodes associated with a node test in the twig query such that the nodes satisfy the node test. A *stream* can be viewed as a pop-only stack. For example, given the document and the query shown in Fig. 1 and Fig. 2, respectively, A, B, C, D_1 and D_2 are associated with a stream of A-, B-, C-, and D-nodes, denoted by T_A, T_B, T_C, T_{D_1} and T_{D_2} and defined by $T_A = [a_1, a_2]$, $T_B = [b_1, b_2]$, $T_C = [c_1, c_2]$, $T_{D_1} = [\,]$ and $T_{D_2} = [\,]$. We will call the nodes that participate in at least one solution the *useful* nodes and the remaining nodes *useless* nodes.

Definition 2. *An algorithm for twig queries is* asymptotically optimal *if it returns the solution of the query by using: (1) a single forward scan of the streams, (2) constant memory for each useful nodes and (3) constant time processing each of the nodes in the streams.*

The problem statement is given as follows. *Given a twig query and its associated streams, is it possible to design an asymptotically optimal algorithm for all twig queries.*

1.2 The Suboptimality of the `TwigStack` Algorithm

The `TwigStack` algorithm [4] has been proved to be asymptotically optimal for evaluating descendent-edge-only twig queries. We demonstrate the relevant parts of the algorithm with two running examples. The first example show an optimal evaluation of a twig query and the second example shows a suboptimal evaluation of a twig query. In both cases, we use the `TwigStack` algorithm.

Example 1. We demonstrate the evaluation of the descendent-edge-only queries in this example. The query $Q_1 = //A\ (//B,\ //C)$ is applied to the document shown in Fig. 1. Nodes in a stream are given in preorder: $T_A = [a_1,\ a_2]$, $T_B = [b_1,\ b_2]$ and $T_C = [c_1,\ c_2]$. Initially, the nodes a_1, b_1 and c_1 are at the *top* of T_A, T_B and T_C, respectively. Since b_1 and c_1 are descendents of a_1, a_1 is a useful node because of the implications of Q_1. The stream T_A is advanced. The next A-node is a_2. We examine the node b_1 which is useful because of a_1. T_B is advanced and b_2 is the next B-node. Similarly, a_2 is also useful. Now there are no further A-nodes in T_A. The B-node b_2 is useful because of a_2. T_B is advanced. The stream T_C is advanced and c_2 is useful because of a_1. Hence the solution is returned by using one forward scan, constant memory for each useful nodes and constant time processing each of the nodes in the stream. We have skipped the details of the reconstruction of the result from the encoding of the intermediate results produced by the `TwigStack` algorithm.

Example 2. We demonstrate the evaluation of a twig query containing some child edges [4]. The `TwigStack` algorithm is suboptimal in this case. The query $Q_2 = //A\ (/B,\ /C)$ and the same streams as in the previous example. No algorithm can determine if a_1 is useful without advancing T_C. If T_C is advanced, however, c_1 is discarded and a_2 will be declared useless. If the A stream is advanced, a_1 is discarded. To ensure that all useful nodes are reported, a_1 must be stored in the main-memory. The `TwigStack` algorithm will process a_2 and then a_1. This shows the suboptimality of the algorithm.

Definition 3. *A label l is* structurally recursive *if $l \to \alpha$, where α is the content model of l in the DTD, and there is at least one occurrence of l in α directly or indirectly. A DTD is* recursive *if it contains a structurally recursive label.*

There are two important observations about the `TwigStack` algorithm. First, the nodes in a stream are sorted by the preorder numbering. Having this fact and the assumptions in Section 1.1, before the query evaluation, we have enough information answering the descendent-edge-only twig queries optimally. It is natural to order the nodes by some other sort keys. Second, the reason for suboptimality is that an A-node can have A-descendents; that is, the label A is *structurally recursive*.

The organization of the rest of the paper is as follows. We show in Section 2 that the data streaming model is too restrictive for processing the twig queries under the assumptions of Section 1.1. In Section 3, we show that when offline processing is allowed, optimal holistic algorithms become possible but the lower bound of offline processing is high. Section 4 shows that when multiple scans of the streams are allowed, the optimality becomes possible but the lower bound of the number of scans is high. We provide conclusions in Section 5.

2 First Few Questions on Optimality

Our first few questions on processing twig queries in the data streaming context are: (1) is it possible to design such algorithms and (2) do they use bounded memory. We argue that the answer to both questions is negative. In Theorem 1, we will show that the data streaming model is too restrictive for the holistic evaluation of twig queries.

Proposition 1. *To satisfy the memory requirement of data streaming, and to allow one scan of the streams, without loss of generality, if* $a_i < a_j$ *then for all* b_i *in* N_i *and* b_j *in* N_j, $b_i < b_j$, *where* $N_i = a_i/p$ *and* $N_j = a_j/p$, p *is a* $Path$.

Proof. Proof by a structural induction on $Path$ and by contradiction. □

Theorem 1. *Given the assumptions in Section 1.1, there is no ordering of nodes such that the holistic evaluation of all twig queries is optimal.*

Proof. Consider the complete binary tree with 15 A nodes shown in Appendix A. Proposition 1 implies that if $a_i < a_j$, then the descendent nodes of $a_i <$ the descendent nodes of a_j. Consider the query $//A$ ($/A$, $/A$). We have three streams of A nodes, T_{A_1}, T_{A_2} and T_{A_3}. Let the top node of the three streams are p_1, p_2 and p_3, respectively.

The possible partial ordering of a_1 and some other A-nodes are determined by performing a case analysis. Without loss of generality, assume that $a_2 < a_3$.

Case 1. Assume that $a_1 < a_2$. It implies that there must be a configuration (*) such that $p_1 = a_1$, $p_2 = a_2$ and $p_3 = a_3$. If the top of T_{A_3} is a_3, then a_2 and all of its descendents in T_{A_3} have been discarded. When p_1 is advanced to a_2, no solution is reported. Hence, the assumption is not valid.

Case 2. Assume that $a_3 < a_1$. The configuration (*) must occur. If the top of T_{A_2} is a_2, then a_3 and all of its descendents in T_{A_2} are not seen yet. If the top of T_{A_1} is a_1, then a_3 and all of its descendents in T_{A_1} have been discarded. Some solution is missed. Hence, the assumption is not valid.

Case 3. Assume that $a_2 < a_1 < a_3$. The configuration (*) must occur. Denote the set of nodes that are a left and right child of some node A_L and A_R, respectively. Among a_2 and its descendents, a_2 must be the last node in A_L. Otherwise, at least one solution will be missing. Similarly, a_3 must be the first node in A_R among a_3 and its descendents. Without loss of generality, assume that $a_4 < a_5$ and $a_6 < a_7$. There must not be useful descendent nodes of a_2 in the rest of T_{A_1} since $p_3 = a_3$. Therefore a_{11} is the only node which can follow a_1. Similarly, a_{12} is the only node which can precede a_1.

The possible orderings are $a_{11} < a_1 < a_{12}$, $a_{11} < a_{12} < a_1$ and $a_1 < a_{11} < a_{12}$.

Similarly, we argue about the position of a_2. The possible orderings are $a_9 < a_2 < a_{10}$, $a_9 < a_{10} < a_2$ and $a_2 < a_9 < a_{10}$. Since a_2 is the last A_L node among a_2 and its descendents, and a_{10} is in A_L, $a_9 < a_{10} < a_2$ is the only possible ordering.

However, this ordering causes the evaluation of some twig queries suboptimal. Consider a similar complete binary tree with all B-nodes except for a_{10}, a_2 and a_5. And the query is $//A/A$. We have $T_{A_1} = T_{A_2} = [a_{10}, a_2, a_5]$ and initially, $p_1 = a_{10}$ and $p_2 = a_{10}$. p_2 cannot be advanced until p_1 is advanced to a_5. That is, a_2 is discarded. Thus, the assumption is wrong. We conclude that it is not possible to have an ordering of nodes for optimal evaluation for all twig queries. □

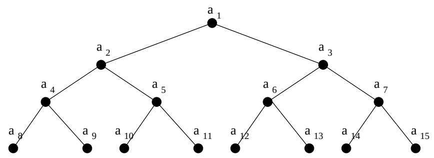

Figure. 3. A complete binary tree with 15 A nodes.

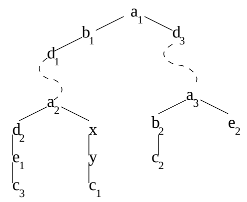

Figure. 4. Example used in Lemma 1 in Sec. 4.

Proposition 2. *Given the assumptions in Section 1.1, some twig queries are not computable with bounded memory for all possible document streams.*

Proof. The twig query //A (/A, /A) is reduced to a SPJ query over streams of tuples. Assume a node is represented by a 3-ary tuple: <*preorder*, *postorder*, *depth*>. The corresponding SPJ query is $\pi_L(\sigma_P(T_{A_1} \times T_{A_2} \times T_{A_3}))$, where the predicate L is all the attributes of the three streams and P is $T_{A_1}.preorder < T_{A_2}.preorder \wedge T_{A_1}.postorder > T_{A_2}.postorder \wedge T_{A_2}.depth = T_{A_1}.depth + 1$ $T_{A_1}.preorder < T_{A_3}.preorder \wedge T_{A_1}.postorder > T_{A_3}.postorder \wedge T_{A_3}.depth = T_{A_1}.depth + 1 \wedge T_{A_2}.preorder \neq T_{A_3}.preorder$. We claim that the twig query is bounded memory computable if and only if the SPJ query is bounded memory computable. Given arbitrary node streams, all the preorder, postorder and depth attributes are not bounded [2]. Hence, there does not exist a constant M such that the evaluation of the SPJ query over all possible stream instances requires less than M. Therefore, the SPJ query is not memory bounded computable. □

3 Offline Sorting

In Theorem 1, we show that the data streaming model is too restrictive for optimal holistic evaluation. We consider a realistic variation in this section. We assume that a powerful server is available managing the XML documents. The client sends the twig query to the server, the server analyzes the query and sends the node streams to the client and the client evaluates the query.

Since we demonstrated that the suboptimality is caused by the structurally recursive labels, we assume that recursions are "removed" by offline sorting on the server side. We illustrate the idea by the following example. Consider the query $Q = //A\ (/B, /C)$. Now we sort the nodes by (1) the preorder number of a node's first A ancestor [3] and (2) the preorder number of a node. The streams become $[a_1, a_2]$, $[b_1, b_2]$ and $[c_2, c_1]$. Note that the subtree rooted at the a_1 node is read before that at the a_2 node. Locating the useful nodes of Q becomes straightforward. However, one can find a twig query which cannot be evaluated optimally by using this ordering. The natural question is to ask the number of necessary orderings for answering all twig queries optimally.

[3] 0 is assigned to the nodes that do not have an A ancestor.

We first define the smallest twig queries that may lead to the suboptimality of the `TwigStack` algorithm. We will use the queries to show the lower bound of the number of necessary orderings.

Definition 4. *A twig query is a* simple child-edge query *if descendent edges never follow child edges in the paths of the twig query.*

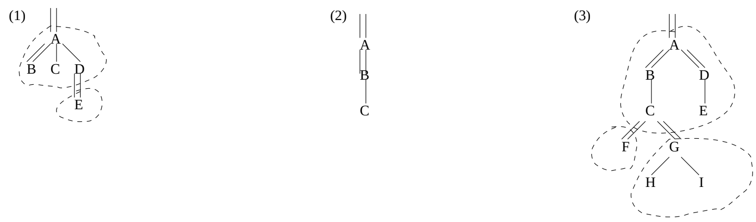

Figure. 5. (1) a twig query which is not a simple child-edge query; (2) a simple child-edge twig query; and (3) a twig query is a tree of simple child-edge twig queries connected by descendent edges.

A twig query is a set of simple child-edge queries in which queries are connected by descendent edges. (See Fig. 5.)

Theorem 2. *Let m be the number of structurally recursive labels in the DTD, d be the depth of the document and n is the number of nodes in the document. $\Omega(m^{min(d,m)} \times n)$ disk space is required to store the orderings of nodes for answering all twig queries optimally.*

Proof. Suppose a simple child-edge twig query Q (see Fig. 5 (1)) with the root A, where A is a structurally recursive label. We sort the nodes by two sort keys: (1) the preorder number of its first A-ancestor and (2) its preorder number. The visualization of this sorting is shown in Fig. 6. For each structurally recursive label X, one needs to spend n storing the nodes in which X-recursions are removed. Suppose m labels are structurally recursive, $m \times n$ space is required. These orderings produced by the sorting are all necessary because if one of these orderings is missing, then we can use the proof of Theorem 1 to show that optimality is not possible.

A twig query is a tree of simple child-edge queries. Hence, after sorting the nodes by X, we need to remove other recursions in each X subtree. The total number of orderings is $m^{min(d,m)}$. Therefore, if we store the nodes by $\Omega(m^{min(d,m)} \times n)$, optimal holistic algorithms become possible. □

3.1 A Restricted Case

In practice, the number of orderings of the nodes in a document is significantly smaller than the one in the worst case. This is due to the fact that structurally recursive labels are not always mutually recursive. Suppose some constraints on the XML document is given, there are some guarantees on the number of necessary orderings. The trivial case is that when the documents conform to a non-recursive DTD, the `TwigStack` algorithm returns all solution optimally. We also identify a restricted case in which the exponential blowup in the number of orderings does not occur.

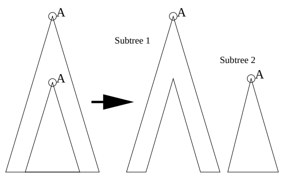

Figure. 6. Visualization of sorting nodes by (1) the preorder # of their first A ancestor and (2) their preorder #.

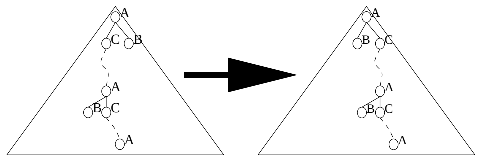

Figure. 7. The transformation described in the proof of Proposition 3

Definition 5. *A label l is* linearly recursive *if and only if $l \to \alpha$, where α is the content model of l in the DTD, and there is a single occurrence of l in α directly or indirectly. A label l is* non-linearly recursive *if and only if $l \to \alpha$ and there is more than one occurrences of l in α directly or indirectly.*

Proposition 3. *There is an ordering of nodes such that all twig queries on the documents conforming to a linearly recursive DTD can be evaluated asymptotically optimally.*

Proof. By definition, if the label of a node (say A) is linearly recursive, there is at most one child node which is an ancestor of its A-descendents. We cite a property of the `TwigStack` algorithm [5]: it guarantees that the ancestor nodes of a node n that uses a partial solution rooted at a are returned before a is returned by the core iterative procedure of the `TwigStack` algorithm. Suppose a_i is an immediate A-ancestor of a_j. Denote the descendents of a node n as $\texttt{desc}(n)$. If the document is transformed such that the nodes in $\texttt{desc}(a_i)$ - $\texttt{desc}(a_j)$ precedes the nodes in $\texttt{desc}(a_j)$ (*), then the `TwigStack` algorithm returns all the solution due to the property. Consider a transformation as such: for each X-node, the child node that is an ancestor of some X-nodes is placed following other child nodes. The transformation is illustrated by the example in Fig. 7. Then we assign the numbers to the transformed document and sort the nodes by their preorder number. The transformed document always satisfies (*). □

A survey on real world DTDs [7] shows that non-recursive DTDs and linearly recursive DTDs are few in practice. In general, the optimal holistic evaluation for all twig queries requires at least an exponential number of orderings.

4 Multiple Scans

The second attempt on supporting more twig queries optimally is to allow multiple scans on the streams. We show that it is possible to return the solution by using the `TwigStack` algorithm repeatedly. However, the lower bound of the number of scan is rather high $\Omega(d^{\,t})$, where d and t are the depth of the document and the number of simple child-edge queries in the twig query, respectively.

Lemma 1. *The number of scans required by the stream evaluation of a simple child-edge query is $\Omega(d)$, where d is the depth of the document.*

Proof. We call an A subtree with k A ancestors an $(A,\, k)$-subtree. Assume the document contains an $(A,\, k)$-subtree and only m scans are allowed, where $m < k$. There must be a scan that processes some $(A,\, i)$-subtrees and some $(A,\, j)$-subtrees, where $i > j$. We argue that one scan is not sufficient to find all solution of these subtrees.

Consider the document shown in Fig. 4. Assume the a_1-subtree is an $(A,\, i)$-subtree and a_2-subtree and a_3-subtree are $(A,\, j)$-subtrees, where $i < j$ and a_1 is an ancestor of a_2 and a_3. Without loss of generality, assume that $a_2 < a_3$. By using Proposition 1, $\texttt{desc}(a_2)$ precedes $\texttt{desc}(a_3)$. Denote N to be nodes in a_1-subtree but not in a_2- and a_3-subtrees. (1) Nodes in N cannot follow $\texttt{desc}(a_2)$ and $\texttt{desc}(a_3)$ since given the query $//A//B//C$, either $(a_1,\, b_1,\, c_1)$ (and c_3) or $(a_3,\, b_2,\, c_2)$ is missed by using a scan. (2) Nodes in N cannot precede $\texttt{desc}(a_2)$ and $\texttt{desc}(a_3)$ since given the query $//A//D//E$, either $(a_1,\, d_3,\, e_2)$ or $(a_2,\, d_2,\, e_1)$ is missed by using a scan. (3) Nodes in N cannot be in between $\texttt{desc}(a_2)$ and $\texttt{desc}(a_3)$ since given the query $//A//D//C$, c_3 must precede c_1 since d_2 precedes d_1. Consider another document in which x and y are swapped with d_2 and e_1, respectively. Since c_3 precedes c_1 in the C-stream and $(a_1,\, d_1,\, c_3)$ is missed.

Since a document with the depth d contains at most $(A,\, d)$-subtrees, the lower bound of the number of scans for simple child-edge query is $\Omega(d)$. □

Theorem 3. *Given the assumptions in Section 1.1, except that multiple scans is allowed, the solution of a twig query is returned by using $\Omega(d^{\,t})$ scans on the data streams, where d is the depth of the document and t is the minimal number of simple child-edge queries in the twig query.*

Proof. Consider a twig query containing t simple child-edge queries, q_1, q_2, ..., q_t, with their root r_1, r_2, ..., r_t, respectively. We use a $t \times 1$ vector $(v_1,\, v_2,\, \ldots\, v_t)$ to denote the subtrees that a scan is processing. For example, consider the query $//A\;/B\;//C\;/D$. q_1 and q_2 are $A\;/B$ and $C\;/D$ and r_1 and r_2 are A and C, respectively. $V = (1,\, 1)$ means that the scan is processing the $(A,\, 1)$-subtrees and $(C,\, 1)$-subtrees. The number of possible values of V is $O(d^{\,2})$ although the scan is useless when $v_1 + v_2 > d$. In general, the number of possible values of V or the twig query containing t simple child-edge queries is in $O(d^{\,t})$.

Suppose the $(k_1, k_2, \ldots, k_t)$-th scan is not performed, there is a scan processing some $(r_i,\, k_i)$-subtrees and some $(r_i,\, k)$-subtrees for some k in one scan. By the same argument used in the proof of Lemma 1, we can construct a case such that some of the solution is not reported. Thus, all $O(d^{\,t})$ scans are necessary. $\square$

5 Conclusions

We studied processing twig queries – the core operation for XML query processing – over streams of XML documents. We showed that it is not possible to develop an asymptotically optimal holistic twig join algorithm in the context of data streaming. We also show that the computation of twig queries is not memory-bounded. These negative results indicate that the data streaming model is too restrictive for twig query processing. We locate that the cause of the suboptimality of the holistic evaluation of twig queries is the structurally recursive labels in the document. Two alternative computation models for twig query processing are presented: (1) offline sorting is allowed and the algorithm is allowed selecting the correct ordering of nodes to be streamed and (2) multiple scans is allowed. We show high lower bounds for the two models.

Acknowledgements

The work of Byron Choi was done while he was visiting HKUST. Byron Choi and Derick Wood were supported under a Research Grants Council Earmarked Research Grant. The authors owe Susan Davidson, Wilfred Ng and Jerome Simeon a large debt of gratitude for insightful discussions.

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, Los Altos, USA, 1999.
2. A. Arasu, B. Babcock, S. Babu, J. McAlister, and J. Widom. Characterizing Memory Requirements for Queries over Continuous Data Streams. In *PODS*, pages 221–232, Jun. 2002.
3. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *PODS*, pages 1–16, Jun. 2002.
4. N. Bruno, N. Koudas, and D. Srivastava. Holistic Twig Joins: Optimal XML Pattern Matching. In *SIGMOD*, pages 310–321, Jun. 2002.
5. N. Bruno, N. Koudas, and D. Srivastava. Holistic Twig Joins: Optimal XML Pattern Matching. Technical Report. Columbia University, 2002.
6. S.-Y. Chien, Z. Vagena, D. Zhang, and V. J. Tsotras. Efficient Structural Joins on Indexed XML Documents. In *ICDE*, pages 141–154, Feb. 2002.
7. B. Choi. What Are Real DTDs Like. In *WebDB*, pages 43–48, Jun. 2002.
8. M. L. Lee, B. C. Chua, W. Hsu, and K.-L. Tan. Efficient Evaluation of Multiple Queries on Streaming XML Data. In *CIKM*, pages 118–125, Nov. 2002.
9. W. Wang, H. Jiang, H. Lu, and J. X. Yu. Containment Join Size Estimation: Models and Methods. In *SIGMOD*, Jun. 2003.
10. C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman. On Supporting Containment Queries in Relational Database Management Systems. *ACM SIGMOD Record*, 30(2):425–436, 2001.