Searching Multi-Hierarchical XML Documents: the Case of Fragmentation ⋆

Alex Dekhtyar, Ionut E. Iacob, Srikanth Methuku

Department of Computer Science, University of Kentucky,
dekhtyar@cs.uky.edu; eiaco0@cs.uky.edu; smeth2@uky.edu

Abstract. To properly encode properties of textual documents using XML, multiple markup hierarchies must be used, often leading to conflicting markup in encodings. Text Encoding Initiative (TEI) Guidelines[1] recognize this problem and suggest a number of ways to incorporate multiple hierarchies in a single well-formed XML document. In this paper, we present a framework for processing XPath queries over multi-hierarchical XML documents represented using fragmentation, one of the TEI-suggested techniques. We define the semantics of XPath over DOM trees of fragmented XML, extend the path expression language to cover overlap in markup, and describe FragXPath, our implementation of the proposed XPath semantics over fragmented markup.

1 Introduction

XML documents are required, by definition, to be well-formed. At the same time, it has been known for some time that text has a multi-hierarchical structure [2]. Features from different hierarchies can have *overlapping scopes*. Two key markup hierarchies for encoding text, physical text organization (pages, lines) and chapter-paragraph-sentence-word structure will produce overlapping markup any time a word is split into two lines, a sentence starts in the middle of one line and ends in the middle of another, or a paragraph starts on one page and ends on the next. Use of additional feature hierarchies, only accentuates the problem. Many prominent examples arise from image-based encodings of manuscripts (Figure 1), where, in addition to folio-line and sentence-word structures, we are also interested in encoding manuscript condition (damages), visibility of text under different lighting conditions, and paleographic information (e.g., which scribe wrote which portions of the manuscript).

The significance of multihierarchical document-centric markup and its proper management has been recognized by the TEI community fairly early [2, 1]. Two problems need to be recognized and addressed: (a) storage and representation and (b) querying and retrieval. TEI Guidelines (P4)[1] propose a number of solutions to the first problem. Among them is markup fragmentation, a technique that breaks the overlapping conflicts by fragmenting one of the conflicting XML elements to the degree that allows proper nesting. Fragmentation allows to represent multihierarchical markup in a

⋆ The work of the first author has been supported in part by NSF grant ITR-0219924 and the NSF grant ITR-0325063. Second author's work has been supported in part by NEH grant RZ-20887-02.

<line no="22"> hu þu me hæfst
afrefredne æg </line>
<line no="23"> þer ge mid
þinre smealican spræ</line>
<line no="24"> ce, ge mid þinre
wynsum nesse þines </line>
(i)

<w>hu</w> <w>þu</w> <w>me</w>
<w>hæfst</w> <w>afrefredne</w>
<w>ægþer</w> <w>ge</w> <w>mid</w>
<w>þinre</w> <w>smealican</w>
<w>spræce</w>, <w>ge</w> <w>mid</w>
<w>þinre</w> <w>wynsum nesse</w>
<w>þines</w>
(ii)

hu <res><restxt>þu m</restxt></res>e hæfst
afrefredne ægþ<dmg>er</dmg> ge <dmg>mid
</dmg> þinre smealican spr<dmg>æ</dmg>ce,
<dmg>g</dmg>e mid
<dmg>þ</dmg>in<dmg>r</dmg>e wynsum
nesse þin<dmg>e</dmg><res><restxt>s
</restxt></res>
(iii)

Fig. 1. A fragment of King Alfred's Boethius manuscript folio [3], the corresponding text, and different XML encodings.

single XML document. However, this comes at a price. Fragmented XML documents are no longer easy to query using traditional XML query languages such as XPath. In fact, certain queries, easily expressible in XPath over regular XML documents, cannot be expressed in XPath over fragmented XML. While fragmentation is used by a large number of humanities scholars to represent overlapping markup in their encodings, there is nor available formalism, neither appropriate software for querying such encodings in a convenient, consistent, and domain-independent manner.

In this paper, we resolve this problem by providing the semantics of the XPath queries over fragmented XML documents. Because XPath is not expressive enough for querying multihierarchical markup, we enhance it with new features, which, in particular, capture overlapping content for elements from different hierarchies. Our contributions are summarized as follows: (i) we formally define multiple hierarchies for XML documents with markup fragmentation (Section 3); (ii) we give new semantics for computing XPath axes for fragmented XML documents with multiple hierarchies (Section 4); (iii) we propose and implement efficient algorithms for computing XPath axes for XML documents with fragmentation; (iv) we present some preliminary experimental results (Section 5).

2 Overlapping Markup in Text Encoding

Overlapping markup occurs in a large number of text encoding tasks. Figure 1 shows a fragment of a tenth century Old English manuscript [3] and the encodings of this fragment in three different markup hierarchies: physical location, sentence structure and condition. Features from these three hierarchies overlap: `<rstxt>` (restored text) overlaps `<w>` in line 22; the word *ægþer* is split between lines 22 and 23 and the word *spræce* is split between lines 23 and 24.

TEI Guidelines (P4)[1] suggest a number of ways for representing multihierarchical markup in a single document. In this paper, we consider one such solution, fragmentation, which works as follows. Whenever the scope of two elements overlaps, one of the elements is broken into parts in a way that allows the inclusion of both elements in the same XML document while preserving proper nesting. For example, one can resolve the conflict between `<w>` and `<line no="23">` elements as shown in Figure 2 Here, the original element `<w>spræce</w>` had been split into two parts: `<w id="W1" next="W2">spræ</w>` and `<w id="W2" prev="W1">ce</w>`. The first fragment nests properly inside the `<line no="23">` element, while the second - inside the `<line no="24">` element. The fact, that these two are the fragments of a single word, rather than two separate words is facilitated by the use of `id`, `prev` and `next` attributes for `<w>` which form a double-linked list of fragments. Markup fragmentation is a simple way of combining conflicting markup in a single XML document. However, [1] leaves open the question of querying the data stored in fragmented form.

```
<line no="23">þer ge mid þinre smeali-
can
<w id="W1" next="W2">spræ</w></line>
<line no="24"><w id="W2"
prev="W1">ce</w>, ge mid þinre
     wynsum nesse þines </line>
```

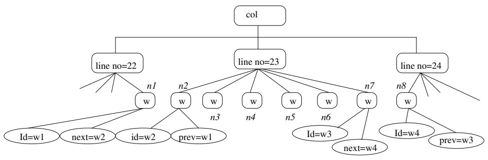

Fig. 2. Fragmentation and the DOM tree for fragmented markup.

Example 1. Figure 2 shows a (part of the) DOM tree of the document that uses fragmentation to include `<line>` and `<word>` elements. Consider the following query: *Find all words that are located completely in line 23.*

For a non-fragmented XML document, we can convert this request into XPath: `/descendant::line[@no="23"]/descendant::w`.

When applied to the DOM tree in Figure 2, this expression evaluates to the nodeset $\{n2, n3, n4, n5, n6, n7\}$. However, this is **not** the right answer to the original information request — nodes $n2$ and $n7$ represent fragments of words, not complete words, in line 23. Thus, we need to reformulate the query. The new version is: *Find all* `<w>` *elements inside the scope of the* `<line no="23">` *element, such that they are either not fragmented, or all their peer fragments are inside the scope of the* `<line no="23">` *element.*

This information request cannot be expressed as a single XPath 1.0 query. It states that whether or not a node is included in the answer set is dependent on whether or not other nodes are included in the answer set (in fact, evaluation of this request is equivalent to building a transitive closure for each `<w>` node by following the `prev` and `next` links). At the same time, in XPath 1.0, decision on whether to include a node in the answer set is made independent of decisions for other nodes.

We note, however, that while expressing the query above in XPath 1.0 is impossible, there is a simple and straightforward procedure for producing the desired result: search the DOM subtree rooted at `<line no="23">`, and include in the answer set each non-fragmented `<w>` node and all fragmented `<w>` nodes that form a single word. The latter can be determined during a single tree traversal by verifying that once a `<w ID="x" next="y">` element (first fragment) is discovered, the matching `<w ID ="z" prev="u">` element (last fragment) is also in the scope of `<line no="23">`.

The example above suggests that the problem is in the fact that the semantics of XPath 1.0 over DOM is incompatible with the semantics of DOM trees for fragmented XML. In the rest of this paper, we show how this can be rectified.

3 Background

We start this section by briefly describing the Document Object Model and the XPath query language for XML together with some notation we use in the rest of the paper.

In Document Object Model (DOM) [4], an XML document is represented as a labeled, unranked tree. We denote by $dom(d)$ the set of nodes in the DOM of d, by $root(d) \in dom(d)$ the root node of d, by $tags(d)$ the set of node labels (tags) in d. In this paper we consider only *element* and *text nodes* in $dom(d)$. We let $type(x)$ to return the type of the argument node: “element” or “text”. For a node $x \in dom(d)$ the function $tag(x)$ returns the label of x for element nodes and $null$ for text nodes. For two nodes $x, y \in dom(d)$, $x < y$ or $y > x$ denotes that x is before y in the document order[5]. We denote by $ancestor_{DOM}(x)$ the set of *ancestor* nodes of x in DOM. For an element node x we use $scope(x)$ to denote the document content interval from the *start tag* of x to the *end tag* of x.

XPath is a language for addressing parts of an XML document [6]. XPath is used as the means of accessing XML documents in XQuery. It can be used to query XML documents by itself. XPath uses a tree of nodes model to represent an XML document. The main syntactical construction of XPath is *expression* and the nodes of a document are located using the *location path* (a special kind of *expression*). A *location path* is composed of one or more *steps*, at each step a set of nodes is selected based on their relationship (specified in *step*) to each node in a current set of *context nodes*. The node set result of a *step* evaluation is the current set of *context nodes* for the next *step* in the *location path*. The core syntax of XPath can be summarized as follows:

```
locationPath := step1/step2/.../stepn
step := axis::node-test predicate*
predicate := [expression]
```

The main syntactical construction for a *step* evaluation is *axis*. XPath uses 13 *axes* to address nodes in a document: *ancestor, ancestor-or-self, attribute, child, descendant, descendant-or-self, following, following-sibling, namespace, parent, preceding, preceding-sibling,* and *self*. Formal semantics of XPath axes is given in[6]. The set of nodes from *axis* evaluation is filtered by the *node-test* (basically a node type test or a name test for *element* nodes) and by the *expression*, which is either an location path (evaluated to *true* if the result node set is not empty), or a boolean expression involving functions from the core function library of XPath[6].

3.1 Multiple Hierarchies for XML documents with fragmentation

We define a multiple hierarchy over an XML document as a mapping of node names (tags, or elements) onto a finite set of labels (hierarchy names). In any multi-hierarchical document, an element node belongs to a single hierarchy (except for the root node, which belongs to all hierarchies) whereas any text node belongs to all hierarchies. Usually, each hierarchy encompasses a specific set of markup features (e.g., in Figure 1 the three hierarchies are physical position of text, sentence structure and manuscript condition). The key syntactic condition is that document markup restricted to any single hierarchy is *well-formed*. We formally define hierarchies and fragmented XML representations as follows.

Definition 1 (muti hierarchies). *Let H be a set of labels (strings). A multi-hierarchy is a function $\mathcal{H} : tags(d) \rightarrow 2^H$ so that*
(a) if x is the root node or $type(x) =$ "$text$" then $\mathcal{H}(x) = H$
(b) if x is an element node, not root node, then $\mathcal{H}(x) = \{a\}$ for some $a \in H$.

Definition 2 (fragmented representation). *Let $\mathcal{H}$ be a multi-hierarchy over the set of labels H. Let $d_1, .., d_{|H|}$ be XML documents encoding the same content string S, having the same label for the root node, and with markup from different hierarchies. An XML document d is called a* fragmented XML representation *of $d_1, ..., d_{|H|}$ iff (a) d is well-formed; (b) for each node $x \in d_j$ there exists a set of nodes $\{x_1, \ldots, x_k\}$ all with the same label as x such that $scope(x) = \cup_{1 \leq i \leq k} scope(x_i)$; (c) for any attribute $prev$ or $next$ there exists a unique id attribute with the same value; no id attribute value can appear in two $next$ or two $prev$ attributes.*

Fragmentation allows to store multi-hierarchical markup in a single well-formed document by breaking elements into fragments. Fragments represent the semantics of the original encoding only when combined. For example, `<w>spræce</w>` is broken into two fragments, `<w id="W1" next="W2"> spræ</w>` and `<w id="W2" prev="W1">ce</w>`, but if we are interested in recovering the full word, we must join these two fragments together. Functions $fragments()$ and LMU (Logical Markup Unit), defined below, recover the actual range of the markup corresponding to a given document node by collecting all fragments "related" to the node and constructing the appropriate content respectively.

Definition 3 (fragment). *$fragments : dom(d) \rightarrow 2^{dom(d)}$ is defined recursively: (i) $x \in frag - ments(x)$; (ii) if $y \in dom(d)$ and there exists $z \in fragments(x)$ such that $prev$ or $next$ attribute of y has the same value as the id attribute of z, then $y \in fragments(x)$.*

Basically, the fragments are element nodes, in a double linked list (using *prev*, *next*, and *id* attribute values), which are covering a continuous range of document content.

Definition 4 (logical markup unit). *The* logical markup unit *(LMU) of the markup corresponding to a node x is the set of all text nodes covered by markup corresponding to each node in $fragments(x)$:*

$$LMU(x) := \{t \in descendant_{DOM}(x) \mid type(t) = \text{“}text\text{”}\}.$$

Recall that fragmentation is a workaround for representing overlapping markup in a single well-formed XML document. Markup conflicts are not immediately visible within the fragmented XML document, but using LMUs, we can "discover" them: markup are in conflict if their corresponding LMUs overlap.

We slightly abuse notation and use $\mathcal{H}(x)$ in lieu of $\mathcal{H}(tag(x))$ to denote the hierarchy of node x. We say that two document nodes $x, y \in dom(d)$ are in the same hierarchy, denoted $\mathcal{H}(x) \cong \mathcal{H}(y)$ if $\mathcal{H}(x) \subseteq \mathcal{H}(y)$ or $\mathcal{H}(x) \supseteq \mathcal{H}(y)$. We use $dom_{\mathcal{H}(x)} := \{y \in dom(d) \mid \mathcal{H}(y) \cong \mathcal{H}(x)\}$ to denote the nodes in $dom(d)$ in the same hierarchy as x. It is clear now that the root node and any text node are in the same hierarchy with any other node in a document. The intuition behind this approach is simple: the root node or a text node have no overlapping range with any other logical markup unit.

4 XPath queries over multi-hierarchical XML documents

As suggested in the examples in Section 1, a natural handling of XPath queries over multi-hierarchical XML documents with fragmentation would require each query evaluation result to be expressed as a set of logical markup units. To avoid expensive joins and, most importantly, query reformulation problems, we extend the XPath axis semantics to handle queries for multi-hierarchical XML with fragmentation.

4.1 XPath axis semantics for fragmented multi-hierarchical XML

The purpose of the new semantics for XPath over fragmented documents is two-fold: (a) restore the proper meaning of XPath axes and (b) extend the expressive power to capture new relationships between nodes from different hierarchies. To achieve (a), we define the semantics of *self, child, parent, descendant, descendant-or-self, ancestor, ancestor-or-self, following-sibling, preceding-sibling, following*, and *preceding* axes over fragmented XML. For goal (b), we define new axes, specific to multi-hierarchical XML documents: *xdescendant, xancestor, preceding-overlapping, following-overlapping*, and *overlapping*. For each Extended XPath axis $\mathcal{X}$, we define the corresponding evaluation function $\mathcal{X} : dom(d) \rightarrow 2^{dom(d)}$, where $\mathcal{X}(x)$ evaluates axis $\mathcal{X}$ for the context node x. The evaluation functions for XPath axes are defined as follows:

$$
\begin{array}{ll}
self(x) := & fragments(x) \\
child(x) := & \cup_{v \in self(x)} \{y \in dom_{\mathcal{H}(x)}(d) \mid y \in descendant_{DOM}(v) \wedge \\
 & \quad \neg(\exists z \in dom_{\mathcal{H}(x)}(d) : \; z \in descendant_{DOM}(v) \wedge \\
 & \quad z \in ancestor_{DOM}(y))\} \\
parent(x) := & \{y \in self(z) \mid z \in ancestor_{DOM}(x) \cap dom_{\mathcal{H}(x)}(d) \wedge \\
 & \quad \neg(\exists v \in dom_{\mathcal{H}(x)}(d) : \; v \in ancestor_{DOM}(x) \wedge \\
 & \quad v \in descendant_{DOM}(z))\} \\
descendant(x) := & \cup_{v \in self(x)} \{y \in dom_{\mathcal{H}(x)}(d) \mid y \in descendant_{DOM}(v)\} \\
descendant{-}or{-}self(x) := & self(x) \cup descendant(x) \\
ancestor(x) := & \{y \in self(z) \mid z \in ancestor_{DOM}(x) \cap dom_{\mathcal{H}(x)}(d)\} \\
ancestor{-}or{-}self(x) := & self(x) \cup ancestor(x) \\
following{-}sibling(x) := & \{y \in dom_{\mathcal{H}(x)}(d) \mid y > x \wedge y \notin self(x) \wedge \\
 & \quad parent(y) = parent(x)\} \\
preceding{-}sibling(x) := & \{y \in dom_{\mathcal{H}(x)}(d) \mid y < x \wedge y \notin self(x) \wedge \\
 & \quad parent(y) = parent(x)\} \\
following(x) := & \{y \in dom_{\mathcal{H}(x)}(d) \mid y > x \wedge y \notin self(x) \wedge \\
 & \quad y \notin descendant(x)\} \\
preceding(x) := & \{y \in dom_{\mathcal{H}(x)}(d) \mid y < x \wedge y \notin self(x) \wedge \\
 & \quad y \notin ancestor(x)\}
\end{array}
$$

The extended XPath axes are defined below:

$$
\begin{array}{ll}
xdescendant(x) & := \{y \in dom(d) - dom_{\mathcal{H}(x)}(d) \mid \forall t \in descendant(y), \\
 & \quad type(t) = \text{“}text\text{”}(t \in descendant(x))\} \\
xancestor(x) & := \{y \in dom(d) - dom_{\mathcal{H}(x)}(d) \mid \forall t \in descendant(x), \\
 & \quad type(t) = \text{“}text\text{”}(t \in descendant(y))\} \\
following{-}overlapping(x) & := \{y \in dom(d) \mid \exists t \in descendant(x)(type(t) = \text{“}text\text{”} \wedge \\
 & \quad t \in descendant(y)) \wedge \exists t \in descendant(x) \\
 & \quad \forall z \in descendant(y)(type(t) = \text{“}text\text{”} \wedge \\
 & \quad t < z) \wedge \exists t \in descendant(y) \forall z \in descendant(x) \\
 & \quad (type(t) = \text{“}text\text{”} \wedge z < t)\} \\
preceding{-}overlapping(x) & := \{y \in dom(d) \mid \exists t \in descendant(x)(type(t) = \text{“}text\text{”} \wedge \\
 & \quad t \in descendant(y)) \wedge \exists t \in descendant(x) \\
 & \quad \forall z \in descendant(y)(type(t) = \text{“}text\text{”} \wedge t > z) \wedge \\
 & \quad \exists t \in descendant(y) \forall z \in descendant(x) \\
 & \quad (type(t) = \text{“}text\text{”} \wedge z > t)\} \\
overlapping(\{x\}) & := preceding{-}overlapping(\{x\}) \cup \\
 & \quad following{-}overlapping(\{x\})
\end{array}
$$

The following theorems establish the basic properties of the Extended XPath over fragmented XML: (a) all fragments are included in the result of evaluation .and (b) with only one hierarchy present, our definitions are equivalent to those in [6] [1].

Theorem 1. *[7] Let $\mathcal{X}$ be an XPath axis evaluation function for multi-hierarchical XML. Let $x \in dom(d)$ and let $y \in \mathcal{X}(x)$. Then for any $z \in self(y)$, $z \in \mathcal{X}(x)$.*

Theorem 2. *[7] For any XML document with a single markup hierarchy, $|H| = 1$, for any axis $\mathcal{X}$ defined for both XPath and Extended XPath, the evaluation of $\mathcal{X}$ using XPath semantics yields the same results as the evaluation of $\mathcal{X}$ using Extended XPath semantics, for any node in d.*

4.2 Searching XML documents using Extended XPath

Let us return to the query from Example 1: *Find all words that are located completely in line 23*. We consider the set of hierarchies shown in Figure 1: “location” (box (i)), “structure” (box (ii)) and “condition” (box (iii)). Because `<line>` and `<w>` are in different hierarchies and we want words completely inside lines (relationship represented by the `xdescendant` axis), the correct Extended XPath expression is: `/descendant::line[@no="23"]/xdescendant::w`. According to the Extended XPath specifications, the query is evaluated to the node set: $\{n3, n4, n5, n6\}$ (see the DOM tree in Figure 2).

Consider now the following query for the same document: *Find all words that are located partially in line 23.* This query concerns markup overlap. The corresponding Extended XPath query is `/descendant::line[@no="23"]/ overlapping::w`. It evaluates to the node set: $\{n1, n2, n7, n8\}$. To retrieve all words that

[1] The latter result is important from the practical point of view: an Extended XPath processor yields correct results when evaluating XPath expressions for any XML document with a single markup hierarchy.

Algorithm 1: Extended XPath axis evaluation
Input: $N, dom(d)$
Output: N'
$\mathcal{X}$-EVALUATION(N)
(1) $N' = \emptyset$
(2) **foreach** $n \in N$
(3) $\quad N' \leftarrow N' \cup \mathcal{X}(self(n), dom(d))$
(4) **return** N'

Algorithm 2: `xancestor`
XANCESTOR($n_1, \ldots, n_f, dom(d)$)
(1) $t_1 \leftarrow FTD(n_1)$
(2) $t_2 \leftarrow LTD(n_f)$
(3) $TEMP1 \leftarrow \{x \in self(y) \mid y \in ancestor_{DOM}(t_1) - dom_{\mathcal{H}(n)}(d)\}$
(4) $TEMP2 \leftarrow \{x \in self(y) \mid y \in ancestor_{DOM}(t_2) - dom_{\mathcal{H}(n)}(d)\}$
(5) $N \leftarrow TEMP1 \cap TEMP2$
(6) **return** N

Algorithm 3: `xdescendant`
XDESCENDANT($n_1, \ldots, n_f, dom(d)$)
(1) $t_1 \leftarrow FTP(n_1)$
(2) $t_2 \leftarrow FTF(n_f)$
(3) $TEMP1 \leftarrow \{x \in self(y) \mid y \in ancestor_{DOM}(t_1) - dom_{\mathcal{H}(n)}(d)\}$
(4) $TEMP2 \leftarrow \{x \in self(y) \mid y \in ancestor_{DOM}(t_2) - dom_{\mathcal{H}(n)}(d)\}$
(5) $TEMP3 \leftarrow \cup_{x \in \{n_1, \ldots, n_f\}}\{y \in (ancestor_{DOM}(x) \cup descendant_{DOM}(x)) - dom_{\mathcal{H}(n)}(d)\}$
(6) $N \leftarrow TEMP3 - (TEMP1 \cup TEMP2)$
(7) **return** N

Algorithm 4: `following- overlapping`
FOLLOWING-OVERLAPPING($n_1, \ldots, n_f, dom(d)$)
(1) $t_1 \leftarrow FTD(n_1)$
(2) $t_2 \leftarrow LTD(n_f)$
(3) $t_3 \leftarrow FTF(n_f)$
(4) $TEMP1 \leftarrow \{x \in self(y) \mid y \in ancestor_{DOM}(t_1) - dom_{\mathcal{H}(n)}(d)\}$
(5) $TEMP2 \leftarrow \{x \in self(y) \mid y \in ancestor_{DOM}(t_2) - dom_{\mathcal{H}(n)}(d)\}$
(6) $TEMP3 \leftarrow \{x \in self(y) \mid y \in ancestor_{DOM}(t_3) - dom_{\mathcal{H}(n)}(d)\}$
(7) $N \leftarrow (TEMP2 \cap TEMP3) - TEMP1$
(8) **return** N

Fig. 3. Algorithms for evaluation of Extended XPath axes.

occur in line 23, the following Extended XPath query can be used: `/descendant::w[xancestor::line[@no="23"] or overlapping::line[@no="23"]]`.

4.3 Algorithms for XPath axis evaluation

Polynomial time evaluation algorithms for XPath queries, using DOM representation of an XML Document, are given in [8, 9]. An algorithm that evaluates XPath axes in linear time (in the size of nodes in the input XML document) is also given in [9]. Similar techniques can be used for evaluating the Extended XPath axes *child, parent, descendant, descendant-or-self, ancestor, ancestor-or-self, following-sibling, preceding-sibling, following*, and *preceding*. The only difference is a node filtering operation, that is, selecting only nodes in a given hierarchy. This can be easily implemented using a *hash* function, so the overall evaluation is still linear. We also point out that evaluation of *self* can be directly carried out using the *id reference* (ID and IDREF) mechanism provided by XML [5] and the DOM API [4].

In Figure 3 we give the algorithms for evaluating *xancestor, xdescendant,* and *following-overlapping*. Note, that the semantics of the Extended XPath axes as described in Section 4.1 is given for *a single* DOM node. In reality, when evaluating an XPath query (see Section 3), at each location step a node set is computed and this node set is used as the the context for the next location step.

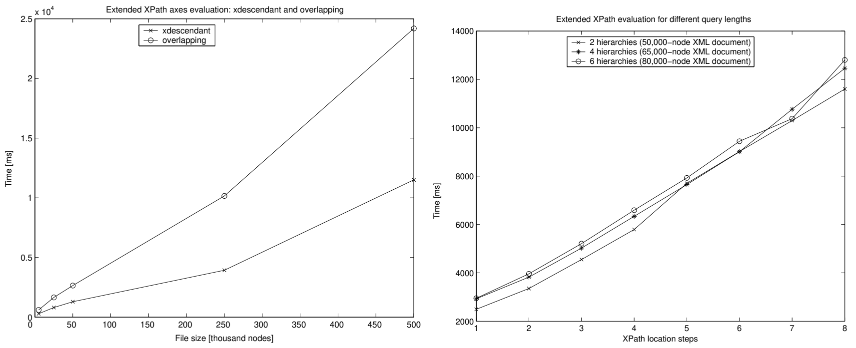

Fig. 4. Results of testing FragXPath.

5 Experiments

Using the semantics described in the paper, we have fully implemented and Extended XPath processor for fragmented XML document (*FragXPath*). *FragXPath* **is a main-memory processor**. Below we show the results of a few tests we used to test FragXPath. The tests were run on a Dell GX240 PC with 1.4Ghz Pentium 4 processor and 256 Mb main memory running Linux. We generated XML files with multple hierarchies (we use 2,4, and 6 hierarchies). In Figure 4 we report the results of two tests.

The first graph shows the results of testing `xdescendant` and `overlapping` axes, which extend traditional XPath, over fragmented XML documents with 4 hierarchies. We used fragmented XML documents with sizes ranging from 50,000 to 500,000 nodes and from 1MB to 45MB size on disk. We used the following Extented XPath queries in our experiments: `/descendant::page//overlapping::*` and `/descendant::page/xdescendant::*`.

The second graph shows the dependence of evaluation time on the number of location steps in the query (each query consisted of `/overlapping::*` location step repeated for 1,2,... 8 times) and on files with 2,4, or 6 hierarchies and 50,000, 65,000, and 80,000 nodes respectively. As expected, the graph presented in Figure 4 shows linear time complexity on the query size.

Finally, we compared the work of *FragXPath* with the work of two widely available XPath processors, *Xalan* and *Dom4j* on comparable workloads. While direct comparison of *FragXPath* to XPath processors is not possible – the expressive power of the languages is different, we should expect *FragXPath* to spend about the same time as an XPath processor searching for comparable information in the same DOM tree.

Processor/Number of steps	1	4	8
$FragXPath$	1208[ms]	1279[ms]	1284[ms]
$Xalan$	1590[ms]	1630[ms]	1632[ms]
$Dom4j$	1336[ms]	1636[ms]	1640[ms]

Table 1. XPath processors comparison.

Table 1 shows the times for all three processors for queries consisting of 1, 4, and 8 location steps of the form `/descendant-or-self::*`. *FragXPath* performs in the same time range as the XPath processors (it is actually faster, but we emphasize, the it is incorrect to say that *FragXPath* works better than *Xalan* or *Dom4j*).

6 Conclusions

Processing of XML with multiple hierarchies has attracted the attention of numerous researchers recently. In [10] we survey the state-of-the-art in the area. Jagadish et al. have considered similar poblem for data-centric XML and proposed the so-called colorful XML to implement multiple hierarchies[11]. In this paper we consider one specific legacy case, stemming from the text encoding community, when XPath semantics needs to be redefined and extended, in order to support efficient querying. Our implementation is faithful to the original XPath semantics, and is immediately applicable to processing queries over the multitude of legacy text encodings prepared using the fragmentation technique. Our current and future work is on considering other legacy cases and also on building a unified framework for processing multihierarchical XML.

References

1. Sperberg-McQueen, C.M., Burnard, L., (Eds.): Guidelines for Text Encoding and Interchange (P4). http://www.tei-c.org/P4X/index.html (2001) The TEI Consortium.
2. Renear, A., Mylonas, E., Durand, D.: Refining our notion of what text really is: The prob lem of overlapping hierarchies. Research in Humanities Computing (1993) (Editors: N. Ide and S. Hockey).
3. Boethius, A.M.S.: Consolation of philosophy. (Alfred The Great (translator), British Library MS Cotton Otho A. vi) Manuscript, folio 36v.
4. Champion, M., Byrne, S., Nicol, G., Wood, L., (Eds.): Document Object Model (DOM) Level 1 Specification. http://www.w3.org/TR/REC-DOM-Level-1/ (1998) World Wide Web Consortium Recommendation, REC-DOM-Level-1-19981001.
5. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F., Cowan, J., (Eds.): Extensible Markup Language (XML) 1.1. http://www.w3.org/TR/2004/REC-xml11-20040204 (2004) W3C Recommendation 04 February 2004.
6. Clark, J., DeRose, S.: XML Path Language (XPath) (Version 1.0). http://www.w3.org/TR/xpath (1999) W3C, REC-xpath-19991116.
7. Dekhtyar, A., Iacob, I.E., Methuku, S.: Searching Multi-Hierarchical XML Documents: the Case of Fragmentation. Technical Report TR 439-05, University of Kentucky, Department of Computer Science (2005) http://www.cs.uky.edu/∼dekhtyar/publications/TR439-05.ps.
8. Gottlob, G., Koch, C., Pichler, R.: XPath query evaluation: Improving time and space eficiency. In: Proceedings of the ICDE, Bangalore, India. (2003) 379–390
9. Gottlob, G., Koch, C., Pichler, R.: Efficient algorithms for processing XPath queries. In: Proc. of VLDB, Hong Kong (2002)
10. Dekhtyar, A., Iacob, I.E.: A Framework for Management of Concurrent XML Markup. Data and Knowledge Engineering **52** (2005) 185–208
11. Jagadish, H.V., Lakshmanan, L.V.S., Scannapieco, M., Srivastava, D., Wiwatwattana, N.: Colorful XML: One Hierarchy Isn't Enough. In: Proc., ACM SIGMOD. (2004) 251–262