Efficient XQuery Support for Stand-Off Annotation

Wouter Alink Raoul Bhoedjang
Nederlands Forensisch Instituut
Laan van Ypenburg 6, 2497 GB
The Hague, the Netherlands
{wouter,raoul}@holmes.nl

Arjen de Vries Peter Boncz
Centrum voor Wiskunde en Informatica
Kruislaan 413, 1098 SJ
Amsterdam, the Netherlands
{arjen,boncz}@cwi.nl

ABSTRACT

XML annotations are a widely occurring phenomenon in many application fields, and XML databases should be used to store and query such data. To provide intuitive and fast querying of annotations, we make a case for extending XPath with four new axis steps, that correspond with so-called StandOff joins, introduced here. The new steps can be efficiently implemented using a region index and fast loop-lifted StandOff MergeJoin algorithms. These techniques were added to the open-source XML DBMS MonetDB/XQuery, and we show in our evaluation it thus becomes capable of interactively querying >GB annotation databases.

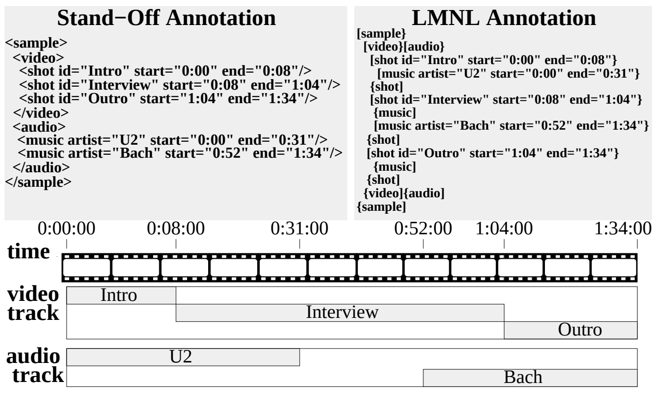
1. INTRODUCTION

One of the many uses of XML is to store and query *annotations*, such as speech-recognized text or shot boundaries detected in audio-visual streams (multimedia information retrieval), the automatically derived grammatical structure of sentences in text corpora (natural language processing, NLP), or for representing and relating the outputs of multiple file system recovery and feature detection tools, run on the raw image of a confiscated hard drive (digital forensics). In some applications we even wish to support annotations of non-contiguous areas (e.g. files reconstructed from a raw disk image may consist of multiple blocks scattered around the file system, and grammatical constructs in some natural languages may be comprised of non-adjacent words).

We should stress here that we have not invented this problem ourselves; handling multiple hierarchies using concurrent markup has for example been treated extensively in [13, Chapter 31], in the context of the scholarly study of texts. Also, the NLP community devoted a workshop to the topic of multi-dimensional markup in XML alone [1].

There have been proposals to store multiple annotation hierarchies inline in a single document (sometimes even together with the data to-be-annotated). Example are LMNL (top right of Figure 1) and GODDAG [14]. In contrast, we focus on a particular case of XML annotations, where the object being annotated is stored *separately* from the XML annotations, and regions of interest can be identified by position ("StandOff" annotations – see the top left of Figure 1). This allows annotation of non-XML objects, as well as maintaining multiple (overlapping) annotation hierarchies, that each have an easy-to-understand XML layout.

XIME-P 2006, 3rd International Workshop on XQuery Implementation, Experiences and Perspectives, June 30, Chicago, Illinois
Copyright 2006 ACM 1-59593-465-0/06/0006 ...$5.00.



```
<sample>
  <video>
   <shot id="Intro" start="0:00" end="0:08"/>
   <shot id="Interview" start="0:08" end="1:04"/>
   <shot id="Outro" start="1:04" end="1:34"/>
  </video>
  <audio>
   <music artist="U2" start="0:00" end="0:31"/>
   <music artist="Bach" start="0:52" end="1:34"/>
  </audio>
</sample>
```

```
[sample}
  [video}[audio}
   [shot id="Intro" start="0:00" end="0:08"}
     [music artist="U2" start="0:00" end="0:31"}
   {shot]
   [shot id="Interview" start="0:08" end="1:04"}
    {music]
    [music artist="Bach" start="0:52" end="1:34"}
  {shot]
  [shot id="Outro" start="1:04" end="1:34"}
    {music]
  {shot]
  {video]{audio]
{sample]
```

Figure 1: Multimedia Annotation Example

Contributions. The challenge is how a XML DBMS should store annotations and how these can be queried intuitively yet efficiently. Our contributions are as follows: (i) a flexible and configurable XML representation for storing StandOff annotations in XQuery database systems. (ii) the concept of StandOff joins to navigate between related XML annotations, and a proposal to include them as four new XPath axis steps. (iii) efficient algorithms for the integration of these algorithms in existing XQuery processors[1].

Outline. This paper is organized as follows: in Section 2 we define how StandOff annotation could be represented in a configurable way. Section 3 introduces four so-called StandOff join operators and discusses various ways to represent these in XQuery. In Section 4, we discuss efficient implementation of StandOff joins. This depends on the introduction of a region index and a region-merge algorithm that allows selection pushdown. Its *loop-lifted* nature ensures that it remains efficient (only a single linear index scan) even if the StandOff Join expressions appear nested in a `for`-loop with many iterations. A performance evaluation on the XMark benchmark shows that this implementation can interactively query >GB annotation databases. In Section 5 we discuss related work before outlining our conclusions and future research in Section 6.

[1]Our work is available in the open source XML DBMS MonetDB/XQuery, see `www.monetdb-xquery.org`

2. XML STANDOFF ANNOTATIONS

Without loss of generality, we call the object on which annotations are created the BLOB (Binary Large OBject). In the video analysis case, the BLOB corresponds with the multimedia file (e.g. mpeg2 video), whereas the BLOB is a file containing natural text (e.g. the Bible) in the natural language processing case, and in the forensic analysis case, the BLOB is the binary image (exact copy) of the confiscated hard drive.

The BLOB may have arbitrary content or structure, though we assume that sub-objects of interest in the BLOB can be identified using one or more *regions*. A region consists of a [*start,end*] range, where the start and end *positions* are from the same data-type (the region includes *start* and *end* and $start \leq end$). This data-type must support *full ordering*. Our current implementation assumes the positions to be machine-representable as 64-bits integers (this supports the use of regions that consists of file-offsets as well as time-ranges), but this is not a conceptual restriction.

Area-Annotations. We define *area-annotations* as those XML element nodes that directly contain region information. Regions can be attached to XML elements (e.g. `<foo> bar</foo>`) either by adding “start” and “end” attributes to it: `<foo start="1" end="10">bar</foo>`, or by adding one or more `<region>` child elements:

```
<foo><region>
       <start>1</start>
       <end>2</end>
     </region>
     bar
</foo>
```

An XML document may contain many such area- annotations, and the descendants of an area-annotation may again contain area-annotations. However, we impose no restrictions on such sub-annotations (thus, the region of a descendant area-annotation does *not* need to be contained in the region of its ancestors that are area-annotations).

The attribute representation for regions is more compact and less intrusive to the structure of the annotation document, while the element representation allows to attach *multiple* regions to an element (i.e. to represent non-contiguous areas). We regard the exact representation a run-time setting. The names “start” and “end” are default, but can be changed as convenient for the application. The type of annotation of interest (e.g. file-offset, word position or date/timestamp) is also highly application dependent. Therefore, our proposal is configurable to support all these types and representations, using the XQuery `declare option` syntax, part of the query preamble:

```
declare option standoff-type   "qualified-name"
declare option standoff-start  "qualified-name"
declare option standoff-end    "qualified-name"
declare option standoff-region "qualified-name"
```

The default settings are:

```
declare option standoff-type   "xs:integer"
declare option standoff-start  "start"
declare option standoff-end    "end"
```

If the `standoff-region` option is specified, the element representation of regions is used. Note that in the former case `standoff-start` and `standoff-end` define attribute names, whereas in the latter case, they define element names.

3. QUERYING STANDOFF ANNOTATIONS

In principle, two intervals (=regions) r_1 and r_2 can be in 13 different relationships with each other [4], ranging at one end of the semantic spectrum from r_1 disjunctively preceding r_2, to $r1$ disjunctively succeeding r_2 at the other end, with $r_1 = r_2$ right in the middle. These relationships play a crucial role in querying related annotations. EXPath, a proposed language for querying GODDAG markup language, (where, in contrast to StandOff annotation, all annotations are stored in interleaved form in the same document) uses 11 such relationships as query predicates [10].

The number of relevant relationships can be significantly reduced if we abstract from the particular ordering of the intervals and focus on the notions of *containment* and *overlap*. This choice is made for two reasons: first, region ordering seems to play no role in the StandOff annotation use cases we encountered [3], and second, is hard-to-define meaningfully for non-contiguous area-annotations (i.e. those that consist of multiple regions); a feature we wish to support.

3.1 StandOff Joins

Taking into account that an area-annotation a consists of a set of *one or more* regions $r_1, .., r_n$ (that do not overlap nor touch each other), we formally define:

`contains`(a_1, a_2)
  $\forall r_2 \in a_2 \exists r_1 \in a_1 : r_1.start \leq r_2.start \leq r_2.end \leq r_1.end$
`overlaps`(a_1, a_2)
  $\exists r_2 \in a_2, r_1 \in a_1 : r_1.start \leq r_2.end \wedge r_1.end \geq r_2.start$

Inspired by [6], we now define the following four *StandOff Joins* between two node sequences S_1 and S_2:

`select-narrow`(S_1, S_2) Containment semi-join: return those area-annotations from S_2 that are contained by some area-annotation in S_1.

`select-wide`(S_1, S_2) Overlap semi-join: return those area-annotations from S_2 that overlap with some area-annotation in S_1.

`reject-narrow`(S_1, S_2) Containment anti-join: return those area-annotations from S_2 that are not contained in any area-annotation in S_1.

`reject-wide`(S_1, S_2) Overlap anti-join: return those area-annotations from S_2 that do not overlap with any area-annotation in S_1.

Similar to XPath steps, we expect as the result of these operators a unique node sequence in document order.

StandOff Joins between U2 and Shots	Matches
select-narrow(`//music[artist="U2"],//shot`)	`Intro`
select-wide(`//music[artist="U2"],//shot`)	`Intro Interview`
reject-narrow(`//music[artist="U2"],//shot`)	`Interview Outro`
reject-wide(`//music[artist="U2"],//shot`)	`Outro`

The above table lists some example StandOff joins on the StandOff annotations in Figure 1 and their results (which are sequences of XML nodes). The second row shows the expression for selecting all video shots during which U2 music was played, whereas the expression in first row selects only those scenes during which this happened all the time. The third row asks for all shots during which time no U2 music was played, whereas the last row yields only those scenes that at some point of time had no U2 music.

```
declare module standoff = "http://w3c.org/tr/standoff/"

declare function select-narrow($input as xs:anyNode*)
                                      as xs:anyNode*
{
 (for $q in $input
   for $p in root($q)//*
   where $p/@start >= $q/@start
     and $p/@end <= $q/@end
   return $p)/.
}
```

Figure 2: StandOff Joins in a Module

```
declare function select-narrow($input as xs:anyNode*,
                          $candidates as xs:anyNode*)
                                      as xs:anyNode*
{
 (for $q in $input
   for $p in $candidates
   where $p/@start >= $q/@start
     and $p/@end <= $q/@end
     and root($p) = root($q)
   return $p)/.
}
```

Figure 3: Function with Candidate Sequence

3.2 StandOff XQuery Syntax

The question now is how to denote the StandOff joins in XPath/XQuery. We focus first on the case where we have a context node sequence (S_1) and where we place no restriction on the output (S_2=`//*`).

Alternative 1: XQuery Functions. Without extending the XQuery standard, we can define a StandOff XQuery library module that implements the four operators as XQuery functions, as shown in Figure 2. Note that we look for matches for `$q` in the *candidate sequence* yielded by `root($q)//` and thus only return matches from the same XML fragment. A final self-axis step `/.` ensures unique results in document order. An example use of this notation is `select-wide(//music)/self::shot`, which returns all `shot` elements which region-wise overlap with some `music`.

Alternative 2: Functions With Candidate Sequence. One problem with the previous approach is that in many cases, queries focus on retrieving annotations of a particular kind (e.g. most commonly an element name test, such as `music` in our example). As finding the result nodes of a StandOff join in XQuery involves a check against all document nodes, it can be beneficial to push down selections, reducing the possible result nodes to a certain candidate sequence. Thus, such a candidate sequence (basically, S_2) could be an additional parameter of our StandOff functions, as shown in Figure 3. Note the additional test on equal roots is needed to ensure that only nodes from the same XML fragment match.

The previous example query can now be expressed as `select-narrow(//music,//shot)`.

Alternative 3: Built-in Functions. One problem with the previous approaches is that they only work for XQuery and not for XPath. Another problem is that evaluation cost of the StandOff joins remains of quadratic order (it compares all context nodes with all document nodes). This problem can only be tackled by adding new evaluation algorithms inside the XPath/XQuery engine, which then implement the StandOff functions as built-ins. In Section 4 we will describe how StandOff joins can be evaluated in almost linear time using a stack-based region-merge algorithm that exploits the containment relationships *between* the context nodes to reduce the comparison work (this principle resembles Staircase Join [9] and Structural Join [2] – but differs since the annotation regions can overlap, something which does not occur in XML tree numbering schemes).

Alternative 4: XPath Steps. If we already are extending the XPath/XQuery evaluation engine, we might consider to introduce the StandOff joins as new XPath steps. The previous example query could then be expressed as `//music/select-narrow::shot`. StandOff steps should then conform to the basic characteristics of XPath steps: they should return duplicate-free node sequences in document order, and match only nodes from the same XML fragment.

3.3 Proposal: StandOff XPath Steps

We recommend Alternative 4 (new StandOff XPath Steps) for the following reasons:
(i) Ease of use. As the main use case of the StandOff joins is to move from a sequence of context nodes to a sequence of result nodes, we noticed end users feel most comfortable with XPath step notation, also because StandOff joins are almost always accompanied by a name-test.
(ii) Ease of indexing. XPath steps only match nodes from the same document, which makes it possible to pre-create effective indices for them. One could possibly try to define StandOff axes in the XQuery function approach to return matches across any XML fragment, though the semantics of this are only clear for Approach 2 (without a candidate sequence parameter, it would be unclear from which documents the candidates should come). This implies, however, that a global index over the entire document collection must be maintained, since a-priori it is unknown which documents will be queried together. This may lead to the index containing many data items that are not needed if a small set of documents is queried, as well as cause needless transaction conflicts among documents in case of updates.
(iii) Ease of Implementation. As the StandOff joins share the same API and main input and output result properties as the standard XPath steps, they can be easily incorporated in the algorithmic infrastructure for executing XPath expressions already in a XML DBMS. Of particularly important point here is query optimization. XPath steps with element-tests or other selections may either evaluate the step first and then restrict the results according to the selection, or alternatively, push down the selection into the path step evaluation. XQuery database systems should have both evaluation options available and use their *query optimizer* to decide between them. StandOff steps as XPath step directly profit from this infrastructure. In contrast; the usual handling of builtin-functions (Approach 3) enforces selection pushdown, which for non-selective predicates may lead to counter-productive evaluation of large intermediate result sequences. Of course, the query optimizer could be changed to treat the StandOff built-in functions in a special way to optimize them like XPath steps, but this requires major optimizer changes (at least, that was our experience in MonetDB/XQuery).

In the following, we focus on the efficient implementation of our new StandOff XPath axis steps `select-narrow`, `select-wide` `reject-narrow` and `reject-wide` in MonetDB/XQuery.

4. IMPLEMENTATION

This section describes how the Standoff XPath axis steps were implemented and successively optimized in the XML DBMS MonetDB/XQuery. We first explain the concept of loop-lifting and the table representation of XQuery results in MonetDB/XQuery, as this will play a role in the implementation of our StandOff axis steps as well.

4.1 MonetDB/XQuery

MonetDB/XQuery is an XML DBMS that uses the relational MonetDB database system as back-end. Its front-end consists of the *Pathfinder* compiler, that compiles XQuery into vanilla relational algebra. Furthermore, when starting the XQuery server, MonetDB loads a *runtime-module* that extends it with a new relational operator, *Staircase Join*, that provides fast XPath evaluation on the relational tables in which the system stores (shredded) XML documents [9].

pos	item
1	*"hello"*
2	*"world"*

Pathfinder translates all XQuery expressions into relational algebra and thus always produces a table as a result. In XQuery, expressions yield *ordered item sequences* as a result, and these get represented as tables with two columns: `pos|item`. That is, to retain the order of the `items` in the otherwise unordered relational model, an integer column `pos` is kept in addition to the value. The above example shows the table representation of `("hello", "world")`.

A central concept of the relational XQuery translation by Pathfinder is *loop-lifting* [8]. Each XQuery is translated bottom-up into a single relational algebra plan consisting only of the classical relational operators (Select, Project, Cartesian Product, Join, Aggregation); that is, the XQuery concept of nested `for`-loops is fully removed and a single bulk (=efficient and optimizable) execution plan is created.

```
for $x in ("twenty", "thirty")
 for $y in ("one", "two")
  let $z := ($x,$y)
  return $z
```

iter	pos	item$_{\$x}$
1	1	*"twenty"*
2	1	*"twenty"*
3	1	*"thirty"*
4	1	*"thirty"*

iter	pos	item$_{\$y}$
1	1	*"one"*
2	1	*"two"*
3	1	*"one"*
4	1	*"two"*

iter	pos	item$_{\$z}$
1	1	*"twenty"*
1	2	*"one"*
2	1	*"twenty"*
2	2	*"two"*
3	1	*"thirty"*
3	2	*"one"*
4	1	*"thirty"*
4	2	*"two"*

We demonstrate how this works in the case of the above example XQuery. The result of an XQuery at each step of bottom-up compilation is a relational plan that yields the result sequence *for each* nested iteration, all stored together. To make this possible, these intermediate tables have three columns: `iter|pos|item`, where `iter` is a logical iteration number. If we focus on the execution state in the innermost iteration body of the example XQuery, there will be three such tables that represent the live variables `$x, $y` and `$z` respectively. As we can see from the `iter` columns, there are four iterations in that scope (numbered from 1 to 4) and as expected, `$x` takes on value `"twenty"` in the first two iterations and `"thirty"` in the second two (similarly, `$y` takes on value `"one"` in the odd iterations and `"two"` in the even ones). Finally, `$z` is a sequence of two values in all four iterations (consisting of the value of `$x` followed by the value of `$y`).

The concept of loop-lifting is also exploited in Staircase Join. That is, if an XPath axis step is executed inside a `for`-loop with n iterations, a naive strategy would call the original Staircase Join iteratively, causing n sequential scans over the shredded document table during its execution. It was shown in [5], that a loop-lifted variant of Staircase Join performs an order of magnitude faster than the iterative strategy, if a query contains such nested XPath axis steps. Loop-lifted Staircase Join is able to compute an XPath axis step for multiple context node sequences in a single sequential pass. Note that the input and output of normal Staircase Join is a sequence of nodes (both represented as `pos|item` tables), whereas the input and output of the loop-lifted variant are `iter|pos|item` tables, thus containing a set of node-sequences (one for each iteration).

4.2 Implementing The StandOff Axes

Our initial approach for experimenting with StandOff querying used the Alternatives 1 and 2 (implementation as an XQuery library module with user-defined functions). These implementations will serve as base-lines for our experiments.

4.3 The Region Index

We added a region index to the relational representation of XML documents. It consists of `start|end|id` that is kept clustered on `start`. Note that we can represent non-contiguous areas, that consist of multiple regions, by repeating the same node-`id` in several entries in the index. MonetDB/XQuery uses the pre-order rank as node-`id`, but in principle any kind of identifier can be used here.

In case of StandOff steps without selections, the entire index is the *candidate sequence*. If a sequence of candidate node-`ids` is passed in (which can be produced e.g. using the element index of MonetDB/XQuery), an index intersection on node-`id` is performed to calculate the candidate sequence, in which the `start` ordering of the region index is preserved.

4.4 Basic StandOff MergeJoin

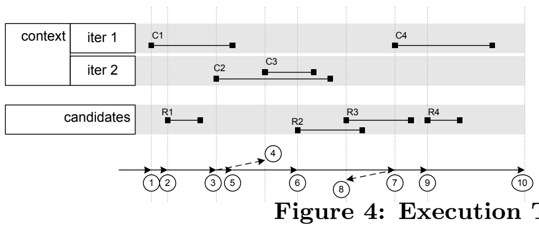
A first step is to partition the context sequence per XML document (fragment). The main algorithm is then repeatedly executed for each distinct XML fragment, and the results concatenated. Thus, all subsequent algorithms only deal with (input) context sequence and the candidate sequence, as described above, that only contain nodes from the same XML fragment.

The next step is to fetch the [`start,end`] values for all context `node-ids`, and to sort the context sequence on `start`.

The main algorithm implements a merge semi-join on `start` between the context and candidate sequences. It keeps a list of all "active" context items sorted on their `end`-value. A context item is active as long as it can produce result nodes. In case of `select-narrow` this means that $\texttt{cur}_{context}$`.end` $\geq$ $\texttt{cur}_{candidates}$ `.start`. If this is no longer so, the context item is removed. Meanwhile, all candidate nodes that are contained in active context items are added to the result. Also, new context items are only added if they are not contained in an item already in the list from the same iteration.

Note that StandOff MergeJoin bears strong resemblance with algorithms for computing the `descendant` step in region encodings, such as Structural Join [2], and Staircase Join [9]. However, these other algorithms strongly exploit the tree properties of such document encodings such that we cannot use them as-is on overlapping annotation regions.

The behavior of `select-wide`, `reject-narrow` and `reject-wide` is highly similar. We omit a detailed algorithm here but discuss that in the following for the loop-lifted variant.



1 add c_1 c list	(line 8)	6 skip r_2	(lines 32-35)
2 add (iter1, r_1) to result	(lines 32-34)	remove c_2 from list	(line 31)
3 push c_2 on list	(line 41)	7 add c_4 to list	(line 41)
4 skip c_3	(lines 11-18)	8 skip r_3	(lines 21-24)
5 remove c_1 from list	(line 31)	9 add (iter1, r_4) to result	(lines 32-34)
		10 exit	(line 38)

Figure 4: Execution Trace of Loop-Lifted StandOff MergeJoin

```
1  [iter,region] ll_select_narrow_join(
2                          [iter,start,end] context,
3                          [pre,start,end] candidates) {
4    result := [,];
5    i := 0;  /* iterate over context */
6    j := 0;  /* iterate over candidates */
7    active_items = [,]; top := 0;
8    active_items[top++] = context[i];
9    while (i < |context|) {
10     next_i := i + 1;
11     /* skip self-overlapping regions in same iter */
12     while(next_i < |context|) {
13       tmp := find_active_item(context[next_i].iter)
14       if (tmp && tmp.end <= context[i].end)
15         next_i++;
16       else
17         break;
18     }
19     next_start := (next_i < |context|) ?
20       context[next_i].start : MAX_INT;
21     /* skip non-possible candidates */
22     while (j < |candidates| &&
23         candidates[j].start < context[i].start)
24       j++;
25     /* analyze potential candidates */
26     while(j < |candidates| &&
27         candidates[j].start < next_start) {
28       /* trim the active items list when needed */
29       while(top > 0 &&
30           active_items[top-1].end < candidates[j].start)
31         top--;
32       for (k := 0; k < top &&
33           active_items[k].end >= candidates[j].end; k++)
34         result += (k.iter, candidates[j])
35       j++;
36     }
37     if (j == |candidates|)
38       break;
39     /* add next context item to active_items */
40     i := next_i;
41     replace_active_items_with(context[i]);
42   }
43   return result;
44 }
```

Listing 1: pseudo code for loop-lifted `select-narrow`

4.5 Loop-lifted StandOff MergeJoin

Listing 1 shows code for the loop-lifted `select-narrow`-step. The input for the step are `iter|start|end` context items sorted on `start` (the `iter`-value is not present in the basic algorithm, and serves to separate the different input context sequences in the loop-lifted version) and `start|end` candidate items.

The code loops over all the candidate items as long as there are context items or candidate items available (line 9 and 37-38). The algorithm maintains a list of active context items. Lines 11-18 skip over context items that are completely contained in the active items, because these nodes will not yield any additional results. If we ran out of context items our next context item will be infinitely far away (lines 19-20), thus we can safely skip over all candidate regions that fall in between context items. Afterwards, line 26-36 will be looping over candidate items as long as the list of active items is valid (no new context items need to be added to the list). The active items list will shrink by removing items which cannot participate in new results anymore(29-31). This happens when the `start`-value of the current candidate comes after the `end`-value of such a context item. For all candidates strictly contained in an active context item a result is produced (lines 32-34). After having processed all possible candidates until the start of the next context item, the new context item is added to the list (40-41).

The listed algorithm produces matching combinations of `iters` and regions. Depending on whether the annotation mode supports areas of multiple regions, some post-processing (omitted) occurs that maps these into node-`ids` (unique and in document order per `iter`).

iter	id	start	end
1	c_1	0	15
2	c_2	12	35
1	c_3	20	30
1	c_4	55	80

context

id	start	end
r_1	5	10
r_2	22	45
r_3	40	60
r_4	65	70

candidates

We illustrate how the loop-lifted StandOff MergeJoin operates on the context and candidate input tables to the left. Figure 4 contains an execution trace; the left part shows the state of the active context item list, while the right part shows the steps of the algorithm.

```
for $b in doc("xmark110MB.xml")
            //site/select-narrow::open_auctions
            /select-narrow::open_auction
return <increase> {
  $b/select-narrow::bidder[1]/select-narrow::increase
} </increase>
```

Figure 5: StandOff XMark Query 2

4.6 Experimental Evaluation

We evaluated the performance of the various implementations of the StandOff axis steps on a StandOff version of the XMark benchmark [12]. We modified the XMark document to a StandOff document, by putting the textual contents of the auctions document in a separate file (the BLOB), whereas the auctions document contains for each element node instead of the text node a region (in attribute format) that refers to the BLOB. The order in which the element nodes appear has also been permuted on a coarse level, thereby removing some of the original parent-child relationships. Queries 1, 2, 6, and 7 of the XMark benchmark were rewritten to use StandOff annotation. This means that `descendant` and `child` steps were replaced by `select-narrow`. Figure 5 shows the translation for XMark query 2.

Our benchmark platform was an Athlon 3800+ (2.4GHz) with 2GB RAM and two 100GB SATA drives running Linux 2.6. We tested against the released version 0.10 of MonetDB/XQuery that contains the StandOff extensions. In our experiments, we compared the three alternatives:

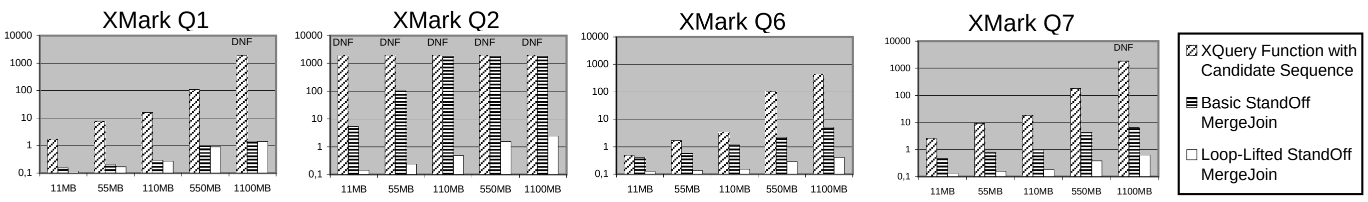


Figure 6: Performance on StandOff XMark Q1, Q2, Q6, and Q7 (in sec.)

XQuery Function with Candidate Sequence. Here, the Standoff axis steps are implemented as user-defined XQuery functions. We use the variant where a candidate node sequence can be passed as a restriction. In the XMark queries, there is always a test on element name, so such a restriction is possible. The variant without Candidate Sequence was also tested, and produced DNF (Did Not Finish within an hour) on all queries and all tested document sizes (11MB and larger). We can see though, that even with the Candidate Sequence, this variant is one to two orders of magnitude slower than the alternatives.

Basic StandOff MergeJoin. This variant performs very well on XMark Q1, but produces DNF results on Q2. The main difference between these two queries is that the path steps in Q2 appear in a `for`-loop. In that case, the Basic StandOff algorithm is called for each iteration, leading to repeated full scans of the region index.

Loop-Lifted StandOff MergeJoin. This variant is clearly superior, beating the other variants on most queries by one or more orders of magnitude. In fact, the overall performance of `select-narrow` is less than 20% slower than the loop-lifted `descendant` Staircase Join. These results underline the significance of the loop-lifting technique and confirm the results obtained on loop-lifting Staircase Join in [5].

5. RELATED WORK

Early notion of multi-dimensional markup stems from the SGML era (the CONCUR feature) and the Text Encoding Initiative (TEI) [13]. Thompson and McKelvie later introduced the notion of *standoff* annotation [15]. Other attempts have been made to create a dialect of XML to represent multiple annotations inline, for example the general ordered-descendant directed acyclic graph (GODDAG) and LMNL [14]. For the GODDAG annotation language there has also been a proposal for a query language called EXPath [10]. Ogilvie issued in [11] an indirect request for a simple XQuery based extension to allow for stand-off querying. The axis steps we introduced in this paper behave exactly like Ogilvie's overloaded descendant step. Finally, the loop-lifted StandOff joins introduced here resemble explicit sort-merge joins defined for temporal databases [7]. Loop-lifted StandOff join is a particular variant, as it implements a semi-join and is nested: instead on node-sets it semi-joins sets of node-sets. This special semantics is exploited in its stack-based algorithm. Like suggested in [7], it could be beneficial to substitute the stack (from which we currently may delete elements in the middle – so it really is a list) by a heap, in data-distributions that cause it to grow long.

6. CONCLUSION AND FUTURE WORK

We proposed the use of XML for representing multiple overlapping annotation hierarchies, defined four StandOff join operators for querying such annotations, and proposed to add these as new XPath axis steps. As an alternative, these new operators could also be supported in XPath/XQuery processors by means of built-in functions, but this is less intuitive for end users and provides less flexibility to the XQuery optimizer to handle selection pushdown.

We outlined a family of new algorithms, called StandOff MergeJoin, that can execute these new XPath axis steps efficiently by making use of an index on the region annotations. The algorithms were implemented in MonetDB/XQuery and released in open source. We evaluated the performance on a StandOff version of the XMark benchmark, which shows that the loop-lifted StandOff MergeJoin is highly efficient and can query >GB annotation documents interactively.

We will continue to use our XQuery extensions to manage and query annotations in the areas of multimedia retrieval, natural language processing and digital forensics, and are on the lookout for more application areas of this versatile technology (e.g. temporal annotations in MPEG-7 and SMIL, but also genome sequence annotations in bioinformatics). Such new application experiences may bring further insight regarding the potential need for more than four axis steps, as well as the usability of the current solution.

7. REFERENCES

[1] D. Ahn. NLP-XML workshop on multi-dimensional markup in natural language processing (in conjunction with EACL 2006).

[2] S. Al-Khalifa, H. V. Jagadish, N. Koudas, J. M Patel, D. Srivastava, and Y. Wu. Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In *ICDE*, 2002.

[3] W. Alink. XIRAF - an XML information retrieval approach to digital forensics. Master's thesis, Univ. Twente, October 2005.

[4] J.F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[5] P. Boncz, T. Grust, M. van Keulen, S. Manegold, and J. Teubner. MonetDB/XQuery: A fast xquery processor powered by a relational engine. In *SIGMOD*, 2006.

[6] F.J. Burkowski. Retrieval Activities in a Database Consisting of Heterogeneous Collections of Structured Text. In *SIGIR*, 1992.

[7] Dengfeng Gao, Christian S. Jensen, Richard T. Snodgrass, and Michael D. Soo. Join operations in temporal databases. *VLDB Journal*, 14(1):2–29, March 2005.

[8] T. Grust, S. Sakr, and J. Teubner. XQuery on SQL Hosts. In *VLDB*, Toronto, Canada, 2004.

[9] T. Grust, M. van Keulen, and J. Teubner. Staircase Join: Teach a Relational DBMS to Watch its (Axis) Steps. In *VLDB*, 2003.

[10] I.E. Iacob and A. Dekhtyar. Towards a Query Language for Multihierarchical XML: Revisiting XPath. In *WebDB*, 2005.

[11] Paul Ogilvie. Retrieval using structure for question answering. In *Twente Data Management Workshop (TDM)*, 2004.

[12] A. Schmidt, F. Waas, M.L. Kersten, M.J. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In *VLDB*, 2002.

[13] C. M. Sperberg-McQueen and L. Burnard. Guidelines for Electronic Text Encoding and Interchange. Technical report, 1992.

[14] C.M. Sperberg-McQueen and C. Huitfeldt. GODDAG: A Data Structure for Overlapping Hierarchies. *Lecture Notes in Computer Science*, 2023:139 – 160, 2004.

[15] H.S. Thompson and D. McKelvie. Hyperlink semantics for standoff markup of read-only documents. In *SGML Europe'97*.