Covering Indexes for XML Queries: Bisimulation − Simulation = Negation

Prakash Ramanan

Department of Computer Science
Wichita State University
Wichita, KS 67260–0083
ramanan@cs.twsu.edu

Abstract

Tree Pattern Queries (TPQ), Branching Path Queries (BPQ), and Core XPath (CXPath) are subclasses of the XML query language XPath, $TPQ \subset BPQ \subset CXPath \subset XPath$. Let $TPQ = TPQ^+ \subset BPQ^+ \subset CXPath^+ \subset XPath^+$ denote the corresponding subclasses, consisting of queries that do not involve the boolean negation operator `not` in their predicates. Simulation and bisimulation are two different binary relations on graph vertices that have previously been studied in connection with some of these classes. For instance, TPQ queries can be minimized using *simulation*. Most relevantly, for an XML document, its **bi***simulation* quotient is the smallest index that *covers* (i.e., can be used to answer) all BPQ queries. Our results are as follows:

- A $CXPath^+$ query can be evaluated on an XML document by computing the *simulation* of the query tree by the document graph.
- For an XML document, its *simulation* quotient is the smallest covering index for BPQ^+. This, together with the previously-known result stated above, leads to the following: For BPQ covering indexes of XML documents, Bisimulation − Simulation = Negation.
- For an XML document, its *simulation* quotient, with the `idref` edges ignored throughout, is the smallest covering index for TPQ.

For any XML document, its simulation quotient is never larger than its bisimulation quotient; in some instances, it is exponentially smaller. Our last two results show that disallowing negation in the queries could substantially reduce the size of the smallest covering index.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003

1 Introduction

We consider a model of XML documents in which we ignore comments, processing instructions and namespaces. Attributes other than `id` and `idref` will be treated as subelements. Then, an XML document can be represented as a tree along with a set of `idref` edges (see [1]); each tree edge denotes an element–subelement relationship.

The Query Classes

XML query languages such as XPath [3] and XQuery [6] allow for navigation in an XML document, to locate desired elements. XPath provides thirteen different axes (directions) for navigation. In our model, we will not consider the `attribute` and `namespace` axes. The remaining eleven axes `self, child, descendant, descendant-or-self, parent, ancestor, ancestor-or-self, preceding, preceding-sibling, following`, and `following-sibling`, will be abbreviated by their initials (first 2 letters in the case of `parent` axis) `s, c, d, ds, pa, a, as, p, ps, f` and `fs`, respectively. In addition, we consider two more axes (for a total of 13 axes): The `idref` and `ridref` axes (abbreviated as `ir` and `rir`) correspond to navigating a single `idref` edge in the forward and backward directions, respectively. In XPath, the `ir` axis is available through the core library function `id`. The `rir` axis is not explicitly

available in XPath, but it can be partly emulated using the node identity operator == available in XPath 2.0 [3].

Gottlob et al. [7] defined a fragment of XPath, called Core XPath (CXPath), that corresponds to the logical core of XPath. CXPath does not contain arithmetic and string operations, but otherwise has the full navigational power of XPath. We let CXPath consist of queries involving the thirteen axes, and predicates involving them and the three boolean operators `and`, `or` and `not`. CXPath queries ignore the values (PCDATA) of atomic elements. Kaushik et al. [9] defined a subclass of CXPath called Branching Path Queries (BPQ). BPQ consists of those CXPath queries that ignore the order of sibling elements in the input document: It allows nine axes, excluding the four order respecting axes `p`, `ps`, `f`, `fs`.

Amer-Yahia et al. [2] defined a subclass of BPQ called Tree Pattern Queries (TPQ). TPQ queries involve only the four axes `s`, `c`, `d`, `ds`, and predicates involving them and the boolean operator `and`; in particular, they do not involve `idref` edges. We have $TPQ \subset BPQ \subset CXPath \subset XPath$.

For any class C of queries, let C^+ denote the subclass of C consisting of those queries that do not involve the boolean operator `not` in their predicates. Note that $TPQ = TPQ^+ \subset BPQ^+ \subset CXPath^+ \subset XPath^+$.

Query Evaluation and Indexing

For an XML document D, an *index* D_I is obtained by merging "equivalent" nodes into a single node. For example, for D in Figures 1a and 2a, an index is shown in Figures 1b and 2b, respectively. For a node n in D_I, let $extent(n)$ be the set of nodes of D that were merged together to create node n. For example, in Figure 1b, the extent of the node labeled b is $\{2, 7\}$.

We say that a query Q *distinguishes* between two nodes in D, if exactly one of the two nodes is in the result of evaluating Q on D. An index in which these two nodes are in the same extent can not be used to evaluate Q on D. An index D_I is a *covering index* for a class C of queries, if the following holds: No query in C can distinguish between two nodes of D that are in the same extent in D_I. A covering index D_I can be used to evaluate the queries in C, without looking at D, as follows: First evaluate the query on D_I; for each node n of D_I that is in the result, output $extent(n)$. Since D_I is smaller than D, this would be faster compared to evaluating the query directly on D.

We study the evaluation of CXPath queries, and covering indexes for subclasses of CXPath. An XPath query is *absolute* if its navigation in an XML document starts from the root; otherwise it is *relative*. It is easily seen that, for relative queries, the smallest covering index is D itself. Of the results discussed below, the results pertaining to indexing apply only to absolute queries; the results pertaining to query evaluation apply to both absolute and relative queries.

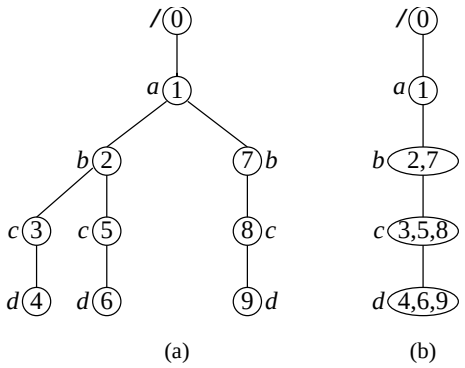

Figure 1: (a). An XML Document D. (b). Its Bisimulation Quotient.

First, let us consider covering indexes for $CXPath^+$ or CXPath. For any node n in an XML document D, we can construct an absolute query $Q \in CXPath^+$ that distinguishes n from all the other nodes, as follows. Consider the tree path (no `idref` edges) in D from the root to node n. For each node in this path, other than the root, count the number of its siblings to the left and to the right (i.e., preceding and following siblings, respectively); the query Q would enforce exactly this count requirement. For example, for D in Figure 1a, the query $Q = /\mathtt{c} :: *[\mathtt{fs} :: *]/\mathtt{c} :: *[\mathtt{ps} :: *]$ distinguishes node 5 from all the other nodes (To enforce the requirement that a node has two preceding siblings, we would use the predicate $[\mathtt{ps} :: *[\mathtt{ps} :: *]]$). Consequently, for any D, the smallest covering index for $CXPath^+$ or CXPath is D itself.

Now, let us consider some nontrivial results. Simulation and bisimulation [10, 13] are two different binary relations on graph vertices. They provide two different notions of dominance/equivalence between the vertices, and have been studied in process equivalence [10, 13, 8] and in graph models for data. In particular, Buneman et al. [5] (also see [1]) used simulation to define a schema for semistructured data. Simulation and bisimulation have also been studied in connection with query minimization and with indexing of documents. Ramanan [14] showed that TPQ queries, without wildcard $*$ for node types, can be minimized using simulation. Milo and Suciu [11] showed that, for a semistructured document, its (backward) simulation and bisimulation quotients are two covering indexes for linear path queries (paths starting from the root; no branching); if the document is a tree, simulation and bisimulation coincide, and the corresponding quotient is the smallest covering index for linear path queries.

Kaushik et al. [9] showed that, for an XML document (possibly containing `idref` edges), its (forward and backward) **bi***simulation* quotient is the smallest covering index for BPQ.

Example 1.1. To illustrate the result of Kaushik et al. in a simple setting, consider the document in Figure 1a, without any `idref` edges. The following nodes are bisimilar: $(2, 7)$, $(3, 5, 8)$ and $(4, 6, 9)$; the bisimulation quotient is shown in Figure 1b. By Kaushik et al.'s result, no BPQ query can distinguish between nodes 2 and 7; between 3, 5 and 8; or between 4, 6 and 9.

We point out that if we allow the node identity operator == for BPQ queries, then Kaushik et al.'s result does not hold: The bisimulation quotient of an XML document is no longer a covering index for the resulting class of queries. For example, in Figure 1a, nodes 3, 5 and 8 are bisimilar. But the query $/\texttt{d}::c[\texttt{not s} :: * == \texttt{pa} :: */\texttt{c} :: *]$ (in abbreviated form, $//c[\texttt{not} .== ../*]$) can distinguish nodes 3 and 5 from node 8. ∘

For an XML document, if its bisimulation quotient is small, then a BPQ query can be evaluated faster by using this index. Kaushik et al. showed that, for many XML documents, the bisimulation quotient is about the same size as the document itself; this is because the bisimulation condition is quite onerous, and only a few pairs of nodes would turn out to be bisimilar. Hence this index is unlikely to offer much speedup in evaluating a BPQ query. So, they considered restricting the class of queries as follows, in order to obtain smaller covering indexes.

- Indexing only certain element types. This corresponds to replacing all the other element types in the document by $*$, before computing the index.
- Indexing only certain `idref` edges, namely, those between specified source and destination node types. The remaining `idref` edges are dropped from the document before computing the index.
- Indexing only paths of specified lengths.

Using these restrictions, they were able to obtain smaller covering indexes for the restricted classes of queries. These covering indexes could be used to speed up the evaluation of the restricted classes of queries. The speed up depends on the size of the covering index, compared to the size of the original XML document.

In this paper, we determine the smallest covering indexes for two subclasses of BPQ, namely, BPQ^+ and TPQ. BPQ^+ is an important subclass, since most real life XPath queries do not involve negation; as an anecdotal evidence, most of the example queries considered by Kaushik et al. do not involve negation. Amer-Yahia et al. [2] argued that many real life queries are TPQ queries. Our results are as follows:

- A $CXPath^+$ query can be evaluated on an XML document by computing the *simulation* of the query tree by the document graph (Section 4). This result leads to an $O(|Q||D|)$ time algorithm for evaluating $CXPath^+$ queries; it is also used to prove our main result.

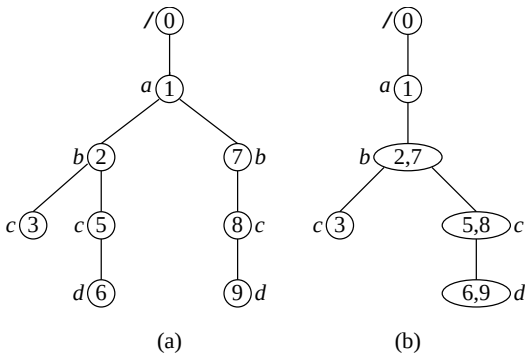

Figure 2: (a). An XML Document D. (b). Its Simulation Quotient.

- Our main result: For an XML document, its (forward and backward) *simulation* quotient is the smallest covering index for BPQ^+ (Section 6).
- The *simulation* quotient (of an XML document), with the `idref` edges ignored throughout, is the smallest covering index for TPQ (Section 7).

Unlike the result of Kaushik et al., our three results above hold if we add the node identity operator == to $CXPath^+$, BPQ^+ and TPQ, respectively. But due to lack of space, we will not discuss this further.

In general, bisimulation is a $refinement$ of simulation: If two nodes are bisimilar, then they are also similar. So, for any XML document, its simulation quotient is never larger than its bisimulation quotient; in some instances, it is exponentially smaller (see Section 5). Our main result shows that disallowing negation in the queries could substantially reduce the size of the smallest covering index.

Example 1.2. We illustrate our main result in a simple setting (no `idref` edges). First, consider the document in Figure 1a. Earlier, we saw its bisimulation quotient in Figure 1b. For this document, simulation is same as bisimulation. No BPQ query, and hence no BPQ^+ query, can distinguish between similar nodes.

Next, consider the document D in Figure 2a. No two nodes are bisimilar; the bisimulation quotient is D itself. But the following pairs of nodes are similar: $(2, 7)$, $(5, 8)$ and $(6, 9)$. The simulation quotient is shown in Figure 2b; as per our main result, this is a covering index for BPQ^+. The BPQ query $//b\,[c\,[\texttt{not}\ d]]$ distinguishes between nodes 2 and 7; any BPQ query that distinguishes between these two nodes (or between nodes 5 and 8; or between 6 and 9) must involve negation. ∘

Sections 2, 3 and 5 contain the preliminary definitions and notations we need. In Section 2, we describe the classes of queries we study, and show that any CXPath query can be represented as a query tree. In Section 3, we define the simulation relation on the vertices of ordinary graphs. In Section 5, we define the

simulation and bisimulation relations on an XML document, and also define their quotients. We explain the difference between the two relations, and show that, in some instances, the simulation quotient is exponentially smaller. Our three results are proved in Sections 4, 6 and 7. In Section 8, we present our conclusions.

2 Queries and Query Trees

An XML document is represented as a graph $D = (N, E, E_{ref})$, where N is a set of nodes, E is a set of tree edges, and E_{ref} is a set of `idref` edges between the nodes; the subgraph $T = (N, E)$ without the `idref` edges is a tree (see [1]). In conformance with the XPath data model [3], the root of D or T, denoted by $root(D)$ or $root(T)$, is a node in N that does not correspond to any element in the document; it has the unique element type `/`. Its unique child node corresponds to the root element of the document. Each node $n \in N - \{root(D)\}$ corresponds to an XML element, and is labeled by its element type (tag name) $\tau(n)$ from a finite alphabet Σ. Each tree edge denotes an element–subelement relationship. When we talk of child, descendant, parent, ancestor and sibling relationships between the nodes in N, we only consider tree edges; i.e., these relationships hold in T, without regard to E_{ref}. The children of a node are ordered from left to right, and represent the content (i.e., list of subelements) of that element. Atomic elements (i.e., those without subelements) correspond to the leaves of the tree; CXPath queries ignore the values (PCDATA) of these elements.

A *context node set* (*cns*) is a set of nodes in an XML document; i.e., it is a subset of N. A CXPath query starts with an initial cns, and computes a new cns which is the result of the query. An *absolute* CXPath query is of the form `/ls1/ls2/...`, where `ls` stands for a *location step*; the first `/` indicates that the navigation starts at $root(D)$; i.e., the initial cns consists only of $root(D)$. A *relative* CXPath query is of the form `ls1/ls2/...`, where the navigation starts from some initial cns (to be specified). Starting from some cns ($\{root(D)\}$ for an absolute query), the location steps are applied from left to right, to compute the result of the query. Each *location step* is of the form `axis::node-test[predicate1][predicate2]...`. It consists of an axis identifier (one of thirteen mentioned earlier), a node test, and zero or more predicates. We consider two kinds of node tests: Particular type in Σ, and wildcard $*$; they match nodes of the specified type, and nodes of all types, respectively. Starting from a previous cns, a location step identifies a new cns: For each node in the previous cns, the axis identifies a new set of possible nodes, which are then filtered based on the node test and the predicates; the nodes that pass the tests are added to the new cns. The *result* of a query is the cns resulting from the last location step.

Each `predicate` is either a boolean combination of predicates, or is a CXPath query. A predicate that is a CXPath query q is `true` if the result of q is nonempty (i.e., contains at least one node).

The class CXPath of queries is defined by the following grammar, where `axis` denotes one of the thirteen axes discussed earlier, and `nt` $\in \Sigma \cup \{*\}$ denotes a node test.

<*cxpquery*> ::= <*acxpquery*> | <*rcxpquery*> (absolute or relative)
<*acxpquery*> ::= /<*rcxpquery*>
<*rcxpquery*> ::= <*location_step*>
| <*location_step*>/<*rcxpquery*>
<*location_step*> ::= `axis` :: `nt` <*predicates*>
<*predicates*> ::= ϵ | [<*predicate*>] <*predicates*>
<*predicate*> ::= <*predicate*> `and` <*predicate*>
| <*predicate*> `or` <*predicate*>
| `not` <*predicate*> | <*rcxpquery*>

BPQ is the subclass of CXPath,
where `axis` $\in \{$`s`, `c`, `d`, `ds`, `pa`, `a`, `as`, `ir`, `rir`$\}$. TPQ is the subclass of BPQ, where `axis` $\in \{$`s`, `c`, `d`, `ds`$\}$, and the boolean operators `or` and `not` are not allowed.

A CXPath query Q can be represented by an unordered *query tree* $tree(Q) = (V, A)$, where V is a set of vertices, and A is a set of arcs. Each vertex $v \in V$ has a type $\tau(v)$, and a boolean operator $bool(v)$ associated with it. $\tau(v) \in \Sigma \cup \{/, *\}$ is the element type of v; / denotes the root type, and $*$ denotes 'any' type. $Bool(v) \in \{$`and`, `or`, `not`$\}$. Each arc $r \in A$ has an axis $axis(r)$ associated with it; $axis(r)$ is one of the thirteen axes we discussed earlier.

For a CXPath query Q, let us see how to construct $tree(Q)$. Let the *primary part* of Q, denoted by $primary(Q)$, be the query obtained from Q by dropping all the predicates from Q. We first construct a linear path $trunk(Q)$ that corresponds to $primary(Q)$. The root vertex v_0 does not correspond to any location step in Q; if Q is an absolute query, $\tau(v_0) = /$; else $\tau(v_0) = *$. For $i \geq 1$, the ith arc r_i and its destination vertex v_i correspond to the ith location step `lsi` in $primary(Q)$. Let `lsi` = `axisi` :: `nti`; then $axis(r_i)$ = `axisi` and $\tau(v_i)$ = `nti`. For all vertices $v_i \in trunk(Q)$, $bool(v_i)$ = `and`. The last vertex on $trunk(Q)$ generates the output of Q; this vertex is called the *output vertex* of Q, denoted by $opv(Q)$, and is marked with a $ sign in the figures.

Now, let us see how to add the predicates to $trunk(Q)$, to construct $tree(Q)$. For each `predicate` attached to `lsi` in Q, there is an arc r from v_i, with destination vertex v; $axis(r)$ = `s` and v is the root of $tree($`predicate`$)$. $Tree($`predicate`$)$ is constructed recursively, as follows. First, $\tau(v) = *$. If `predicate` is the *boolop* $\in \{$`and`, `or`, `not`$\}$ of predicate(s), then $bool(v) = boolop$; there is one arc (with axis `s`) from v for each operand, and the construction proceeds recursively for each operand, from the destination vertex of the corresponding arc. Else, `predicate` is a relative

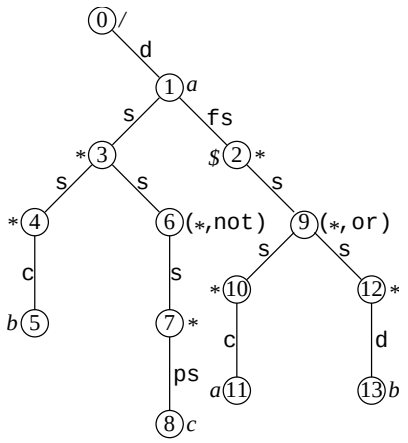

Figure 3: Q = /d :: a[c :: b and not ps :: c] /fs :: $*$[c :: a or d :: b]

CXPath query, and the construction proceeds recursively from v.

Example 2.1. For the CXPath query Q = /d :: a[c :: b and not ps :: c]/fs :: $*$[c :: a or d :: b], $tree(Q)$ is shown in Figure 3. The vertices are numbered in the order they were created using the procedure described above. $Trunk(Q)$ consists of vertices 0, 1 and 2, and the two arcs between them (corresponding to the two location steps in $primary(Q)$). For each vertex v, the pair $(\tau(v), bool(v))$ is shown next to v. We follow the convention that if $bool(v)$ is not specified, then it should be taken to be and. For each arc r, $axis(r)$ is shown next to it. ∘

It is easily seen that, in general, $|tree(Q)|$ is linear in $|Q|$. From now onwards, we will not distinguish between Q and $tree(Q)$; by Q we will mean $tree(Q)$.

From now onwards, to minimize confusion, we will use the terms *nodes* and *edges* while referring to components of the document graph D or tree T; we will use the terms *vertices* and *arcs* while referring to the corresponding components of Q. Note that, while Q consists of arcs with thirteen different axes, T (resp. D) consists of only one (resp. two) kind(s) of edges; also, while some vertices in Q might have the wildcard type ($*$), all the nodes in T and D (except $root(D)$) have types from Σ.

3 The Simulation Relation for Ordinary Graphs

In this section, we consider the simulation relation between two *ordinary* directed graphs: The graphs contain only one kind of arcs (directed, unlabeled), and only one kind of vertex label (type τ). Consider two directed graphs $G_1 = (V_1, A_1)$ and $G_2 = (V_2, A_2)$; for $i = 1, 2$, V_i is the set of vertices and A_i is the set of arcs of G_i. Let Σ be a finite alphabet of vertex labels. Each vertex v in V_1 or V_2 has a type $\tau(v) \in \Sigma$ associated with it.

Simulation [10] (also see [1]) is a *binary relation* between the vertex sets V_1 and V_2. It provides one possible notion of dominance/equivalence between the vertices of the two graphs. For a vertex v, let $post(v)$ denote the set of vertices to which there is an arc from v. *Forward simulation* (abbreviated as $Fsimulation$) of G_1 by G_2 is the largest binary relation $\preceq_{Fs} \subseteq V_1 \times V_2$ such that the following holds: If $v_1 \preceq_{Fs} v_2$, then

- Preserve vertex types: $\tau(v_1) = \tau(v_2)$.
- Preserve outgoing arcs: For each $v_1' \in post(v_1)$, there exists $v_2' \in post(v_2)$ such that $v_1' \preceq_{Fs} v_2'$.

If $v_1 \preceq_{Fs} v_2$, we say that v_1 is *Fsimulated by* v_2; let $Fsim(v_1) \subseteq V_2$ denote the set of all Fsimulators of v_1.

Sometimes, we are interested in the Fsimulation of a graph by itself. It is well-known that such Fsimulation is reflexive and transitive, but it may not be symmetric. Vertices v_1 and v_2 are $Fsimilar$, denoted by $v_1 \approx_{Fs} v_2$, if $v_1 \preceq_{Fs} v_2$ and $v_2 \preceq_{Fs} v_1$; Fsimilarity is an equivalence relation.

Backward simulation ($Bsimulation$) and $Bsimilar$ (denoted by $\preceq_{Bs}$ and $\approx_{Bs}$) are analogous to Fsimulation and Fsimilar, respectively; they deal with the *incoming* arcs at a vertex, as opposed to Fsimulation that deals with the *outgoing* arcs.

For trees, we can compute Fsimulation and Bsimulation bottom-up and top-down, respectively.

Example 3.1. For the tree in Figure 2a, computing Fsimulation bottom-up, we have $Fsim(6) = Fsim(9) = \{6, 9\}$, $Fsim(3) = \{3, 5, 8\}$, $Fsim(5) = Fsim(8) = \{5, 8\}$, $Fsim(2) = Fsim(7) = \{2, 7\}$, $Fsim(1) = \{1\}$, and $Fsim(0) = \{0\}$. So, the only nontrivial relational pairs are $6 \approx_{Fs} 9$, $3 \preceq_{Fs} 5$, $3 \preceq_{Fs} 8$, $5 \approx_{Fs} 8$, and $2 \approx_{Fs} 7$.

Computing Bsimulation top-down, we have $2 \approx_{Bs} 7$, $3 \approx_{Bs} 5 \approx_{Bs} 8$, and $6 \approx_{Bs} 9$. ∘

The *forward and backward simulation* ($FBsimulation$) of G_1 by G_2 deals with both the incoming and the outgoing arcs at a vertex. For a vertex v, let $pre(v)$ denote the set of vertices from which there is an arc to v. FBsimulation is the largest binary relation $\preceq_{FBs} \subseteq V_1 \times V_2$, such that the following holds: If $v_1 \preceq_{FBs} v_2$, then

- Preserve vertex types: $\tau(v_1) = \tau(v_2)$.
- Preserve outgoing arcs: For each $v_1' \in post(v_1)$, there exists $v_2' \in post(v_2)$ such that $v_1' \preceq_{FBs} v_2'$.
- Preserve incoming arcs: For each $v_1' \in pre(v_1)$, there exists $v_2' \in pre(v_2)$ such that $v_1' \preceq_{FBs} v_2'$.

For $v_1 \in V_1$, let $FBsim(v_1) \subseteq V_2$ denote the set of all FBsimulators of v_1. For FBsimulation of a graph by itself, $FBsimilar$ (denoted by $\approx_{FBs}$) is analogous to Fsimilar; it is an equivalence relation.

For trees, FBsimulation can be computed by first computing Fsimulation bottom-up, and then using it

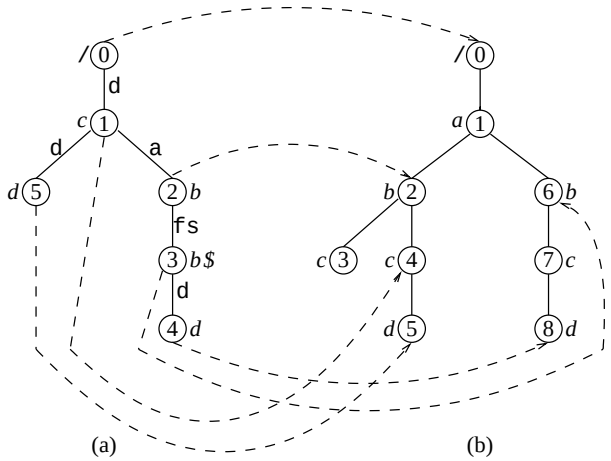

Figure 4: (a). A $CXPath^+$ Query Q, (b). An XML document D, and an Embedding β of Q in D

in the computation of FBsimulation top-down. This is explained further in Section 4. For the tree in Figure 2a, FBsimulation is identical to Fsimulation that we saw in Example 3.1.

Bloom and Paige [4] and Henzinger et al. [8] presented an $O(|G_1||G_2|)$ algorithm for computing the Fsimulation relation between two graphs. Their algorithm can be easily modified to compute the Bsimulation and FBsimulation relations between two graphs, without changing the runtime.

4 Query Evaluation thru Simulation

In this section, we extend the definition of simulation (for ordinary graphs), to simulation of a $CXPath^+$ query $Q = (V, A)$ by a document $D = (N, E, E_{ref})$. We then show that Q can be evaluated on D by computing the FBsimulation of Q by D. This result is used in Section 6, to prove our main result.

This result is also of independent interest, as it leads to an $O(|Q||D|)$ time in-memory algorithm for evaluating Q on D; the constant factors in the O notation are small. Previously, Gottlob et al. [7] presented an $O(|Q||D|)$-time in-memory algorithm; their algorithm is based on formal semantics, and does not involve simulation. They also pointed out that three commercially available XPath query engines (XALAN, XT and Microsoft Internet Explorer 6) take exponential time ($O(|D|^{|Q|})$) to evaluate a $CXPath^+$ query.

For a vertex z in Q, let Q_z denote the subtree of Q consisting of z and all its descendants. For a node n in D, and an `axis` (one of thirteen discussed earlier), let `axis`$[n]$ denote the set of nodes in D that bear the `axis` relationship to n. For example, `s`$[n] = \{n\}$, `c`$[n]$ is the set of all children of n, and so on; `ir`$[n]$ is the set of all nodes reachable from n by following a single `idref` edge. For $N' \subseteq N$, let `axis`$[N'] = \cup_{n' \in N'}$`axis`$[n']$. Recall that, for an arc r in Q, $axis(r)$ is the `axis` of r; let $axis(r)[n]$ and $axis(r)[N']$ denote `axis`$[n]$ and `axis`$[N']$, respectively.

An *embedding* of Q_z in D is a *partial* mapping $\beta : Q_z \rightarrow D$, from the vertices of Q_z to the nodes of D, that satisfies the following conditions:

1. $\beta(z)$ is defined.
2. Preserve vertex types: For each vertex v in Q_z such that $\beta(v)$ is defined:
   - If $\tau(v) = /$, then $\beta(v) = root(D)$.
   - If $\tau(v) \in \Sigma$, then $\tau(\beta(v)) = \tau(v)$.
3. Preserve boolean vertex labels and outgoing arc labels: For each vertex v in Q_z such that $\beta(v)$ is defined, consider two cases depending on $bool(v)$.
   (a) $Bool(v) =$ `and`: We require that for *each* arc $r = (v, v')$, $\beta(v') \in axis(r)[\beta(v)]$.
   (b) $Bool(v) =$ `or`: We require that for *some* arc $r = (v, v')$, $\beta(v') \in axis(r)[\beta(v)]$.

β is a *partial* mapping because, in para 3(b), $\beta(v')$ might not be defined for some vertices v' in Q_z.

Let $Q(D, S)$ denote the result of evaluating Q on D, for a given original context node set (cns) S. $Q(D)$ corresponds to the case when Q is an absolute query; then $\tau(root(Q)) = /$, and $S = \{root(D)\}$. Computing $Q(D, S)$ requires finding all embeddings β of Q in D such that $\beta(root(Q)) \in S$. We have:
$Q(D, S) = \{\beta(opv(Q)) \mid \beta$ is an embedding of Q in D such that $\beta(root(Q)) \in S\}$.

Example 4.1. In Figure 4, we show an embedding β of a $CXPath^+$ query Q in an XML document D. Since this is the only possible embedding, $Q(D) = \{\beta(opv(Q))\} = \{\beta(3)\} = \{6\}$. ∘

We show that $Q(D, S)$ can be computed using the FBsimulation of Q by D. First, we need to redefine the concept of simulation to account for the presence of boolean labels $bool(v)$ (in addition to $\tau(v)$) on the vertices in Q, and the labels $axis(r)$ on the arcs in Q. We define the *forward simulation* (abbreviated as *Fsimulation*) of Q by D to be the largest binary relation $\preceq_{Fs} \subseteq V \times N$ such that the following holds: If $v \preceq_{Fs} n$, then

1. Preserve vertex types:
   - If $\tau(v) = /$, then $n = root(D)$.
   - If $\tau(v) \in \Sigma$, then $\tau(n) = \tau(v)$.
2. Preserve boolean vertex labels and outgoing arc labels: Consider two cases depending on $bool(v)$ (see Figure 5).
   (a) $Bool(v) =$ `and`: For *each* arc $r = (v, v')$, there exists a node $n' \in axis(r)[n]$ such that $v' \preceq_{Fs} n'$.

Figure 5: Definition of Simulation: $v \in Q$, $n \in D$

(b) $Bool(v) =$ `or`: For *some* arc $r = (v, v')$, there exists a node $n' \in axis(r)[n]$ such that $v' \preceq_{Fs} n'$.

For $v \in V$, let $Fsim(v) \subseteq N$ denote the set of all Fsimulators of v.

The above definition combines the features from the definition of Fsimulation for ordinary graphs (Section 3) and the definition of embedding given above.

Example 4.2. For (Q, D) in Figure 4, we have $Fsim(4) = Fsim(5) = \{5, 8\}$, $Fsim(3) = \{2, 6\}$, $Fsim(2) = \{2\}$, $Fsim(1) = \{4\}$ and $Fsim(0) = \{0\}$.

Fsimulation can be computed in $O(|Q||D|)$ time, by proceeding bottom up in Q. Concerning the significance of Fsimulation, we have the following.

Lemma 4.1. Consider the Fsimulation of Q by D. There exists an embedding β of Q_v in D with $\beta(v) = n$ iff $n \in Fsim(v)$.

Note that $Fsim(v)$ is completely determined by Q_v and D; so it is independent of

- the cns S.
- the vertices and arcs in Q that are outside of Q_v, and how they can be embedded in D.

Recall that $Q(D, S)$ consists of $\beta(opv(Q))$ for those embeddings β of the entire query Q in D, such that $\beta(root(Q)) \in S$; clearly, $Q(D, S) \subseteq Fsim(opv(Q))$. The two restrictions listed above can be transmitted downwards in Q by adding backward simulation to Fsimulation. The *forward and backward simulation* (abbreviated as $FBsimulation$) of Q by D is the largest binary relation $\preceq_{FBs} \subseteq V \times N$, such that the following holds: If $v \preceq_{FBs} n$, then

1. Preserve vertex types:
   - If $v = root(Q)$, then $n \in S$.
   - If $\tau(v) \in \Sigma$, then $\tau(n) = \tau(v)$.
2. Preserve boolean vertex labels and outgoing arc labels: Consider two cases depending on $bool(v)$ (see Figure 5).
   (a) $Bool(v) =$ `and`: For *each* arc $r = (v, v')$, there exists $n' \in axis(r)[n]$ such that $v' \preceq_{FBs} n'$.
   (b) $Bool(v) =$ `or`: For *some* arc $r = (v, v')$, there exists $n' \in axis(r)[n]$ such that $v' \preceq_{FBs} n'$.
3. Preserve incoming arc label: If $v \neq root(Q)$, let v' be the parent of v in Q; let $r = (v', v)$. Then there exists a node $n' \in N$ such that $v' \preceq_{FBs} n'$, and $n \in axis(r)[n']$. This should hold independent of $bool(v')$.

For $v \in V$, let $FBsim(v) \subseteq N$ denote the set of all FBsimulators of v. Concerning the significance of FBsimulation, we have the following.

Lemma 4.2. Consider the FBsimulation of Q by D. For any vertex $v \in V$, there exists an embedding β of Q in D with $\beta(root(Q)) \in S$ and $\beta(v) = n$, iff $n \in FBsim(v)$.

Specializing this lemma for $v = opv(Q)$, we have the following.

Theorem 4.3. Consider the FBsimulation of Q by D. $Q(D, S) = FBsim(opv(Q))$.

Since Q is a tree, FBsimulation can be computed by first computing Fsimulation, and then adding backward simulation, as follows:

1. First compute $Fsim(v)$ for all $v \in V$. This can be done bottom-up in Q.
2. Set $FBsim(root(Q)) = Fsim(root(Q)) \cap S$.
3. Add backward simulation top-down in Q, starting from $root(Q)$, as follows. Suppose that for some vertex $v \in V$, we have computed $FBsim(v)$. Let v' be a child of v, and $r = (v, v')$ be the arc from v to v' (see Figure 5). Set $FBsim(v') = Fsim(v') \cap axis(r)[FBsim(v)]$.

Step 1) can be performed in $O(|Q||D|)$ time. For any $N' \subseteq N$ and `axis`, `axis`$[N']$ can be computed in $O(|D|)$ time. So, for each vertex v' in step 3), $FBsim(v')$ can be computed in $O(|D|)$ time; overall, step 3) takes $O(|Q||D|)$ time. So, the above algorithm runs in $O(|Q||D|)$ time.

Example 4.3. For (Q, D) in Figure 4, we computed the Fsimulation of Q by D in Example 4.2. Using the above procedure, we now compute FBsimulation. We have $FBsim(0) = \{0\}$, $FBsim(1) = \{4\}$, $FBsim(5) = \{5\}$, $FBsim(2) = \{2\}$, $FBsim(3) = \{6\}$ and $FBsim(4) = \{8\}$; $Q(D) = FBsim(opv(Q)) = FBsim(3) = \{6\}$. ∘

We have the following result.

Theorem 4.4. A $CXPath^+$ query Q can be evaluated on an XML document D by computing the FBsimulation of Q by D, in $O(|Q||D|)$ time.

5 Simulation, Bisimulation and Quotients of D

In this section, we define the simulation and bisimulation relations on an XML document $D = (N, E, E_{ref})$,

and also define the resulting quotient graphs; these definitions are used in Section 6. Then we show that the simulation quotient could be exponentially smaller than the bisimulation quotient. Simulation and bisimulation are *binary relations* on N.

The Simulation Relation

For simulation, we need to modify the definitions from Section 3, to account for the special root node and the presence of two kinds of edges in D. We define the *Fsimulation* of D to be the largest binary relation $\preceq_{Fs}$ on N such that the following holds: If $n_1 \preceq_{Fs} n_2$, then

- Preserve node types:
  - If $n_1 = root(D)$, then $n_2 = root(D)$
  - Else $\tau(n_2) = \tau(n_1)$.
- Preserve outgoing tree edges: For each tree edge (n_1, n_1'), there exists a tree edge (n_2, n_2') such that $n_1' \preceq_{Fs} n_2'$.
- Preserve outgoing `idref` edges: For each `idref` edge (n_1, n_1'), there exists an `idref` edge (n_2, n_2') such that $n_1' \preceq_{Fs} n_2'$.

As in the case of ordinary graphs, Fsimulation of D is reflexive and transitive, but it may not be symmetric. Nodes n_1 and n_2 are said to be *Fsimilar*, denoted by $n_1 \approx_{Fs} n_2$, if $n_1 \preceq_{Fs} n_2$ and $n_2 \preceq_{Fs} n_1$; Fsimilarity is an equivalence relation.

Example 5.1. Consider D in Figure 2a. Since D does not have any `idref` edges, the Fsimulation relation is same as the one we saw in Example 3.1: The only nontrivial relational pairs are $6 \approx_{Fs} 9$, $3 \preceq_{Fs} 5$, $3 \preceq_{Fs} 8$, $5 \approx_{Fs} 8$, and $2 \approx_{Fs} 7$.

If we add an `idref` edge from node 7 to node 5 in Figure 2a, the new Fsimulation relation will differ from the one above only in that $2 \preceq_{Fs} 7$, but $7 \not\preceq_{Fs} 2$.

Bsimulation and *Bsimilar* are analogous to Fsimulation and Fsimilar, respectively; they deal with the *incoming* tree and `idref` edges at a node, as opposed to Fsimulation that deals with the *outgoing* edges.

The *FBsimulation* of D deals with both the incoming and the outgoing edges at a node. It is the largest binary relation $\preceq_{FBs}$ on N such that the following holds: If $n_1 \preceq_{FBs} n_2$, then

- Preserve node types:
  - If $n_1 = root(D)$, then $n_2 = root(D)$
  - Else $\tau(n_2) = \tau(n_1)$.
- Preserve outgoing tree edges: For each tree edge (n_1, n_1'), there exists a tree edge (n_2, n_2') such that $n_1' \preceq_{FBs} n_2'$.
- Preserve outgoing `idref` edges: For each `idref` edge (n_1, n_1'), there exists an `idref` edge (n_2, n_2') such that $n_1' \preceq_{FBs} n_2'$.
- Preserve incoming tree edges: For each tree edge (n_1', n_1), there exists a tree edge (n_2', n_2) such that $n_1' \preceq_{FBs} n_2'$.
- Preserve incoming `idref` edges: For each `idref` edge (n_1', n_1), there exists an `idref` edge (n_2', n_2) such that $n_1' \preceq_{FBs} n_2'$.

FBsimilar is analogous to Fsimilar; it is an equivalence relation.

Example 5.2. In Example 5.1 above, we considered the Fsimulation relation for D in Figure 2a; for this D, FBsimulation turns out to be same as Fsimulation.

Now, consider adding an `idref` edge from node 7 to node 5 in Figure 2a. In Example 5.1, we saw that this caused only a small change in Fsimulation; but it causes a substantial change in FBsimulation. The only nontrivial relational pair is $3 \preceq_{FBs} 5$. $5 \not\preceq_{FBs} 8$ because 5 has an incoming `idref` edge, whereas 8 doesn't. As a consequence, $6 \not\preceq_{FBs} 9$, $2 \not\preceq_{FBs} 7$, and $3 \not\preceq_{FBs} 8$. $7 \not\preceq_{FBs} 2$ because 7 has an outgoing `idref` edge, whereas 2 doesn't. As a consequence, $8 \not\preceq_{FBs} 5$ and $9 \not\preceq_{FBs} 6$. ∘

In the absence of `idref` edges, FBsimulation can be computed as described in Section 4: First compute Fsimulation bottom-up, then add Bsimulation top-down. In the presence of `idref` edges, computation of FBsimulation is quite complicated, as seen from the preceding example. In all cases, the algorithms of Bloom and Paige [4] and Henzinger et al. [8] can be used to compute the Fsimulation, Bsimulation and FBsimulation relations in $O(|N|^2 + |N||E_{ref}|)$ time.

The Bisimulation Relation

For ordinary graphs, bisimulation was defined in [13] (also see [1]); it provides another notion of equivalence between the nodes. We define the *forward bisimulation* (abbreviated as *Fbisimulation*) of D to be the largest binary relation $\approx_{Fbi}$ on N such that the following holds: If $n_1 \approx_{Fbi} n_2$, then

- Preserve node types:
  - If $n_1 = root(D)$, then $n_2 = root(D)$; and vice versa
  - Else $\tau(n_2) = \tau(n_1)$.
- Preserve outgoing tree edges: For each tree edge (n_1, n_1'), there exists a tree edge (n_2, n_2') such that $n_1' \approx_{Fbi} n_2'$; and vice versa.
- Preserve outgoing `idref` edges: For each `idref` edge (n_1, n_1'), there exists an `idref` edge (n_2, n_2') such that $n_1' \approx_{Fbi} n_2'$; and vice versa.

Fbisimulation is reflexive, symmetric and transitive; so, it is an equivalence relation. If $n_1 \approx_{Fbi} n_2$, we say that n_1 and n_2 are *Fbisimilar*. Let $Fbisim(n) \subseteq N$ denote the set of all nodes that are Fbisimilar to n.

Example 5.3. For D in Figure 1a, the following nodes are bisimilar: $(4, 6, 9)$, $(3, 5, 8)$ and $(2, 7)$. For this D, Fsimulation is identical to Fbisimulation. If we add an `idref` edge from node 7 to node 5, then nodes 2 and 7 would not be Fbisimilar; for Fsimulation, we would have $2 \preceq_{Fs} 7$, but $7 \not\preceq_{Fs} 2$.

For D in Figure 2a, we saw Fsimulation in Example 5.1. The only Fbisimilar pairs are $(6, 9)$ and $(5, 8)$; $3 \not\approx_{Fbi} 8$ and $2 \not\approx_{Fbi} 7$. ∘

Backward bisimulation and *forward and backward bisimulation* (abbreviated as *Bbisimulation* and *FBbisimulation*) are defined analogously [11, 1, 9]. We let $n_1 \approx_{Bbi} n_2$ and $n_1 \approx_{FBbi} n_2$ denote that n_1 and n_2 are Bbisimilar and FBbisimilar, respectively.

A clue about our abbreviations and subscripts: Capital letters F and B are used only for directions (Forward and Backward); in the subscripts, s is used for simulation (or similar), and bi is used for bisimulation (or bisimilar).

Example 5.4. For D in Figure 1a, FBbisimulation (also FBsimulation) is same as Fbisimulation. If we add an `idref` edge from node 7 to node 5, then no two nodes would be FBbisimilar. For D in Figure 2a, no two nodes are FBbisimilar; compare this to the FBsimilar relation in Example 5.2. ∘

In the absence of `idref` edges, FBbisimulation can be computed in the same manner as FBsimulation: First compute Fbisimulation bottom-up, then add Bbisimulation top-down. In the presence of `idref` edges, computation of FBbisimulation is quite complicated. In all cases, the algorithm of Paige and Tarjan [12] can be used to compute the bisimulation relations in $O(|D| \log |D|)$ time.

The Quotients

An equivalence relation $\approx$ on N partitions N into equivalence classes: Any two nodes in the same class are related, and two nodes from different classes are not related. For $n \in N$, let $n_{\approx} \subseteq N$ denote the equivalence class containing n. Note that if $m \approx n$, then $m_{\approx} = n_{\approx}$. The *quotient graph* $D_{\approx}$ is obtained from D by merging the nodes of each equivalence class into a single node.

In the following sections, we let D_{Fs}, D_{Bs}, D_{FBs}, D_{Fbi}, D_{Bbi} and D_{FBbi} denote the quotient graphs of D corresponding to the equivalence relations $\approx_{Fs}$, $\approx_{Bs}$, $\approx_{FBs}$, $\approx_{Fbi}$, $\approx_{Bbi}$ and $\approx_{FBbi}$, respectively. D_{Fs}, D_{Bs} and D_{FBs} will be called the Fsimulation, Bsimulation, and FBsimulation quotients, respectively. D_{Fbi}, D_{Bbi} and D_{FBbi} will be called the Fbisimulation, Bbisimulation, and FBbisimulation quotients, respectively.

Example 5.5. For D in Figure 1a, D_{FBbi} is shown in Figure 1b; D_{FBs} is same as D_{FBbi}. For D in Figure 2a, D_{FBs} is shown in Figure 2b; D_{FBbi} is same as D.

Bisimulation is a Refinement

Let $\approx_1$ and $\approx_2$ be two equivalence relations on N. We say that $\approx_2$ is a refinement of $\approx_1$ if, for any pair of nodes $m, n \in N$, whenever $m \approx_2 n$ holds, $m \approx_1 n$ also holds. So, $\approx_2$ is a refinement of $\approx_1$, if each equivalence class of $\approx_2$ is contained in some equivalence class of $\approx_1$. It is easily seen that if $m \approx_{Fbi} n$, then $m \approx_{Fs} n$; so, $\approx_{Fbi}$ is a refinement of $\approx_{Fs}$. Similarly, $\approx_{Bbi}$ is a refinement of $\approx_{Bs}$, and $\approx_{FBbi}$ is a refinement of $\approx_{FBs}$.

Since $\approx_{Fbi}$ is a refinement of $\approx_{Fs}$, D_{Fbi} will have more nodes compared to D_{Fs}; in fact, $D_{Fs} = (D_{Fbi})_{Fs}$. Since computing D_{Fs} is more expensive than computing D_{Fbi}, we can speed up the computation of D_{Fs} by first computing D_{Fbi} ($|D_{Fbi}| \leq |D|$), and then computing *its* simulation quotient. A similar statement holds for D_{Bs} and D_{FBs}.

The Difference Between Sim and Bisim

To get an intuitive feel, let us consider the difference in the absence of `idref` edges. Then, there is no difference between Bsimilar and Bbisimilar: $m \approx_{Bs} n$ iff $m \approx_{Bbi} n$. Two nodes are Bsimilar (or Bbisimilar) iff the sequences of node types on the path from the root to the two nodes are identical. But, there is a big difference between Fsimulation and Fbisimulation (and consequently between FBsimulation and FBbisimulation). We present instances where the FBsimulation quotient is exponentially smaller than the FBbisimulation quotient.

Two nodes are Fbisimilar iff the trees rooted at the two nodes are identical except for duplicate subtrees. For example, in Figure 1a, the subtrees rooted at nodes 3 and 5 are duplicates; each is identical to the subtree at node 8; so, nodes 2 and 7 are Fbisimilar. The duplication can be at any level: If another child labeled d is added to node 3, 5 or 8, nodes 2 and 7 would still be Fbisimilar. In Figure 2a, the subtree at node 5 is identical to that at node 8; but the subtree at node 3 is not a duplicate; so nodes 2 and 7 are not Fbisimilar.

For simulation, $m \preceq_{Fs} n$ iff the tree rooted at m can be obtained from the tree at n by duplicating subtrees and/or dropping subtrees at any level. For example, in Figure 2a, the tree at node 2 can be obtained from the tree at node 7 as follows: Duplicate the subtree at node 8, then drop the subtree at the node labeled d from one of the copies. So, $2 \preceq_{Fs} 7$. Conversely, the tree at node 7 can be obtained from the tree at node 2 as follows: Drop the subtree at node 3. So, $7 \preceq_{Fs} 2$.

Now, we present document instances where the FBsimulation quotient is exponentially smaller than the FBbisimulation quotient. For each positive integer k, we present an XML document D whose the bisimulation quotient $D_{FBbi} = D$ has more than 2^{2^k-1} nodes, but the simulation quotient D_{FBs} has only about $k2^k$ nodes. Figures 2a and 6a correspond to the cases $k = 1, 2$, respectively. We define the document D as follows. Let the alphabet Σ consist of types $\{a, b, c\}$

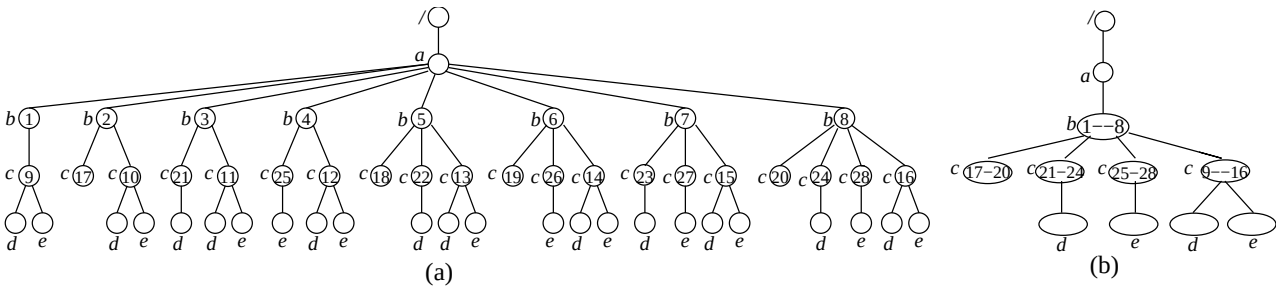

Figure 6: (a). An XML Document D (and D_{FBbi}). (b). Its Simulation Quotient D_{FBs}.

and k other *special* types (d and e in Figure 6). A *full* c is a node labeled c with k children labeled by the special types (nodes 9–16 in Figure 6a). A *partial* c is a node labeled c with $i < k$ children labeled by i distinct special types (nodes 17–28 in Figure 6a). There are $2^k - 1$ partial c's, including a c with no children. A b-node is a node labeled b that has one full c and a *set* of partial c's as children. There are 2^{2^k-1} distinct b-nodes. Document D has root labeled / with a child labeled a. This node labeled a has the distinct b-nodes as children. All the b nodes in D are Bbisimilar. No two b-nodes are Fbisimilar; so no two are FBbisimilar; the FBbisimulation quotient of D is D itself. All the b nodes are Fsimilar and so FBsimilar. The FBsimulation quotient D_{FBs} has only one b node, with one full c and all the $2^k - 1$ partial c's as children (see Figures 2b and 6b, for $k = 1, 2$); $|D_{FBs}| < k2^k + 3$. This leads to the following.

Theorem 5.1. There exist large XML documents with FBbisimulation quotients exponentially larger than FBsimulation quotients.

6 FBsimulation Quotient is the Smallest Covering Index for BPQ+

Let $D = (N, E, E_{ref})$ be an XML document. Kaushik et al. [9] showed that D_{FBbi}, the FB**bi**simulation quotient of D, is the smallest covering index for BPQ. We show that D_{FBs}, the FBsimulation quotient of D, is the smallest covering index for BPQ^+.

Let $\approx$ be an equivalence relation on N. Recall (from Section 1) that $D_{\approx}$ is a *covering index* for a class $C \subseteq CXPath$ of queries, if the following holds: No absolute query $Q \in C$ can distinguish between two nodes of D in the same equivalence class; i.e., $Q(D)$ is the union of some of the equivalence classes.

Now, we show that D_{FBs} is *a* covering index for BPQ^+; the proof is completely different from that for the analogous result (that D_{FBbi} is a covering index for BPQ) in [9].

Lemma 6.1. D_{FBs} is a covering index for BPQ^+.

Proof. Let $Q = (V, A) \in BPQ^+$; let $m, n \in N$ and $m \approx_{FBs} n$. We need to show that Q can not distinguish between m and n; i.e., either both m and n are in $Q(D)$, or neither is in $Q(D)$. Let $m \in Q(D)$; we will show that $n \in Q(D)$. In our proof, we consider two different kinds of simulations; simulation of Q by D, defined in Section 4, and simulation of D by D, defined in Section 5; we differentiate between them using the initials Q and D, respectively, prepended to the relation names. Since $m \in Q(D)$, we have (from Theorem 4.3) that $opv(Q) \preceq_{QFBs} m$. Summarizing the above, we need to prove the following: If $opv(Q) \preceq_{QFBs} m$ and $m \approx_{DFBs} n$, then $opv(Q) \preceq_{QFBs} n$. We will prove the following more general result: For $v \in V$ and $n_1, n_2 \in N$,

$$\text{If } v \preceq_{QFBs} n_1 \text{ and } n_1 \preceq_{DFBs} n_2, \text{ then } v \preceq_{QFBs} n_2 \quad (1)$$

This looks similar to the well-known transitivity result for FBsimulation of ordinary graphs. But things are more complicated here: If B is dropped from all the subscripts in Statement (1), the resulting statement is *not* true (while the transitivity result holds for Fsimulation of ordinary graphs).

We prove Statement (1) in two parts.

Part I. We first prove the following

$$\text{If } v \preceq_{QFs} n_1 \text{ and } n_1 \preceq_{DFBs} n_2, \text{ then } v \preceq_{QFs} n_2 \quad (2)$$

We prove the statement by induction on $height(v)$. The *height* of a vertex in Q is the length of the longest path from that vertex to a leaf; a leaf has height 0, and $root(Q)$ has the largest height.

To prove that $v \preceq_{QFs} n_2$, we need to prove that two things are preserved: a). vertex type $\tau(v)$,
and b). boolean vertex label $bool(v)$ and arc labels outgoing from v.

The proof pertaining to the preservation of $\tau(v)$ is the same for the base case as well as for all the subcases of the induction step; so we present it here. We need to separately consider two cases: $\tau(v) =$ / and $\tau(v) \in \Sigma$. If $\tau(v) =$ /, then $n_1 = root(D)$; since $n_1 \preceq_{DFBs} n_2$, $n_2 = root(D)$; so, we are done. Else if $\tau(v) \in \Sigma$, then $\tau(n_1) = \tau(v)$; since $n_1 \preceq_{DFBs} n_2$, $\tau(n_2) = \tau(n_1) = \tau(v)$; so, we are done. Now, we prove the preservation of b).

Base Case. $Height(v) = 0$; i.e., v is a leaf in Q. Since v has no outgoing arcs, b) is preserved.

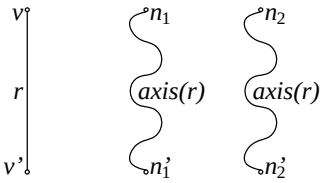

Figure 7: Proof of Lemma 6.1: $v \in Q$, $n_1, n_2 \in D$

Induction Step. The two values for $bool(v)$ (`and` and `or`) can be handled together. Consider an arc $r = (v, v')$ outgoing from v. Suppose that there exists a node $n_1' \in axis(r)[n_1]$ such that $v' \preceq_{QFs} n_1'$. We will show that there exists a node $n_2' \in axis(r)[n_2]$ such that $n_1' \preceq_{DFBs} n_2'$ (see Figure 7). Then, by induction hypothesis (since $height(v') < height(v)$), it would follow that $v' \preceq_{QFs} n_2'$. From this we can conclude that b) is preserved. We need to consider nine cases, depending on the value of $axis(r)$.
$axis(r) = \mathtt{s}$. n_1' must be n_1; we take n_2' to be n_2.
$axis(r) \in \{\mathtt{c}, \mathtt{pa}, \mathtt{ir}, \mathtt{rir}\}$. Existence of n_2' follows from the definition of $DFBsimulation$ (along with $n_1 \preceq_{DFBs} n_2$).
$axis(r) = \mathtt{d}$. $n_1' \in \mathtt{d}[n_1]$; let $n_{1,0} = n_1, n_{1,1}, n_{1,2}, \ldots, n_{1,k} = n_1'$ be the path in $T = (N, E)$ from n_1 to n_1'. Since $n_1 \preceq_{DFBs} n_2$, n_2 must have a child $n_{2,1}$ such that $n_{1,1} \preceq_{DFBs} n_{2,1}$; this in turn implies that $n_{2,1}$ must have a child $n_{2,2}$ such that $n_{1,2} \preceq_{DFBs} n_{2,2}$; and so on. So, n_2 must have a descendant n_2' such that $n_1' \preceq_{DFBs} n_2'$.
$axis(r) = \mathtt{a}$. Similar to the previous case, except that we use the fact that $n_1 \preceq_{DFBs} n_2$, to get a sequence of parents (and hence an ancestor).
$axis(r) = \mathtt{ds}$. Follows from the cases for `s` and `d`.
$axis(r) = \mathtt{as}$. Follows from the cases for `s` and `a`.

Part II. Now, we prove Statement (1), using (2). The proof is by induction on $depth(v)$. The *depth* of a vertex in Q is the length of the unique path in Q from $root(Q)$ to that vertex; $root(Q)$ has depth 0.

Consider the algorithm we described for computing QFBsimulation, following Theorem 4.3. This algorithm first computes QFsimulation, then adds backward simulation top-down, starting from $root(Q)$.
Base Case. $Depth(v) = 0$; so $v = root(Q)$. Since Q is an absolute query, $\tau(v) = /$. From $v \preceq_{QFBs} n_1$, we have $n_1 = root(D)$; since $n_1 \preceq_{DFBs} n_2$, we have $n_2 = root(D) = n_1$. So, $v \preceq_{QFBs} n_2$.
Induction Step. From Part I, we have $v \preceq_{QFs} n_2$. Consider the unique arc $r = (v', v)$ incoming at v. Since $v \preceq_{QFBs} n_1$, there exists a node n_1' such that $v' \preceq_{QFBs} n_1'$ and $n_1 \in axis(r)[n_1']$. We need to show that there exists a node n_2', such that

$$n_1' \preceq_{DFBs} n_2' \text{ and } n_2 \in axis(r)[n_2'].$$

Then, by induction hypothesis (since $depth(v') < depth(v)$), it would follow that $v' \preceq_{QFBs} n_2'$. Then, from the equation $QFBsim(v) = QFsim(v) \cap axis(r)[QFBsim(v')]$, we can conclude that $n_2 \in QFBsim(v)$. Proving the existence of a node n_2' as required above is similar to the proof in Part I, and is omitted. □

Due to lack of space, we refer the reader to [15], for proof that D_{FBs} is the smallest among all covering indexes for BPQ^+. This, together with the above lemma, leads to the following.

Theorem 6.2. D_{FBs} is the smallest covering index for BPQ^+.

7 Smallest Covering Index for TPQ

Let TPQ^- be the class TPQ augmented with the boolean operator `not`; we have $TPQ = TPQ^+ \subset TPQ^- \subset BPQ$. In this section, we present the smallest covering indexes for TPQ and TPQ^-.

Recall that TPQ queries do not involve `idref` edges. For an XML document $D = (N, E, E_{ref})$, consider the tree $T = (N, E)$. Let T_{FBs} be the FBsimulation quotient of T. By Lemma 6.1, T_{FBs} is *a* covering index for TPQ.

We show that it is the *smallest* such covering index. For each node $n \in N$, we show how to construct a query $Q_n' \in TPQ$, such that $Q_n'(D) = Q_n'(T) = FBsim(n)$ (in T). The query tree Q_n' is constructed from T_{FBs} by setting the boolean label of each node to `and`, and $opv(Q_n')$ to $[n_{FBs}]$.

Example 7.1. For T in Figure 2a, T_{FBs} is shown in Figure 2b. For node 5, $Q_5' = /a/b\,[c]/c\,[d]$. ∘

We have the following.

Lemma 7.1. $Q_n'(T) = FBsim(n)$.

Theorem 7.2. For an XML document D, T_{FBs} is the smallest covering index for TPQ.

Now, consider the smallest covering index for TPQ^-. By Kaushik et al.'s result, T_{FBbi} is *a* covering index for this class. We show that it is the *smallest* such covering index. For each node $n \in N$, we show how to construct a query $\hat{Q}_n \in TPQ^-$, such that $\hat{Q}_n(D) = \hat{Q}_n(T) = FBbisim(n)$ (in T). The query tree $\hat{Q}_n$ is constructed from T_{FBbi} as follows:

- Set the boolean label of each node in T_{FBbi} to `and`.
- For each node m in T_{FBbi} and each type $\tau \in \Sigma$ such that m does not have a child node of type τ, add the predicate `[not c::`τ`]` to node m.
- Set $opv(\hat{Q}_n)$ to $[n_{FBbi}]$.

Example 7.2. For T in Figure 2a, T_{FBbi} is T itself. For node 5, $\hat{Q}_5 = /a\,[\mathtt{not}\ a]\,[\mathtt{not}\ c]\,[\mathtt{not}\ d]$
$/b\,[\mathtt{not}\ a]\,[\mathtt{not}\ b]\,[\mathtt{not}\ d]\,[c\,[\mathtt{not}\ *\,]]$
$/c\,[\mathtt{not}\ a]\,[\mathtt{not}\ b]\,[\mathtt{not}\ c]\,[d\,[\mathtt{not}\ *\,]]$.
Compare this to the query Q_5' in Example 7.1. ∘

We have the following.

Lemma 7.3. $\hat{Q}_n(T) = FBbisim(n)$.

Theorem 7.4. For an XML document D, T_{FBbi} is the smallest covering index for TPQ^-.

8 Conclusions

Tree Pattern Queries (TPQ), Branching Path Queries (BPQ) and Core XPath (CXPath) are important subclasses of XPath, $TPQ \subset BPQ \subset CXPath \subset XPath$. Let $TPQ = TPQ^+ \subset BPQ^+ \subset CXPath^+ \subset XPath^+$ denote the corresponding subclasses consisting of queries that do not involve negation. Simulation and bisimulation are two different binary relations on graph vertices that have previously been studied in connection with some of these classes. Ramanan [14] showed that TPQ queries, without wildcard $*$ for node types, can be minimized using *simulation*. Kaushik et al. [9] showed that, for an XML document, its **bi***simulation* quotient is the smallest covering index for BPQ.

In this paper, we further extended the application of *simulation* to the evaluation of $CXPath^+$ queries and the indexing of XML documents to answer such queries. We showed the following:

- A $CXPath^+$ query Q can be evaluated on an XML document D by computing the *simulation* of Q by D.
- For an XML document, its *simulation* quotient is the smallest covering index for BPQ^+.
- For an XML document, its *simulation* quotient, with the `idref` edges ignored throughout, is the smallest covering index for TPQ.

For any XML document, its simulation quotient is no larger than its bisimulation quotient. We showed that, in some cases, the simulation quotient is exponentially smaller. So, our latter two results give smaller covering indexes for two important subclasses of queries.

Our three results above can be easily extended to queries that consider `text` elements and string values, by appropriately modifying the definition of simulation: Two `text` nodes or string values are similar iff they are identical.

9 Acknowledgements

The author would like to thank Phil Bohannon and Raghav Kaushik for information concerning the sizes of simulation and bisimulation quotients.

References

[1] S. Abiteboul, P. Buneman and D. Suciu. *Data on the Web*. Morgan Kaufman, San Francisco, CA.

[2] S. Amer-Yahia, SR. Cho, L. V. S. Lakshmanan and D. Srivastava. Minimization of Tree Pattern Queries, *Proc. ACM SIGMOD Intl. Conf. Management of Data*, 2001, pp. 497–508.

[3] A. Berglund, S. Boag, D. Chamberlin, M. Fernandez, M. Kay, J. Robie and J. Simeon. XML Path Language (XPath) 2.0, www.w3.org/TR/xpath20.

[4] B. Bloom and R. Paige. Transformational Design and Implementation of a New Efficient Solution to the Ready Simulation Problem, *Science of Computer Programming* **24**(1995), pp. 189–220.

[5] P. Buneman, S. Davidson, M. Fernandez and D. Suciu. Adding Structure to Unstructured Data, *Intl. Conf. Database Theory*, 1997, pp. 336–350.

[6] D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Simeon and M. Stefanescu. XQuery 1.0: An XML Query Language, www.w3.org/TR/xquery.

[7] G. Gottlob, C. Koch and R. Pichler. Efficient Algorithms for Processing XPath Queries, 28th *Intl. Conf. Very Large Data Bases* (VLDB), 2002.

[8] M. R. Henzinger, T. A. Henzinger and P. W. Kopke. Computing Simulations on Finite and Infinite Graphs, *Proc. IEEE Symp. Foundations of Computer Science*, 1995, pp. 453–462.

[9] R. Kaushik, P. Bohannon, J. F. Naughton and H. F. Korth. Covering Indexes for Branching Path Queries, *Proc. ACM SIGMOD Intl. Conf. Management of Data*, 2002, pp. 133–144.

[10] R. Milner. *A Calculus for Communicating Processes*. LNCS, Vol. 92, Springer–Verlag, NY, 1980.

[11] T. Milo and D. Suciu. Index Structures for Path Expressions, *Proc.* 7th *Intl. Conf. Database Theory*, 1999, pp. 277–295.

[12] R. Paige and R. E. Tarjan. Three Partition Refinement Algorithms, *SIAM J. Comput.* **16** (1987), pp. 973–989.

[13] D. Park. Concurrency and Automata on Infinite Sequences, *Proc.* 5th *GI-Conf.*, LNCS, Vol. 104, Springer–Verlag, NY, 1981.

[14] P. Ramanan. Efficient Algorithms for Minimizing Tree Pattern Queries, *Proc. ACM SIGMOD Intl. Conf. Management of Data*, 2002, pp. 299–309.

[15] P. Ramanan. Covering Indexes for XML Queries, Tech. Rep. WSUCS–02–3, CS Dept, Wichita State Univ, 2002.