# A Simulation-based XML Routing Synopsis in Unstructured P2P Networks

Qiang Wang
Univ. of Waterloo, Canada
*q6wang@uwaterloo.ca*

Abhay Kumar Jha*
IIT, Bombay
*abhaykj@cse.iitb.ac.in*

M. Tamer Özsu
Univ. of Waterloo, Canada
*tozsu@uwaterloo.ca*

## Abstract

*Many emerging applications that use XML are distributed, usually over large peer-to-peer (P2P) networks on the Internet. The deployment of an XML query shipping system over P2P networks requires a specialized synopsis to capture XML data in routing tables. In this paper, we propose a novel graph-structured routing synopsis, called kd-synopsis, for deployment over unstructured super-peer based P2P networks. This synopsis is based on length-constrained FBsimulation relationship, which allows the balancing of the precision and size of the synopsis according to different space constraints on peers with heterogeneous capacity. We report comprehensive experiments to demonstrate the effectiveness of the kd-synopsis.*

## 1 Introduction

In recent years, Peer-to-Peer (P2P) architecture has become a popular decentralized platform for many Internet-scale applications such as file-sharing[1], instant messaging[2], and computing resource sharing[3]. Meanwhile, XML data are increasingly used as a format for data exchange and storage on the Internet, such as XML-based sensor data [17] and Web service data defined in WSDL and SOAP. Thus, it is increasingly important to process queries efficiently over data deployed in large-scale P2P networks, where centralized catalogs are not always available and peers may join and leave arbitrarily.

Distributed (XML) query processing can follow either data-shipping or query-shipping approaches. While data shipping moves data to a query processing site, query shipping systems route queries to where the data are located for processing. Query shipping is preferable in P2P systems because it can exploit the computational power of peers to process queries in parallel, and the cost of sending queries is usually much smaller than that of sending data.

Query shipping strategies vary based on different P2P overlay network architectures and routing protocols. In structured P2P architectures, data are placed on peers that are organized in a structured overlay network by using distributed hashing, and each peer manages, along with data, hash-based routing information for "neighboring" peers. In contrast, unstructured P2P architectures keep data on original peers and route queries either by flooding (e.g., Gnutella v0.4 [3]) or by using super-peers (e.g., Gnutella v0.6 [4]). While flooding is effective when searching popular data with a lot of replicas [26], a super-peer based architecture makes routing more efficient by organizing peers hierarchically where upper-level peers maintain information about the content of their lower-level ones and use this for routing. Such an architecture combines the advantages from both centralized systems (e.g., the exploitation of heterogeneity of peers) and distributed systems (e.g., scalability and robustness), and has been employed in real P2P file-sharing systems [6, 7].

*Work done while the author was visiting the University of Waterloo.
[1]Gnutella: http://www.gnutella.com
[2]Skype: http://www.skype.com
[3]Seti@home: http://setiathome.ssl.berkeley.edu

In this paper, we focus on XML query routing in query-shipping P2P systems under super-peer based overlay network architectures. More specifically, we propose an effective XML synopsis that can be used for maintaining routing state. For the sake of simplicity, we assume that each peer contains a tree-structured XML document (i.e., without ID or IDREF). In the case that a peer has multiple documents, we can model them under a virtual root. We are concerned with a commonly used subset of XPath [14], called Branching Path Query (BPQ) [22]. Briefly, BPQ covers *self*, *child*, *descendant*, *descendant-or-self*, *parent*, *ancestor* and *ancestor-or-self* axes. Moreover, we consider the conjunction (i.e., *and* operator) of predicates. Since we do not consider *negation* in queries, we actually work on a subclass of BPQ, defined as $\mathrm{BPQ}^{+}$ [31].

Our solution, called *kd-synopsis*, is a graph-structured synopsis over XML data that can easily balance its precision and size. The $k$ and $d$ are length constraints to be imposed over the Backward and Forward-simulation relationships (Section 2), which are theoretical foundations of our routing synopsis. To be more specific, our major design novelties are as follows:

- Most proposed routing state data structures are simple, such as the pair of <file name, IP address> used in file-sharing P2P systems, or the <IP address, physical address> used in Internet routing. These simple routing states usually capture, quite precisely, the information necessary for routing, and small-sized routing tables improve the scalability of the system with respect to both the storage size and the routing decision making process. However, XML data are much more complex, requiring a hierarchically-richer synopsis to capture its information. Furthermore, modern P2P systems are heterogeneous, in that different peers have different capacities including storage space size[4], which requires a more adaptive routing state representation whose size can be easily adjusted. Kd-synopsis better captures XML hierarchical information, while using a length-constrained FBsimulation relationship to control the size of the synopsis.

- The organization of the routing synopsis into routing tables is nontrivial, because it is not straightforward to aggregate synopses from multiple peers. In particular, for kd-synopsis, this is more critical since each peer's routing synopsis may have different $k$ and $d$ values. In this work, we propose a method to aggregate multiple kd-synopses into a routing state organized as a sequence of synopses ordered by $k$ and $d$ values, which keeps the precision of the synopses as much as possible with respect to specific space constraints.

Consequently, the contributions of this paper consist of three parts: First, we give the first formal definition of length-constrained FBsimulation relationship over XML data, which we call *kd-simulation*. Second, we develop a novel size-adjustable routing synopsis, named as kd-synopsis. Finally, we address the aggregation and update maintenance issues for routing tables consisting of kd-synopses, and demonstrate the effectiveness of our design with extensive experiments.

The organization of the paper is as follows. We introduce the background and related work in Section 2. In Section 3 we precisely define the kd-simulation relationship and address the generation of kd-synopses over XML documents. Section 4 focuses on the XML query shipping enforced in super-peer based unstructured P2P networks. Experimental results are presented in Section 5, demonstrating the effectiveness of our routing synopsis. Finally, we conclude in Section 6.

## 2 Related Work and Background

### 2.1 Related Work

XML query shipping problem is addressed in both structured [19, 16, 18] and unstructured P2P [24] domains. Since we are interested in unstructured P2P systems in this paper, Koloniari and Pitoura's work [24] is the most relevant, where Bloom filters are used to capture the set of elements on document tree levels and paths with same lengths. The basic idea is to encode the set of all the elements on each document level into a specific Bloom filter (the Breadth Bloom filter), and encode the set of all the linear path strings with a fixed length into a specific Bloom filter (the Depth Bloom filter). For example, Figure 1 shows the Breadth and Depth Bloom filters of an XML document that adheres to the DTD in the figure. Here the Bloom filters are presented in rectangles while their corresponding set of elements or paths are attached next to each entry for clarity. For simplicity, we use the first characters to represent element names.

[4]This space especially indicates main memory space used to store routing information.

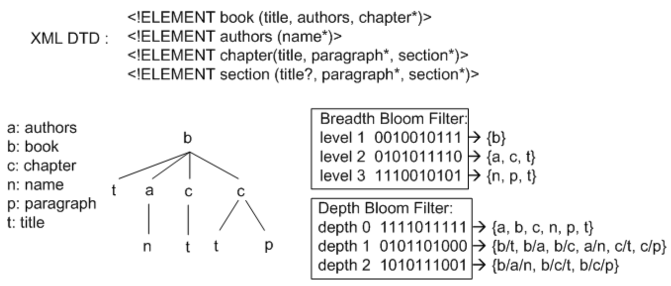

XML DTD :
<!ELEMENT book (title, authors, chapter*)>
<!ELEMENT authors (name*)>
<!ELEMENT chapter(title, paragraph*, section*)>
<!ELEMENT section (title?, paragraph*, section*)>
a: authors
b: book
c: chapter
n: name
p: paragraph
t: title
Breadth Bloom Filter:
level 1 0010010111 → {b}
level 2 0101011110 → {a, c, t}
level 3 1110010101 → {n, p, t}
Depth Bloom Filter:
depth 0 1111011111 → {a, b, c, n, p, t}
depth 1 0101101000 → {b/t, b/a, b/c, a/n, c/t, c/p}
depth 2 1010111001 → {b/a/n, b/c/t, b/c/p}

**Figure 1. A Bloom-filter based synopsis**

Although Bloom filter is space efficient in checking membership over a set of elements, it has several disadvantages when used to encode hierarchically-rich XML data. First, it cannot precisely encode the ancestor-descendant relationship among elements, because Breadth Bloom filter does not capture the parent-child relationship among nodes at different levels, and Depth Bloom filter only captures linear XML paths with specific lengths by uniform hashing. Thus a query with ancestor-descendant axis cannot be checked against the data effectively. For example, using the Bloom filter synopsis proposed in [24], the query "$c//n$" will be falsely determined as positive against the document in Figure 1 (a query is positive to a document if the evaluation result is non-empty). Second, to guarantee that there are no false negatives, which is desirable in query shipping systems, each document level has to correspond to a Breadth Bloom filter, so the size of the synopsis increases sharply when used to encode deep-structured documents, such as those following a recursive schema [25]. Finally, each peer in the system needs to use the same hash functions and agree on the size of the Bloom filters (i.e., size of bit sequences of the Bloom filter) corresponding to a specific level or a fixed path length. This may not be feasible in large heterogeneous P2P systems. In this work, we develop a graph-structured synopsis to represent routing state, which can capture the XML hierarchial information more easily and is size-adjustable so as to fit the heterogeneous capacity on different peers.

Compact graph-structured synopses have been developed to index XML data. Milo et al. employ Forward-bisimulation equivalence relationship (described in Section 2.2) to devise a reduced index for XML data [27] with respect to linear path XML queries. Polyzotis et al. develop XSketch synopsis for selectivity estimation, which also exploits the Bisimulation relationship to reduce the common structures in XML data [30]. Further, Ramanan proves that FBsimulation quotient is the smallest covering index[5] of XML data with respect to BPQ$^+$ queries [31], and the quotient can be much smaller than Bisimulation-based synopses. Kd-synopsis shares the graph structure of these synopses and builds on the same foundation of the FBsimulation relationship, but it can be significantly smaller than FBsimulation quotient. Moreover, to make the routing synopsis size-adjustable according to the heterogeneous capacity on different peers, we impose length constraints over the FBsimulation relationship such that we can easily balance the size and the precision through the constraint parameters.

Kaushik et al. also consider a length constrained equivalence relationship (i.e., k-bisimilarity) in their A(k)-index for XML indexing [23], but they impose length constraints only on the backward direction of the Bisimulation relationship, such that document nodes with different subtree structures can not be distinguished, leading to a less precise synopsis for BPQ$^+$ queries.

Homomorphic Compression has been proposed to encode XML data into a compressed form. Tolani et al. propose the XGrind system, which employs dictionary encoding and Huffman encoding to compress tags and data values respectively [32]. Min et al. use reverse arithmetic encoding to compress XML linear paths in the XPress system [28]. While both of the compression schemes provide reduced queriable XML data encodings, they do not compress structures of original XML data, so basically they are orthogonal work and their encoding strategies over element names can be applied directly over our kd-synopsis.

[5]An index $DI$ of document $D$ is a covering index with respect to a query set if no query in the set can distinguish between two nodes of $D$ that correspond to the same indexing node in $DI$.

## 2.2 Background

FBsimulation relationship has been shown to capture similar structures in XML data for BPQ$^+$ queries [31]. We use it as the theoretical foundation of kd-synopsis to remove redundancy with respect to the positive check[6] of BPQ$^+$ queries. FBsimulation is an extensively researched concept, which is used to describe the structure of semi-structured data [13], and has recently been employed to build compact covering indexes for XML data [31]. Given a directed graph $G = (V, E)$, where $V$ is the node set and $E$ is the edge set, for a node $v \in V$, let $label(v)$ return the label of $v$. Then

- Backward-simulation ($\ll_B$) is a binary relation over $V^2$, such that, $\forall u, v \in V$, $u \ll_B v$ iff
  - $label(u) = label(v)$;
  - for every parent node $u'$ of $u$ in $G$, there is a parent $v'$ of $v$ such that $u' \ll_B v'$.
- Forward-simulation ($\ll_F$) is a binary relation over $V^2$, such that, $\forall u, v \in V$,$u \ll_F v$ iff
  - $label(u) = label(v)$;
  - for every child node $u'$ of $u$, there is a child $v'$ of $v$ such that $u' \ll_F v'$.
- FBsimulation is a binary relation ($\ll_{FB}$) over $V^2$, such that $\forall\ u, v \in V$, $u \ll_{FB} v$ (called $u$ is FBsimulated to $v$) iff
  - $label(u) = label(v)$;
  - for every child node $u'$ of $u$, there is a child $v'$ of $v$ such that $u' \ll_{FB} v'$;
  - for every parent node $u'$ of $u$, there is a parent $v'$ of $v$ such that $u' \ll_{FB} v'$.

Specifically, with respect to an XML document tree, if node $u$ is FBsimulated to node $v$, two properties will hold: first, the incoming path starting from the document root to $u$ exactly matches that from the document root to $v$; second, for any node $u'$ on the matching incoming path (including $u$ itself), all subtree patterns rooted at $u'$ can be found in the subtrees rooted at $v'$, which is $u$'s counterpart node on the matching incoming path affiliated with $v$. For example, consider a document tree shown in the Figure 2, where we use subscripts to distinguish nodes with same labels for demonstration (i.e., $c_1$, $c_2$, $c_3$ all have label $c$). While $t_2$ is FBsimulated to $t_4$, $t_1$ is not FBsimulated to $t_4$ because they do not share matching incoming paths. Similarly, $s_1$ is FBsimulated to both $s_2$ and $s_3$, while $s_2$ is not FBsimulated to $s_4$ because they have distinct subtree patterns. Once a node is FBsimulated to the other, the former becomes redundant when used in the positive check of BPQ$^+$ queries against documents. For example, given a query */b/c/s*, we can correctly determine that the query is positive to the document in Figure 2 without considering $s_1$. We will exploit this finding to build the kd-synopsis later.

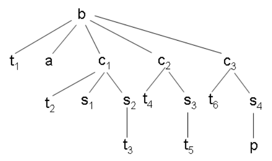


**Figure 2. An XML document tree**

FBsimulation quotient is a FBsimulation-based synopsis that can be used as a covering index of XML data with respect to BPQ$^+$ queries. Briefly, nodes FBsimulated to one another are merged so as to build a smaller sized index to answer BPQ$^+$ queries. For example, in the XML document shown in Figure 3, $s_2$ and $s_3$ are merged into the same node in the FBsimulation quotient because they are FBsimulated to each other, while $s_1$ is not collapsed because it is not FBsimulated to any node. While distinguishing $s_1$ from other nodes is crucial for answering queries precisely, it is unnecessary when we are concerned with the positive check of queries, where we only care whether a query can be answered by the data. Thus we can take a more aggressive reduction strategy than FBsimulation

[6]The check of whether the result set of evaluating a query against a document (or a synopsis) is non-empty.

quotient, which will be addressed in Section 3. Moreover, in a heterogeneous P2P environment, peers have different capacities to store routing information, such that we need a strategy to adjust the size of the synopsis dynamically, which requires an elegant way to trade off the precision of the synopsis.

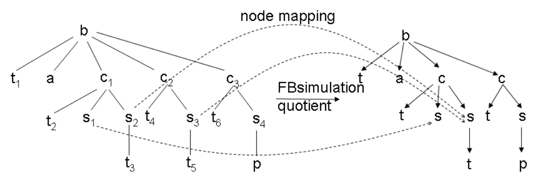


**Figure 3. FBSimulation quotient**

## 3 kd-synopsis

An XML routing synopsis should satisfy two properties to be effective in P2P environments. First, it should be as precise as possible such that the routing synopsis can discriminate queries accurately – this will save the communication cost since false positive queries[7] will not be routed. Second, the routing synopsis should adapt to heterogeneous capacities (i.e., storage space for routing information) on different peers. In this Section, we describe kd-synopsis, a novel routing synopsis that is based on kd-simulation, a length-constrained version of FBsimulation relationship. Through the adjustment of length constraints over kd-simulation, kd-synopsis can acquire a level of precision with respect to specific size constraints.

### 3.1 Definition of kd-simulation

In this section, we define kd-simulation relationship that can identify XML element nodes that have similar structures within the specified incoming and outgoing levels. Specifically, based on the notions of Backward-simulation, Forward-simulation, and FBsimulation, we define their length-constrained variants: *k-Backward-simulation*, *d-Forward-simulation* and *kd-simulation*.

- Given a graph $G = (V, E)$, **k-Backward-simulation** ($\ll_B^k$) is a binary relation over $V^2$ defined inductively as follows ($k \geq 0$). Given $u, v \in V$,,
  - $u \ll_B^0 v$ iff $label(u) = label(v)$;
  - $u \ll_B^k v$ iff (1) $label(u) = label(v)$; (2) for every parent node $u'$ of $u$, there is a parent $v'$ of $v$ such that $u' \ll_B^{k-1} v'$.
- Given a graph $G = (V, E)$, **d-Forward-simulation** ($\ll_F^d$) is a binary relation over $V^2$ defined inductively as follows ($d \geq 0$). Given $u, v \in V$,
  - $u \ll_F^0 v$ iff $label(u) = label(v)$;
  - $u \ll_F^d v$ iff (1) $label(u) = label(v)$; (2) for every child node $u'$ of $u$, there is a child $v'$ of $v$ such that $u' \ll_F^{d-1} v'$.

Briefly, k-Backward-simulation identifies similar incoming paths within $k$ XML tree levels, while d-Forward-simulation identifies similar outgoing structures within $d$ XML tree levels. For example, in the XML document tree shown in Figure 2, $t_1 \ll_B^0 t_2$ holds because they share the same label, but $t_1 \ll_B^1 t_2$ does not because they have different parents (i.e., different incoming edge one level away); $c_2 \ll_F^1 c_3$ holds but $c_2 \ll_F^2 c_3$ does not hold because they have distinct outgoing edges two levels away (i.e., $s/t$ and $s/p$ respectively).

Now we define kd-simulation as a reflexive and transitive relation over $V^2$, with $k$ and $d$ values as the length constraints over Backward and Forward-simulation relationship respectively.

[7]A query is false positive when it is determined as positive to the synopsis while it is actually negative to the data on which the synopsis is generated.

- Given a graph $G = (V, E)$, **kd-simulation** ($u \ll^{(k)(d)} v$, called $u$ is kd-simulated to $v$) is a binary relation over $V^2$ defined inductively as follows. Given $u, v \in V$,
  - $u \ll^{(0)(d)} v$ iff $label(u) = label(v)$ and $u \ll_F^d v$;
  - $u \ll^{(k)(0)} v$ iff $label(u) = label(v)$ and $u \ll_B^k v$;
  - $u \ll^{(k)(d)} v$ iff (1) $label(u) = label(v)$; (2) for every child node $u'$ of $u$, there is a child $v'$ of $v$ such that $u' \ll^{(k)(d-1)} v'$; (3) for every parent node $u'$ of $u$, there is a parent $v'$ of $v$ such that $u' \ll^{(k-1)(d)} v'$.

For example, in the Figure 2, $t_4$ is kd-simulated to $t_6$ with respect to $k = 1, d = 1$ because their parent nodes $c_2 \ll^{(0)(1)} c_3$, and neither of them has children nodes. However, $t_4$ is not kd-simulated to $t_6$ anymore under $k = 1$ and $d = 2$, since $c_2 \ll^{(0)(2)} c_3$ (i.e., $c_2 \ll_F^2 c_3$) does not hold. Note that when both $k$ and $d$ are equal to $G$'s graph diameter (i.e., the tree depth when $G$ is an XML document tree), kd-simulation evolves into the FBsimulation relationship, which is the basis of the covering index graph for BPQ$^+$ [31], and when both $k$ and $d$ are equal to zero, it degrades into the label-equivalence relationship, which is the basis of the label-splitting graph[8] that captures all the basic label and edge information in $G$.

Efficient algorithms have been designed to compute the Forward-simulation relationship among graph nodes [15, 21]. We propose an algorithm (Algorithm 1) to cover our special circumstances. This algorithm is similar to that given by Henzinger et al. [21], but has two important differences. First, instead of only Forward-simulation, we consider both Backward and Forward-simulation. Second, we apply length constraints over the FBsimulation relationship. The implementation is straightforward, containing iterations for the computation of d-Forward-simulation (k-Backward-simulation), where in each iteration, the distinction among nodes incurred by different outgoing edges (incoming edges) are propagated to an upper (lower) level in the tree. It is more economical to put the computation of d-Forward-simulation before that of the k-Backward-simulation, because the latter computation can easily take into account the distinction of same-labelled nodes that do not have d-Forward-simulation relationship. Otherwise, an extra step is needed to propagate such distinctions $k$ levels down the tree. This asymmetry over the computation order is caused by the characteristic of the tree structure that each node has at most one incoming path while it probably has multiple outgoing edges. Since Algorithm 1 is a length constrained version of Henzinger et al. [21], the time complexity is still $O(|V||E|)$ with respect to the graph $G$. In the following, we give the correctness proof of the algorithm.

*Proof.* Given an XML document tree $D = (V, E)$ and $u, v \in V$ with $label(v) = label(u)$, if ($v$, $u$) belong to the kd-simulation relationship under $k$ and $d$ values, two sufficient conditions as follows: first, $\forall p_v$ that is the ancestor node of $v$ that is within $l$ ($0 \leq l \leq k$) tree levels from $v$, there exists a node $p_u$ as the ancestor node of $u$ that is $l$ tree levels away from $u$ such that $p_v \ll^{(k-l)(d)} p_u$; second, $\forall c_v$ that is a descendant node of $v$ that is within $l'$ ($0 \leq l' \leq d$) tree levels from $v$, there exists a node $c_u$ as the ancestor node of $u$ that is $l'$ levels away from $u$ such that $c_v \ll^{(k)(d-l')} c_u$. Consider the iteration (lines 12 to 15 in Algorithm 1), $\forall x \in V$, $sim(x)$ (i.e, the set of the potential nodes that $x$ is kd-simulated to) changes in each iteration according to its children nodes. We claim invariant 1 before the $i_{th}$ ($1 \leq i \leq d$) iteration: $\forall y \in sim(x)$, $x \ll^{(0)(i-1)} y$. Initially before the iteration, $sim(x)$ contains all the nodes that share the same lable as $x$, so the invariant holds (i.e., $\forall y \in sim(x)$, $x \ll^{(0)(0)} y$). Then consider the $i_{th}$ iteration, let us assume that for a node $y \in sim(x)$, there exist at least one child node $c_x$ of $x$ such that there is no child node $c_y$ of $y$ and $c_x \ll^{(0)(i-1)} c_y$. Thus, in this $i_{th}$ iteration, $y$ will be removed from $sim(x)$ (line 18), such that after this iteration, it holds that $\forall z \in sim(x)$, $label(z) = label(x)$ and for any child node $c_x$ of $x$, there exists a child node $c_z$ of $z$ such that $c_x \ll^{(0)(i-1)} z$, which leads to the conclusion that $x \ll^{(0)(i)} y$, satisfying the invariant 1. As for the second iteration (lines 28 to 41), we claim another invariant (i.e., invariant 2) before each iteration: $\forall y \in sim(x)$, $x \ll^{(i-1)(d)} y$. Initially before the first iteration, the invariant holds because it is obvious that $\forall y \in sim(x)$, $x \ll^{(0)(d)} y$, resulting from the invariant 1. Then consider the $i_{th}$ iteration, and let us assume that before this iteration, $\forall y \in sim(x)$, $x \ll^{(i-1)(d)} y$, then in this $i_{th}$ iteration, for an node $y \in sim(x)$, for a parent node of $x$, $p_x$, it there does not exist a parent node $p_y$ of $y$ such that $\ll^{(i-1)(d)} p_y$, then $y$ will be removed from the $sim(x)$ (line 34), thus after the $i_{th}$ iteration, it holds that $\forall z \in sim(x)$, $label(z) = label(x)$ and for any parent node $p_x$ of $x$, there exists a parent node $p_z$ of $z$ such that $p_x \ll^{(i-1)(d)} p_z$. Meanwhile, after the $i_{th}$ iteration, it naturally becomes true that, for an arbitrary child node $c_x$ of $x$, there exists a child node $c_y$ of $y$ such that $c_x \ll^{(i)(d-1)} c_y$, because $c_x \ll^{(0)(d-1)} c_y$ holds according to invariant 1, meanwhile there is no distinction on

[8]In such a graph, nodes are merged if they have common labels, and edges are merged if they have common labels on both ends.

the incoming paths within $i$ tree levels away (otherwise, $y$ will be removed from $sim(x)$ in this iteration by line 34). Consequently, if $y$ still belongs to $sim(x)$ after the $i_{th}$ iteration, $x \lll^{(i)(d)} y$ will hold, thus after $k$ iterations, $\forall y \in sim(x)$, $x \lll^{(k)(d)} y$. $\square$

**Algorithm 1** *compute_kd_simulation*

```
1: Input: $G = (V, E)$, $k$, $d$
2: Output: for each node $v \in V$, the set of vertices $sim(v)$ such that $sim(v)$ contains all the nodes $v$ satisfying $v \lll^{(k)(d)} u$
3: Auxiliary functions: $\forall x, y \in V$, $pre(y) = \{x|(x, y) \in E\}$, $post(x) = \{y|(x, y) \in E\}$
4:
5: for $v \in V$ do
6:    $sim(v) \leftarrow$ the set of the vertices with the same label as $v$'s;
7: end for
8:
9: $k_{tmp} \leftarrow k$; $d_{tmp} \leftarrow d$;
10:
11: //d-Forward-simulation
12: for all $v \in V$ do
13:    $sim'(v) \leftarrow sim(v)$;
14: end for
15: while $d_{tmp} > 0$ do
16:    for all $v \in V, u \in sim(v)$ do
17:       if $\exists v_c \in post(v)$, such that there does not exist $u_c \in sim(v_c)$, where $u_c \in post(u)$ then
18:          remove $u$ from $sim'(v)$;
19:       end if
20:       $d_{tmp} \leftarrow d_{tmp} - 1$;
21:       for all $v \in V$ do
22:          $sim(v) \leftarrow sim'(v)$;
23:       end for
24:    end for
25: end while
26:
27: //k-Backward-simulation
28: for all $v \in V$ do
29:    $sim'(v) \leftarrow sim(v)$;
30: end for
31: while $k_{tmp} > 0$ do
32:    for all $v \in V, u \in sim(v)$ do
33:       if $\exists v_p \in pre(v)$, such that there does not exist $u_p \in sim(v_p)$, where $u_p \in pre(u)$ then
34:          remove $u$ from $sim'(v)$;
35:       end if
36:       $k_{tmp} \leftarrow k_{tmp} - 1$;
37:       for all $v \in V$ do
38:          $sim(v) \leftarrow sim'(v)$;
39:       end for
40:    end for
41: end while
```

### 3.2 Building kd-synopsis

In this section, we develop our new graph-structured synopsis (kd-synopsis) based on the kd-simulation relationship. Algorithm 1 already determines the kd-simulation relationship among document nodes, based on which we collapse an arbitrary graph node $u$ to $v$ ($u \neq v$) if $u \lll^{(k)(d)} v$. This process continues until no node remains that can be kd-simulated by any other. Accordingly, an arbitrary edge $e_1 = (u_1, v_1)$ is collapsed into $e_2 = (u_2, v_2)$ if both $u_1 \lll^{(k)(d)} v_1$ and $u_2 \lll^{(k)(d)} v_2$. Additionally, since absolute BPQ$^+$ queries (i.e., queries starting from root level) require the synopsis to capture level information of the original document, we keep a mark for the kd-synopsis vertex[9] that the root node of the original document is collapsed into. We propose Algorithm 2 to build the kd-synopsis for an XML document, where $sim(v)$ is the set of all the nodes that $v$ is kd-simulated to (i.e., the output from Algorithm 1). For clarity, we use $V$ and $E$ to denote the node and edge set of the original XML document tree, and use $\mathbb{V}$ and $\mathbb{E}$ to denote their counterparts in the synopsis graph. Briefly, we go through each node $v$ of an XML document tree, and if there does not exist other nodes that $v$ is kd-simulated to (lines 11 and 12), we insert it into $final\_set$; otherwise, we make sure that $final\_set$ includes at least one node $u$ that $v$ is kd-simulated to (lines 15 to 19). Meanwhile, to facilitate the synopsis generation, we use $ext(u)$ to track all the document tree nodes that are kd-simulated to $u$ (line 20). After all the nodes are traversed, no nodes in $final\_set$ will be kd-simulated to another one, thus we can build the kd-synopsis by generating a separate synopsis vertex for

[9]To distinguish from the nodes in original document trees, we use vertex when we refer to a node in kd-synopsis.

each node in $final\_set$ and establish synopsis edges accordingly (lines 24 to 38). Figure 4 depicts the kd-synopses of the example document (Figure 2 and 3) under different $k$ and $d$ values. Note that in the kd-synopsis with $k = 1$ and $d = 2$, $c_3$ is not collapsed because it has a distinct outgoing edge (i.e., $s/p$) two levels away.

**Algorithm 2** *compute_synopsis*

```
1: Input: $sim(v)$, a set of nodes that $v$ ($v \in V$) is kd-simulated to; $G = (V, E)$, the original data graph
2: Output: a kd-synopsis graph $S = (\mathbb{V}, \mathbb{E})$
3: Auxiliary function: $\forall v \in V$, $label(v)$ returns $v$'s label.
4:
5: for all $v \in V$ do
6:    $sim(v) \leftarrow sim(v) - \{v\}$;
7:    $ext(v) \leftarrow \{v\}$;
8: end for
9: $final\_set \leftarrow \phi$;
10: for all $v \in V$ do
11:    if $sim(v) = \phi$ then
12:       $final\_set \leftarrow final\_set \cup \{v\}$;
13:    else
14:       $final\_set \leftarrow final\_set - \{v\}$;
15:       if $final\_set \cap sim(v) = \phi$ then
16:          choose any $u \in sim(v)$;
17:          $final\_set \leftarrow final\_set \cup \{u\}$;
18:       end if
19:       choose any $u \in final\_set \cap sim(v)$;
20:       $ext(u) \leftarrow ext(u) \cup ext(v)$;
21:    end if
22: end for
23: //generate synopsis vertices
24: for all $v \in final\_set$ do
25:    generate a synopsis vertex $v_s$;
26:    $\mathbb{V} \leftarrow \mathbb{V} \cup \{v_s\}$;
27:    establish a mapping from $u \in ext(v)$ to $v_s$;
28:    if $\exists u \in ext(v)$ that has no parent vertex in $G$ then
29:       mark $v_s$ as a root;
30:    end if
31: end for
32: //establish synopsis edges
33: for all $e = (u, v) \in E$ do
34:    if $\exists u_s, v_s$ such that $u$ maps to $u_s$ and $v$ maps to $v_s$ then
35:       generate a synopsis edge $(u_s, v_s)$ in $S$;
36:       $\mathbb{E} \leftarrow \mathbb{E} \cup \{(u_s, v_s)\}$;
37:    end if
38: end for
```

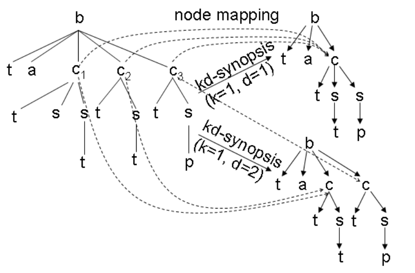


**Figure 4. kd-synopsis**

The size of the kd-synopsis is bounded by the size of the original document, and the reduction ratio depends on $k$ and $d$ values. The smaller $k$ and $d$ are, the higher the reduction ratio will be, and in the extreme case when $k = d = 0$, the kd-synopsis is the same as the label-splitting graph. The time cost of Algorithm 2 primarily consists of three parts: first, The cost to building the $final\_set$, where the most expensive part is due to the set intersection between $final\_set$ and $sim(v)$ (line 15 and 19). In this work, we maintain the membership of each node $u \in G$ in $final\_set$, such that the cost of the set intersection can be bounded by $|V|$, leading to a total cost of this part as $O(|V|^2)$. Second, since the extent set corresponding to the $final\_set$ is a partition of $V$, line 27 is executed at most $V$ times, so the generation of $\mathbb{V}$ (line 24 to 28) costs at most $|V|$. Third, the establishment of $\mathbb{E}$ (line 33 to 38) iterates over each edge in $G$, leading to a cost of $|E|$. In all, the time complexity of Algorithm 2 is bounded by $|V^2| + |V| + |E|$, which is efficient. To demonstrate that our construction algorithm is optimum, we prove that the kd-synopsis $S$ constructed by Algorithm 2 has no redundant nodes that can be kd-simulated by others. To prove this, we argue that, given $u, v \in G$, first, when $v \ll^{(k)(d)} u$ while $u \ll^{(k)(d)} v$ does not hold, $v$ will not be kept in $S$; second, when both $v \ll^{(k)(d)} u$ and $u \ll^{(k)(d)} v$ hold, at most one of them (actually the one to be first processed in Algorithm 2) will be kept in $S$.

*Proof.*

- According to the lines 11 to 21 of Algorithm 2, when vertex $v$ is processed, if $v \in final\_set$ and $sim(v)$ is non-empty, then $v$ will be removed, and a vertex $u \in sim(v)$ (rather than $v$) will be kept in $final\_set$ such that $v \ll^{(k)(d)} u$; later even if $u$ is replaced by another vertex $w$ in $final\_set$, it always holds that $\exists w \in final\_set$ such that $v \ll^{(k)(d)} w$. Based on the property of the simulator set, $\forall x \in V$, if $x \ll^{(k)(d)} v$, $sim(x)$ contains all the vertices that $v$ is also kd-simulated to, so $\forall x$ processed after the removal of $v$, $sim(x) \cap final\_set \neq \phi$, thus $v$ will never get a chance to enter $final\_set$ any more.
- Without loss of generality, let us assume that $v$ is processed after $u$. When $v$ is processed, if $v \in final\_set$, it will be removed according to Algorithm 2 (line 14); if $u \in final\_set$, then $u$ will be kept in $final\_set$, rejecting $v$ to enter $final\_set$ later, as analyzed in proof of (1); or if $u$ does not belong to $final\_set$ when $v$ is processed, there must be $w \in V$ such that $u, v \ll^{(k)(d)} w$, and since $sim(v) \neq \phi$, $w$ must belong to $sim(v) \cap final\_set$, rejecting $v$ to enter $final\_set$ in the same way.

□

kd-synopsis has two important properties: first, the synopsis has no false negatives when checked against BPQ$^+$ queries; second, the synopsis maintains a well-defined precision that is determined by length constraints $k$ and $d$. The first property is obvious, since when a graph node (i.e., document tree node) is collapsed into another, all its edge information is kept, which indicates that there is no loss of label or edge information in the synopsis. Note that we assume that root information is kept so that absolute BPQ$^+$ queries can be checked correctly. With this guarantee, all data relevant to a query can be identified, which is desirable in query shipping systems. As for the precision of the synopsis, a kd-synopsis can be used to precisely check the positive of BPQ$^+$ queries that can be characterized by *kd-pattern*, a parameter-constrained query pattern defined in the following.

- *(1)-descendant* is a query pattern defined as *child::c*, where *child* is the axis defined in XPath, and $c$ is an arbitrary label. *(d)-descendant* ($d > 1$) is a query pattern recursively defined as *child::c[(d-1)-descendant]**.
- *(1)-ancestor* is a query pattern defined as *parent::p*, where *parent* is an axis defined in XPath, and $p$ is an arbitrary label. *(k)-ancestor* ($k > 1$) is a query pattern defined as *parent::p[(k-1)-ancestor][(d)-descendant]**.
- Then *kd-pattern* is a query pattern defined as *x[(k)-ancestor][(d)-descendant]**, where $x$ is an arbitrary label. Moreover, to model absolute BPQ$^+$ queries, we can impose a root constraint over $x$.

To visualize the kd-pattern, we illustrate it as a kd-pattern tree in Figure 5 (a), where $p_i$ denotes an ancestor element that is $i$ edges away from $x$, and $c_{i,j}$ denotes the $j_{th}$ element on the $i$ level of a *(d)-descendant pattern tree*. For future reference, the node labelled as $x$ is called *pivot* of the kd-pattern tree. For example, given a BPQ$^+$ query *t[parent::c[child::s[child::t][child::p]]]*, its kd-pattern tree is shown in Figure 5 (b) where $k = 1$ and $d = 2$. Since the modelling of BPQ$^+$ queries into kd-patterns is straightforward, we do not discuss it further in this paper. While the kd-pattern basically captures parent-child axis in BPQ$^+$ queries, ancestor and descendant axes can also be modelled as kd-patterns if the document depth information is available. For example, given a query *c[descendant::t]* and the information that the depth of the documents to be queried over is up to three, the query can be modelled as three kd-patterns *c[child::t]*, *c[child::*[child::t]]*, and *c[child::*[child::*[child::t]]]* with $k = 0$ (i.e., no (k)-ancestor) and $d = 1, 2, 3$ respectively. Note here the "*" denotes wild card labels. Note, further, that kd-pattern tree is a general pattern tree not defined only for modelling queries, and we will use it to characterize a subtree in XML documents as well.

The following theorem states that kd-synopsis maintains a well-defined precision with respect to the positive check of BPQ$^+$ queries.

**Theorem 3.1.** *If a BPQ$^+$ query can be modelled as kd-patterns (under k and d values), its positive/negative check against an XML document can be done correctly on all kd-synopses that are constrained by $k'$ and $d'$ values, where $k' \geq max(k, d)$ and $d' \geq max(k, d)$.*

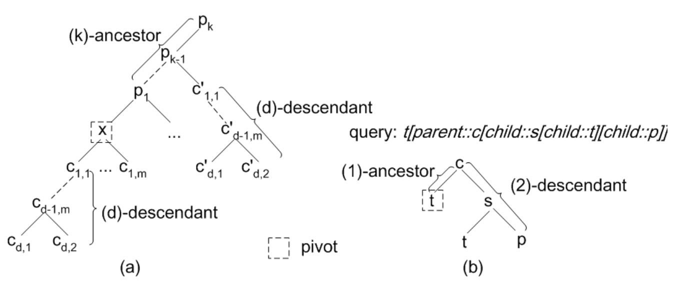


**Figure 5. kd-pattern tree**

*Proof.* We first prove that all queries positive to documents will also be positive to their kd-synopses. Given a BPQ$^+$ query $q$, let us consider a specific XML document $D$ that $q$ is positive to. $q$ will definitely be positive to $D$'s arbitrary kd-synopses because we already claimed that kd-synopsis has no false negatives when checked against BPQ$^+$ queries.

Now we prove that if $q$ is negative to $D$, while $q$ can be modelled as a kd-pattern tree $T$ under $k$ and $d$ values, it will also be negative to $D$'s kd-synopses under $k'$ and $d'$, where $k' \geq k$ and $d' \geq d$. If any node or edge in $T$ does not exist in $D$, then $q$ will be directly determined negative for all the kd-synopses because we do not add label or edge there. Thus let us assume that nodes labelled with $x$ exist in $D$ and focus on all the $D$'s kd-pattern (under $k$ and $d$) subtrees that have a pivot node labelled as $x$ (other kd-pattern subtrees will not affect the positive check of $q$). Consider nontrivial kd-patterns with $k > 1$ and $d > 1$ (the conclusion can be easily extended to kd-patterns with $k \leq 1$ and $d \leq 1$). Since $q$ is negative to $D$, there exists at least one node in $T$ that can not be satisfied by $D$, called the failure point ($fp$). There are two possible reasons for this failure. First, an edge incident to $fp$ is missing from any $D$'s kd-pattern subtree that has the pivot node with the same label as $x$. Let us denote the missing edge as $e = (u_e, v_e)$ and consider an arbitrary $D$'s kd-pattern subtree $S$ that includes another node $u_x$ with the same label as $u_e$. Since $S$ does not include $e$, $u_x$ will not have an edge that ends with another node that has the same label as $v_e$, so $u_x$ is not kd-simulated to $u_e$ under $k' > 1$ and $d' > 1$. Meanwhile, the unavailability of the pivot node within $S$, whose scope is bounded by $k'$ and $d'$, distinguishes $u_x$ from $u_e$ with respect to the kd-simulation relationship. Thus $u_e$ is not kd-simulated to $u_x$ either. Consequently, $e$ will not merge into $S$ in any kd-synopsis constrained by $k'$ and $d'$, and $T$ can not match with any kd-pattern subtree in the kd-synopsis, such that $q$ is definitely negative against the synopsis. Second, consider the case that $fp$ in $T$ has several edges (or paths) incident to it, and we will prove that $fp$ can not be satisfied by any kd-synopsis under $k'$ and $d'$ values. We will focus on edges in the following, but the same proof holds for paths. Let us consider two edges $e_1$ and $e_2$ incident on $fp$, which exist in $D$ but do not appear in the same kd-pattern subtree. Without loss of generality, take two kd-pattern subtrees $S_1$ and $S_2$, which have different pivot nodes $x_1$ and $x_2$, and include $e_1$ and $e_2$ respectively. $x_2$ is not kd-simulated to $x_1$ because the unavailability of $e_1$ will be propagated to $x_2$ within $S_2$, according to the definition of the kd-simulation. Similarly, $x_1$ is not kd-simulated to $x_2$ because of the unavailability of $e_2$ in $S_1$. Consequently, $x_1$ and $x_2$ will not collapse into each other in any kd-synopsis under $k'$ and $d'$ values. Moreover, both $e_1$ and $e_2$ have an end node that shares the same label as $fp$, denoted as $e_1^{fp}$ and $e_2^{fp}$ respectively. Since $e_1^{fp}$ and $e_2^{fp}$ are incident to edges that end with distinct labels (i.e., $e_1$ and $e_2$), neither is kd-simulated to the other (under $k' > 1$ and $d' > 1$), so they do not collapse into each other in any kd-synopsis. Thus, no kd-pattern subtree with a pivot node labelled as $x$ in the synopsis will contain both $e_1$ and $e_2$, leading to the conclusion that $q$ is negative against the synopsis. $\square$

Consider the example document tree and its two kd-synopses in Figure 6 (a), which are constrained by $k' = d' = 1$ and $k' = d' = 2$ respectively. Further consider two kd-pattern trees shown in Figure 6(b) and 6(c) that are negative to the document. The failure point $c$ in the kd-pattern tree with $k = 2$ and $d = 1$ (Figure 6 (b)) has a missing edge $c/s$, while the failure point $b$ in the kd-pattern tree with $k = 1$ and $d = 2$ (Figure 6 (c)) has two paths (i.e., $s/t$ and $s/p$) incident to it, which belong to different kd-pattern subtrees in the document. While both kd-pattern trees are false positive when checked against the kd-synopsis under $k' = d' = 1$, they are correctly determined as negative against the kd-synopsis under $k' = d' = 2$, where $k' \geq k$ and $d' \geq d$.

In summary, kd-synopsis qualifies as a routing synopsis in P2P environments since its size can be easily adjusted

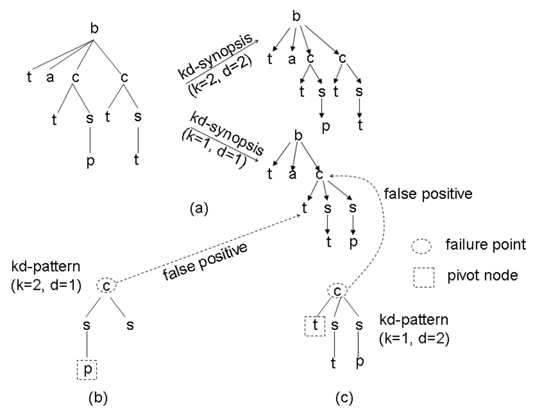


**Figure 6. Check kd-patterns**

through $k$ and $d$ value so as to adapt to the heterogeneous capacity on different peers, while maintaining a well-defined precision to check the positive/negative of BPQ$^+$ queries. On peers with larger space, $k$ and $d$ values can be set higher such that we can get a more precise synopsis of the data, while on peers with less space, we can use smaller $k$ and $d$ values, trading off some precision. A label-splitting graph is normally compact, therefore, in this work we do not consider more aggressive reductions such as collapsing nodes or edges with different labels into each other.

## 4 kd-synopsis Based Routing

kd-synopsis can be used in all kinds of routing-based P2P networks. In this work, we consider super-peer based unstructured P2P networks, where there are plain peers and super-peers, both of which keep XML data, but only the latter act as routers for query shipping. Since peers are heterogeneous with respect to the storage space for the routing information, the routing states are organized under space constraints (Section 4.1). A query can be originated on an arbitrary peer. If the peer is a plain peer, the query will be forwarded to its super-peer, otherwise if the peer is a super-peer, the query will be routed to peers where the query can be potentially answered. In parallel, the query will also be forwarded to upper-level super-peers. For simplicity, we assume that top-level super-peers broadcast queries among each other such that no propagation of routing information happens among them. Since peers may join and leave the network dynamically, we will also discuss the update of the routing information (Section 4.2).

### 4.1 Routing State and Routing Decision Making

The routing tables on super-peers are managed in a bottom-up way. To establish a routing entry on super-peers for a plain peer, a kd-synopsis of the document stored on the plain peer will be generated and sent over to its super-peer. This synopsis generation algorithm can be obtained from Algorithm 2 by setting the initial values of $k$ and $d$ as the depth of the document, and then decreasing the $(k, d)$ values step-by-step until the size of the synopsis satisfies the space constraint imposed by the super-peer. To establish a routing entry for a super-peer on upper-level super-peers, we first need to aggregate the routing information of the lower-level super-peers, which constitutes a set of kd-synopses. Given a space constraint for this super-peer $S$ (i.e., the space allowed to store the information of $S$ on its upper-level super-peers), a naive way to aggregate is to merge the synopses into one kd-synopsis under the minimum $k$ and $d$ values among all the $k$ and $d$ values of the synopses in routing table of $S$, denoted as $k_{min}$ and $d_{min}$ respectively. In this way, the resulting synopsis is precise in positive check of the BPQ$^+$ queries that can be modelled as kd-patterns under $k = d = min(k_{min}, d_{min})$, which is a natural conclusion from Theorem 3.1. However, it is obvious that the precision of the synopsis is limited by $k_{min}$ and $d_{min}$, which is undesirable when either $k_{min}$ or $d_{min}$ is small. So, instead of merging all synopses into a single synopsis, we organize the kd-synopses by using a sequence, which keeps the precision of the original kd-synopses as much as possible. The reason of employing a sequence instead of a set is that we can easily track where a synopsis is aggregated. The algorithm is given in Algorithm 3. Briefly, given a set of kd-synopses, we first sort them in a decreasing lexicographic order over $(k, d)$ value pair. Then we union the synopses with same $(k, d)$ value pair, which leads to a synopsis sequence with a strict decreasing order over $(k, d)$ value pair. Afterwards, if the size of this sequence violates the specified space constraint, we start to reduce the size of the sequence by pairwise aggregating synopses: in each iteration, we choose a pair of kd-synopses $S_1$ (e.g., under $k = k_1$ and $d = d_1$) and $S_2$ (e.g., under $k = k_2$ and $d = d_2$) such that $S_1$ and $S_2$ are the first pair in the sequence that satisfies $k_1 = k_2$ and $d_1 \leq d_2$. $S_1$ and $S_2$ are then aggregated into $S_{agg}$, a kd-synopsis with $k = k_1$ and $d = min(d_1, d_2)$. If the pair of $S_1$ and $S_2$ does not exist, which indicates that all the kd-synopses in the sequence have different $k$ values, we aggregate the first pair of kd-synopses in the sequence. This aggregation process continues until the space constraint is satisfied. In this way, we can easily locate an original kd-synopsis in the aggregated sequence: it is either located at a synopsis with the same $k$ and $d$ values, or located at the foremost synopsis with the same $k$ value but a lower $d$ value, otherwise, the synopsis must be aggregated into the foremost synopsis (i.e., with a lower $k$ value) of the sequence. This will facilitate the update maintenance of the routing information which requires the location of a synopsis already aggregated in routing tables on super-peers.

**Algorithm 3** *aggregate_synopsis*

```
 1: Input: a set of synopses $\{S_i\}$, with $S_i$ following kd-simulation under $k_i$ and $d_i$ constraints; the space constraint $C$
 2: Output: a sequence of (aggregated) synopses satisfying $C$.
 3:
 4: Sort the set $\{S_i\}$ in an decreasing lexicographic order on $(k_i, d_i)$;
 5: while there exist two synopses $S_1$ and $S_2$ following the same (k)(d)-simulation do
 6:    $S_{agg} \leftarrow S_1 \cup S_2$;
 7:    $compute\_synopsis(S_{agg}, k, d)$;
 8:    remove $S_1$ and $S_2$ and add $S_{agg}$ into the sequence, keeping the decreasing order;
 9: end while
10: while (the aggregated size of the sequence) $\geq C$ do
11:    if there exists only one kd-synopsis $S$ in the sequence that is under $k$ and $d$ values then
12:       if $k \geq d$ then
13:          $k \leftarrow k - 1$;
14:       else
15:          $d \leftarrow d - 1$;
16:       end if
17:       $S_{agg} \leftarrow compute\_synopsis(S, k, d)$;
18:       replace $S$ with $S_{agg}$;
19:    else
20:       if there exists a pair of kd-synopses $S_1$, $S_2$ in the sequence such that $S_1$ conforms to kd-simulation under $k_1$ and $k_2$ values and $S_2$
          conforms to kd-simulation under $k_2$ and $d_2$ values, and $k_1 = k_2$ and $d_1 \leq d_2$ then
21:          get the foremost such pair $S_1$, $S_2$ from the sequence;
22:       else
23:          get the foremost pair $S_1$, $S_2$ from the sequence;
24:       end if
25:       $k_{min} = min(k_1, k_2)$; $d_{min} = min(d_1, d_2)$;
26:       $S_{agg} \leftarrow S_1 \cup S_2$;
27:       $S_{agg} \leftarrow compute\_synopsis(S_{agg}, k_{min}, d_{min})$;
28:       remove $S_1$ and $S_2$ from the list and add $S_{agg}$ into the list, keeping the decreasing order;
29:    end if
30: end while
```

The routing decision making of a query $q$ against a routing entry works in a straightforward way. When $q$ is routed to a super-peer, it is checked against the kd-synopses in the sequence of each routing entry. If $q$ is positive to any kd-synopsis, it will be routed to the peer corresponding to the routing entry, where the routing process continues in the same way; in contrast, if $q$ is negative to all the kd-synopses in the routing entry, $q$ will not be sent to the corresponding peer. The check of whether a query $q$ is positive against a kd-synopsis can be reduced to the evaluation of the query over the synopsis. A FBsimulation-based query evaluation strategy addressed by Ramanan [31] is directly applicable. Another efficient XPath evaluation algorithm is addressed in [20], but we prefer Ramanan's strategy since it evaluates queries over a graph-based synopsis rather than a tree-structured document.

### 4.2 Maintenance of Routing Information

In a super-peer based P2P network, the routing information (i.e., kd-synopsis) is propagated and replicated on multiple peers. So it needs to be updated when peers join or leave the network. Specifically, when a joining peer $P$ is connected to its super-peer $SP$, it propagates its routing information (i.e., the kd-synopsis) to $SP$ and then $SP$ forwards changes over its own routing table to upper-level super-peers in an iterative way. When $P$ leaves the overlay network, its corresponding routing information needs to be removed from all upper-level super-peers that contain $P$'s information.

Since kd-synopses are aggregated together in routing tables, we need counter mechanisms to record the frequencies of common edges and vertices. Given a kd-synopsis $S$, the corresponding frequency-enabled synopsis is built by attaching to each edge $e$ an integer counter that keeps the sum of the occurrences of common edges that collapse into $e$. Then two synopses $S_1$ and $S_2$ are aggregated (through Algorithm 3) into $S_{agg}$, we track all edges in $S_1$ and $S_2$ that contribute to the same edge in $S_{agg}$, and then assign the summary of the frequencies to the edge in $S_{agg}$. Since the aggregation of kd-synopses is based on the computation of the kd-synopsis, we integrate this process into *compute_synopsis* (i.e., Algorithm 2), which leads to Algorithm 4. Besides the addition of the frequency accumulation, a tricky part of this algorithm is that the extent for each synopsis vertex now corresponds to all the original document nodes (or synopsis vertices in the case of computation over synopses) that can be aggregated into it (see Algorithm 4 lines 7-9), in other words, a node in the original tree (or synopsis) is merged into all the nodes that can kd-simulate it. In this way, we waive off the cost to track where a node or edge is collapsed, simplifying the maintenance task. The computation cost of adding the counter mechanism over *compute_synopsis* is contributed by the frequency summarization in Algorithm 4 lines 28 and 38. Since extents now can overlap, the frequency updates for vertices (line 28) and edges (line 38) execute $O(|V|^2)$ and $O(|E|^2)$ times in the worst case, leading to a total cost of $O(|E|^2)$. Accordingly, we replace all calls to *compute_synopsis* in *aggregate_synopsis* (i.e., Algorithm 3 lines 7,18 and 28) with *compute_synopsis_freq* to support the frequency maintenance. To ease the presentation, without explicit indication, in the remainder of this paper, we assume that all kd-synopses are frequency-enabled.

**Algorithm 4** *compute_synopsis_freq*

```
1: Input: a simulator set for each vertex $sim(v)$; the original data graph $G = (V, E)$
2: Output: a frequency-enabled kd-synopsis graph $S = (\mathbb{V}, \mathbb{E})$
3: Auxiliary function: $\forall v \in V$, $label(v)$ returns $v$'s label.
4:
5: for all $v \in V$ do
6:     $sim(v) \leftarrow sim(v) - \{v\}$;
7:     for all $u \in sim(v)$ do
8:         $ext(u) \leftarrow ext(u) \cup \{v\}$;
9:     end for
10: end for
11: $final\_set \leftarrow \phi$;
12: for all $v \in V$ do
13:     if $sim(v) = \phi$ then
14:         $final\_set \leftarrow final\_set \cup \{v\}$;
15:     else
16:         $final\_set \leftarrow final\_set - \{v\}$;
17:         if $final\_set \cap sim(v) = \phi$ then
18:             choose any $u \in sim(v)$;
19:             $final\_set \leftarrow final\_set \cup \{u\}$;
20:         end if
21:     end if
22: end for
23: //generate synopsis vertex
24: for all $v \in final\_set$ do
25:     generate a synopsis vertex $v_s$;
26:     $\mathbb{V} \leftarrow \mathbb{V} \cup \{v_s\}$;
27:     establish a mapping from $u \in ext(v)$ to $v_s$;
28:     $freq(v_s) \leftarrow$ the size of $ext(v)$;
29: end for
30: //establish synopsis edge
31: for all $e = (u, v) \in E$ do
32:     $u_s, v_s \leftarrow$ the synopsis vertex corresponding to $u, v$;
33:     if there does not exist $(u_s, v_s) \in \mathbb{E}$ then
34:         generate a synopsis edge $e_s = (u_s, v_s)$ in $S$;
35:         $\mathbb{E} \leftarrow \mathbb{E} \cup \{(u_s, v_s)\}$;
36:         $freq(e_s) \leftarrow 0$;
37:     end if
38:     $freq(e_s) = freq(e_s) + freq(e)$;
39: end for
```

Based on *compute_synopsis_freq*, the maintenance of the routing information becomes straightforward. When a peer $P$ joins, it will send its kd-synopsis $S$ to its direct super-peer $SP$, where a routing entry is established. Then $S$, as the change over $SP$'s routing information, is sent to $SP$'s direct super-peer $SSP$, followed by a call of the aggregation mechanism (Section 4.1) to guarantee that the routing entry still satisfies the space constraint. Then all the changes that have occurred over this routing entry, represented as removal and addition of specific kd-synopses, are sent to $SSP$'s super-peers iteratively. When a peer $P$ leaves, its corresponding routing entry is removed from $SP$'s routing table. If $P$ is a super-peer, we need to move its children and descendant peers under other super-peers. Since this work is focused on synopsis issues, we do not consider a complicated mechanism for super-peer selection, and just move the $P$'s children peers up a level. Accordingly, the routing information corresponding to $P$'s children peers is sent to $SP$, where a new space constraint will be imposed over all routing entries. Afterwards, all the changes over $SP$'s routing information are sent to its super-peer $SSP$, where we take the routing entry corresponding to $SP$, remove and insert kd-synopses accordingly, and adjust the sequence according to the space constraint. Algorithm 5 shows the update of a synopsis sequence with respect to a kd-synopsis' join or leaving, where we use *changes* to collect all the removal and addition of synopses on the current peer. Since this is an incremental update approach, we do not need to generate sequences from a scratch for each join/leaving operation. To illustrate the maintenance process, we show an example of routing information propagation progress in Figure 7, where when $S_5$ is added with $p5$, it is aggregated into $S_{4,6}$ that already aggregates $S_4$ and $S_6$ on super-peer $p7$, and when $S_6$ is removed with the leaving of $p6$, we first figure out that it is aggregated into $S_{4,5,6}$, and then we remove its frequency information from $S_{4,5,6}$ on $p7$. Since $p6$ is a super-peer, we change hand of children and descendant peers to $p7$, with the routing entries adjusted there according to the new space constraint.

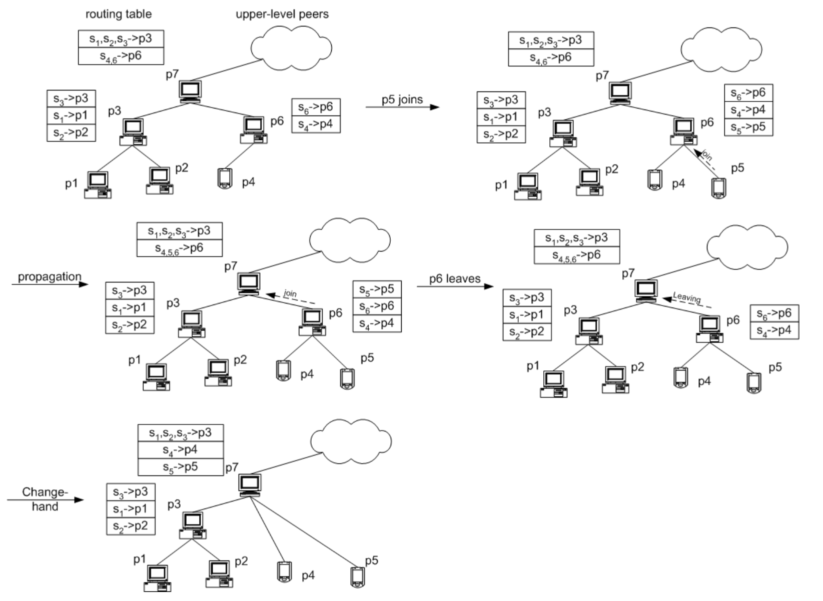


**Figure 7. Routing information update**

## 5 Performance Evaluation

### 5.1 Testbed

We generate random XML data from a repository of XML documents with rich structures, including DBLP [1], TPC-H XML data [9], XMark [12], Treebank [10], and XBench [11, 33]. The size of the data that we populate on each peer is uniformly distributed between 2k and 20kbytes. Because most of the XPath query generators do not consider backward axes (i.e., parent and ancestor axes), we develop a query generator to generate random BPQ$^+$ queries over the repository.

Since there is no widely deployed P2P XML query processing system, we considered using P2P simulators [5, 8]. Unfortunately, they are not directly applicable to this work for two reasons. First, they do not support general hierarchical overlay networks discussed in this work. Both [5] and [8] basically address a two-tier hierarchy. Second, we need a storage model to simulate the size for holding routing tables on super-peers, and it is unclear how to integrate them into the existing simulators. So, we developed a simulator that simulates a super-peer based hierarchical overlay network, where the space constraint on peers is uniformly distributed between 10k to 20k

**Algorithm 5** *update_routing_table*

```
1: Input: $\{S_i\}$, a sequence of (aggregated) synopses on a super-peer; $S = (V, E)$, a synopsis following kd-simulation under $k$ and $d$ values; an operation option $opt \in \{join, leaving\}$; the space constraint $C$ for this sequence
2: Output: the updated sequence of (aggregated) synopses after join/leaving $S$
3: Auxiliary function: $abs(x)$ returns absolute value of $x$
4:
5: if $opt = leaving$ then
6:    for $e \in E$ do
7:       $freq(e) = -abs(freq(e))$;
8:    end for
9: end if
10: $S' \leftarrow \phi$;
11: locate $S'$ in the sequence $\{S_i\}$ such that $S'$ conforms to kd-simulation under $k$ and $d$;
12: if $S' = \phi$ then
13:    if opt = join then
14:       locate the $S'$ that conforms to kd-simulation under $k'$ and $d'$ values, where $k' = k$ and $d'$ is next lower to $d$;
15:       if $S'$ is not an aggregated synopsis then
16:          $S' \leftarrow \phi$;
17:       end if
18:    else
19:       locate the $S'$ that conforms to kd-simulation under $k'$ and $d'$, where $(k', d')$ is next lower to $(k, d)$;
20:    end if
21: end if
22: if $S' \neq \phi$ then
23:    $S \leftarrow compute\_synopsis\_freq(S, k', d')$;
24:    $S_{agg} \leftarrow S \cup S'$;
25:    mark $S_{agg}$ as an aggregated synopsis;
26:    $S_{agg} \leftarrow compute\_synopsis\_freq(S_{agg}, k', d')$;
27:    if opt = leaving then
28:       for all vertices and edges $v, e \in S_{agg}$ such that $freq(e), freq(v) = 0$ do
29:          remove $v, e$ from $S_{agg}$;
30:       end for
31:    end if
32:    remove $S'$ from the sequence and add $S_{agg}$ into the sequence;
33:    record the deletion of $S'$ and insertion of $S_{agg}$ in changes
34: else
35:    //this branch should only satisfy for join operation
36:    add $S$ into the sequence directly;
37:    record the insertion of $S$ in changes
38: end if
39: if (the size of the sequence) $> C$ then
40:    $aggregate\_synopsis(\{S_i\}, C)$;
41:    record every deletion and insertion of synopses in changes
42: end if
43: send changes to the direct super-peer
```

bytes. Specifically, we extract a specific number of maximum spanning trees from transit-stub hierarchical graphs generator by GT-ITM [2], then connect roots of the trees as top-level super-peers, which will broadcast all queries among each other. To simulate the dynamism feature of the P2P networks (i.e., the join and leaving of peers), we follow the common practice [29] to use a poisson distribution function to simulate the arrival of new nodes, and an exponential distribution function to simulate the uptime of peers. Our simulation is conducted centrally under Linux on a 3G MHz PC with 1G memory. Since the Bloom-filter based strategy [24] is a closely related work, we re-implement it and use it as a comparison system in part of the experiments.

## 5.2 Precision and Size of kd-synopsis

In this experiment we try to demonstrate the precision that the Bloom-filter based synopsis [24] and kd-synopsis can provide vis-a-vis the space consumed by them. We use the metric of false-positive ratio to measure the precision. Given a document and a set of queries[10] *negative* to the document, the false-positive ratio is computed by dividing the number of queries that are positive to the synopsis by the size of the query set.For this experiment we extract 250 small documents between 2k to 20k from each document in the repository (e.g., DBLP, XMark) and a certain number(25 in our case) of false(negative) queries are generated over each document.We note the average values including the size of FBSimulation quotient, the size and false-positive ratio of the Bloom-filter based synopsis [24], and kd-synopses under $k \leq 2$ and $d \leq 2$. Here we take FBsimulation quotient into account so as to show that kd-synopses can be significantly smaller. The experimental results are recorded in Table 1. It is obvious that

[10]For comparison purpose, in this experiment we consider a subset of BPQ$^+$ queries containing only forward axes, since Bloom-filter based synopsis can not handler backward axes.

with the increasing of $k$ and $d$ values, the false-positive ratio of the kd-synopsis decreases sharply, while the size of the synopsis keeps smaller than Bloom-filter based synopsis, which demonstrates the effectiveness of kd-synopsis in balancing the precision and size.

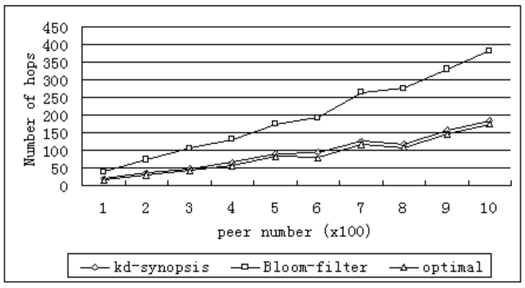
| | doc | FBQ | BF | | kd (0, 0) | | kd (0, 1) | | kd (0, 2) | | kd (1, 0) | | kd (1, 1) | | kd (1, 2) | | kd (2, 0) | | kd (2, 1) | | kd (2, 2) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | size | size | S | FP | S | FP | S | FP | S | FP | S | FP | S | FP | S | FP | S | FP | S | FP | S | FP |
| 1 | 8612 | 1211 | 1609 | 0.99 | 286 | 1 | 569 | 0 | 569 | 0 | 318 | 0.997 | 875 | 0 | 876 | 0 | 318 | 0.997 | 875 | 0 | 876 | 0 |
| 2 | 8142 | 4246 | 2400 | 1 | 573 | 1 | 1465 | 0.103 | 1742 | 0.035 | 573 | 1 | 2796 | 0.075 | 3167 | 0 | 573 | 1 | 2910 | 0.075 | 3311 | 0 |
| 3 | 8293 | 3538 | 6840 | 0.758 | 669 | 0.79 | 1357 | 0.26 | 1716 | 0.115 | 1137 | 0.395 | 1965 | 0.193 | 2297 | 0.062 | 1787 | 0.315 | 3032 | 0.142 | 3282 | 0.031 |
| 4 | 10418 | 2348 | 4000 | 0.936 | 1158 | 1 | 1240 | 0.398 | 1244 | 0.398 | 1233 | 0.208 | 1378 | 0.001 | 1422 | 0.001 | 1294 | 0.001 | 1375 | 0.001 | 1507 | 0 |
| 5 | 11303 | 3871 | 2400 | 0.818 | 717 | 0.059 | 834 | 0 | 834 | 0 | 806 | 0.059 | 1233 | 0 | 1233 | 0 | 806 | 0.059 | 1233 | 0 | 1233 | 0 |

1: DBLP 2: TPC-H 3: Treebank 4: XBench(DCSD) 5: Xmark
FBQ: FBsimulaion quotient BF: Bloom-filter based synopsis S: size (in bytes) FP: False-positive ratio

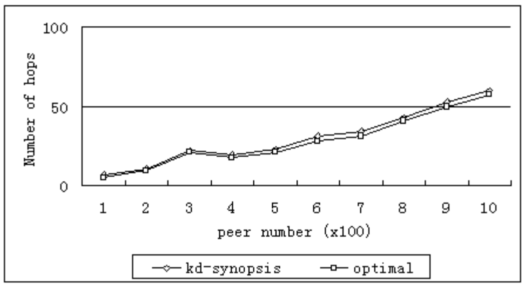
**Table 1. Precision and size**

## 5.3 Query Shipping Performance

In this experiment, we use the number of hops on the overlay network for routing queries to their relevant data as the metric of the query shipping cost. We consider randomly generated networks up to 1000 peers, where each peer keeps one XML document randomly generated over the repository, with its size between 2k and 20k bytes. To demonstrate the advantage of the kd-synopsis over Bloom-filter based synopsis, we deploy the two synopses as the routing synopsis separately over the same network, and use 100 queries to test the average shipping cost. We also compare the query shipping cost with the optimal shipping cost, defined as the smallest number of hops to be used to ship queries to all the peers that the queries are positive to. Since Bloom-filter based synopsis does not handle backward axes (i.e., parent and ancestor axes) in BPQ$^+$ queries, we run separate experiments for queries with and without backward axes, where we only compare performance of kd-synopsis and Bloom-filter based synopsis under the latter scenario. Figure 9(a) shows the comparison between the kd-synopsis and Bloom-filter based synopsis, which demonstrates that the shipping cost based on kd-synopsis is 53% smaller than that based on Bloom-filter based synopsis. Moreover, the shipping cost based on kd-synopsis keeps close to the optimal shipping cost, further demonstrating the effectiveness of the kd-synopsis. In Figure 9(b), we test the shipping cost for a different set of BPQ$^+$ queries including backward axes. The result once again shows that the average cost is very close to the optimal shipping cost.



(a)



(b)

**Figure 8. Query shipping cost**

In this experiment, we also demonstrate that the positive check of queries against kd-synopsis is still time-efficient, and wins over the Bloom-filter based synopsis. Again, we use the same set of BPQ$^+$ queries that do not contain backward axes. The average time for positive check is recorded for networks up to 1000 peers, and the result (Figure 9) shows that the time of positive check against kd-synopsis is sharply (86%) lower than that against Bloom-filter based synopsis. since this work is focused on the design of a routing synopsis rather than a routing protocol, the time we measure here only captures the computation time required by the routing decision making, not including the communication latency on the physical network.

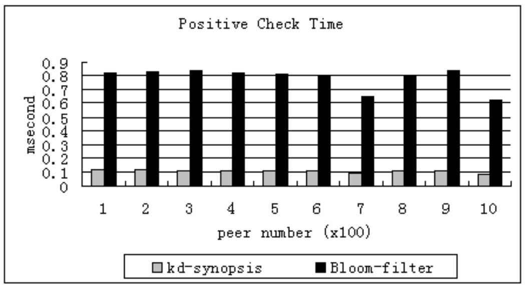


**Figure 9. Routing decision making cost**

### 5.4 Maintenance Cost

The join of peers is modeled by Poisson distribution, and the uptime of each peer is independently modeled by using exponential distribution. Whether a peer join or leave, the update of the routing information will be propagated upward the overlay topology until it reaches the top-level peer, so we can focus on a tree-like network topology. To evaluate the maintenance cost, we measure two metrics: first, the number of hops needed to propagate the update, which basically depends on the network topology and network size; second, the average size of the update messages, which is proportional to the communication cost incurred by the maintenance. The experiment results are shown in Figure 10, where in Figure 10 (a), we illustrate the average number of hops for peer join and leaving over tree-like networks containing 10 to 100 peers, and in Figure 10 (b), we show the average size of the update messages for the same setting. All the data are collected during 10 logical time intervals, with the mean of Poisson distribution as 5 (i.e., on average, 5 peers join the network per unit time), and the mean of the exponential distribution as 5 (i.e., on average, a peer leaves the network after 5 time units). Since the leaving of a super-peer needs other super-peers to take over its children peers, the maintenance cost is expected to be higher, so we also compare the average size of update messages regarding the leaving of plain peers and super-peers (in Figure 10 (c)), which demonstrates that on average, the former (i.e., super-peers leave the network as well) takes cost than the latter when the network becomes large.

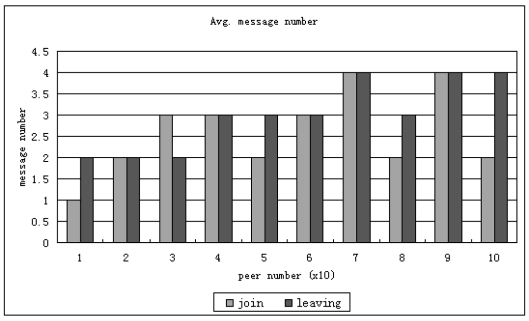
## 6 Conclusion

We discussed the design of a novel routing synopsis for distributed XML query processing over P2P network. The primary problem we are interested in is how to locate distributed XML data relevant to XML queries in a super-peer based unstructured P2P network. A key part of the problem is on the design of a routing synopsis that can capture the rich hierarchical information of XML data. Taking both the precision of routing synopses and the heterogeneity of P2P systems into account, we propose kd-synopsis, a novel synopsis that is built over length constrained FBsimulation relationship. Based on this routing synopsis, we address the issues on query routing and routing information maintenance. Experiments show that our scheme is much cheaper on query shipping than a comparison system proposed by Koloniari [24].

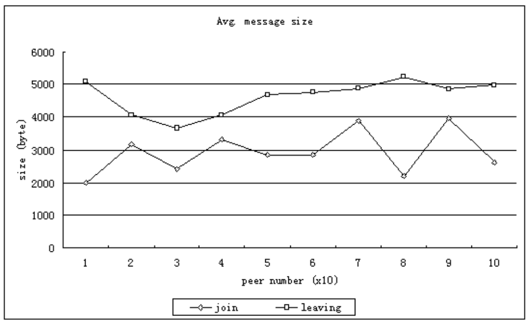
Since we are considering BPQ$^+$ that is a subset of XPath language, we expect to extend our strategy to include more constructs such as IDREF, negation operator, sibling axes and others. To implement those, we need to capture order and ID/IDREF information from the data. These information must be encoded together with the basic graph-structure in a compact way so as to avoid a blowup of the synopsis size. Another problem we are working on is the optimization of queries to facilitate the check of the positive of queries against routing information. Finally, building a super-peer based overlay network topology that takes multiple factors (e.g., content similarity, storage size, network bandwidth) into account is still an open and interesting problem on its own.

## References

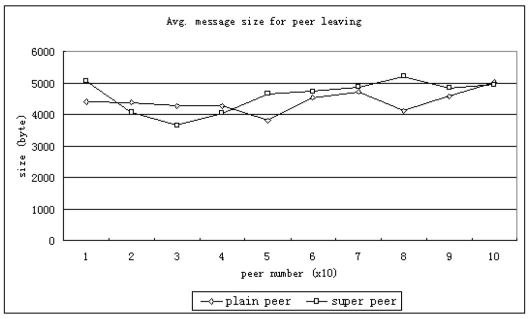
[1] DBLP. http://dblp.uni-trier.de/xml/.



(a)



(b)



(c)

**Figure 10. Maintenance cost**

[2] Georgia Tech Internetwork Topology Models. http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html.

[3] Gnutella v0.4. http://www9.limewire.com/developer /gnutella_protocol_0.4.pdf.

[4] Gnutella v0.6. http://rfc-gnutella.sourceforge.net /Proposals/Ultrapeer/Ultrapeers.htm.

[5] GnutellaSim. http://www.cc.gatech.edu/computing/ compass/gnutella/.

[6] KaZaA. http://www.kazaa.com.

[7] Limewire. http://www.Limewire.com.

[8] Peersim. http://peersim.sourceforge.net/.

[9] TPC-H. http://www.cs.washington.edu/research/ xmldatasets/www/repository.html#tpc-h.

[10] Treebank. http://www.cis.upenn.edu/ treebank/.

[11] XBench. http://db.uwaterloo.ca/ ddbms/projects/xbench/.

[12] XMark. http://monetdb.cwi.nl/xml/index.html.

[13] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.

[14] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernandez, M. Kay, J. Robie, and J. Simeon. XML path language (XPath) 2.0 W3C working draft. http://www.w3.org/TR/xpath20/, November 2003.

[15] B. Bloom and R. Paige. Transformational design and implementation of a new efficient solution to the ready simulation problem. *Science of Computer Programming*, 24:189–220, 1995.

[16] A. Bonifati, U. Matrangolo, A. Cuzzocrea, and M. Jain. XPath lookup queries in P2P networks. In *International Workshop on Web Information and Data Management*, 2004.

[17] A. Deshpande, S. K. Nath, P. B. Gibbons, and S. Seshan. Cache-and-query for wide area sensor databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 503–514, 2003.

[18] G. Erice, P. A. Felber, E.W. Biersack, G. Urvoy-Keller, and K.W.Ross. Data indexing in peer-to-peer DHT networks. 2004. Proc. 24rd Int. Conf. on Distributed Computing Systems.

[19] L. Galanis, Y. Wang, S. R. Jeffery, and D. J. DeWitt. Locating data sources in large distributed systems. In *Proc. 29th Int. Conf. on Very Large Data Bases*, pages 874–885, 2003.

[20] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. In *Proc. 28th Int. Conf. on Very Large Data Bases*, 2002.

[21] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *Proc. IEEE Symp. Foundations of Computer Science*, pages 453–462, 1995.

[22] R. Kaushik, P. Bohannon, J. Naughton, and H. Korth. Covering indexes for branching path queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2002.

[23] R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. Exploiting local similarity for indexing paths in graph-structured data. In *Proc. Int. Conf. on Data Engineering*, pages 129–140, 2002.

[24] G. Koloniari and E. Pitoura. Content-based routing of path queries in peer-to-peer systems. In *Advances in Database Technology — EDBT'04*, pages 29–47, 2004.

[25] R. Krishnamurthy, V. T. Chakaravarthy, R. Kaushik, and J. F. Naughton. Recursive XML schemas, recursive XML queries, and relational storage: XML-to-SQL query translation. In *Proc. Int. Conf. on Data Engineering*, 2004.

[26] B. T. Loo, J. M. Hellerstein, R. Huebsch, S. Shenker, and I. Stoica. Enhancing P2P file-sharing with an Internet-scale query processor. In *Proc. 30th Int. Conf. on Very Large Data Bases*, pages 432–443, 2004.

[27] T. Milo and D. Suciu. Index structures for path expressions. In *Proc. 7th Int. Conf. on Database Theory*, 1999.

[28] J. Min, M. Park, and C. Chung. XPRESS: a queriable compression for XML data. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 122 – 133, 2003.

[29] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter P2P networks. In *Proc. IEEE Symp. Foundations of Computer Science*, pages 492–499, 2001.

[30] N. Polyzotis and M. N. Garofalakis. Statistical synopses for graph-structured XML databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 358–369, 2002.

[31] P. Ramanan. Covering indexes for xml queries: Bisimulation - simulation = negation. In *Proc. 29th Int. Conf. on Very Large Data Bases*, pages 165–176, 2003.

[32] P. M. Tolani and J. R. Haritsa. XGRIND: A query-friendly XML compressor. In *Proc. Int. Conf. on Data Engineering*, pages 225–234, 2002.

[33] B. B. Yao, M. T. Özsu, and N. Khandelwal. XBench benchmark and performance testing of XML DBMSs. In *Proc. 20th Int. Conf. on Data Engineering*, pages 621–633, 2004.