# FLUX: Fuzzy Content and Structure Matching of XML Range Queries

Hua-Gang Li   S. Alireza Aghili   Divyakant Agrawal   Amr El Abbadi
Department of Computer Science, University of California, Santa Barbara, CA 93106
{huagang, aghili, agrawal, amr}@cs.ucsb.edu

## ABSTRACT

An XML range query may impose predicates on the numerical or textual contents of the elements and/or their respective path structures. In order to handle content and structure range queries efficiently, an XML query processing engine needs to incorporate effective indexing and summarization techniques to efficiently partition the XML document and locate the results. In this paper, we propose a dynamic summarization and indexing method, FLUX, based on Bloom filters and B$^+$-trees to tackle these problems. The results of our extensive experimental evaluations indicated the efficiency of the proposed system.

**Categories and Subject Descriptors:** H.2.4 [Database Management]: Systems – *Query Processing*

**General Terms:** Algorithms.

**Keywords:** XML database, XPath, Range Query.

## 1. INTRODUCTION

The XML data model, due to its rich presentation (content and semi-structuredness) poses unique challenges to effectively support complex queries. Queries on such ordered trees generally impose predicates on the textual content of `ELEMENT` (keyword search) and/or their corresponding *structural relationships* (structure pattern search). Numerous research efforts have been conducted [1, 5] to provide powerful and flexible query capabilities to extract structural patterns from XML documents. Nevertheless, how to efficiently address the structural pattern queries with range predicate support over the textual content of `ELEMENT` is not addressed yet. In this paper, the class of *content-and-structure (CAS)* single path queries are extended to include (*i*) *range* predicates, and (*ii*) *fuzzy content-and-structure* predicates. We call this class of queries *Fuzzy-Range (FuR)* since they provide efficient support for *exact* and *approximate (fuzzy)* matching of queries with *path*, *content* and *range* predicates. The fuzzy/approximate matching is supported as an extra feature without requiring any specific instructions from the user. For example, the query

`Q = /dblp//article[2004` $\leq$ `/year` $\leq$ `2005]`

matches the journal articles published in the year range `[2004-2005]` from `dblp` database [3]. *Fuzzy* ***structure*** *matching* feature additionally reports those instances that match the range predicate of the query, but whose structure *resembles* the query's structure, for instance

`p`$_1$ `= /dblp/article/year/2005`, and
`p`$_2$ `= /dblp/articles//year//2004`.

Due to the heterogeneity of the XML data, it is essential to provide the support for *fuzzy matching* in current XML query engines. This paper introduces the proposition of an XML query processing system for FuR queries named FLUX. FLUX employs an efficient B$^+$-tree based index structure to locate the leaf matches $n_i$ to the range predicate of a query in its initial stage. Each leaf match $n_i$ of the document tree stores a *compact path signature* of the root-to-leaf path structure of $n_i$, using the notion of Bloom Filters [2]. In the next step, the path signatures of each matched leaf instance $n_i$ is compared with the query's path signature to eliminate those instances whose path signature is very different from that of the query. To the best of our knowledge, this is the first work to specifically address exact and approximate matching of FuR class of queries in XML document collections. The following sections will roughly provide the range and path matching procedures. Details can be found from our FLUX technical report [4].

## 2. RANGE MATCHING

The *range matching phase* of FLUX employs an indexing technique based on B$^+$-trees on the indexable attributes (elements) (e.g., numerical, textual, date, ...) of the XML document dataset for the effective reduction of the search space. Figure 1(2) depicts a portion of one such index tree, constructed on the `age` attribute of a typical XML employee database. Each key in a leaf bucket corresponds to the content of an indexable element and the data associated with it stores the element ID information (pre-order traversal rank of the corresponding node in the XML document) and the bit-vector signature of the actual path leading to the node (more details provided in the following section). For instance, the node instance with `age` $= 66$ has preorder rank of 72, which is shown in the document tree of Figure 1, named as the node $^{72}66$.

## 3. PATH MATCHING

The range matching phase provides all the matching instances residing inside the range of the given FuR query. The *path matching phase* performs the necessary steps to identify those instances whose path component (approximately) matches the path expression of the given FuR query. It is mainly based on the bit-vector signature for each instance. Bloom filter [2] is a space-efficient data structure to proba-

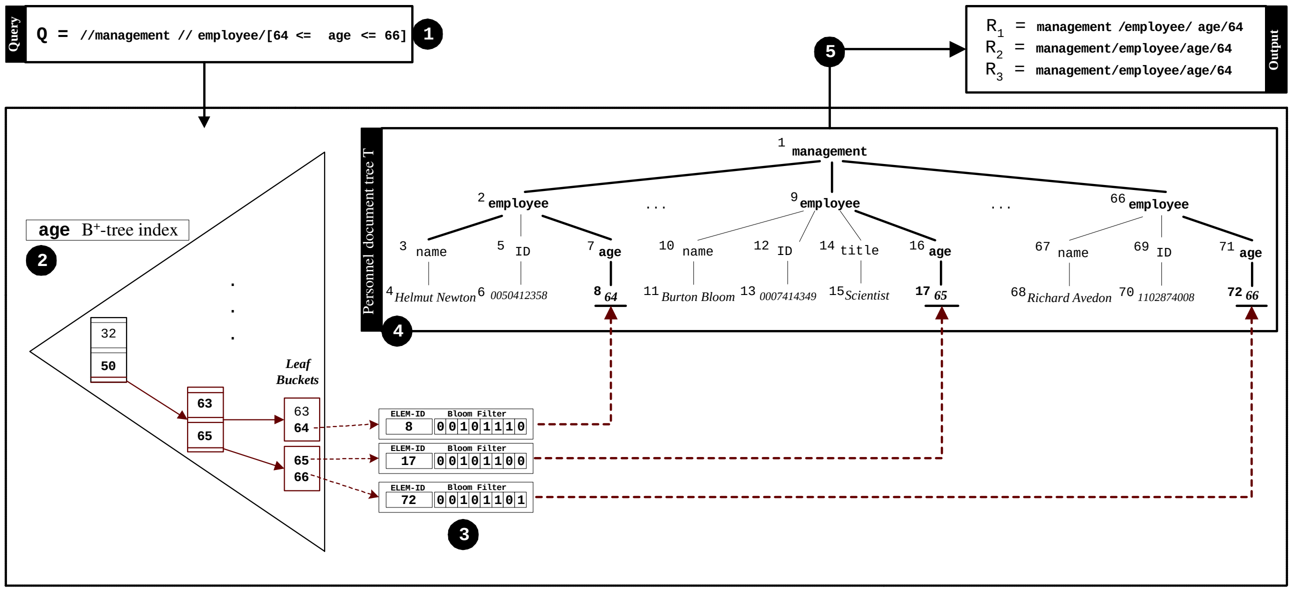

**Figure 1: FLUX Search Model.**

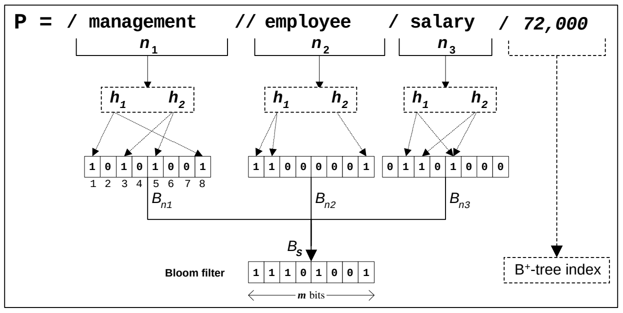

**Figure 2: A Bloom Filter Example.**

blistically represent a set and its elements using bit-vector signature to support highly accurate *set membership* queries. Figure 2 depicts the construction of a bloom filter bit-vector signature using two independent $m = 8$-bit hash functions $h_1$ and $h_2$, on the path set $S = \{$ `management, employee, salary`$\}$, where $n = |S| = 3$, from an employee database. Given a query's path component, the bloom filter for it is constructed. The bloom signatures of all the matching candidate instances obtained from the range matching phase will be checked against the query path's bloom signature to test how closely they are matched. If the calculated distance among the bit-vector signatures is smaller than a the application's similarity threshold, then the instance will be reported as a matching one.

## 4. EXPERIMENTAL EVALUATION

We implemented the FLUX system using *Java 1.4.2* and ran our experiments on a *Pentium M-2GHz* processor with *2GB* of main memory, using a page size of 1KB (determine the number of indexed data items a leaf node can have and number of key/pointers an internal node can have for the B+-tree.), cache size of 100KB, and LRU cache replacement policy. We compared our proposed technique with PathStack [5] which is one of the leading techniques in the literature for simple XPath queries. The experimental evaluations were performed on a set of both synthetic (XMark [6]) and real (dblp) XML datasets. Random noises were imposed on the datasets to create path structure variation at the element names. We studied the effect of *query range length*, *bloom filter size*, and the *scalability analysis* on the filtration efficacy, false positive rate (introduced by Bloom filter), and response time of FLUX. The experimental results are promising and demonstrate that FLUX consistently outperforms PathStack. Also, more details on the experimental results can be referred to from our FLUX technical report [4].

## 5. CONCLUSION

This paper proposed an efficient technique, named FLUX, for answering complex range queries in a database of XML documents. FLUX incorporated a $B^+$-tree based index structure on the contents of range attributes. It uses the notion of Bloom filters to associate a structure signature to each range attribute instance. The filtration performed by the bloom signatures of FLUX reduced the search space to a minor fraction of the intermediate result set.

## 6. REFERENCES

[1] S. Al-Khalifa, H.V. Jagadish, J.M. Patel, Y. Wu, N. Koudas and D. Srivastava, Structural Joins: A Primitive for Efficient XML Query Pattern Matching. ICDE, 141–152 (2002).

[2] B.H. Bloom, Space/Time Trade-offs in Hash Coding with Allowable Errors. Communications of the ACM **13(7)**, 422–426 (1970).

[3] DBLP Bibliography Server, *http://dblp.uni-trier.de/*.

[4] H.-G. Li, S. A. Aghili, D. Agrawal and A. El Abbadi, FLUX: Fuzzy Content and Structure Matching of XML Range Queries. *http://www.cs.ucsb.edu/research/tech_reports/reports/2005-24.pdf/*.

[5] N. Bruno, N. Koudas and D. Srivastava, Holistic twig joins: optimal XML pattern matching. SIGMOD, 310–321 (2002).

[6] A. R. Schmidt et al., The XML Benchmark Project. Technical Report INS-R0103, CWI (2001).