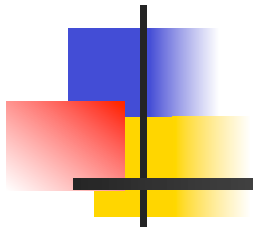


Architecture and Implementation of Databases

Last part: XML Databases



Ralf Möller

Hamburg University
of Technology

This lecture is based on the following presentation

Holistic Twig Joins

Optimal XML Pattern Matching



Nicolas Bruno

Columbia University

Nick Koudas

AT&T Labs-Research

Divesh Srivastava

With extensions from Shing-Hang Wang

SIGMOD 2002



XML Query Processing

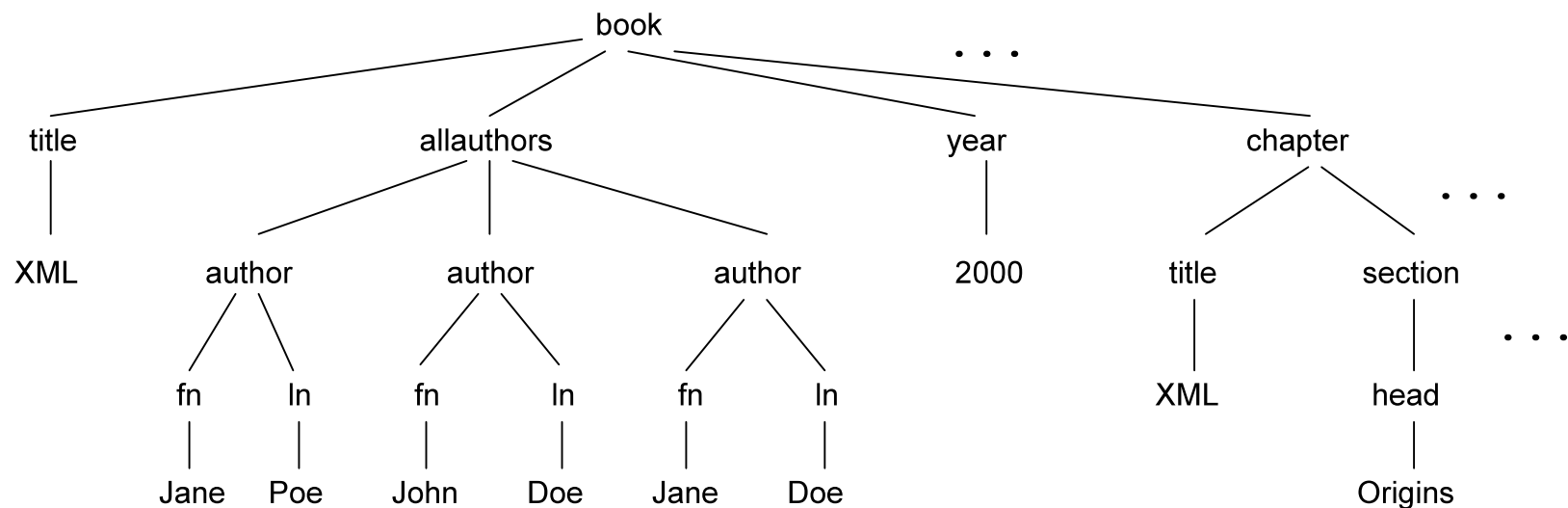
- XML query languages are complex, with many features.
- Natural and pervasive operation: matching XML data with a tree structured pattern.
- Previous attempts decompose query into small pieces and solve them separately: complex optimization problem.



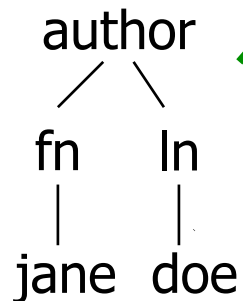
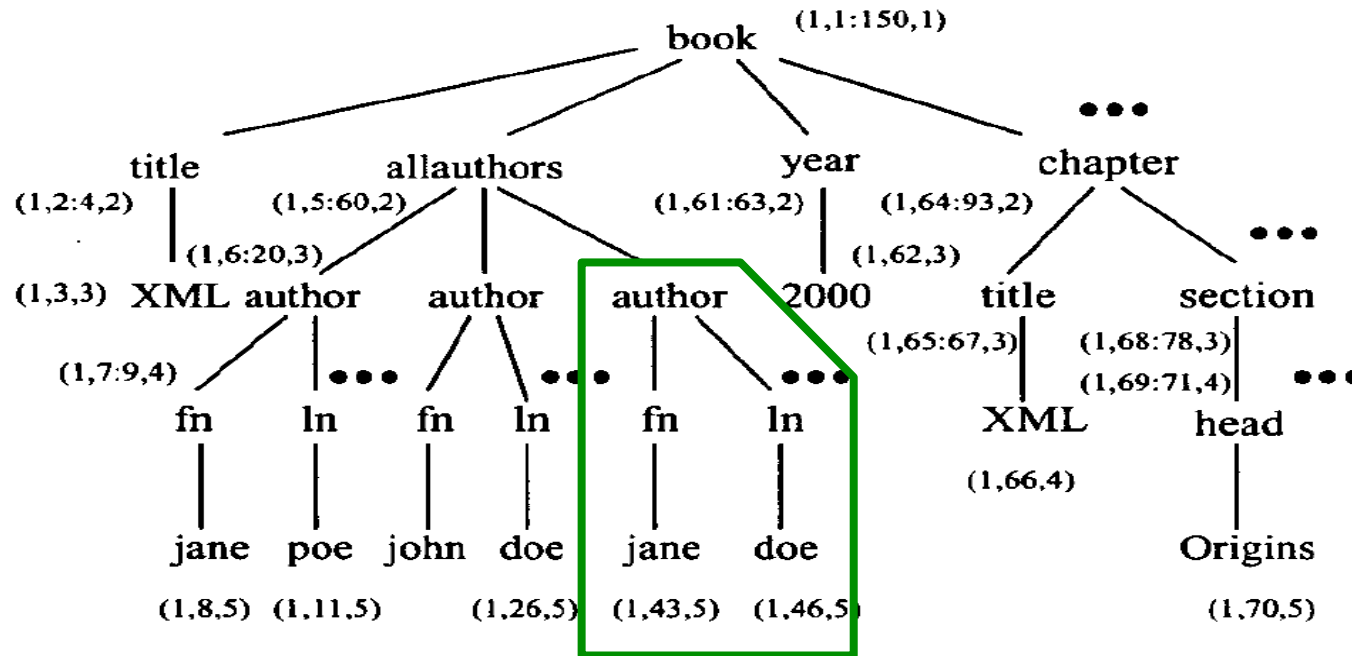
Data Model

XML database: forest of rooted, ordered, labeled trees:

- Nodes represent elements or values.
- Edges model direct containment properties.



Example query



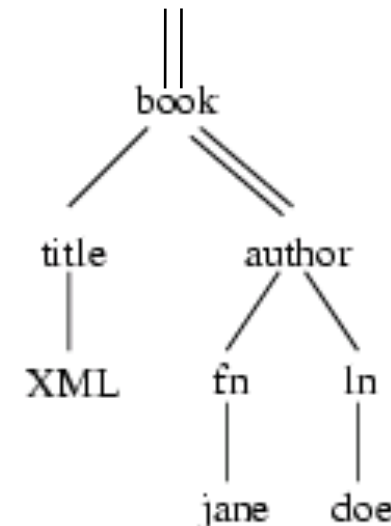
Query twig patterns

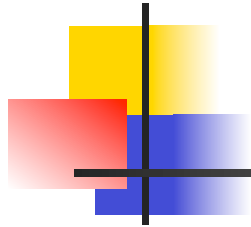
Query Model: Subset of XQuery

Specific twig patterns can match relevant portions of the XML database.

Find the year of publication of all books about "XML" written by "Jane Doe".

```
FOR $b IN document("books.xml")//book
  $a IN $b//author
WHERE contains($b/title, 'XML') AND
  $a/fn = 'jane' AND
  $a/ln = 'doe'
RETURN
  <pubyear> $b/year <pubyear/>
```





Outline

- Problem formulation.
- PathStack: Path Queries.
- TwigStack: Twig Queries.
- XB-Trees: Sub-linear pattern matching.
- Experimental evaluation.



Twig Pattern Matching

Given a query twig pattern Q and an XML database D , compute the set of all matches for Q on D .

Exploit indexes over the XML document:
document not needed in main memory.

Indexing XML Documents

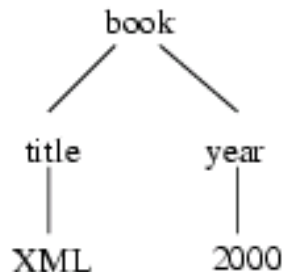
- Element positions represented as tuples (**DocID, Left:Right, Level**), sorted by **Left**.
- Child and descendant relationships between elements easily determined.

author	→ (1,1:150,1) (1,180:200,1) ...
book	→ (1,6:20,3) (1,22:40,3) ...
jane	→ (1,8:8,5) (1,43:43,5) ...
...	...
title	→ (1,65:67,3) ...
XML	→ (1,66:66,4) (2,140:140,6) ...
year	→ (1,61:63,2) (1,88:90,2) ...

Extension to
classical IR
inverted lists

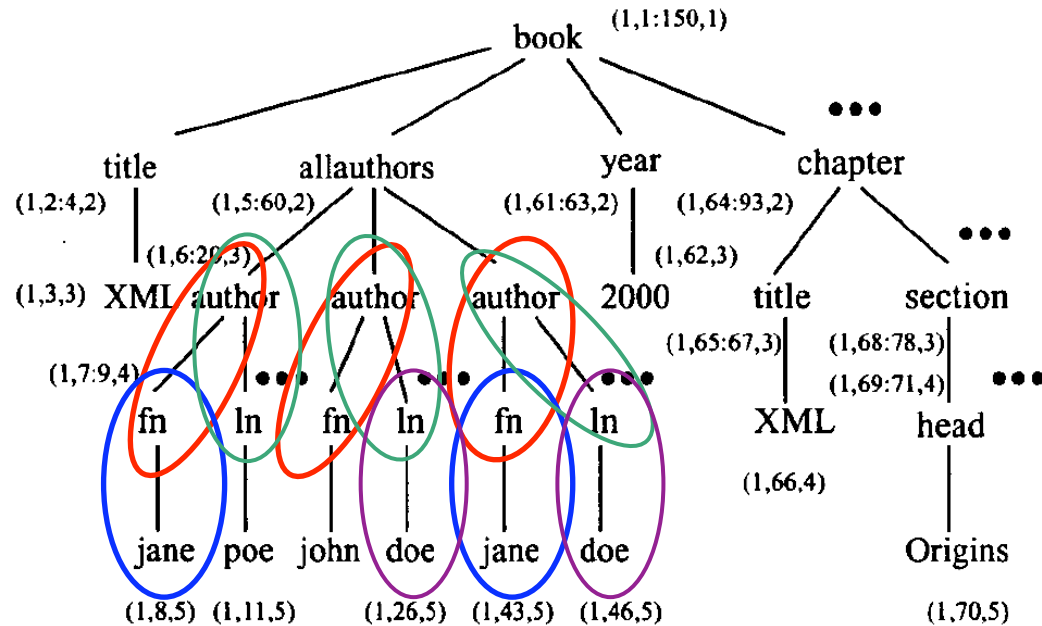
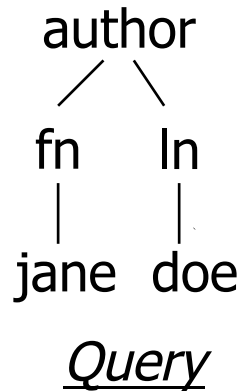
Previous Attempts

- Based on binary joins [Zhang'01, Al-Khalifa'02].
 - Decompose query into binary relationships.
 - Solve binary joins against XML database.
 - Combine together "basic" matches.
- Main drawbacks:
 - Optimization is required.
 - Intermediate results can be large.



- ((book ⋈ title) ⋈ XML) ⋈ (year ⋈ 2000)
 - (((book ⋈ year) ⋈ 2000) ⋈ title) ⋈ XML
- many other possibilities...

Binary Structural Joins



Decomposition

author-fn → **3**
 fn-jane → **2**

author-ln → **3**
 ln-doe → **2**



New Approach: Holistic Joins

- Solve the entire twig query in two phases:
 - 1- Produce “guaranteed” partial results using one pass.
 - 2- Combine (merge join) partial results.
- Partial result smaller than final result.
- Exploit indexes.
 - Skip irrelevant document fragments.
 - Use containment relationships between query nodes.



Data Structures

- Each node q in query has associated:
 - A **stream** T_q , with the positions of the elements corresponding to node q , in increasing “left” order.
 - A **stack** S_q with a compact encoding of partial solutions (stacks are chained).

A₁
|
C₁
|
A₂
|
C₂
|
B₁
|
D₁

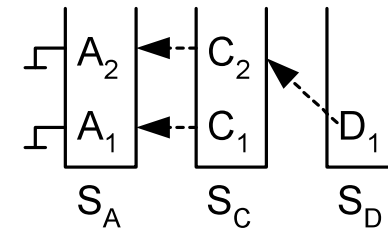
XML fragment

A
||
C
||
D

Query

[A₁, C₁, D₁]
[A₁, C₂, D₁]
[A₂, C₂, D₁]

Matches



Stacks



PathStack: Holistic Path Queries

- Repeatedly constructs stack encodings of partial solutions by iterating through the streams T_q .

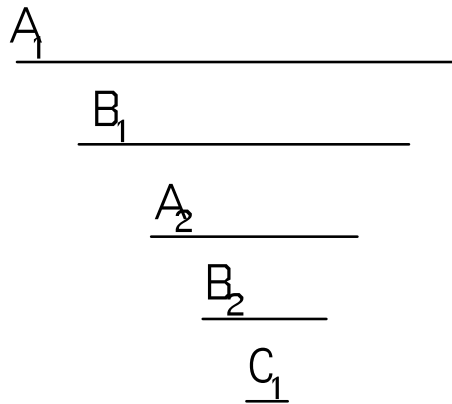
```
WHILE (!eof)
  qN = "getMin(q)"
  clean stacks
  push  $T_{qN}$ 's first element to  $S_{qN}$ 
  IF qN is a leaf node, expand solutions
```

- Stacks encode the set of partial solutions from the current element in T_q to the root of the XML tree.

PathStack Example 1

Document

A₁
|
B₁
|
A₂
|
B₂
|
C₁



Query

A
||
B
||
C

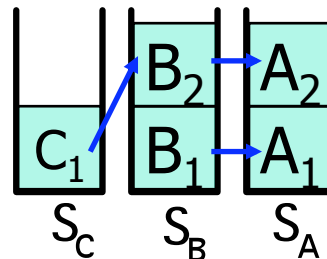
Streams

T_A: A₁, A₂

T_B: B₁, B₂

T_C: C₁

Stacks



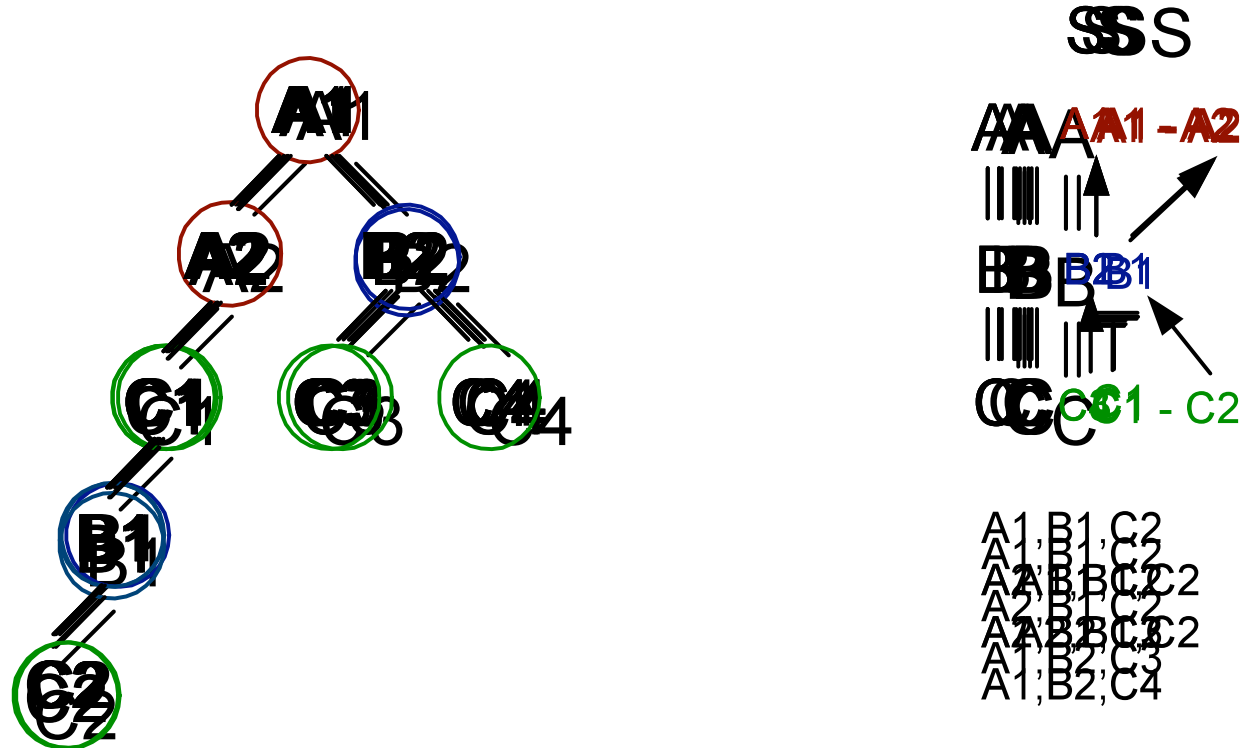
Output

C₁ B₂ A₂

C₁ B₂ A₁

C₁ B₁ A₁

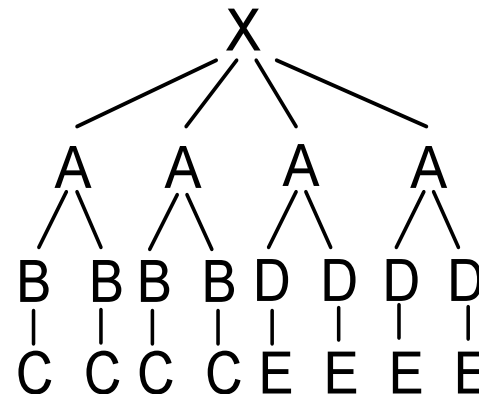
PathStack Example 2

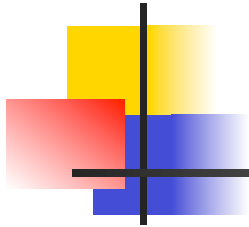


■ **Theorem:** PathStack correctly returns all query matches with $O(|\text{input}| + |\text{output}|)$ I/O and CPU complexity.

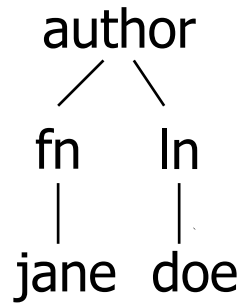
Twig Queries

- Naïve adaptation of PathStack.
 - Solve each root-to-leaf path independently.
 - Merge-join each intermediate result.
- Problem: Many intermediate results might not be part of the final answer.

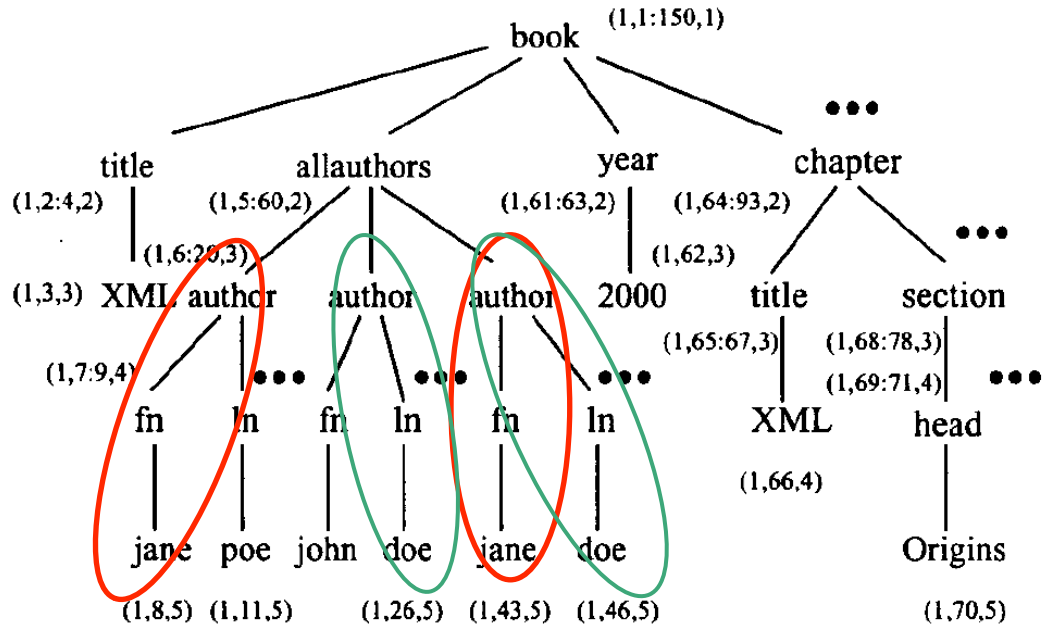




Path Stack



Query



Docume

nt

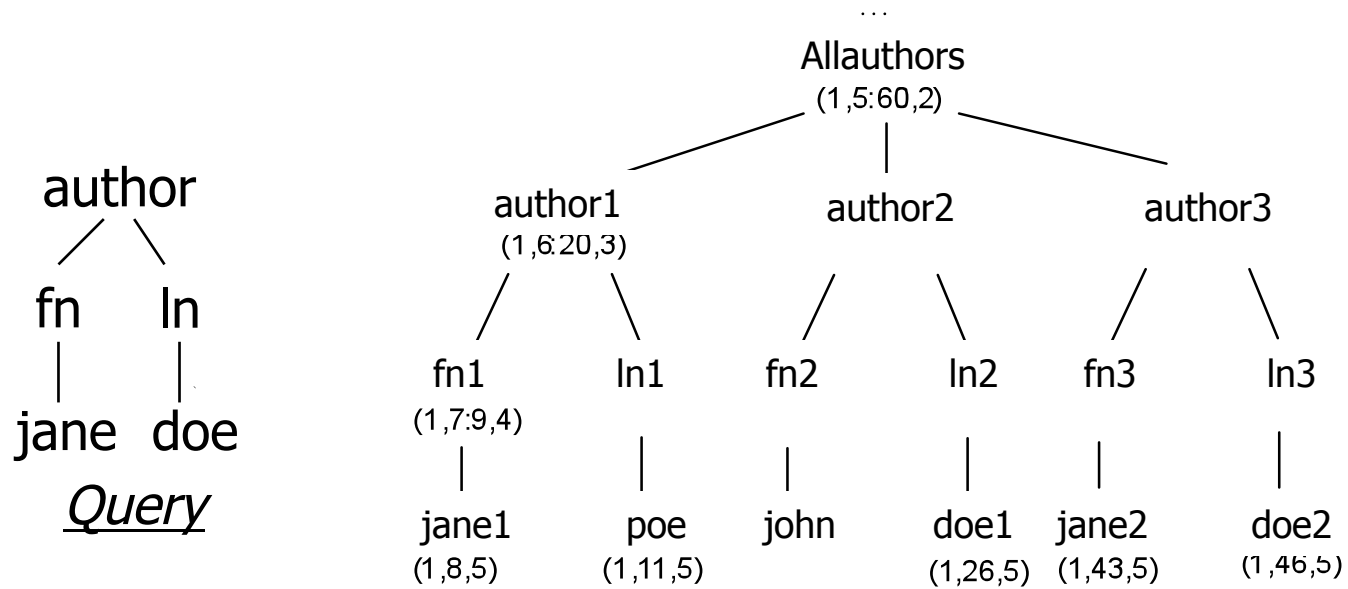
Decomposition

author-fn-jane → **2**

author-ln-doe → **2**



Twig Stack



Streams

T_a : a1, a2, a3

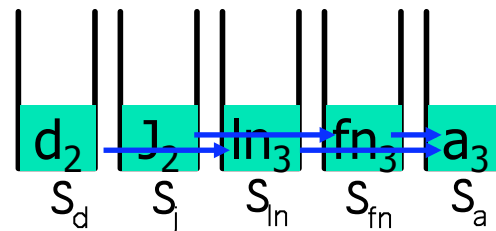
T_{fn} : fn1, fn2, fn3

T_{ln} : ln1, ln2, ln3

T_j : j1, j2

T_d : d1, d2

Stacks



Document

1 -> (j2, fn3, a3)

2 -> (d2, ln3, a3)



TwigStack

- 1) Compute only partial solutions that are guaranteed to extend to a final solution.

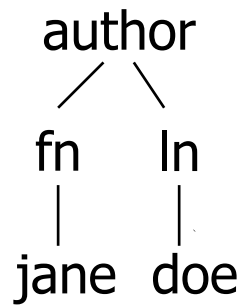
```
WHILE (!eof)
  qN = "getNext(q)"
  clean stacks
  IF TqN's first element is part of a solution, push it
  IF qN is a leaf node, expand solutions
```

getNext might advance the streams in subTree(q) that are guaranteed not to be part of a solution

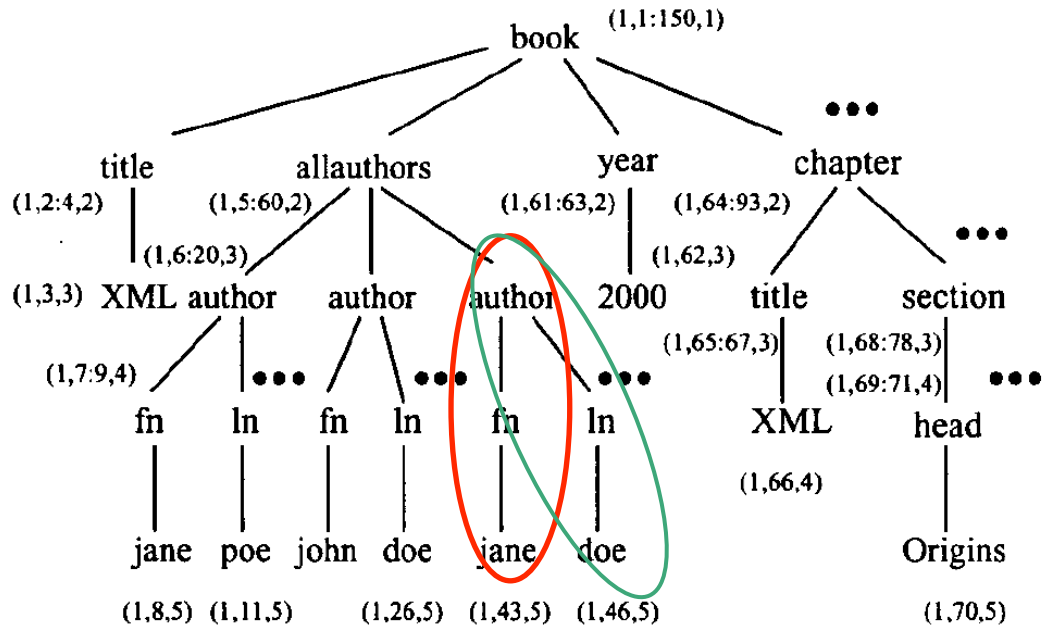
- 2) Merge partial solutions to obtain all matches.



Twig Stack



Query



Document

Decomposition

author-fn-jane → **1**

author-ln-doe → **1**



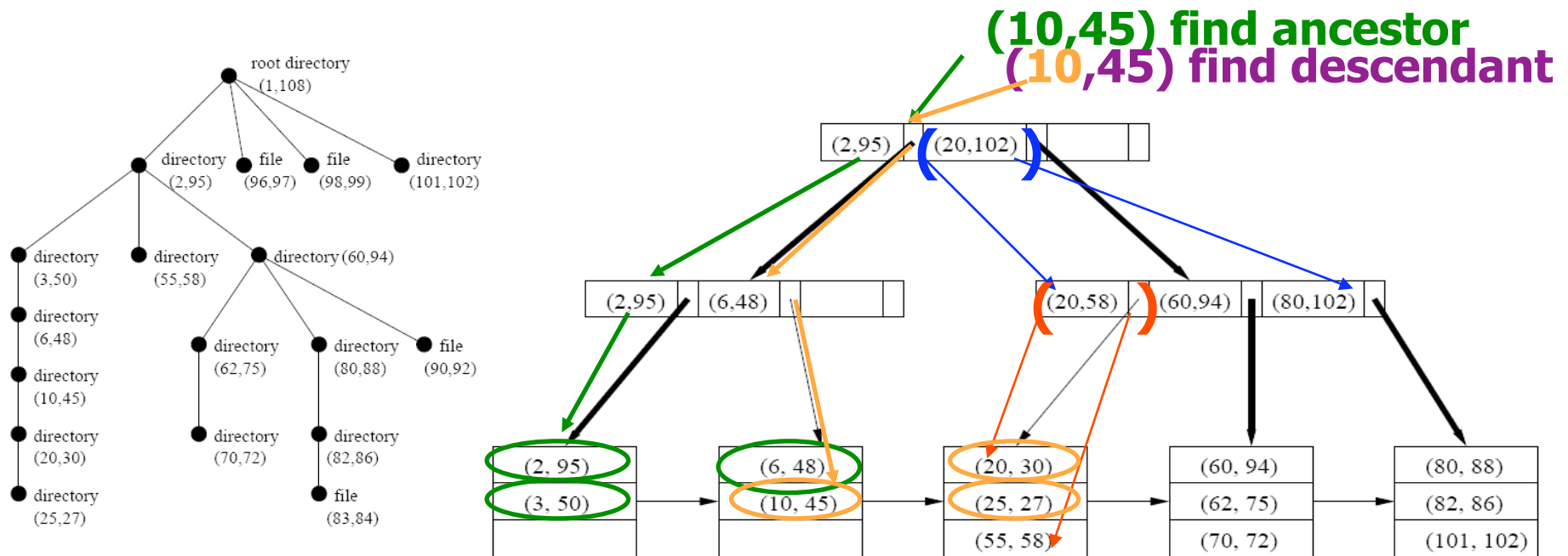
Analysis of TwigStack

- If $\text{getNext}(q)=q_N$, then:
 - Sub-tree q_N has a solution using the stream heads.
 - q_N is “maximal”.
- getNext returns nodes in topological order.
- Stacks encode the set of partial solutions from the current element in getNext to the root of the XML tree.
- **Theorem:** TwigStack correctly returns all query matches with $O(|\text{input}|+|\text{output}|)$ I/O and CPU complexity for ancestor/descendant relationships.



XB-Tree

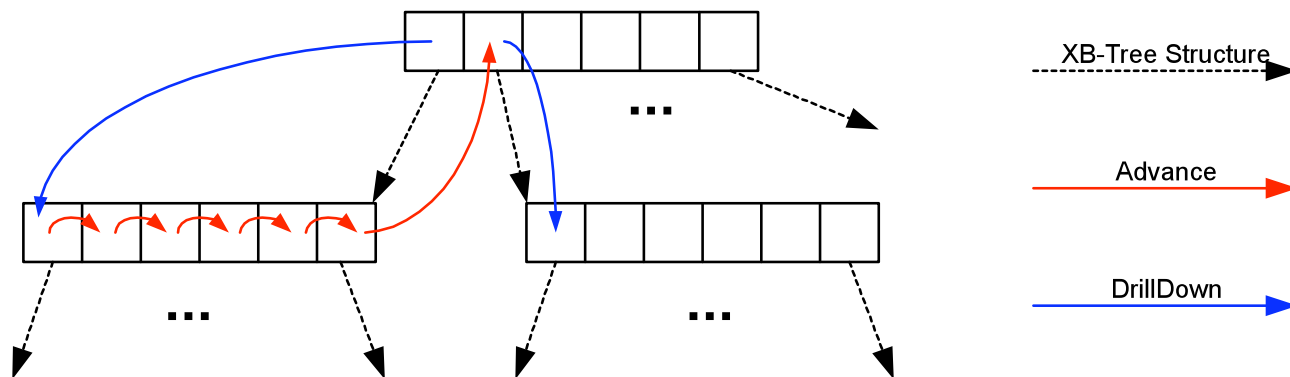
- Add an index to increasing efficiency.
- XB-Trees are like R-tree and B⁺-trees
 - Internal nodes have the form [L:R], sorted on L
 - Parent node interval includes child node intervals



XB-Trees: A Variant of B-Trees

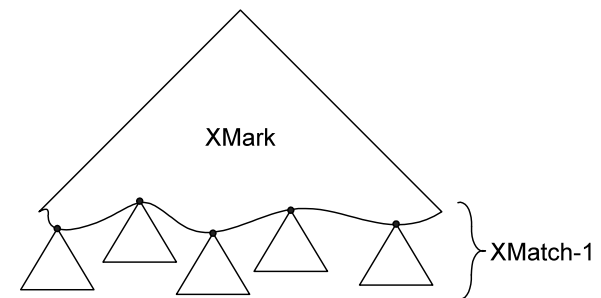
- Index positions of elements in the document.
- Allows adaptive granularity for consuming streams: advance and drillDown.

TwigStack can be adapted to use XB-Trees with minimal changes.

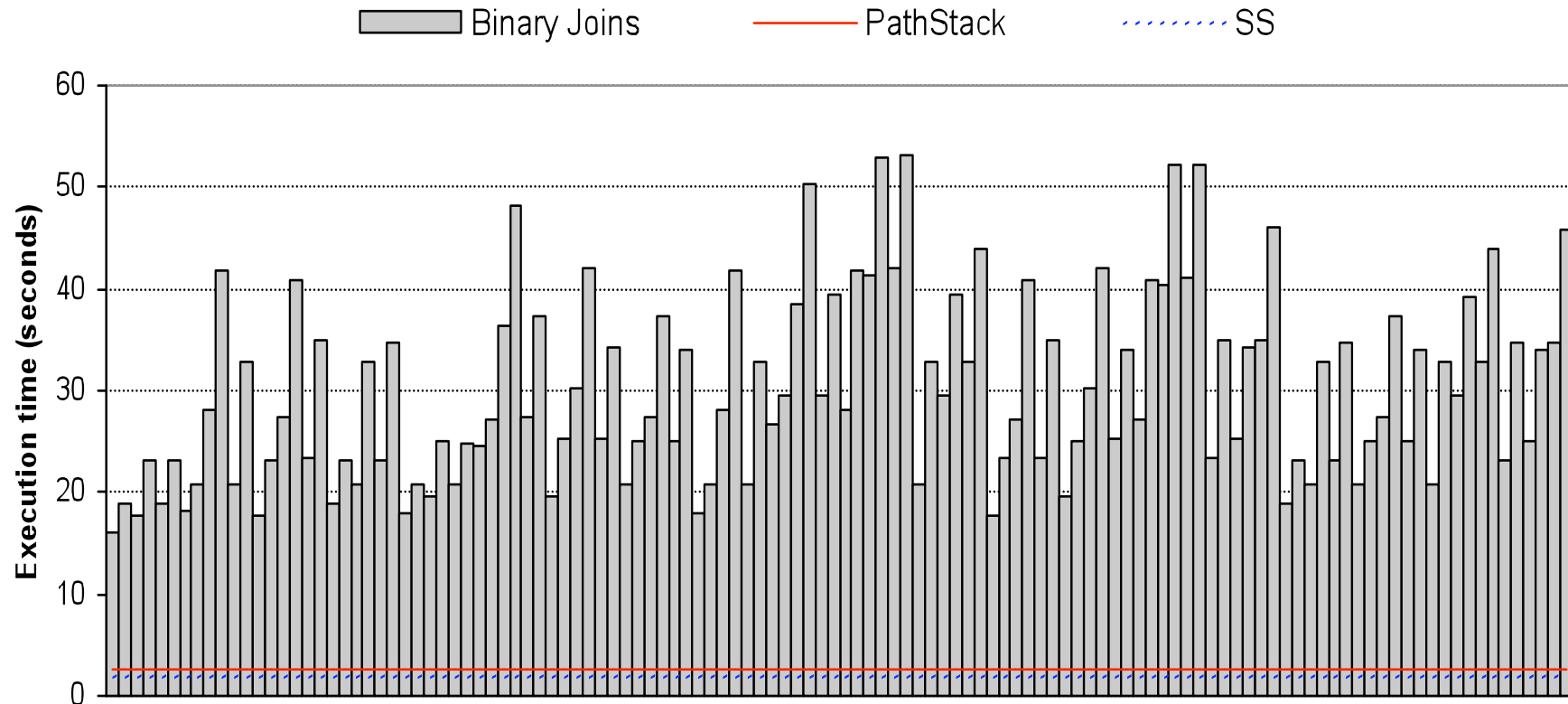


Experimental Setting

- Implemented all algorithms in C++ using the file system as a simple storage engine.
- Synthetic and real databases.
 - Unfolded DBLP database.
 - X-Match + X-Mark benchmarks.
 - Random XML documents.
- Techniques compared:
 - Binary Join techniques.
 - PathStack.
 - TwigStack.

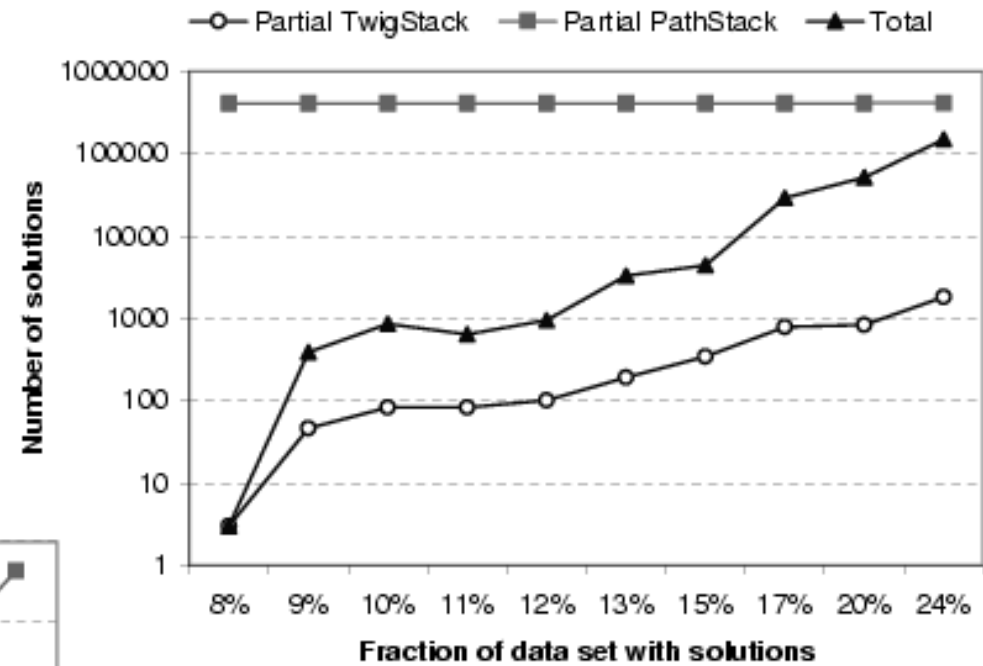
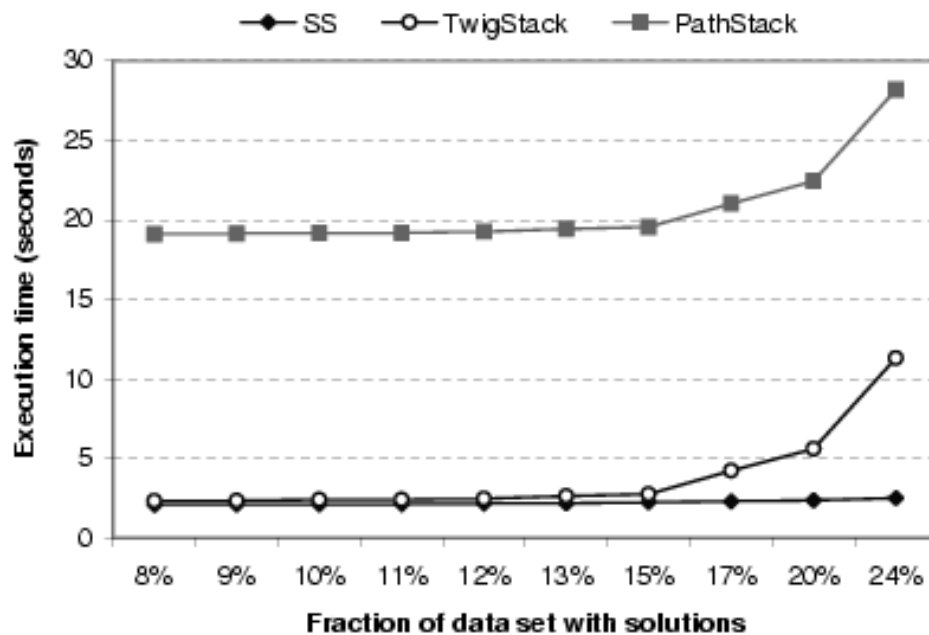


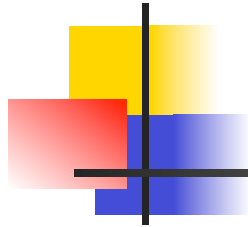
PathStack vs. Binary Joins



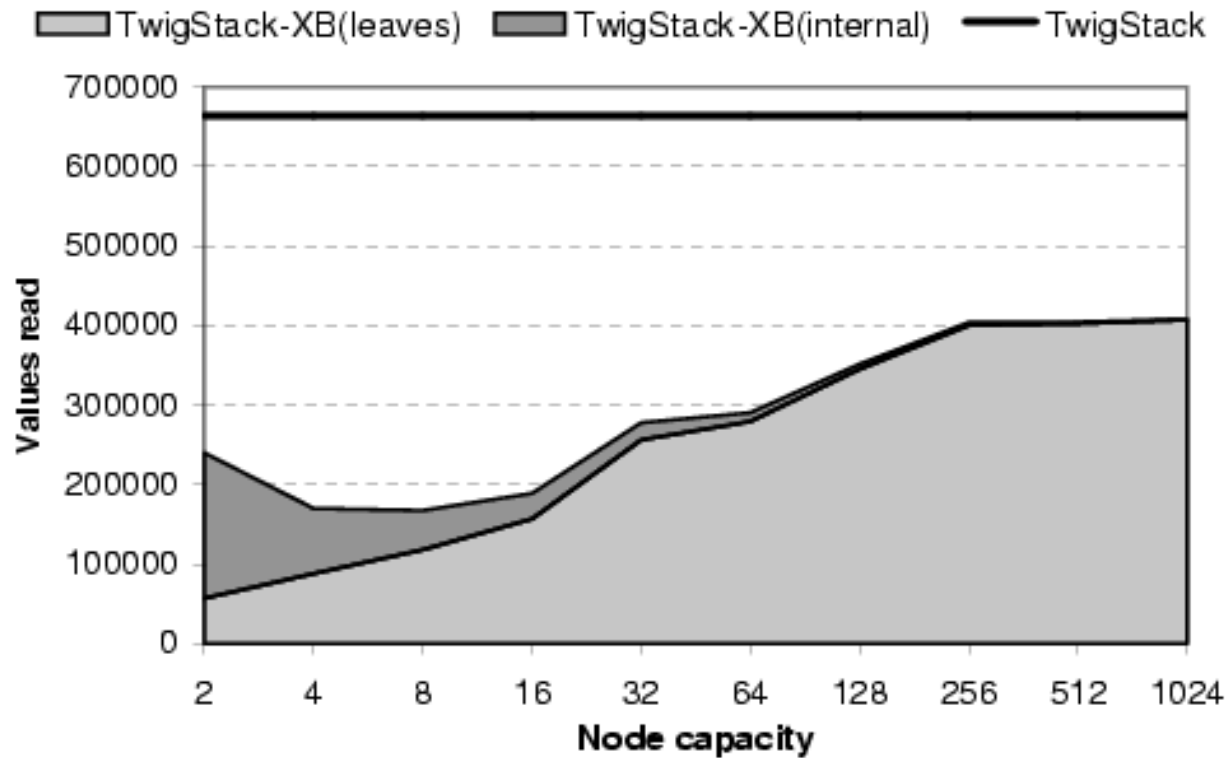
XML database fragment: 1 million nodes.
Path Query: A1//A2//A3//A4//A5//A6

PathStack vs. TwigStack





XB-Trees



XML database fragment: 1 million nodes.
Twig Query



Summary

- Holistic path join algorithms independent of size of intermediate results.
- TwigStack generalizes PathStack for twig queries.
- XB-Trees integrated to TwigStack.