

# Tree Pattern Relaxation

Sihem Amer-Yahia

AT&T Labs–Research

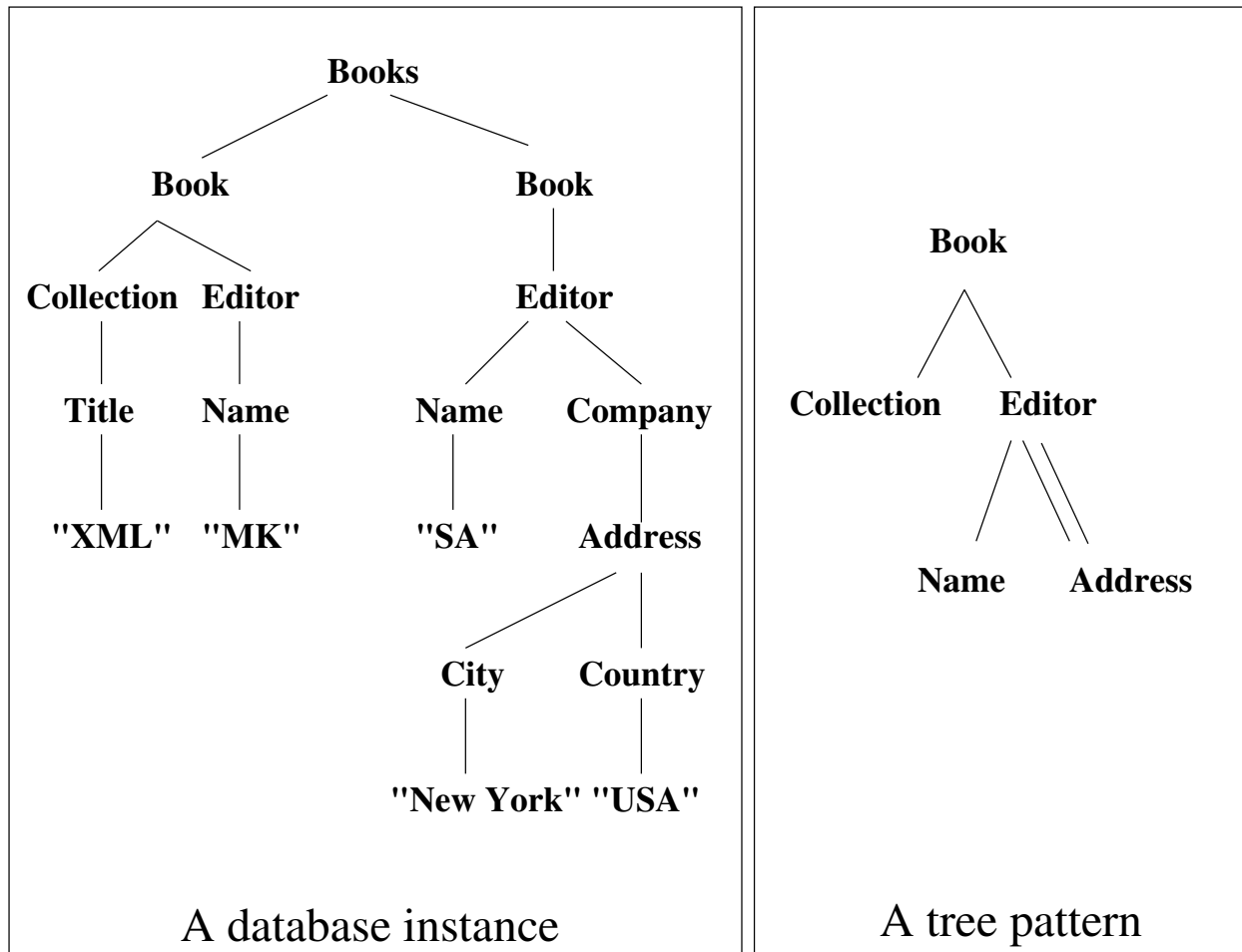
<http://www.research.att.com/~sihem/>

Joint work with SungRan Cho and Divesh Srivastava (EDBT'02)

# Outline of Talk

- XML preliminaries
- Why relaxations?
- Which relaxations?
- Threshold problem
- Experimental results

# XML Example: Database and Query



# Motivation: Why Approximate Matching?

- Traditional query semantics: exact matching
  - embedding query in database instance
- Motivation for approximate matching: XML data heterogeneity
  - schema might not be defined or known
  - schema often allows optional elements
  - too few or no exact matches

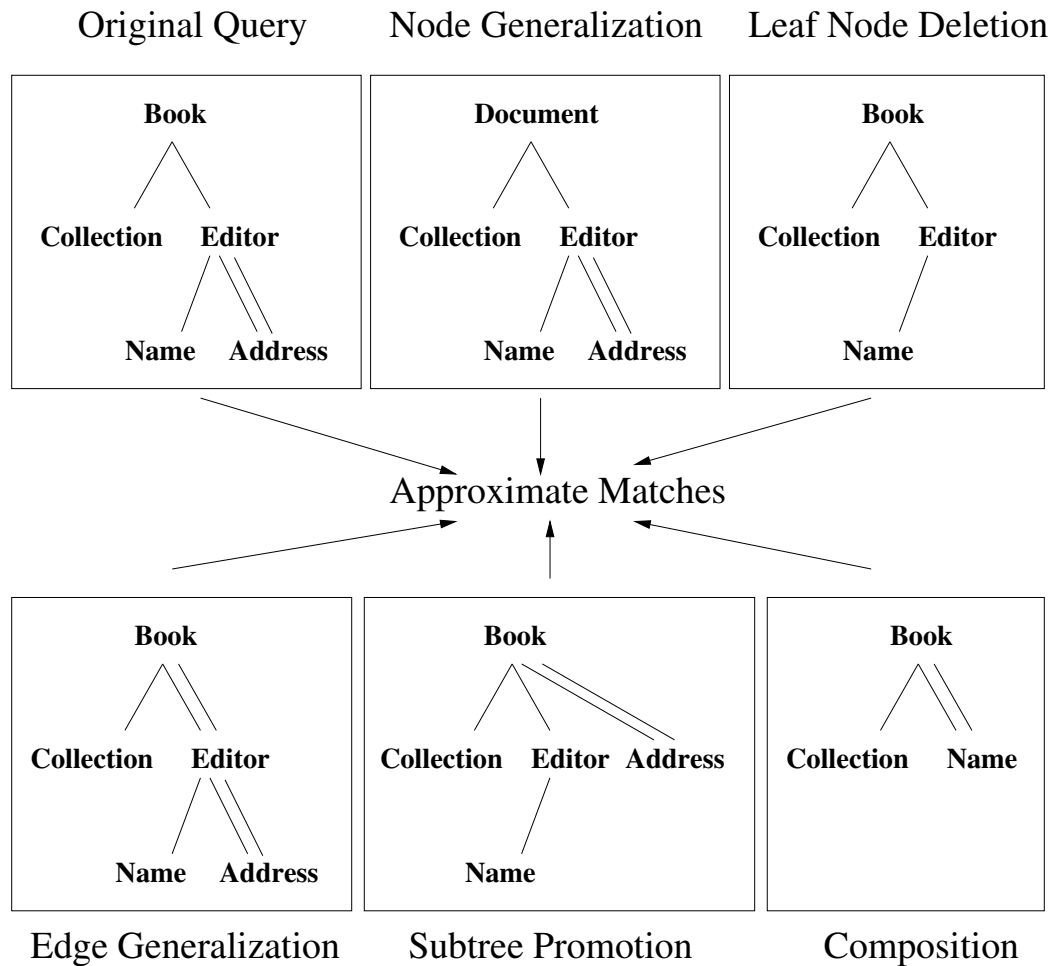
## Motivation: Approximation via Relaxation

- Relaxation = single query modification
  - compute superset of exact matches
- Relaxed query  $Q' = Q +$  composition of relaxations
- Approximate match to  $Q =$  exact match to  $Q'$ 
  - enumerate relaxed queries  $Q'$  of  $Q$
  - efficiently compute exact matches to each  $Q'$
- Goal: return “best” approximate matches in ranked order

## Which Query Relaxations?

- Node Generalization: node type  $\rightarrow$  supertype
- Leaf Node Deletion: remove leaf node and edge to parent
- Edge Generalization: pc-edge  $\rightarrow$  ad-edge
- Subtree Promotion: subtree  $\rightarrow$  ad-edge with grandparent

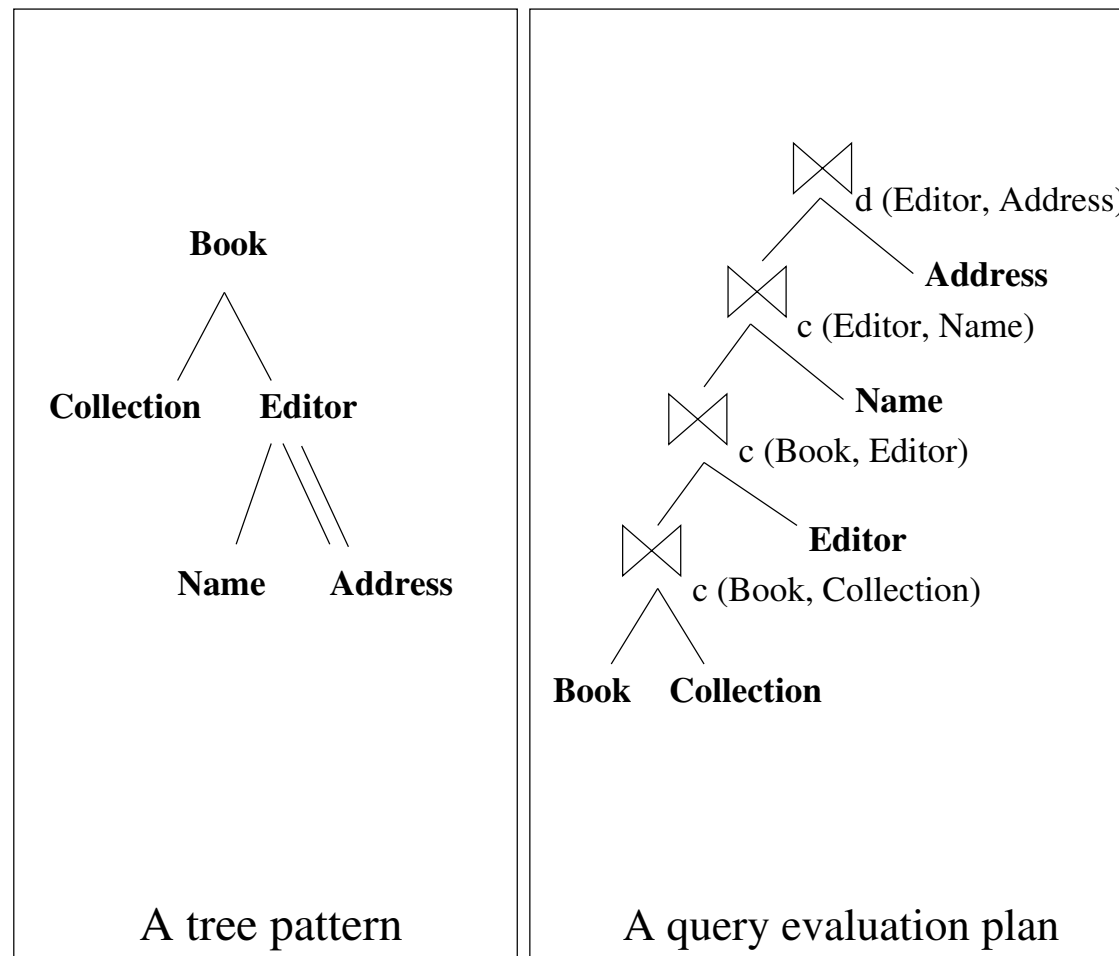
# Example: Relaxed Queries



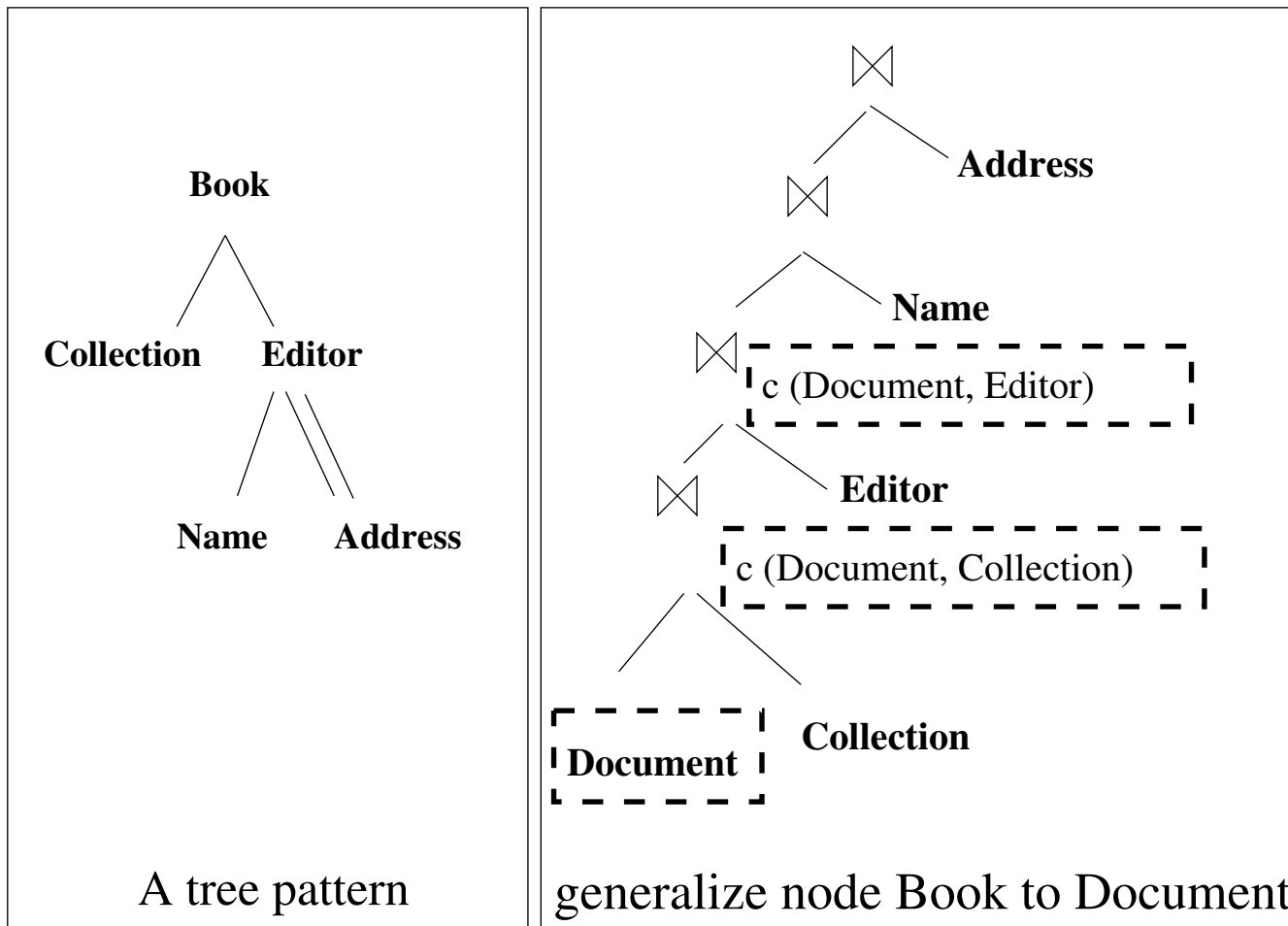
# Efficient Evaluation of Relaxed Queries

- Exact matching of original query: use a join plan
  - binary containment/structural joins
- Encode desired query relaxations into the join plan
  - node generalization: node predicate  $\rightarrow$  supertype
  - leaf node deletion: innerjoin  $\rightarrow$  outerjoin
  - edge generalization: join predicate  $c(A, B) \rightarrow d(A, B)$
  - subtree promotion: modified join predicate

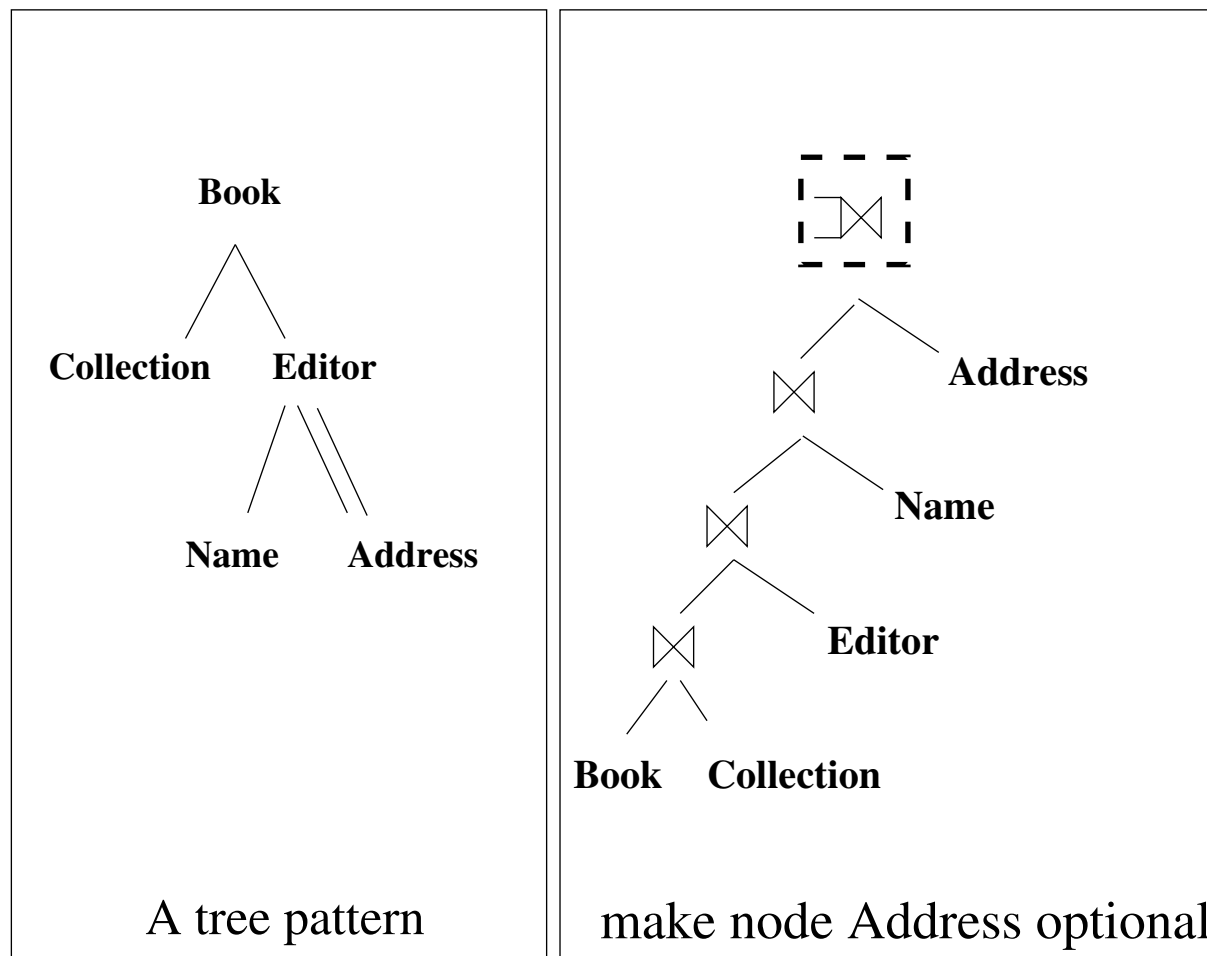
## Example: Original Join Plan



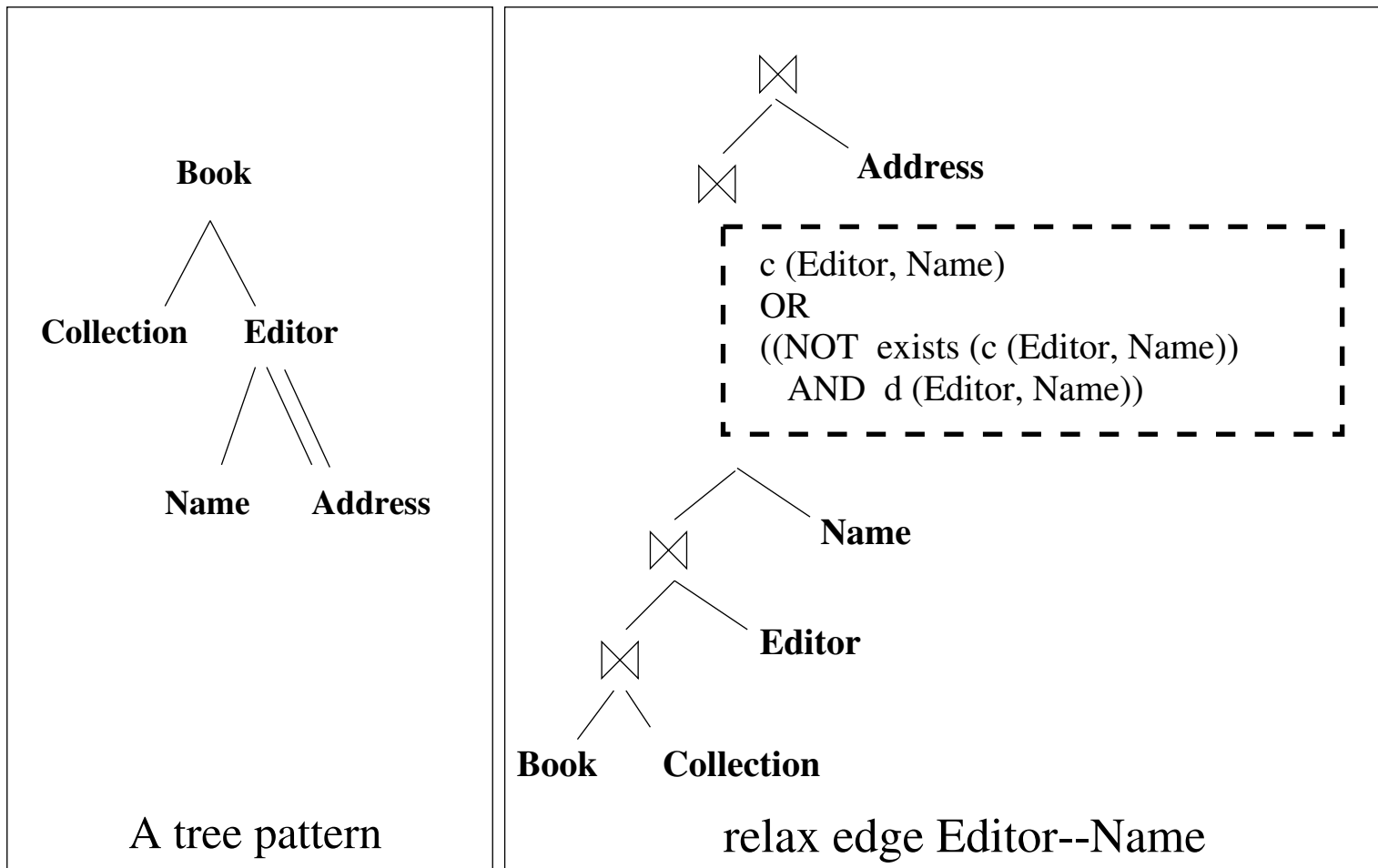
## Example: Encoding Node Generalization



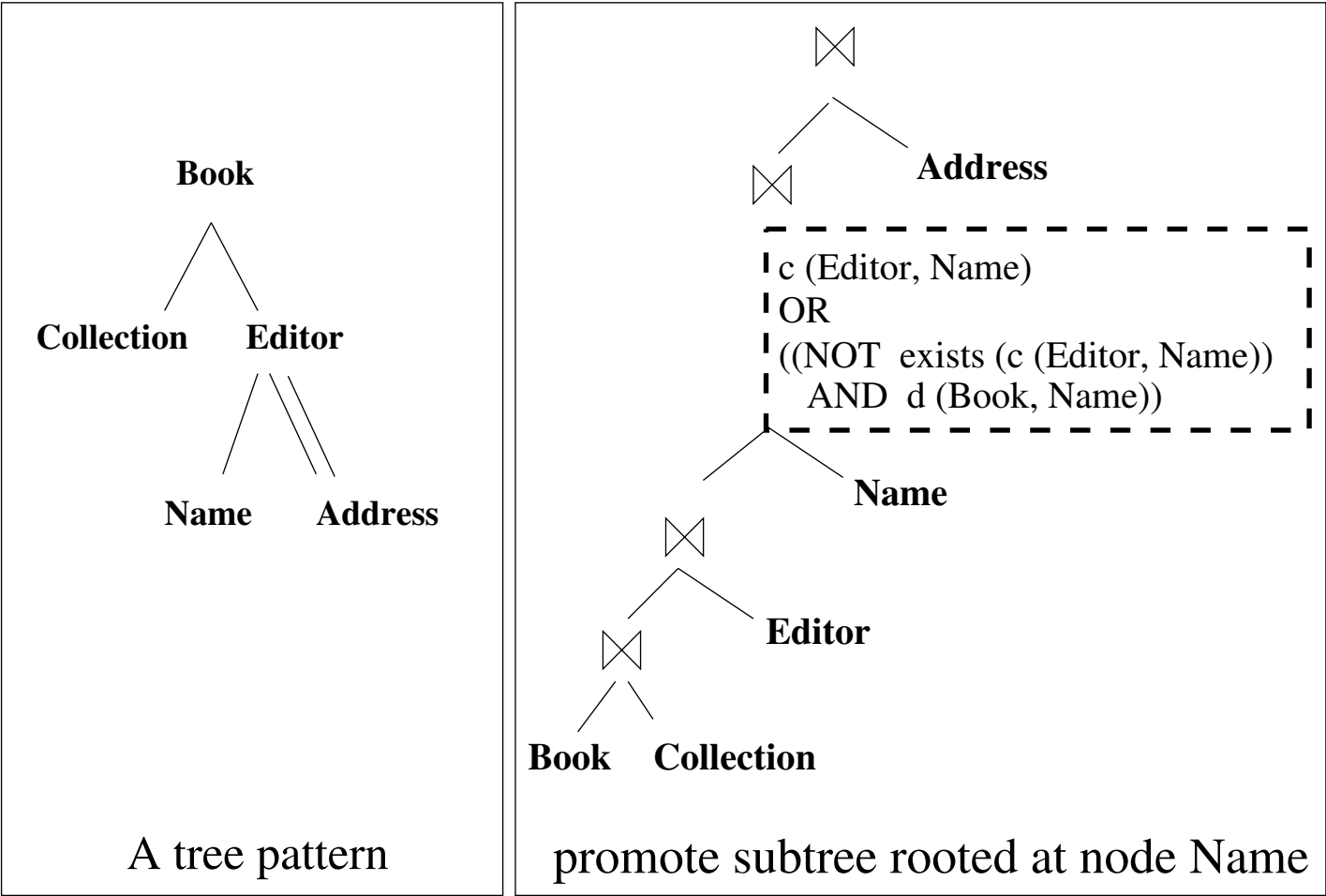
## Example: Encoding Leaf Node Deletion



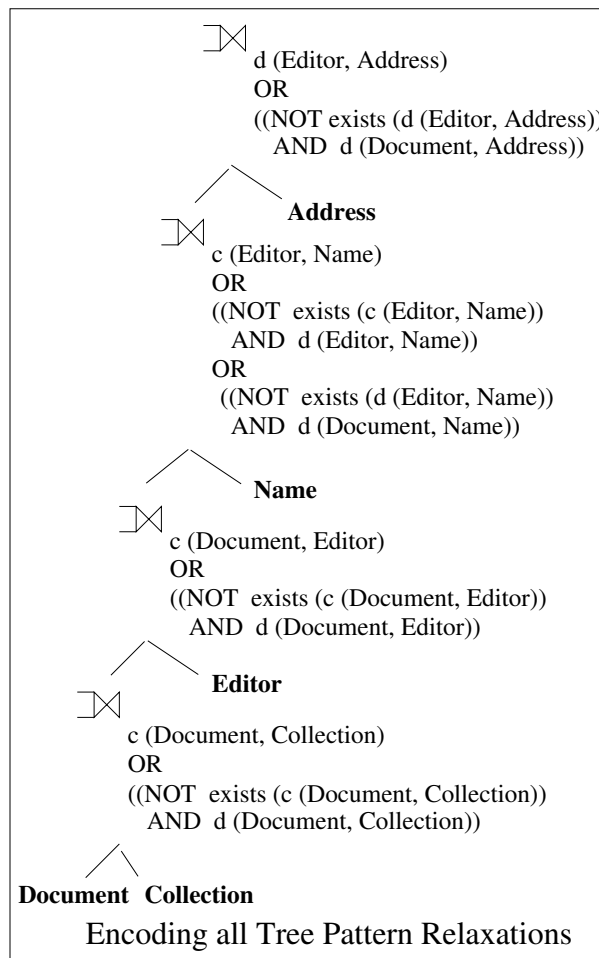
## Example: Encoding Edge Generalization



# Example: Encoding Subtree Promotion



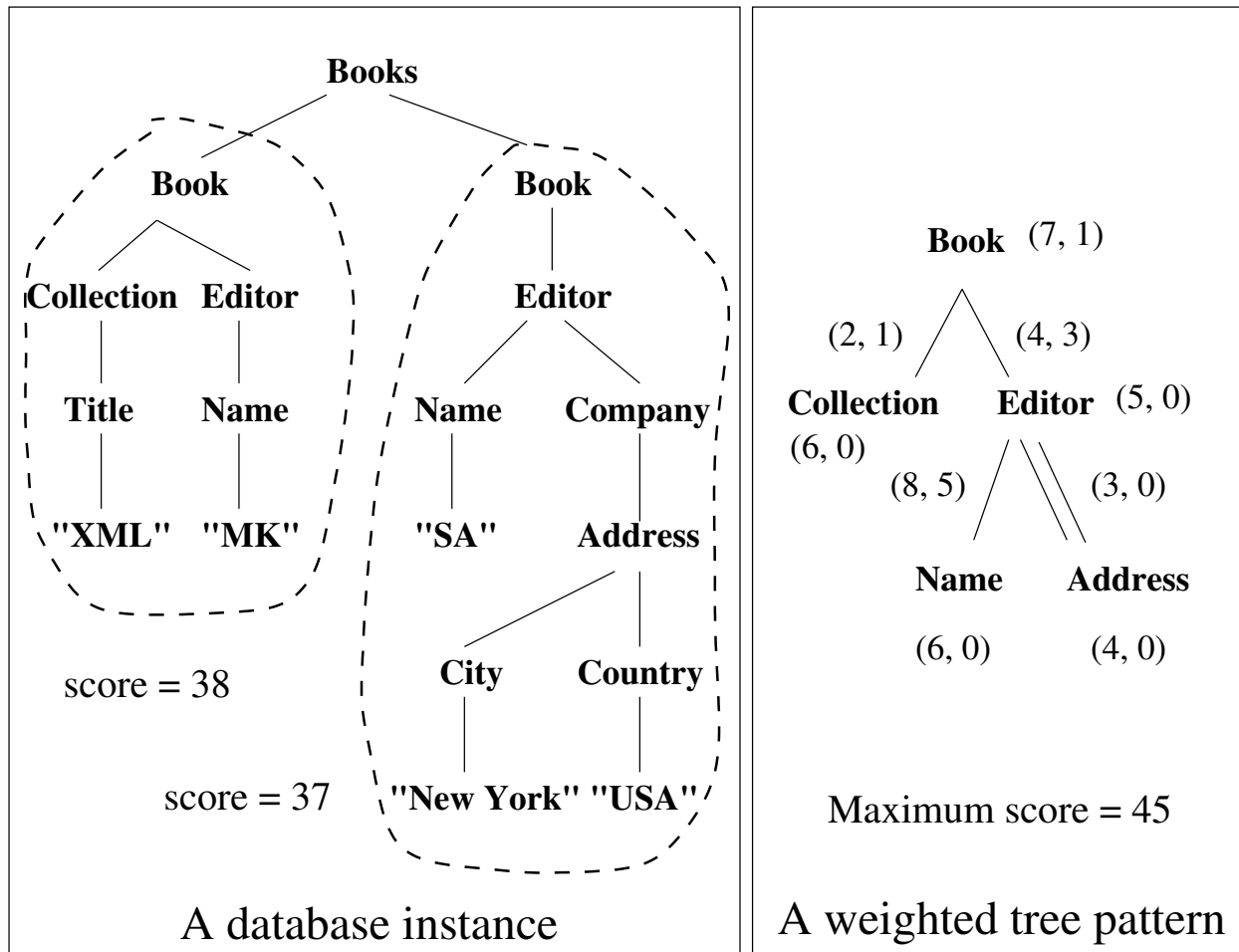
# Example: Encoding a Composition



# Assigning Scores to Approximate Matches

- Essential for ranking approximate matches
  - score = measure of “closeness” to original query
- Use weighted tree patterns
  - (exact, relaxed) weights with nodes and edges
  - flexible mechanism to represent different preferences
- Score of an approximate match
  - sum of exact and relaxed contributions
  - relaxed contribution =  $f(\text{relaxations encoded})$

# Example: Weighted Tree Pattern and Scores



# Threshold Problem

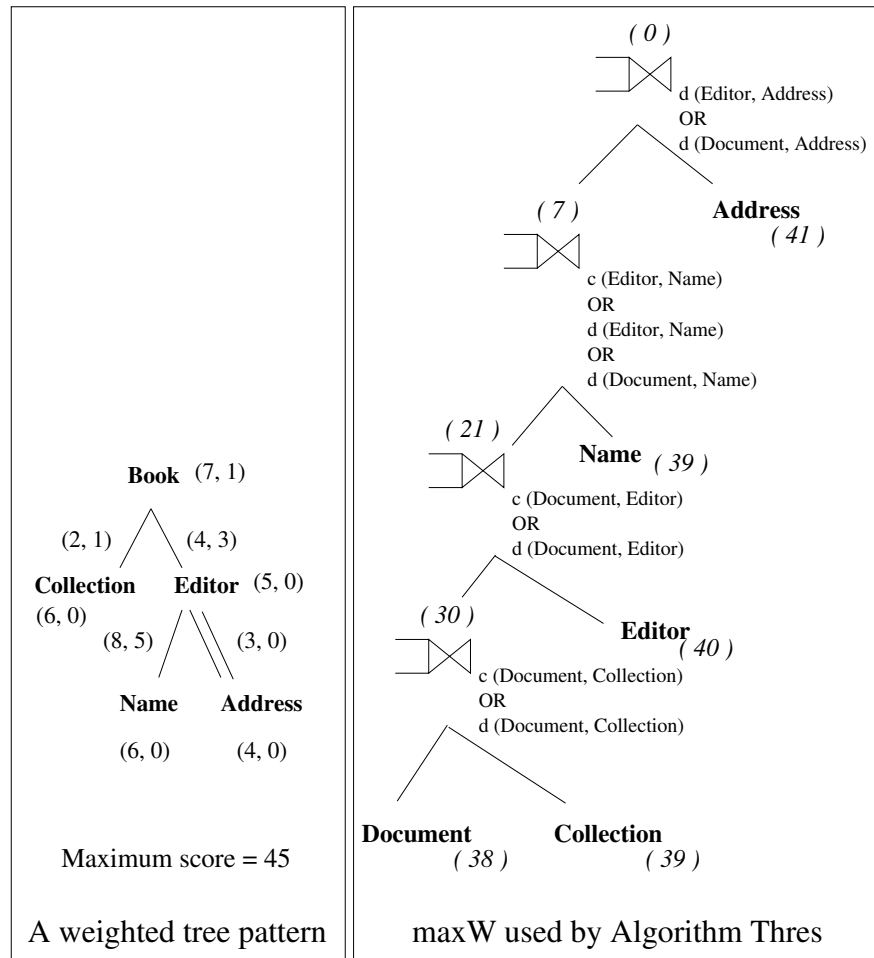
- Threshold problem: Given a weighted tree pattern  $Q$ , and a threshold  $t$ , find all approximate matches to  $Q$  whose score  $\geq t$ 
  - one can also formulate a Top-K problem
- Evaluation problem: Given a join evaluation plan that encodes all relaxations, efficiently compute all matches whose score  $\geq t$ 
  - Algorithm Thres: prune partial results early
  - Algorithm OptiThres: unencode irrelevant relaxations

# Algorithm Thres

```
Algorithm Thres(Node n)
  if (n is leaf) {
    list = evaluateLeaf(n);
    for (r in list)
      if ( $r \rightarrow \text{score} + n \rightarrow \text{maxW} \geq \text{threshold}$ ) append r to results;
    return results; }
  list1 = Thres(n  $\rightarrow$  left); list2 = Thres(n  $\rightarrow$  right);
  for (r1 in list1) {
    for (r2 in list2) {
      if (checkPredicate(r1,r2,n  $\rightarrow$  predicate))
        s = computeScore(r1,r2,n  $\rightarrow$  predicate);
      if ( $s + n \rightarrow \text{maxW} \geq \text{threshold}$ )
        append (r1,r2) to results with score s; }
    if ( $\nexists$  r2 that joins with r1)
      if ( $r1 \rightarrow \text{score} + n \rightarrow \text{maxW} \geq \text{threshold}$ )
        append (r1,-) to results with score r1  $\rightarrow$  score;
  }
  return results;
```

- maxW: how much can the score of a partial result increase?

# Example: Weights Used by Algorithm Thres

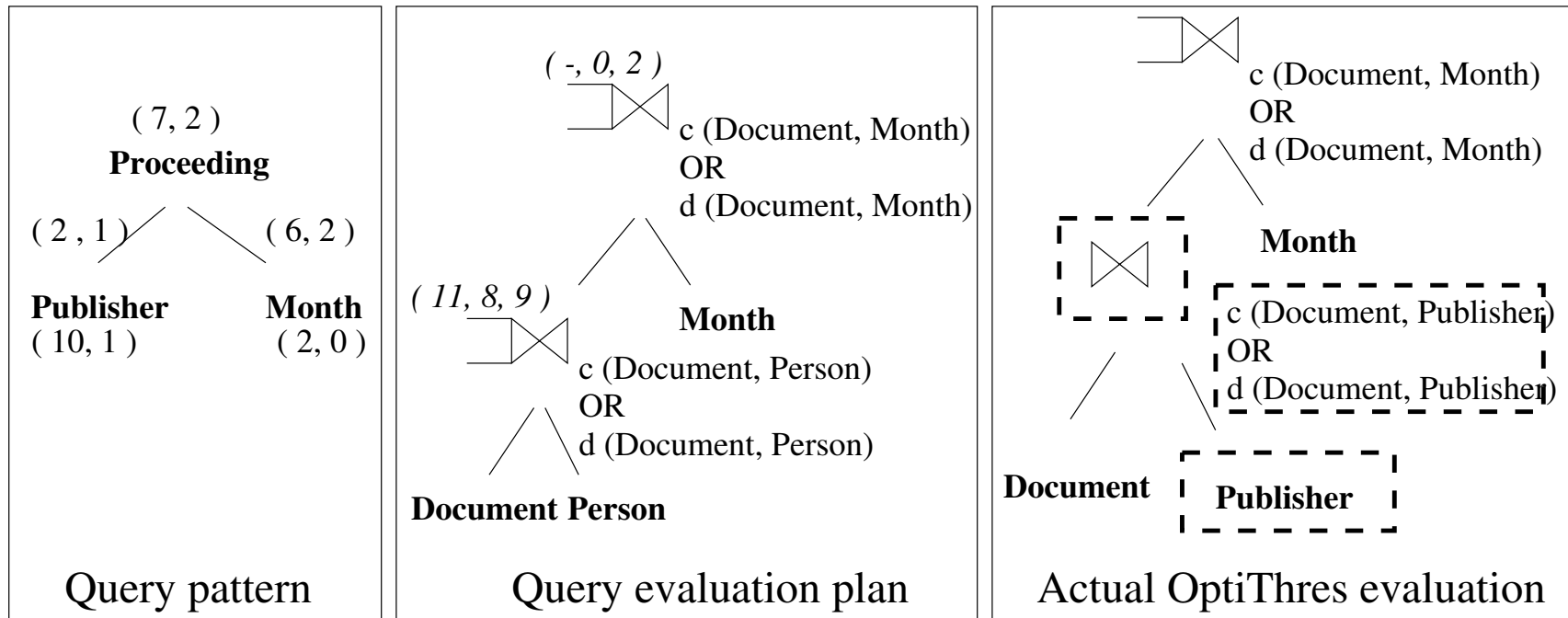


# Algorithm OptiThres

```
Algorithm OptiThres(Node n)
  if (n is leaf) {
    // evaluate, prune, and return results as in Algorithm Thres
  }
  list1 = OptiThres(n→left);
  /* maxLeft is set to the maximal score of results in list1 */
  if (maxLeft + relaxNode < threshold) unrelax(n→right);
  list2 = OptiThres(n→right);
  /* maxRight is set to the maximal score of results in list2 */
  if (maxLeft + relaxJoin < threshold) unrelax(n→join);
  if (maxLeft + maxRight + relaxPred < threshold)
    unrelax(n→join→predicate);
  // now, evaluate, prune and return join (and possibly outer join)
  // results as in Algorithm Thres
```

- relaxNode: should right child of join node remain relaxed?
- relaxJoin: should join remain an outerjoin?
- relaxPred: should join predicate remain relaxed?

# Example: Weights Used by Algorithm OptiThres



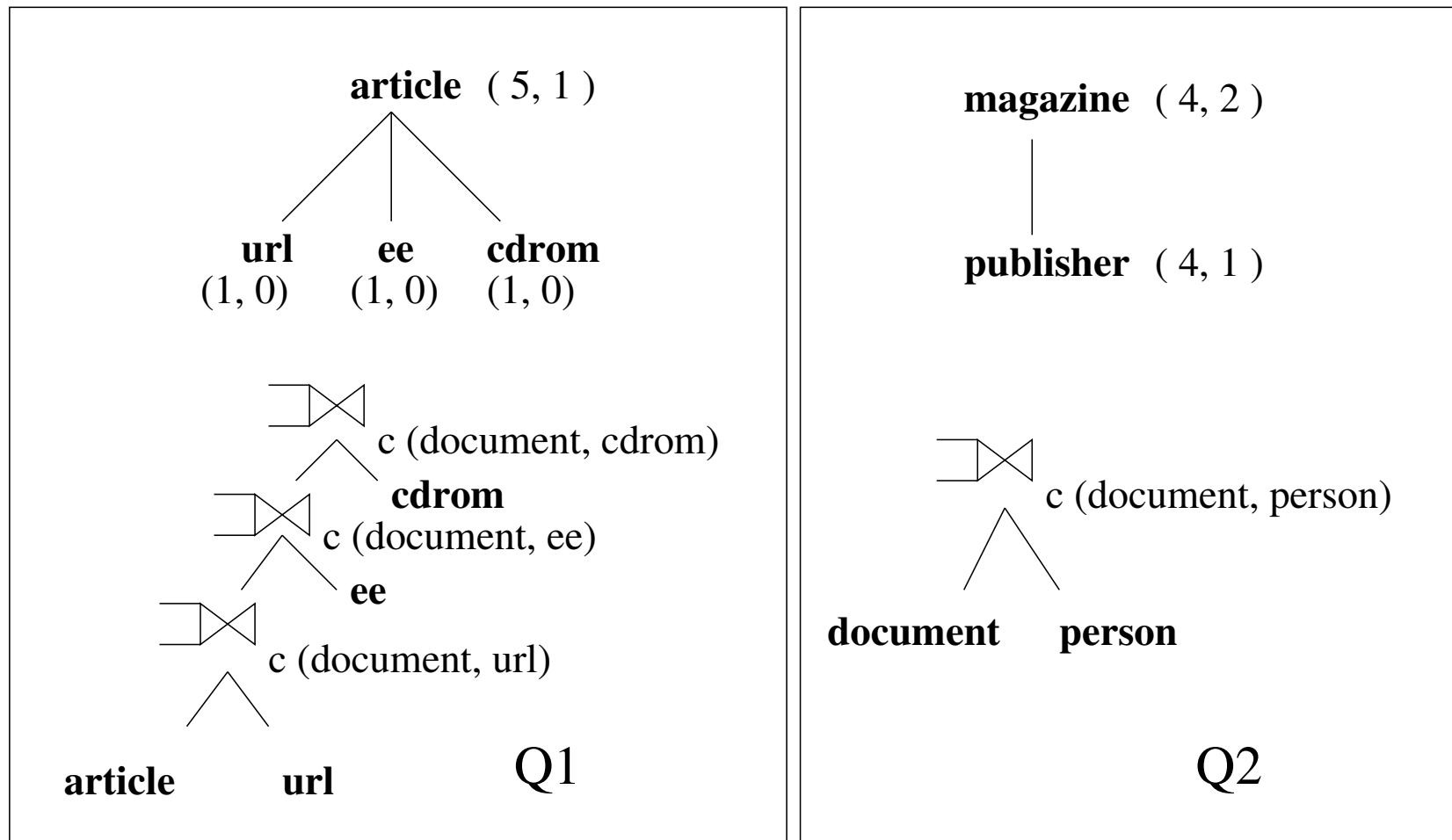
## Experimental Evaluation: Setup

- DBLP dataset with 2.1 million elements
  - DTD and dataset: <http://dblp.uni-trier.de/db>

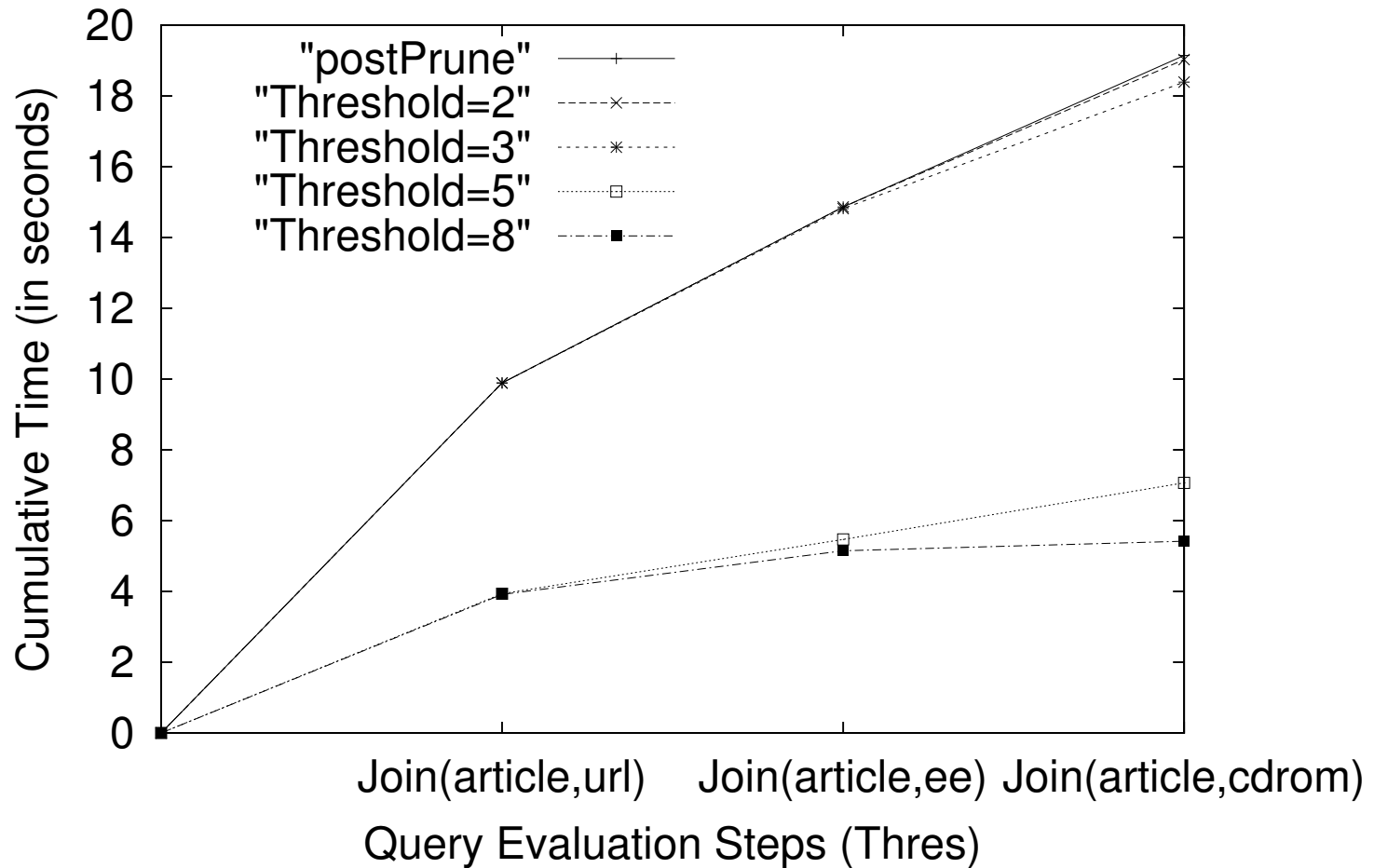
Label	No. of elements	Label	No. of elements
article	87,675	url	212,792
cdrom	13,052	ee	55,831
document	213,362	magazine	0
publisher	1,199	person	448,788

- dummy type **magazine** and supertypes **document**, **person**

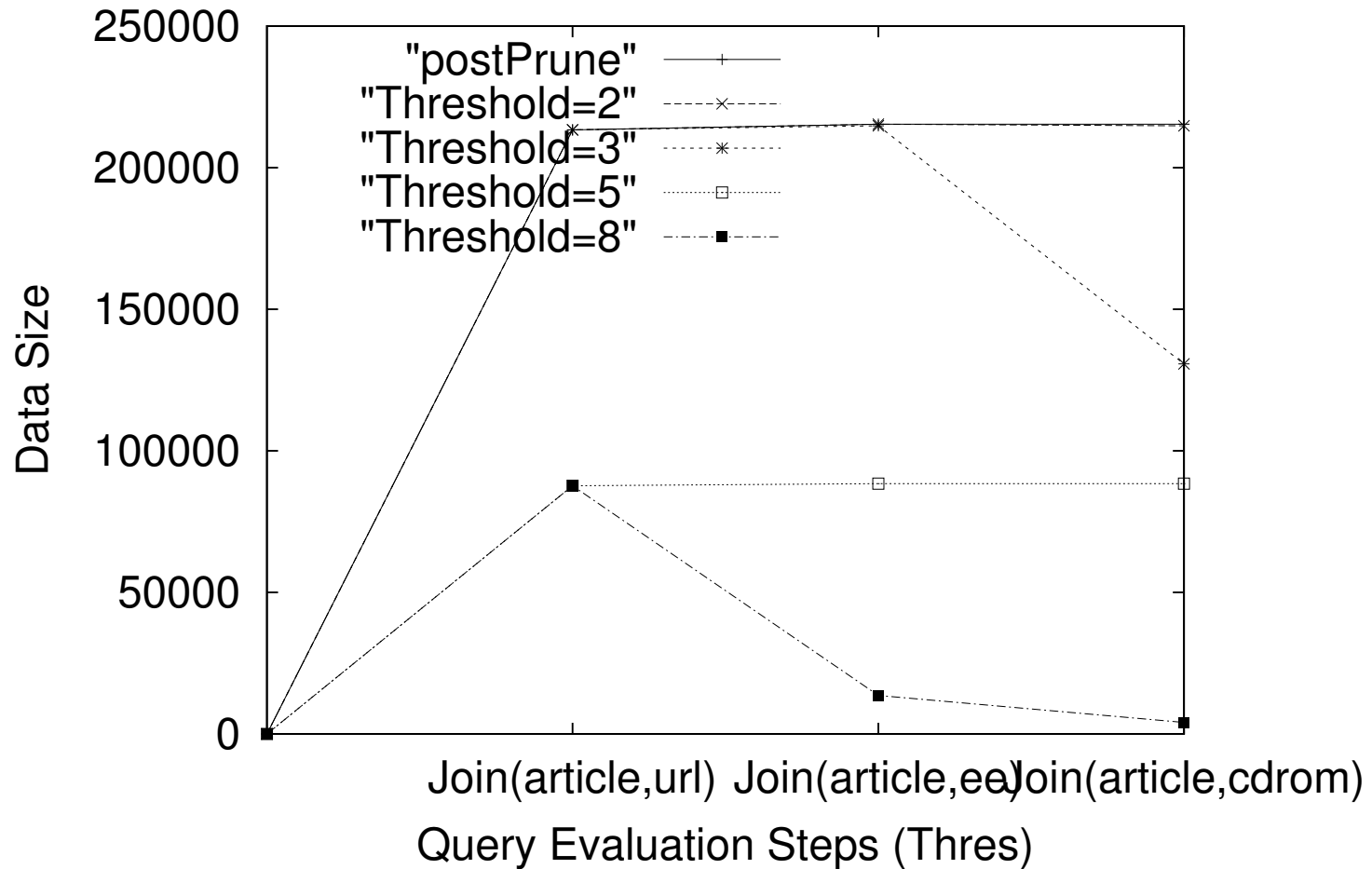
# Experimental Evaluation: Queries Used



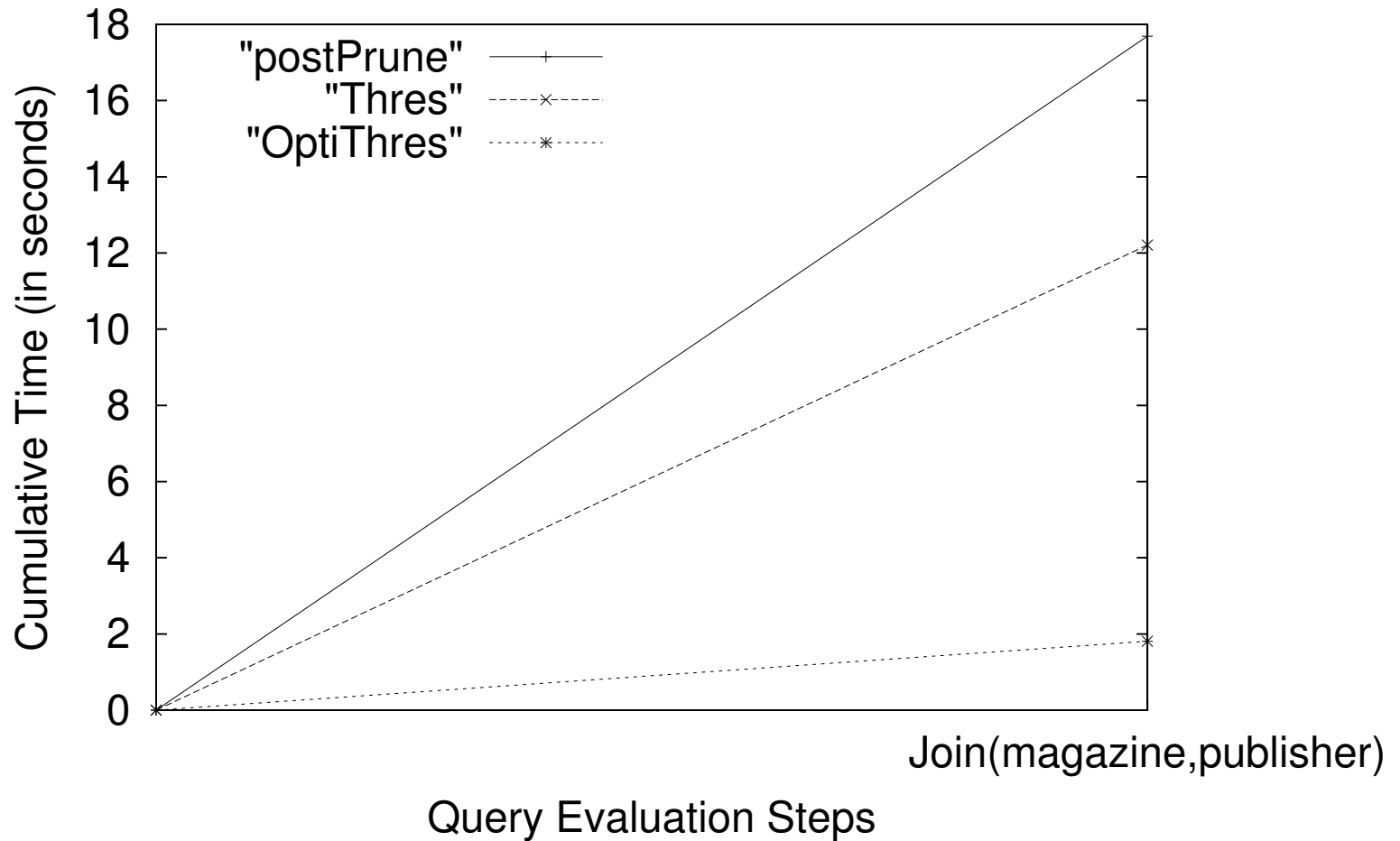
## Time Comparison: Thres vs postPrune



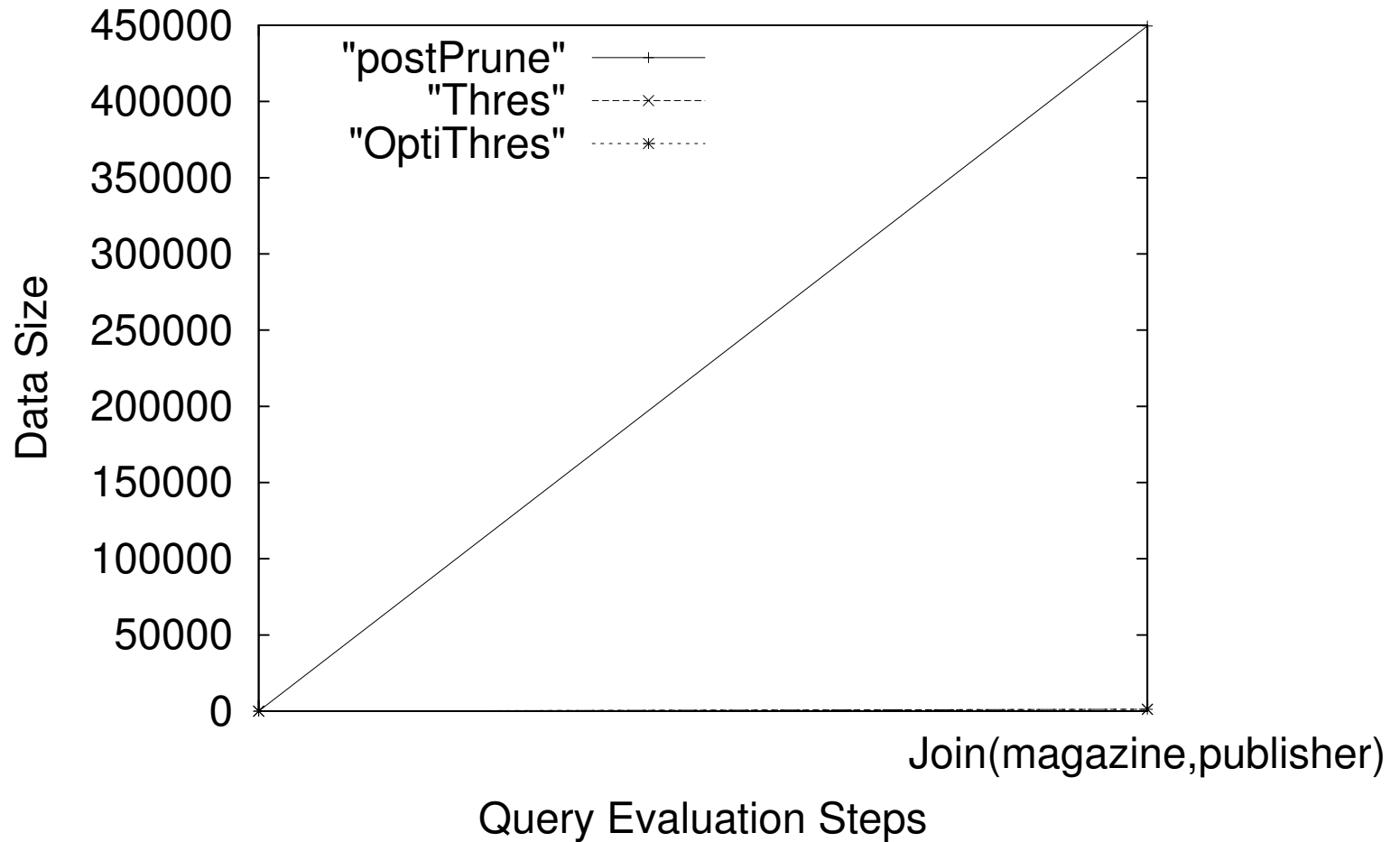
## Data Size Comparison: Thres vs postPrune



# Time Comparison: OptiThres, Thres, postPrune



# Data Size Comparison: OptiThres, Thres, postPrune



## Comparison with Rewriting-Based Approaches

Evaluation Method	Evaluation Time	Number of Joins
OptiThres	18.550s	3
postPrune	22.384s	3
MultiQ	40.842s	10
MultiQOptim	30.782s	8

- Query Q1, with threshold set to 2
  - to select a large number of answers

## Related Work

- Language proposals for approximate matching
  - XXL [TW00]: extends XML-QL for ranked retrieval
  - approXQL [Sch01]: extends XQL
- Specification and evaluation
  - relaxations in a mediator [DR01]: no evaluation
  - rewriting based on tree-edit distance [Sch01]: no weighting
  - flexible/semi-flexible [KS01]: no scoring, ranking

# Conclusions

- Novel tree pattern relaxation techniques
  - encoding relaxations in join evaluation plans
  - substantial benefits over post-pruning and rewriting
- Open problems
  - analog of  $tf * idf$  used in Information Retrieval
  - integration with cost-based join ordering
  - estimation of number of answers for a given threshold