Data Mining Research Laboratory

THE OHIO STATE UNIVERSITY

LCS-TRIM

Dynamic Programming Meets XML Indexing and Querying

Shirish Tatikonda
Srinivasan Parthasarathy
Matthew Goyder

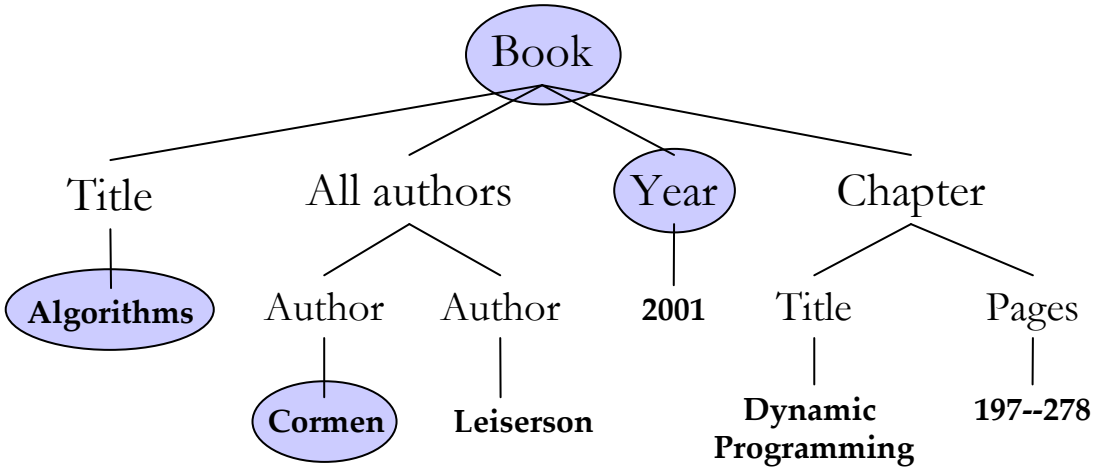
VLDB
September, 2007

Outline

- Problem definition
- Our approach
  - Data transformation
  - Tree matching
  - Tree Indexing
- Optimizations
- Evaluation
- Related work
- Conclusions

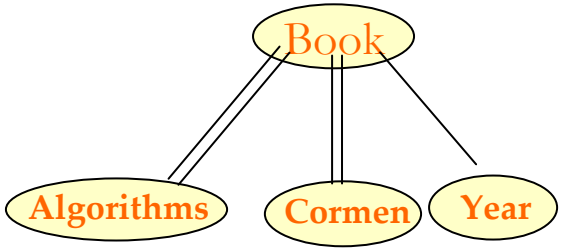
Problem definition

XML Document



XPath Query (twig)

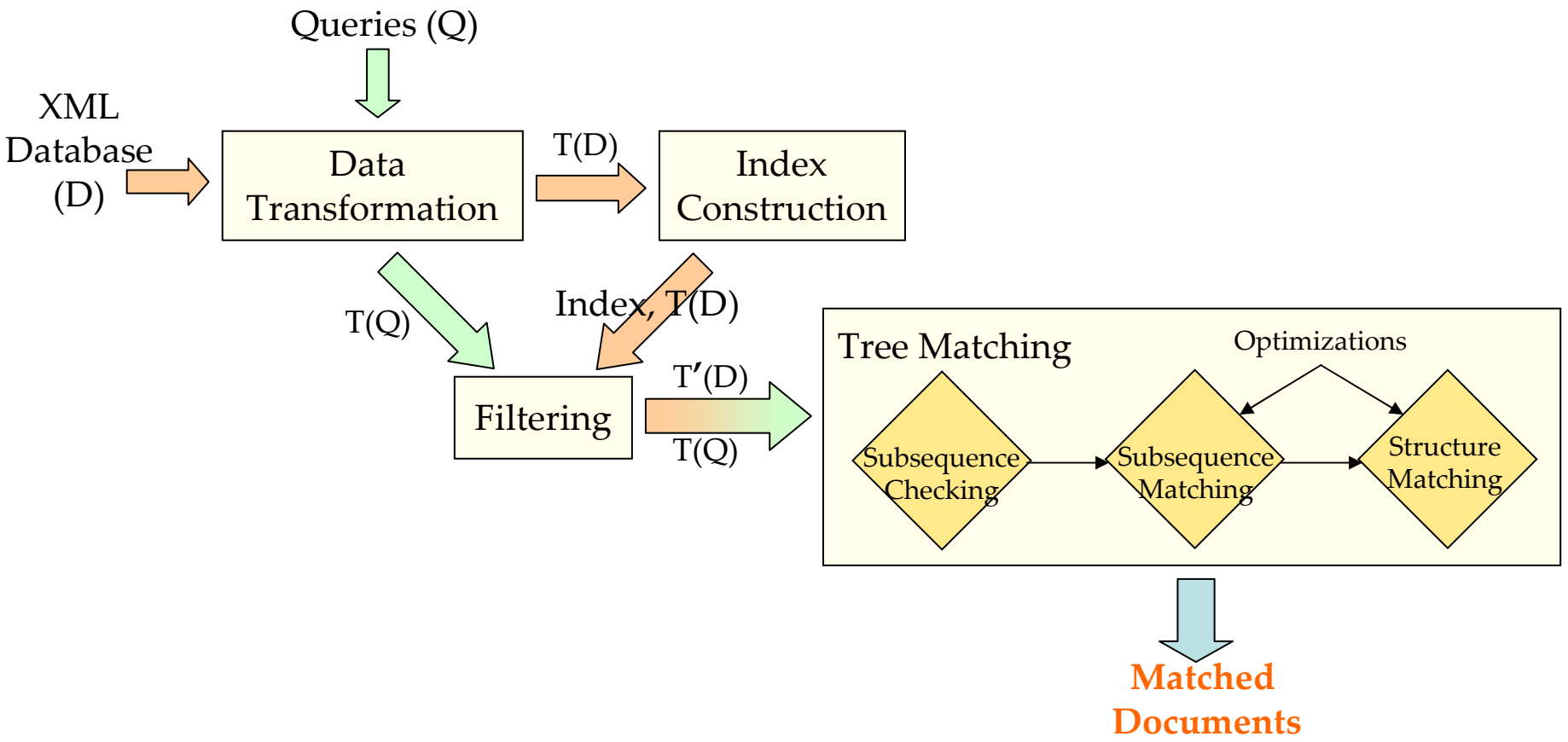
//Book [//Algorithms] [//Cormen] [Year]



Enumerate all the embedded subtrees of the given query tree

Rooted, labeled, ordered isomorphisms

LCS-TRIM: LCS-based TRee Indexing & Matching

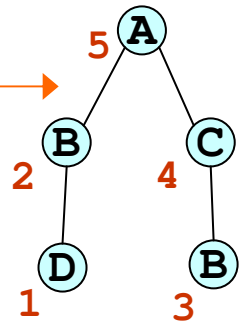


Data transformation

- Transforms trees in sequences
- Prüfer & Depth-first sequences

Tree transformations

Rooted, ordered, labeled tree



Numbered Prüfer Sequence

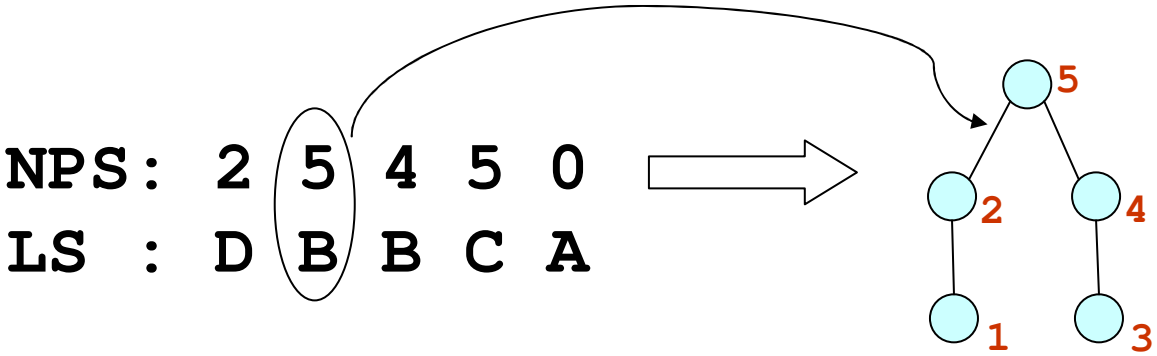
NPS: 2 5 4 5 0

LS : D B B C A

Label Sequence

1. Traverse the tree in *post-order (PO)*
2. Remove the node with smallest *PO* number
   - Add its label to LS and its parent to NPS

Data transformation



Properties

- Non-redundant complementing sequences
  - NPS provides structure
  - LS provides labels
- Concise bijective transformation
- Each pair corresponds to an edge
- No pointers => improved ILP

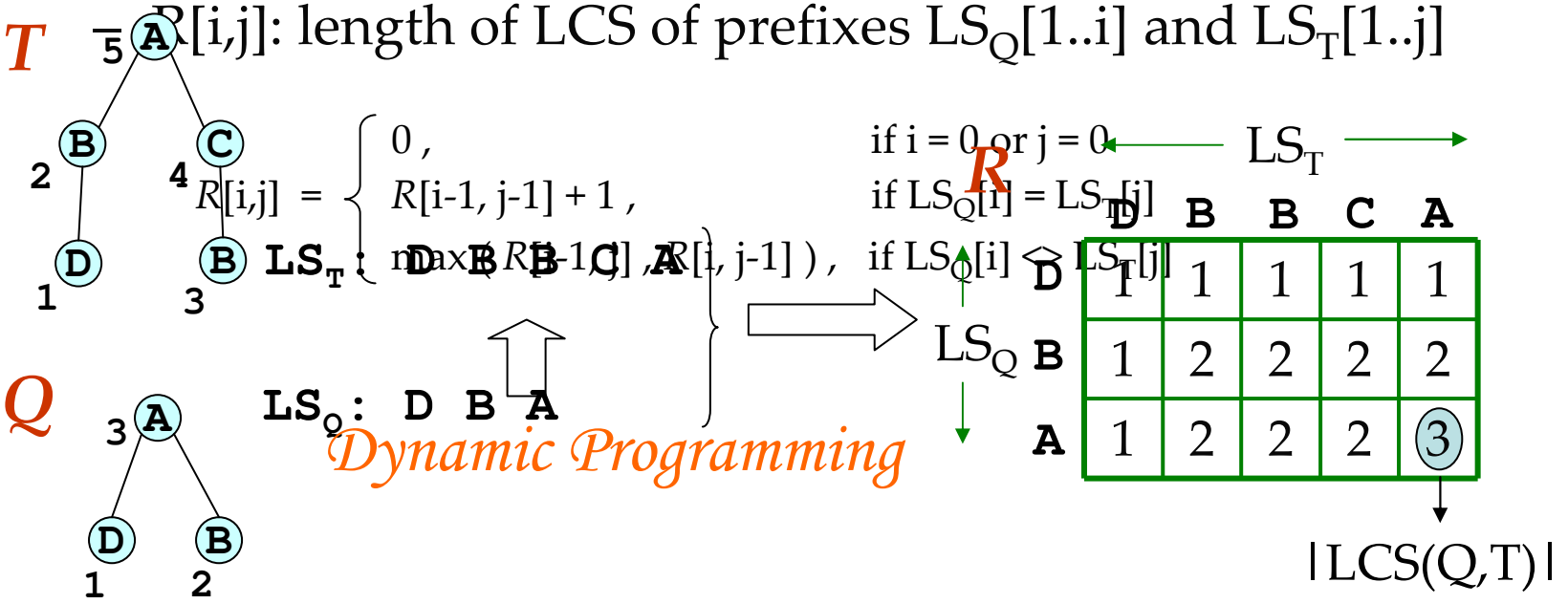
Tree matching

- Finds the matches of a query Q in a database tree T
- Recasted in to subsequence matching
  - If Q is a subtree of T then LS_Q is a subsequence of LS_T
- Consists of 3 steps
  - Subsequence checking
    - Check if LS_Q is the LCS of LS_Q and LS_T
  - Subsequence matching
    - Enumerate all the subsequence matches of LS_Q in LS_T
  - Structure matching
    - Filter the subsequence matches by using NPS_Q and NPS_T

Subsequence checking

An example

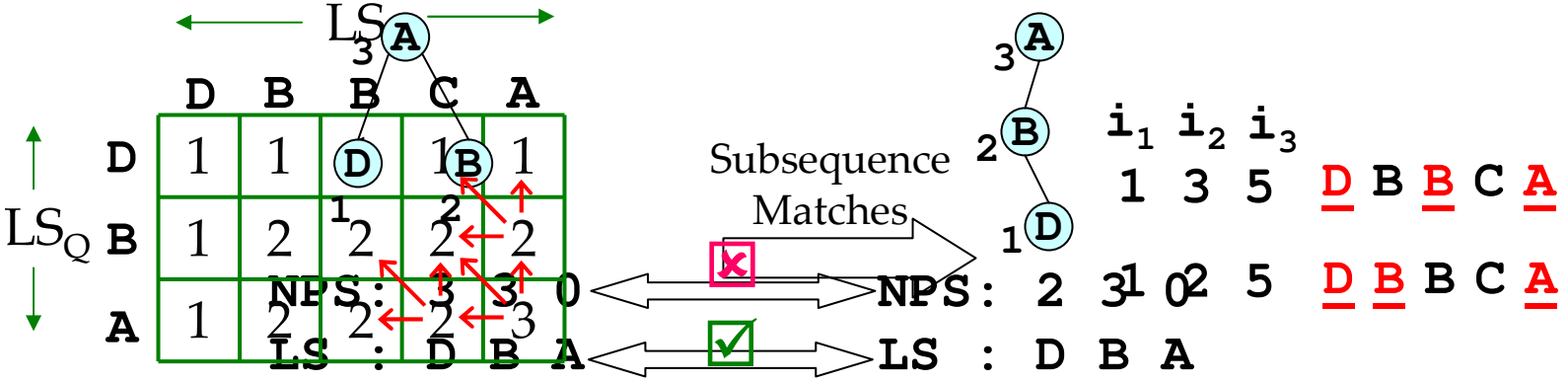
- Checks if LS_Q is a subsequence of LS_T or not
  - LS_Q has to be the Longest Common Subsequence of LS_Q, LS_T
- Construct the *R* matrix
  - R[i,j]: length of LCS of prefixes $LS_Q[1..i]$ and $LS_T[1..j]$



If |LCS(Q,T)| ≠ |Q| then Q is not a subtree of T

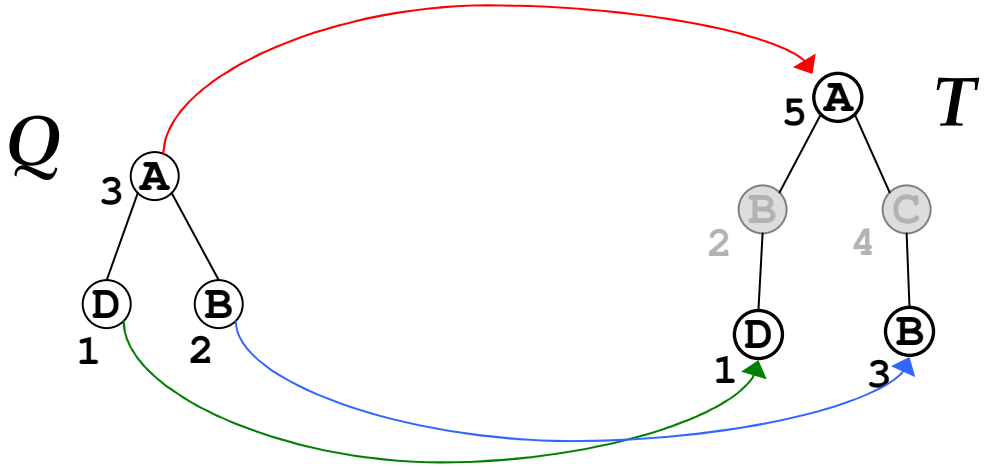
Subsequence matching

- Enumerates all the occurrences of LS_Q in LS_T
  - By *backtracking* on R from $R[m,n]$: $m=|Q|$ & $n=|T|$
- Resulting match: $(i_1, i_2, \ldots, i_m)$, $LS_Q[j] = LS_T[i_j]$



- *Matches are checked (LS) from right to left*
- *Matches are checked (LS) from right to left to avoid false positives*
- *Prune them by considering the structure (NPS)*

Structure matching



Subsequence match: $(i_1, i_2, i_3) = (1, 3, 5)$

- Map the structure of Q to the structure formed by subsequence match $(i_1, \ldots, i_m)$
  - Map each *edge* in Q to an ancestor-descendant in T
  - *Structure agreement checks* at each node *in Q*

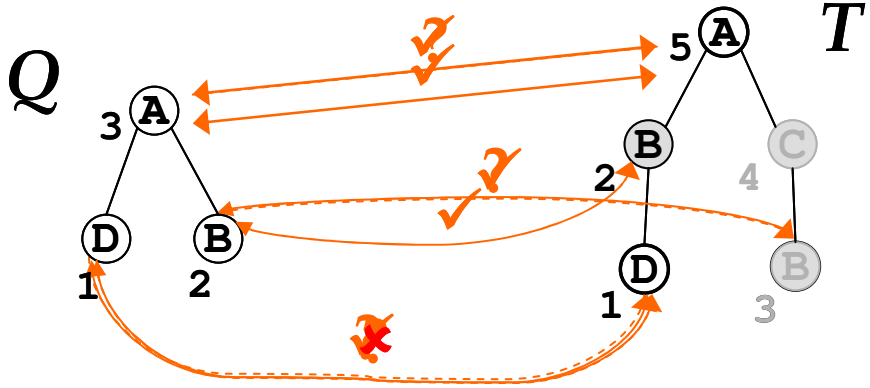
Structure agreement check

⇨ *At k=m, map the root node to i_k without any checks*

⇨ *At k=m-1 … k=1 (i.e., right to left)*

→ *check if k^{th} parent in Q = nearest mapped ancestor of i_k in T*

Subsequence match: $(i_1, i_2, i_3) = (1, 2, 5)$ *False positive*

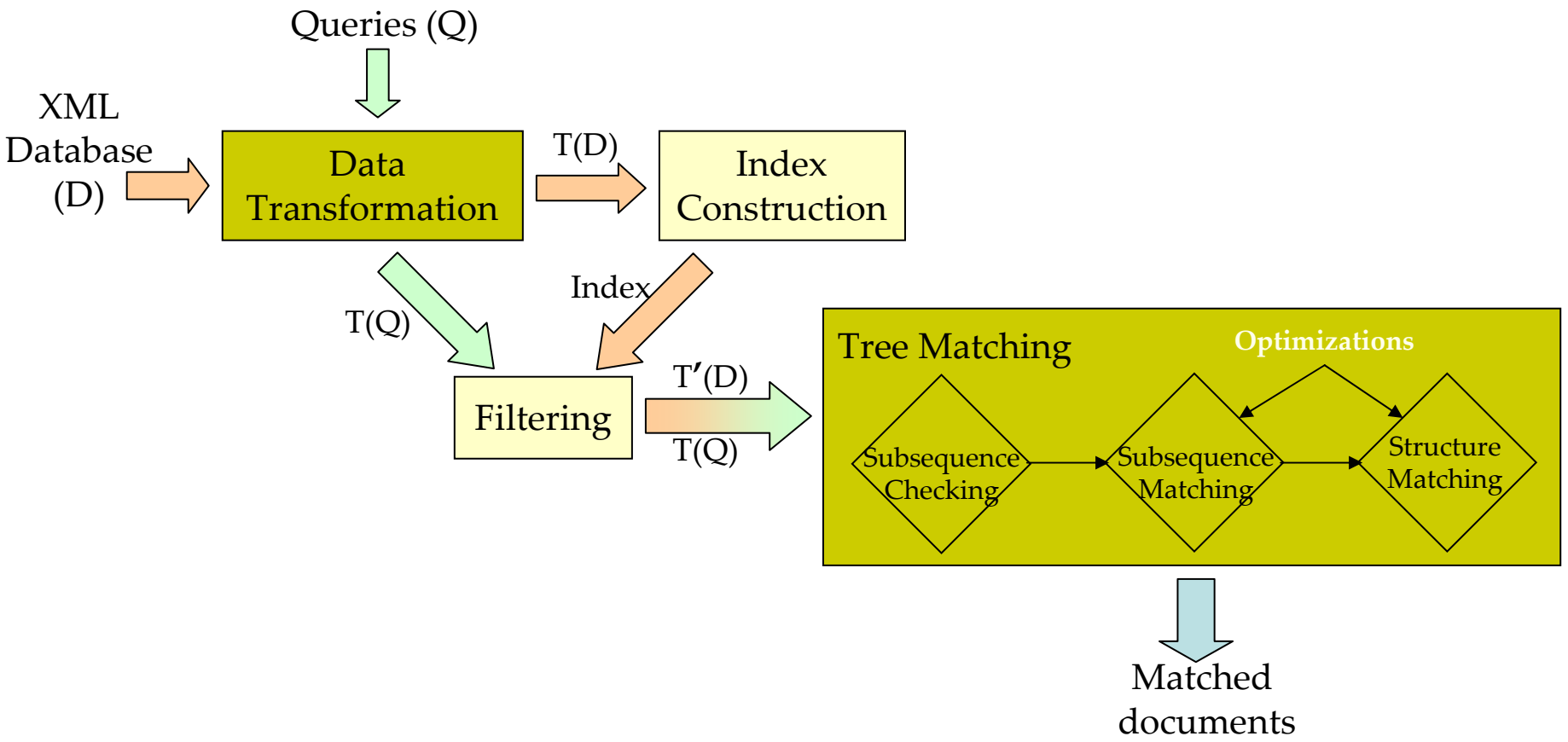


map

1	2	3
1	2	5

- k=3: map the root directly to i_3
- k=2: check if *map*[2's parent] is an ancestor of i_2=3 in T
- k=1: check if *map*[1's parent] is an ancestor of i_2=1 in T

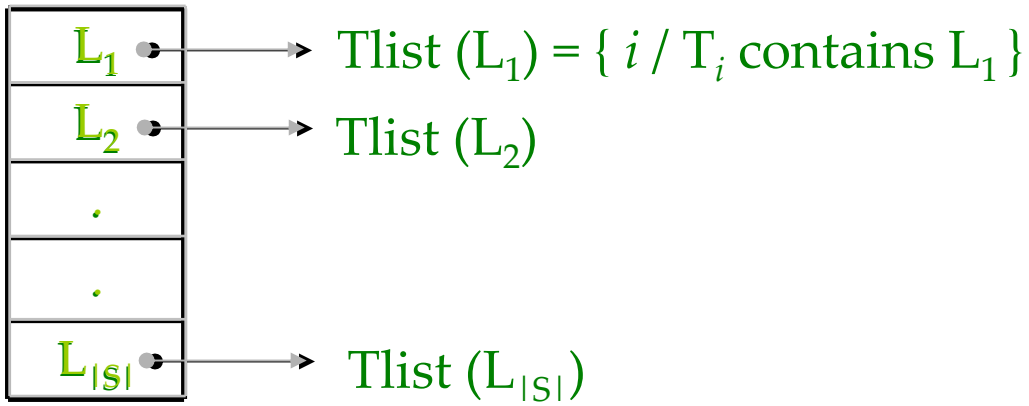
LCS-TRIM


Queries (Q)
XML Database (D)
Data Transformation
T(D)
Index Construction
T(Q)
Index
Filtering
T′(D)
T(Q)
Tree Matching
Optimizations
Subsequence Checking
Subsequence Matching
Structure Matching
Matched documents

Index

- Selects a subset of trees from the data base
  - *Tree matching* is done only on this subset
- Simple inverted index

S = set of distinct labels in data base



- Given a Q with labels $\{L_{q1}, \ldots, L_{qm}\}$
- Tree matching is performed on the following set

$$\text{Tlist}(L_{q1}) \cap \ldots \cap \text{Tlist}(L_{qm})$$

Outline

- Problem definition
- Our approach
  - Data transformation
  - Tree matching
  - Tree Indexing
- Optimizations
- Evaluation
- Related work
- Conclusions

Optimizations for tree matching

- Alleviate the computation overhead
- Preserve the correctness
- Reduce the number of recursions
  - *Label filtering* (LF)
  - *Dominant match processing (DOM)*
- Reduce the number of false positives
  - *Simultaneous subsequence and structure matching (SIMUL)*

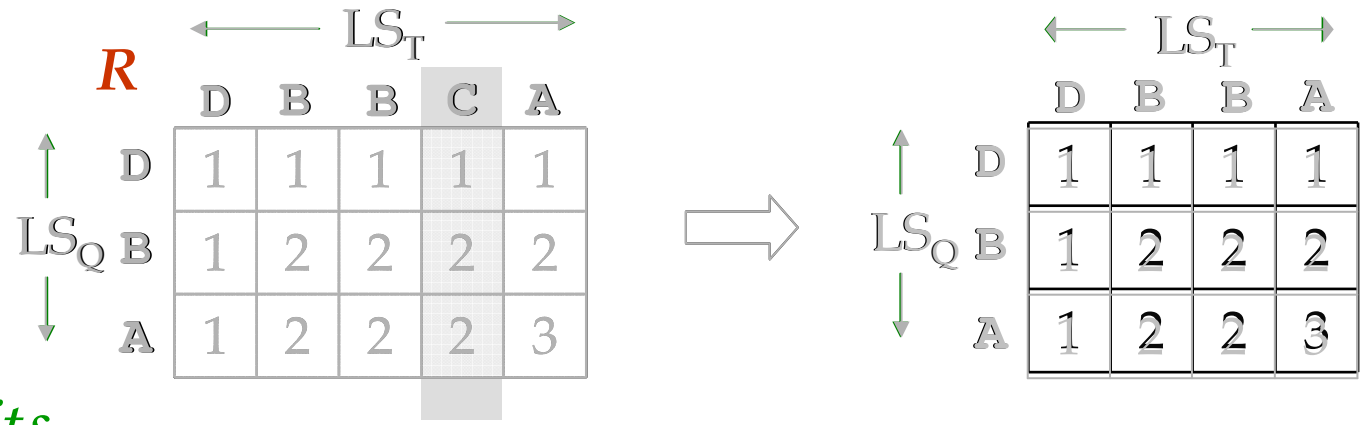
Label filtering (LF)

Observation

Not all the labels in data base tree appear in the query, Q

Strategy

Prune LS_T from the labels *not in* Q



Benefits

- ✓ Shrinks *R* making it fit in a few cache lines
- ✓ Reduces the number of recursions on *R*

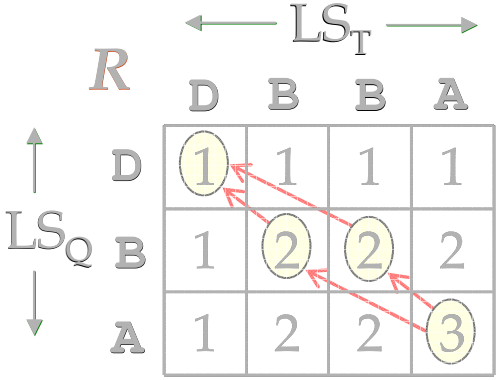
Dominant match processing (DOM)

Observation

Only few entries of *R matrix* are interesting

Strategy

Identify the entries at which subseq. length is increased



Benefits

✓ Completely *eliminates* the redundant recursions

Simultaneous subsequence and structure matching

Observation

Both matching steps operate from *right to left*

Strategy

As soon as the labels match at k^{th} position,
perform the structure agreement check at k

Benefits

- ✓ Detects the false positives as early as possible
- ✓ *Never* generates them completely
- ✓ Significantly improves the performance

Outline

- Problem definition
- Our approach
  - Data transformation
  - Tree matching
  - Tree Indexing
- Optimizations
- Evaluation
- Related work
- Conclusions

Evaluation

- Used a AMD 250 Opteron dual processor system with 8GB of memory
- Performance results against PRIX [Rao06]
- Evaluated on different data sets and workloads

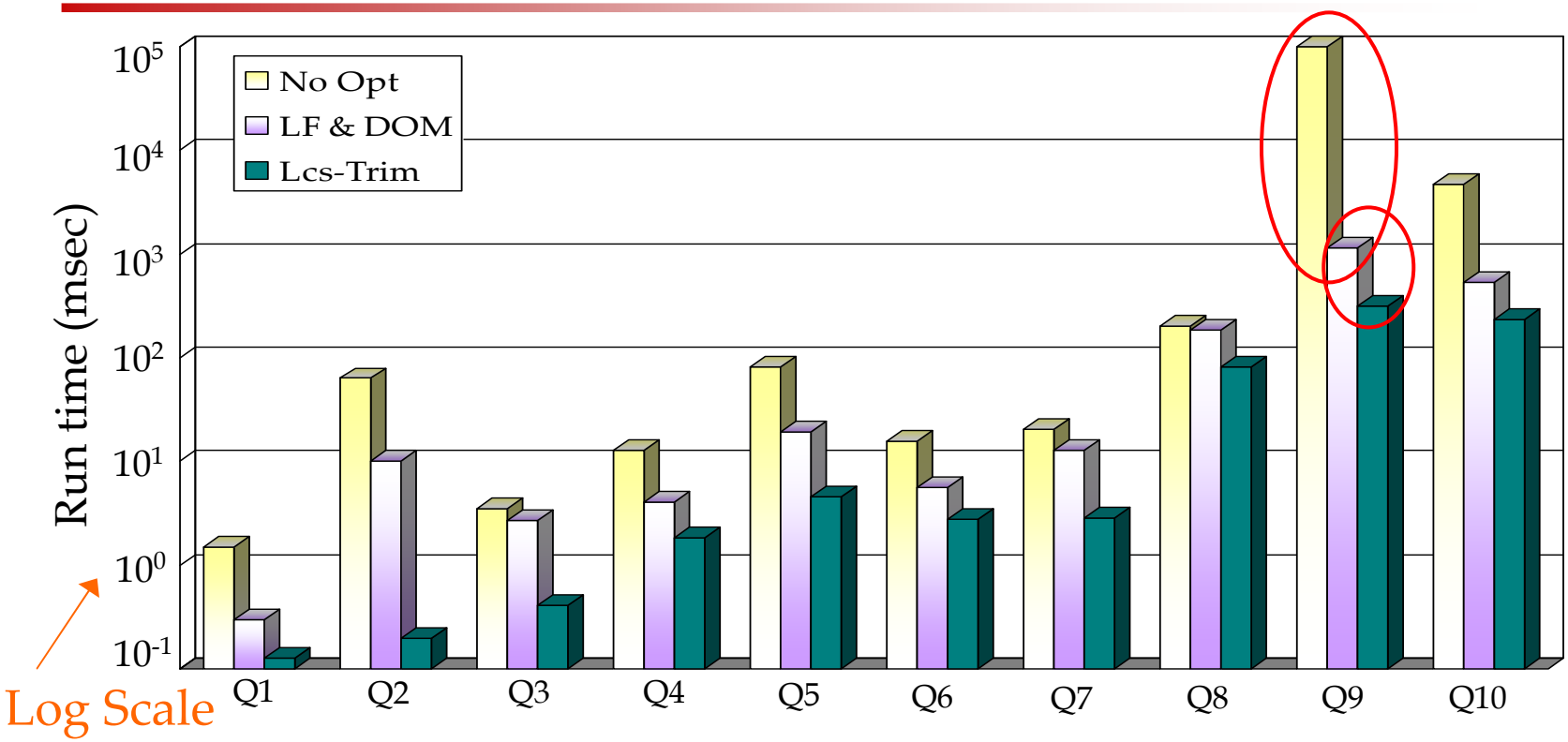
Dataset	# of trees	Max depth
Swissprot	50,000	5
Treebank	52,851	36
DBLP	328,858	6
Cslogs	59,691	85
NLM	450K-1M	8

[Rao06] Sequencing XML Data and Query Twigs for Fast Pattern Matching, TODS, 2006

Comparison with PRIX

PRIX	LCS-TRIM
Data transformation	
– Prufer-based but Longer Sequences – Suffer from redundancy	– Space-efficient representation – Absolutely *no redundancy*
Subsequence matching	
– Relies on large disk-based indices – Too many probes to the index	– Efficient *dynamic programming* – Operates on simple matrices
Structure matching	
– Multiple stages of refinements – Each makes a pass over the twig	– *Single* pass over the twig
– Parameter dependent – Poor locality	– Parameter free – Very good locality

Effect of optimizations - swissprot



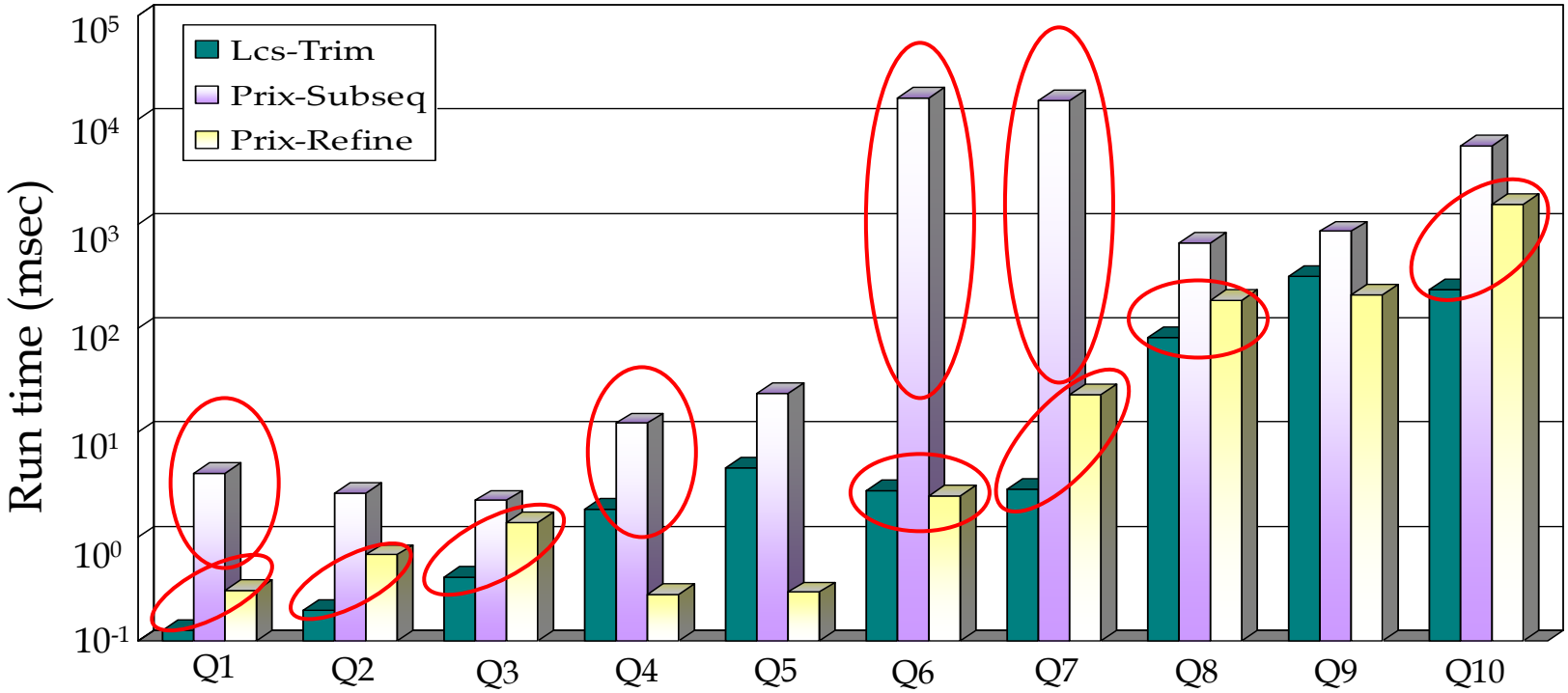
For Q9

⇨ *Reduce the computational overhead*

- # of recursive calls reduced from *16 billion* to *5 million*
- The size of R is reduced by *5 times*
- Processing of *~3 million* false positives is avoided

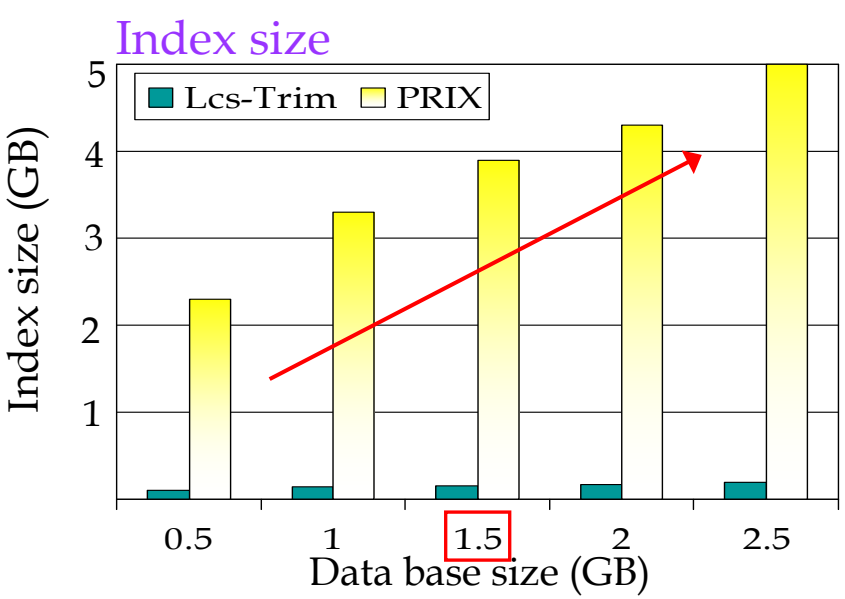
⇨ *Demonstrate more than two orders of improvement*

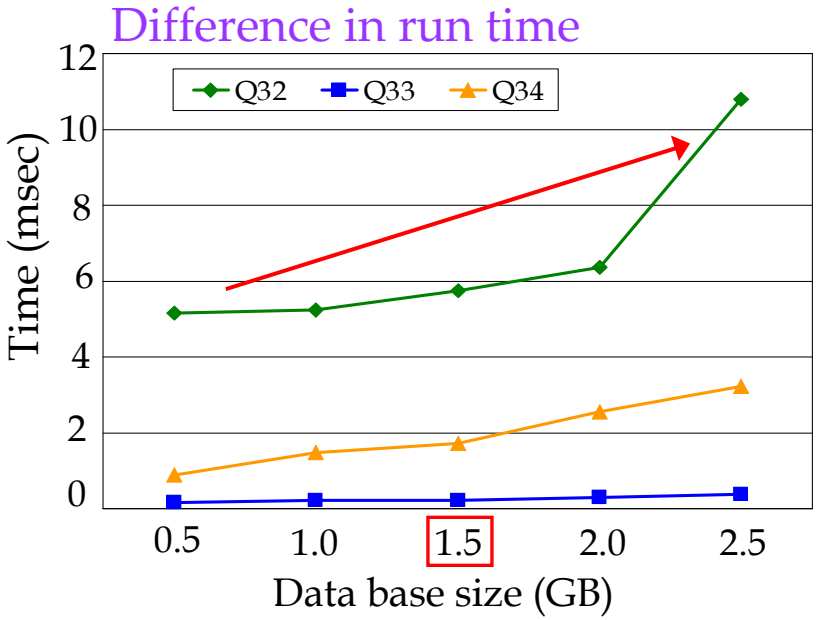
Comparison with PRIX - swissprot



- ⇨ *Subsequence matching in PRIX is way too expensive*
- ⇨ *LCS-TRIM is comparable or faster than refinements in PRIX*
- ⇨ *Overall, up to 2-3 orders of magnitude faster than PRIX*

Effect of data base size - NLM





- PRIX
  - Index size increases proportionately with data base size
  - Workload dependent
- Our index is small & tunable
- Difference in run time increases with data base size

Effect of workload characteristics

Average improvement in run time

- Query size
  - Small: 60-times
  - Large: 400-times

  } Large # of probes to the index in PRIX

- Node selectivity
  - High: 170-times
  - Low: 250-times

  } LCS-TRIM is not affected by selectivity of nodes

- Recursive structure
  - Deep: 240-times
  - Bushy: 190-times

  } Large # of index probes for deep queries in PRIX

- Wildcards
  - Up to 2,500 times

  } Large # of false positives and PRIX detects them very late

Related work

- Path-based
  - Structural joins [Al-Khalifa02]
  - DataGuides[Goldman97], APEX[Chung02], and others
- TwigStack-based
  - Holistic joins [Bruno02]
  - B^+-tree [Chien02], XR-TwigStk [Jiang03], and others
- Sequence-based
  - ViST [Wang03]
  - PRIX [Rao06]
  - Zezula's [Zezula03]

Current & future work

- Comparison with TwigStack–based algorithms
- Extensions
  - Unordered matching
  - Approximate matching
  - Handling NOT predicates
  - Parallel matching
    - We have seen up to 6.3-fold speedup on 8 processors
    - Suitable for emerging multi-core server architectures

Conclusions

- Non-redundant data transformation
- Dynamic programming -based tree matching
  - Pruning steps are embedded
- A very simple index structure
- Up to 2-3 orders of magnitude faster

Acknowledgements

- NSF grants
  - CCF-0702587, CNS-0406386, IIS-0347662, and CNS-0403342
- All the reviewers
- Dr. Praveen Rao (PRIX) & Dr. Haifeng Jiang (XR-TwigStk)

Our lab DMRL: http://dmrl.cse.ohio-state.edu/index.jsp
My page : http://www.cse.ohio-state.edu/~tatikond