【VLDB2010勉強会】

Session 28: XML Data

担当：駒水孝裕（筑波大学）

XPath Whole Query Optimization

/ S. Maneth (NICTA & UNSW) and K. Nguyen (NICTA)

- 先行研究[D. Arroyuelo et al. ICDE2010] で利用されている技術についての論文 （SXSL: XPath エンジン）
- [背景] XPath が様々な状況で利用されている
  - e.g. XQueryやXPath, XACML, JavaScriptエンジンなど
  - 効率的なXPath 評価が重要
  - 非決定性ツリーオートマトンを利用する
  - Koch et al. [6] が$O(|D|\cdot|Q|)$で評価できることを示した.
    （$|D|$: ドキュメントサイズ, $|Q|$: クエリサイズ）
- [アプローチ] $|Q|$-Optimization と $|D|$-Optimization
  - インメモリで処理➔ state-of-the-art succinct tree [18]
  - 文書中で辿るノード数を減らす➔ **関連ノード（*relevant node*）**
    - **インデックスを用い関連ノードのみに“jump”する.**

アプローチ

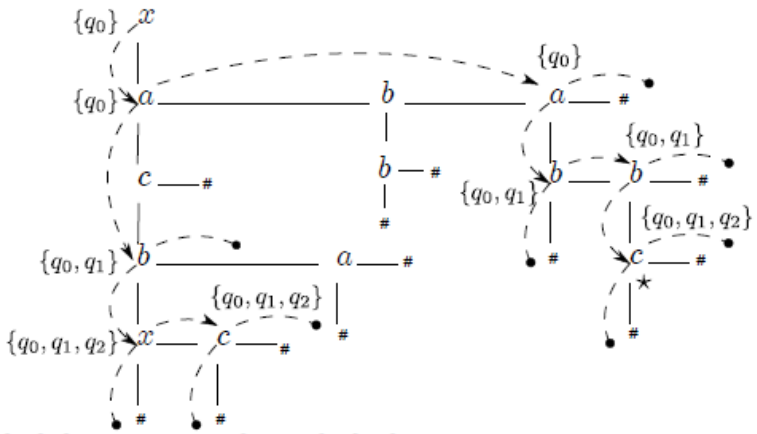
- 関連ノード：オートマトンのステートを変えるノード
- 非決定性オートマトンでは不要な遷移が存在する
  - どのように必要な遷移だけにするか➔オートマトンの最小化
    オートマトンの最小化は “EXPTIME-complete”
  - Top-down Deterministic Evaluation
    - Top-down Relevant nodes
    - Top-down Jumping Functions/Algorithm
  - Bottom-up Deterministic Evaluation
    - Bottom-up Relevant nodes

➔Start anywhere (top-down + bottom-up)

e.g. //a//b[c] のとき,

bからtop-down ([c])と bottom-up (//a) のように用いる

アプローチ（続）



$\{q_0\}, \{a\} \rightarrow \{q_0, q_1\}, \{q_0\}$
$\{q_0\}, \Sigma \setminus \{a\} \rightarrow \{q_0\}, \{q_0\}$
$\{q_0, q_1\}, \{b\} \rightarrow \{q_0, q_1, q_2\}, \{q_0, q_1\}$
$\{q_0, q_1\}, \Sigma \setminus \{b\} \rightarrow \{q_0, q_1\}, \{q_0, q_1\}$
$\{q_0, q_1, q_2\}, \{b\} \rightarrow \{q_0, q_1, q_2\}, \{q_0, q_1, q_2\}$
$\{q_0, q_1, q_2\}, \{c\} \rightarrow \{q_0, q_1\}, \{q_0, q_1\}$
$\{q_0, q_1, q_2\}, \Sigma \setminus \{b\} \rightarrow \{q_0, q_1\}, \{q_0, q_1, q_2\}$

Figure 1: Top-down approximation for `//a//b[c]` and corresponding jumps

実験結果

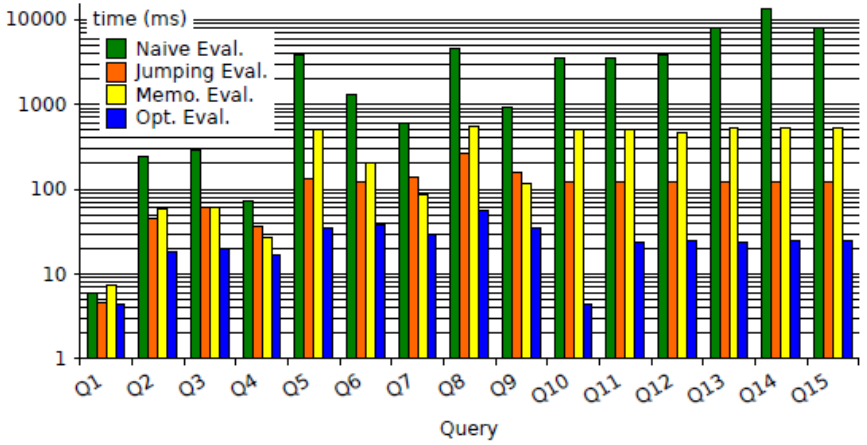


Figure 4: Impact of the jumping and memoization on query evaluation time

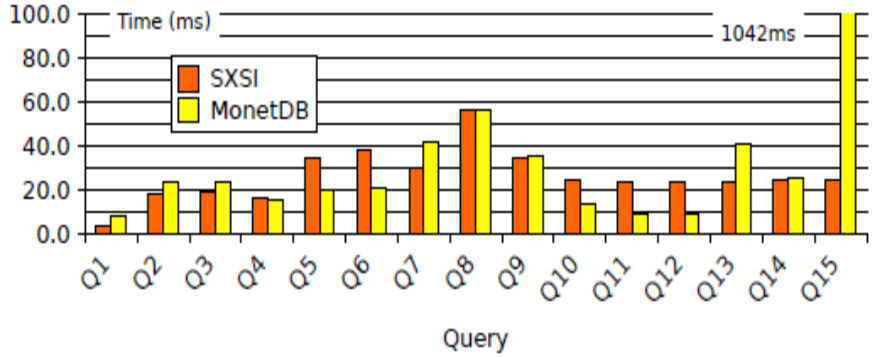
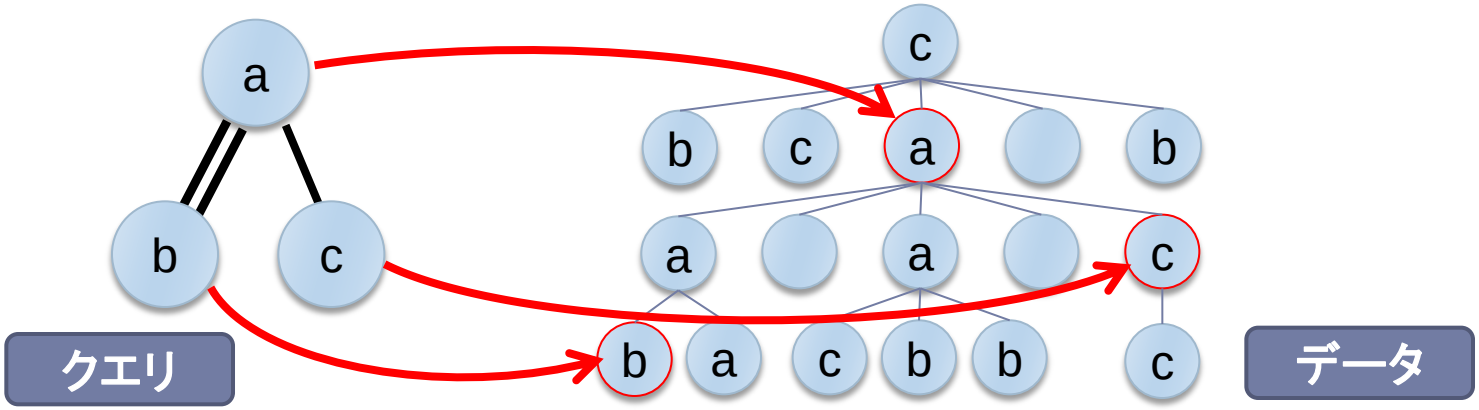


Figure 8: Query answering time for the SXSI and MonetDB

Fast Optimal Twig Joins

/ N. Grimsmo, T. A. Bjørklund and M. L. Hetland (Norwegian Univ.)

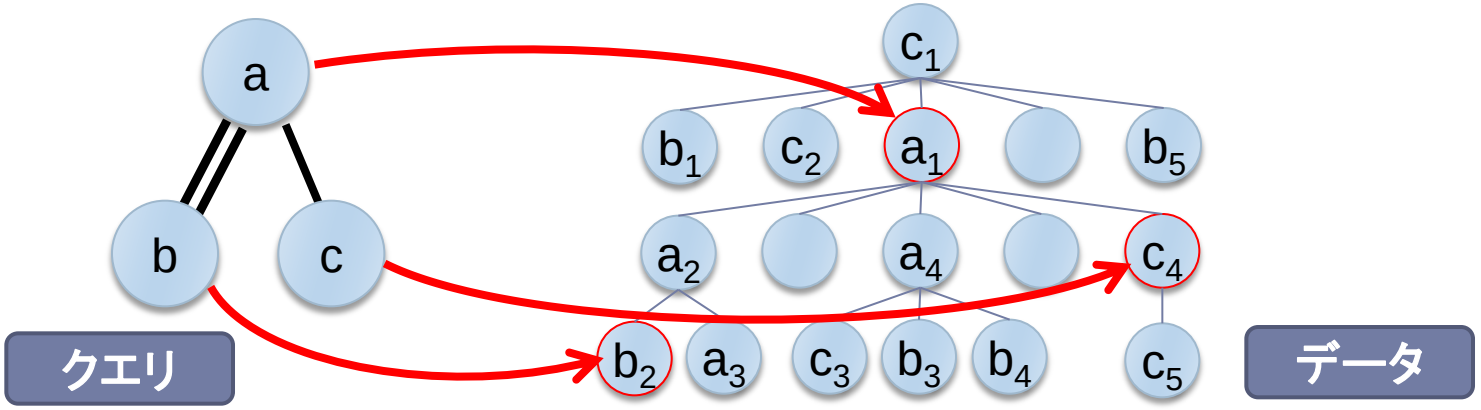
- [背景] XML検索システムでは構造を指定するために twig クエリを用いる ➔ **Twig join アルゴリズム**
  - Twig Pattern Matching (TPM) はXML検索システムのワークロードの大半を占めるため重要
  - 多くの twig join アルゴリズムが実用上は高パフォーマンスだが，最悪計算量が指数関数的なものが多い
- [アプローチ] 新しいデータ構造と実用上，最悪ケースの両方で効果的な twig join アルゴリズムを提案する

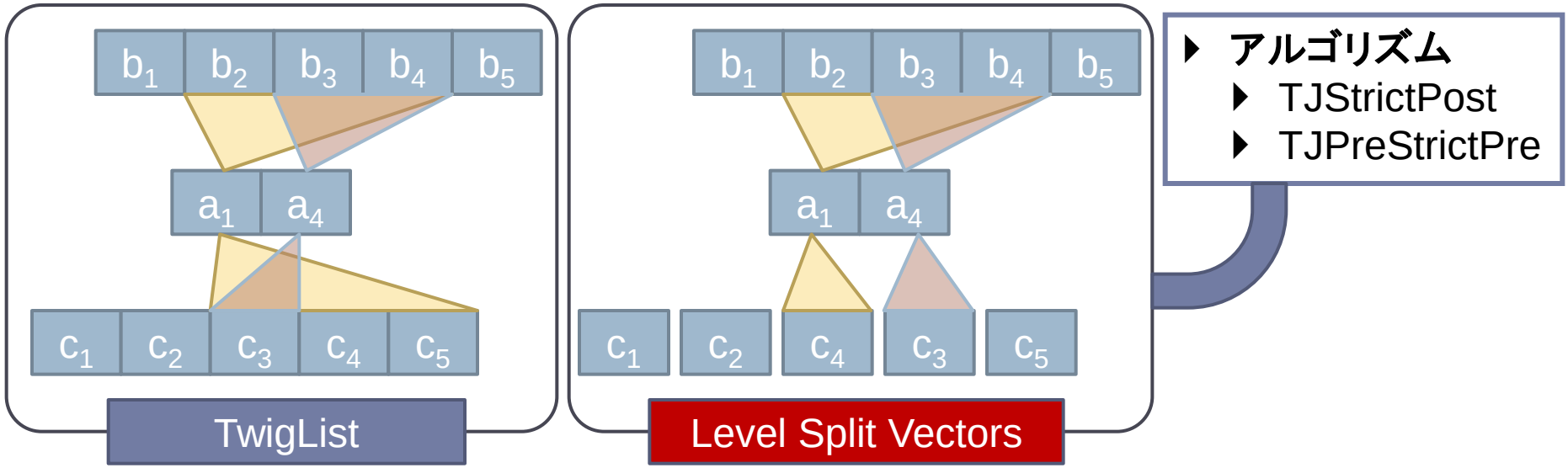


過去のアルゴリズム

アルゴリズム	特徴	比較
TwigStack [2]	親子関係と祖先子孫関係が混ざると twig クエリを線形時間では処理できない	
Twig^{2}Stack [3]	中間結果としてポストオーダーのストリームに変換する	最初の線形アルゴリズム
HolisticTwigStack [8]	中間結果としてプリオーダーのストリームに変換する	最悪ケースで最適ではないが，平均的なケースではTwig^{2}Stack よりも早い
TwigList [11]	中間結果をベクトルの形でポストオーダーで保持する	最悪ケースでは非線形だが，実用上ではTwig^{2}Stackよりも効果的
TwigFast [10]	中間結果をベクトルの形でプリオーダーで保持する	実用上では最も優れている

アプローチ


a
b
c
クエリ
c1
b1 c2 a1 b5
a2 a4 c4
b2 a3 c3 b3 b4 c5
データ

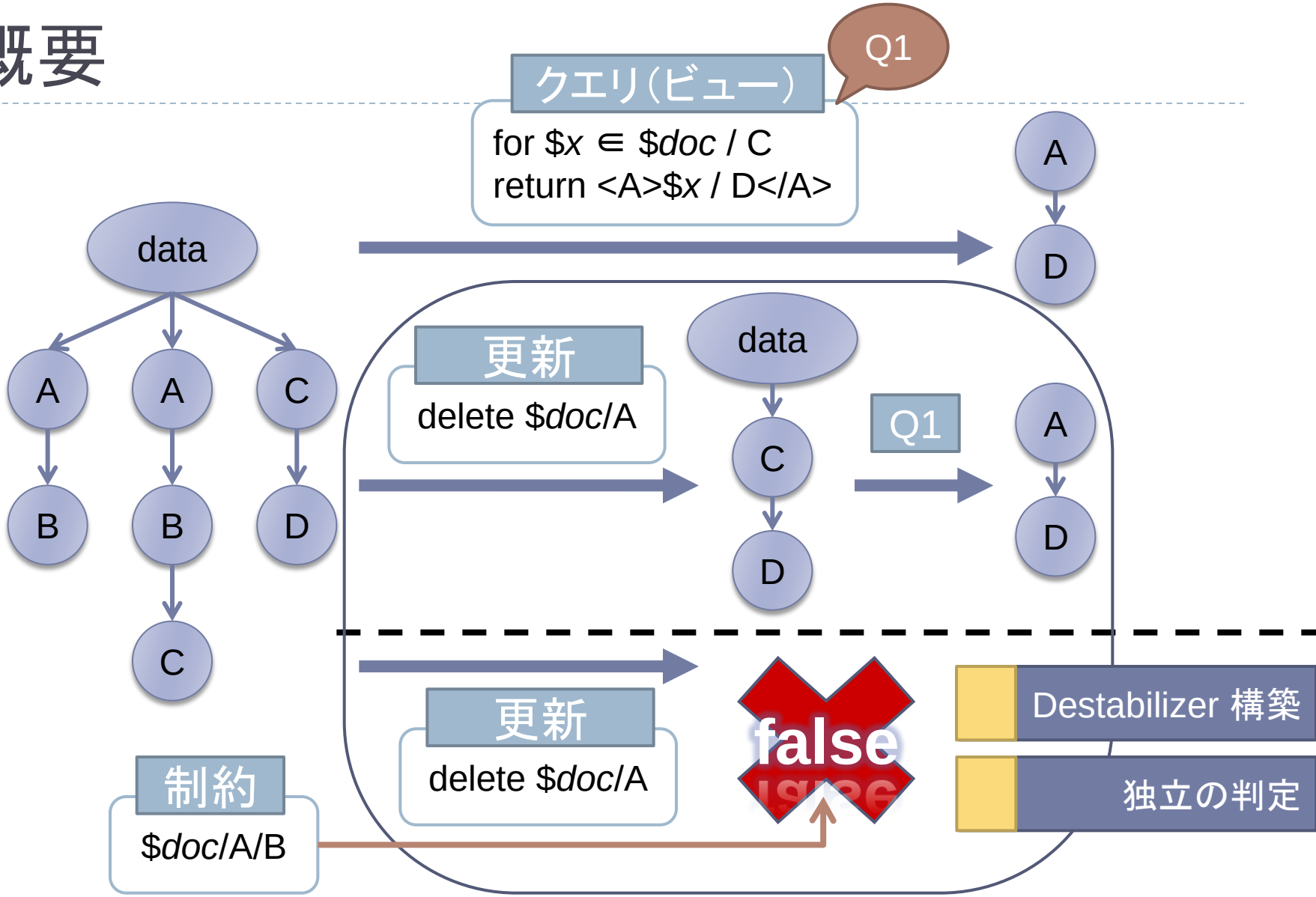

b1 b2 b3 b4 b5
a1 a4
c1 c2 c3 c4 c5
TwigList
b1 b2 b3 b4 b5
a1 a4
c1 c2 c4 c3 c5
Level Split Vectors
アルゴリズム
TJStrictPost
TJPreStrictPre

Destabilizers and Independence of XML Updates

/ M. Benedikt (Oxford Univ.) and J. Cheney (Univ. of Edinburgh)

- [背景] ビューのメンテナンスや制約の評価
  - データの更新がビューを変化させる
  - 制約を満たすかの判定をする
  - これまでは. . .
    - SQL ➔ かなり研究されてきた
    - XQuery ➔ さほど研究されていない
- [アプローチ] Destabilizer を特定し，更新とのオーバーラップの有無を確認する
  - Destabilizer：ビューの更新や制約の評価が必要となるようなデータの更新

概要


Q1
クエリ(ビュー)
for $x ∈ $doc / C
return <A>$x / D</A>
data
A
A
C
B
B
D
C
A
D
更新
delete $doc/A
data
C
D
Q1
A
D
更新
delete $doc/A
false
制約
$doc/A/B
Destabilizer 構築
独立の判定

アイデア

1. 変換ルール (Fig. 2, 3 and 4) を用いてクエリを Destabilizer に変換する

例）

Q1

Δ^{v} (for $x ∈ $doc/C return <A>$x/D</A>)
= Δ^{for} ($doc/C, $x, Δ^{v} (<A>$x/D</A>)
= Δ^{n} ($doc/C), for $x ∈ $doc/C return Δ^{v} (<A>$x/D</A>)
= ($doc, $doc/C, $doc/C/D/desc_or_self:*)

2. 独立性の判定をする
   - Destabilizer と更新に共通のものがなければ独立
   - Existential First-Order logic over tree (EFO(tree)) やヒューリスティックを利用