パスプルーニングによる決定性有限オートマトンを用いたXQuery処理の提案

A Proposal for XQuery Processor with Deterministic Automaton and Path Pruning

石野　明* 竹田　正幸†‡
Akira ISHINO Masayuki TAKEDA

本稿では，文字列検索技術を応用した単純で高速なXQuery処理手法を提案する．XMLデータからAho-Corasick法を拡張したXMLデータ走査器によって高速にタグやキーワードの検出を行い，パスに応じた処理を行うことでXQueryの質問式を処理する．特にパスの照合を高速に処理するために，XMLデータから得られるパススキーマを用いることで決定性有限オートマトンを構築するパスプルーニングを提案する．また，実際に本手法の実装を行い，その実験結果について述べる．

This paper describes a simple, efficient XQuery processor by applying text search methods. First, the processor detects tags and keywords by using a pattern matching machine, which is based on the Aho-Corasick method. Second, the processor uses the DFA obtained from path pruning to process the sequence from the pattern matching machine. Finally, the processor outputs the result of a query. We also validate the processor and provide performance results.

1. はじめに

今日，科学研究，電子商取引，WWWをはじめとした，様々な分野においてXMLデータが共通のフォーマットとして利用されている．このような大規模なXMLデータに対して，特定の条件に合致する部分を見つけ出し，それらを特定のフォーマットへと加工したり，ときには集計などの処理結果を出力することは重要な問題である．

そこで本稿では，World Wide Webコンソーシアムによって提案されたXMLデータに対する検索言語XQuery [16] を文字列検索技術を応用することで処理する手法を提案する．手法は大きく3つの部分からなる．まず最初に，XMLデータからタグやキーワードの検出を高速に行うためにAho-Corasick法を拡張したXMLデータ走査器を用いる．次に，XMLデータから得られるパススキーマを用い質問式に対してパスプルーニングを行い，決定性有限オートマトン（以下，DFA）を構築する．最後に，このDFAを用いてXMLデータを処理することで検索結果を得る．

DFAを用いてXQueryを処理することによって，XMLデータのDOM木への変換や，関係データベースへの格納などの必要がなく，少ない領域で手軽にXMLデータに対する検索・統計・変換処理を行うことができる．

XQuery処理に関してはこれまでにも多くの手法が提案されている [3, 4, 10, 11, 12, 13]．これらの手法と比較すると，我々の手法は扱えるXQueryが制限されながらもDFAを用いた単純で高速な手法であることが特徴である．

また，XQueryの一部であるパスの照合に関するXPath処理を非決定性有限オートマトン（以下，NFA）あるいはそれらから生成されるDFAを用いて処理をする手法も提案されている [8, 9, 18]．我々の手法では，パスプルーニングを用いることで質問式中のパスを具体化し，NFAからDFAへと変換するのではなく，直接，コンパクトなDFAを構成する．

また，本手法を実装し実際のXMLデータを用いた実験およびXMarkベンチマーク [14] の結果を示す．

本稿の構成は次の通りである．2節では，XQueryの部分族，XMLデータ走査器，NFAによるパスパターン照合について述べ，本稿における議論の準備をおこなう．3節では，パスプルーニングとそれによって得られるDFAによってパスパターンの照合が行えることを示す．4節ではDFAに対する処理の割り当てによるXQueryの質問式の処理手法を述べ，5節では実装と実験の結果を報告する．最後に6節でまとめを行う．

2. 準備

2.1 XQueryの部分族

本稿ではXQuery [16] のすべてを扱うことはせず，XMLデータを先頭から順に読み込んで処理が可能な範囲に止める．

まず，パスとして子軸と子孫軸のみを扱い，親軸や先祖軸等の他の軸は扱わない．親軸や先祖軸を考慮した場合，処理の途中で注目しているある場所の前方や後方を調べる必要があるため，処理時間がパスの大きさに対して指数関数的に増大するケースがあることが知られている [6]．

また，条件式はパスは最後だけに現れるものとする．例えば，パス /dblp/www[year="2001"]/title は，条件式がパスの途中に出現しており，パスの最後ではない．このパスは本質的に /dblp/www/tilte[../year="2001"] と同じであり，先に述べた親軸に関する問題に帰着される．条件式中には，比較演算，算術演算，論理演算およびcountやmaxなどの統計関数を考える．最後に，FLOWR (for-let-order-where-return) 形式のうち基本となるfor-return文を扱う．入れ子になったfor, return中のパスは，外のforで指定されたパスの子軸あるいは子孫軸に制限される．

多くのNFAを用いたXPath処理手法では，パスが検出されたことをNFAの状態を用いて表現している [8, 9, 18]．しかし，条件式の処理は単純なNFAでは行えない．さらに統計関数やfor文を扱うためには，それ以上の情報をパスの検知と同時に保持せねばならない．

2.2 XMLデータ走査器によるストリーム処理

XMLデータを高速に走査し，タグの検出と同時に，与えられたキーワードのパターン照合も行いたい．このような目的のために，タグ文字列（<name>や</email>など）やキーワードの検出のために，複数文字列照合アルゴリズムのひとつであるAho-Corasick法 [1] を応用する．しかし，タグ文字列は，不定長の空白文字列を含んでいたり，属性に関する記述を含んでおり，単純な文字列ではない．また，当然ながらタグ名とキーワードは区別されなければならない．そのため，Aho-Corasick法をそのまま適用することができない．

文献 [15, 17] において，著者らは，拡張語頭符号法のもとで表現された文字列に対しても，Aho-Corasick法が拡張できることを示した．この手法を適用すれば，単一のXMLデータ走査器によって，キーワードとタグの検出を効率的に行うことができる．XMLデータ走査器の状態数および構築時間は，いずれも入力である質問式のサイズに関して線形である．

XMLデータ走査器は字句解析とタグやキーワードの検出は同時に行う．それにも関わらず，XMLデータ走査器はSAXパーサと同等の速度で動作する．

2.3 NFAによるパスパターンの照合

XQueryの質問式は，パスによって示される部分に対する操作を記述したものといえる．一般に，XQueryによって処理の対象と

*正会員 九州大学 大学評価情報室 ishino.uoc@mbox.nc.kyushu-u.ac.jp
†九州大学大学院システム情報科学研究院情報理学部門 takeda@i.kyushu-u.ac.jp
‡科学技術振興機構 戦略的創造研究推進事業

なる部分はデータ全体のうち僅かな部分であり，それらの部分を見つけた後の処理をどうするかよりも，それらの部分をいかに高速に見つけるかということが重要である．つまり，パスを高速に処理することが XQuery の処理の高速化に直接，結びつく．

最初にパス $p_1[p_2]$ をパスの基本形として考える．ここで，p_1, p_2 は，いずれも条件式を含まないパスであるとする．このパスは根からの経路が p_1 である節点 x のうち，x からの経路が p_2 となる節点 y が存在する，そのような節点 x を指定している．このとき，パス p_1 のことを**親パス**といい，パス p_2 のことを**子パス**と呼ぶことにする．また，質問式中のパスには//で表されるワイルドカード文字が含まれることがある．そこで，質問式中のパスを**パスパターン**と呼んで，//を含まない通常のパスと区別する．

パスパターン $p_1[p_2]$ に対する **NFA によるパスパターン照合器**は次の 2 つの非決定性有限オートマトン (*NFA*) $N_{p1,p2}$, N_{p2^R} からなる．N_{p1} をパスパターン p_1 に対する NFA であり，N_{p2} を p_2 に対する NFA であるとすると，$N_{p1,p2}$ は N_{p1} と N_{p2} を ε 遷移で繋いだものである．N_{p2^R} はパスパターン p_2 の転置パターン p_2^R に対する NFA である．

あるタグの系列 T に対してパスパターン $p_1[p_2]$ に対応した $N_{p1,p2}$ が受理状態となった場合は，T の先頭からあるタグ x までがパス p_1 に対応し，x から T の最後までが p_2 となる対応が必ず存在する．しかし，そのような x は必ずしも一意には定まらない．また，T の先頭からタグ x までがパス p_1 に対応したからといって，x 以降の T の系列が p_2 に対応するとは限らない．このことから N_{p2^R} が必要となっている．

例えば，a, b をタグ名とするとき，パスパターン $a//b[a//b]$ を考える．パスパターン $a//b[a//b]$ に対する NFA によるパスパターン照合器を図 1 に示す．このとき，パスパターン $a//b[a//b]$ とパス $abbaabab$ の対応を考えると，$abb[aabab]$ と $abbaab[ab]$ の 2 つが考えられる．また，親パス $a//b$ に対して ab が対応するが，残りの $baabab$ に対して子パス $a//b$ は対応しない．さらに接尾辞 $abab$ に対して小パス $a//b$ が対応するが，残りの接頭辞 $abba$ に対して親パス $a//b$ は対応しない．

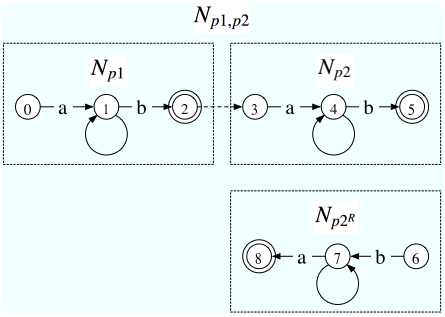

図 1. パスパターン $a//b[a//b]$ に対するパスパターン照合器．

Fig. 1. A path pattern matching machine for a//b[a//b].

3. パスプルーニングと DFA によるパスパターン照合

NFA によるパスパターン照合器は，質問式中に含まれる各パスごとに用意する必要があり，かつ，それらすべてを開始タグや終了タグが現れるたびに動作させる必要がある．結果，質問式中に現れるパスの数に比例して検索時間がかかることになる．

そこで，対象とする XML データからパススキーマを抽出し，そのパススキーマを用いて質問式中のすべてのパスに対してただ 1 つの DFA を構成する手法を述べる．これにより，質問式中に現れるパスの数に検索時間が依存することはなく，また DFA のみを用いるため高速な検索が可能となる．

3.1 パススキーマ

パススキーマとは，ある XML データに含まれるすべてのパスの種類を表したものである．形式的には次のように定義される．

定義 1 XML データ d に対するパススキーマ s とは，d 中のすべての根からのパスをただ一つだけ持ち，かつ，s 中のすべての根からのパスは d 中に現れるもののことである．

パススキーマの定義はデータガイド [5] およびグラフスキーマ [2] を基にしている．データガイドやグラフスキーマは，グラフ構造によってデータベースにおけるデータの構造を示したものである．それに対して，XML データは木構造をしており，パススキーマは XML データ中に現れるすべてのパスの接頭辞木となる．

パススキーマは質問処理に先立ってあらかじめ入力された XML データを読み込むことによって構築される．パススキーマ構築にかかる時間は入力 XML データのサイズに対して線形時間であり，入力が固定されれば構築は質問式によらず 1 度だけである．

一般に，パススキーマは対象となっている XML データが「しっかりとした」構造を持つほど元のデータサイズに比してずっと小さなものとなる．

3.2 パスプルーニング

ここでは，パススキーマを用いることによってパス中の曖昧さを解消し，その結果，ただ 1 つの DFA だけを用いてパスパターンの照合を高速に処理する手法を提案する．

まず，パススキーマを根から走査することによって，XML データが入力されたときと同様に，開始タグと終了タグの系列を得ることができる．例えば，次の出力をパススキーマから得たとする．

```
1:  START TAG:  employees
2:  START TAG:  emp
3:  START TAG:  name
4:  END TAG:    name
5:  START TAG:  email
6:  END TAG:    email
7:  START TAG:  ssn
8:  END TAG:    ssn
9:  END TAG:    emp
10: END TAG:    employees
```

この出力をもとに 2.3 節で述べた NFA によるパスパターン照合器によるパスの照合を行い，各 NFA が終了状態となるパスを調べ上げる．例えば，//name というパスパターンに対して構成される NFA に上記の出力を与えると 3 行目で終了状態となる，すなわち，パスパターン//name に対してパス/employees/emp/name が対応することが分かる．同様に，//emp[//email] というパスパターンに対して構成される NFA に上記の出力を与えると 5 行目，すなわち/employees/emp/email で終了状態となる．このとき，親パスおよび子パスの位置はそれぞれ/employees/emp と/employees/emp/email であることが分かる．

このようにして，すべての終了状態となるパスを調べ上げる．そうして得られたパスは，質問式中のパスパターンとは異なり，ワイルドカード文字//あるいは*を持たないものとなる．このようにパススキーマを用いてワイルドカード文字を含むパスパターンを具体的なパスへと変換する操作を**パスプルーニング**と呼ぶことにする．パスプルーニングされたパスに対しては，図 2 に示されるように容易に DFA によるパスパターン照合器を構築することができる．

パスプルーニングを行った結果，XML データ走査器の出力に対してただ 1 つの DFA を遷移させるだけでパスパターンの照合が行えるようになる．

すべてのパスはパススキーマ上に現れることから，DFA は高々パススキーマのサイズ程度の状態数となり，かつ，遷移に用いら

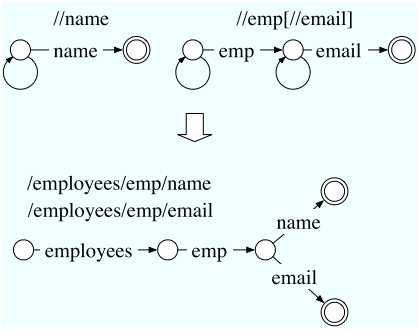

図 2. パスプルーニング.

Fig. 2. Path pruning.

れるタグの種類も入力データにもよるが一般にはその種類数は少ない．したがって，DFA の実装は単純な表参照を用いた実装でよく，非常に高速に動作が可能である．

また，子パスが対応した際に，N_{p2^R} を実行し，どこが親パスの位置であるかを調べていたが，パスプルーニングを行うことで，予め親パスの位置を特定することができ，親パスの位置を探すという必要もなくなる．

4. XQuery 処理

前節で述べた DFA によるパスパターン照合器を用いると，質問式 $p[q]$ に対して，親パス p と子パス p/q それぞれに対応するノード n_p と n_q を見つけることができる．それらのノードをそれぞれ**親ノード**と**子ノード**と呼ぶことにする．

単純な質問式 $p[q]$ の場合は，子ノードが見つかった時点で親ノードを出力すればよい．この処理を形式的には次のように書くこととする．

$$\begin{aligned} n_p &: output(n_p) \text{ if } v(n_p, p/q), \\ n_q &: v(n_p, p/q) := true. \end{aligned}$$

これは，子ノード n_q においては，子パス p/q が存在するということを親ノード n_p に記録し，親ノード n_p においては，子パス p/q が存在することが記録されていたならば親ノード n_p を出力するということを表している．

なお，親ノード n_p に設定された処理を行う際には，子ノード n_q に関する処理は終わっていなければならないため，処理は終了タグの時点，すなわち帰りがけ順で行われる．

このように質問式に応じた処理を設定しておくことで，該当するパスが見つかった際に，設定された処理を順次，実行することによって質問式に対する回答を得ることができる．同様にして，質問式に応じた処理を親ノードおよび子ノードに設定することで，他の XQuery の質問式に関しても処理が可能であることを示す．

質問式中で問題となるのはパスを持つ部分に関してのみである．パス以外の部分については数や文字列は定数であり，算術演算，論理演算，比較演算のいずれの演算結果もその部分式の値によって定まる値である．いずれもパスに関する値が求まれば質問式全体の値が定まる．

例えば，論理演算を含むパス $p[q \text{ and } r]$ に対して，各パス p, q, r に対するノード n_p, n_q, n_r での処理は次の通りとなる．

$$\begin{aligned} n_p &: output(n_p) \text{ if } v(n_p, p/q) \wedge v(n_p, p/r), \\ n_q &: v(n_p, p/q) := true, \\ n_r &: v(n_p, p/r) := true. \end{aligned}$$

パスパターンが入れ子になっている場合，例えば，$p[q[r]]$ といった場合は，一番，内側の子パスから再帰的に処理を行う．まず，親パスを p/q，子パスを r として考え，次に，親パスを p，子パスを $q[r]$ として考える．

$$\begin{aligned} n_p &: output(n_p) \text{ if } v(n_p, p/q[r]), \\ n_q &: v(n_p, p/q[r]) := true \text{ if } v(n_q, p/q/r), \\ n_r &: v(n_q, p/q/r) := true. \end{aligned}$$

このように再帰的に処理を設定することによって，入れ子のパスも同様に扱うことができる．

関数 $count$ は，$p[count(q) > 1]$ のように用いられるが，このとき子パス q に対する子ノード p において親パス p に対する親ノード n_p に保存された $count(p)$ の値を 1 ずつ増加させ，親ノードでは保存された値を式の値とすることで処理される．すなわち，

$$\begin{aligned} n_p &: output(v(n_p, count(q))) \\ &\qquad \text{if } v(n_p, count(q)) > 1, \\ n_q &: v(n_p, count(q)) := v(n_p, count(q)) + 1. \end{aligned}$$

代表的な関数として $count$ を取り上げたが，その他の関数 sum, max, min, avg なども同様にして扱うことができる．

最後に for 文 `for $v in` p `return` e は，式 e の値をパス p の親ノード n_p へと保存し，親ノードでは保存された値を式の値することによって処理される．つまり，設定するべき処理は

$$n_p \quad : \quad output(v(p, e)).$$

となる．ここで，$v(p, e)$ の値は，e 中の質問式によって設定される．

例えば，次の質問式を考える．

```
for $a in /dblp/*[year="2005"]
return $a/title
```

ここで，p を/dblp/*，q を/dblp/*/year，r を/dblp/*/title とするとき，それらのノードに対して設定される処理は以下の通りとなる．

$$\begin{aligned} n_p &: output(v(n_p, e)) \text{ if } v(n_p, p[year = \text{"2005"}]), \\ n_q &: v(n_p, p[year = \text{"2005"}]) := true \\ &\qquad \text{if } n_q = \text{"2005"}, \\ n_r &: v(n_p, e) := n_r. \end{aligned}$$

5. 実装

上記の手法を実際に計算機上に実装を行い実験を行った．実験に用いた環境は 1.8GHz Pentium 4, メモリ 1GB, Gentoo Gnu/Linux 2.6.9, J2SE 1.4.2 である．

本稿で述べた XQuery 処理手法を XMark ベンチマーク [14] を用いて評価する．XMark ベンチマークは XML 質問処理の性能を評価するための XML データ生成ツールと 20 の質問式からなる．我々の XQuery 処理手法では結合処理とソート処理について実装を行っていないため，それらの質問式については評価を行っていない．

XMark ベンチマークの結果は図 3 に示した．100 MB の XML データを例にとると，処理時間は 2.29 秒から 3.47 秒の間となった．

いくつかの商業的な XQuery プロセッサや XMLTK では，XML データをバイナリ形式に変換することによる実行時間の改善が行われている [4, 7]．本手法でも同様にタグ名などのバイナリ形式への変換も行った．また，実行時に DFA が失敗状態に入ったときを考える．このとき，Green ら [7] における NFA と同様に，本稿の手法の DFA においても次の対応する終了タグが現れるまで XML データを読み飛ばして構わない．なぜなら，その時点まで失敗状態から DFA が抜け出すことは決してないからである．そこで変換の際に，各開始タグごとに，開始タグから終了タグまでのオフ

セット値をあらかじめ書き込んでおく．バイナリ形式を用いた場合は 0.11 秒から 1.03 秒となった．

いずれの場合も，本稿で述べた XQuery 処理手法は入力 XML データのサイズに対して処理時間は線形に安定して増加していることが確認された．

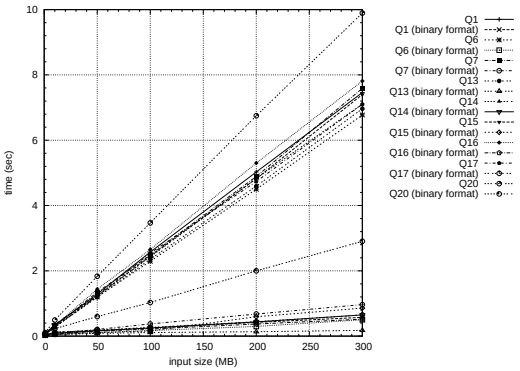

図 3. XMark ベンチマークの結果．

Fig. 3. The results of XMark benchmark queries.

6. まとめ

本稿では，パスプルーニングと決定性有限オートマトン (DFA) を用いて XQuery 質問式を処理する単純で効率の良い手法を提案した．XML データの構造を縮約したパススキーマを用いたパスプルーニングによって，質問式中のパスの曖昧さを解消し具体的なパスの集合へと変換することによって，パスパターンの照合が DFA を用いて可能となることを示した．さらに，その DFA の状態に適切な処理を割り当てることによって XQuery の検索・統計・変換といった処理が行えることを示した．また，本稿の手法を実装し，実際の XML データを用いた実験を行い，本稿で述べた手法がどのような質問式に対しても，入力データのサイズに対して線形時間で出力が得られることをみた．

今後は，大量の質問式に対する同時処理といった大規模な問題に対する応用を考えている．また，同時に本稿では XQuery の部分族を扱うにとどまったが，より豊富な XQuery の取り扱いについても拡張を行っていきたい．

[文献]

[1] Aho, A. V. and Corasick, M. J.: Efficient String Matching: an aid to Bibliographic Search, *Commun. ACM*, Vol. 18, No. 6, pp. 333–340 (1975).

[2] Buneman, P., Davidson, S. B., Fernandez, M. F. and Suciu, D.: Adding Structure to Unstructured Data, *ICDT '97: Proceedings of the 6th International Conference on Database Theory*, pp. 336–350 (1997).

[3] Diao, Y. and Franklin, M.: Query Processing for High-Volume XML Message Brokering, *Proc. of the 29th VLDB Conference* (2003).

[4] Florescu, D., Hillery, C., Kossmann, D., Lucas, P., Riccardi, F., Westmann, T., Carey, M. J. and Sundararajan, A.: The BEA Streaming XQuery Processor, *The VLDB Journal The International Journal on Very Large Data Bases*, Vol. 13, No. 3, pp. 294–315 (2004).

[5] Goldman, R. and Widom, J.: DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases, *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, Morgan Kaufmann Publishers Inc., pp. 436–445 (1997).

[6] Gottlob, G., Koch, C. and Pichler, R.: Efficient Algorithms for Processing XPath Queries, *VLDB 2002*, pp. 95–106 (2002).

[7] Green, T. J., Gupta, A., Miklau, G., Onizuka, M. and Suciu, D.: Processing XML streams with deterministic automata and stream indexes, *ACM Trans. Database Syst.*, Vol. 29, No. 4, pp. 752–788 (2004).

[8] Green, T. J., Miklau, G., Onizuka, M. and Suciu, D.: Processing XML Streams with Deterministic Automata, *ICDT '03: Proceedings of the 9th International Conference on Database Theory*, Vol. LNCS 2572, pp. 173–189 (2003).

[9] Gupta, A. K. and Suciu, D.: Stream Processing of XPath Queries with Predicates, *Proc. SIGMOD*, pp. 431–442 (2003).

[10] Ives, Z. G., Halevy, A. Y. and Weld, D. S.: An XML Query Engine for Network-Bound Data, *The VLDB Journal The International Journal on Very Large Data Bases*, Vol. 11, No. 4, pp. 380–402 (2002).

[11] Josifovski, V., Fontoura, M. and Barta, A.: Querying XML streams, *The VLDB Journal*, Vol. 14, pp. 197–210 (2005).

[12] Koch, C., Scherzinger, S., Schweikardt, N. and Stegmaier, B.: FluXQuery: An Optimizing XQuery Processor for Streaming XML Data, *Proceedings of the 30th VLDB Conference*, pp. 1309–1312 (2004).

[13] Ludäscher, B., Mukhopadhyay, P. and Papakonstantinou, Y.: A Transducer-Based XML Query Processor, *Proceedings of the 28th VLDB Conference* (2002).

[14] Schmidt, A., Waas, F., Kersten, M. L., Carey, M. J., Manolescu, I. and Busse, R.: XMark: A Benchmark for XML Data Management, *Proc. of the 28th VLDB conference*, pp. 974–985 (2002).

[15] Takeda, M., Miyamoto, S., Kida, T., Shinohara, A., Fukamachi, S., Shinohara, T. and Arikawa, S.: Processing Text Files as Is: Pattern Matching Over Compressed Texts, Multi-Byte Character Texts, and Semi-Structured Texts, *SPIRE 2002*, Vol. LNCS 2476, pp. 170–186 (2002).

[16] W3C: XQuery 1.0: An XML Query Language, W3C Working Draft (2004).

[17] 竹田正幸，石野　明，辻　寿嗣，　宮本：ストリーム指向の高速 XML データ処理技法について，データベースと Web 情報システムに関するシンポジウム (DBWeb2003) 予稿集 (2003).

[18] 森川裕章，浅井達哉，有村博紀：データストリーム処理のための効率良い XPath 問合せ機構，情報処理学会研究報告 DBS-131，Vol. 71, pp. 211–218 (2003).

石野　明　Akira ISHINO

九州大学大学評価情報室助手．1999 北海道大学大学院工学研究科電子情報工学専攻博士課程修了．博士（工学）．半構造データベース，自律型ロボットに関する研究と開発に従事．日本データベース学会会員．

竹田　正幸　Masayuki TAKEDA

九州大学大学院システム情報科学研究院情報理学部門教授．科学技術振興機構戦略的創造研究推進事業．1989 九州大学大学院総合理工学研究科情報システム学専攻修士課程修了．博士（工学）．系列データの高速パターン照合処理，大規模系列データからの知識発見に関する研究と開発に従事．情報処理学会創立 40 周年記念論文賞，山下記念研究賞などを受賞．