パストライを用いた高速軽量なXML文書フィルタリング

Light-weight Acceleration For Streaming XML Document Filtering

御手洗 秀一* 石野 明† 竹田 正幸‡§
Shuichi MITARAI Akira ISHINO Masayuki TAKEDA

XML データに極めて単純な前処理を施すことで，質問数の増大に頑健な XML フィルタリング方式を開発した．その手法は，データ配信側で XML データを事前に走査し，根から葉へ向かうパスに沿ったタグ名の列全体を表すパストライと，バイナリデータを作成する。質問式に現れるパスパターンの照合は，パストライを対象として行う．パストライは XML データに比べ非常に小さいため，高速に処理を完了することが可能である．パスパターンの照合結果はバイナリ XML データ走査時に参照することができ，これをテキスト節点を対象としたキーワード照合の結果と結合させることで質問処理は完了する．代表的なストリーム型 XML プロセッサである XMLTK との比較において，実行速度で 2〜6 倍高速であり，メモリ使用量は 1/6 程度であることが判明した．

This paper proposes a scalable XML filtering basing on preprocessing of XML data. XML data is preprocessed and transformed into a pair of a path trie and a binary XML data. The path trie is the trie representing the set of strings of tag names along with root-to-leaf paths. Path pattern matching is performed against the path trie. Since the path trie is much smaller than the XML data, a drastic speedup is possible. Query pattern processing is done by combining the keyword occurrences found in scanning of the binary XML data with the information added to path trie node implied by node IDs embeded in the binary XML data.

Experimental results show that, the processing time and memory requirement of our method are, respectively, $1/6$**–**$1/2$ **and** $1/6$ **compared with XMLTK, a state-of-the-art streaming XML processor.**

1. はじめに

インターネット上のニュース記事や広告等のメディア，日々観測されるセンサー情報や株価表示器など，膨大な量の情報が絶え間なく生成されている．これらの情報を有効に活用するための技術が強く求められている．SDI (Selective Dissemination of Information) は，そのようなサービスの一つであり，日々生成される大量の機械可読文書を購読者が選択的に，かつ，リアルタイムに取得することができるこのような publish/subscribe 型のデータ配送システムは，従来，キーワードや “bag-of-words” に基づくフィルタリングによる比較的単純なものであった．

しかし近年，XML (eXtensible Markup Language) の台頭により，XPath を用いたより高度なフィルタリングが可能となった．XPath は，W3C の勧告による，XML 文書の特定の部分 (要素，属性，テキストなど) を指定するための言語で，事実上の標準となっている．

代表的な XML データストリーム処理器である YFilter [5] や XMLTK [3] は，非常に高速に XPath を処理できるが，扱う軸は順方向軸 (forward axes) のみと限定している．これに対して，最近では先祖軸 (ancestor axes), 兄弟軸 (sibling axes) にも対応したストリーム処理手法が提案されている [4] が，順方向軸のみに限定した手法に比べ規模耐性，処理速度等を犠牲にしている．本論文で提案する手法は，扱う軸は順方向軸のみで，オートマトンを用いて XML データを走査する点では YFilter や XMLTK と同様であるが，XML データに前処理を施すことによって高速化・軽量化を可能とした点で異なる．

YFilter は，XFilter [2] のパス照合処理を改良したもので，質問式毎に非決定性有限オートマトン (NFA) を構築するのではなく，質問式全体の共通接頭辞パスを共通の状態として束ねたもので，XFilter よりも質問数に関して規模耐性があり，高速に動作する．しかしながら，NFA は同時に複数の状態遷移を処理しなければならないため，決定性オートマトン (DFA) と比較して照合速度は遅い．NFA を等価な DFA に変換すると，一般に状態数が指数的に増大する．そこで，XMLTK では，XML データ走査中に必要に応じて NFA を部分的に DFA に変換する *Lazy-DFA* [3] と呼ばれる手法を採用している．XMLTK は，質問数に関して規模耐性があり高速であるが，一方，DFA の状態数が増大し大量のメモリを必要とするという欠点がある．

本論文で提案する前処理手法は，XML データをパストライとバイナリ XML データの対に変換する．パストライは，XML データ中の根から葉へ向かうパスに沿って現れるタグ名の列全体を表すトライである．また，バイナリ XML データは，XML データ中の開始タグと終了タグを各々特殊コードで置き換えたもので，開始タグを表すコードの直後に，タグ名を表す番号ではなく，対応するパストライ節点を識別するための番号（以下，パストライ節点番号）を埋め込んでいる．データ送信サーバは，このパストライとバイナリ XML データを対として送信する．データ受信側では，まず，パストライを対象にパス照合を行い，その結果をパストライの節点に付加する．これにより，バイナリ XML データの走査時にはもはやパス照合の必要はなく，バイナリ XML データ中に埋め込まれたパストライ節点番号を頼りにパス照合の結果を参照することが可能となる．一般に，パストライは XML データ全体の木に比べて十分小さいため，処理速度は劇的に向上する．またパストライは，DataGuide と呼ばれるデータ構造の一種である．XML データ処理の効率化のために DataGuide を用いるアイデアは新しくないが，本論文のような用い方は，著者らの知る限り，これまで存在していない．

2. パストライを用いた高速なフィルタ処理

石野ら [9] では XQuery の部分族をパスプルーニングと DFA を用いて効率的に処理する手法を提案した．その手法は，まず入力となる XML データを事前に調べてパストライを得る．このパストライを用いることで，質問式に含まれるパスパターン中の//や*を展開し，これによって DFA の状態爆発を抑えるというものであった．本論文では，石野らの手法をさらに発展させ，パスパターン照合をデータ走査時に実行するのではなく，質問式を得た段階で完了させておくことによってより高速なフィルタ処理を実現することに成功した．

以下では用語を定義したあと，フィルタ処理に必要な問題を定式化し，本手法のアルゴリズムを紹介する．

2.1 定義

Σ を文字の有限集合とし，$\mathcal{N}$ をタグ名から成る集合とする．*XML* **木**とは，$\mathcal{N}$ のタグ名を内部節点のラベルとし，Σ 上の文字列を葉 (テキスト節点) のラベルとするようなラベル付き順序木をいう．¶

*正会員 九州大学情報基盤研究開発センター mitarai@cc.kyushu-u.ac.jp
†正会員 東北大学大学院情報科学研究科システム情報科学専攻 ishino@ecei.tohoku.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

¶本論文では，“name=value” に対応する属性節点は，“value” をラベルとする葉を唯一の子とし，“@name” というラベルをもつ内部節点として扱う．

単純パスパターンとは $\mathcal{N}$ のタグ名と特殊な記号 "*" から成る記号列をいい，各々の記号は "/" または "//" で区切られているものとする．ここで，"/" と "//" は，それぞれ，親子関係，先祖-子孫関係に対応する．単純パスパターン π が XML 木のパスに**マッチする**とは，"*" を任意のタグ名にマッチするワイルドカード，"//" を任意のタグ名列にマッチする可変長ワイルドカードとみなした際に，π がそのパスに沿ったタグ名の列に合致するときをいう．

パスパターンとは単純パスパターン π_1，π_2 の順序対であり，$\pi_1[\pi_2]$ と表す．XML 木の任意の節点 x とその子孫 y（$x = y$ の場合も含む）に対し，パスパターン $\pi_1[\pi_2]$ が**位置** (x, y) **に出現する**とは，π_1，π_2 が，それぞれ，根から x へのパスと x' から y へのパスにマッチするときをいう．ここで，x' は，根から y へ向かうパスに沿った x の子節点である．

$e = e(w_1, \ldots, w_m)$ を，Σ 上の空でない文字列 $w_1, \ldots, w_m$ の生起の有無に関する論理式であるとする．Σ 上のテキスト d が論理式 e を**充足する**とは，d におけるキーワード $w_1, \ldots, w_m$ の生起に対応した真偽値割り当て (truth assignment) のもとで e が真となるときをいう．

XPattern とは，パスパターン $\pi_1[\pi_2]$ と論理式 e の組をいい，$\pi_1[\pi_2 : e]$ と表す．XPattern $\pi_1[\pi_2 : e]$ が XML 木の**節点** x **に出現する**とは，x の子孫 y が存在して，パスパターン $\pi_1[\pi_2]$ が位置 (x, y) に出現し，かつ，y の子であるテキスト節点の少なくとも一つが e を充足するときをいう．この定義より，パスパターン $\pi_1[\pi_2]$ は，XPattern $\pi_1[\pi_2 : \mathbf{true}]$ と見なすことができる．

本論文では，以下の問題に取り組む．

定義 1
Given: *XML* データ T.
Query: *XPattern* $P_1, \ldots, P_\ell$.
Answer: 各 $i = 1, \ldots, \ell$ に対し，T に対応する *XML* 木において P_i が生起する節点 x を返せ．

ここで，入力として与えられる XML データ T は，複数の XML 文書の系列（XML 木の系列）となっていることを仮定している．

2.2 パストライとバイナリデータ

本手法では XML データに対して一度だけ前処理を行い，パストライと，バイナリ XML ファイルを作成する．XML データと前処理によって作成されるパストライ及びバイナリ XML データとの関係を図 1 に示す．図 1 右上は XML 木を示しており，四角の節点はテキスト節点を表している．また，この XML データから前処理を経て生成されるバイナリ XML データとパストライをそれぞれ，左下と右下に示した．パストライの節点の傍にある数は，その節点の ID（パストライ節点番号）である．バイナリ XML データにおいて，開始タグと終了タグはそれぞれ，特殊記号 "[" と "]" に置き換えられ，"[" の直後にはタグを表す番号ではなく，根からのパスに沿ったラベルの列を表すパストライ節点番号が埋め込まれていることに注意されたい．

このことによって，パスパターン照合はパストライを対象としたパターン照合を行うだけで良く，パストライが XML データに比べ非常に小さいため，高速に処理を完了することが可能となる．

この前処理に要する時間は $O(n \log |\mathcal{N}| + |T|)$ であり，表 2 に示すとおり高速な処理が可能である．ここで，n は XML データ T 中の開始タグの個数を表し，$|\mathcal{N}|$ はタグ集合の要素数を，$|T|$ は XML データのサイズを表す．また，XML データと生成されるデータサイズの関係を表 1, 2 に示す．このことから，パストライは，XML 木に対して十分に小さく，バイナリ XML データに関しても入力ファイルサイズの 76%と小さくなっていることが分かる．

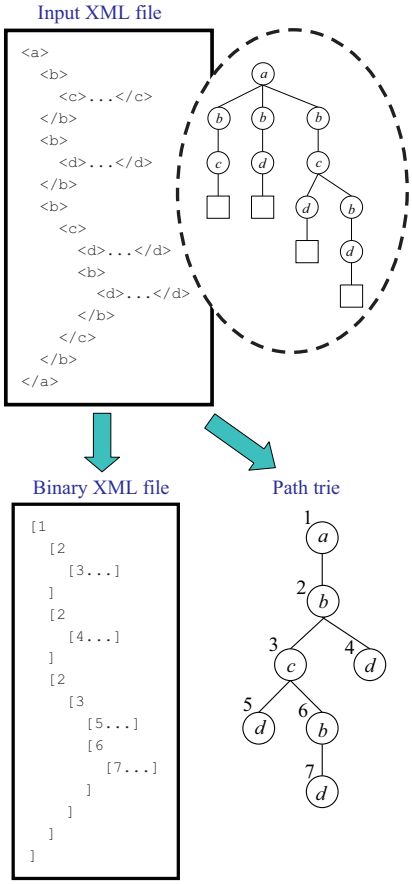

図 1. XML データと前処理で作成されるパストライ，バイナリ XML データ

Fig.1. Preprocessing of XML data.

2.3 パスパターンの照合

本論文では，XML 木に対するパスパターン照合の問題を，パストライに対するパスパターン照合に置き換える．

図 1 で示した XML データを例にとり，XML 木中，および，パストライ中におけるパスパターン P_1 = a//b[//d] の生起の様子を図 2 に示した．

この例では，パスパターン P_1 = a//b[//d] は XML 木中の位置 $(x, y) = (4, 5), (6, 8), (6, 10), (9, 10)$ で生起している．これらの位置は，図右に示したパストライの節点に付加された情報から得ることができる．一方，パストライ中においては，生起位置は $(x, y) = (2, 5), (2, 7), (6, 7), (2, 4)$ である．そこで，パストライ中

表 1. DBLP [6] と xmlgen [7] によってランダムに生成された入力 XML データの大きさ

Table 1. Characteristics of two input XML data.

	XML データ		
	サイズ (MB)	節点数	タグ数
DBLP	352	8,632,812	35
random	111	1,666,310	74

表 2. 前処理で構築されるバイナリ XML データとパストライのサイズ及びその構築時間

Table 2. Sizes of the binary XML data and the path tries.

	バイナリ XML データ	パストライ	CPU 時間
	サイズ (MB)	節点数	(sec)
DBLP	208	138	100.41
random	84	515	73.26

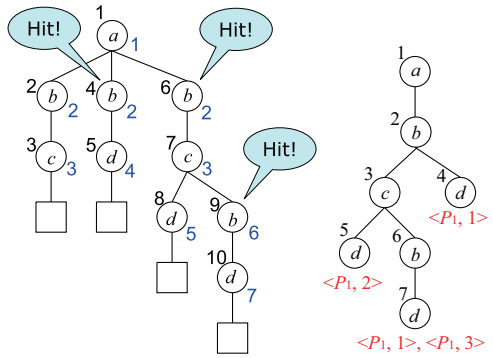

図 2. パストライ・XML 木の関係と，パスパターン P_1 = a//b[//d] の照合結果

Fig.2. Path-pattern matching using path trie.

の生起位置 (x, y) の節点 y にパスパターンの番号と節点 x, y の深さの差の情報を付加する．すると，XML 木中の生起位置は，パストライに付加された情報から得ることができる．たとえば，XML 木中の位置 $(6, 8)$ における P_1 の生起は，パストライの節点 5 のもつ出力 $\langle P_1, 2\rangle$ から得ることができる．すなわち，バイナリ XML データ中において，XML 木の節点 $y = 8$ に対応する箇所にはパストライ節点番号 5 が埋め込まれており，パストライの節点 5 に付加された情報 $\langle P_1, 2\rangle$ によって，y から 2 つ上の先祖である節点 $x = 6$ が得られる．

次に，パストライに対するパスパターン照合の手法について述べる．質問式に含まれる各パスパターンに対し，パストライ中の出現位置 (x, y) をすべて求めなければならない．このために，竹田ら [8] が XML 木に対するパスパターン照合に用いた方法を採用する．

本手法を用いることで，バイナリ XML データ走査時のパスパターン照合処理は不要となり，XML データ走査時にパスパターンの照合を行っている YFilter や XMLTK と比べ，処理時間を大幅に短縮することができる．

2.4 バイナリ XML データ走査アルゴリズム

バイナリ XML データを走査してキーワード照合を行い，その結果とパスパターン照合の結果を結合して質問処理を行うアルゴリズムの概略を以下に示す．

与えられた XPattern $P_1, \ldots, P_\ell$ の論理式 e 中に含まれるキーワードからなる集合を $W = w_1, \ldots, w_m$ とする．W から Aho-Corasick(AC) 機械 M を構築する．バイナリ XML ファイル先頭からの位置を表す変数 $offset$ と，入力を XML 木として見たときの節点の深さを表す変数 $depth$ を用意する．ブール値を格納する 2 次元配列 Occ と Q を用意する．配列 Occ は，現在巡回している深さ d の節点の子であるテキスト節点にキーワード w_i が生起するときに $Occ[d][i]$ の値が真となるというものであり，配列 Q は，現在巡回している深さ d の節点に XPattern P_q が出現するときに $Q[d][q]$ の値が真となる．本アルゴリズムは，バイナリ XML ファイルを 1 バイトずつ走査し，読み込んだバイト c に対して以下を実行する．

- c が開始タグを表す特殊記号 “[” であるとき．パストライ節点番号 v を読み込み，$(v, offset)$ をスタック S にプッシュする．$depth$ を 1 インクリメントする．$Occ[depth][1 \ldots m]$ と $Q[depth][1 \ldots l]$ を偽に初期化する．M の状態を初期状態に設定する．
- c が終了タグを表す特殊記号 “]” であるとき．スタック S より $(v, offset)$ をポップする．もしパストライのノード v が出力をもっていたら，それぞれの $\langle q, d\rangle$ に対して，XPattern P_q の論理式 e を，$Occ[depth][1 \ldots m]$ の値に基づいて計算し，もし e が真ならば，$Q[depth - d][q]$ を真と設定する．$depth$ を 1 デクリメントする．M の状態を初期状態に戻す．
- c がそれ以外のバイトコードであるとき．M を文字 c で状態遷移させる．もし遷移先の状態が出力をもつならば生起したキーワードに対応して $Occ[depth][1 \ldots m]$ を更新する．

3. 実装実験

実験に用いた計算機環境は，RedHat Linux Advanced Server 2.1, CPU 2.4GHz Intel Pentium4, 2.0GB RAM で，入力に用いた XML データは，xmlgen [7] によって生成されたランダムデータでサイズは 111MB である．質問式は，XMLTK が任意の位置ステップ (location step) での述語の使用を許していないため，単純パスパターンを用いた．パスは，YFilter によって提供されている pathgenerator [1] によってランダムに生成され，//, *の生起確率はそれぞれ (1%, 1%), (10%, 10%) としている．

3.1 パスパターン照合に要する時間

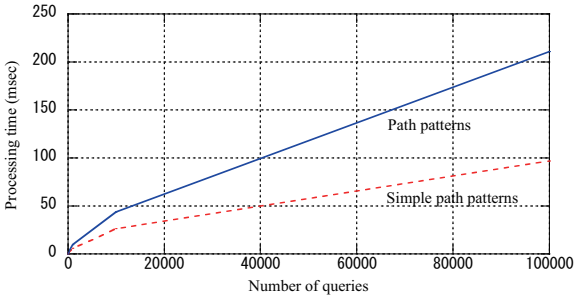

図 3. パストライとパスパターン照合に要する時間

Fig.3. Times for path-pattern matching against path trie.

図 3 は，質問数を変化させたときのパスパターン照合に要する時間を測定したものである．NFA の構築に要する時間も含んでいる．この結果より，質問数が多くなっても実際にバイナリ XML データを走査する時間 (表 4) と比較して，ほとんど無視できる時間であることが分かる．これは，パストライのサイズが入力 XML 木の節点数 1,666,310 個に対して 515 個と小さい（表 2 参照）ことと，ビット並列化技術による NFA の構築・照合がきわめて高速に動作することによる．

3.2 XMLTK および YFilter との性能比較

表 3 は，質問数が 10,000, 100,000 のときのメモリ使用量 (//, *の生起確率は (1%, 1%), (10%, 10%) の 2 通り) 表している．メモリ使用量は，論理空間上にどれだけメモリを確保したのかではなく，プログラム実行中に実際に物理メモリにアクセスしたサイズの最大値を測定している．

質問数が 100,000 のとき，YFilter は本計算機環境では，メモリ枯渇により動作不能となった．質問数が 10,000 のとき (//, *の生起確率は 10%) を見ると，本手法が 4,925KB であるのに対して，

表 3. メモリ使用量の比較

Table 3. Memory usage comparison.

	質問数	メモリ使用量 (KB)		
		本手法	YFilter	XMLTK
単純パスパターン	10,000	3,320	1,169,975	30,288
1%	100,000	18,632		285,328
単純パスパターン	10,000	4,952	1,494,845	34,412
10%	100,000	28,543		318,560

※空欄はメモリ枯渇のため，実施不能であったことを示す．

XMLTK 34,412KB, YFilter 1,494,845KB と優に 6 倍以上であることが分かる．

次に，表 4 は，単純パスパターンを 1〜100,000 個ランダムに与えたときの実行時間を測定したものである．この結果より，本手法が YFilter，XMLTK と比較して圧倒的に高速であることが分かる．また，質問数が 100,000 のときに，本手法と XMLTK の処理時間が大幅に遅くなっているのは，マッチする節点数が多くなりすぎて，出力依存の要因が全体の処理時間の大半を占めるようになっているためで，マッチする節点数の増加率と実行時間の増加率はほぼ一致することを確認している．

表 4. 実行時間の比較

Table 4. Execution time comparison.

質問数	経過時間 (sec)		
	本手法	YFilter	XMLTK
1	0.57	39.24	2.27
10	0.57	42.54	2.57
100	0.57	45.22	3.09
1,000	0.67	61.30	4.14
10,000	1.85	155.30	11.83
100,000	142.04		270.81

※空欄はメモリ枯渇のため，実施不能であったことを示す．

3.3 SIX による高速化検証

本手法をさらに高速化させるための技法として，Stream IndeX (SIX) [3] がある．SIX とは，開始タグとそれに対応した終了タグの位置情報の組から成る集合であり，本手法もこのデータ構造を用いることで，関係のない部分木を読み飛ばすことが可能である．

表 5 が表 4 で示した実験に SIX を追加したときの実行時間である．この結果より，質問数が少ないときには 1.6〜7.2 倍高速化されており，たとえ質問数が多くほとんど読み飛ばしができない場合であっても性能劣化がないことが分かる．また，表 4 との比較より，同じ入力データを用いているにも関わらず XMLTK に比べ，SIX による効果が高いことが分かる．これは，XMLTK が//や*に相当する状態に遷移した後，どのようなタグが現れてもその状態から抜けない限り読み飛ばせるか否かの判断がつかないのに対し，本手法ではそれらをパストライにより予め展開しており，任意のタグで判別可能であるためと考える．

表 5. SIX による高速化の比較

Table 5. Comparison of speedups with SIX.

質問数	経過時間 (sec)	
	本手法	XMLTK
1	0.00	0.14
10	0.07	1.53
100	0.18	2.49
1,000	0.39	4.37
10,000	1.81	12.25
100,000	139.62	275.74

4. まとめと今後の課題

本論文では，パストライ上のパス照合処理と AC 法を用いたキーワード照合処理を結合したストリーム型のフィルタリング技法を提案した．XPath の複数の質問式を同時に処理する手法について述べ，節 3 で示したとおり，一連の実験によって，パストライがフィルタ処理の高速化に寄与することを示し，本手法が YFilter や XMLTK と比較して高い性能を有することが判明した．

また，本手法が効率的に大量の質問式を同時に処理することができることを実験により確認した．これにより，節 1 で述べたように，SDI サービスにおいて大量のクライアントが質問式をサーバに登録するような大規模 SDI システムの構築も可能である．また，質問式が少ない際は，SIX によりさらにフィルタリング性能を向上させることができる．

また，配信するべきクライアント数が急増し，単一計算機で処理をまかなえなくなるような場合でも，同じ XML データを複数の計算機に重複して担当させることで対応可能である．この場合，バイナリ XML データは配信元となるマスターサーバーでただ一度作成するだけでよく，前処理の時間は問題にならない．

今後の課題としては，(1) 単一計算機環境におけるフィルタ処理をさらに高速化する手法を検討することと，(2) 本論文の成果をもとに，前処理を施さない XML データをそのまま扱うタイプのフィルタリング手法を開発すること，(3) 分岐パターン (twig pattern) や親軸・先祖軸などへの対応が挙げられる．

[文献]

[1] : Filtering and Transformation for High-Volume XML Message Brokering, http://yfilter.cs.berkeley.edu/code_release.htm.

[2] Altinel, M. and Franklin, M.: Efficient filtering of XML documents for selective dissemination, *VLDB'00*, pp. 53–64 (2000).

[3] Avila-Campillo, I., Green, T. J., Gupta, A., Onizuka, M., Raven, D. and Suciu, D.: XMLTK: An XML Toolkit for Scalable XML Processing, *PLANX'02* (2002).

[4] Barton, C., Charles, P., Goyal, D., Raghavachari, M., Josifovski, V. and Fontoura, M.: Streaming XPath processing with forward and backward axes, *ICDE*, pp. 455–466 (2003).

[5] Diao, Y., Altinel, H., Franklin, M. J., Zhang, H. and Fischer, P. M.: Path Sharing and Predicate Evaluation for HighPerformance, *ACMTOD* (2003).

[6] Ley, M.: DBLP Computer Science Bibliography, http://dblp.uni-trier.de/.

[7] Schmidt, A., Waas, F., Kersten, M. L., Carey, M. J., Manolescu, I. and Busse, R.: XMark: A benchmark for XML data management, *VLDB'02*, pp. 974–985 (2002).

[8] 竹田正幸，石野　明，辻　寿嗣，宮本　哲：ストリーム指向の高速 XML データ処理技法について，*DBWeb2003* (2003).

[9] 石野　明，竹田正幸：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，日本データベース学会 Letters，Vol. 4, No. 4 (2006).

御手洗 秀一　Shuichi MITARAI

九州大学情報基盤研究開発センター特任助教．2002 九州大学大学院システム情報科学府情報理学専攻修士課程修了．修士（理学）．半構造データベースに関する研究と開発に従事．日本データベース学会会員．

石野 明　Akira ISHINO

東北大学大学院情報科学研究科助教．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．