2005－DBS－137（Ⅱ）（66）
2005／7／14

拡張P-labelによるXMLデータに対する等価条件を含む問合せ処理の効率化の提案

根本 潤 †　遠山 元道 ‡

† 慶應義塾大学大学院 理工学研究科 開放環境科学専攻
‡ 慶應義塾大学 理工学部 情報工学科
E-mail: † jun@db.ics.keio.ac.jp, ‡ toyama@ics.keio.ac.jp

本稿では，XML データにおける値ノードの取り扱いに着目し，等価条件を含む問合せを効率良く処理することができる拡張 P-label を提案する．XPath による問合せでは述語を評価することでノードの絞り込みを行うことができるため，該当する値をもつようなノードを検索したいケースは多くある．したがって，XML データの構造だけでなく，要素内の値も問合せにおいて重要であるにもかかわらず，従来の P-label では値ノードを対象としていなかった．そこで，提案手法では，ハッシュ関数を用いることにより，値の種類数が膨大な場合でもそれらに対して区間を割り当てられるように改良する．プロトタイプシステムを用いた実験では，提案する拡張 P-label の性能について検証し，等価条件を含む問合せにおいて処理速度が向上することを示す．

キーワード：XML，　ラベル付け手法，　等価条件

A Proposal of Efficient Processing Method for Querying XML Data with Equality Predicates Using Extended P-label

Jun NEMOTO †　Motomichi TOYAMA ‡

†School of Science for OPEN and Environmental Systems,
Faculty of Science and Technology, Keio University.
‡Department of Information and Computer Science, Faculty of Science and Technology,
Keio University.
E-mail : †jun@db.ics.keio.ac.jp ‡toyama@ics.keio.ac.jp

This paper proposes an efficient labeling method for querying XML data with equality predicates paying attention to dealing with value nodes. Since XPath can filter nodes using predicates in queries, retrieving nodes, which have a specified value is not rare. Therefore, it is important for processing XPath queries to consider not only the structure of XML data but also the values in elements. However, the traditional P-labeling is not intended for value nodes. Then, this paper attempts to enable the P-label to allocate intervals for a variety of values using a hash function. Experimental results by a prototype system demonstrate that proposed extended P-label provides high-performance processing queries with equality predicates.

keyword : XML , Labeling , Equality Predicates

1 はじめに

XML[11] データに対する XPath[12] や XQuery[13] などの問合せ処理を高速化することは重要な研究課題である．XML データはラベル付き順序木であるため，そのコアオペレーションはノードの先祖子孫関係や親子関係を判定することにある．特に，先祖子孫関係の処理は DOM[10] や SAX[6] でナイーブに処理してしまうと，検索がデータ全体に及んでしまいパフォーマンスが非常に悪い．

そこで，XML データに対してノードの包含関係を維持可能なラベル付けを行い，問合せ中のノードの包含関係をそのラベルに対する構造条件へと変換して処理することで，そうした全体検索を避けるのが一般的である．こうしたラベル付け手法は各種提案されおり，範囲ラベル付け手法 [4] や Dewey Order[8] などがその例である．また，それらに基づいた構造結合のアルゴリズムや索引付け手法も多く提案されている [1, 5].

その一方で，XML データの構造だけでなく，要素内の値の取り扱いも重要である．XPath では述語を評価することでノードの絞り込みを行うことができるため，実際の問合せでは該当する値をもつようなノードを検索したい場合が多々ある．

このように，構造および値の両側面から問合せ処理を考えなければならないにも関わらず，これらを統一的に扱うものは少ない．そこで，本稿では従来の構造結合の処理だけでなく，値ノードの取り扱いにも着目し，等価条件を含む問合せの処理を効率的に行うことができるラベル付け手法について述べる．提案手法では，値ノードについては考慮されていなかった Chen らの P-label [2] を拡張することで，等価条件を含む XPath 問合せの処理を効率化する．拡張 P-label は，P-label と同様に関係データベースにおいて利用することも，ファイルシステム上でネイティブに実装することも可能であるが，本稿では便宜上関係データベース上で利用することを前提に議論を進める．

本稿の構成は以下の通りである．まず，2 章で従来技術とその問題点について述べる．次に，3 章で拡張 P-label とそれを用いた XPath 問合せ手法を提案する．そして，4 章で実装されたプロトタイプによる実験とその評価を述べ，最後に 5 章でまとめと今後の課題について述べる．

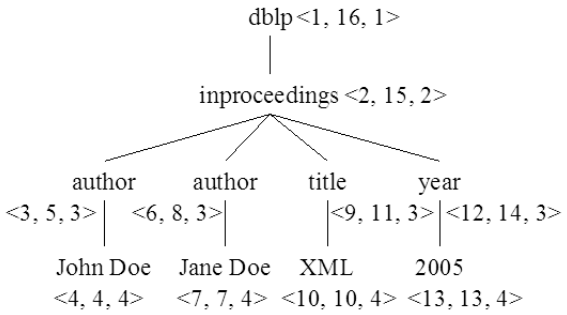

図 1: XML データに対する範囲ラベル付けの例

2 従来技術と問題点

2.1 範囲ラベル付け手法

XML 木中のノードの先祖子孫関係を効率良く判定するために，様々なラベル付け手法が提案されているが，中でも範囲ラベル付け手法が最もよく研究されている．範囲ラベルは，XML データ中の各ノードの開始位置，終了位置および深さによって表される．これを $< start, end, level >$ とすると，図 1 のように XML データにラベル付けすることができる．範囲ラベル付けを行ったとき，異なる 2 つのノード m と n において，n が m の子孫ノードであるとは次の条件で表される．

$$m.start < n.start \text{ かつ } m.end > n.end$$

さらに，n が m の子供ノードであるためには上記の条件に加え，$m.level + 1 = n.level$ を満たせばよい．

しかしながら，範囲ラベル付けのみでは，問合せ中に多数の親子関係や先祖子孫関係が存在する場合に高コストな結合操作を何度も行わなければならない．

2.2 P-labeling

前述の問題に対して，Chen らは P-labeling と呼ばれるラベル付け手法を提案している [2].

P-labeling について述べるにあたり，接尾辞経路式，単純経路式および起点経路式の 3 つの経路式を定める．ここでいう接尾辞経路式は child 軸 (/) または descendant 軸 (//) で始まり，0 個以上の child

軸が続く経路式のこととする．また，単純経路式は child 軸だけを含む接尾辞経路式のこととする．さらに，起点経路式とは XML 木中のルートノードから各ノードまでの一意な単純経路式のこととする．

P-label は連続する child 軸を含む問合せ，すなわち接尾辞経路問合せを結合操作なしに処理することが可能である．というのは，各要素ノードの起点経路に基づいて付与されるラベルと接尾辞経路問合せに付与されるラベルとを比較することで経路式の包含関係を効率良く判定できるためである．

どのような 2 つの接尾辞経路式 P，Q についても以下の性質を満たすような区間 $< p_1, p_2 >$ が P に対する P-label である．

- $P.p_1 \leq P.p_2$
- $P \subseteq Q \Leftrightarrow Q.p_1 \leq P.p_1$かつ $P.p_2 \leq Q.p_2$
- $P \cap Q = \emptyset \Leftrightarrow P.p_1 > Q.p_2$または $P.p_2 < Q.p_1$

上記の性質により，接尾辞経路問合せを Q とすると，XML 木中のノード n のうち解として該当するのは，起点経路式 $SP(n)$ が $Q.p_1 \leq SP(n).p_1 \leq Q.p_2$ を満たすものとなる．P-label の実際の算出方法は次の通りである．ただし，$t_1, \cdots, t_n$ は異なる要素名で，h を XML 木の高さとすると，$m \geq (n+1)^{h-1}$ である必要がある．また，$r = 1/(n+1)$ とする．

1. 経路//に区間 $< 0, m-1 >$ を割り当てる．
2. 区間 $< 0, m-1 >$ を $n+1$ で分割し，先頭から順に経路/，経路$//t_i$ に対して割り当ててゆく．すなわち，経路/には $< 0, m*r-1 >$，経路$//t_i$ には $< p_i, p_{i+1}-1 >$ が割り当てられる．
3. 経路$//t_i$ に対する区間をさらに $n+1$ で分割し，各区間を経路$/t_i$，経路$//t_j/t_i$ に対して割り当ててゆく．
4. XML 木の高さに応じて必要な経路の分だけ再帰的に区間を求める．

図 1 の XML データ中の接尾辞経路に区間を割り当てていった様子を図 2 に示す．

P-label は接尾辞経路問合せに対して結合操作なしに，選択操作のみで答えることができる点で非常に有用であるが，最大の問題点は値ノードに対する考慮がなされていないことにある．

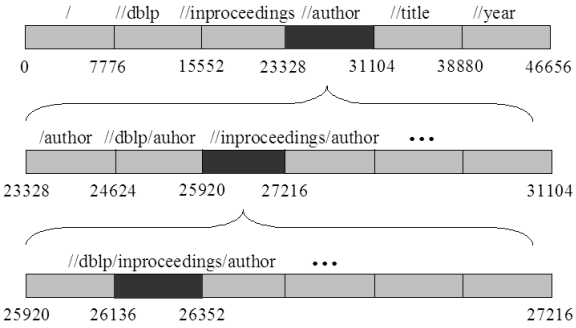

図 2: P-label による区間割り当ての例

例えば，//author[.='John Doe'] といった問合せを考える．//author ノードは多数あるが，その値が Yoko Maeda であるようなものは少数であるといった状況では，解と関係のない多数の//author ノードにアクセスしなければならない．これは結合操作こそ必要としないものの非常に非効率である．

2.3 シーケンスベースアプローチ

XML データに対する効率的な問合せ処理手法としては，ラベル付けに基づくもの以外にも様々なアプローチがある．Rao らの提案する PRIX では，XML データと問合せの双方を，構造情報を保持したシーケンスに変換し，サブシーケンスマッチングを行うことで，結合操作のいらない問合せ処理を実現している [7]．これらの手法では等価条件を扱うことは可能であるものの，主として XML データの構造を効率的に扱うものであり，値ノードを扱う際の効率性については考慮されていない．Wang らはシーケンス中のノードの順序を考慮することで，値ノードの選択度の問題にも対応している [9]．しかしながら，各シーケンス同士の共通部分を重視するか，選択度を重視するかはトレードオフの関係にあり，構造と値を統一的に扱う解決策とは言えない．

2.4 研究目的

こうした従来技術の問題点を踏まえ，本稿では XML データにおける値ノードの取り扱いに着目し，

等価条件を含む問合せを効率良く処理することができるラベル付け手法の確立を目的とする．

3 提案手法

3.1 拡張 P-label

P-label に用いられる値は，大小関係さえ表現できればよいので整数である必要もなければ，その最大値・最小値の制限もない．しかしながら，演算のパフォーマンスを考えたとき，その数値は 32 ビットもしくは 64 ビットの整数として扱うほうが望ましい．仮にそれ以上の整数を扱わざるを得ない場合でも，可能な限り小さな整数区間で表現すべきである．すなわち，P-label の値に割り当てられる区間は実質有限である．したがって，P-labeling において単純に値ノードを扱おうとすると，種類数が増大するために容易に区間を使い果してしまうこととなる．

そこで，拡張 P-labeling では要素ノードに対する区間の割り当ては従来通りとし，値ノードに関してはハッシュされた値に対して区間を割り当ててゆく．ハッシュテーブルのサイズを k とすると，$m \geq (n+k+1)^h$ を満たすような m を用いて区間 $< 0, m-1 >$ を各接尾辞経路式に割り当てればよい．

$n=5$，$h=7$，$k=4$，ハッシュ関数を $H(x)$ としたとき，図 1 の XML データにおける接尾辞経路/dblp/inproceedings/author/'John Doe' に対し拡張 P-label が決定されてゆく様子を表 1 に示す．なお，拡張 P-label の区間割り当て順序は/，dblp，inproceedings，author，title，year，$h_1 \cdots h_4$ であるとする．また，$m = 10000000$ である．

まず，H('John Doe')= h_1 であったとすると，//H('John Doe') に割り当てられる拡張 P-label は $< 6000000, 6999999 >$ となる．次に，$< 6000000, 6999999 >$ の部分区間で author に割り当てる区間を決定する．区間割り当ての順序から//author/H('John Doe') の拡張 P-label は $< 6300000, 6399999 >$ となる．以降も同様に割り当ててゆき，最終的に/dblp/inproceedings/author/'John Doe' に対する拡張 P-label は $< 6321000, 6321099 >$ となる．また，3.2 節の問合せ処理の説明のために，//author，//inproceedings/author，/dblp/inproceedings/author のそれぞれに対して同様にして求めた拡張 P-label も併せて表 1 に示しておく．

等価条件を含む問合せを拡張 P-label により処理すると，大幅なパフォーマンス向上が見込まれる．例えば，//author が 15000 ノード，その値が 5000 通り，ハッシュテーブルのサイズが 500 エントリで，かつ値が理想的に分散していると仮定すると，//auhtor[.='John Doe'] という問合せに対しては，従来の P-label を用いた検索では 15000 ノードにアクセスしなければならないのに対し，拡張 P-label を用いた場合は $30(= (5000/500) * (15000/5000))$ ノードにアクセスするだけで済む．

拡張 P-label を実際に XML データの各ノードに付与するにあたって，1 つ考慮しなければならないのは，値ノードを関係データベースのタプルとしてどのよう扱うかという問題である．P-label では $(start, end, level, plabel, data)$ というスキーマで，XML データの全要素ノードについて，1 ノードを 1 タプルとして関係データベースに格納する．そして，要素ノードが値を含んでいた場合，$data$ 属性に文字列が格納される．一方，提案手法においてはスキーマに相違はないものの，値ノードに対しても P-label を割り当てるため，それ自体が 1 つのタプルとなり，//author[.='John Doe'] といった問合せに容易に答えられるようになる．しかし，//author/text() といった問合せには逆に効率が悪くなる可能性がある．//author という経路の全ての値を得ようとする場合，//author ノードのタプルの $data$ 属性に値が格納されていないと，全ての値ノードのタプルから非連続的に//author 経路上にあるものを検索してゆかなければならないからである．したがって，問合せのパフォーマンスを優先するならば，各要素ノードが値を含んでいる場合には値ノードのタプルとは別に冗長に保持しておかなければならない．

3.2 問合せ処理

本節では，拡張 P-label を用いた XPath 問合せ処理手法について述べる．XML データに対する問合せの大半が child 軸，desendant 軸を中心としたものであることから，これらを中心とした問合せ処理についてのみ述べる．しかしながら，各ノードは開始位置，終了位置，すなわち前置順，後置順を維

表 1: 接尾辞経路式に対する拡張 P-label

接尾辞経路式	拡張 P-label
//H('John Doe')	$< 6000000, 6999999 >$
//author/ H('John Doe')	$< 6300000, 6399999 >$
//inproceedings/author/ H('John Doe')	$< 6320000, 6329999 >$
//dblp/inproceedings/author/ H('John Doe')	$< 6321000, 6321999 >$
/dblp/inproceedings/author/ H('John Doe')	$< 6321000, 6321099 >$
//author	$< 3000000, 3999999 >$
//inproceedings/author	$< 3200000, 3299999 >$
//dblp/inproceedings/author	$< 3210000, 3219999 >$
/dblp/inproceedings/author	$< 3210000, 3210999 >$

持したラベルが付与されているため，XPath における全ての軸を処理することが可能である．

なお，本節では図 1 の XML データを対象に，表 1 を用いながら説明を行う．

3.2.1 接尾辞経路問合せ

2.2 節で述べたように，接尾辞経路とは child 軸 (/) または descendant 軸 (//) で始まり，0 個以上の child 軸が続く経路のことである．このような経路は拡張 P-label を用いることで等価条件の有無に関わらず，結合操作なしに効率良く処理することができる．以下に，等価条件を含まない接尾辞経路問合せと，等価条件を含む接尾辞経路問合せの例を示す．

//inproceedings/author 等価条件を含まない接尾辞経路問合せの場合，まず拡張 P-label を求める．//inproceedings/author に該当するのは，表 1 より $< 3200000, 3299999 >$ である．したがって，$3200000 \leq xplabel \leq 3299999 >$ を満たすような拡張 P-label の値 $xplabel$ をもつノードを探せばよい．全てのノードが $nodes$ テーブルに格納されているとし，属性 $xplabel$ に拡張 P-label の値を表すとすると，この接尾辞経路問合せは次の SQL を評価すればよい．

```
SELECT * FROM nodes;
WHERE nodes.xplabel >= 3200000
AND nodes.xplabel <= 3299999
```

//inproceedings/author[.='John Doe'] 等価条件を含む接尾辞経路問合せの場合，まず拡張 P-label を付与する際に用いたものと同様のハッシュ関数 $H(x)$ で'John Doe' のハッシュ値を求める．そして，ハッシュ値 H('John Doe') を用いて，//inproceedings/author/ H('John Doe') の拡張 P-label を求める．表 1 より，$< 6320000, 6329999 >$ であることがわかるので，次の SQL を評価すればよい．

```
SELECT * FROM nodes;
WHERE nodes.xplabel >= 6320000
AND nodes.xplabel <= 6329999
```

3.2.2 枝分かれ問合せ

枝分かれ問合せは，$s_1 l_1 p_1 s_2 l_2 p_2 \cdots s_n l_n p_n$ という形で表現できる．ここで，s_i を child 軸 (/) または descendant 軸 (//)，l_i を要素名とする．また，p_i は述語であり，部分枝分かれ問合せを q_i として，$[q_i]$ という形で表される．枝分かれ問合せを処理する場合，まず複数の接尾辞経路問合せに分解する．この際，本稿では Chen らの提案する Push-up アルゴリズム [2] を利用する．これは，分岐点において，そこに至るまでの接尾辞経路を保持した分解処理を行うものである．そして，分解処理後は，各々の接頭辞経路問合せが SQL に変換される．さらに，それらを分解処理中に保持していた先祖子孫関係を用いて 1 つの SQL に統合され，問合せが行われる．

表 2: XPath 問合せ

No.	XPath
QA1	/site/regions/asia/item/name
QA2	/site/regions/asia/item/id[.='item777']
QA3	/site/regions/asia/item[/id='item777']/name
QD1	//inproceedings/tilte
QD2	//author[.='Yoko Maeda']
QD3	//inproceedings[/author='Yoko Maeda']/title

//inproceedings[/author='John Doe']/title
まず，接尾辞経路問合せに分解する．Push-up アルゴリズムを用いると，//inproceedings，//inproceedings/author[.='John Doe']，//inproceedings/title の 3 つに分解される．これらを SQL に変換したときの問合せ結果をそれぞれ *inproc*，*JonDoe*，*title* とすると，先祖子孫関係から 1 つに統合された SQL は以下のようになる．

```
SELECT title.*
FROM inproc, JohnDoe, title
WHERE inproc.start < JohnDoe.start
AND JohnDoe.end < inproc.end
AND iproc.start < title.start
AND title.end < inproc.end
```

表 3: XML データの概要

	Auction	DBLP
サイズ	111MB	268MB
ノード数	2048180	7926463
タグ種類数	77	41
値種類数	405559	2715303
最大深度	12	6

表 4: 属性 *plabel* のデータ型

	P-label	拡張 P-label
Auction	NUMERIC(23, 0)	NUMERIC(33, 0)
DBLP	BIGINT	BIGINT

4 実験と評価

本章では提案する拡張 P-label が従来の P-label を利用した場合よりも等価条件を含む問合せを効率的に処理できるかことを実験により示す．実験では，まず，プロトタイプシステムによって XML データの各ノードを関係データベース上に格納した．そして，それらに対して XPath を SQL に変換して問い合わせることで性能を計測した．

4.1 実験環境

実験対象の XML データとして，Auction[14] および DBLP[3] を用いた．各データの特徴の概要は表 3 の通りである．また，これらを格納するテーブルのスキーマは $(name, start, end, level, plabel, value)$ とした．属性 *plabel* には P-label もしくは拡張 P-label の値が入る．この属性のみ XML データやラベルによってデータ型が異なるため，それらを表 4 に示す．

実験環境は CPU: Pentium III 1.4Ghz Dual，メモリ:2GB，OS: Linux kernel 2.4.18，DBMS: PostgreSQL 7.4 である．

4.2 テーブルサイズ

それぞれの XML データの各ノードを 4.1 節で示したスキーマのテーブルに格納し，テーブルサイズを測定した．従来の P-label がオリジナルの XML データの 2 倍強程度であるのに対し，提案する拡張 P-label ではオリジナルの 4 倍強程度のテーブルサ

表 5: テーブルサイズ

	オリジナル	P-label	拡張 P-label
Auction	111MB	239MB	471MB
DBLP	268MB	724MB	1295MB

イズとなっている．これは，拡張 P-label が P-label よりも大きな区間を使用するため，それに合わせて大きなデータ型を使用しなければならなかったためである．また，3.1 節で述べたように，拡張 P-label では値ノードも 1 つのタプルとして扱っている点も容量を増加させてしまう大きな要因である．

4.3 問合せ処理時間

表 2 の XPath 問合せを用いて処理時間を測定した．XPath 問合せは，Auction データと DBLP データのそれぞれに対して 3 タイプ用意した．1 つ目は接尾辞経路問合せだけで構成されて等価条件を含まないもの，2 つ目は接尾辞経路問合せで構成されて等価条件を含むもの，3 つ目は枝分かれ問合せである．Auction データに対する問合せ処理時間の測定結果を図 3 に，DBLP に対するものを図 4 に示す．図中の問合せ処理時間はそれぞれの問合せを 10 回ずつ行い，平均をとったものである．なお，拡張 P-label を求める際のハッシュテーブルのサイズは Auction データに関しては 257，DBLP データに関しては 467 とした．Auction データの場合，木構造が比較的深く，値を考慮せずとも区間の上限が多倍長整数の範囲を超えてしまうため，適当な素数を選んで実験を行った．一方，DBLP データの場合，多倍長整数の範囲に収まる最大の素数をを用いた．

図から明らかなのは，タイプ 2 の問合せ，すなわち等価条件を含む接尾辞経路問合せにおいて，拡張 P-label を用いることにより大幅にパフォーマンスが向上したことである．QA2 においては，約 20 倍，QD2 においては約 100 倍速くなっている．

一方で，タイプ 1 の問合せ，すなわち等価条件を含まない接尾辞経路問合せに関しては，いずれも従来の P-label を用いた方が高速に問合せを処理できるという結果となった．これは，アクセスするタプル数が同一であっても，拡張 P-label の方がより

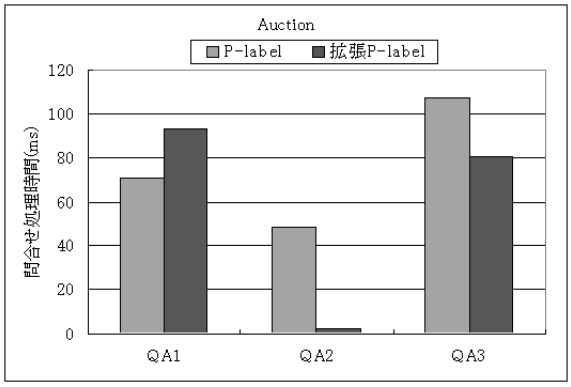

図 3: Auction データに対する問合せ処理時間

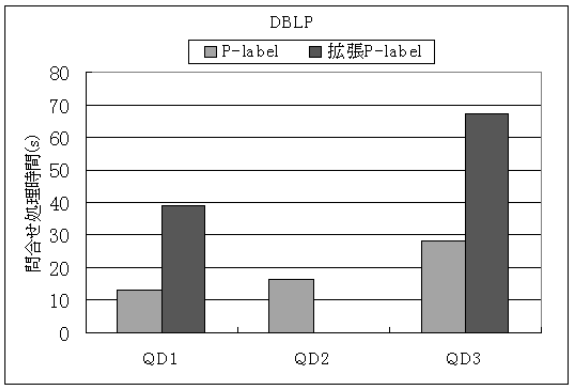

図 4: DBLP データに対する問合せ処理時間

大きな整数区間を用いるため，1 タプルのサイズが大きくなり，結果としてディスク IO が増加してしまったからだと推測される．しかし，DBLP データに関しては，区間を格納するためのデータ型として，P-label，拡張 P-label ともに固定長の多倍長整数を選択しており，パフォーマンス低下の詳細な原因は調査中である．

タイプ 3 の問合せ，すなわち枝分かれ問合せにおいては，Auction データに関しては若干パフォーマンスが向上したものの，DBLP データに関しては 2 倍程度の問合せ処理時間がかかってしまっている．枝分かれ問合せの場合，等価条件を含む部分の問合せ処理結果とそうでない部分の結果を親子関係や先祖子孫関係を用いて結合することになる．そのため，等価条件を高速に処理できても最終的にはそれが相殺されてしまったと考えられる．

5 まとめと今後の課題

本稿では，XML データにおける値ノードの取り扱いに着目し，等価条件を含む問合せを効率良く処理することができる拡張 P-label を提案した．従来の P-label では値ノードは考慮していなかったが，ハッシュ関数を用いることにより値の種類数が膨大な場合でもそれらに対しても区間を割り当てられるように改良した．実験においては，提案する拡張 P-label の性能について検証した．通常の問合せでは場合によってパフォーマンスが低下するものの，等価条件を含む問合せにおいては飛躍的に速度が向上した．

今後の課題として，まず，等価条件を含まない通常の問合せに関しても従来通りの性能を得られるよう改善する必要がある．そのためには，拡張 P-label のために割り当てる区間を可能な限り小さくする必要があると思われる．そこで，XML データ中に実際には現れない接尾辞経路に対しては区間を割り当てないようにすることで，不必要な区間消費を防ぐ手法を検討中である．また，用意する区間のサイズとパフォーマンスの関係を明らかにし，適切なハッシュテーブルのサイズを検討することも課題の 1 つである．さらに，拡張 P-label では値ノードを効率良く扱うことができるため，XQeury における値ベースの結合手法についても検討してゆきたい．

参考文献

[1] N. Bruno, N. Koudas and D. Srivastava. Holistic Twig joins: Optimal XML pattern matching. In *Proceedings of ACM SIGMOD*, pp.310-321, 2002.

[2] Y. Chen, S. B. Davidson and Y. Zheng. BLAS: An Efficient XPath Processing System. In *Proceedings of ACM SIGMOD*, pp.47-58, 2004.

[3] DBLP Computer Science Bibliography. http://dblp.uni-trier.de/

[4] P. F. Dietz. Maintaining order in a linked list. In *Proceedings of ACM STC* pp.122-127, 1982.

[5] T. Grust. Accelerating XPath Location Steps. In *Proceedings of ACM SIGMOD*, pp.109-120, 2002.

[6] D. Megginson. Simple API for XML (SAX). http://www.saxproject.org/.

[7] P. Rao and B. Moon. PRIX: Indexing And Querying XML Using Prufer Sequences. In *Proceedings of IEEE ICDE*, pp.288-299, 2004.

[8] I. Tatarinov, S. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita and C. Zhang. Storing and Quering Ordered XML Using a Relational Database System. In *Proceedings of ACM SIGMOD*, pp.204-215, 2002.

[9] H. Wang and X. Meng. On the Sequencing of Tree Structures for XML Indexing. In *Proceedings of IEEE ICDE*, pp.372-383, 2005.

[10] W3C. Document Object Model (DOM). http://www.w3.org/DOM/.

[11] W3C. Extenshible Markup Language (XML). http://www.w3.org/TR/REC-xml.

[12] W3C. XML Path Language (XPath) Version 1.0. http://www.w3.org/TR/xpath.

[13] W3C. XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery/.

[14] XMark: An XML Benchmark Project. http://monetdb.cwi.nl/xml/index.html