2003－DBS－131　(31)
2003／7／17

半構造データに対するコンパクトな構造索引

洪 仁基†　朴大一†　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

神奈川県横浜市港北区日吉 3-14-1 慶應義塾大学情報工学科遠山研究室

045-563-1141 内線 43244

† hong@db.ics.keio.ac.jp, park@db.ics.keio.ac.jp # toyama@ics.keio.ac.jp

概 要

近年は Web や XML などの半構造データが多く使われている。この半構造データは今までの RDB や OODB などの構造化されたデータとは違いその構造が不規則的である。そのため既存のデータベースシステムではこの半構造データの処理が困難である。スタンフォード大学の Lore システムではクエリの評価のためパスを用いる。パスを評価するためデータ構造の簡略で正確な要約であるグラフ要約が必要となる。しかし、このグラフ要約を生成する際にサイズや時間の問題が生じる。この問題を解決するために、これまでに様々な構造索引が提案されてきた。本研究ではこれまで提案されてきた構造索引の問題点に着目し、索引のサイズを減らしながらクエリに対して正確な答えを得られるコンパクトな構造索引を提案する。

キーワード　：構造索引 、 半構造データ管理システム 、 検索最適化

Compact structure index for semi-structured data

Inki Hong†　daeil Park†　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

3-14-1, Hiyoshi Kouhoku-ku Yokohama-shi, Kanagawa-ken

+81-45-563-1141 ext.43244

†hong@db.ics.keio.ac.jp, park@db.ics.keio.ac.jp # toyama@ics.keio.ac.jp

As for recent the semi-structure data such as Web and XML are used a lot. The structure is irregular unlike the data that this semi-structure data was turned the structure such as RDB and OODB until now. Therefore the processing of this semi-structure data is confusion in the database system of established. it use the path expression due to the evaluation of query in the Lore system of the Stanford University. The graph summary that is an exact summary with the simplification of data structure to evaluate the path becomes needed. However, size and matter of time result when this graph summary is generated. The various structure index has been proposed until now to solve this problem. While decreasing the size of the index that became the problem of the structure index that has been proposed until now in this research I propose the compact structure index that is obtained the exact answer of query.

Key word ： Structure index 、 DBMS of semi-structure data 、 Searching optimization

1 はじめに

近年 XML や Web などの半構造データが多く使われるようになってきた。この半構造データにはスキーマがないのでスキーマの利点を提供するため構造の要約であるグラフ要約を使う。このグラフ要約を通じてユーザはデータの構造を分かるようになり構造を知った上でクエリの実行ができる。これは構造に対して部分的な知識があればパス式で有用に利用できるという利点を持つ

ex . Dinner item は Menu の中に存在する

しかし、ある半構造データから構造要約を生成する際、莫大な時間が掛かり、

生成された構造要約のサイズも元のソースデータと同じくらい大きくなる場合がある。また構造を要約しても全ての情報を含まなければならない。情報が失われると、クエリに対して正しい評価ができなくなるためである。このような問題点を解決するためこれまでに様々な構造索引が提案されてきた。まず 1-index は兄弟のラベルが同じであって同じ親を持つ全てのノードを結合したものである。この場合、すべての情報が含まれてありクエリに対しては正しい評価ができる。しかし索引のサイズが元のデータと同じくらい大きくなる可能性がある。A(k)-index は部分的な構造に基づいた構造索引である。例えば A(0)-index の場合は親子関係なく兄弟だけを見て、ラベルが同じであるものを結合したものである。この場合は索引のサイズは小さくなるが正しいパスの評価ができなくなり、クエリに対する正確な答を得ることができない。

本研究では、上記のような提案されてきた構造索引の問題点となっている索引のサイズとクエリの正確度を考え、索引のサイズを減らしながらも正しい評価ができるようにするための新たな構造索引を提案する。

2 関連研究

2.1 データモデル

XML などの半構造データは入れ子構造のオブジェクトモデルで記述される。Labeled graph G=(V_G, E_G, $root$, Σ_G, $label$, Oid, $value$) は図 1 のように表す。E_Gの各 $edge$ は $object - subobject, object - value$ の関係を表す。V_Gの中で $Simple$ ノードは $outgoing\ egde$ を持たず、そして $value\ function$ を通じて値が与えられる。各ノードは $label\ function$ によりラベルが付けられ、これらは $Oid\ function$ によって区別される。このデータモデルは $sub - object$ を持っている $complex\ object$ と値を持っている $atomic\ object$ で構成されている。

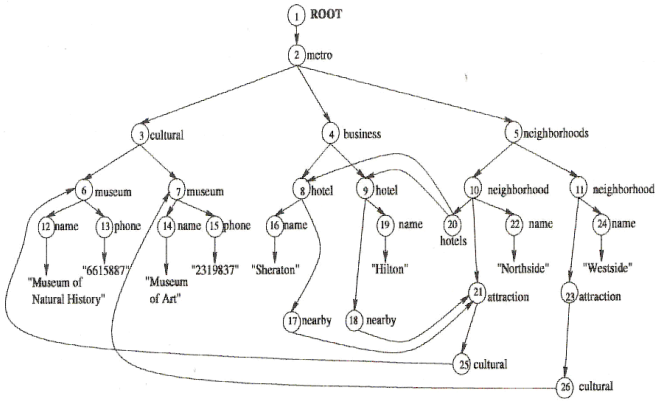

図 1: sample data graph

2.2 1-index

1-index は次のような Bisimilarity に基づいてグラフを要約する。

2.2.1 Bisimilarity

もしある 2 つのデータノード u,v が bisimilarity 関係 $u \approx v$ であれば

- u,v は同じラベルを持つ
- U が u の親であれば v もその親である V がある。そして $U \approx V$ の関係が成立する

1-index の目的は $q \in inst(Px)$ の関係にあるクエリの効率的な計算である。クエリのパスは全てのクラス S_i($extent(S_i)$) を参照して評価される。またクエリの答えは全ての $extent(S_i)$ の $union$ である。この $1-index$ の貯蔵においては要約したグラフに対する空間とその $extents$ に対する空間が必要となる。要約グラフに対しては各ノードに $Oids$ を付けデータベースの各ノード s に対して同等クラス ($extent(s)$) を記述する。もし s が $[v]$ の Oid であれば $extent(s) = [v]$ である。図 1 のサンプルデータグラフから $1 - index$ を生成すると図 2 のようになる。こ

こで例えば $metro.business.hotel$ の名前を探すクエリであるなら、$metro.business.hotel.name$ のパスを持っている $16, 19$ 番のノードに対する $extent$ の $union$ がクエリに対する答えになる。$1-index$ はデータグラフより高くはならないが、悪い場合はデータグラフとあまり変わらないサイズを持つ間合いもある。計算にかかる時間は $O(mlgn)$ である。ここで m は $edge$ の数であり、n はノードの数である。

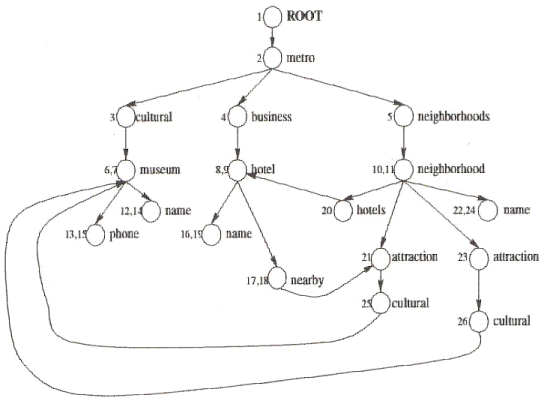

図 2: sample data graph から生成された 1-index

2.3 A(k)-index

次に A(k)-index に関して説明する。1-index はある集合に対して大きくなる可能性がある。この問題点を解決するため提案されたのが A(k)-index である。この A(k)-index は k-bisimilarity という概念に基づいたものである。

2.3.1 k-bisimilarity

2 つのノード u,v に対して

- u,v が同じラベルを持つと $u \approx {}^{0}v$ の関係が成立もし $u \approx {}^{k}-1u$ であり、全ての親ノード u‘に対しての親である v‘が存在する場合　$u' \approx {}^{k}-1v'$が成立する。

A(k)-index の概念は木の深さ k までを考慮し、パスを比較するということである。すなわちグローバルパス情報の変わりにローカル構造に基づいたグループノード、特に k までのパス長さを用いてクエリを評価する。

図 3 を例に挙げて説明する。

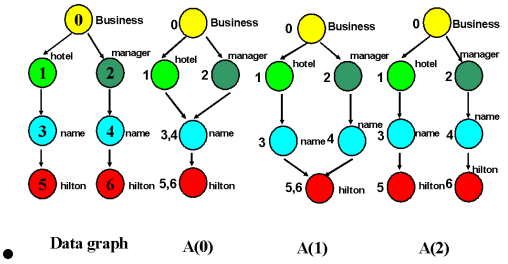

図 3: A(k)-index の流れ

まずある部分的なデータグラフに対して兄弟ごとにラベルが同じであるものを全て結合したものが A(0)-index である。しかしこの場合、name ノードが hotel の name であるのか manager の name であるのか分からなくなる。従って、このノードは分ける必要がある。この場合、A(1)-index となる。3,4 番のノードを分けてその親との関係を考えると正しい評価ができるようになる。同じ理由から、5,6 番の hilton ノードも分ける必要があり、分けたものが A(2)-index となる。この例の部分的なデータグラフから見ると、A(2)-index が元のデータグラフと同じ形になり、また 1-index とも同じ形になる。結局この場合は正しい評価のため k=2 まで考える必要がある。この場合各 A(0)-index,A(1)-index,A(2)-index は 0-bisimilarity,1-bisimilarity,2-bisimilarity に基づくものである。この索引は部分的な構造から検索をするというものであり、もし長くて複雑なパスがユーザから与えられた重要なキーである場合は 1-index に基づいて検索を行う。しかし、このケースはあまりないと考えられ A(k)-index が有効であると考えられる。

3 本研究の概要

今まで提案されてきた構造索引は様々な問題点がある。まず問題点となっているのは索引のサイズである。索引のサイズを減らすとクエリに対する答の正確率は低くなる。つまり、索引のサイズを減らすとパスに対しての正しい評価ができなくなるということである。本研究は上記のような理由、すなわち、索引のサイズを減らすとク

エリに対する答の正確率が低くなるといった問題点を改善するための新たな提案である。現在まで提案されてきた構造索引に中では A(0)-index が最小のサイズである。しかし、パスの正しい評価のためには 1-index が一番良いと考えられる。従って、本研究は 1-index を元にし、索引のサイズを A(0)-index と同等まで減らすための新たな構造索引を提案する。

3.1 Complex object index

構造データは次のような object で構成されている。

- Atomic object : 値を含んでいる object
- Complex object : subobject を含んでいる object

構造要約の目的であるデータの構造をユーザに知らせるという観点から見ると、そのデータ構造だけをユーザに知らせるには Complex object のみを示しても良いと考えられる。従って、索引を Complex object だけで生成し、その索引にある条件を与える時に Atomic object を用いる方法である。XML データの場合、タグの中で実際の値 (integer,real,string,img など) を含んでいるタグに対して、その親のタグは Atomoc object に対するポインタを持つようにし、実際の索引には Atomic object が現れない方法である。また where 句に必要な Atomic object はユーザインタフェースで提供する。この Complex object index と A(0)-index をデータグラフから生成し、比較すると次の図 4，5 のようになる。ここで注目すべきところは、Complex object index は 1-index を元にしたため正しいパスの評価ができるのにもかかわらず、そのサイズは A(0)-index とあまり変わらないという点である。

3.1.1 アルゴリズム

Complex object index の計算アルゴリズムは bisimulation 計算に対する標準アルゴリズムの変形である。まず compute bisimulation の重要な概念を説明する。グラフノードのある集合に対する stability は、あるノード集合 A に対して Succ(A) と表す。これは次の関係である集合を意味する

$\{ v \mid$ *there is a node u in A with an edge from u to v*$\}$

定義 :与えられたデータグラフの 2 つの集合 A,B に対して次の条件を満たすと A は B に対して Stable という

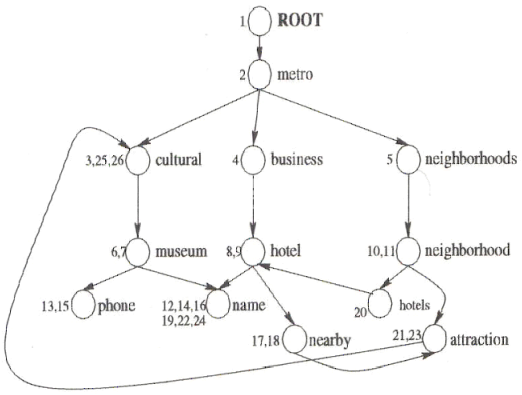

図 4: A(0)-index

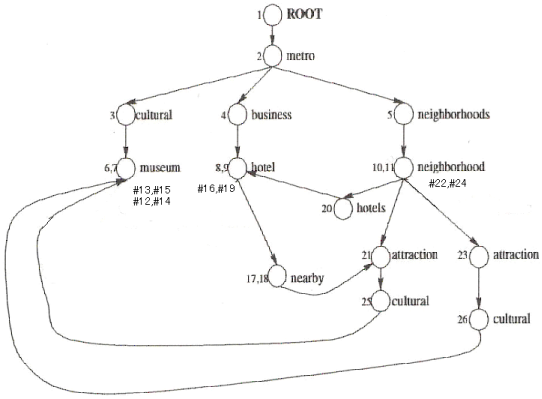

図 5: Complex object index

- A は Succ(B) の subset である
- A と Succ(B) は disjoint である。

もし 2 つのノード（or node set)A,B がある場合、B に対して stable である A を作る。この場合 A $\cap Succ(B)$ と $A-Succ(B)$ によって分割する方法を用いる.まず *bisimulation* の *copy* を生成し、この *copy* から、*bisimulation* の同等クラスに対して *stable* である限り同等クラスを分割する。この接近には $O(nm)$ の時間と $O(m)$ の空間がかかる。ここで m はグラフ G におけるエッジの数である。*bisimulation* を計算すると *compute* $CO-index$ の手順によって *Complex object index* が生成される。

```
Procedure compute bisim(G,n)
begin
1. Q and X are each a list of node-sets
2. Q=partition VG by label
3. X=(a copy of)Q
4. for i=1 to n do
5.   foreach X in X do //stabilize Q w.r.t X
6.     compute Succ(X)
7.     foreach Q in Q do //split
8.       replace Q by Q∩Succ(X) and Q−Succ(X)
9.   if there was no split then
10.    break
11.  X=(a copy of )Q
end
```

```
Procedure compute CO-index(G,n)
begin
1. Compute bisim(G,n)
2. A=nodes that has a value
3. foreach equv.class in bisimulation do
4.   Create an index node I
5.   ext[I]=data nodes in the equiv.class
6. foreach edge from u tov in G do
7.   I[u]=index node containing u
8.   I[v]=index node containing v
9.   if there is no edge from I[u] to I[v] then
10.    add an edge from I[u] to I[v]
11.  for i=1 to n do
12.    foreach I[u],I[v] in I do //delete Atomic object
13.    replace I by I−A
14.    if there was no A then
15.    break
end
```

図 6: Complex object index computation

3.1.2 ソースとグラフ要約の関係

本研究では XML データを用いて実験を行う。図 6 は実際のソースからグラフ要約を生成する時を表している。

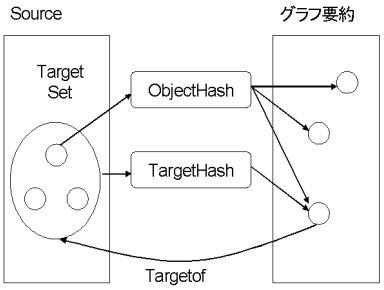

図 7: ソースからグラフ要約を生成

ソースデータのある Target となる集合に対して各ノードは ObjectHash を通じてグラフ要約に表される。またその Target 集合に対しては TargetHash を通じてグラフ要約に表される。TargetHash によって生成された object は Targetof というポインタを持ち、実際のソースの位置を示している。

3.1.3 データ索引ファイル

Complex object index はソースデータに含まれている Atomic object を索引に表示しないという面から索引には含まれないがその Atomic object の管理はしなくてはならない。そのためデータ索引ファイルを生成する。ソースデータから生成された Complex object index はデータの構造をユーザに知らせるのには十分と考えられる。しかし、実際クエリを実行するためには Atomic object の情報を管理する必要がある。図 7 は XML データの例であり、図 8 はその XML ファイルから生成された Complex object index と各 object の情報を格納しているデータ索引ファイルの関係を表している。

```
<Root>
 <metro>
  <culture>
   <museum>
    <name>Museum..</name>
    <phone>6615887</phone>
   </museum>
   <museum>
    <name>museum of..</name>
    <phone>2319837</phone>
   </museum>
  </culture>
  <business>
                ……
</Root>
```

図 8: xml data の例

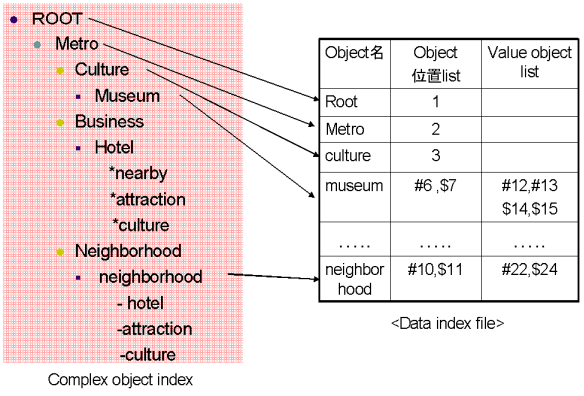

図 9: data index との関係

データ索引ファイルはデータファイルに格納されているタグ名、タグ属性を要素名としてリスト化して、各要素名がデータファイルに格納されている位置と関連付けられている索引ファイルである。構造索引からそのタグの位置は value object 含んでない時は object 位置リストからソースデータへの位置が分かるようになり、もし value object を含んでいる場合は object 位置リスト

と value object リストを union し、ソースデータを検索する。

3.2 パターンマッチを利用した構造索引

この索引は構造データの中で親子関係なくラベルが同じである物を結合する方法である。この場合は索引からパスの評価ができなくなる問題点がある。その問題点を解決するためパスガイドを用いる。

3.2.1 path guide

コンパクトな索引だけではその構造データの構造が分からなくなり、正しいパスの評価ができなくなる問題点がある。それを解決するためにこのパスガイドを用いる。例えばユーザが business.hotel.name といったパスから検索をしたい場合 A(0)-index ではそのパスが分かるようになっていて、そのパスから検索をすることができる。しかし今回提案するこの索引ではそのパスが分からないため、どんなノードからでも検索ができるようにする必要がある。パスガイドを用いて索引の中にある business や hotel,name というノードの内どんなノードからでみ同じパスを得られるようにパタンマッチを利用する。

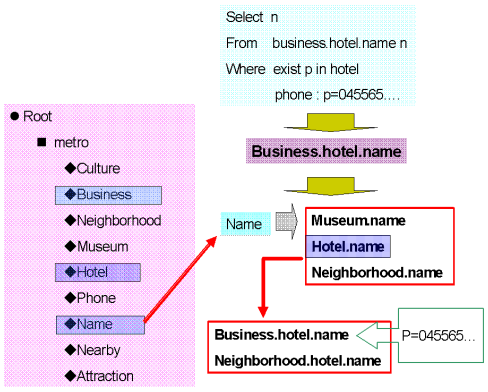

図 10: path guide

XML データの場合、タグのネームが同じであるものを全て結合する方法であって実際の索引のオブジェクトはネームが同じであってもそれらが持っている Oid は親子のオブジェクトが異なる場合を考え別に貯蔵する。検索をする時はそのポインタによってパスをユーザに表す方法である。この場合索引は一番フラットでコンパクトであると考えられるがやはり色々雑音が増えると考えられる。実際この提案はユーザインタフェースに適用するものである。

4 実験評価

OEM データや XMark データを用いて実験を行う予定である。

4.1 実験の目的

- 以前提案された構造索引とのサイズ比較
- 索引を生成する時の時間比較
- クエリに対する正確度や検索時間の比較

4.2 実験計画

本研究の目的は今まで提案されてきた構造索引のサイズを減らすことである。その実験のためにソースデータとなる XML 文書と XML parser, そして SAX クラスを利用して生成された索引のサイズを比較する。今回は Apache Xerces parser とこれに含まれている SAX Class を利用する。XML 文書を分析するため SAX を用い、Swing の JTree を使用して XML 文書のデータを視覚的に見せる。また XMLReader インタフェースを具現したクラスは SAX の Call-back 機能を使用し、XML 文書を処理できる。全ての XML parser はインタフェースを具現したクラスを含んでいて、このクラスのインスタンスを使用して XML 文書を分析する

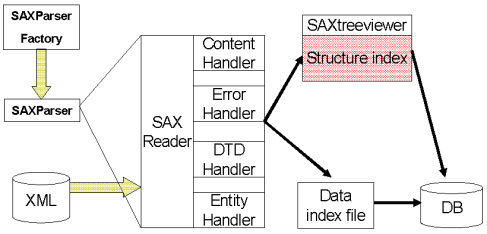

図 11: システム図

A(k)-index の実験結果を説明し、それと比べて本研究の予想的な結果、目標としての結果を説明する。図 1 1 は

A(k)-index に対して Internet Movie database(IMDB) と Open Directory Project(ODP) を用いて索引のサイズに関して実験を行った結果である。２つのデータベースに対して k が６になってから overbound が存在し、これは 1-index と同じ形になるのを意味する。すなわち、これらのデータベースに対しては 1-index は多くても実際のデータの 45％ (ODP の場合) のサイズであり IMDB の場合は 10％未満のサイズを持つのが分かる。特に ODP の場合 A(0)-index と 1-index のサイズの差が多くなっている。本研究で提案した Complex object index の場合は 1-index を元にしてサイズを減らしたものであるためそのサイズは k が０の場合と６の場合の間になると予想できる。

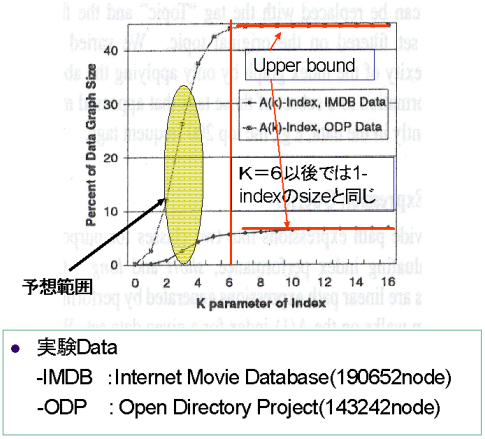

図 12: サイズの比較

図１のデータグラフから生成された A(0)-index,1-index, Complex index のノード数とエッジ数の比較は表１のようである。この場合含まれていた Atomic object は８個であった。そして、データグラフの中に多くの Atomic object を含んでいる、ある XML 文書に対して各ノードとエッジの数を比較した結果は表２のようである。この場合データグラフに含まれていた Atomic object は３３個であった。結果的に表２のデータグラフのノード数は表１のデータグラフの 2.7 倍であるのにもかかわらず、実際生成された Complex object index のサイズはあまり変わらないというのを表している。

	Graph data	A(0)-index	1-index	Complex index
Node	34	13	18	14
Edge	39	16	21	17

図 13: 図 1 のデータグラフに対するサイズ比較

	Graph data	A(0)-index	1-index	Complex index
Node	90	20	27	15
Edge	98	24	29	17

図 14: ある XML 文書に対するサイズ比較

5 まとめ

今回は半構造データにおける構造索引のサイズを減らすのが研究の目的である。今まで提案されてきた構造索引の中で 1-index,A(k)-index の良い所を利用し最小の構造索引から正しい検索結果を出せるための研究であった。これらを利用するためには新たな DBMS の貯蔵管理、クエリ処理、最適化、ユーザインタフェースなどが必要となる。特に実際この索引の良さを示すためのユーザインタフェースが必要だと考えられる。提案した構造索引の内 Complex objecdt index はソースとなるデータに Atomic object が多く含まれている時その効果が期待される。またパタンマッチを利用した索引はユーザインタフェースに必要な構造要約を生成する時掛かる時間を減らすのにその効果が期待される。しかし、これらの構造索引は生成する時のサイズ的の利点を提供するだけであって実際その情報を管理するデータ索引ファイルは逆に大きくなる恐れがある。

6 今後の課題

本研究は半構造データに対する構造要約の必要性を説明し、そのため提案されてきたいくつの構造索引を紹介した。これらの交通的な問題点は索引を生成する時掛かる時間や索引のサイズが大きくなるということだった。本研究は索引のサイズを減らすための研究であって、ク

エリの最適化などは既存の 1-index を元にしたため正しい結果を出せると仮定した。今後の課題としてはこの Complex object index に対するクエリ最適化やパスガイドを利用したユーザインタフェースに関する研究をする予定である。ただ、パスガイドを用いるとパタンマッチのため全ての連結リストが必要となり、実際雑音の方が多くなると考えられる。この問題点を解決するアルゴリズムを提案したいと思っている。

参考文献

[1] T.Milo,D.Susiu. Index Structures for Path Expressions.In ICDT : 7th International Conference on Database Theory,1999

[2] R.Kaushik,P.Bohannon,J.F.Naughton and H.F. Korth. Covering indexes for branching path queries. In Proceedings of SIGMOD, 2002

[3] R.Kaushik,P.Shenoy,P.Bohannon,and E.Gudes. Exploiting local similarity for efficient indexing of paths in graph structured data.in Procedings of ICDE,2002

[4] J.McHugh, S.Abiteboul, R.Goldman, D.Quass, and J.Windom. Lore : A Database Management System for Semistructured Data. SIGMOD 1997 Data. SIGMOD 1997

[5] R.Goldman and J.Widom. DataGuide : Enabling Query Formulation and Optimization in Semistructured Database. VLDB 1997

[6] J.McHugh,J.Windom,S.Abiteboul,Q.Luo and A. Rajaraman. Indexing Semistructured Data. Technical Report, 1998

[7] Raghav Kaushik, Philip Bohannon, Jeffrey F Naughton, Pradeep Shenoy. Updates for StructureIndexes.Proceeding of the 28th VLDB Conference,HongKong,China,2002

[8] Masakazu Hattori, Katuhiko Nonomura, Takuya Kanawa, Naomichi Sueda. Retrieval optimization technique using a query graph in XML database.TOD16 Vol.43.2002

[9] R.Goldman and J.Widom. Appriximate Dataguides. In proc. ofthe Workshop on Query Processing for Semistructured DAta and Non-Standard Data Formats. 1999

[10] Jason McHugh,Jennifer Widom,Serge Abiteboul,Qings han Luo,Anand Rajaraman. Indexing Semistructured Data.