データベースシステム　122－30
(2000. 7. 27)

半構造データモデルによる画像処理履歴の問合せの性能評価

田幡 勝，有次 正義，金森 吉成

群馬大学工学部情報工学科
〒 376-8515 桐生市天神町 1-5-1
tel. 0277-30-1852　fax. 0277-30-1801
email: {tabata,aritsugi,kanamori}@dbms.cs.gunma-u.ac.jp

あらまし　画像処理の詳しい知識を持たないユーザは画像処理演算の実行順序を決定したり適切なパラメータを指定することができない．このため，我々は画像処理の専門家が作成した画像処理演算の実行手順を再利用する方法を提案してきた．この実行手順を画像処理履歴と呼び，半構造データモデルを用いて表現する．画像処理履歴の表現方法とそれらの履歴を問合せる方法を示す．正規表現やワイルドカードを含んだ一般パス表現は画像処理履歴の問合せに柔軟性を提供するが，その処理には時間がかかる．一般パス表現を用いて画像処理履歴を効率的に問合せるため4種のインデックスについて検討し，それらのインデックスを用いる問合せ演算を試作した．そして，それらの演算による問合せの実験結果を述べる．

キーワード　半構造データ，画像処理履歴，インデックス，問合せ処理，性能評価

Performance evaluation of querying histories of image processing with a semistructured data model

Masaru Tabata, Masayoshi Aritsugi, and Yoshinari Kanamori

Department of Computer Science, Gunma University
1-5-1 Tenjin-cho, Kiryu 376-8515, Japan
tel. +81-277-30-1852　fax. +81-277-1801
email: {tabata,aritsugi,kanamori}@dbms.cs.gunma-u.ac.jp

Abstract　Users who are not familiar with image processing are not able to determine a correct sequence for executing image processing operations or to assign appropriate parameters to the operations. We have therefore proposed a method for reusing execution plans for image processing operations designed by image processing specialists. We call these plans "histories of image processing," and represent them using a semistructured data model. We show a way of representing and querying the histories. We demonstrated that generalized path expressions which include wild cards and/or regular expressions provide flexibilities for querying histories. However, executions for these queries become inefficient. We therefore examine four indexes for querying efficiently histories with generalized path expressions and prototype four query operators which use one of the indexes respectively. We also describe experimental results of queries with the operators.

key words　semistructured data, histories of image processing, index, query processing, performance evaluattion

1 はじめに

画像処理の詳しい知識を持たないユーザは画像処理演算の実行順序を決定したり適切なパラメータを指定することができない．このため，我々は画像データベースから検索した画像に対して一連の画像処理を実行したときに生じる結果を画像オブジェクトの版としてデータベースで管理し，同時に画像処理の手順に関する情報も管理するモデルを提案し，その機構を実現した [7]．ここで，画像処理演算の順序およびそれらの画像処理演算で用いられたパラメータ値のことを画像処理履歴と呼ぶことにする．複数のユーザが画像処理履歴のデータベースを共有することで，過去に作成された一連の画像処理演算の実行手順を再利用することができる．

そこで，画像処理履歴をどのようなデータモデルで表現し，問合せを実現するかが課題となる．一般に，画像処理履歴に含まれる

- 実行された画像処理演算の種類および個数．
- 実行された画像処理演算の組合せ．
- 個々の画像処理演算が必要とするパラメータの型および個数．

などは履歴毎に異なるものとなる．したがって，画像処理履歴は不規則なデータであり，あらかじめ画像処理履歴に対するスキーマを定義することは困難である．半構造データに対する厳密な定義は存在しておらず数多くのデータ群が半構造データとして定義されているが [9]，画像処理履歴のような「構造が不規則なデータの集まりではっきりしたスキーマが定義できないようなデータ群」 [9] も半構造データであるとみなすことができる．

そこで，我々は画像処理履歴の検索機構を実現するため，半構造データモデル OEM(Object Exchange Model)[1] と OEM に対する半構造データベースシステム Lore を利用した画像処理履歴の表現方法を示した [8]．これらを利用した理由は，このデータモデルがグラフ構造で表現されるため画像処理履歴の表現に適しているからである．このモデルにおいて画像処理履歴はエッジとして画像処理演算，ノードとしてその画像処理演算の実行前後の画像を保持する有向グラフによって表現される．

画像処理履歴を検索するとき，その全ての構造を正確に指定することは一般に困難である．半構造データモデルは従来のデータモデルよりデータ型の指定が緩やかであるため，半構造データに対する問合せ言語は従来の SQL や OQL のような問合せ言語よりも柔軟なものとなる．この特徴の一つに正規表現とワイルドカードを用いたパス表現 (path expression) の記述が挙げられる．正規表現とワイルドカードを用いた一般パス表現 (generalized path expression) を用いることによって，画像処理手順の一部分を指定するだけで画像処理履歴が検索できるようになる．

しかし，一般パス表現を含む問合せは表現能力が高い一方で，問合せ処理の効率は悪くなる．このため，従来より一般パス表現の書換えに基づいた最適化手法 [2, 3, 4] が研究されてきた．我々の研究では，画像処理の実行手順をグラフとして表現するため，従来の XML 等の構造化文書を半構造データモデルで表現する場合と比較すると，画像処理履歴を表すグラフは深くなりエッジに割当てられるラベルの種類が多くなる傾向がある．このようなグラフに対して一般パス表現の書換えは適さない．そこで，本研究では一般パス表現を含んだ画像処理履歴の問合せを効率的に実行するためのインデックスを検討し，Lore の問合せ処理に対してそれらのインデックスを用いる問合せ演算を試作した．そして，それらの演算を用いたときの画像処理履歴に対する問合せの実験結果を述べる．

2 OEM による画像処理履歴の表現

画像処理履歴を OEM に従って表現する方法を示す．OEM は，オブジェクトとオブジェクト識別子 (oid) の概念を基本とし，各データは int，string 等の atomic 型またはオブジェクトリファレンスの集合である complex 型のいずれかとなる．complex 型のオブジェクトをノード，atomic 型のオブジェクトをリーフ，オブジェクトリファレンスをエッジとするグラフが構成される．オブジェクトリファレンスは，(ラベル，参照先のオブジェクトの oid) の対として記述される．スキーマ情報は予め定義されるのではなく，各オブジェクトの型を表すラベルとそのオブジェクトが保持する属性のラベルによって記述される．したがって，画像処理履歴を表すラベルと各オブジェクトの属性を定義することが必要となる．

まず，ラベルの定義から説明する．図 1では，oid &23 のオブジェクトは，二つのオブジェクトリファレンス (parameters, &24) と (ip_d, &27) を持つ complex 型のオブジェクトであり，&23 から画像処理演算 ip_d によって&27 を作成したことが表されている．また，oid &25 のオブジェクトは，値 0.8 を持つ real 型の atomic オブジェクトである．画像処理履歴は，以下に示すラベルによって構成される．

HDB 画像処理履歴データベースのエントリポイントを表すラベル．図 1では&1 がエントリポイントである．

*input_*〈入力画像数〉 画像処理履歴の出発点およびその画像処理履歴を作成するのに必要な入力画像の枚数を表す．図 1では，&2 が HIPO-A が保持する画像処理履歴の先頭を示している．

*ip_*〈画像処理オブジェクト名〉 画像処理演算はオブジェクトとして実現されている．このオブジェクトを画像処理オブジェクト (Image Processing Object, IPO) [7] と呼び，個々の IPO は単一の画像処理演算に対応する．そこで，ある画像処理演算によってある画像から他の画像が作成されたことを，その画像処理オブジェクトの名前で表現する．例えば，画像処理オブジェクト a を用いて&2 の表す画像から&7 の表す画像が作成されたことを，それらの間のラベル ip_a で表現されている． 画像処理オブジェクト名には，そのラベルが画像処理オブジェクト名を表すことを意味する接頭語として必ず

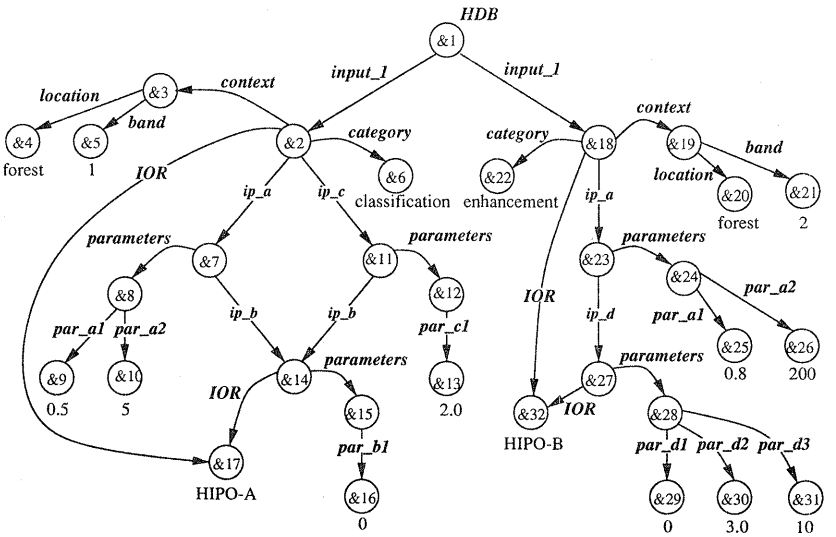

図 1: 画像処理履歴の OEM による表現

“ip_” を付加させる．また，画像処理前後の画像は 1 対 1 に対応するとは限らず，複数枚の画像を入力として，それらを融合する場合もある．例えば，ip_b は 2 枚の画像&7 と&11 を入力とし&14 を出力としている．

parameters および〈パラメータ名〉 *parameters* は，画像処理演算に用いたパラメータの集合を表す．また，個々のパラメータを表すラベルは，そのパラメータの意味を表す文字列で表現する．例えば，図 1では，&2 から&11 への画像処理演算 ip_c のパラメータの集合として&12 を用いていることを表している．この画像処理演算は一つのパラメータだけをとり，そのパラメータ par_c1 に用いられた値が 2.0 であったことを&13 が表している．

IOR 画像処理履歴は，それぞれのユーザが管理する版データベースに画像処理履歴オブジェクト (Histories of Image Processing Objects, HIPOs) として保存されている [7]．*IOR* は，それらの HIPO の識別子を表している．版データベースに格納されている HIPO は，CORBA オブジェクトとして実装されており，その識別子として IOR(Interoperable Object Reference)[6] を持っている．IOR は 16 進数に符号化した文字列に変換され，ラベル *IOR* の指すオブジェクトの属性値として保持される．ただし，図 1では，図を単純化するために&17 の持つ値は単に HIPO-A としている．*IOR* は，実行した画像処理演算の先頭と末尾のノードだけに保持される．図 1では，HIPO-A に含まれる画像処理演算列の先頭と末尾はそれぞれ&2，&14 であり，この二つのノードが&17 を参照するエッジ *IOR* を保持する．末尾のノードが保持するラベル *IOR* は，画像処理履歴が終端したことも表現している．画像処理手順を表す他のノード，例えば&7 や&11 からもラベル *IOR* が&17 を参照するようにした場合，その履歴が終端したときの画像処理名を求めることができなくなるため，他のノードは *IOR* を保持できない．ただし，先頭のノードが保持するラベル *IOR* は，*IOR* の値を直接参照するために必要である．

context 最初の版として用いた画像データに関する情報．どのような画像を対象とした画像処理履歴であるかを判別するために用いられる．コンテキストは，画像処理履歴を作成したときに用いられた画像データオブジェクト [7] の属性値として保持されている．図 1は単純なコンテキストの例であり，*context* は撮影場所と撮影したセンサのバンド番号から構成されている．

category 画像処理履歴の大まかな分類を示す属性．どのような目的を持った画像処理履歴であるかを判別するために用いられる．カテゴリは，画像処理履歴を作成したユーザによって与えられる．この例では，HIPO-A のカテゴリは属性値として土地被覆区分を表す値 classification を保持している．

どの画像処理演算が実行されるかは，画像処理履歴毎に決定される．したがって，図 1に示すような構造は，画像処理履歴毎に全く違ったものとなる．このように画像処理履歴は不規則なため，あらかじめスキーマを定義することができない．このため，画像処理履歴の表現には半構造データモデルが適している．

画像処理履歴を表現するためにはラベル名の定義の他に，上記のラベルが表すオブジェクトが保持する属性も定義する必要がある．この定義の一覧を表 1に示す．この表では，c, a はそれぞれ complex 型，atomic 型を意味し，atomic 型の場合は，そのオブジェクトが保持する属性の型を示している．complex 型の場合は，そのオブジェクトが属性として

表 1: オブジェクトの属性

オブジェクト	型	属性
HDB	c	*input_1,...,input_n*
input_n	c	*category*, {*context*}?, IOR, {*ip_(...)*}+, {*component*}*
ip_(...)	c	{*parameters*}?, {*ip_(...)*}*,{IOR}?
parameters	c	{⟨パラメータ名⟩}+
⟨パラメータ名⟩	a	**string** | **int** | **real** | ...
IOR	a	**string**
context	c	アプリケーション毎に定義
category	a	**string**
component	c	*root* と同上

保持するラベルの集合を示している．また，記号?，*，+はそれぞれ任意選択のオプション，0個以上，1個以上を意味する．以上のラベル名の定義と各オブジェクトが保持する属性の定義によって，画像処理履歴を表すグラフの構造が表現される．より複雑な複数枚の画像を処理したときの表現方法 [8] は本稿では省略する．

3 画像処理履歴の問合せ

Lore の問合せ言語である Lorel[1] による画像処理履歴の問合せ例を図 1に示すデータベースを用いて説明する．これらの問合せによって画像処理履歴の検索がどのように実現されるかを示す．

まず，単純パス表現 (Simple Path Expressions) を用いた画像処理履歴の問合せ例を以下に示す．単純パス表現は，$l_1, ..., l_n$ がラベルであり，Z がオブジェクト名またはオブジェクトを示している変数であるとき，列 $Z.l_1....l_n$ として表現される [1].

[問合せ 1]

```
Q: select N
   from HDB.input_1.ip_a.ip_d.IOR N
A: IOR HIPO-B
```

問合せ 1は，from 節で input_1 から ip_a，ip_d の順に画像処理を実行し，かつ ip_d で終了した処理履歴を指定している．したがって，この問合せの結果 HIPO-B が返される．このように，画像処理の実行順序そのものをパス表現で指定することが可能である．この問合せ 1の例は，画像処理履歴に含まれる画像処理演算の実行順序を正確に指定している．しかし，例えば，二つの画像処理演算のうちどちらを用いれば良いかが分からないときや，最初に実行する画像処理演算は分かっているがその後処理として必要な画像処理演算を知らないとき，画像処理履歴に含まれる正確な画像処理演算の実行順序を指定することはできない．このような場合，正規表現とワイルドカードを用いることによって，画像処理履歴の一部分だけを指定することができる．この正規表現やワイルドカードを含んだパス表現を一般パス表現 (Generalized Path Expressions, GPEs)[1] と呼ぶ．画像処理履歴の問合せはより強力な表現力を持つことができる．まず，正規表現を用いた問合せ例として以下のものを示す．

[問合せ 2]

```
Q: select N
   from HDB.input_1.ip_a(.ip_b|.ip_d).IOR N
A: IOR HIPO-A
   IOR HIPO-B
```

from 節で用いられている正規表現(.ip_b|.ip_d) によって，ip_b または ip_d を含んだパスが指定される．したがって，この問合せは最初に ip_a を実行し，その後 ip_b または ip_d を実行しかつ終了した画像処理履歴を求める．

全てのラベルを知らない場合や正確な順番を知らないといった場合がある．このため，ワイルドカードの概念を持つことも重要である．ワイルドカードは，次の二つが定義されている．% は 0 以上の任意の長さの文字列に，# は 0 以上の任意の長さのパスに一致するワイルドカードである．

パス表現を用いて画像処理履歴を指定する場合，ワイルドカードを用いて履歴のある一部分の知識だけを指定することが多い．そこで，画像処理演算名には接頭語として ip_を付加させている．接頭語を用いることで，以下のように正確に画像処理名だけと一致するパスを指定することが可能になる．

[問合せ 3]

```
Q: select N
   from HDB.input_1(.ip_%)*.ip_d.IOR N
A: IOR HIPO-B
```

* は 0 回以上の繰り返しを表す正規表現であるから，この表現は input_1 と ip_d の間に存在する画像処理演算名のパスと一致する．このような問合せの表現は，最後に実行する画像処理演算は分かっているが，その前にどのような画像処理演算を実行すればよいかが分からない場合に有効である．同様に，ある画像処理演算で開始する画像処理履歴やある画像処理演算を含んだ画像処理履歴も指定することができる．以上のように，ワイルドカードと正規表現を用いることによって画像処理履歴に含まれる画像処理演算の実行順序の一部分だけを指定した問合せができる．

さらに，画像処理履歴の構造に関する問合せを記述することもできる．以下の問合せは，input_1 と IOR の間のパスを求めるものである．

[問合せ 4]

```
Q: select distinct pathof(P)
   from HDB.input_1(.ip_%)+@P.IOR
A: path ip_a ip_b
   path ip_c ip_b
   path ip_a ip_d
```

from 節で 1 回以上の出現を表す正規表現+ を用いて，input_1 から IOR の間に存在する任意の長さの画像処理演算名の列にパス変数 P を割り当てている．pathof 関数はパス変数に割当てられたパスを返すため，この問合せの結果は上記の集合となる．

以上のように画像処理履歴の構造を検索することによって，ある画像データを作成したときの画像処理演算の組合せやそれらに用いたパラメータを知ることができる．これは，他のユーザによって作成された画像処理履歴を容易に理解する助けとなる．

4 一般パス表現に対するインデックス

オブジェクトを o_i，ラベルを l_i と表したとき，あるオブジェクト o_0 からオブジェクト o_n の間に存在するオブジェクトとラベルの列 $o_0, l_1, o_1, \ldots, l_n, o_n$ をデータパスと呼ぶ．o_0 をそのデータパスの祖先，o_n を子孫と呼ぶこととする．特に $n = 1$ の場合，祖先と子孫をそれぞれ親および子と呼ぶことにする．

Lore は Lindex,Bindex,Vindex,Pindex(DataGuide) などのインデックスを提供している [5] が，ここでは一般パス表現に対応する演算にのみ着目する．なお，Lore の query optimizer における問合せプランの作成および問合せの実行については文献 [5] に詳細に記述されており，本稿では省略する．Lore の query optimizer は，一般パス表現に対応する物理問合せプランのノードに対して Scan 演算または Lindex 演算を割当てる．これらの演算は物理問合せプランを実行するときに次のように働く．

- Scan (x, l, y) 演算

  親 x からラベル l を持つエッジで到達する全ての子 y を求める．

- Lindex (x, l, y) 演算

  子 y からラベル l を持つエッジで到達する全ての親 x を求める．これは，Scan 演算の逆向きの演算であり，ボトムアップ方式でグラフを探索するときに用いられる．

Scan 演算は，単にオブジェクト間のポインタをたどる．Lindex 演算はキーとして (子の OID，ラベル ID) の組，データエントリとして親の OID で構成される extendible hashing を用いることで，逆向きのポインタをたどる．Scan 演算と Lindex 演算は，単に与えられた一般パス表現に含まれるラベルを持つエッジを繰返し走査するだけであり，一般パス表現に対する最適化は考慮されていない．そこで，我々はこの二つの演算に対応した一般パス表現を考慮したインデックスを考える．

まず，Scan 演算および Lindex 演算をラベルの代わりに一般パス表現 gpe を指定できるように拡張したものを示す．

- Generalized Scan (x, gpe, y) 演算

  祖先 x から一般パス表現 gpe で到達する子孫の集合 y を求める．

- Generalized Lindex (x, gpe, y) 演算

  子孫 y から一般パス表現 gpe で到達する祖先の集合 x を求める．

さらに，例えば問合せ内に`A X, X(.ip_%)*.Y, Y.B Z` なるパス表現が存在したとき，変数 Y は変数 X から一般パス表現`(.ip_%)*` によって到達するものに束縛されるが，さらに変数 Y はラベル B を出力辺として持つものに制限される．そこで，`(.ip_%)*` に続くエッジのラベル B もインデックスを探索するときに指定するならば，B を出力辺のラベルとして持たないオブジェクトを探索から除外することができる．逆向きに`(.ip_%)*` を探索する場合，同様にラベル A をインデックスを探索するときに指定することができる．したがって，上記のインデックスを以下のように拡張できる．

- Extended Generalized Scan (x, gpe, y, L) 演算

  祖先 x から一般パス表現 gpe で到達する子孫の集合 y を求める．ただし，y はラベル集合 L に含まれる全てのラベル l を出力辺として持つ complex object である．また，任意のラベルを表す ANY が l として指定された場合，y は祖先 x から一般パス表現 gpe で到達する atomic object を含む任意の子孫の集合である．例えば，このインデックスは`A(.ip_%)* X, X.ip_b B, X.ip_c C` のように一般パス表現の先が分岐する場合，あるオブジェクト x から一般パス表現`(.ip_%)*` の先にラベル集合{ip_b, ip_c} を持つオブジェクトを求める．

- Extended Generalized Lindex (x, gpe, y, l) 演算

  子孫 y から一般パス表現 gpe で到達する祖先の集合 x を求める．ただし，x はラベル l を入力辺として持つ complex object である．また，l に ANY が指定された場合，x は子孫 y から一般パス表現 gpe で到達する任意の祖先の集合である．オブジェクト x に複数の入力辺がある場合，問合せ処理ではこれらは全く別のパス表現として扱われるため，Extended Generalized Scan 演算のように入力辺のラベル集合について考慮しない．

これらのインデックスは extendible hashing を用いて作成している．図 1のデータベースにおいて一般パス表現`(.ip_%)*` に対する Generalized Scan 演算および Extended Generalized Scan 演算のためのインデックスを作成する例を示す．それぞれの演算に対応するハッシュテーブルのキーおよびデータを表 2，表 3に示す．例えば，Generalized Scan 演算を用いた場合，オブジェクト&2 から一般パス表現`(.ip_%)*` で到達するオブジェクトは&7，&11 および&14 であることが分かる．また，Extended Generalized Scan 演算を用いた場合，オブジェクト&2 から一般パス表現`(.ip_%)*` で到達しラベル ip_b を出力辺として持つオブジェクトは&7 と&11 であることが分かる．また，Extended Generalized Scan 演算で用いられているラベル ANY は，問合せ内でその一般パス表現の先にさらにパス表現が指定されなかった場合，例えば，問合せ中に指定されたパス表現が`A(.ip_%)* N` のような場合に用いられる特殊なラベルである．`(.ip_%)*` の正規表現演算 * が 0 回繰返しをする場合，子孫の OID は祖先の OID をそのまま返せばよいためインデックスには保存しない．

また，3節の問合せ 4のように構造に関する問合せを行う場合，一般パス表現の前後の祖先と子孫の OID だけでなく，その一般パス表現に一致するデータパスも必要であ

表 2: Generalized Scan 演算に対するハッシュテーブル

キー		データ	
祖先 x	GPE ID	子孫 y	データパス ID
2	(.ip_%)*	7	1
2	(.ip_%)*	14	2
2	(.ip_%)*	11	3
2	(.ip_%)*	14	4
7	(.ip_%)*	14	5
11	(.ip_%)*	14	6
18	(.ip_%)*	23	7
18	(.ip_%)*	27	8
23	(.ip_%)*	27	9

表 3: Extended Generalized Scan 演算に対するハッシュテーブル

キー			データ	
祖先 x	GPE ID	ラベル ID	子孫 y	データパス ID
2	(.ip_%)*	parameters	7	1
2	(.ip_%)*	ip_b	7	1
2	(.ip_%)*	ANY	7	1
2	(.ip_%)*	parameters	14	2
2	(.ip_%)*	IOR	14	2
2	(.ip_%)*	ANY	14	2
2	(.ip_%)*	parameters	11	3
2	(.ip_%)*	ip_b	11	3
2	(.ip_%)*	ANY	11	3
2	(.ip_%)*	parameters	14	4
2	(.ip_%)*	IOR	14	4
2	(.ip_%)*	ANY	14	4
7	(.ip_%)*	parameters	14	5
7	(.ip_%)*	IOR	14	5
7	(.ip_%)*	ANY	14	5
11	(.ip_%)*	parameters	14	6
11	(.ip_%)*	IOR	14	6
11	(.ip_%)*	ANY	14	6
18	(.ip_%)*	parameters	23	7
18	(.ip_%)*	ip_d	23	7
18	(.ip_%)*	ANY	23	7
18	(.ip_%)*	parameters	27	8
18	(.ip_%)*	IOR	27	8
18	(.ip_%)*	ANY	27	8
23	(.ip_%)*	parameters	27	9
23	(.ip_%)*	IOR	27	9
23	(.ip_%)*	ANY	27	9

表 4: データパス

データパス ID	長さ	OID, ラベル	前方データパス ID
1	1	(7,ip_a)	NULL
2	1	(14,ip_b)	1
3	1	(11,ip_c)	NULL
4	1	(14,ip_b)	3
5	1	(14,ip_b)	NULL
6	1	(14,ip_b)	NULL
7	1	(23,ip_a)	NULL
8	1	(27,ip_d)	7
9	1	(27,ip_d)	NULL

る．そこで，祖先と子孫の間のデータパスを表 4のように管理する．データパスの管理に必要な容量を削減するため，一般パス表現に一致するデータパスの一部分が以前に一致したことのある場合，その以前に一致したデータパスの ID を前方データパス ID に記録する．例えば，一般パス表現(.ip_%)* は&2 と&14 の間のパスと一致するが，これは&2 と&7 とも一致している．したがって，&2 から&14 へのデータパスは (OID, ラベル ID)=(14,ip_b) と前方データパス ID=1 によって表現できる．このようにデータパスを管理することで，Generalized Scan 演算および Extended Generalized Scan 演算から求められるデータパス ID から一般パス表現に一致するデータパスを求めることができる．

Extended Generalized Scan 演算は一致する一般パス表現のさらに先のエッジに付けられたラベルもキーとするため，ハッシュテーブルのサイズが大きくなる．例えば，画像処理演算が 100 種類存在し，一つの画像処理履歴が平均 10 回の画像処理演算を実行して作成された履歴を 500 個含んだデータベース (約 1317K バイト) に対し Generalized Scan 演算と Generalized Lindex 演算を作成した場合と Extended Generalized Scan 演算と Extended Generalized Lindx を作成した場合では，前者は約 5460K バイト，後者は約 10757K バイトのディスク容量をそれぞれ必要としている．インデックスの容量の効率化が今後の課題である．

Generalized Lindex 演算および Extended Generalized Lindex 演算も同様に extendible hashing を用いて実装される．

5 問合せの性能評価

実験には画像処理履歴作成用のアルゴリズムによって作成した画像処理履歴を用いる．履歴を作成するときに設定した主なパラメータを以下に挙げる．

- 画像処理演算の種類．20 種類または 100 種類．
- 画像処理演算名の長さの最小値 5，最大値 10.
- 画像処理演算の引数の個数の最小数 0，最大数 5.
- 画像処理演算の引数名の長さの最小値 1，最大値 10.
- ある画像処理履歴に含まれる画像処理演算の最小数 2，最大数 (8 ～ 28).
- あるノードから 1 回だけ画像処理演算を実行する確率 =0.7，2 回実行する確率 =0.2，3 回実行する確率 =0.1
- 二つの画像を結合する画像処理を実行する確率 =0.2

また，画像処理履歴を作成するとき，どの画像処理演算が選択されるかを決定する選択率をそれぞれの演算毎に設定している．

以上の設定によって作成したデータベースに対して，任意の 2 種類の画像処理演算 ip_a および ip_b を選択して以下の問合せを実行する．この問合せは，画像処理演算 ip_a の後に画像処理演算 ip_b を実行した画像処理履歴を求める．

[問合せ 5]

```
select N
from HDB.input_1 X,
     X(.ip_%)*.ip_a(.ip_%)*.ip_b, X.IOR N;
```

データベースに存在する画像処理演算名の中からランダムに100通りの組合せを選択し，この問合せ中の2種類の画像処理演算の部分を指定する．実験結果は，これら100回の問合せを実行したときの平均時間である．

上記の問合わせに対して以下に示す3種類のインデックスの設定を行い，それぞれの設定における問合せの実行時間を測定した．

(1) Scan, Lindexおよび Bindexを用いた方法.

(2) (.ip_%)*に対しGeneralized Scanと Generalized Lindexを(1)に追加した方法.

(3) (.ip_%)*に対する Extended Generalized Scanと Extended Generalized Lindexを(1)に追加した方法.

方法2，方法3では query optimizerが一般パス表現に対して Scan演算または Lindex演算を割当てる代わりに (Extended) Generalized Scan演算，(Extended) Generalized Lindex演算を割当てている．この実験に用いた計算機環境を表5に示す．

表 5: 実験環境

CPU	Pentium III 600MHz
OS	LASER5 LINUX 6.0
メモリ	256 MB
2次記憶	IBM DNES-309170W (9.1GB)
コンパイラ	gcc,g++(egcs-2.91.66)
DBMS	Lore 3.3

まず，画像処理履歴が平均で10回 (最小2回，最大18回) の画像処理演算によって作成された場合について，画像処理履歴の個数を50個から500個まで変化させたときの問合せの実行時間を図2に示す．図2において横軸は画像処理履歴の個数，縦軸は問合せの実行時間を示している．方法 (3)は従来の方法 (1)と比較すると速度の向上が見られる．これは Extended Generalized Scanおよび Extended Generalized Lindexを用いることによって探索する必要のあるグラフ内の空間を大きく減少させることができるためである．また，明らかに方法 (2)は他の二つの手法と比べて遅くなっている．方法 (2)で探索する必要のあるグラフ内の空間は方法 (1)と全く同じであり，extendible hashingを探索するコストが大きなためである．

次に，画像処理履歴の個数を500個に固定し，それぞれの履歴を作成するときの画像処理演算の平均数を5から15まで変化させた場合について図3に示す．方法 (2)は明らかに遅いため除外した．横軸は各画像処理履歴に含まれる平均の画像処理演算数，縦軸は問合せの実行時間を示している．方

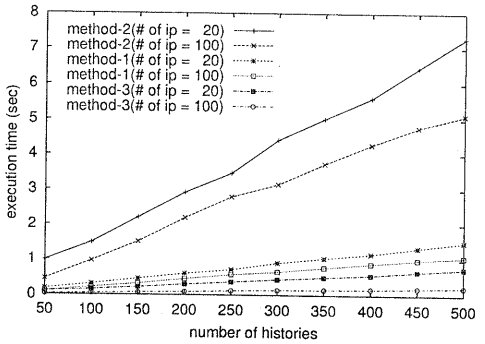

図 2: 平均10回の画像処理演算を実行した画像処理履歴に対する検索時間

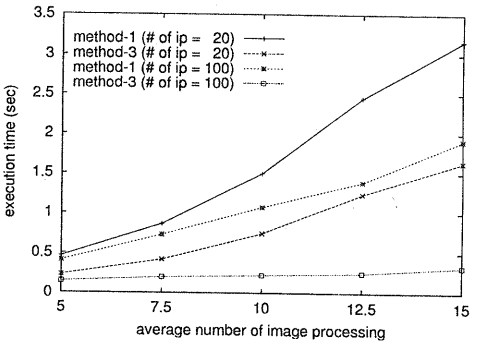

図 3: 500個の画像処理履歴に対する検索時間

法 (3)が全体的に方法 (1)より速いが，画像処理履歴が長くなるとさらに方法 (3)が有効であることが分かる．

また，画像処理演算が100種類の場合と20種類の場合を比較すると，100種類の場合の方が速く問合せが実行できる．これは，画像処理演算が20種類ある場合では類似した画像処理履歴が多数あるため問合せの結果が多くなり，100種類の場合では問合せの結果が少ないためである．

Extended Generalized Scanと Extended Generalized Lindexを可能性のある全ての一般パス表現に対して設定することは不可能である．しかし，以上の結果は，Extended Generalized Scanおよび Extended Generalized Lindexが長いパスに一致する一般パス表現を含んだ問合せの実行に有効であることを示している．画像処理履歴を問合せるとき，その履歴を省略して指定するには一般パス表現(.ip_%)*を指定することが不可欠である．したがって，この一般パス表現に対する Extended Generalized Scanと Extended Generalized Lindexを作成することは有効である．

6 関連研究

従来の半構造データに関する研究は汎用的なデータモデルの設計に重点を置いたものが多かったため，実際にそのデータモデルをどのようなアプリケーションで使うかが不明確であった [9]．我々の研究では，画像処理履歴を半構造データ

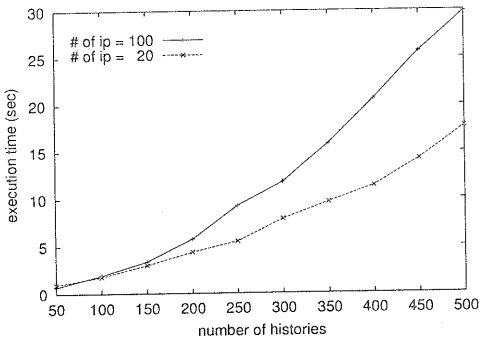

図 4: 平均 10 回の画像処理演算を実行した画像処理履歴に対する問合せの書換え時間

モデルによって表現することで，画像処理履歴の検索が実現できることを明らかにした．特に，画像処理履歴の検索では，画像処理演算の実行順序の条件を指定するためワイルドカードと正規表現が非常に有効であることを示した．

これまでにも一般パス表現を含んだ問合せの最適化手法 [2, 3, 4] が議論されてきた．Christophides 等 [2] は，一般パス表現を持つオブジェクト代数の最適化について議論している．Fernandez 等 [3] は，query pruning と query rewriting の手法を提案している．query pruning は，graph schema とよばれるデータベース内のグラフの構造と問合せとの間の直積を求め，その直積から探索空間を減らした新たな問合せを作成する．query rewriting は，state extents と呼ばれるグラフのルートよりも深いノードから探索を開始することで探索空間を減少させる．文献 [4] でも，一般パス表現を正規表現演算を含まない形に書換える手法を提案している．一般パス表現の書換えは，まず正規表現演算`*`,`+`,`?` を含む表現を正規表現演算`|` を用いて単純パス表現の論理和形式へと置き換え，さらにこの正規表現演算`|` を union 演算子に置き換えて除去する．

我々のアプリケーションでは，画像処理演算を表現するラベルが数多く存在しグラフの深さが深くなる傾向にある．このため，5節の実験で用いた`(.ip_%)*` のような一般パス表現を書換えるためには，長い単純パス表現を列挙する必要がある．図 2で作成したデータベースに対して問合せ 5を書換えるのに要した時間を図 4に示す．書換えに要する時間は，特に履歴数が多くなった場合，図 2に示した問合せの実行時間よりも長くかかってしまうことが分かる．しかし，我々のアプローチでは，前もって一般パス表現をインデックスに登録してあるため，問合せの書換えを必要としない．このため，我々のアプローチは全ての一般パス表現をインデックスに登録しておくことはできないが，画像処理履歴のようによく用いられる一般パス表現が予め分かっている場合には有効であると考える．

7 おわりに

この論文では，画像処理履歴の表現方法と問合せ方法を示し，その問合せに対し Extended Generalized Scan 演算および Extended Generalized Lindex 演算が有効であることを議論した．これらのインデックスを全ての一般パス表現に対して予め準備しておくことは不可能であるが，よく用いられる既知の一般パス表現に対してインデックスを設定することは可能である．特に，画像処理履歴の問合せでは履歴の一部分だけを指定することが必要であるため，これらのインデックスは有効である．今回は単純に extendible hashing を用いてこれらのインデックスを作成したが，格納サイズや検索がより効率的なインデックス構造について検討する必要がある．また，Extended Generalized Scan 演算および Extended Generalized Lindex 演算のコストモデルを検討し，問合せの最適化の向上を図る予定である．さらに，問合せプランの作成を見直すことによって Lore で準備されている他のインデックスも一般パス表現に対応するように拡張することを検討している．

参考文献

[1] Abiteboul, S., Quass, D., McHugh, J., Widom, J. and Wiener, J.: The Lorel Query Language for Semistructured Data, *Int. Journal on Digital Libraries*, Vol. 1, No. 1, pp. 68–88 (1997).

[2] Christophides, V., Cluet, S. and Moerkotte, G.: Evaluating Queries with Generalized Path Expressions, *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, pp. 413–422 (1996).

[3] Fernandez, M. and Suciu, D.: Optimizing Regular Path Expressions Using Graph Schemas, *Proc. of the 14th Int. Conf. on Data Engineering*, pp. 14–23 (1998).

[4] McHugh, J. and Widom, J.: Compile-Time Path Expansion in Lore, *Proc. of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats* (1999).

[5] McHugh, J. and Widom, J.: Query Optimization for XML, *Proc. of the 25th Int. Conf. on VLDB*, pp. 315–326 (1999).

[6] Object Management Group, Inc.: *The Common Object Request Broker: Architecture and Specification, Revision 2.2* (1998).

[7] 田幡勝, 有次正義, 金森吉成: 分散環境における画像オブジェクトの版管理機構の実現, 情報処理学会論文誌：データベース, Vol. 40, No. SIG5(TOD2), pp. 79–90 (1999).

[8] 田幡勝, 有次正義, 金森吉成: 半構造データモデルによる画像処理履歴の管理, 情報処理学会論文誌：データベース, Vol. 41, No. SIG1(TOD5), pp. 64–75 (2000).

[9] 田島敬史: 半構造データのためのデータモデルと操作言語, 情報処理学会論文誌: データベース, Vol. 40, No. SIG3(TOD1), pp. 152–170 (1999).