XMLデータベースにおける検索グラフによる検索最適化手法

服部雅一[†] 野々村克彦[†]
金輪拓也[†] 末田直道[††]

企業内には半構造データである文書が多く存在する．これらは多くの場合，テキストデータとしてキーワード検索や全文検索などで共有・再利用がなされているのが現状である．これら，半構造データをうまく表現する言語としてXMLが注目されている．XMLのタグにより，ある程度の意味が与えられるため，きめ細かい検索が期待できる．これらXMLデータを格納するための1つの方法としてRDBへの格納が考えられる．しかし，RDBがテーブル構造のデータモデルであるのに対して，XMLは木構造のデータモデルであり，かつスキーマがない場合，格納・検索に無理が生じている．我々は，XMLのデータモデルに適応したデータベースシステムを開発した．半構造であるがゆえの検索の冗長性が発生する．この問題に対して，本論文では検索の高速化を図るための最適化手法を提案する．これは，問合せを検索グラフに展開して，この各ノードに対して検索コスト最小化のためのルールを適用することで，最適探索パスを生成するものである．この結果，最適化による効果は問合せによっては1,000倍以上の結果を得ることができた．

Retrieval Optimization Technique Using A Query Graph in XML Database

Masakazu Hattori,[†] Katuhiko Nonomura,[†] Takuya Kanawa[†]
and Naomichi Sueda[††]

In a company, there are many documents that are semi-structured data. In order to get these documents from a computer, we usually use a keyword search or a full-text search method. Recent years, XML that can handle semi-structure data is attracting attention. Since XML document is able to have tags that are defined optionally by a user, a system can retrieve more effectively. Many researches are how to store XML data into RDB. A data model of RDB is a table structure, but XML data model is a tree structure. Because of difference of data model between RDB and XML, to store XML data into RDB by force causes a fall of retrieval capability. We developed a native XML database system that fits for the tree structure data model. In this paper, we describe a retrieval optimization technique that can realize high-speed retrieval. By creating a query graph from a query, the system generates an optimized retrieving path by heuristics pattern matching to each graph node. Our system could retrieve XML data 10 to 1000 times as fast as the conventional systems.

1. はじめに

インターネットは，企業活動や個人生活に不可欠なものとなってきている．インターネット，Web上で入札，オークション，資材調達などの商取引が行われている．これらWebアプリケーションを構築するために必要な技術としてXML（Extensible Markup Language）が注目されている．

これは業界標準のデータ交換の表現形式として,「自由タグによるデータの意味付け」,「記述の柔軟性」,「可読性の容易性」が有効であるという理由によるものである．

しかし，XMLは単にデータ交換用言語としての利用にとどまらず，XMLのベースとなったSGML（Standard Generalized Markup Language）やHTML（Hyper Text Markup Language）のようなドキュメント記述言語としても有効である．

我々はWebの普及により大量の情報が入手できるようになったが，その一方で必要な情報が膨大なデータの中に埋没してしまい，十分に活用できないという

† 株式会社東芝研究開発センター
Toshiba Corporation Corporate Research & Development Center
†† 大分大学工学部知能情報システム工学科
Department of Computer Science and Intelligent Systems, Faculty of Engineering, Oita University

弊害も生じている．このような情報を単にテキスト情報として検索，再利用を行おうとしたとき，我々は意図した結果を得ることの困難さを多く経験している．

文書データを XML で記述して意味あるタグ情報を付加することにより，単なるテキストデータよりも有効な検索・再利用が行える．また，XML は半構造データ表現にも適しているため，様々な文書データに適応できる．

このような発想は，企業内における特定の個人や部門が保有するノウハウや業務データのうち企業の経営に重要なものを蓄積して「経営資産」として活用しようとする活動，いわゆるナレッジマネジメントに有効である．

そこで文書データを XML で表現し，計算機がある程度，意味を扱えることができるようになると，きめ細かい検索，様々な観点からのデータ分析などが行えるようになる．つまり，従来基幹系で行われている RDB（Relational Data Base），データウェアハウスによる OLAP（OnLine Analytical Processing）が，デジタルドキュメントをベースにした情報系でも行えるようになる．我々はこれをテキスト版 OLAP [1] と呼んでいる．

テキスト版 OLAP に代表される検索・再利用・分析などの処理を行うためには，XML データの効率の良い格納と検索方式が必要になってくる．現在，RDB に XML データ構造を格納する試みがある．RDB は表形式を基本とし，スキーマを前提としているため，XML のような木構造のデータを格納するため様々な工夫をする必要がある．また，半構造の特徴を生かすため，スキーマレスで RDB に格納する方式も提案されているが，格納，変更処理に効率が著しく低下する問題点がある [2]．

木構造には親和性の良い OODB（Object Oriented Database）も，RDB と同様にスキーマを前提にしているのでデータ構造の変化に柔軟に対応できない．

我々は上記のような既存データベースの問題点を鑑み，XML データの特性にあった DBMS（Data Base Management System）である KnowledgeFactory（KF）を開発した．

本論文は，RDB と同様に検索・再構成を行うために，XML データ構造特有の特性を考慮に入れた検索グラフによる検索最適化手法について論じる．

2. 関 連 研 究

本研究の目的は大量に存在する半構造文書データを XML で管理し，これらを共有，再利用／再構成できる XML データベースマネージメントシステムを構築することにある．そのためには，XML 文書データの高速な検索を行うためのデータの論理モデル，物理モデルの明確化，XML 問合せ言語およびその検索最適化技術，DB としてのトランザクション処理，信頼性のための障害復旧，セキュリティなどが研究・開発課題である．

本論文では，XML 問合せにおける検索最適化に関して提案する．そのためには，まず，そのベースとなるデータ格納方式，XML 問合せ言語，最適化技術の関連研究に関して概観する．

2.1 XML データ格納，検索方式

XML データを格納する方法としては，いくつか方法が提案されている．

一番単純な方法としては，XML データをそのまま，テキスト情報としてファイルに格納する方法である．これは，非常に簡便な方法であるが，XML として格納しているにもかかわらず，文書の構造情報，タグによる意味情報を有効に利用することができない．

2 番目の方法として，現在広く利用されている RDB で XML データをうまく管理するものである．

そして 3 番目の方式は，XML の持っている階層木構造をうまく管理する XML データ用のネイティブなデータベースのアプローチである．我々のアプローチは，基本的には，このネイティブなデータベースを構築することによって，高速な検索を実現するとともに，XML データの加工，再構成も可能にすることを目指している．

そこで，2 番目と 3 番目の方式に関する関連研究について述べる．

2.1.1 RDB による格納方式

XML データを RDB に格納する場合，DTD（Document Type Definition）をベースにするか否かで大別できる．

＜DTD に基づく格納方式＞ Shanmugasundaram らは DTD を解析して関係スキーマを自動的に作成する方式を提案している [3]．これは，DTD を簡略化し DTD グラフを作成し関係スキーマに変換する方法である．この変換方法として *Shared* と *Hybrid* が提案されている．どのような要素に対して関係スキーマを作成するかのルールは *Shared* 法ではおおむね以下のようになっている．

- DTD において “*”（繰返し）の指定された要素
- 複数の要素から共有されている要素
- 親を持たない要素（root）

唯一の親しかない要素は，その祖先の関係スキーマに組み込む．

＜DTDに依存しない格納方式＞ Florescu らは以下の3つのアプローチを示している[4),5)]．

- Edgeアプローチ
  この方式はXML文書を表すグラフのすべてのedgeを1つのテーブルの中に格納してしまう方式である．具体的にはテーブルの属性は以下のとおりである．
  $T_{Edge}(Source, ordinary, name, flag, target)$
  - Source：XMLの文書グラフのノード番号
  - ordinary：そのSourceノードを親に持つ子ノードの発生順番
  - name：ノード名（要素名／属性名）
  - flag：そのノードの型，または参照フラグ
  - target：その値の値テーブルへのポインタまたは参照におけるSourceへのポインタ

  代表的なシステムとしてSTORED[6)]がある．
- 二項関係（Binary）アプローチ
  上記方式に対して，同じ要素名を1つのグループにまとめ関係テーブルを作成する方式である．
  $B_{name}(Source, ordinarl, flag, target)$
- 包括的（universal）アプローチ
  この方式はXML文書を表すグラフのすべてのedgeを1つの包含されたテーブルに統合する方式である．上記二項関係テーブルのすべての表を外部ジョインした表を作成することを意味している．

DTDに基づいた方法はもとより，二項関係および包括方式はXML文書の論理構造が変化すれば，関係スキーマを変更する必要が出てきて，半構造データを扱うにはふさわしくない．

スキーマの変更の柔軟性という意味ではEdgeアプローチで対応可能であるといえるが，問合せの経路式が長くなると，それに比例した結合演算が必要になり，検索の負荷が高くなる．この問題点を解決するためにYoshikawaらは木構造の各ノードに対して，その経路情報と位置情報を関係表で管理する方法を提案している[7)]．しかし，陽に経路情報，位置情報を持つことにより，更新処理のオーバヘッドが大きくなる問題点がある．

2.1.2 XMLネイティブなデータベース

XMLデータは階層木構造であり，半構造の特性からDTDを前提としないような領域で利用する場合，前項で述べたようにRDBに格納することは，かなり無理が生じることになる．そこで，階層木データモデルに合ったデータベースを構築する動きが活発になっている．

すでに商用レベルとして提供されているXMLネーティブなデータベース（XML-DBと呼ぶ）がいくつか存在する．特にSoftware AG社[8)]のTamino，eXcelon社のeXcelon[9)]が有名である．Software AG社はRDBであるAdabasの開発を行っており，これらRDBとシームレスに連携できることを特徴としている．一方，eXcelonはオブジェクト指向データベース（OODB）をベースに構築している．

XML-DBにおいては，特にその論理モデルが特徴になってくる．XMLはドキュメント的側面とデータ的側面の両方を持つため，それらをシームレスに扱えるモデルが望まれる．

管理するドキュメントのモデルとして考えられるのは，ドキュメント格納時にファイルという物理的イメージを残さない方式と残す方式の2通りに大別される．前者の方式は，ドキュメントを格納する際に，このドキュメントに対して一意な論理ID付けを行い，以後はこのIDを用いて管理する方式である．一般的に1ドキュメントは1ファイルで表現される場合が多いので，論理IDとドキュメントが1対1となることが多い．この方式は，比較的小さいサイズのドキュメントの大量格納するモデルであり，多くのXML-DBがこの論理モデルを採用している．

また，ドキュメントの管理方法としてツリー方式とコレクション方式の2通り存在する．

(1) ツリー方式

図1 (A)のように，1つのデータベースにおいて，複数のドキュメントが任意の階層上にツリーとして構成されるモデルがツリー方式モデルである．データベース中の位置はパスにより一意に識別され，ドキュメントの意識なしに，任意のパスに対して操作を行うことが可能である．KFはこの方式を採用している．物理的ファイルイメージは完全に論理的ツリーに展開されており，ドキュメントを横断した操作が容易に実現できる．長所と短所は以下のとおりである．

［長所］

- 複数のドキュメントを横断した検索が容易．
- マニュアルのような階層的なドキュメントの構築が可能．
- パスをベースに考えることができるので，

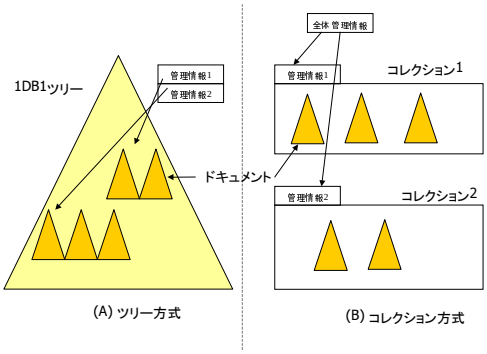


図 1 論理データモデル
Fig. 1 Logical data model.

モデルの柔軟性が高い．

- スキーマに相当するものをパスに埋め込むことも可能．

［短所］

- トランザクションやセキュリティの管理が複雑．
- 物理的なファイル情報を残したい場合，別管理が必要．

(2) コレクション方式

図 1 (B) のように，複数のドキュメントを，コレクションという概念で管理する．コレクションはファイルシステムにおけるディレクトリに役割に近く，Tamino や eXcelon などはこの方式を採用している．この方式の長短は基本的にツリー方式と逆になる．

データ的側面を考えた場合，スキーマ（DTD）に対する対応がポイントになる．KF は，DTD 有無に関係なく，同一の枠組みでドキュメントを管理することができる．

2.2 最適化方式

問合せの最適化は RDB における SQL の問合せ最適化技術が長年研究・開発がなされている．RDB が今日，多くの分野で利用されているのも，この最適化技術の進展が大きく寄与している．その意味で，RDB をベースにした XML データのストレージ方式は，その資産をそのまま利用できるため有利である．多くのアプローチは XML 問合せ言語を SQL 問合せに変換して，問合せ実行する方式をとっている．

Fernandez らは，Web サイトの検索の最適化についての提案を行っている[10]．これは，企業のイントラネット上での Web サイトからグラフスキーマを抽出して，このグラフをベースに 2 種類の正規表現形式に対応した問合せ最適化方式を提案している．1 つは，探索をグラフの断片に限定するような枝刈りされた質問に書き換えるアルゴリズムと，2 つめは RDB が関係代数の等価変換を行うように，スキーマの各ノードの状態範囲（state extent）を利用して問合せを書き換え，グラフの探索を減らす方式である．これは，弱構造のスキーマを抽出して，これをベースに最適化するというものである．

また，OODB における最適化の研究もなされている．Gardarin らはスキーマをベースにした OODB において，コストモデルを用いて深さ優先探索，前向き，後向きなどアルゴリズムを提案して比較検討している[11]．

XML-DB として我々の研究と近いアプローチとしてはスタンフォード大学の Lore[12]がある．Lore では，RDB と同様の SCAN，JOIN，SELECT の 3 オペレータを中心とし，高速化のため LINDEX，VINDEX などの索引情報を用いたオペレータを提案している．Lore における最適化は，まず与えられた問合せから論理問合せプランを Set，Glue，Chain などの論理オペレータを用い生成する．次にその論理問合せプランを操作可能な物理問合せプラン（Scan，Lindex，Vindex などのオペレータ）を生成する．生成方式は論理問合せプランをサブプランに分割し，各サブプランをトップダウンとボトムアップ探索のテンプレートを利用して置き換えることにより，ハイブリッドな物理問合せプラン集合を作成する．この物理問合せプラン集合の中からコストモデルを用いて，最もコスト評価の良いプランを選択する方式を提案している[13]．

我々のアプローチも問合せをグラフに展開（Lore における論理問合せプラン）する．このグラフに対してコストを考慮した最適化ルールを用い，展開しやすいノードをグラフパターンマッチによって求める方式をとっている．つまり，問合せを制約グラフととらえ，制約充足問題に定式化した方式を提案している．

このように我々の方式は，Lore のような置換えではなく，展開しやすいノードをルールのパターンマッチにより選択し，展開する方式をとっている．したがってパターンマッチによって生成する解の組合せは Lore よりも多くなり，より良い解を生成できる可能性があるというメリットがある．

2.3 XML 問合せ言語

W3C において，XML 問合せ言語の標準化活動が実施されている．1998 年 11 月に開催された「QL'98 The Query Language Workshop」[14]において，約 70 種類の問合せ言語が発表された．その後，標準化を進めるべく 1999 年 9 月に「XML Query Working

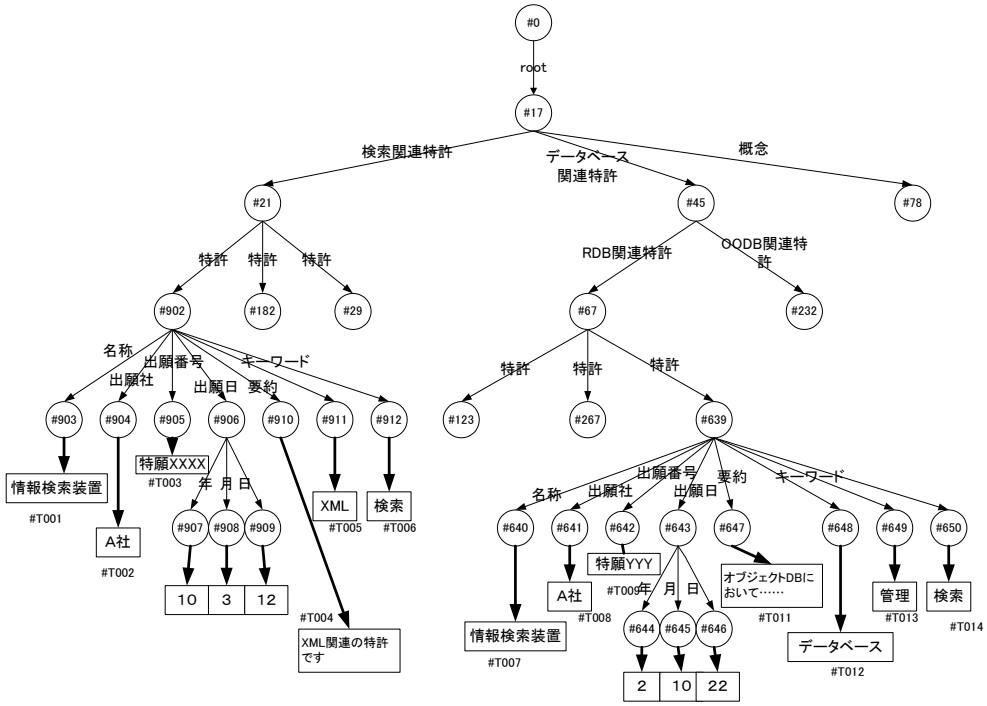


図 2 構造化文書の論理的格納構造
Fig. 2 Logical image of database.

Group」[15]が結成された．代表的な問合せ言語には，XQL[16]，XML-QL[17]などがある．

XQL，XML-QL を統合する形で Quilt[18]がある．現在この Quilt をベースにして XQuery が Working Draft 段階にある[19]．

KF は独自の XML 問合せ言語 KF-QL を実装している☆．KF-QL 自体 XML 表現になっており，言語仕様としては「XQuery のサブセット＋独自仕様」という位置付けである（XQuery の XML シンタックスとして XQueryX が W3C Working Draft 段階にある[20]）．

3. KF の概要

3.1 XMLドキュメントの論理的格納方式

図 2 に KF における階層木構造である XML データを格納した論理的イメージを示す．この階層木構造はノードとアークで構成されており，階層木の部分木はパスによって指定される．

(1) 階層木表現
図 2 では**図 3** に示す特許公報データがタグ付けされて表現されている．このようなコンテンツが複数，1 つのデータベース（特許.DB）に格納されている様子を示している．また，ここでは，それぞれの特許公報のコンテンツの格納位置が階層木構造上，どこの位置にきてもよいことを示している．

特許公報情報のほかにも例のように「概念」情報といった異なるスキーマ（構造）のデータも同一データベース内に混在して格納できる．つまり，このような構造化文書を集めた構造化文書データベース（特許.DB）は UNIX や Windows のディレクトリ構造のように階層的に格

```
<特許リスト>
<特許>
  <名称>情報検索装置</名称>
  <出願者>A社</出願者>
  <出願番号>特願平XXXX</出願番号>
  <出願日><年>10</年><月>3</月><日>12</日></出願日>
  <要約>
    情報の提示形式の変更が利用者側の観点で自由に行え、
    情報活用の範囲が広がると共に、情報活用の促進が図れる
    を提供する。
  </要約>
  <キーワード>XML</キーワード>
  <キーワード>検索</キーワード>
</特許>
</特許リスト>
```

図 3 XML 文書データ例
Fig. 3 Example of XML data.

☆ 現在，XQuery 対応に移行中である．

納されている．

この階層木の各ノード（図 2 では番号が付された円形で示されたもの）をドキュメントノード（D ノード）と呼ぶ．任意の D ノード以下の部分階層木は，構造化文書データベースから切り出された構造化文書を示している．

各 D ノードにはオブジェクト ID（図 2 では円内の番号）が割り振られている．オブジェクト ID は構造化文書データベース内ではユニークな数値を持つ．図 2 では，階層木のルートとなる D ノードに，それが根 D ノードであることを特定できるように「#0」が割り振られている．各ノードからのアークにはタグ名が付されている．

具体的なタグ内のデータは末端 D ノードのリーフに位置付けられている（図 2 では D ノードから太線矢印でリンクされている四角内のデータ）．このタグ内データタイプがテキストの場合はテキスト ID（例：#T001）が付与されている．

各 D ノードは検索の効率を高めるため，親子関係の双方向リンク情報を持っている．

（2） パス表現

XML データベースを作り出すうえで重要な概念として「パス」がある．パスは XML データベース上の特定のロケーションを指し示すための手段である．KF プロトコルにおけるパス表現として以下のように行う．

uix://root/検索関連特許/特許

UNIX や Windows のディレクトリ構造と同じように “/” で区切られた文字の並びになるが，それらと異なるところは，同じラベル（タグ名）を持つ部分木が同一ノードに存在しうることである．KF ではロケーションを明確化するために，“[n]” というノード順序を付加する．同じラベルを持つ子ノード群の中で “[0]” が先頭子ノード，“[1]” が次子ノードということになる．

uix://root/検索関連特許/特許 [0]

uix://root/検索関連特許/特許 [1]

（3） 格納機能コマンド

KF における格納機能におけるコマンドは**表 1** のとおりである．

3.2 インデックスファイル

KF では XML データを高速に検索するために 2 種類のインデックスファイルを作成している．

（1） 要素名称生起インデックス

表 1 格納機能コマンド
Table 1 Command list for storing XML data.

機能	コマンド	パラメータ	概要
追加	setSchema	path , schemaData	指定されたパス(path)の下にXMLデータが従うべきスキーマデータ(schemaData)を指定する
	regist	path , xmlData	指定されたパス(path)の下にXMLデータ(xmlData)を挿入する
更新	update	path , xmlData	指定されたパス(path)以下をpathも含めてXMLデータ(xmlData)と置き換える
削除	remove	path	指定されたパス(path)以下をpathも含めて削除する
取得	getXML	path	指定されたパス(path)以下のXMLデータを取得する
	getSchema	path	指定されたパス(path)の下にXMLデータが従うべきスキーマデータを取得する

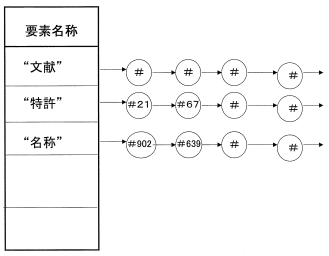


図 4 要素名称生起インデックス
Fig. 4 Index of element name occurrence.

データベース内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，各要素名がデータベースに格納されている位置（D ノード）と関連付けているインデックスファイルである．**図 4** にそのイメージを示す．図 2 において「特許」という要素名の D ノード集合は（#902, #182,···, #639,···）である．この各 D ノードの親 D ノード（#21, #67,···）をインデックスとして登録する．このように親 D ノードでインデックス化することにより，インデックスファイルサイズを圧縮することができる．

（2） データ生起インデックス

データ生起インデックスは N-gram をベースとする方式で実装した．**図 5** にその概略構造を示す．図 2 において D ノード #910「要約」にテキストとして「XML 関連の特許です」と記述されている．このテキスト情報にテキスト ID「#T004」が割り振られている．このテキストを 2-gram として 2 文字（2 バイト／文字）で切り出し，テキスト中の位置オフセットを語彙テーブルにセットする．それぞれの語彙を語彙ファイルにリンクしてデータインデックスファイルに関連付ける．データインデックスファイルは，(D ノード番号，親ノード内での自ノード

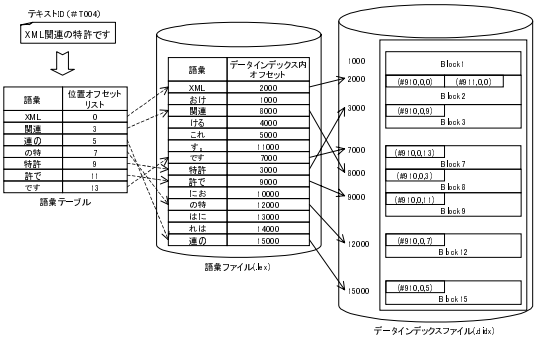


図 5 データ生起インデックス
Fig. 5 Index of data occurrence.

の順番，テキスト中の位置オフセット）のデータセットを管理する．

検索は問合せのテキストを 2-gram で語彙を抽出し，語彙ファイルより問合せテキストを含んでいる D ノードをデータインデックスファイルより獲得することにより容易に検索できる．

3.3 問合せ言語 KF-QL

2.3 節で述べたように，KF-QL は XQuery のサブセットに独自仕様を加えたものである．

XQuery は，非常に広範な仕様を提案している．KF-QL はその中で，ドキュメント検索，再構成には必須であると考える機能を実現している．XQuery のサブセットという位置付けでは以下のような仕様範囲をカバーしている．

- 対象データは「文字列データ」（リスト形式，型付きデータは対象外）
- XPath におけるパス条件，AND，比較オペレータ，正規表現（ドキュメントオーダ，OR，xf.not() 条件，関数解釈，AXIS 解釈は対象外）
- FLWR 構文における変数タグ，ネスト，複数 DB バインド，SORTBY など（BEFORE，AFTER，SOME，EVERY，REF，DEDEF は対象外）

KF 独自仕様と，そのための特長を以下に述べる．

- 問合せの変数化：条件付けに用いる値を変数化することができる．このことにより，1 つの問合せを様々な問合せに利用することができる．
- 検索結果のパスの取得機能：検索した結果のパスも同時に取得することができる．
- KF-QL の表現は XML：問合せ自体を XML データとして管理できるので，問合せ文の共有化・再利用が容易になる．つまり従来のナレッジマネジメントが “What” の共有が主体であったのに対し，様々な分析方法を共有することができるため “How” の共有が可能になる．また，検索した問合せを利用する際は，上記の「問合せの変数化」により，その変数を自分の問題に適用することで簡単に問合せを再利用できる．

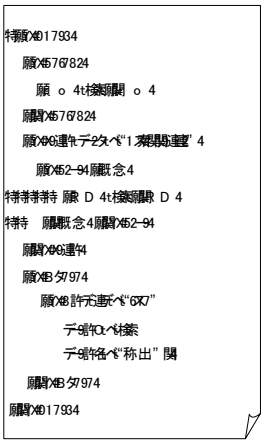


図 6 検索要求例
Fig. 6 Example of query data in KF-QL.

4. XML データ検索最適化方式

RDB などスキーマをベースにしたデータベースでは，属性の所在・関係が明確に定義されている．一方，XML で扱うデータは半構造であることが多い．つまり，XML データは以下のような特徴を有している．

- 同一構造を有する文書データの発生階層が必ずしも一定ではない．
- 同一構造の入れ子状態が発生する可能性がある．
- 上記の理由から，ユーザは検索したい情報の明確な位置関係が見えないので，曖昧なパス指定での検索になる．

以上の理由により，従来 RDB の検索最適化処理の援用は困難である．つまり，関係代数式の等価変換による処理木に対するヒューリスティック適用に基づく最適プラン選択という手法が直接的には利用できない．

そこで，我々は関係代数演算に代わる問合せの操作モデルを階層木や部分木に対する操作制約モデルとして定式化した．このモデルに基づき問合せを検索グラフに変換して，最適化ルールによるパターンマッチで検索プランを生成し実行する方式を開発した．以下，その詳細について述べる．

4.1 検索グラフ

検索グラフは，KF-QL で記述された検索要求文の構文解析を行い，双方向リンクとノードを含むネットワークを形成する．**図 6** は，『指定された文書パス

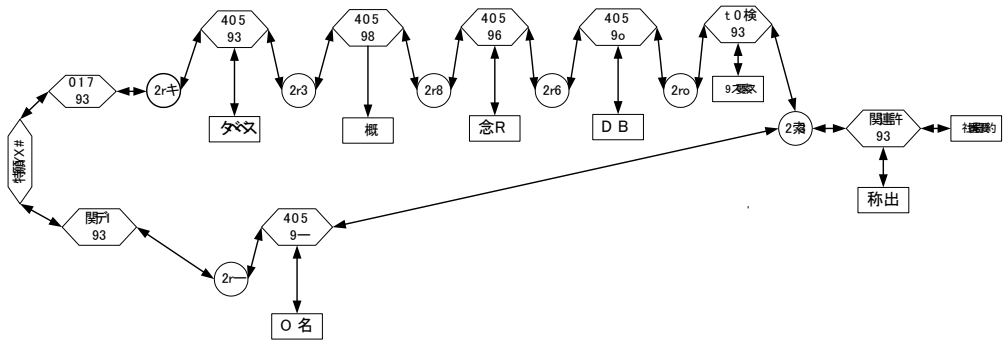
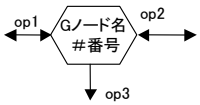


図 7 Query に対する検索グラフ
Fig. 7 Query graph corresponding to Fig. 6.

表 2 G ノードの種類
Table 2 G-Nodes.

Gノード	リンク		
	op1	op2	op3
QUERY	「CON」	「AND」	—
AND	「QUERY」	「TAG」	—
TAG	上位変数 Gノード／「AND」	下位変数 Gノード	データ
ATT	上位変数 Gノード	下位変数 Gノード	データ
CON	「QUERY」	「TAG」	—
VAL	上位変数 Gノード	下位変数 Gノード	ノード種
CMP	上位変数 Gノード	比較子	データ



(⟨kf:from path=”uix://root”⟩) 中に任意 (⟨kf:star⟩) に出現する「特許」情報 (⟨特許⟩) に対して，下位の「名称」情報 ($x) が「検索」という文字列を含んでいるならば (⟨kf:cmp op=”like” parm1=$x parm2=”検索”/⟩),「名称」情報を抽出して「文献」情報として構成せよ（select 文）』という検索要求文の事例である．

我々は，階層木や部分木に対する操作制約モデルとして以下の 4 種類の制約タイプに分類して**表 2** に示す 7 つの検索ノードを設計した．

(1) ノード間階層制約
- 要素間階層制約（TAG，ATT）：ノード間の階層（タグ，属性）に関する条件を記述する．
- ノード種別制約（VAL）：指定ノードの種別（要素，テキスト，数値など）に関する条件を記述する．

(2) ノード順序制約☆：ノードの順序に関する条件を記述する．

(3) ノード比較制約（CMP）：ノード比較（=, ⟨,⟩, like など）に関する条件を記述する．

(4) メタ制約（QUERY，AND，CON）：上記制約群を参照する．

図 7 は，図 6 の検索要求を上記操作制約モデルに基づいて生成した検索グラフである．問合せ文から検索グラフへの展開は，問合せ文にある，タグ，属性，変数，データ，演算は検索グラフ側各 G ノード，変数ノード，データと一意に対応付けられるため容易に展開できる．

四角形で示されるノードは具体的なデータ（文字列）を表している．六角形で示される検索ノードを G ノードと呼び，円形で示されるものを変数ノードと呼ぶことにする．変数ノードは，変数を表すノードであり，「$」で始まる文字列を持っている．変数ノードは内部的に生成された変数と，検索要求文に含まれる変数と 2 種類がある．

表 2 に示すように各 G ノードは 2 つまたは 3 つの双方向リンク（op1，op2，op3）を持っており，その接続規則が定められている．

「Query」G ノードは図 7 のように検索要求文全体に対応しており,「AND」G ノード以下は Where 句,「CON」G ノード以下は Select 句に対応している．

上記の検索要求事例においては,『指定された文書パス「root」以下の任意の「特許」情報…』の条件は 4 つの「TAG」G ノード（「root」,「*」,「特許」,「名称」）で表現されている．2 つの G ノードをつなぐ変数ノード（「$_1」,「$_2」など）は図 2 で示される D ノードで束縛可能な変数である．たとえば「$_2」の変数 G ノードは上流（左側）の 2 つの「TAG」G ノードから解釈すると『指定された文書パス「root」以下の任意の文書』を表し，下流（右側）の「TAG」G ノードとも接続されているため，それもあわせて解釈すると『指定された文書パス「root」以下で,「特許」タグを先

☆ KF の本バージョンでは仕様外．

表 3 最適化ルール
Table 3 Optimization rules.

番号	Gノード	コスト	IF				THEN
			op1	op2	op3	その他条件	オペレータ
1	TAG	1.0	AND	未	具		PATHINST
2	TAG	0.5	具	未	具		PATHEXPAND1
3	TAG	0.2	未	未	具	op3に要素名称生起インデックスが存在	PATHEXPAND2
4	TAG	0.1	未	具	具		PATHEXPAND3
5	TAG	0.3	具	具	具		PATHCHECK
6	TAG	0.6	未	具	"*"		PATHEXPAND3
7	TAG	1.0	AND	具	root		NOP
⋮							
11	VAL	0.5	具	—	具		JOIN
⋮							
21	VAL	0.2	具	—	未		VALUE
⋮							
31	CMP	1.0	具	—	未		SELECT
32	CMP	0.1	未	—	具	op3に対するインデックスが存在する	FIND
⋮							
101	CON	1.0	QUERY	—	—	QUERYのAND条件がすべて処理された	CONSTRUCT

頭に持ち，その中の「名称」タグがあり･･･』を表す．

4.2 最適化ルール

本検索最適化方式は，検索グラフ上での各 G ノードに対して，最適化ルールとパターンマッチさせながらコストの少ない G ノードを選択し，その値を具体化してゆき，その具体化された状況を伝播させながら探索プランを立てていく方式をとっている．

ここでいう最適化ルールは**表 3** に示すようになる．すなわち，各ルールは,「ルール番号」,「G ノード」の種類，検索にかかわる「コスト」およびパターンマッチを行ううえでの「条件（IF)」と「そのときのオペレータ（THEN)」からなる．

4.2.1 コ ス ト

各最適化ルールに割り当てられているコストは，そのルールの展開のしやすさを全ルールにおける相対値で設定したものである．コスト設定基準項目は以下のとおりである．

(1) 展開対象の双方向リンクがインデックスである場合
　データ生起インデックス＜要素名称生起インデックス
(2) すでに展開されている双方向リンクの数
　すべて展開＜ 2 個展開＜ 1 個展開＜すべて未展開
(3) 比較演算の対象データ種類
　数値データ＜日付データ＜テキストデータ

各コスト設定基準項目は (1)，(2)，(3) の順でコストは高くなるとした．また各基準内においては，その構成要素のコスト順序付けは上記のとおりとした．

4.2.2 条件部（IF）

条件部は各 G ノードの双方向リンク（op1，op2，op3）の状態を表している．各リンクの状態がすでに具体化されているか（「具」）またはいまだ具体化されていない（「未」）状態か，固定先リンクかなどを条件としている．また，その他の条件として，そのリンク要素がインデックスファイルに対応しているかなども条件に入れられている．当然，インデックスファイルに対応した条件を持っているルールは，検索コストが低く設定されている．

4.2.3 実行部（THEN）

実行部におけるオペレータは以下の 10 種類がある．各オペレータは具体化されていない 1 つまたは 2 つの双方向リンク（op1，op2，op3）の値を具体化する．

- **PATHINST：**パス「root」を取り出す．
- **PATHEXPAND1：**指定された要素名をキーにして上位 D ノード群からマッチする D ノード群を算出する．
- **PATHEXPAND2：**指定された要素名をキーにデータベース内で発生する親子 D ノードのペアを算出する．
- **PATHEXPAND3：**指定された要素名をキーとして，子供ノードから親ノードを算出する．
- **PATHCHECK：**2 つの D ノード集合が与えられたとき，それらが指定された要素名で親子関係にある 2 つの G ノードの組合せを算出する．
- **JOIN：**変数ノード「x」が op リンクで接続している複数の G ノードから具体化が進行して，x の重なり合ったときに行われる結合演算．
- **VALUE：**変数ノード「x」のノード種に対応する要素データの候補を算出する．
- **SELECT：**変数ノード「x」に対する要素データを選択するときの比較演算をする．
- **FIND：**インデックス化されている要素データの候補を算出する．
- **CONSTRUCT：**対象とするのは，CONST ノードである．CONST ノード以下は，先に具体化された変数を末端ノードとした出力形式のテンプレートである．テンプレートは，簡略化のため G ノードを用いて表現されている．CONSTRUCT オペレータは，出力形式のテンプレートと最終の変数バインドテーブルを組み合わせて，DOM や XML 文字列などを出力する．

4.3 検索プラン生成・実行処理

4.3.1 動 作 概 要

検索プランの生成・実行は，プロダクションシステ

Procedure 検索プラン生成・実行処理
begin
・検索グラフのすべての G ノード要素を WM に格納する．
・変数 G ノートテーブルを作成し初期化する
V_Table={$_1{},$_2{},···}
・CS={}
while(WM !={})
begin
・***WM* とルールとのパターンマッチ**
マッチしたルールと WM 要素の組を CS に入れる
・***CS* の中でコスト最小なセットを選ぶ**
CS_k=< G_i, R_j, C_m >
・**ルール R_j を G ノード G_i に対して実行する**
* ルール実行によって具体化された該当変数 G ノード テーブルに値を埋める
$V\_Table\_\$n$={ 具体化された D ノード番号, または値 }
・***WM*，*CS* を更新する**
* CS_k を CS から削除する．
* G_i を WM から削除する．
* G_i に対してオペレータが作用することにより
* op_n が具体化されるそれにともない接続するリンクノードの該当 op の状態が変化するので WM にある該当 G ノード G_h の op の値を変更する．
end
end

図 8 検索プラン生成・実行処理
Fig. 8 Pseudo code of query plan generation and execution.

ム的[21]な動きで実現している．つまり，『ワーキングメモリ（WM）とルールの「パターンマッチ」を行い，実行候補ルール集合（コンフリクトセット（CS））の「競合解消」を行い，選択されたルールを「実行」しワーキングメモリの状態を変化させ，このサイクルを CS がなくなるまで続ける』という「認識—行動サイクル」をベースにしている．

(1) ワーキングメモリ（WM）
WM には，すべての G ノードの状態は以下のように格納される．
(node ^G_kind ^G_node ^op1 ^op2 ^op3)
G_kind：G ノードの種類，G_node：ノード名

(2) ルール（rule）
ルールは以下のように記述される．
rule_name[cost]{
(node ^G_kind ^op1 ^op2 ^op3)
–〉オペレータ }
たとえば表 3 のルール番号 1 は以下のようになる．
rule1[1.0]{
(node ^g_kind=”TAG” ^op1=AND
^op2=”未” ^op3=”具”)
–〉PATHINST}

表 4 プラン生成結果
Table 4 Result of query plan generation.

ステップ	G ノード	ルール番号	オペレータ
ステップ 1	CMP#1	rule32	FIND
ステップ 2	TAG#3	rule03	PATHEXTEND2
ステップ 3	TAG#4	rule02	PATHEXTEND1
ステップ 4	VAL#1	rule21	VALUE
ステップ 5	VAR#1	rule11	JOIN
ステップ 6	TAG#2	rule06	PATHEXPAND3
ステップ 7	TAG#1	rule07	NOP
ステップ 8	CON#1	rule99	CONSTRUCT

また，表 3 の rule4 のようにその他の条件は Check_index(g_node,op) のような関数が条件文に記述されチェックされる．

(3) コンフリクト・セット（CS）
CS はパターンマッチした結果の候補をリストとして格納する．各要素は G ノード名，ルール名，コストのセットになっている．
CS={〈**G_node,rule_name,cost**〉,/*cdots*，〈···〉}

(4) 処理フロー
図 8 に，検索プラン生成・実行処理の処理フローを示す．検索対象 G ノード選択と，その結果の検索実行を逐次的に行っているフローとなっている．一方，検索実行処理を逐次的に行わず，プランを何種類かを生成して最終的に最もコストの低いプランを実施するプラン生成と検索実行を分けて実行するモードも存在する☆．前者は，山登り法的探索であるので局所最適に陥る可能性もあるが，後者の複数プラン生成におけるプラン生成時間とのトレードオフになる．

4.3.2 実行イメージ

表 4 にプランの生成結果を示す．各ステップの動作は以下のようになる．**図 9** に，変数を結合するまでのステップにおける各変数の遷移状態を示す．

- **<ステップ 1>**「検索」という文字がデータ生起インデックスに存在するため，G ノード “CMP#1” がマッチしデータ生起インデックスにある「検索」の要素データを抽出する（FIND）．
- **<ステップ 2>** G ノード “TAG#3” において「特許」は要素名称生起インデックスに存在するため，「特許」をキーにその親子のペア（$_2,$_3）を抽出する（PATHEXPAND2）．
- **<ステップ 3>** G ノード “TAG#4” において「名称」は要素名称生起インデックスが存在し，かつ親 G ノードはステップ 3 で具体化されているた

☆ 本バージョンにおいては逐次的処理のみサポート．

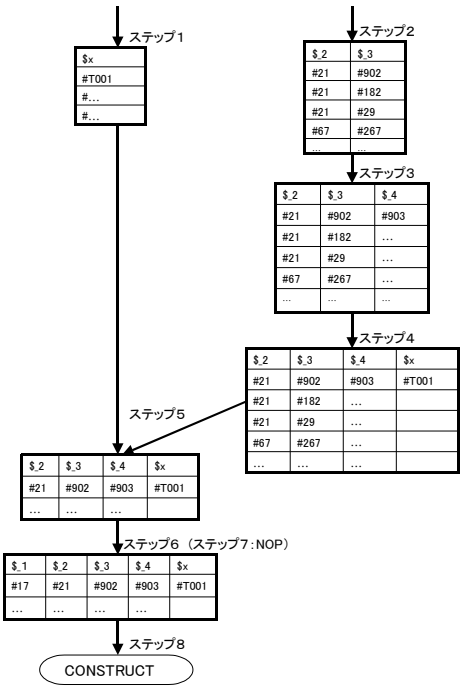


図 9 各ステップにおける変数の遷移
Fig. 9 Image of query plan execution.

め，子ノードを具体化する（PATHEXPAND1）．

- **＜ステップ 4＞** G ノード “VAL#1” において，親ノード値（$_4）を要素名に持つ子ノード群を $x に取り出す（VALUE）．
- **＜ステップ 5＞**ステップ 1 で抽出した$x とステップ 4 で抽出した$x とを結合する（JOIN）．
- **＜ステップ 6＞** G ノード “TAG#2” おいて子ノードからすべての親の集合を抽出して$_1 に入れる（PATHEXPAND3）．
- **＜ステップ 7＞** G ノード “TAG#1” おいて子ノードから親が “root” のものを抽出して$_0 に入れる（NOP）．
- **＜ステップ 8＞**上記 G ノード値を利用して「文献」情報を作り出す（CONSTRUCT）．

5. 実験評価

まず，KF の検索速度に関して実験・評価を行い，次に，我々の提案する最適化の効果を実験・評価する．実験環境は Pentium III 733 MHz，主メモリ 756 MB で OS は WindowsNT4.0 で行った．

5.1 検索速度の性能

5.1.1 実験データ

使用したテストデータは，シェークスピアの戯曲を Jon Bosak がタグ付けした XML 文書☆を用いた．諸元を**表 5** に示す．

表 5 テストデータ諸元
Table 5 Details of test data.

文書数	37
サイズ（MB）	7.65
要素数	179,689
属性数	0
テキストノード数	147.442
単パス数	57

表 6 問合せ文
Table 6 Test queries.

番号	問合せ文
Q1	/PLAY/ACT
Q2	/PLAY/ACT/SCENE/SPEECH/LINE/STAGEDIR
Q3	//SCENE/TITLE
Q4	//ACT//TITLE
Q5	/PLAY/ACT[2]
Q6	(/PLAY/ACT)[2]/TITLE
Q7	/PLAY/ACT/SCENE/SPEECH[SPEAKER = ’CURIO’]
Q8	/PLAY/ACT/SCENE[//SPEAKER = ’STEWARD’]/TATLE

表 7 再構成を含む問合せ文
Table 7 Test queries including XML reconstruction.

番号	問合せ文	備考
Q9	各シーンのタイトルを列挙せよ	付録 A2
Q10	各シーンのタイトルに ‘‘IV’’ を含むもののタイトルを集計しタイトルとその回数のリストを作成する	付録 A3
Q11	各シーンのタイトルに ‘‘IV’’ を含み発言が 100 回以上あるシーンのタイトルと，その回数を列挙せよ	付録 A4

XPath レベルの検索テストは**表 6** に示す 8 種類で実験を行った．また，再構成を行う問合せテストは**表 7** に示す問合せ文で行った．Q9 は SELECT，from 節からなる問合せ，Q10 は Q9 に WHERE 節を加えた問合せ，Q11 は Q10 に加え FROM 節の中に問合せが入れ子に入っている問合せの例である．

5.1.2 実験結果と考察

KF への格納サイズはデータやインデックスデータの合計で 31.4MB になった．元サイズの 4 倍強になる．

表 8 に各問合せの応答時間を示す．応答時間とは，クエリの投入から，結果がサーバから返ってくるまでの時間を計測している．同様な実験を Yoshikawa らは RDB をベースにした XRel をベースに行っている[7),22)]．

☆ URL http://metalab.unc.edu/bosak/xml/eg/shaks200.zip

表 8 検索時間（秒）
Table 8 Executing time for queries in Table 6 (second).

実験番号	Q1	Q2	Q3	Q4
時間	0.02	0.58	0.17	0.18
実験番号	Q5	Q6	Q7	Q8
時間	0.10	0.20	0.01	0.01

表 9 再構成を含む検索時間（秒）
Table 9 Executing time for queries in Table 7 (second).

実験番号	Q9	Q10	Q11
時間	2.66	0.25	0.28

表 9 は，再構成を含む検索実験の結果である．

考察 1： Q1，Q2 はパスが決定的に指定されている問合せである．検索の深さに対応して時間がかかっている．

考察 2： Q3 と Q4 は正規表現 “//” を含む問合せである．Q3 は最初に “//” が現れる問合せである．Q4 のように複数の “//” が任意に発生するような問合せには通常は極端に効率が悪化するのが一般的である．KF は前述の最適方式のように，導出しやすいノードから検索していくため，“//” の位置に関係なく安定して効率の良い検索が行われている．

考察 3： Q5 と Q6 は順序指定の検索である．この検索においても，安定して，高効率な検索が実現されている．しかし，KF-QL には陽に順番を指定して検索を行えるオペレータが用意されていないため，クエリ作成時に工夫を要した（付録 A.1 参照）．

考察 4： Q7 と Q8 とはテキストマッチングの問合せ例である．KF が要素名生起インデックス，データ生起インデックスを持っているため高速化が実現できている．

考察 5： Q9 においては，最適化の場合は，まず構造の中で最も深いタグの 1 つである SPEECH を要素名称生起インデックスから具体化する．その後，親に SCENE があるかどうかをチェックする必要があるので，Q3 などの場合と比較してもコストが高い．Q10 の場合は，WHERE 節により条件付けがなされているために，最適化においてまず TITLE が具体化され，候補が絞られるために Q9 と比較しても非常に高速である．Q11 では Q9 と Q10 を融合させたものであるが，この場合もまず TITLE 部分が最初に具体化されるプランが実行されるので，ほとんど Q10 と等しい計算時間でクエリ実行できる．

表 10 最適化ルール使用の効果（秒）
Table 10 Effect of optimization rules (second).

番号	最適化	前向き	後向き
Q1	0.02	0.02	0.02
Q2	0.58	4.35	1.15
Q3	0.17	13.80	0.20
Q4	0.18	27.31	0.22
Q5	0.10	0.10	0.10
Q6	0.20	0.15	0.20
Q7	0.01	2.78	0.80
Q8	0.01	14.90	0.87
Q9	2.66	7.50	2.10
Q10	0.25	8.10	2.26
Q11	0.28	13.50	2.10

5.2 最適化の効果

5.2.1 実 験 方 法

最適化のチューニングは最適化ルールのコスト値をいかに決めるかにある．つまり，このコストにより，ルールの発火順番が決まることになる．表 8 の結果は，試行錯誤的にコストを決定して実行した結果である．したがってプラン生成結果は導出しやすい G ノードを前後しながら実行している．

比較対象として，検索グラフの上流から下流に評価していくように最適化ルールのコスト値を決めたルールセット（前向き）と，逆に下流から上流に評価するルールセット（後向き）の 2 パターンを設定した☆．

5.2.2 実験結果と考察

表 10 に結果を示す．「最適化」の列は，チューニングしたルールセットを利用したもので表 8，表 9 と同じ値になる．

考察 6 Q1，Q5 は非常に単純でパスも短い問合せであるため 3 者とも差はなかった．また，Q6 以外は，前向きは極端に悪化している．これは，最初の G ノードから評価していくと候補リストが大きいものどうしでのジョインが発生するからである．

考察 7 Q6 に関しては前向きが一番早くなっている．これは順番指定であり，指定順番が [2] であったため早い段階でマッチした結果であると考えられる．

考察 8 Q1，Q5，Q6 は最適化と後向きは同じ値を示している．これは双方同じプランを生成した結果である．基本的に検索グラフの末端にインデックスを持っている G ノードがきたときは同じプランになる可能性が高い．つまり最適化ルールも基本的には後向きのルールのコストが低く設定されている．

☆ インデックスのある G ノードは優先的に展開するルールは双方にも入れてある．データを伝播する方向が一方的に前か後かという意味である．

考察 9 Q9では後向きのほうが速い結果になった．データの内容を見てみるとTITLEは1,000件程度で，SPEECHは30,000件以上あった．[考察5]で述べたようにTITLE，SPEECHともに「要素名称生起インデックス」を持っているため，最適化の探索に関しては，SPEECHが先に展開された後，SCENEにおいてTITLEを含むかどうかをチェックしてしまう．後向きの場合はTITLEとSPEECHを先に展開した結果をジョインした後に，ルートまで展開するので，先にTITLEによる候補限定の効果があるので高速である．つまり，現在の最適化はデータベースの統計量を考慮するルールになっていないためである．

Q10，Q11のようにインデックスを有しているGノードが中間にきている場合は，最適化ルールは前向きと後向きを混在させたプランを生成するので，単純な後向きよりも効率的に探索している．後向き探索では，まずは要素名称生起インデックスや，データ生起インデックスにより展開できるものを優先的に展開し，伝播に関してはルートまで1方向に展開するが，展開の途中にさらに候補を限定することにより，処理を高速化できる場合が多いからである．また，優先度により，できる限りインデックスを利用しようとするので，前向き伝播により候補が絞られるにもかかわらず再びインデックスを適用し，そのジョインをとるのも原因の1つと考えられる．

6. まとめと今後の課題

本論文はXMLの木階層データモデルに適合したXMLネーティブなデータベースシステム（KF）における探索最適化の手法を示した．検索グラフへの展開と最適化ルールによるプラン生成によって，様々な検索パターンにも安定した高速検索が実現できる．また，本論文では記述をしていないが，データーボリュームに対してもほぼ線形な検索速度を示している（付録A.5参照）．

今後の課題としては，Q9の結果にも現れたように，まだコスト評価に関しては不十分である．RDBでも用いているデータベースの統計量も利用して，固定的なコスト計算ではなく，ダイナミックにコスト設定が行える機構が必要である．Loreのコストモデルも参考になる．ただし，どこまで精緻にコストモデルを考えるかは，コストを評価するコストも考慮する必要もあり検討が必要であろう．

また，学習機構を組み込むことにより，自動的に最適化ルールをチューニングしていく機構も興味ある研究対象である．

参 考 文 献

1) 野々村ほか：XMLによるナレッジナマネージメントのためのテキスト版OLAPとその構築環境，情報処理学会論文誌：データベース，Vol.43, No.SIG5, pp.75–86 (2002).
2) Tian, F., Dewitt, D.J., Chen, J. and Zhang, C.: The Design and Performance Evaluation of Alternative XML Storage Strategies, Technical Report, Dept. of Computer Science, University of Wisconsin (2000). Available at http://wwwCS.wisc.edu/niagara/
3) Shanmugasundaram, J., et al.: Relational Databases for Querying XML Documents: Limitation and Opportunities, *Proc. 25th International Conference on Very Large Data Bases*, pp.302–314 (1999).
4) Florescu, D., et al.: A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database, *INRIA Rapport de recherche* (1999).
5) Florescu, D.: Storing and Querying XML Data using an RDMBS, *IEEE Data Engineering Bulletin*, Vol.22, No.3, pp.27–34 (1999).
6) Deutsch, A., et al.: Storing Semistructured Data with STORED, *SIGMOD Record* (*ACM Special Interest Group on Management of Data*), Vol.28, No.2 (1999).
7) Yoshikawa, M., et al.: XRel: A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Database, *ACM Trans. Internet Technology*, Vol.1, No.1 (2001).
8) AG, S.: Tamino. http://www.softwareag.com/
9) eXcelon Corp.: eXcelon. http://www.exln.com/
10) Fernandez, M., et al.: Optimizing Regular Path Expressions using Graph Schemas, *International Conference on Data Engineering* (*ICDE*), pp.14–23 (1998).
11) Gardarin, G., Gruser, J.-R. and Tang, Z.-H.: Cost-based Selection of Path Expression Processing Algorithms in Object-Oriented Databases, *VLDB'96, Proc. 22th International Conference on Very Large Data Bases*, September 3-6, 1996, Mumbai (Bombay), India, Vijayaraman, T.M., Buchmann, A.P., Mohan, C. and Sarda, N.L. (Eds.), pp.390–401, Morgan Kaufmann (1996).
12) McHugh, J., et al.: Lore:A database management system for semistructured data, *ACM SIGMOD Record*, Vol.26, No.3, pp.54–66

(1997).
13) McHugh, J. and Widom, J.: Query Optimization for XML, *VLDB'99, Proc. 25th International Conference on Very Large Data Bases*, September 7-10, 1999, Edinburgh, Scotland, UK, Atkinson, M.P., Orlowska, M.E., Valduriez, P., Zdonik, S.B. and Brodie, M.L. (Eds.), pp.315–326, Morgan Kaufmann (1999).
14) QL'98: The Query Language Workshop. http://www.w3.org/TR/NOTE-xml-ql/
15) W3C: W3C XML Query Working Group. http://www.w3.org/XML/Group/Query
16) Microsoft: XQL. http://www.w3.org/TandS/QL/QL98/pp/xql.html
17) Deutsch, A., et al.: XML-QL:A Query Language for XML. http://www.w3.org/TR/NOTE-xml-ql/
18) Chamberlin, D.D., et al.: Quilt: An XML Query Language for Heterogeneous Data Sources, *3rd International Workshop on the Web and Databases*, Vol.WebDB2000, pp.53–62 (2000).
19) W3C: XQuery 1.0: An XML Query Language (XQuery). http://www.w3.org/TR/xquery/
20) W3C: W3C XML Syntax for XQuery 1.0 (XQueryX). http://www.w3.org/TR/xqueryx
21) Forgy, C.L.: Rete: A Fast Algorithm for the Many Pattern/Many object Pattern Match Problem, *Artificial Intelligence*, Vol.19, pp.17–37 (1982).
22) 吉川正俊ほか：オブジェクト関係データベースを用いた XML 文書の格納と検索，情報処理学会論文誌：データベース，Vol.40, No.SIG3, pp.115–131 (1999).

付　　録

A.1　Q5/Q6 のクエリ

KF-QL では陽に順序指定できない代わりに，そのパス名とバインドさせる ⟨kf:as path="$x"⟩ の表記により,「パス名」そのものを「文字列」として取得できる．この文字列を正規表現で [] 内の指標をマッチさせる，という少々回りくどいやりかたで行っている．

Q5: /PLAY/ACT[2]

```
⟨newTag⟩
⟨kf:query xmlns:kf=" "⟩
⟨kf:select⟩
 ⟨result⟩$path⟨/result⟩
⟨/kf:select⟩
⟨kf:from path="uix://root"⟩
 ⟨PLAY⟩
  ⟨ACT⟩⟨kf:as path="$path"/⟩⟨/ACT⟩
 ⟨/PLAY⟩
⟨/kf:from⟩
⟨kf:where⟩
 ⟨kf:cmp op="like" param1="$path" param2=".*ACT.2."/⟩
⟨/kf:where⟩
⟨/kf:query⟩
⟨/newTag⟩
```

Q6: (/PLAY/ACT)[2]/TITLE

```
⟨newTag⟩
⟨kf:query xmlns:kf=" "⟩
⟨kf:select⟩
 ⟨result⟩$title⟨/result⟩
⟨/kf:select⟩
⟨kf:from path="uix://root"⟩
 ⟨PLAY⟩
  ⟨ACT⟩
   ⟨kf:as path="$path"/⟩
   ⟨TITLE⟩$title⟨/TITLE⟩
  ⟨/ACT⟩
 ⟨/PLAY⟩
⟨/kf:from⟩
⟨kf:where⟩
 ⟨kf:cmp op="like" param1="$path" param2=".*PLAY.ACT.2."/⟩
⟨/kf:where⟩
⟨/kf:query⟩
⟨/newTag⟩
```

A.2　データ抽出．仮想的データの場合あり

「各シーンのタイトルを列挙せよ」

```
⟨newTag⟩
 ⟨kf:query xmlns:kf="AAA"⟩
  ⟨kf:select⟩
   ⟨title⟩$title⟨/title⟩
  ⟨/kf:select⟩
  ⟨kf:from path="uix://root"⟩
   ⟨PLAY⟩
    ⟨ACT⟩
     ⟨SCENE⟩
      ⟨TITLE⟩$title⟨/TITLE⟩
      ⟨SPEECH/⟩
     ⟨/SCENE⟩
    ⟨/ACT⟩
   ⟨/PLAY⟩
  ⟨/kf:from⟩
 ⟨/kf:query⟩
⟨/newTag⟩
```

A.3　条件付けデータ集計から，仮想的データ作成

「各シーンのタイトルに “IV” を含むもののタイトルを集計し，タイトルとその回数のリストを作成」

```
⟨newTag⟩
 ⟨kf:query xmlns:kf="AAA"⟩
  ⟨kf:select⟩
   ⟨result⟩
    ⟨TITLE⟩$title⟨/TITLE⟩
    ⟨COUNT⟩$cnt()⟨/COUNT⟩
   ⟨/result⟩
  ⟨/kf:select⟩
  ⟨kf:from path="uix://root"⟩
   ⟨PLAY⟩
    ⟨ACT⟩
     ⟨SCENE⟩
      ⟨TITLE⟩$title⟨/TITLE⟩
      ⟨SPEECH/⟩
     ⟨/SCENE⟩
    ⟨/ACT⟩
   ⟨/PLAY⟩
  ⟨/kf:from⟩
  ⟨kf:groupBy vars="$title"/⟩
  ⟨kf:where⟩
   ⟨kf:cmp op="include" param1="$title"
    param2="IV"/⟩
  ⟨/kf:where⟩
 ⟨/kf:query⟩
⟨/newTag⟩
```

A.4　入れ子型クエリ

「各シーン（SCENE）において，タイトル（TITLE）に「IV」を含み，発言（SPEECH）が 100 回以上あ

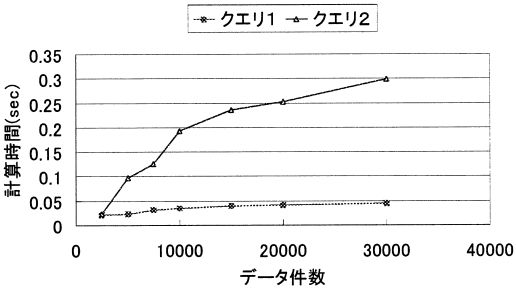


図 10 データ容量と検索速度との関係
Fig. 10 Required data size and query speed.

るシーンのタイトルと，その回数を列挙せよ」

```
⟨newTag⟩
 ⟨kf:query xmlns:kf=" "⟩
  ⟨kf:select⟩
   ⟨result⟩$elt⟨/result⟩
  ⟨/kf:select⟩
  ⟨kf:from⟩
   ⟨result⟩
  ⟨kf:query⟩
   ⟨kf:select⟩
    ⟨result⟩
     ⟨TITLE⟩$title⟨/TITLE⟩
     ⟨COUNT⟩$cnt()⟨/COUNT⟩
    ⟨/result⟩
   ⟨/kf:select⟩
   ⟨kf:from path="uix://root"⟩
    ⟨PLAY⟩
     ⟨ACT⟩
      ⟨SCENE⟩
       ⟨TITLE⟩$title⟨/TITLE⟩
       ⟨SPEECH/⟩
      ⟨/SCENE⟩
     ⟨/ACT⟩
    ⟨/PLAY⟩
   ⟨/kf:from⟩
   ⟨kf:groupBy vars="$title"/⟩
   ⟨kf:where⟩
    ⟨kf:cmp op="include" param1="$title" param2="IV"/⟩
   ⟨/kf:where⟩
  ⟨/kf:query⟩
  ⟨/result⟩
  ⟨result⟩⟨result⟩
   ⟨kf:as⟩$elt⟨/kf:as⟩
   ⟨COUNT⟩$count⟨/COUNT⟩⟨/result⟩
  ⟨/result⟩
 ⟨/kf:from⟩
 ⟨kf:where⟩
  ⟨kf:cmp op="NUMgeq" param1="$count"
    param2="100"/⟩
 ⟨/kf:where⟩
 ⟨/kf:query⟩
⟨/newTag⟩
```

A.5 スケーラビリティについて

クレーム情報をデータ（2,500 件）として，これをコピーして 5,000 件から 30,000 件までデータを増やし，検索速度の性能をテストした．結果を**図 10** に示す．クエリ 1 は「商品名称はクラッカであるトラブルの情報を検索する」であり，検索対象データは全データの約 5%である．クエリ 2 は「商品名称は牛肉であるトラブルの情報を検索する」であり，検索対象データは全データの約 20%である．

(平成 13 年 12 月 21 日受付)
(平成 14 年 10 月 7 日採録)

（担当編集委員 遠山 元道）

服部 雅一（正会員）

1988 年慶応義塾大学理工学部管理工学科卒業．1990 年同大学大学院修士課程修了．同年,（株）東芝入社．現在，同社研究開発センター知識メディアラボラトリー主任研究員．知識工学，高次推論技術の研究・開発を行い，現在，XML データベースによるナレッジマネジメントの研究および開発に従事．人工知能学会会員．

野々村克彦

1991 年東京工業大学工学部情報工学科卒業．1993 年同大学大学院理工学研究科情報工学専攻修士課程修了．同年,（株）東芝入社．現在，同社研究開発センター知識メディアラボラトリー研究主務．知識処理技術，XML データベースによるナレッジマネジメントの研究および開発に従事．

金輪 拓也

1996 年神戸大学工学部情報知能工学科卒業．1998 年同大学大学院自然科学研究科情報知能工学専攻修士課程修了．同年,（株）東芝入社．現在，同社研究開発センター知識メディアラボラトリーにおいて，XML データベースによるナレッジマネジメントの研究および開発に従事．日本 OR 学会会員．

末田 直道（正会員）

1973 年武蔵工業大学経営工学科卒業．同年東京芝浦電気（株）（現（株）東芝）入社．同社の研究・開発部門にて知識処理技術，高次推論，ナレッジマネジメント，XML データベース等の研究に従事．現在，大分大学工学部知能情報システム工学科教授．博士（工学）．人工知能学会，電子情報通信学会，AAAI 各会員．