DTD를 이용한 XML 데이타에 대한 질의 최적화 기법

(The Query Optimization Techniques for XML Data using DTDs)

정 태 선 † 김 형 주 ††
(Tae-Sun Chung) (Hyoung-Joo Kim)

요 약 XML이 웹상에서 정보 교환의 표준으로 채택되면서 XML을 데이타베이스의 데이타로 보고 정보를 추출하는 분야가 주목받고 있다. XML은 그래프 기반의 비정형 데이타(semistructured data) 모델과 매우 비슷하기 때문에 XML 데이타를 그래프 기반의 비정형 데이타 모델로 매핑한 후, 이에 대하여 질의를 처리할 수 있다. 본 논문에서는 XML 데이타에 대하여 스키마 정보를 가지는 DTD(Document Type Definition)를 이용한 질의 최적화 기법을 제안한다. 제안하는 기법은 인덱싱 효과를 내면서도 기존 방법에 비하여 부가의 메모리를 적게 필요로하고, 입력 데이타의 구조를 그대로 유지하기 때문에 다양한 형태의 질의를 효율적으로 처리할 수 있다. 간단한 예제 데이타베이스에 대하여 제안하는 기법의 실험 결과를 보였다.

Abstract As XML has become an emerging standard for information exchange on the World Wide Web it has gained attention in database communities to extract information from XML seen as a database model. Data in XML can be mapped to semistructured data model based on edge-labeled graph, and queries can be processed against it. Here, we propose new query optimization techniques using DTDs(Document Type Definitions) which have the schema information about XML data. Our techniques reduce the large search space significantly while at the same time requiring less memory compared to the traditional index techniques. Also, as they preserve source database's structure, they can process many kinds of complex queries. We implemented our techniques and provided preliminary performance results.

1. 서 론

최근에 XML[1]이 웹상의 문서 교환의 표준으로 채택되면서 XML을 데이타베이스의 데이타로 보고 정보를 추출하는 분야가 주목받고 있다. 즉, XML은 스스로 자신을 묘사하는(self describing) 성질을 가지므로, 우리는 네트워크 상에 분산된 이질의(heterogeneous) 정보 근원으로부터 질의를 수행하고 필요한 정보를 추출해 낼 수 있다.

· 본 연구는 두뇌한국21 사업에 의하여 지원 받았음.
† 비 회 원 : 서울대학교 컴퓨터공학부
tschung@papaya.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
hjk@oopsla.snu.ac.kr
논문접수 : 2000년 3월 17일
심사완료 : 2001년 7월 18일

이러한 XML 데이타는 레이블을 가진 그래프(labeled-edge) 모델을 기반으로 하는 비정형 데이타(semi-structured data) 모델과 매우 비슷한 성격을 가진다. 비정형 데이타 모델은 XML 데이타와 같이 데이타 안에 스키마 정보를 포함함으로써 웹 데이타와 같이 특별히 규칙적인 형태가 없고, 또 자주 그 형태가 변하는 데이타의 처리에 있어서 유리하다.

이렇게 그래프 기반의 비정형 데이타 모델은 데이타 모델링에 있어서 융통성을 가진다는 장점을 가지지만 스키마 정보가 없거나, 불완전하기 때문에 대개는 특정 질의를 탐색하기 위한 탐색 공간이 매우 커진다는 단점을 가진다. 이러한 탐색 공간을 줄이려는 시도로는 [2,3,4,5,6] 등이 있다. 그런데 XML 데이타는 기존 비정형 데이타 처리 모델과는 달리 DTD라는 스키마 정보를 미리 제공하기 때문에 이 정보를 이용하면 그래프의 탐색

공간을 줄여서 질의 최적화에 이용할 수 있다.

예를 들면, 다음과 같은 DTD[1]가 주어진 상태에서 질의 Q1을 처리한다고 하자.

<!ELEMENT MLB (National|American)+>
<!ELEMENT National (East|Central|West)>
<!ELEMENT American (East|Central|West)>
<!ELEMENT East (stadium?, name, player+)>
<!ELEMENT Central (stadium?, name, player+)>
<!ELEMENT West (stadium?, name, player+)>
<!ELEMENT Player (nickname?, name, ERA?, AVG?, win?)>
Q1: select MLB.National.Central.Player.name
where MLB.National.Central.Player.nickname = "Big Mac"

주어진 질의는 MLB에서 내셔널리그 중부지구 팀의 선수 중 별명을 "Big Mac"을 가지는 선수의 이름을 출력하는 질의이다. 기존의 비정형 데이타 모델에서는 아무 인덱스가 없다고 가정할 경우 먼저 MLB.National.East.Player.nickname의 경로를 만족하는 객체를 모두 찾은 후 조건을 만족하는 객체를 리턴 한다. 그런데 XML 데이타 처리에 있어서는 주어진 질의에 대하여 DTD가 다음과 같은 정보를 제공한다.

1. MLB은 National과 American으로 구분된다.
2. National 리그는 동부, 서부, 중부지구 중의 하나이다.
3. player는 nickname 항목을 가질 수도 있고, 안 가질 수도 있다.

이러한 정보를 이용하면 주어진 질의는 National리그 중부지구의 nickname을 가진 선수에서 주어진 조건을 만족하는 선수만 탐색하면 되므로 탐색 범위를 크게 줄일 수 있다. 본 논문에서는 이렇게 DTD를 이용한 질의 최적화 기법을 제안한다.

2. 관련 연구

2.1 데이타 모델과 질의 언어

XML은 최근에 활발한 연구가 수행되고 있는 그래프 기반의 비정형 데이타 모델(semistructured data model)의 한 인스턴스로 간주될 수 있기 때문에 비정형 데이타 모델의 여러 연구 결과들을 적용할 수 있다. 본 논문에서도 이와 같이 XML 데이타를 비정형 데이타 모델의 대표격인 OEM[7]에 매핑하여 질의를 수행한다. 이 밖의 방법으로는 RDBMS에 XML 데이타를 저장하여 RDBMS가 제공하는 질의 언어를 이용하는 방법[8, 9]이 있다. 이 방법은 XML과 같이 스키마가 비정규적인 데이타를 의미정보를 최대한 유지하면서 테이블 형태로 변환시키는 기법이 핵심이 되는데 본 논문과는 접근 방법이 다르다고 할 수 있다.

비정형 데이타 모델의 질의 언어는 경로식에 정규식을 포함한 정규 경로식을 기반으로 한다. 이렇게 정규 경로식을 기반으로 한 언어에는 Lorel[10], XML-QL[11], Quilt[12], XQuery[13] 등이 있다. 우리는 정규 경로식으로 도달되는 모든 객체를 찾는 기법을 제안하는데 이 기법은 정규 경로식을 기반으로 하는 모든 언어에 적용될 수 있다.

2.2 질의 처리 기법

비정형 데이타 모델에서 기존에 제안된 질의 최적화 기법은 주로 스키마 정보를 어떻게 효과적으로 뽑아내어 질의 처리에 이용하는지 하는 문제가 핵심이었다. 이러한 시도로는 [3,4,5,6,14]가 있다.

이러한 연구들은 하나의 정규식으로 이루어진 질의만을 처리할 수 있거나 특정 질의에 대해서만 효율성을 기대할 수 있거나, 또는 인덱스 자체의 생성, 유지를 위한 부담이 큰 단점이 있다.

우리의 연구는 XML이 제공하는 DTD[2]를 이용하여 질의 처리기에 힌트를 주는 기법으로 기존 방법보다 효율적으로 정보를 추출하고, 인덱스 자체를 위한 부담도 덜하다. 또한 정규식 여러 개를 포함한 다양한 형태의 질의에 적용될 수 있다.

3. 기본 개념

3.1 비정형 데이타 모델

본 논문에서는 그래프 기반의 비정형 데이타 모델 중 비정형 데이타 모델을 대표한다고 할 수 있는 OEM (Object Exchange Model)을 기반으로 한다. 그렇지만 본 논문에서 제안하는 기법은 그래프 기반의 비정형 데이타 모델에서 모두 적용될 수 있을 것이다.

OEM 데이타 모델은 레이블이 있는 그래프로 표현된다. 이때 그래프의 노드는 객체를 나타내고, 에지는 객체의 애트리뷰트를 나타내는데 OEM 객체는 다음과 같이 두 가지 종류로 구분된다.

- 원자 객체(atomic object): 기존 DBMS 모델에서 실제 데이타 값에 해당된다. 원자 객체의 값(value)

1) <!ELEMENT element_name (#PCDATA)> 형태는 생략하였다.

2) XML데이타가 DTD를 제공하지 않을 수 있지만 [15]의 방법으로 DTD를 만들어낼 수 있다.

은 임의의 데이타 타입으로 정수형(integer), 문자열(string), 이미지(image), 소리(sound) 등이 될 수 있다.

- 복합 객체(complex object): 기존 DBMS 모델에서 스키마 정보를 나타낸다고 할 수 있다. 복합 객체의 값은 (링크, 객체 식별자)의 집합으로 구성된다.

그림 1은 MLB(Major League Baseball)을 OEM 모델로 표현한 예제 데이타베이스를 나타낸다. 여기서 객체 o3, o4 등은 원자 객체에 해당하고, 객체 o5, o10 등은 복합 객체에 해당한다.

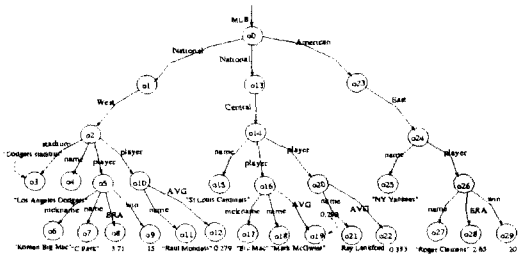


그림 1 OEM 모델

3.2 XML

XML(eXtensible Mark-up Language)은 최근에 웹상에서 문서 교환의 표준으로 채택된 문서 정의 언어이다. XML은 그 문서의 구조를 묘사하는 DTD와 실제 XML 데이타로 이루어지는데 XML 데이타는 요소(element)라는 항목이 연속된 형태로 구성되고, 중첩될 수 있다. 또한, 각 요소는 애트리뷰트를 가질 수 있다.

이러한 XML은 데이타 자체에 스키마 정보를 담는다는 성격을 가지므로 OEM과 같은 비정형 데이타 모델과 비슷하고 따라서 비정형 데이타의 그래프 모델로 매핑될 수 있다. 단, 본 논문에서는 OEM 모델에 없고 XML에 있는 순서(order)는 무시하고, XML 요소의 애트리뷰트로 ID와 IDREF[1]만을 가정한다. 일반 애트리뷰트는 [16]의 방법으로 애트리뷰트를 가지지 않는 XML로 정의될 수 있다.

다음은 앞 절의 OEM 모델과 같은 내용을 표현하는 XML 데이타중 일부를 나타낸다.

```
<MLB>
  <National>
    <West>
      <stadium> Dodgers stadium </stadium>
      <name> Los Angeles Dodgers </name>
      <player>
        <nickname> Korean Big Mac </nickname>
        <name> C Park </name>
        <ERA> 3.71 </ERA>
        <win> 15 </win>
      </player>
...
</MLB>
```

4. DTD 요소의 구분

DTD는 XML 데이타 요소(element) 구조에 대한 정규식(regular expression)을 표현함으로써 XML 데이타에 대한 구조 정보를 제공한다. 본 논문에서는 DTD를 가지는 XML 문서를 가정하고, [17]에서의 DTD에 대한 제약 조건을 적용함으로써 DTD를 (n: P)의 짝의 집합(단, N은 요소 이름의 집합, n∈N이고, P는 N에 대한 정규식 또는 PCDATA, 즉 문자열이다.)으로 추상화한다.

4.1 DTD 오토마타

DTD가 제공하는 정보 중에 질의 처리기에게 힌트가 될 수 있는 부분은 XML 데이타 인스턴스를 구분해주는 부 엘리먼트이다. 즉, 모든 데이타 인스턴스가 같은 형태로 가지는 부 엘리먼트는 질의 처리기에게 정보가 되지 못한다.

그러므로 본 논문에서는 DTD로부터 질의 처리시 필요한 정보 만을 뽑아내기 위하여 다음과 같은 완화된 정규식을 정의한다.

정의 1 (완화된 정규식(relaxed regular expression)) 완화된 정규식은 다음의 다섯 가지 규칙에 의하여 구해진다.

1. r1, r2 ⇒r1, r2
2. r1|r2 ⇒r1|r2
3. $r^+ \Rightarrow r$
4. $r^* \Rightarrow r^+ | \perp \Rightarrow r | \perp$ (3에 의하여)
5. r? ⇒r| ⊥

예제 1 앞의 MLB DTD에서 player에 대한 항목은 (player: (nickname?, name, ERA?, AVG?, win?))으로 나타낼 수 있고, 완화된 정규식을 적용하면 (player: ((nickname|⊥), name, (ERA|⊥), (AVG|⊥), (win|⊥)))이 된다.

DTD 오토마타는 다음과 같은 방법으로 생성된다. DTD의 각 요소 (n_i, P_i)에 대하여 P_i에 완화된 정규식을 적용한 결과를 (n_i, P_i')이라 한면, 새로운 정규식 $n_i P_i'$[3]에 대하여 오토마타 A_i를 알고리즘 1에 따라서

알고리즘 1 DTD 오토마타의 구성

입력: 완화된 정규식 $r = n_i P_i'$
출력: 오토마타 M_i
procedure Make_DTD_Automata(regular expression r)
if $r = a(a \in \Sigma)$ **then**
  그림 2와 같이 오토마타 M 생성;
  return M;
else if $r = r_1 | r_2$ **then**
  $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1) \leftarrow$ Make_DTD_Automata(r_1);
  $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2) \leftarrow$ Make_DTD_Automata(r_2);
  오토마타 M_1과 M_2로부터 새로운 오토마타
  $M_i = (Q_1 - \{q_1\} \cup Q_2 - \{q_2\}, \Sigma_1 \cup \Sigma_2, \delta, [q_1, q_2], F_1 \cup F_2)$를
  생성, 이때 δ는 다음과 같이 정의됨
  1. $\delta(q, a) = \delta_1(q, a)$ for $q \in Q_1 - \{q_1\}$ and $a \in \Sigma_1$,
  2. $\delta(q, a) = \delta_2(q, a)$ for $q \in Q_2 - \{q_2\}$ and $a \in \Sigma_2$,
  3. $\delta([q_1, q_2], a) = \delta_1(q_1, a)$ *where* $a \in \Sigma_1$,
  4. $\delta([q_1, q_2], a) = \delta_2(q_2, a)$ *where* $a \in \Sigma_2$;
else { $r = r_1, r_2$}
  $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1) \leftarrow$ Make_DTD_Automata(r_1);
  $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2) \leftarrow$ Make_DTD_Automata(r_2);
  M_1의 종료 상태 F_1을 $f_1, f_2, \ldots, f_m (m \geq 1)$이라 하자. 오토마타
  M_1과 M_2로부터 새로운 오토마타
  $M_i = (Q_1 - F_1 \cup Q_2 - \{q_2\} \cup \{[f_1, q_2], [f_2, q_2], \ldots, [f_m, q_2]\}, \Sigma_1 \cup \Sigma_2, \delta, q_1, F_2)$
  를 생성, 이때 δ는 다음과 같이 정의됨
  1. $\delta(q, a) = \delta_1(q, a)$ for $q \in Q_1 - F_1, \delta_1(q, a) \neq f_k$(*where* $1 \leq k \leq m$), and $a \in \Sigma_1$,
  2. $\delta(q, a) = \delta_2(q, a)$ for $q \in Q_2 - \{q_2\}$ and $a \in \Sigma_2$,
  3. $\delta([f_k, q_2], a) = \delta_2(q_2, a)$ for *all* k(*where* $k = 1, 2, \ldots, m$) and $a \in \Sigma_2$,
  4. $\delta(q_f, a) = [f_k, q_2]$for *all* q_f *which satisfies*
     $\delta_1(q_f, a) = f_k$(*where* $1 \leq k \leq m$) and $a \in \Sigma_1$;
end if
return M_i;

생성한다. 여기서 생성한 오토마타는 DTD가 제공하는 정보를 이용하여 질의 탐색 영역을 구분함으로써 궁극적으로 질의 최적화에 이용하게 된다.

정리 1 알고리즘 1에 의해 생성되는 오토마타 M은 항상 존재하고, M이 받아들이는 언어를 L(M)이라고 하고, 해당 정규식 r이 나타내는 언어를 L(r)이라고 하면 L(M)=L(r)이다.

증명은 생략한다.

예제 2 앞의 player에 대한 DTD에서 완화된 정규식을 적용한 결과인 (player: ((nickname|⊥), name, (ERA|⊥), (AVG|⊥), (win|⊥)))에 대하여 알고리즘 1을 수행하면 그림 3과 같다.

4.2 DTD 오토마타를 이용한 DTD 요소의 구분

이 절에서는 앞 절에서 구한 각 요소(element)에 대한 DTD 오토마타를 이용하여 각 요소를 구분한다. DTD 오토마타는 완화된 정규식에서 생성되었으므로 DTD 정규식의 조합(concatenation)과 OR 조건만을 나타내게 된다. 이중에서 OR 조건의 분기점을 나타내는 부분이 탐색 영역을 구분하는 분기점이 되므로 이러한 분기점의 레이블을 기억하면 이를 통하여 요소를 구분할 수 있다.

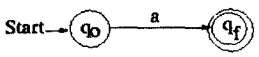


그림 2 r=a 형태

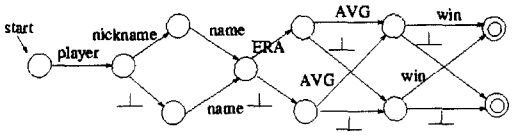


그림 3 DTD 오토마타

3) $n_i P_i'$는 n_i, P_i와 같이 조합(concatenation)을 나타낸다.

알고리즘 2 DTD 요소(element) 종류 구분 트리 생성

```
1:  입력: M_k=(Q_k, Σ_k, δ_k, q_k, F_k) (for k=1,2,...,n)
2:  출력: 분류 트리 T_k (for k=1,2,...,n)
3:  procedure Make_classification_tree(state q, automaton M_k)
4:  if q∈F_k then
5:     make a vertex q' corresponding to a state q;
6:     return q';
7:  else
8:     if transition(q) has more than two states then
9:        make a vertex q' corresponding to a state q;
10:       let T be a tree rooted q' and having children of
            Make_classification_tree(w, M_k) for all w where
            w∈transition(q) with edges labeled a in the transition;
11:       return T;
12:    else
13:       q←δ_k(q, a);
14:       No_effect_label_k=No_effect_label_k∪a;
15:       Make_classification_tree(q, M_k);
16:    endif
17: endif
```

알고리즘 2는 이러한 레이블을 저장하는 트리를 구성하는 알고리즘을 나타낸다. 생성된 트리의 루트에서 각 리프 노드까지 레이블의 집합이 요소를 구분하는 정보가 된다. 알고리즘 2의 입력은 DTD 오토마타 $M_k=(Q_k, \Sigma_k, \delta_k, q_k, F_k)$이고 여기서 k는 각 요소에 대한 오토마타중 k 번째를 나타낸다. 알고리즘 2는 오토마타의 시작 상태(start state)에서 종료 상태(final state)까지 재귀적으로 탐색하면서 트리를 생성한다. 알고리즘에서 transition(state)는 $\delta_k(q, a)=p$의 전이 함수가 있을 때 transition(q)=p를 리턴하는 함수를 나타낸다. $No_effect_label_k$는 DTD의 k번째 요소에 대하여 그가 가지는 레이블이 요소를 구분하는데 영향을 미치지 않는 레이블의 집합을 가지는 변수이다. 예를 들어 player에 대한 No_effect_label은 {player,name}이 되는데 이것은 player의 애트리뷰트 중에서 {nickname, ERA,AVG, win}은 선수 중에서 가질 수도 있고 안 가질 수도 있기에 질의 처리기에 정보가 되지만 {player, name}은 어느 선수나 전부 가지는 애트리뷰트이므로 질의 처리 과정에서 정보가 되지 못한다. 따라서 필요한 정보만을 뽑아내기 위하여 이 단계에서 No_effect_label을 구한다.

그림 4는 알고리즘 2에 따라 생성한 DTD 오토마타의 player에 대한 분류 트리와 해당 분류 테이블을 나타낸다. 예를 들면 DTD의 한 요소인 player는 그가 가지는 레이블에 따라 {nickname, ERA, AVG, win}, {nickname, ERA, AVG}, {nickname, AVG, win} 등의 16개의 그룹으로 나누어진다.

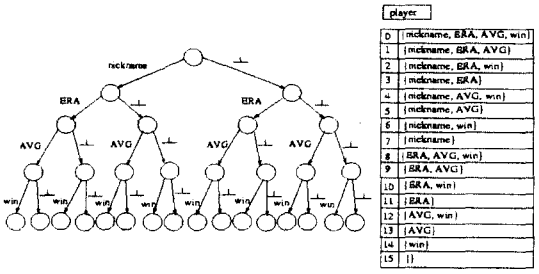


그림 4 DTD 요소 분류 트리와 분류 테이블

5. 질의 최적화 기법

여기서는 앞 절에서 구한 분류 테이블을 이용하여 입력 데이타 그래프에 DTD 정보를 첨가하여 질의 처리기가 그래프 탐색 범위를 줄이는 두 가지 기법을 제안한다. 먼저 질의에 존재하는 경로식(path expression)을 다음과 같이 정의한다.

정의 2 (정규 경로식(Regular Path Expression)) 정규 경로식은 H.P 형태로

1. H는 객체 이름 또는 객체를 표시하는 변수,
2. P는 OEM 그래프의 레이블에 대한 정규식이다. 즉, P=label|(P|P)|(P.P)|P*이다.

정의 3 (단순 정규 경로식(Simple Regular Path Expression)) 단순 정규 경로식은 $H.p_1.p_2....p_n$의 형태로

1. H는 객체 이름 또는 객체를 표시하는 변수,

2. $p_i(1 \le i \le n)$는 OEM 그래프 상의 레이블, 또는 임의 길이의 레이블 연속을 나타내는 *이다.

5.1 NodeInfo

NodeInfo 기법은 각 객체에 대하여 분류 정보를 질의 처리기에게 제공한다. 즉, 이 분류 정보는 변수 node_info에 저장되는데 그림 5는 그림 1의 OEM 그래프에서 node_info와 다음 절에서 다룰 merge_node_info가 첨가된 그림이다. 예를 들어, 노드 o5는 node_info 값으로 2를 가지고 이 값은 그림 4의 분류 테이블에서 레이블 집합 {nickname,ERA,win}의 인덱스에 해당한다.

NodeInfo의 생성 비용은 최악의 경우(worst case)에 k를 각 DTD 요소에 대한 최대 그룹 수라고 하고, n을 노드 수라고 하면 O(kn)이다. 이것은 m 개의 에지와 n 개의 노드로 이루어진 그래프에서 O(mlog n)의 복잡도를 가지는 1-index[6]와 최악의 경우 지수승의 복잡도를 가지는 DataGuides[4,5]에 비하여 우수하다.

NodeInfo 기법은 단순 정규 경로식 만을 포함한 질의를 처리할 수 있다. 먼저 단순 정규 경로식에 대하여 classification_info 변수를 정의한다.

정의 4 $H.p_1.p_2.....p_n$를 질의에 존재하는 단순 정규 경로식이라고 하자. $p_i(1 \le i \le n)$의 classification_info는 다음과 같이 정의된다.

1. p_{i+1}=*이거나, $p_{i+1} \neq$*이고, $p_{i+1} \in$ No_effect_label 일 때, {},
2. $p_{i+1} \neq$*이고, $p_{i+1} \notin$ No_effect_label[4] 일때, { p_{i+1}}.

단순 정규 경로식의 classification_info는 질의의 종류에 대한 정보를 가진다. 다음으로 질의 처리기의 관점에서 다음과 같이 객체 탐색을 정의한다.

정의 5 (객체 탐색) 객체 탐색은 객체의 값을 읽는 것을 의미한다. 이때 탐색되는 객체가 복합 객체의 경우, 객체의 값은 <label, object-id, (merge_)node_info of the object-id>이다.

NodeInfo 방법에서 질의 처리는 다음과 같이 수행된다. 단순 정규 경로식 $H.p_1.p_2.....p_n$에 의하여 도달되는 모든 객체를 찾는 과정에서, 질의 처리기가 w로의 에지를 가지는 객체 v를 탐색하고, 해당 레이블이 $p_i(1 \le i \le n-1)$라고 하면, 탐색이 객체 w로 확장되는 경우는 다음 식을 만족하는 경우이다. classification_info of p_i-classification_table[element_label of w][w.node_info] = ∅.

4) No_effect_label은 모든 k에 대한 No_effect_label$_k$의 합집합이다.

예제 3 단순 정규 경로식 MLB.National.West.player.nickname을 처리한다고 하자.

이 경로로 도달되는 모든 객체를 찾기 위해서는 그림 5에서 객체 o0, o1, o2, o5, o6가 탐색되고 "Korean Big Mac"이 리턴된다. 기존의 일반 방식에서는 객체 o2의 자식인 객체 o10이 탐색된 후 그 객체가 nickname 애트리뷰트를 가지지 않음을 알고, 탐색이 종료된다. 그런데 NodeInfo 방법에서는 객체 o2를 탐색하는 중 객체 o10에 대한 값을 읽을 때 p_i=player의 classification_info의 값이 {nickname}이고 classification_table[element_label of o10][o10.node _info]의 값이 {AVG}이다. 그러므로 classification_info of p_i-classification_table [element_label of o10][o10. node_info]={nickname} ≠ ∅이 되어 탐색이 종료된다.

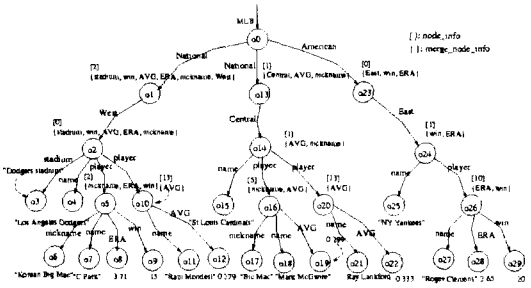


그림 5 (merge_)node_info가 첨가된 OEM 그래프

5.2 MergeNodeInfo

MergeNodeInfo 방법은 변수 merge_node_info가 그 값으로 대상 객체 자체에 대한 정보만을 가지는 것이 아니라 대상 객체 하위의 모든 객체에 대한 정보를 가지다. 그러므로 merge_node_info는 대상 객체 자신의 node_info와 그 자식 객체의 merge_node_info의 합집합으로 구해진다.

MergeNodeInfo의 크기는 t를 DTD에 존재하는 요소 이름의 집합과 No_effect_label의 레이블의 집합의 차집합이라 할 때 O(tn)이다. DataGuides에서는 그 크기가 데이타베이스 크기의 지수승에 비례할 수 있다.

MergeNodeInfo 기법은 단순 경로식을 포함한 질의와 정규 경로식을 포함한 질의 모두를 처리할 수 있다. 먼저 단순 정규 경로식에 대하여 p_i에 대한 merge_classification_info를 다음과 같이 정의한다.

정의 6 $H.p_1.p_2.....p_n$를 질의에 존재하는 단순 정규 경로식이라고 하자. $p_i(1 \le i \le n-1)$에 대한 merge_classification_info는

$\bigcup_{k=i}^{n-1}$classification_info of p_i

로 정의된다.

MergeNodeInfo 기법에서의 질의 처리는 다음과 같이 수행된다. 단순 정규 경로식 $H.p_1.p_2.....p_n$에 의하여 도달되는 모든 객체를 찾는 과정에서 질의 처리기가 객체 w로의 에지를 가지는 객체 v를 탐색한다고 하고, 그 레이블을 $p_i(1 \le i \le n-1)$로 하자. 객체 w로 탐색이 확장되는 경우는 merge_classification_info of p_i - w. merge_node_info= $\emptyset$ 조건을 만족하는 경우이다.

예제 4 다음의 단순 정규 경로식 MLB.*.West.stadium를 처리한다고 하자.

일반적 하향식 방법에서는 주어지 경로에 의하여 도달되는 모든 객체를 찾기 위해서 '*' 다음의 West를 찾기 위하여 전체 그래프가 탐색된다. 그렇지만 Merge NodeInfo 방법에서는 그림 5의 객체 o0를 탐색할 때 객체 o13에 대하여 p_i=*이고, merge_classification_info={West, stadium}이다. 그리고 o13.merge_node_info of p_i의 값은 {Central,AVG,nickname}이다. 따라서 merge_classification_info of p_i-w.merge_node_ info={West, stadium} $\neq \emptyset$이므로 탐색이 종료된다. 객체 o23의 하위 영역도 같은 이유로 탐색에서 제외된다.

MergeNodeInfo 기법은 '|' 연산이 없는[5] 정규 경로식을 가지는 질의에 이용될 수 있다. 먼저 다음의 변수를 정의한다.

정의 7 질의 처리기가 객체 w로의 에지를 가지는 객체 v를 탐색할 때, 변수 P,Q,R이 다음과 같이 정의된다.

- P: 질의에 존재하는 레이블의 집합과 No_effect_label의 레이블 집합 간의 차집합.
- Q: 노드 w에 대한 merge_node_info의 레이블의 집합.
- R: 루트에서 노드 w까지의 경로에 존재하는 레이블의 집합.

여기서 P-(Q R)= $\emptyset$ 조건을 만족하면 노드 v에서 노드 w로 탐색이 확장된다.

예제 5 가상의 참고 서적에 대한 데이타베이스에서 정규 경로식 Bib.paper(.section)*. figure를 처리한다고 하자.

객체 v가 데이타베이스에서 한 섹션을 나타내고, 그의 하위 객체 전체는 그림을 가지지 않는다고 가정하자. 그러면 질의 처리기가 객체 v를 탐색할 때 P={paper, section, figure}, Q={section} 이고 R={paper, section} 일 수 있다. 이때 P-(Q $\cup$R)={figure} $\neq \emptyset$이 되어서 탐색이 종료된다.

6. 평 가

6.1 비용 계산

비정형 데이타 모델에서는 객체 클러스터링 효과를 얻기 힘들기 때문에[4,18] 각 방법의 비용을 검사하기 위하여 객체 탐색 수를 계산하는 것이 합리적이다. 따라서 naive한 방법과 NodeInfo, MergeNodeInfo의 객체 탐색 수를 계산하면 다음과 같다. 단, 단순 경로식 $H.p_1.p_2.....p_n$을 처리한다고 하고, $p_i(0 \le i \le n)$, H에 매핑되는 객체를 나타내는 변수를 각각 vp_i, vH라 하자. $|vp_i|$은 vp_i에 바인딩되는 객체 수를 나타낸다. 예를 들어 그림1에서 단순 경로식 MLB.National.West.stadium에 대하여 p_i=National이고, vp_1은 객체 {o1,o13}을 나타내는 변수이다. Fanout(vp_i,l)은 변수 vp_i에 바인드되는 객체에서 레이블 l로 도달되는 평균 객체 수이다.

$$
\begin{aligned}
Cost_{naive} &= Fanout(vH, p_1) + |vp_1| Fanout(vp_1, p_2) + \ldots \\
&\quad + |vp_{n-1}| Fanout(vp_{n-1}, p_n) \\
&= Fanout(vH, p_1) + Fanout(vH, p_1) \\
&\quad Fanout(vp_1, p_2) + \ldots + \\
&\quad Fanout(vH, p_1) Fanout(vp_1, p_2) \ldots \\
&\quad Fanout(vp_{n-2}, p_{n-1}) Fanout(vp_{n-1}, p_n)
\end{aligned}
$$

$$
\begin{aligned}
Cost_{NodeInfo} &= Fanout(vH, p_1) nc_1 + Fanout(vH, p_1) \\
&\quad Fanout(vp_1, p_2) nc_1 nc_2 + \ldots \\
&\quad + Fanout(vH, p_1) Fanout(vp_1, p_2) \ldots \\
&\quad Fanout(vp_{n-2}, p_{n-1}) Fanout(vp_{n-1}, p_n) \\
&\quad nc_1 nc_2 \ldots nc_n
\end{aligned}
$$

$$
\begin{aligned}
Cost_{MergeNodeInfo} &= Fanout(vH, p_1) mc_1 + Fanout(vH, p_1) \\
&\quad Fanout(vp_1, p_2) mc_1 mc_2 + \ldots \\
&\quad + Fanout(vH, p_1) Fanout(vp_1, p_2) \ldots \\
&\quad Fanout(vp_{n-2}, p_{n-1}) Fanout(vp_{n-1}, p_n) \\
&\quad mc_1 mc_2 \ldots mc_n
\end{aligned}
$$

여기서 nc_i, mc_i는 각각 NodeInfo와 MergeNodeInfo 방법에 의하여 각 단계에서 탐색이 생략되는 비율로 $0 \le nc_i \le 1, 0 \le mc_i \le 1$이고 $nc_i \le mc_i$이다. 따라서 항상 $Cost_{MergeNodeInfo} \le Cost_{NodeInfo} \le Cost_{naive}$을 만족한다.

6.2 실험 결과

제안된 기법이 질의 처리시에 효과적임을 보이기 위하여 약 3000 줄의 Java 코드로 MergeNodeInfo와 NodeInfo를 구현하였다. 그리고 널리 알려진 질의 처리 기법인 DataGuide를 구현하여 비교하였다. 질의 처리시의 데이타베이스로는 그림 1을 확장한 MLB 데이타베이스[6]를 이용하였다. MLB 데이타베이스는 60개의 팀과 2400명의 선수를 포함하는 14646 개의 객체로 구성된다.

DataGuide는 근원 데이타베이스를 비결정적 오토마타

5) '|' 연산을 가진 정규 경로식은 '|' 연산을 가지지 않는 두개 이상의 정규 경로 식으로 변환될 수 있다.

6) 이 데이타베이스는 프로그램 기법으로 얻어진 인공적인 데이타베이스이지만 실제 MLB 데이타베이스와 비슷하다.

(nondeterministic automata)로 보았을 때 이것을 결정적 오토마타(deterministic automata)로 변환시킴으로써 그래프 탐색 범위를 줄이는 기법이다. 이 기법은 루트 노드에서 특정 정규식을 만족하는 객체의 집합을 효율적으로 찾지만 변수로 연결된 두개 이상의 정규식을 처리할 수 없고, 최악의 경우(worst case)에 생성 비용과 인덱스 크기가 지수승에 비례하는 단점을 가진다. 우리 실험에서는 작은 크기의 DataGuide가 만들어지는 경우에도 MergeNodeInfo와 NodeInfo의 생성 비용과 인덱스 크기가 작음을 보인다. 표 1은 생성 비용과 인덱스 크기를 나타낸다.

표 1 생성 비용과 인덱스 크기

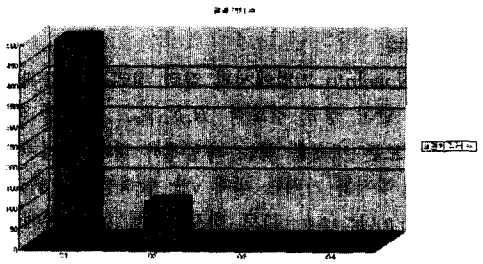
	생성 비용(초)	인덱스 크기(바이트)
DataGuide	116.397	63132
NodeInfo	37.168	38912
MergeNodeInfo	36.703	43308

다음으로 단일 정규식의 처리에 있어서 성능 비교를 보인다. 먼저 실험에 사용한 질의는 표 2와 같다.

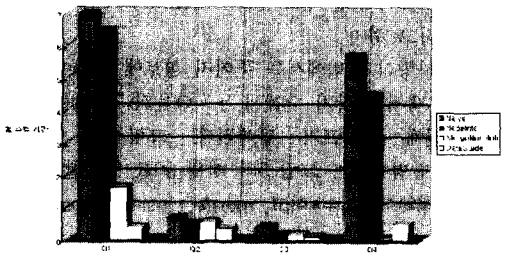
표 2 실험에 이용된 정규 경로식

Q1	MLB.*.Central.player.RBI
Q2	MLB.American.East.player.win
Q3	MLB.National.West.player.nickname
Q4	MLB.*.West.stadium

그림 6의 (a)는 주어진 데이타베이스에서 각 질의에 대하여 최종 리턴되는 객체 수를 보인다. 이것은 잠재적으로 질의 처리기가 탐색하여야 할 객체 수를 보인다. 그림 6의 (b)는 각 방법, 즉, naive한 방법, NodeInfo, MergeNodeInfo, DataGuide 방법으로 각 질의를 처리할 때의 탐색하는 객체 수를 보인다. 여기서 NodeInfo, MergeNodeInfo 방법은 naive한 방법에 비하여 탐색 범위를 현저히 줄임을 알 수 있다. DataGuide와 비교한 경우에는 질의 1,2,3의 경우에 NodeInfo와 Merge NodeInfo에 비하여 DataGuide가 좋은 성능을 보이지만, '*' 연산이 들어가는 경우에는 MergeNodeInfo가 NodeInfo방법보다 훨씬 우수하고 DataGuide에 비하여도 우수함을 알 수 있다. 이것은 '*' 연산을 처리하기 위하여 DataGuide 그래프를 대부분 탐색하여야 하기 때문이다. 따라서 DataGuide 그래프가 큰 경우에는 단일 정규식의 처리에 있어서도 MergeNodeInfo의 처리가 우수할 것이다. 그림 7은 객체 탐색시 이루어지는 페이지 I/O의 수와 실제 수행 시간을 보인다.

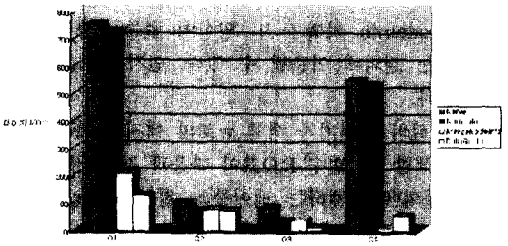


(a) 질의에 대한 결과 객체 수

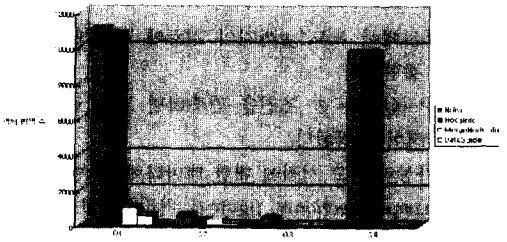


(b) 총 수행 시간

그림 6 질의의 성격과 객체 탐색 수



(a) 페이지 I/O 수



(b) 객체 탐색 수

그림 7 성능 평가

7. 결 론

본 논문에서는 그래프 기반의 비정형 데이타 모델에 매핑될 XML 데이타 대한 질의 처리에 있어서, XML이

제공하는 DTD를 이용한 두 가지의 최적화 기법 NodeInfo와 MergeNodeInfo를 제안하였다. NodeInfo는 탐색하는 객체에 대한 애트리뷰트 정보를 가지고 질의의 종류에 따라 그로부터 연결된 객체의 탐색 영역을 줄이는 방법이다. MergeNodeInfo는 대상 객체로부터 도달 가능한 모든 객체에 대한 애트리뷰트 정보를 가지고 탐색 영역을 줄인다. 이 두 가지 기법은 DTD의 완화된 정규식으로부터 얻어진 DTD 오토마타로부터 해당 정보를 추출한다. 제안하는 기법은 기존의 여러 비정형 데이타의 질의 최적화 기법에 비하여 효율적이면서도 인덱스를 위한 메모리를 덜 필요로 한다. 또한 근원 데이타베이스의 구조를 유지하기 때문에 정규식 여러 개를 포함한 복잡한 형태의 질의 처리가 가능하다.

참 고 문 헌

[1] T. Bray, J. Paoli, and C. Sperberg-McQueen,"Extensible markup language(XML) 1.0," Technical report, W3C Recommendation, 1998.

[2] J. McHugh and J. Widom,"Query optimization for XML," In Proceedings of the Conference on Very Large Data Bases, 1999.

[3] Mary Fernandez and Dan Suciu,"Optimizing regular path expressions using graph schemas," In IEEE International Conference on Data Engineering, 1998.

[4] Roy Goldman and Jennifer Widom,"DataGuides: enabling query formulation and optimization in semistructured databases," In Proceedings of the Conference on Very Large Data Bases, 1997.

[5] Svetlozar Nestorov, Jeffrey Ullman, Janet Wiener, and Sudarshan Chawathe,"Representative objects: concise representations of semistructured, hierarchical data," In IEEE International Conference on Data Engineering, 1997.

[6] Tova Milo and Dan Suciu,"Index structures for path expressions," In Proceedings of the International Conference on Database Theory, 1999.

[7] Yannis Papakonstantinou and Serge Abiteboul, "Object fusion in mediator systems," In Proceedings of the Conference on Very Large Data Bases, 1996.

[8] Alin Deutsch, Mari Fernandez, and Dan Suciu, "Storing semistructed data with STORED," In Proceedings of the ACM SIGMOD International Conference on the Management of Data, 1999.

[9] Jayavel Shanmugasundaram, H. Gang, Kristin Tufte, Chun Zhang, David DeWitt, and Jeffrey F. Naughton,"Relational Databases for Querying XML Documents: Limitations and Opportunities," In Proceedings of the Conference on Very Large Data Bases, 1999.

[10] S. Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, Janet Wiener,"The lorel query language for semistructured data," International Journal on Digital Libraries 1996.

[11] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu,"Query language for XML," In Proceedings of Eighth International World Wide Web Conference, 1999.

[12] Don Chamberlin, Jonathan Robie, and Daniela Florescu,"Quilt: An XML Query Language for Heterogeneous Data Sources," In Invited paper, WebDB, 2000.

[13] D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu,"XQuery: A Query Language for XML," Technical report, W3C Working Draft, February 2001.

[14] Dan Suciu, Mary Fernandez, Susan Davidson, and Peter Buneman,"Adding structure to unstructured data," In Proceedings of the International Conference on Database Theory, 1997.

[15] Minos Garofalakis, Aristides Gionis, Rajeev Rastogi, S. Seshadri, and Kyuseok Shim, "XTRACT : A System for Extracting Document Type Descriptors from XML Documents," In Proceedings of the ACM SIGMOD International Conference on the Management of Data, 2000.

[16] Dan Suciu,"Semistructured data and XML," In Proceedings of International Conference on Foundations of Data Organization, 1998.

[17] Y. Papakonstantinou, and P. Velikhov,"Enhancing semistructured data mediators with document type definitions," In IEEE International Conference on Data Engineering, 1999.

[18] S. Abiteboul, J. McHugh, M. Rys, V. Vassalos, and J. Weiner,"Incremental maintenance for materialized views over semistructured data," In Proceedings of the Conference on Very Large Data Bases, 1998.

정 태 선
정보과학회논문지 : 데이타베이스
제 28 권 제 2 호 참조

김 형 주
정보과학회논문지 : 데이타베이스
제 28 권 제 1 호 참조