타임스탬프된 이벤트 시퀀스를 위한 효율적인 검색 방법

이우준° 노국필 강성구 박상현
포항공과대학교 컴퓨터공학과
{crough°, noh9pil, exodus, sanghyun}@postech.ac.kr

An Effective Searching Method for Timestamped Event Sequences

Woojun Yi° Gookpil Roh Seonggoo Kang Sanghyun Park
Dept. of Computer Science and Engineering, Pohang University of Science and Technology

요 약

시퀀스로부터 원하는 패턴을 효율적으로 검색하는 것은 타임 시리즈 분석이나 네트웍 침입 탐지와 같은 응용 환경에서 필수적이다. 예로서, 특정한 이벤트가 발생할 때마다 이벤트의 유형과 발생 시각을 기록하는 네트웍 이벤트 관리 시스템을 생각해보자. 네트웍 이벤트들의 연관 관계를 발견하기 위한 전형적인 질의 형태는 다음과 같다: "CiscoDCDLinkUp이 발생한 후 20초 이내에 MLMStatusUP이 발생하며 그 후 40초 이내에 CiscoDCDLinkUP이 발생하는 모든 경우를 검색하라." 이 논문은 위와 같은 질의를 효율적으로 처리할 수 있는 방안으로 빈도수 기반, 조인 기반, 트리 순회 기반의 검색 기법들을 제시한다.

1. 서 론

시퀀스로부터 원하는 패턴을 효율적으로 검색하는 것은 타임 시리즈 분석, 멀티미디어 데이터베이스 관리, 단백질 구조 예측, 네트웍 침입 탐지와 같은 응용 환경에서 필수적이다 [1, 4]. 예로서, 복잡한 네트웍 환경을 위한 이벤트 관리 시스템을 생각해 보자. 네트웍 상에서 특정한 이벤트가 발생할 때마다 그림 1과 같이 이벤트의 유형과 발생 시각이 기록된다.

Event	Timestamp
:	:
CiscoDCDLinkUp	19:08:01
MLMSocketClose	19:08:07
MLMStatusUp	19:08:21
:	:
MiddleLayerManagerUp	19:08:37
CiscoDCDLinkUp	19:08:39

그림 1. 네트웍 상에서의 이벤트 시퀀스

위와 같은 이벤트 시퀀스 상에서 우리가 생각하는 전형적인 질의 형태는 다음과 같다: " CiscoDCDLinkUp이 발생한 후 20초 이내에 MLMStatusUP이 발생하며 그 후 40초 이내에 CiscoDCDLinkUP 이 발생하는 모든 경우를 검색하라."

위의 질의는 Wang et al. [6] 이 고려한 질의를 일반화 한 것이라고 볼 수 있다. Wang et al. 은 이벤트간의 간격이 주어진 상수 값과 같은 경우를 대상으로 했으며 이를 위해 ISO-Depth 인덱스를 제안하였다. 하지만 위의 질의에 ISO-Depth 인덱스를 적용하면 인덱스의 검색 공간이 급격히 늘어나는 단점이 있다. 이 논문은 이와 같은 단점을 극복할 수 있는 방안으로 빈도수 기반, 조인 기반, 트리 순회 기반의 검색 기법들을 제시한다.

2. 타임스탬프된 이벤트 시퀀스

정의 1. 타임스탬프된 이벤트 시퀀스
타임스탬프된 이벤트 시퀀스란 (이벤트, 발생시각)의 순차적인 서열이다: $T = \langle(e_1,t_1), (e_2,t_2), \cdots, (e_n, t_n)\rangle$
e_i는 이벤트, t_i는 이벤트가 발생한 시각을 나타낸다. ■

T	타임스탬프된 이벤트 시퀀스
T_i	시퀀스 T에서의 i번째 아이템
$e(T_i)$	i번째 아이템에서의 이벤트
$t(T_i)$	i번째 아이템에서의 이벤트 발생 시각
A	이벤트들의 집합, $A = \cup_i \{e(T_i)\}$
$\|T\|$	시퀀스 T에서의 아이템 수
{T}	시퀀스 T에서의 시간 범위 $\{T\} = t(T_{\|T\|}) - t(T_1)$
$T' \subset T$	T' 은 T의 서브 시퀀스 (연속되지 않아도 가능)
v_i	T_{i+1}과 T_i의 시간 간격 $v_i = t(T_{i+1}) - t(T_i)$
M	윈도우 크기

그림 2. 용어 정의

타임스탬프된 이벤트 시퀀스를 간단히 이벤트 시퀀스라고 표현한다. 이 논문에서는 발생 시각이 오름차순으로 정렬되어 있는 이벤트 시퀀스에 초점을 맞추고자 한다. 즉, 시간이 흐르면서 이벤트가 발생될 때마다 (이벤트, 발생시각) 아이템이 데이터베이스에 저장되는 시퀀스를 대상으로 한다. 따라서 $i \le j$ 이면 $t_i \le t_j$ 이 성립한다. 그림 2는 논문에서 사용하는 용어를 정의한다.

{T}는 이벤트 시퀀스 T에서의 처음과 끝 이벤트 발생 시각의 차이, |T|는 T 안에 들어 있는 아이템의 수를 나타낸다. T 안에서 몇몇의 아이템을 제외하면 서브 시퀀스 T'을 얻는다.
우리가 찾으려고 하는 패턴 P는 {P}가 M을 넘지 않는다는 가정하에서 이벤트와 이벤트 사이의 시간 간격이 m_i를 넘지 않는 특별한 이벤트 시퀀스이다.

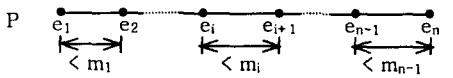

그림 3. 패턴 P

정의 2. 타임스탬프된 이벤트 시퀀스 검색
주어진 패턴 P와 서브 시퀀스 T'가 1부터 |P|-1 까지의 모든 i에 대하여 다음 식을 모두 만족하면 T'와 P는 매치된다고 말한다:

$$|P| = |T'|,\ e(P_i) = e(T_i),\ t(T'_{i+1}) - t(T'_i) < m_i$$

이벤트 시퀀스 검색이란 데이터베이스에 저장된 시퀀스 T 안에서 우리가 원하는 패턴 P와 매치되는 모든 서브 시퀀스 T' 을 찾아내는 연산을 의미한다. ■

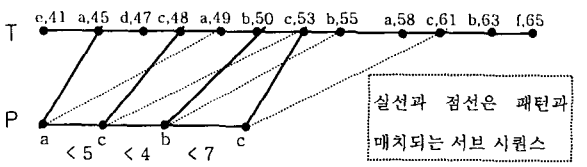

그림 4. 이벤트 시퀀스 검색

그림 4에서 패턴 P는 4개의 이벤트로 구성되어 있으며 각각의 이벤트 사이에는 m_1=5, m_2=4, m_3=7 인 최대 허용 시간을 가지고 있다. 이벤트 시퀀스 T 안에서 패턴 P와 매치되는 두 개의 서브 시퀀스 <(a,45), (c,48), (b,50), (c,53)>과 <(a,49), (c,53), (b,55), (c,61)>를 검색할 수 있다.

3. 이벤트 시퀀스 검색 방법

3.1 이벤트 빈도수를 기반으로 한 순차적인 검색 방법

데이터베이스에 저장된 이벤트 시퀀스를 차례대로 읽으면서 패턴과 매치되는 서브 시퀀스를 찾는 방법을 순차적인 검색 방법이라고 한다. 이 장에서는 패턴과 비교되는 서브 시퀀스의 수를 줄이기 위해 이벤트 빈도수를 기반으로 하는 검색 방법을 제시한다. 이를 위해 각각의 이벤트가 데이터베이스에서 몇 번 발생했는가를 계산하여 표에 저장한다. 그림 5는 그림 4의 이벤트 시퀀스에 대한 이벤트 빈도표를 보여준다. 패턴이 주어지면 패턴에 포함된 이벤트 중에서 데이터베이스 내의 빈도수가 가장 적은 것을 선택하여 이것을 중심으로 매칭을 시도한다. 그러나 이벤트가 데이터베이스 내에서 적은 빈도수를 가지고 있더라도 패턴 내에서 상대적으로 많이 발생한다면 비교 횟수가 증가하게 된다. 따라서 데이터베이스뿐만 아니라 패턴에서의 빈도수도 고려해야 한다.

	a	b	c	d	e	f
DB	3	3	3	1	1	1

그림 5. 그림 4의 이벤트 시퀀스에 대한 이벤트 빈도표

$f_e(T)$를 이벤트 시퀀스 T에서의 이벤트 e의 빈도수라고 한다면 $f_e(T)*f_e(P)$의 값은 이벤트 e를 기준으로 T에서 패턴을 비교하는 횟수이다. 따라서 $f_e(T)*f_e(P)$의 값을 최소로 갖는 이벤트 e_m를 찾아 이를 기준으로 하여 그림 6에서처럼 앞과 뒤의 M 시간 간격만큼의 이벤트들을 비교해 나가면 최적의 순차 검색 성능을 얻을 수 있다.

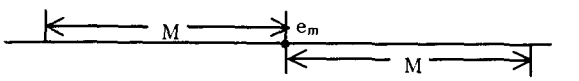

그림 6. e_m을 기준으로 2*M 시간 범위를 가진 윈도우

3.2 조인 기반의 검색 방법

Li et al.[3]은 XML 경로 질의를 효율적으로 처리하기 위해 질의를 다수의 서브 질의로 분해하여 실행한 후 각각의 결과들을 조인하여 최종 결과를 얻는 방법을 사용하였다. 본 장에서는 위와 유사한 방식을 사용하여 원하는 패턴을 찾는 방법에 대해 기술한다. n개의 이벤트로 구성된 패턴을 생각해보자. 바로 인접한 두 개의 이벤트들을 중첩하면서 묶으면 n-1개의 서브 패턴들을 구할 수 있다. 예를 들면 그림4의 패턴 P로부터 3개의 서브 패턴 (a, c), (c, b), (b, c)를 추출할 수 있다. 다음으로 각각의 서브 패턴을 만족하는 이벤트 쌍들을 데이터베이스로부터 검색한다. n-1개의 서브 패턴이 있으므로 데이터베이스를 검색하여 n-1개의 중간 결과 집합을 구한다. 마지막으로 중간 결과 집합을 이벤트의 발생 시각을 고려하여 조인하면 최종 결과 집합을 얻을 수 있다.

3.2.1 서브 패턴 검색 방안

서브 패턴을 검색하는 방법 중 가장 간단한 방법은 순차적 검색을 이용하는 것이다. 그러나 검색 속도의 향상을 위해 우리는 인덱스를 사용하는 방안을 제안한다.

먼저 데이터베이스에 저장된 이벤트 시퀀스로부터 그림 7의 왼쪽과 같은 이벤트 전이표를 작성한다. 이 표의 열과 행에는 이벤트의 집합 A를 위치시킨다. 따라서 표는 $|A|^2$ 개의 셀을 가지게 된다. 이벤트 시퀀스 상에서 이벤트 e_i 발생 후 M보다 작은 t_i 시간 후에 e_{i+1}이 발생했다면, e_i와 e_{i+1}의 교차 지점의 셀에 (시간간격:v_i, 시작시각:$t(e_i)$) 아이템이 저장된다. 여기서 하나의 셀에는 많은 수의 아이템이 저장될 수 있음에 유의해야 한다. 따라서 우리는 하나의 셀에 포함되는 아이템들을 파일에 따로 저장하고, 셀에는 파일의 위치 정보만을 기록한다. 아이템들은 시간간격을 주요키, 시작 시각을 보조키로 하여 그림 7의 오른쪽 그림과 같이 순차적으로 저장된다. 아이템의 동적인 삽입과 삭제를 지원하기 위하여 순차 파일 대신에 B^+ 트리를 이용할 수도 있다.

이벤트 전이표는 크기가 크지 않으므로 주기억 장치에 저장한다. 서브 패턴이 주어지면 이벤트 전이표로부터 해당하는 셀을 찾고, 이 셀이 가리키는 파일을 처음부터 순차적으로 읽으면서 중간 결과 집합을 작성한다. 시간 간격이 m_i에 도달하면 읽기를 멈추고 중간 결과 집합을 돌려준다.

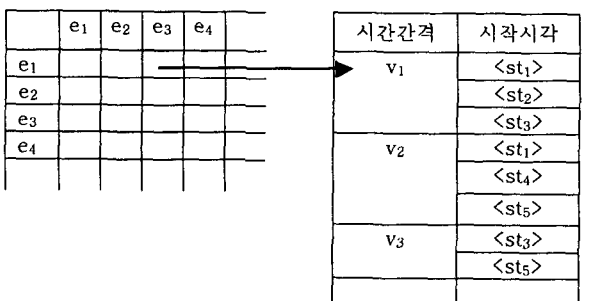

그림7. 이벤트 전이표와 저장파일

3.2.2 조인 방안

n-1개의 서브 패턴으로부터 n-1개의 중간 결과 집합을 구한 후에는 조인을 수행하여 최종 결과를 작성한다. i번째 서브 패턴의 결과를 R_i로 표현한다면, n-1개의 결과 집합을 조인하는 식은 다음과 같이 표현할 수 있다. 여기서 st는 서브 패턴의 첫번째 이벤트가 발생한 시각, et는 두번째 이벤트가 발생한 시각을 말한다. et는 st에 시간 간격을 더해서 구한다.

$$R_1 \triangleright\triangleleft R_2 \triangleright\triangleleft R_3 \cdots R_{n-2} \triangleright\triangleleft R_{n-1}$$
$$R_1.et=R_2.st \quad R_2.et=R_3.st \quad R_{n-2}.et=R_{n-1}.st$$

조인을 수행하는 순서는 성능에 중요한 영향을 미친다. 우리는 중간 결과의 크기를 줄이기 위해 빈도수를 이용해 조인 순서를 결정한다. 그림4의 예제에서 그림 8과 같은 중간 결과 집합들을 구할 수 있다. 서브 패턴 (a, c)와 (c, b)의 결과 집합의 원소의 개수는 각각 3개이고 (b, c)는 2개이다. 따라서 빈도수가 가장 적은 (b, c)를 시작점으로 조인을 하게 된다. 즉, ((a, c) ▷◁(c, b)) ▷◁(b, c)) 대신 (a, c) ▷◁((c, b) ▷◁(b, c))를 수행한다.

(a, c)	(45, 48) (49, 53) (58, 61)
(c, b)	(48, 50) (53, 55) (61, 63)
(b, c)	(50, 53) (55, 61)

그림8. 그림 4의 이벤트 시퀀스에 대한 서브 패턴의 검색 결과

3.3 트리 순회 기반의 검색 방법

조인을 이용한 검색 방법은 서브 패턴 처리 시 인덱스를 이용할 수 있다는 장점이 있으나 중간 결과를 조인하는데 많은 비용이 드는 단점이 있다. 따라서 이 장에서는 조인이 필요 없는 트리 순회 기반의 검색 방법을 제안한다. 이것은 이벤트 시퀀스에 대한 트라이(trie)[5]를 만들어 놓은 다음 패턴 P를 만족하는 경로를 트라이를 순회하면서 찾는 방법이다.

본래 영단어의 효율적인 검색을 위해 제안된 트라이는 문자열뿐만 아니라 공간 데이터의 인덱싱에도 자주 사용되는 자료 구조이다. 부모 노드와 자식 노드를 연결하는 에지에는 하나의 문자가 저장되며 루트 노드와 단말 노드를 연결하는 경로 위의 문자들을 연결하면 트라이에 저장된 영단어를 얻을 수 있다.

이벤트 시퀀스에 대한 트라이를 작성하기 위해 시간 간격이 M인 윈도우를 사용한다. 즉 가능한 모든 위치에 시간 간격이 M인 윈도우를 올려놓고 윈도우 안에 들어있는 시퀀스를 하나의 단위로 하여 트라이에 저장한다. 영단어를 위한 트라이와는 다르게 이벤트 시퀀스를 위한 트라이는 이벤트뿐만 아니라 이벤트의 발생 시각도 저장해야 한다. 따라서 윈도우 내의 시퀀스를 트라이에 저장하기 전에 각각의 이벤트에 대해서 바로 다음 이벤트와의 시간 간격을 그 이벤트 뒤에 붙이는 변환을 수행한다. 윈도우 내의 마지막 이벤트는 다음 이벤트가 없으므로 0을 뒤에 붙인다. 예를 들어 <(e, 41), (a, 45), (d, 47), (c, 48), (a, 49), (b, 50)>은 <e_4, a_2, d_1, c_1, a_1, b_0>로 변환된 후 트라이에 저장된다.

패턴이 주어지면 트라이의 루트 노드로부터 깊이 우선 탐색을 수행하여 패턴을 만족하는 경로들을 찾는다. (경로를 찾는 알고리즘에 대한 설명은 공간 부족으로 생략한다.) 루트 노드로부터 중간 노드 N을 연결하는 경로가 패턴 P를 만족한다면 N을 조상으로 가지는 모든 단말 노드를 방문하여 단말 노드에 저장된 값을 결과 집합에 포함시킨다.

4. 예비 실험

합성 데이터를 사용하여 논문에서 제안된 방법들에 대한 성능 평가 실험을 수행하였다. 네트웍 상의 이벤트 발생 환경을 모델링하기 위해 이벤트의 발생은 매개 변수가 1인 zipf 분포를 따르게 하였으며 이벤트간의 도착 간격은 평균이 10인 지수 분포를 따르도록 하였다. 이벤트 빈도수를 기반으로 한 순차 검색 방법을 fscan, 이벤트 빈도수를 고려하지 않는 순차 검색 방법을 nscan, 조인 기반의 검색 방법을 ijoin 이라고 표기하자.

먼저 fscan이 nscan에 비하여 얼마나 비교 횟수를 줄일 수 있는가를 측정하였다. 20 가지의 이벤트 유형으로부터 100만개의 이벤트가 발생한 시퀀스의 경우에 fscan은 대략 3만개, nscan은 대략 10만개 정도의 서브 시퀀스를 패턴과 비교하였다. 즉 fscan은 nscan에 비하여 비교 횟수를 약 66% 줄일 수 있었다. 다음으로 fscan과 ijoin의 질의 처리 성능을 비교하였다. 이벤트의 유형의 수, 전체 이벤트의 수, 윈도우의 크기 등을 변경하면서 실험한 결과 ijoin은 fscan에 비해 대략 2배에서 8배 정도의 성능 향상을 보였다. 그러나 찾으려고 하는 패턴이 데이터베이스에서 아주 빈번하게 발생하는 이벤트들로 구성되어 있는 경우에는 중간 결과 집합의 크기가 매우 커져 ijion의 성능이 급격히 떨어지는 것을 발견할 수 있었다. ijoin을 위한 인덱스의 크기는 데이터 시퀀스 크기의 약 40~50% 정도였다. 현재 트리 순회 기반의 검색 방법에 대한 성능을 평가하고 있는 중이다.

5. 결론

타임 시리즈나 네트웍 감시 데이터로부터 패턴과 매치되는 서브 시퀀스를 찾는 것은 이벤트 간의 연관 규칙을 발견하기 위한 필수적인 연산이다. 기존의 연구[6]에서는 패턴을 구성하는 이벤트들의 발생 간격이 주어진 상수와 '일치'하는 경우를 대상으로 하였으며 인덱싱을 위해 ISO-depth 트리를 제시하였다. 그러나 보다 일반적인 패턴 (즉, 이벤트들의 발생 간격이 주어진 상수 '이내'에 포함되는 경우)을 찾기 위해 ISO-depth 트리를 사용한다면 검색 공간이 급격이 커지는 단점이 있다. 이 논문은 이러한 단점을 극복하기 위한 방안으로 빈도수 기반, 조인 기반, 트리 순회 기반 검색 기법을 제시하였다.

이벤트 빈도수를 고려한 검색 방법은 패턴과 비교되는 서브 시퀀스의 수를 줄일 수는 있지만 여전히 순차 검색에 의존하고 있으므로 대용량의 데이터베이스에는 적합하지 않다. 조인 기반의 검색 방법은 인덱스 사용을 용이하게 하지만 조인 자체에 많은 비용이 소요되는 단점이 있다. 트리 순회를 이용한 검색 방법은 패턴을 분해없이 하나의 단위로 검색할 수 있지만 인덱스 구조를 페이지화 하기 어려운 단점이 있다. 따라서 향후에는 다차원 인덱스 구조[2]를 사용하여 패턴을 분해 없이 검색할 수 있는 방안에 대해 연구할 예정이다.

참고 문헌

[1] R. Agrawal and R. Srikant, "Mining Sequential Patterns", in ICDE, 1995.

[2] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An Efficient and Robust Access Method For Points and Rectangles", in SIGMOD, 1990.

[3] Q. Li and B. Moon, "Indexing and Querying XML Data For Regular Path Expressions", in VLDB, 2001.

[4] R. Srikant and R. Agrawal, "Mining Generalized Association Rules", in VLDB, 1995.

[5] G. A. Stephen, *String Searching Algorithms*, World Scientific Publishing, 1994.

[6] H. Wang, C. Perng, W. Fan, S. Park, and P. S. Yu, "Indexing Weighted Sequences in Large Databases" in ICDE, 2003.