타임스탬프를 갖는 이벤트 시퀀스의 인덱스 기반 검색

(Index-based Searching on Timestamped Event Sequences)

박 상 현 † 원 정 임 †† 윤 지 희 ††† 김 상 욱 ††††

(Sang-Hyun Park) (Jung-Im Won) (Jee-Hee Yoon) (Sang-Wook Kim)

요 약 시퀀스 데이타베이스로부터 원하는 질의 패턴과 일치하는 모든 서브 시퀀스를 검색하는 것은 데이타 마이닝이나 바이오 인포매틱스 등 응용 분야에서 필수적인 연산이다. 예를 들어, 특정한 이벤트가 발생할 때마다 이벤트의 유형과 발생 시각을 기록하는 네트웍 이벤트 관리 시스템에서 네트웍 이벤트들의 연관 관계를 발견하기 위한 전형적인 질의 형태는 다음과 같다. "CiscoDCDLinkUp이 발생한 후 MLMStatusUP과 TCPConnectionClose가 각각 20초 이내와 40초 이내에 순차적으로 발생하는 모든 경우를 검색하라." 본 논문에서는 대규모 이벤트 시퀀스 데이타베이스를 대상으로 하여 위와 같은 질의를 효율적으로 처리할 수 있는 인덱싱 방법을 제안한다. 기존의 방법들이 비효율적인 순차적 검색이나 페이지화하기 어려운 인덱스 구조에 의존하는데 반하여, 제안하는 방법은 저장 및 검색 효율이 입증된 다차원 공간 인덱스를 사용하여 질의를 만족하는 모든 서브 시퀀스를 착오 기각(false dismissal) 없이 신속하게 검색한다. 다차원 공간 인덱스의 입력은 이벤트 시퀀스 데이타베이스 상의 슬라이딩 윈도우 내에서 각 이벤트 유형이 최초로 발생한 시각을 기록한 n 차원 벡터가 된다. 여기서 n은 발생 가능한 이벤트 유형의 수이다. n이 큰 경우는 차원 저주(dimensionality curse) 문제가 발생할 수 있으므로 차원 선택이나 이벤트 유형 그루핑을 이용하여 차원을 축소한다. 실험 결과에 의하면 제안된 방법은 순차적 검색이나 ISO-Depth 인덱스 기법에 비하여 몇 배에서 몇 십 배의 성능 향상 효과를 갖는 것으로 나타났다.

키워드 : 시퀀스 데이타베이스, 이벤트 시퀀스, 다차원 인덱스

Abstract It is essential in various application areas of data mining and bioinformatics to effectively retrieve the occurrences of interesting patterns from sequence databases. For example, let's consider a network event management system that records the types and timestamp values of events occurred in a specific network component(ex. router). The typical query to find out the temporal casual relationships among the network events is as follows: "Find all occurrences of CiscoDCDLinkUp that are followed by MLMStatusUP that are subsequently followed by TCPConnectionClose, under the constraint that the interval between the first two events is not larger than 20 seconds, and the interval between the first and third events is not larger than 40 seconds". This paper proposes an indexing method that enables to efficiently answer such a query. Unlike the previous methods that rely on inefficient sequential scan methods or data structures not easily supported by DBMSs, the proposed method uses a multi-dimensional spatial index, which is proven to be efficient both in storage and search, to find the answers quickly without false dismissals. Given a sliding window W, the input to a multi-dimensional spatial index is a n-dimensional vector whose i-th element is the interval between the first event of W and the first occurrence of the event type Ei in W. Here, n is the number of event types that can be occurred in the system of interest. The problem of 'dimensionality curse' may happen when n is large. Therefore, we use the dimension selection or event type grouping to avoid this problem. The experimental results reveal that our proposed technique can be a few orders of magnitude faster than the sequential scan and ISO-Depth index methods.

Key words : Sequence database, Event sequence, Multi-dimensional index

· 본 연구는 2003년도 한양대학교 교내 연구비(HY-2003)와 한국과학재단 목적기초연구(R04-2003-000-10048-0)의 지원으로 수행되었습니다.

† 종신회원 : 연세대학교 컴퓨터과학과 교수
sanghyun@cs.yonsei.ac.kr
†† 정 회 원 : 연세대학교 컴퓨터과학과 교수
jiwon@cs.yonsei.ac.kr
††† 정 회 원 : 한림대학교 정보통신공학부 교수
jhyoon@hallym.ac.kr
†††† 종신회원 : 한양대학교 정보통신학부 교수
wook@hanyang.ac.kr

논문접수 : 2003년 9월 9일
심사완료 : 2004년 6월 4일

1. 서 론

시퀀스 데이타베이스로부터 원하는 패턴을 효율적으로 검색하는 것은 데이타 마이닝 분야의 데이타 클러스터링 및 연관 규칙 발견과 바이오 인포매틱스 분야의 모티프 검색 및 단백질 구조 예측 등의 다양한 응용 분야에서 필수적인 기술로서 작용한다[1-3]. 예를 들어, 네트웍 이벤트 관리 시스템을 생각해보자. 네트웍 상에서 특정한 이벤트가 발생할 때마다 그림 1과 같이 이벤트의 유형과 발생 시각이 기록된다.

Event	Timestamp
:	:
CiscoDCDLinkUp	19:08:01
MLMSocketClose	19:08:07
MLMSatusUp	19:08:20
:	:
MiddleLayerManagerUp	19:08:37
TCPConnectionClose	19:08:39
:	:

그림 1 네트웍 상에서의 이벤트 시퀀스

이와 같은 이벤트 시퀀스 상에서 이벤트들의 연관 관계를 발견하기 위한 전형적인 질의 형태는 다음과 같다. "CiscoDCDLinkUp이 발생한 후 MLMStatusUP과 TCPConnectionClose가 각각 20초 이내와 40초 이내에 순차적으로 발생하는 모든 경우를 검색하라." 이 질의는 지정된 이벤트들이 반드시 연속적으로 발생해야 하는 것을 요구하지는 않는다. 즉, 지정된 이벤트들이 정해진 시간 간격 내에서 발생하는 것만을 대상으로 할 뿐 다른 이벤트들이 지정된 이벤트들 사이에 발생하는 것에 대해서는 개의치 않는다. 이와 같이 이벤트들 간의 비연속성을 허락하는 것은 검색 알고리즘을 복잡하게 만든다는 단점이 있지만 이벤트 시퀀스 검색의 응용 범위를 넓힌다는 장점이 있다.

이와 같은 질의 유형은 Wang 등[4]이 고려한 이벤트 패턴 질의를 보다 일반화 한 것이라고 볼 수 있다. Wang 등[4]은 지정 이벤트 사이의 간격이 주어진 상수 값과 일치하는 경우를 대상으로 하였으며, 이를 위하여 트라이[5]의 변형인 ISO-Depth 인덱스를 제안하였다. 그러나 본 논문에서 대상으로 하는 일반화 된 이벤트 패턴 질의를 처리하기 위해 ISO-Depth 인덱스를 사용한다면 검색 공간이 급격히 늘어나는 단점이 있다. 이에 대한 대안으로서 트라이를 이용하는 방법과 서브 패턴으로 분해하여 처리하는 방법을 생각해 볼 수 있다. 하지만 트라이는 페이지화 하기 어려운 인덱스 구조이며, 서브 패턴으로 분해하여 처리하는 방법은 인덱스 검색을 여러 번 수행하여야 하는 단점이 있다.

본 논문에서는 이와 같은 단점을 극복하는 방안으로서 다차원 공간 인덱스를 이용한 효율적인 인덱싱 방법을 제안한다. 처리 대상의 이벤트 시퀀스 상에 일정 길이의 슬라이딩 윈도우를 연속적으로 설정하며, 모든 슬라이딩 윈도우에 대하여 각 이벤트 유형의 최초 발생 시각을 기록한 n 차원 벡터를 추출하여 이를 다차원 공간 인덱스에 저장한다. 여기에서 n은 발생 가능한 이벤트 유형의 수를 나타낸다. 그러나 n이 큰 경우는 차원 저주(dimensionality curse) 문제가 발생하여 다차원 인덱스의 효율이 저하할 수 있으므로[6], 선별력이 높은 차원만을 인덱싱에 사용하거나, 이벤트 유형 그루핑을 수행하여 차원을 축소한다. 질의 패턴은 다차원 공간 상의 질의 영역으로 표현되며, 질의 영역에 포함된 모든 후보점들을 검색한다. 마지막으로 후보점에 포함되어 있는 착오 채택(false alarm)[7,8]을 제거하여 결과 집합을 생성한다.

논문의 구성은 다음과 같다. 2장에서는 사용하는 용어와 논문에서 해결하려고 하는 문제를 정형적으로 정의한다. 3장에서는 문제 해결을 위한 기존의 연구 방법들을 소개한다. 4장에서는 본 논문에서 제안하는 인덱스 생성 방식과 질의 처리 알고리즘에 대하여 기술한다. 5장에서는 실험에 의한 성능 평가 결과에 대해서 논의한다. 6장에서는 논문을 요약하고 앞으로의 연구 방향을 제시한다.

2. 용어 및 문제 정의

발생 가능한 이벤트 유형의 개수가 n인 이벤트 유형의 집합을 $E = \{E_1, E_2, E_3, ..., E_n\}$으로 나타내면, 타임스탬프를 갖는 이벤트 시퀀스는 다음과 같이 정의된다.

정의 1. 타임스탬프를 갖는 이벤트 시퀀스

타임스탬프를 갖는 이벤트 시퀀스(time stamped event sequence) T는 (이벤트 유형, 발생 시각)의 쌍들의 리스트로서 아래와 같이 정의된다.

$$T = <(e_1, t_1), (e_2, t_2), ..., (e_m, t_m)>$$

여기에서 e_i는 발생한 이벤트의 유형을 나타내고 ($e_i \in E$), t_i는 e_i 이벤트가 발생한 시각을 나타낸다(i=1,2,..., m). 또한, (e_i, t_i)를 T의 i번째 아이템이라 부른다.

이벤트 시퀀스 T 안에 들어 있는 아이템의 수를 |T|로 표기하며, T의 처음과 마지막 이벤트 발생 시각의 차이를 {T}로 표기한다. T의 i번째 아이템을 T_i로 표기하며, T_i의 이벤트 유형과 발생 시각을 각각 $e(T_i)$와 $t(T_i)$로 표기한다. □

본 논문에서는 이후부터 타임스탬프를 갖는 이벤트 시퀀스를 간략히 이벤트 시퀀스라고 표현한다. 또한, 본 논문에서는 발생 시각이 오름차순으로 정렬되어 있는 이벤트 시퀀스에 초점을 맞추고자 한다. 즉, 시간이 흐

르면서 이벤트가 발생될 때마다(이벤트 유형, 발생 시각)의 아이템이 데이타베이스에 순차적으로 저장되는 시퀀스를 대상으로 한다. 따라서 T_i와 T_j에 대하여 $i \le j$ 이면 $t_i \le t_j$가 성립한다.

정의 2. 이벤트 시퀀스 T의 비연속 서브 시퀀스 T′

두 이벤트 시퀀스 $T(=<T_1, T_2, ..., T_n>)$와 $T'(=<T'_1, T'_2, ..., T'_k>)$가 아래의 조건을 만족하는 경우, T′은 T의 비연속 서브 시퀀스(non-contiguous subsequence)라 한다.

- 조건 1: 모든 $j(\in 1, 2, ..., k)$에 대하여 $T'_j = T_{ij}$를 만족시키는 T의 서브 스크립트들의 리스트 $<i_1, i_2, ..., i_k>$가 존재
- 조건 2: 모든 $j(\in 1, 2, ..., k-1)$에 대하여 위의 서브 스크립트들의 리스트 $<i_1, i_2, ..., i_k>$는 항상 $i_j < i_{j+1}$을 만족함.

직관적으로 설명하면, T′은 이벤트 시퀀스 T로부터 임의의 수의 아이템들을 제거함으로써 만들어진 시퀀스이다. 이후부터는 비연속 서브 시퀀스를 간략히 서브 시퀀스라 표현한다. □

정의 3. 질의 패턴 QP

질의 패턴(query pattern) QP는 아래와 같이 정의되는 특별한 형태의 이벤트 시퀀스이다.

$QP = <(e_1, c_1), (e_2, c_2), ..., (e_k, c_k)>$

단, 여기서 $c_1 = 0$이다. □

효과적인 인덱싱을 위하여 실제 응용에서 사용이 허용된 최대 {QP} $(= c_k - c_1)$ 값에 해당되는 시간 간격을 사용한다[4]. 인덱싱에 사용되는 이 시간 간격을 윈도우 크기 M으로 정의한다.

정의 4. 이벤트 시퀀스 검색

질의 패턴 QP와 서브 시퀀스 T′이 다음 조건을 모두 만족하면 T′와 QP는 매치한다고 말한다.

- 조건 1: $|QP| = |T'|$,
- 조건 2: $e(QP_i) = e(T'_i)$ $(i = 1,2, ..., |QP|)$,
- 조건 3: $t(T'_i) - t(T'_1) \le c_i$, $(i = 2,3, ..., |QP|)$.

이벤트 시퀀스 검색이란 데이타베이스에 저장된 시퀀스 T 안에서 주어진 패턴 QP와 매치되는 모든 서브 시퀀스 T′을 찾아내는 연산을 의미한다. □

정의 4는 이벤트 시퀀스 검색 문제를 데이타베이스에 저장된 하나의 이벤트 시퀀스를 대상으로 정의하였다. 그러나 실제 응용에서는 데이타베이스 내에 다수의 이벤트 시퀀스들이 존재하는 경우가 발생할 수 있다. 이 경우에는 (1) 모든 이벤트 시퀀스들을 하나의 긴 시퀀스로 연결한 후 이벤트 시퀀스 검색을 수행하거나, (2) 각 이벤트 시퀀스에 대하여 독립적으로 이벤트 시퀀스 검색을 수행한 후 전체 결과들을 병합하는 것이 가능하다. 따라서 본 논문에서는 이후부터 이러한 다수의 시퀀스들을 대상으로 하는 이벤트 시퀀스 검색에 대해서는 별도로 논의하지 않는다.

표 1은 본 논문에서 사용되는 주요 기호들의 의미를 요약 및 정리한 것이다.

표 1 논문에서 사용되는 용어들에 대한 정의

T	타임스탬프된 이벤트 시퀀스
T_i	시퀀스 T에서의 i번째 아이템
$e(T_i)$	i번째 아이템에서의 이벤트 유형
$t(T_i)$	i번째 아이템에서의 이벤트 발생 시각
$\|T\|$	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)
M	윈도우 크기

3. 기존 연구

본 장에서는 이벤트 시퀀스 검색을 위한 기존의 접근 방법들에 대하여 간략히 요약하고, 그 장단점을 지적한다.

3.1 순차적 검색 방법

저장된 이벤트 시퀀스 데이타베이스를 차례대로 읽으면서 질의 패턴과 일치하는 서브 시퀀스를 찾는 방법을 순차적 검색 방법이라고 한다. 이 방법은 구현이 간단하지만 이벤트 시퀀스 데이타베이스 전체를 액세스하여야 하며, 부분적으로 동일 구간을 반복적으로 액세스하게 되므로 대용량 데이타베이스의 검색에는 적합하지 않다.

3.2 서브 패턴을 이용한 검색 방법

주어진 질의 패턴을 서브 질의로 분해하여 각각의 서브 질의에 대하여 이벤트 시퀀스 데이타베이스를 검색한 후, 그 결과를 취합하는 방식이다. 예를 들어, 질의 패턴이 k개의 이벤트들로 구성된 경우, 질의 패턴으로부터 e_1e_i $(i=2,3,...,k)$의 형태를 갖는 서브 패턴을 k-1개 구할 수 있다. 이 들 각각의 서브 패턴을 만족하는 이벤트 쌍들을 데이타베이스로부터 검색하여 k-1개의 중간 결과 집합을 구한 후 교집합을 취하면 최종 결과 집합을 얻을 수 있다. 이와 같은 서브 패턴을 이용한 검색 방법은 인덱스 사용을 비교적 용이하게 하지만 인덱스를 여러 번 반복하여 검색해야 하는 단점이 있어 역시 대규모 데이타베이스의 경우 적합하지 않다.

3.3 트라이를 이용한 검색 방법

트라이 구조는 다수의 시퀀스들을 인덱싱하기 위하여 사용되며, 주어진 질의 시퀀스와 정확하게 일치하는 서브 시퀀스의 위치를 신속하게 찾도록 하는데 유용하다[5]. 이벤트 시퀀스의 모든 위치에 시간 간격이 M인 '윈도우'를 올려놓고 윈도우 안에 들어가는 각 시퀀스를 하나의 단위로 하여 트라이에 저장하여 트라이 인덱스를

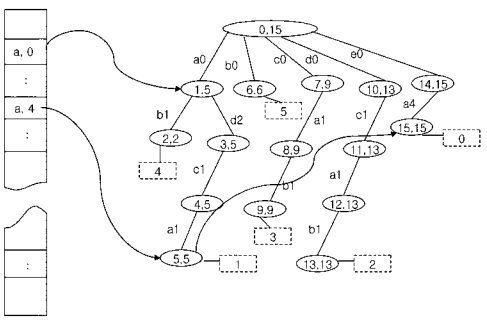
구성할 수 있다. 그러나 영어 단어 등으로 이루어진 일반 텍스트와 같은 시퀀스 데이타베이스의 경우와 달리, 이벤트 시퀀스에는 이벤트 유형과 이벤트 발생 시각이 쌍을 이루어 출현하므로, 이벤트 시퀀스를 위한 트라이에는 이벤트 유형과 함께 이벤트의 발생 시각도 저장하여야 한다. 따라서 윈도우 내의 시퀀스를 트라이에 저장하기 위하여, 각각의 이벤트 유형에 대해서 바로 이전에 발생하는 이벤트 유형과의 시간 간격을 이벤트 유형 뒤에 붙이는 변환을 수행한다. 그러나 윈도우 내에서 처음 발생한 이벤트에는 그 이전에 발생한 이벤트가 없으므로 이벤트 유형 뒤에 0을 붙여 표현한다. 예를 들어 <(e, 41), (a, 45), (d, 47), (c, 48), (a, 49), (b, 50)>와 같은 이벤트 시퀀스에 대하여 윈도우 크기 M을 4로 하여 시퀀스를 추출하여 변환을 수행하면, <e0, a4>, <a0, d2, c1, a1>, <d0, c1, a1, b1>, <c0, a1, b1>, <a0, b1>, <b0>의 6개의 시퀀스를 얻게 된다. 이 들 시퀀스를 대상으로 트라이를 구축하며, 트라이의 단말 노드에는 각 윈도우의 시작 위치를 나타내는 이벤트 시퀀스에 대한 오프셋 정보가 저장된다.

이벤트 시퀀스 검색시에는 먼저 트라이의 루트 노드로부터 깊이 우선 탐색을 수행하여 질의 패턴 QP를 만족하는 경로들을 찾게 된다. 단, QP는 비연속적인 이벤트 발생을 허용하므로 각 이벤트의 비연속적인 발생과 발생 시각 등을 고려한 경로 검색이 필요하다. 이와 같은 방식에 의하여 루트 노드로부터 중간 노드 V를 연결하는 경로가 QP와 매치된다면 노드 V 이하의 모든 단말 노드가 가지는 오프셋 정보가 검색 결과로서 리턴된다. 이러한 방법은 첫 이벤트와 이후의 이벤트의 발생 간격이 주어진 상수 값과 정확히 일치하는 질의 패턴을 찾는 경우에 비교적 좋은 효율을 기대할 수 있다. 그러나 질의 패턴의 첫 이벤트와 이후의 이벤트의 발생 간격이 일정 범위의 형태로 주어지는 경우에는 트라이 내의 동일한 경로를 여러 번 반복하여 검색해야 하므로 검색 성능이 저하된다. 또한, 인덱스 구조를 페이지화하기 어렵다는 트라이의 단점으로 인하여 DBMS와의 밀결합에 어려움이 있다[4,5].

3.4 ISO-Depth 인덱스를 이용한 검색 방법

ISO-Depth 인덱스는 3.3절에서 설명한 트라이를 기반으로 생성되며, 트라이의 단점들을 보완한 새로운 구조이다[4]. ISO-Depth 인덱스는 ISO-Depth 배열들의 집합과 오프셋 리스트들의 집합으로 이루어진다. 생성 과정을 간략히 요약하면 다음과 같다. 먼저, 트라이의 각 노드를 깊이 우선으로 탐색하면서 각 노드 V에 (V_s, V_m) 형태의 번호를 할당한다. 여기서 V_s는 노드 V의 식별자를, V_m은 노드 V의 자손 노드들이 가지는 식별자들 중 최대 값을 나타낸다. 다음으로, 각 이벤트 유형 x와 각 시간 간격 d(d=1,2,...,M)에 대하여 (x, d)의 엔트리를 갖는 ISO-Depth 배열을 작성한다. 각 (x, d)의 엔트리는 윈도우 내에서 처음 이벤트가 발생한 후 d의 시간 간격이 지난 후 이벤트 유형 x가 발생한 경우를 의미하며, 트라이 상의 해당 노드들에 대한 (V_s, V_m)을 그 값으로 갖는다. 또한, 트라이의 모든 단말 노드 정보는 오프셋 리스트 구조 내에 저장된다.

그림 2는 이벤트 시퀀스 T(=<(e, 41), (a, 45), (d, 47), (c, 48), (a, 49), (b, 50)>)에 대한 ISO-Depth 인덱스를 생성하는 과정을 예로 나타낸 것이다. 그림 2(a)는 생성된 트라이 인덱스의 각 노드에 번호 할당을 수행된 후, ISO-Depth 인덱스의 동일한 (x, d) 엔트리에 속하는 노드들을 연결하는 과정을 보이고 있다. 예를 들어 (a, 4)의 엔트리는 첫 이벤트 발생 후 4의 시간 간격이 지난 후 이벤트 유형 a가 출현한 경우를 의미하는 것으로 (5,5)와 (15,15)의 노드 정보를 값으로 갖는다. 그림 2(b)는 이와 같은 과정에 의하여 생성된 ISO-Depth 배열과 오프셋 리스트의 예를 나타낸 것이다. 이들은 순차 파일이나 B^+-트리 인덱스를 이용하여 디스크 상에 저장된다.



(a) 트라이의 노드 번호 할당과 동일한 ISO-Depth 배열 엔트리에 속하는 노드들의 연결 과정

(x, d)	(V_s, V_m)
(a, 0)	(1, 5)
(a, 1)	(8, 9)
(a, 2)	(12, 13)
(a, 4)	(5, 5), (15, 15)
(b, 0)	(6, 6)
(b, 1)	(2, 2)
⋮	⋮

V_s	Offset
2	4
5	1
6	5
9	3
13	2
15	0

(b) ISO-Depth 배열과 오프셋 리스트

그림 2 ISO-Depth 인덱스 생성 예

이벤트 시퀀스 검색 과정은 3.3절에서 논의한 트라이를 이용하는 경우와 의미적으로는 유사하다. 그러나 질의 패턴 QP에 출현하는 각 이벤트와 매치하는 관련 노드의 정보를 ISO-Depth 배열로부터 직접 얻을 수 있다는 것이 큰 차이점이다. 그러나 QP에 출현하는 각 이벤트는 QP에 표현된 순서로 발생하는 것을 의미하므로, 각 이벤트의 관련 노드는 그 전에 출현한 이벤트의 관련 노드와 트라이 상에서 자손 관계를 가져야 한다. 이와 같은 방식에 의하여 질의 QP와 매치되는 최종 노드 정보가 얻어지면 오프셋 리스트를 검색하여 단말 노드의 오프셋 정보를 얻게 된다.

ISO-Depth 인덱스는 첫 이벤트와 이후의 이벤트의 발생 간격이 주어진 상수 값과 정확히 일치하는 질의 패턴을 찾는 경우에 가장 효율적인 검색 성능을 보인다. 그러나 본 논문에서 고려하는 질의 패턴에서는 이벤트 사이의 시간 간격이 "≤ 상수값(c_i)"과 같이 주어진다. 이와 같은 형태의 질의를 처리하기 위해서는 방문해야 하는 ISO-Depth 인덱스의 엔트리 수가 증가하므로 검색 공간이 늘어나게 되며, 따라서 검색 효율이 저하하게 된다. 또한, 새로운 이벤트들이 지속적으로 추가되는 동적인 환경에 적용하기에 큰 어려움이 있다.

4. 제안하는 방법

3장에서 지적된 단점들을 극복하기 위해서는 (1) 비순차적 검색, (2) 인덱스를 한 번만 검색, (3) 페이지화하기 쉬운 인덱스 구조, (4) 작은 검색 공간 등을 지원해야 한다. 이 장에서는 이런 4가지 특징을 지원하는 인덱싱 방안에 대해 기술한다.

4.1 인덱싱 방안 스케치

먼저 이벤트 시퀀스의 가능한 모든 위치에 시간 간격의 크기가 M인 슬라이딩 윈도우를 위치시킨다. 슬라이딩 윈도우에 덮여지는 부분을 '데이타 윈도우'라고 한다. 다음으로, 모든 데이타 윈도우를 각 이벤트 유형의 최초 발생 시각을 고려하여 n 차원 상의 점으로 표현한다. n은 발생 가능한 이벤트 유형의 수이다. 데이타 윈도우의 수가 무척 많을 것이기 때문에 R^*-트리[9]와 X-트리[10] 같은 페이지 기반의 다차원 인덱스를 이용하여 이들을 저장하고 관리한다.

질의 패턴이 주어지면 이것을 n 차원 상의 공간 영역으로 표현한다. 이 공간 영역을 '질의 영역'이라고 부른다. 다음으로, 다차원 공간 인덱스로부터 질의 영역에 포함되는 점들을 검색한다. 이 점을 '후보점'이라고 부른다. 최종적으로, '착오 채택(false alarm)'[7,8]을 제거하기 위하여 각 후보점이 표현하는 데이타 윈도우를 이벤트 시퀀스로부터 읽어서 정말로 패턴과 매치되는가를 조사한다.

4.2 인덱스 작성하는 방법

먼저 데이타 윈도우 W를 다음의 서명으로 표현한다: $Sig(W) = (E_f, (w_1, w_2, ..., w_n), o)$. 여기서 E_f는 W의 첫 번째 이벤트 유형을, o는 전체 이벤트 시퀀스 상에서 W의 시작 위치를 나타낸다. w_i는 W의 첫 번째 이벤트 E_f가 발생한 후 E_i 이벤트 유형이 W 내에서 처음으로 나타날 때까지의 시간을 의미한다. 만일 E_i 이벤트 유형이 W 내에서 발생하지 않았다면 w_i 값에 M을 할당한다.

예제 1: 발생 가능한 이벤트 유형의 수가 5이며 M이 20인 이벤트 시퀀스를 생각해보자. 100번째 위치에서 시작하는 데이타 윈도우 W가 <(E_3, 12), (E_4, 15), (E_3, 21), (E_4, 22), (E_5, 30)>이면 서명은 다음과 같다: $Sig(W) = (E_3, (20, 20, 9, 3, 18), 100)$. W 내에서 E_1과 E_2가 발생하지 않았으므로 w_1과 w_2는 M의 값(=20)을 가진다. 첫 번째 이벤트가 발생한 후 9초 후에 E_3 이벤트 유형이 발생했으므로 w_3는 9가 된다. 같은 방식으로 w_5는 18이 된다. 첫 번째 이벤트가 발생한 후 3초 혹은 10초가 지나면 E_4 이벤트 유형을 볼 수 있다. 이런 경우가 발생하면 가장 작은 값을 선택한다. 즉 w_4는 3이 된다.

W를 다차원 인덱스에 표현하기 위하여, o를 식별자로 가지는 n 차원의 '데이타점' (w_1, w_2, ..., w_n)을 인덱스 I_f에 저장한다. 여기서, I_f는 첫번째 이벤트 유형이 E_f인 데이타 윈도우들을 위한 인덱스를 나타낸다. 즉, 우리는 크기가 큰 하나의 인덱스를 사용하는 대신, n개의 서로 다른 인덱스 I_1, I_2, ..., I_n를 인덱스 테이블과 함께 사용한다(그림 3 참조). 인덱스 테이블은 n개의 엔트리로 구성되며 각 인덱스의 디스크 상의 위치를 저장한다. 인덱스 테이블은 크기가 작으므로 주기억 장치에 상주시킬 수 있다.

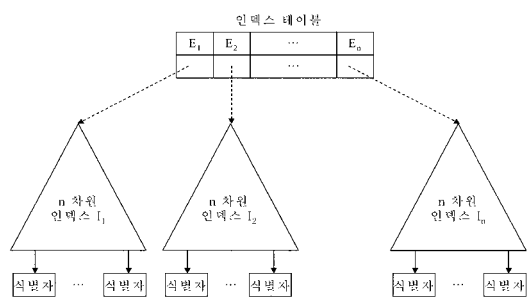


그림 3 인덱스 구조(인덱스 테이블과 n개의 n 차원 인덱스들로 구성됨)

4.3 질의를 처리하는 방법

2장에서 설명된 바와 같이, k개의 이벤트로 구성된

질의 패턴 QP는 다음과 같이 표현된다: QP = <(e_1, 0), (e_2, c_2), ..., (e_k, c_k)>. 즉, 이벤트는 e_1, e_2, ..., e_k의 순서대로 발생해야 하며, e_1이 발생한 후 c_i 이내에 e_i가 발생해야 함을 나타낸다(i=2,...,k).

질의를 처리하기 위한 첫 단계로 질의 패턴으로부터 질의 영역을 구한다. 우선, 4.2절에서 소개된 방법을 동일하게 적용하여 QP로부터 패턴의 서명 Sig(QP) = (E_f, (q_1, q_2, ..., q_n), 1)을 계산한다. 다음으로, n 차원의 '질의점' (q_1, q_2, ..., q_n)을 같은 차원의 질의 영역 ([0:q_1], [0:q_2], ..., [0:q_n])으로 변환한 후, 질의 영역에 포함되는 데이타 윈도우 점들을 다차원 인덱스 I_f으로부터 검색한다. 인덱스 I_f의 위치는 주기억 장치에 저장된 인덱스 테이블을 이용하여 쉽게 구할 수 있다. 인덱스 검색의 결과로서 질의 패턴을 만족하는 모든 데이타 윈도우들의 시작 위치 및 착오 채택(false alarm)에 해당하는 데이타 윈도우들의 시작 위치가 함께 전달된다. 질의 처리의 마지막 단계인 후처리 과정에서는 인덱스 검색 결과에 포함된 각 데이타 윈도우를 이벤트 시퀀스로부터 읽어서 착오 채택의 여부를 판별한다. 인덱스 검색 결과로부터 착오 채택을 모두 제거하고 나면 주어진 패턴을 만족하는 데이타 윈도우들만이 남게 된다. 아래의 정리 1은 데이타 윈도우 W 내에 질의 패턴 QP를 만족하는 비연속 서브 시퀀스가 하나라도 존재한다면 W를 표현하는 데이타 점이 QP를 표현하는 질의 영역에 반드시 포함된다는 것을 나타낸다. 정리 1을 통하여, 질의 패턴을 만족하는 모든 데이타 윈도우가 인덱스 검색 결과에 포함됨을 확신할 수 있다.

정리 1. 데이타 윈도우 W와 질의 패턴 QP를 생각해 보자. W의 첫 번째 아이템을 시작 아이템으로 가지는 W의 비연속 서브 시퀀스들 중에서 QP를 만족시키는 것이 하나라도 있으면, W를 표현하는 데이타 점이 QP를 표현하는 질의 영역에 반드시 포함된다. □

증명. W의 첫 번째 아이템을 시작 아이템으로 가지는 W의 비연속 서브 시퀀스들 중에서 질의 패턴 QP를 만족하는 임의의 것을 V = <(e(V_1),t(V_1)), (e(V_2),t(V_2)), ..., (e(V_k),t(V_k))>라고 하자. W를 표현하는 다차원 점을 (w_1, w_2, ..., w_n), V를 표현하는 다차원 점을 (v_1, v_2, ..., v_n), QP를 표현하는 다차원 점을 (q_1, q_2, ..., q_n)이라고 하자. 정리 1의 증명은 다음 3단계로 구성된다.

단계 1. W와 V의 첫 번째 아이템이 동일하므로 t(W_1) = t(V_1)이다. 또한, W를 구성하는 아이템의 집합이 V를 구성하는 아이템의 집합을 포함하므로, 각 이벤트 유형에 대해 그것이 W에서 최초로 발생한 시각이 그것이 V에서 최초로 발생한 시각보다 언제나 작거나 같다. 따라서, 1부터 n까지의 모든 i에 대해 $w_i \le v_i$이 성립한다.

단계 2. V와 QP를 좀 더 쉽게 비교하기 위하여 V를 구성하는 각 아이템의 발생 시각 t(V_i)를 t(V_i)-t(V_1)으로 변경하자. V가 QP를 만족하므로 1부터 n까지의 모든 i에 대해 t(V_i)-t(V_1) $\le c_i$가 성립한다. 따라서, 각 이벤트 유형에 대해 그것이 V에서 최초로 발생한 시각이 그것이 QP에서 최초로 발생한 시각보다 언제나 작거나 같다. 그러므로 1부터 n까지의 모든 i에 대해 $v_i \le q_i$가 성립한다.

단계 3. 단계 1과 단계 2를 종합하면, 1부터 n까지의 모든 i에 대해 $w_i \le v_i \le q_i$가 성립한다. 따라서 W를 표현하는 다차원 점 (w_1, w_2, ..., w_n)이 QP를 표현하는 다차원 영역 ([0,q_1], [0,q_2], ..., [0,q_n])에 반드시 포함된다. □

4.4 차원 축소 방법

발생 가능한 이벤트 유형의 수 n이 크다면 논문에서 제안하는 인덱스는 매우 높은 차원을 갖게 된다. 대부분의 다차원 인덱스는 차원이 높아짐에 따라 검색 성능이 급격히 떨어지는 차원 저주 문제를 가지고 있으므로 차원을 축소하는 방법을 적용해야 한다. 여기서는 차원 축소를 위한 방법으로 차원 선택과 이벤트 유형 그루핑을 소개한다.

4.4.1 차원 선택

n 차원 상의 데이타점 X = (x_1, x_2, ..., x_n)와 질의점 Y = (y_1, y_2, ..., y_n)를 생각해보자. 1부터 n까지의 모든 i에 대해서 $x_i \le y_i$이면 X $\le$ Y라고 표기한다. X $\le$ Y이면 질의점 Y로부터 구해지는 질의 영역에 데이타점 X가 반드시 포함된다. 착오 기각(false dismissal)이 없다는 것을 보장하기 위해서 우리는 다음의 조건을 만족하는 차원 축소 함수 F를 찾으려고 한다.

n 차원 상의 임의의 두 점 X와 Y에 대하여 X $\le$ Y이면 F(X) $\le$ F(Y)

위의 조건을 만족시키는 간단한 방법으로 차원 선택을 생각할 수 있다. 차원 선택은 n개의 차원 중에서 선별력이 가장 높은 m개의 차원을 선택하는 것이다(m$\le$ n). 각각의 인덱스 I_i는 서로 다른 데이타 분포를 가질 수 있으므로 선택되는 차원이 서로 다를 수 있다.

인덱스 I_i를 위한 차원 선택에 대해 생각해보자. 먼저, 이벤트 시퀀스로부터 추출한 데이타 윈도우들 중에서 첫 번째 이벤트 유형이 E_i인 것들의 데이타 점들을 분석하여 1부터 n까지의 각 차원별로 빈도수 히스토그램을 그린다. 빈도수 히스토그램의 x축에는 1부터 M까지의 시간 간격을, y축에는 데이타 점의 출현 빈도수를 표현한다. 세 개의 차원에 대해 임의로 그린 빈도수 히스토그램의 예를 그림 4에 보인다.

다음으로 빈도수 히스토그램을 분석하여 선별력이 가장 높은 m개의 차원을 선택한다. 엔트로피[11]가 높은

차원이 선별력이 높다고 쉽게 생각할 수도 있다. 즉, 그림 4(a)의 경우처럼, 모든 시간 간격이 동일한 빈도수를 가질 때 최대 선별력을 가진다고 생각할 수 있다. 그러나 각 차원에 대하여 질의가 "≤ c"의 형태로 주어지므로 작은 시간 간격일수록 검색 결과에 자주 포함된다. 따라서 작은 시간 간격의 빈도수가 낮을수록 결과 집합이 작아진다. 그림 4의 경우를 보면, 두 번째 차원이 최대의 선별력을 가지며 세 번째 차원이 최소의 선별력을 가짐을 알 수 있다.

각 차원의 선별력을 정량적으로 표현해보자. 질의 "≤ c"을 p_c로 나타내고, c 시간 간격의 빈도수를 f_c로 나타내자. p_1은 f_1개, p_2는 f_1+f_2개, p_c는 $f_1+f_2+...+f_c$개의 결과를 가진다. 1부터 M까지의 모든 c에 대하여 p_c가 제출될 확률이 1/M로 같다고 가정하면, 결과 집합의 예상 크기 (Estimated Result Set Size)는 다음과 같이 표현된다.

$$ERSS = (\sum_{c=1}^{M}\sum_{j=1}^{c} f_j)/M$$

ERSS가 작을수록 선별력이 높으므로, ERSS의 오름차순으로 차원들을 정렬한 후 앞에서 m개를 선택한다.

4.4.2 이벤트 유형 그루핑

차원 축소를 위한 간단하면서도 효율적인 방법으로 이벤트 유형 그루핑을 생각해 볼 수 있다. 우리는 먼저 n개의 이벤트 유형으로부터 m개의 이벤트 유형 그룹을 생성한다(m≤n). 이를 위해 이벤트 유형에 대한 거리 함수가 주어져 있어야 한다. 그러나 이러한 거리 함수가 주어지지 않는 경우가 보편적이므로 단지 각 그룹이 (거의) 동일한 수의 이벤트를 포함하도록 그루핑을 수행한다. 그루핑이 완료되면 각 그룹에 유일한 식별자를 부여한다.

우리는 n개의 서로 다른 인덱스를 사용한다는 것과 인덱스 I_i는 시작 이벤트 유형이 E_i인 데이타 윈도우들을 저장한다는 것을 기억하자. 그루핑에 의해서 생성된 m 개의 그룹을 G_1, G_2, ..., G_m으로 표현하자. 서명이 (E_f, (w_1, w_2, ..., w_n), o)인 데이타 윈도우 W를 생각해 보자. 그루핑 정보를 이용하여 n 차원의 점 (w_1, w_2, ..., w_n)을 m 차원의 점 (g_1, g_2, ..., g_m)으로 매핑한 후 인덱스 I_f에 저장한다. 여기서 g_i는 W의 첫 번째 이벤트가 발생한 후 g_i 타임 유닛이 지나면 G_i 이벤트 유형 그룹에 속하는 이벤트 유형을 최초로 W 내에서 볼 수 있음을 나타낸다.

예제 2: 4.2절에서 사용된 예를 다시 한 번 생각해보자. 발생 가능한 이벤트 유형의 수가 5이며 M이 20인 이벤트 시퀀스의 100번째 위치에서 시작하는 데이타 윈도우 <(E_3, 12), (E_4, 15), (E_3, 21), (E_4, 22), (E_5, 30)>이 (E_3, (20, 20, 9, 3, 18), 100)으로 표현되었다. E_1과 E_2가 G_1, E_3와 E_4가 G_2, E_5가 G_3로 그루핑 된다고 가정하면, 5차원의 점 (20, 20, 9, 3, 18)은 3차원의 점 (20, 3, 18)로 매핑된다.

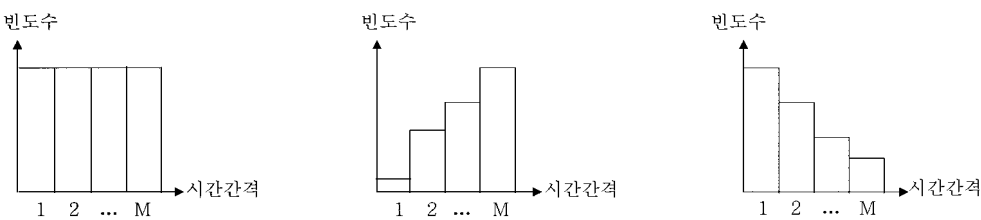
5. 성능 평가

본 장에서는 실험에 의한 성능 분석을 통하여 제안하는 기법의 우수성을 규명한다. 5.1절에서는 실험 환경을 설명하고, 5.2절에서는 실험 결과를 분석한다.

5.1 실험 환경

실험 데이타 T 내의 각 이벤트 아이템은 (이벤트 유형, 시간 간격)의 쌍으로 이루어지며, 각각 4바이트의 정수형 숫자를 이용하여 표현한다. 따라서 실험 데이타의 전체 크기는 8|T| 바이트가 된다. 실험에는 서로 다른 이벤트 유형의 개수(|E|)와 서로 다른 크기를 갖는 합성 데이타를 사용하며, 이벤트 유형의 데이타 생성에는 균등 분포와 Zipf 분포를 갖는 두 가지 유형의 데이타 분포를 사용한다. 한편 이벤트와 이벤트 사이의 시간 간격을 위한 데이타 생성에는 지수 분포의 랜덤 변수를 사용한다.

성능 평가는 다음 세 가지의 서로 다른 기법을 대상으로 한다. Ours는 본 논문에서 제안된 기법으로 다차원 인덱스를 이용하는 방식이다. 다차원 인덱스로는 현재 시퀀스 데이타베이스 분야에서 가장 널리 사용되고 있는 페이지 기반의 인덱스 구조인 R*-트리[9]를 사용



(a) 첫 번째 차원을 위한 빈도수 히스토그램 (b) 두 번째 차원을 위한 빈도수 히스토그램 (c) 세 번째 차원을 위한 빈도수 히스토그램

그림 4 세 개의 차원에 대한 빈도수 히스토그램

하여 실험한다. 사용된 R*-트리는 Maryland 대학의 Faloutsos 교수 팀에서 개발한 R*-트리 Version 2.0이며, 페이지 크기로는 1KB를 사용한다. 성능 비교를 위한 기존의 기법으로서 순차적 검색 방식(Seq-Scan)과 ISO-Depth 인덱스 검색 방식(ISO-Depth)의 두 가지를 사용한다. 단, ISO-Depth 인덱스는 트라이를 기반으로 생성되며, 트라이의 단점을 보완한 인덱스 구조이므로 트라이를 이용한 검색 방식은 비교 대상에서 제외한다.

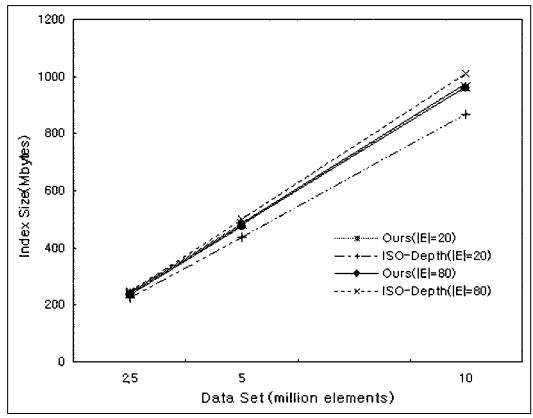
실험을 위한 하드웨어 플랫폼으로는 Windows 2000 Server를 운영체제로 사용하고, 512MB의 주기억 장치를 갖는 Pentium IV 2GHz의 PC를 사용한다.

5.2 실험 결과 및 분석

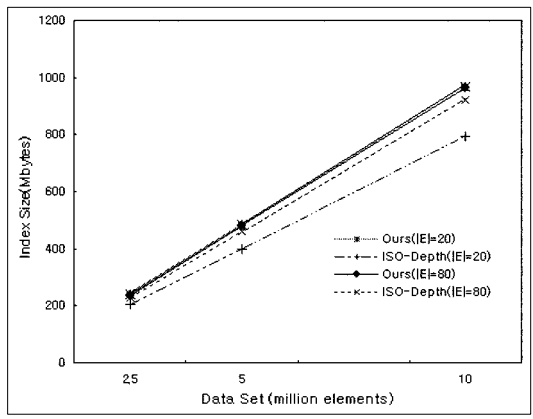
먼저, 실험 1에서는 인덱스 크기를 기준으로 하여 각 기법의 성능을 비교한다. Ours에서는 이벤트 유형의 수에 무관하게 5차원의 유형 그루핑을 수행하여, 5차원의 R*-트리 인덱스를 사용한다. ISO-Depth는 ISO-Depth 배열과 오프셋 리스트로 구성되며, B+-트리 인덱스를 갖는 순차 파일 구조를 사용한다.

그림 5와 그림 6은 데이타 크기 증가에 따르는 Ours와 ISO-Depth의 인덱스 크기 변화를 나타낸다. 가로축의 실험 데이타 크기는 이벤트 아이템의 개수를 나타내며, 이벤트 아이템의 개수가 250만개(20Mbytes), 500만개(40Mbytes), 1000만개(80Mbytes)로 증가하는 경우에 대한 인덱스의 크기 변화를 나타낸다. 그림 5(a)와 그림 5(b)는 이벤트 유형 분포로서 균등 분포와 Zipf 분포를 각각 사용한 경우를 나타내며, 윈도우의 크기를 고정시키고(M=40), 이벤트 유형의 개수(|E|)를 변화시키는 경우의 인덱스 크기 변화를 나타낸다. 실험 결과로부터, Ours와 ISO-Depth는 데이타의 크기 증가에 따라 인덱스 크기가 선형적으로 증가함을 알 수 있다. 그러나 이벤트 유형의 수가 증가함에 따라 ISO-Depth는 그 크기가 증가하지만, Ours는 크기에 거의 변화가 없음을 볼 수 있다. 그 이유는 ISO-Depth의 ISO-Depth 배열은 (이벤트 유형, 시각 간격)의 엔트리 구조를 가지므로 이벤트 유형의 개수가 증가하면 인덱스의 크기도 증가하기 때문이다. 반면에 Ours에서는 유형 그루핑 방식을 사용하므로 이벤트 유형 수의 변화는 인덱스 크기에 영향을 미치지 않고 있다.

다음의 그림 6은 이벤트 유형의 개수를 고정시키고(|E|=20), 윈도우 크기를 변화시키는 경우의 인덱스 크기 변화를 나타낸다. 여기에서는 균등 분포의 이벤트 유형 데이타를 사용하였다. 실험 결과로부터, 윈도우 크기가 증가함에 따라 ISO-Depth는 그 크기가 급격히 증가하지만, 제안된 기법은 역시 거의 변화가 없음을 볼 수 있다. 그 이유는 ISO-Depth는 윈도우 크기가 증가하면, 시간 간격 항목이 증가하므로 따라서 인덱스의 크기도 증가하게 된다. 반면에 Ours는 윈도우 크기에 거의 영향을 받지 않는다.



(a) 균등 분포의 경우



(b) Zipf 분포의 경우

그림 5 이벤트 유형 수의 증가에 따른 인덱스 크기 변화 (M=40의 경우)

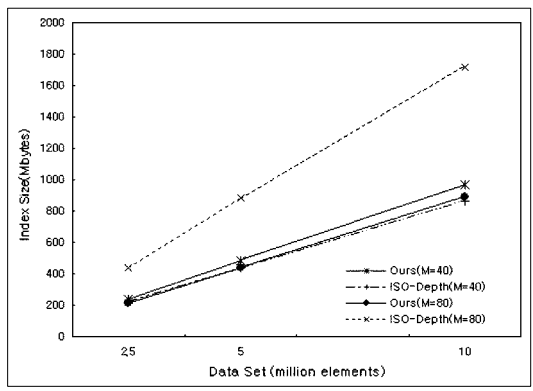
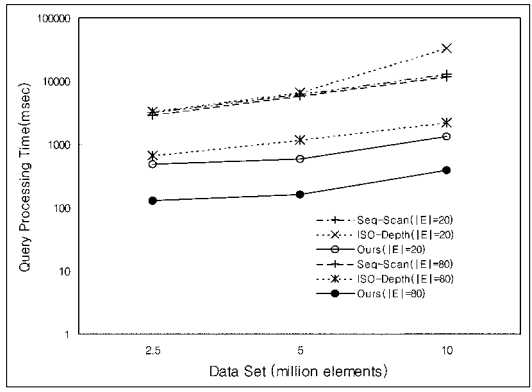


그림 6 윈도우 크기 증가에 따른 인덱스 크기 변화 (|E|=20의 경우)

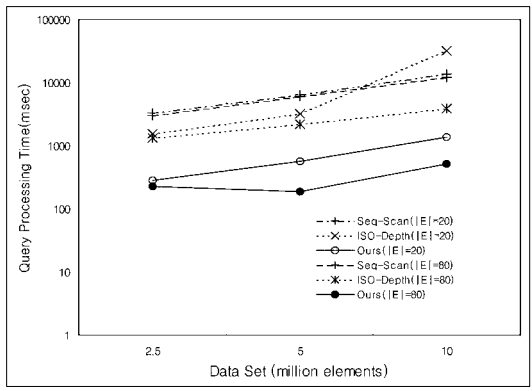
실험 2에서는 본 논문에서 제안한 인덱스 기법인 Ours의 질의 처리 시간을 Seq-Scan과 ISO-Depth 방식의 질의 처리 시간과 비교, 평가한다. 질의는 <(E_{q1}, 0>, (E_{q2}, c_2), (E_{q3}, c_3)>의 패턴을 사용한다. 여기에서 E_{q1}, E_{q2}, E_{q3}, c_2, c_3의 값은 각각 실험 데이타 생성과 같은 방식을 사용한다.

그림 7은 이벤트 시퀀스 데이타의 크기 증가에 대한 질의 평가 시간의 변화를 나타낸다. 가로축은 실험 데이타의 이벤트 아이템의 개수를 나타내고, Y축은 로그 스케일링 되어 있으며, 측정 단위는 msec이다. 그림 7(a)와 그림 7(b)는 이벤트 유형 분포로서 균등 분포와 Zipf 분포를 각각 사용한 경우를 나타내며, 윈도우의 크기를 40으로 하고, 이벤트 유형의 수를 각각 20과 80으로 하는 경우의 질의 처리 시간을 함께 보인다. 이벤트 시퀀스 데이타의 크기 증가에 따라 세 방식 모두 선형적으로 질의 처리 시간이 증가하지만, 실험 결과로부터 Ours가 Seq-Scan 혹은 ISO-Depth에 비하여 질의 처리 성능이 매우 우수함을 알 수 있다. 이 들 실험 결과로부터 Seq-Scan은 이벤트 유형의 수의 변화에 거의 무관하게 일정한 질의 처리 시간을 나타내고 있음을 알 수 있다. 반면에 ISO-Depth와 Ours는 이벤트 유형 수가 증가함에 따라 질의 처리 시간이 감소하고 있다. 그 이유는 두 경우 모두 이벤트 유형 수의 증가에 따라 검색 대상인 후보 집합이 줄어들기 때문이며, 특히 Ours는 이벤트 유형 수의 증가에 따라 현저히 질의 평가 시간이 감소하고 있음을 볼 수 있다. 그림 7(a)의 실험 결과로부터 이벤트 아이템의 개수가 1000만개인 균등 분포의 실험 데이타를 사용한 경우, Ours는 Seq-Scan에 비하여 9.6배에서 30.3배의 성능 향상 효과를 가지며, ISO-Depth에 비하여 5.7배에서 24.9배의 성능 향상 효과를 가지는 것으로 나타났다. 또한, 그림 7(b)의 실험 결과로부터 이벤트 아이템의 개수가 1000만개인 Zipf 분포의 실험 데이타를 사용한 경우, Ours는 Seq-Scan에 비하여 9.9배에서 23.3배의 성능 향상 효과를 가지며, ISO-Depth에 비하여 7.5배에서 23.2배의 성능 향상 효과를 가지는 것으로 나타났다.

다음 실험 3에서는 제안한 기법에 대하여 질의의 길이 증가에 따르는 질의 처리 시간의 변화를 평가하고, 이를 Seq-Scan과 비교한다. 그림 8에서는 이벤트 유형의 개수가 20이고 이벤트 개수가 250만개인 균등 분포의 실험 데이타를 이용하고, 윈도우 크기를 40으로 설정한 경우의 평균 질의 처리 시간을 나타낸다. 실험 결과에 의하면, Seq-Scan은 질의의 길이에 거의 영향을 받지 않는 일정한 질의 처리 시간을 보인다. 반면에 제안한 기법에서는 질의의 길이가 증가하면 오히려 질의 처리 시간이 감소함을 볼 수 있다. 그 이유는 질의의 길이가 길어지면 다차원 공간 상에서 질의 영역을 제한적으로(작게) 표현 가능하며, 따라서 후보 점의 개수가 감소하여 그 결과 실행 시간이 감소하는 효과를 얻게 된다.



(a) 균등 분포의 경우



(b) Zipf 분포의 경우

그림 7 데이타 크기 증가에 따른 평균 질의 처리 시간 비교 (M=40의 경우)

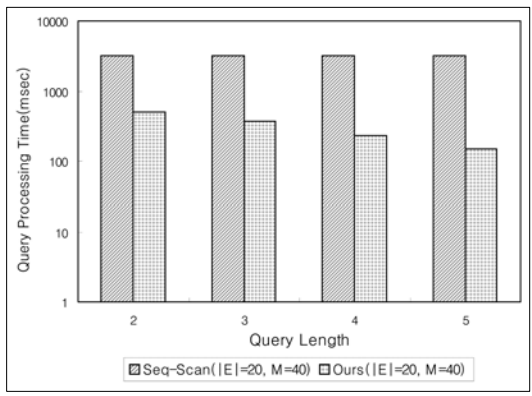


그림 8 질의 길이 변화에 따른 평균 질의 처리 시간 비교

6. 결 론

시퀀스 데이타베이스로부터 원하는 질의 패턴을 검색하는 연산은 데이타 마이닝, 데이타 웨어하우징, 바이오인포매틱스 분야에서 중요한 기본 연산자로서의 의미를 갖는다[1-3]. 본 논문에서는 대규모 이벤트 시퀀스 데이타베이스를 대상으로 하여 이벤트들의 상호 연관관계를 발견하는 효율적인 질의 처리 방식에 관하여 논의하였다.

본 연구에서 처리 대상으로 하는 질의 패턴은 Wang 등[4]이 고려한 이벤트 패턴 질의를 보다 일반화 한 것으로서, 지정 이벤트 사이의 간격이 주어진 상수 값을 초과하지 않는 모든 경우를 대상으로 하며, 이와 같은 질의 유형은 이벤트 시퀀스 검색의 응용 범위를 한층 넓히는 효과를 갖는다. 일반화된 이벤트 패턴 질의를 처리하기 위한 방안으로서, 참고문헌 [4]에서 제안하고 있는 기존의 ISO-Depth 인덱스를 사용하면 방문해야 하는 ISO-Depth 인덱스의 엔트리 수가 급격히 증가하여 검색 공간이 늘어나게 되며, 따라서 검색 효율이 저하하게 된다.

본 논문에서는 비순차적 검색 방식, 인덱스를 한 번만 검색, 페이지화 하기 쉬운 인덱스 구조, 작은 검색 공간의 4가지 특징을 동시에 지원하는 다차원 인덱싱 방안과 이를 위한 효율적인 질의 처리 기법을 제안하였다. 다차원 공간 인덱스의 입력은 이벤트 시퀀스 데이타베이스 상의 데이타 윈도우 내에서 각 이벤트 유형이 최초로 발생한 시각을 기록한 n 차원 벡터를 사용한다. 그러나 n이 큰 경우는 다차원 인덱스의 효율이 저하할 수 있으므로, 선별력이 높은 차원만을 인덱싱에 사용하거나, 이벤트 유형 그루핑을 수행하여 차원을 축소한다. 질의 패턴은 다차원 공간 상의 질의 영역으로 표현되며, 질의 영역에 포함된 모든 후보점들을 검색한 후, 후처리 과정에 의하여 착오 채택을 제거하여 결과 집합을 생성한다.

제안된 방식에 의하여 질의를 만족하는 모든 서브 시퀀스가 착오 기각 없이 검색된다는 점을 이론적 과정에 의하여 증명하였으며, 제안된 기법의 성능을 평가하기 위하여 기존의 기법들과의 다양한 실험을 통한 성능 비교를 수행하였다. 실험 결과(균등 분포의 실험 데이타의 경우)에 의하면, 제안된 기법은 순차 검색에 비하여 9.6배에서 30.3배의 성능 향상 효과를 보였으며, ISO-Depth 인덱스 방식에 비하여 5.7배에서 24.9배의 성능 향상 효과를 보였다. 특히 기존 방식과 비교하여 제안된 방식은 (1) 이벤트 유형의 수가 증가 할수록, (2) 윈도우 크기가 증가할수록 (3) 질의 시퀀스의 길이가 길어질수록 성능 개선 효과가 증가하는 것으로 나타났다.

제안된 기법의 기본 아이디어를 상대적 시간 간격 패턴(Relative Interval Pattern)의 질의 유형으로 확장, 적용할 수 있다. 상대적 시간 간격 패턴은 각 이벤트 e_i와 이전 이벤트 e_{i-1}의 시간 간격이 지정된 상수값 c_i를 초과하지 않는 것이다. 그러나 이 경우, 다차원 인덱스의 차원이 n^2이 되므로 차원 축소를 위한 적절한 방식의 선정이 인덱싱 기법의 효율에 더욱 큰 영향을 미칠 것으로 예상된다.

참 고 문 헌

[1] M. Chen, J. Han, and P. Yu, "Data Mining: An Overview from Database Perspective," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 8, No. 6, pp. 866-883, 1996.

[2] S Park, D Lee, and W. Chu, "Fast Retrieval of Similar Subsequences in Long Sequence Databases," In *Proc. 3rd IEEE Knowledge and Data Engineering Exchange Workshop* (IEEE KDEX), pp. 60-67, 1999.

[3] L. Hammel and J. Patel, "Searching on the Secondary Structure of Protein Sequences," In *Proc. 28th Int'l Conf. on Very Large Data Bases*, pp. 634-645, 2002.

[4] H. Wang, C. Perng, W. Fan, S. Park, and P. Yu, "Indexing Weighted Sequences in Large Databases," In *Proc. 19th Int'l Conf. on Data Engineering*, pp. 63-74, 2003.

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

[6] K. Chakrabarti, and S. Mehrotra, "The Hybrid Tree: An Index Structure for High Dimensional Feature Spaces," In *Proc. 15th Int'l Conf. on Data Engineering*, pp. 440-447, 1999.

[7] R. Agrawal, C. Faloutsos, and A. Swami, "Efficient Similarity Search in Sequence Databases," In *Proc. Int'l. Conference on Foundations of Data Organization and Algorithms*, FODO, pp. 69-84, 1993.

[8] C. Faloutsos and K. Lin, "FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets," In *Proc. Int'l. Conf. on Management of Data*, ACM SIGMOD, pp. 163-174, 1995.

[9] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," In *Proc. Int'l conf on Management of Data*, ACM SIGMOD, pp. 322-331, 1990.

[10] S. Berchtold, D. A. Keim, and H.-P. Kriegel, "The X-tree: An Index Structure for High-Dimensional Data," In *Proc Int'l. Conf. on Very Large Data Bases*, VLDB, pp. 28-39, 1996.

[11] C. Shannon and W. Weaver, *The Mathematical*

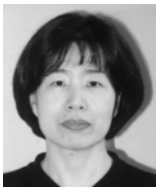
Theory of Communication, University of Illinois Press, 1964.

박 상 현

1989년 2월 서울대학교 컴퓨터공학과(학사). 1991년 2월 서울대학교 컴퓨터공학과(석사). 2001년 2월 UCLA대학교 전산학과(박사). 1991년 3월~1996년 8월 대우통신 연구원. 2001년 2월~2002년 6월: IBM T. J. Watson Research Center Post-Doctoral Fellow. 2002년 8월~2003년 8월 포항공과대학교 컴퓨터공학과 조교수. 2003년 9월~현재 연세대학교 컴퓨터과학과 조교수. 관심분야는 데이타베이스, 데이타 마이닝, 바이오인포매틱스, XML

원 정 임

1992년 2월 한림대학교 전자계산학과(학사). 1997년 8월 한림대학교 컴퓨터공학과(석사). 2003년 2월 한림대학교 컴퓨터공학과(박사). 2000년 3월~2004년 2월 한림대학교 교양교육부 강의전담. 2004년 3월~현재 연세대학교 컴퓨터과학과 연구교수. 관심분야는 데이타베이스 시스템, 데이타 마이닝, XML 응용, 바이오 정보공학, 데이타베이스 보안



윤 지 희

1982년 2월 한양대학교 전자공학과 졸업(학사). 1985년 3월 일본 구주대학교 정보공학과 졸업(석사). 1988년 3월 일본 구주대학교 정보공학과 졸업(박사). 1998년 3월~1999년 2월 미국 UCLA대학교 전산학과 방문교수. 1988년 4월~현재 한림대학교 정보통신공학부 교수. 관심분야는 시계열 데이타베이스, 데이타 마이닝, XML, 공간 데이타베이스/GIS

김 상 욱

1989년 2월 서울대학교 컴퓨터공학과 졸업(학사). 1991년 2월 한국과학기술원 전산학과 졸업(석사). 1994년 2월 한국과학기술원 전산학과 졸업(박사). 1991년 7월~8월 미국 Stanford University, Computer Science Department 방문 연구원. 1994년 2월~1995년 2월 KAIST 정보전자연구소 전문 연구원. 1999년 8월~2000년 8월 미국 IBM T.J. Watson Research Center Post-Doc. 1995년 3월~2000년 8월 강원대학교 컴퓨터정보통신공학부 부교수. 2003년 3월~현재 한양대학교 정보통신대학 정보통신학부 부교수. 관심분야는 데이타베이스 시스템, 저장 시스템, 데이타 마이닝, 바이오 정보공학, 멀티미디어 정보 검색, 공간 데이타베이스/GIS, 주기억장치 데이타베이스, 트랜잭션 관리