工學碩士學位論文

데이터 스트림 환경에서 계층형 패턴트리를 이용한 임의 시간 범위의 순차패턴 분석 방법

Sequential Patterns Analysis Method in a Range of Random Time using Hierarchical Pattern Tree in Data Stream Environment

2007년 8월

仁 荷 大 學 校 大 學 院

컴 퓨 터·情 報 工 學 科

申 載 眞

工學碩士學位論文

데이터 스트림 환경에서 계층형 패턴트리를 이용한 임의 시간 범위의 순차패턴 분석 방법

Sequential Patterns Analysis Method in a Range of Random Time using Hierarchical Pattern Tree in Data Stream Environment

2007년 8월

指導敎授　　裵　海　英

이 論文을 碩士學位請求 論文으로 提出함

仁 荷 大 學 校 大 學 院

컴 퓨 터·情 報 工 學 科

申　載　眞

요 약

지속적으로 대량의 입력이 발생하는 데이터 스트림 환경에 맞추어 다수의 순차들로부터 빈번하게 등장하는 패턴을 분석해 내는 순차패턴 분석은 새롭게 연구가 되고 있다. 그러한 연구들은 현재 입력되고 있는 데이터 스트림에 대한 실시간 패턴 분석에 초점을 맞추고 있으나 패턴의 트랜드 분석을 위해선 과거 임의 시간 범위에 대한 순차패턴 분석도 이루어져야 한다.

순차패턴 분석은 데이터 마이닝에 속하는 연구로서, 데이터 스트림에서의 데이터 마이닝은 실시간 연산, 적은 메모리 소비, 과거 데이터 스트림에 대한 데이터 분석이 가능해야 한다. 이러한 데이터 스트림 환경에서 후보 패턴을 생성하지 않는 PrefixSpan, 중복된 패턴 탐색이 없는 MILE, 실시간 순차패턴 분석을 위한 SMDS, 최근부터 과거 어느 시점까지의 패턴 분석을 지원하는 FP-stream 등의 순차패턴 분석이 제안되었다. 그러나 이러한 방법들은 실시간으로 순차패턴 분석을 할 수는 있으나 그 과정에서 발견된 문제점 분석을 위한 과거의 임의 시간 범위에 대한 패턴은 분석하지 못한다.

본 논문은 데이터 스트림 환경에서 계층형 패턴트리를 이용한 임의 시간 범위의 순차패턴 분석 방법을 제안한다. 제안방법은 기존에 분석되었던 순차가 분석하고 있는 패턴의 부분집합으로 등장하면 기존에 분석된 순차와 현재 분석된 순차를 이어주는 방법으로 순차패턴을 분석한다. 이러한 순차패턴 분석의 결과로 생성된 패턴트리들은 1 초 동안 모아지며, 모아진 패턴트리들은 합병되어 초 범위 패턴트리를 이루며 초 범위 패턴트리들은 분 범위 패턴트리를, 또한 분 범위 패턴트리들이 모여서 시 범위 패턴트리를 생성하여 계층형 구조를 형성한다. 더 넓은 시간 범위의 패턴트리를 생성 할 때는 패턴트리들의 노드의 빈발횟수를 합해주는 트리 합병 과정을 거친다. 이렇게 시간의 계층적으로 생성된 패턴트리들은 디스크에 저장이 되고, 임의 범위 시간에 대한 패턴 분석 요청이 들어오면 입력된 시간 범위를 시, 분, 초로 나누어 각각의 시간에 해당하는 패턴트리를 디스크에서 읽고 합병하고 빈번하지 않은 이벤트를 삭제하여 결과 패턴트리를 생성한다.

제안방법과 기존 패턴 분석 방법으로 구현된 과거 패턴 분석 방법의 패턴 분석 시간, 패턴 검색시간, 메모리 사용률의 비교 평가를 통하여 제안방법의 우수성을 보인다.

Abstract

A data streams environment which generates enormous data endlessly has been appeared, so a sequential patterns analysis that obtains frequent patterns from many sequences has been researched again according to the data streams environment. Such researches are focused on real-time patterns analysis of current incoming data streams, but patterns analysis of historical data streams have to be processed to analysis trends of the patterns.

A data mining which sequential pattern analysis belongs to in data streams environment have to be real-time operation, low memory consumption and mining historical data streams. In such data streams environment, PrefixSpan that do not produces candidate patterns, MILE that have no repeated pattern analysis, SMDS for real-time sequential patterns analysis and FP-stream that support historical patterns analysis from desired point to present time have been proposed. But these algorithms are hard to analysis historical patterns.

In this thesis, sequential patterns analysis methods in a range of random time using hierarchical patterns tree are proposed. The proposed method analyzes sequential patterns in a way that connects current constructing patterns to previous analyzed patterns when previous analyzed patterns are appeared during patterns analysis. Constructed pattern trees from the patterns analysis are collected and make seconds-unit pattern trees. The seconds-unit pattern trees make minutes-unit pattern trees, and minutes-unit pattern trees make hour-unit pattern trees. Like this, pattern trees are constructed hierarchically, and stored in a disk. If pattern analysis request in a random time range is entered, the time range is divided into each time unit, and pattern trees of the time unit are read from disk. A result pattern tree is constructed after merging the obtained pattern trees and refining the merged the pattern tree.

Advantages such as pattern analysis time, pattern searching time and low memory usage of proposed method are proved in a comparison with pattern analysis methods implemented from previous analysis methods.

목차

1. 서론

최근에 주목을 받고 있는 증권 정보, 네트워크 패킷, 센서 네트워킹, 물류 감시 시스템, 웹 클릭 스트림등의 응용들은 데이터 스트림이라고 하는 새로운 형태의 데이터를 다루고 있다[1-5,20]. 기존의 데이터베이스관리시스템에서는 데이터 입력 연산에 비하여 검색 연산의 비율이 높기 때문에 디스크에 저장된 데이터에 대한 빠른 검색의 연구가 주를 이루었다[10]. 그러나 기존의 데이터베이스관리시스템에서와는 다르게 데이터 스트림 환경에서는 지속적으로 대량의 데이터가 입력되는 특징을 가지므로 메모리에 저장된 현재 입력되고 있는 데이터 스트림에 대한 실시간 연산에 연구의 초점을 맞추고 있다.

데이터 스트림 환경에서의 실시간으로 진행되는 연산 중에 순차패턴 분석이란 한 트랜잭션에서 실행된 순서있는 이벤트의 집합을 의미하는 순차들 중에 빈번하게 등장하는 패턴을 찾아내는 방법이다[19]. 기존의 데이터베이스관리시스템에서 순차패턴 분석은 디스크에서 원하는 순차들을 읽어서 패턴을 분석하였다. 그러나 순차 데이터 스트림의 빠르고 많은 입력으로 인하여 데이터 스트림 환경에서의 순차패턴 분석 연구들은 현재 입력되고 있는 일정한 개수의 순차 데이터 스트림을 메모리에 저장하여 패턴을 분석하는 연구가 활발히 이루어졌다[6-8,11,13-16,21]. 이렇게 현재 입력되고 있는 순차 데이터 스트림에 대한 패턴 분석을 실시간 순차패턴 분석이라고 한다.

데이터 스트림에서의 실시간 순차패턴 분석 방법들은 GSP(Generalized Sequential Patterns)와 PrefixSpan 방법을 기본으로 제안되었다[17,18]. GSP 방법은 어떤 패턴이 발생하면 그 패턴의 일부분도 빈번하게 발생한다는 사실을 기반으로 순차패턴을 찾는다. 그러나 실제 패턴의 후보패턴이 몇 번 나오는가 계산하는 단계는 빠른 처리 시간을 요구하는 데이터 스트림에 적용하기 어렵다. PrefixSpan 방법은 후보패턴을 생성하지 않고 순차들의 패턴을 추출해 내는 방법이지만 같은 패턴을 중복해서 탐색하는 단점이 있다.

데이터 스트림 환경에서의 순차패턴 분석 연구 중에는 한 이벤트가 발생한 후 정해진 시간 안에 다른 이벤트가 발생하는 법칙을 이끌어내는 방법이 연구되었다[8]. 또한 두 이벤트 집합끼리 정해진 시간 안에 발생하는 순차패턴을 발견하는 연구도 이루어졌다[16]. 또 다른 데이터 스트림에서의 순차패턴 분석은 StreamPath 와 DSM-TKP (Data Stream Mining for Top-k Path Traversal Patterns)가 있다[13,14]. 이 두 방법 또한 데이터 스트림 환경에서 빠른 순차패턴 분석을 가능하게 하지만 PrefixSpan 과 마찬가지로 중복되는 패턴 탐색의 과정이 있다. MILE 방법은 이러한 단점을 극복하여 중복되는 패턴 탐색을 제거한 데이터 스트림의 순차패턴 탐색 방법이다[7]. MILE 은 한 개의 데이터 스트림 뿐만 아니라 다수의 데이터 스트림 스키마에서도 순차패턴 분석이 가능하다.

데이터 스트림은 입력되는 패턴이 계속해서 바뀔 수 있으므로 이러한 실시간 패턴 분석은 입력되는 데이터 스트림을 모니터링 하는데 중요하게 사용될 수 있다. 하지만 쇼핑 트랜잭션 분석 같은 계절이나 패션의 흐름과 같은 일정한 주기를 가지고 패턴이 바뀔 수 있는 응용에서는 현재의 트랜잭션 뿐만 아니라 미래 트랜잭션의 예측을 위해서는 과거 트랜잭션에 대한 패턴 분석이 필요하다. 또한 불법적인 웹 접속을 탐지하기 위해서는 실시간으로 사용자의 접속 과정을 모니터링 하는 실시간 패턴 분석도 중요하지만, 그 침입의 원인과 과정을 분석하려면 과거 몇 분 동안 침입자의 트랜잭션 패턴 또한 분석해야 할 필요성이 있다. 따라서 이러한 데이터 스트림의 트랜드와 변화 분석을 위해서는 실시간 패턴 분석 뿐만 아니라 사용자가 원하는 과거의 어떤 시간 범위에 대한 패턴 분석도 요구가 된다. 그러나 기존의 데이터 스트림 환경에서의 순차패턴 분석들은 과거에 입력되었이던 순차 데이터 스트림의 패턴 분석에 대한 연구가 부족하였다.

본 논문은 계층형 트리 구조를 이용하여 임의 시간 범위에 대한 데이터 스트림의 순차패턴 분석 방법을 제안한다. 임의 시간 범위란 현재 입력되고 있는 데이터 스트림에 대한 실시간 패턴 분석이 될 수도 있고 과거의 어떠한 시간 범위가 될 수도 있다. 따라서

제안방법은 실시간 순차패턴 분석과 과거의 순차패턴 분석을 가능하게 한다. 데이터 스트림은 빠른 속도로 입력이 되기 때문에 제안방법에서는 순차 데이터 스트림을 저장하지 않고 순차 데이터 스트림에서 생성해 낸 패턴을 디스크에 저장한다. 패턴을 생성하기 위해서는 지속적으로 입력되는 순차 데이터 스트림은 정해진 개수가 입력 될 때마다 패턴트리를 생성한다. 패턴트리 생성은 짧은 순차패턴은 긴 순차패턴의 부분집합이 될 수 있다는 사실을 기반으로 한다[20]. 따라서 패턴 분석 도중 이미 분석된 패턴이 등장하면 그 패턴을 포함한다고 표시하여 중복된 패턴분석을 방지한다. 생성된 패턴트리는 1 초 동안 모아서 초 범위 패턴트리로 합병된다. 합병 방법은 합병될 모든 패턴트리의 이벤트 노드와 이벤트의 등장 횟수를 더하는 방법으로 합병한다. 만들어진 초 범위 패턴트리는 분 범위 패턴트리를 생성하고 또 분 범위 패턴트리는 모아져서 시 범위 패턴트리를 생성한다. 이와 같이 계층적으로 생성된 패턴트리들은 빠른 접근을 위하여 인덱스와 함께 디스크에 저장된다. 임의 시간에 대한 질의가 입력되면 입력된 시간을 초, 분, 시 범위로 나누어 디스크에서 해당하는 패턴트리를 읽은 다음 읽혀진 패턴트리를 합병하고 합병된 트리에서 빈번하지 않은 이벤트를 삭제함으로써 사용자가 원하는 시간 범위의 패턴을 찾아낸다.

본 논문의 구성은 다음과 같다. 2 장에서는 데이터 스트림 환경에서의 데이터 마이닝에 관해 언급을 한 후 다른 순차패턴 분석 알고리즘들과 그에 대한 장.단점을 설명한다. 그리고 3 장에서 제안하는 순차패턴 분석 알고리즘을 설명하고, 4 장에서는 다른 순차패턴 분석 알고리즘과 제안방법의 성능 비교 분석을 통하여 제안방법의 장점을 증명을 한 후 5 장에서 결론과 향후 연구에 관해 설명한다.

2. 관련연구

순차패턴 분석 방법은 데이터 마이닝의 주요 기법 중에 하나이다. 따라서 본 장에서는 기존 환경에서 데이터 스트림 환경으로 적용이 되면서 변화되는 데이터 마이닝의 특징에 관해 설명한 후, 기존에 제안되었던 순차패턴 분석 방법과 그 방법들의 장.단점에 대하여 분석한다.

2.1 데이터 스트림 환경에서의 데이터 마이닝

순차패턴 분석은 데이터 마이닝 분야에 속하는 연구로서 입력된 데이터에 대한 안전한 저장과 빠른 검색을 할 수 있는 데이터베이스관리시스템에서 수행되었다. 이렇게 기존에 저장된 데이터에 대한 데이터 마이닝을 실행하는 것을 폐쇄 데이터 마이닝이라고 한다[21]. 그러나 이와 같은 데이터베이스관리시스템과는 다르게 데이터 스트림 환경에서는 지속적으로 대량의 입력이 발생한다. 따라서 기존의 폐쇄 데이터 마이닝에서 수행되었던 마이닝 연산을 데이터 스트림 환경에서 실행할 수는 있으나 끊임없이 입력되는 데이터 스트림으로 인하여 분석된 마이닝의 결과는 옛날 데이터에 대한 분석 결과가 되는 현상이 발생한다. 따라서 입력되는 데이터 스트림을 이용하여 기존의 마이닝의 결과를 계속해서 갱신하는 새로운 형태의 마이닝이 수행되어야 한다. 이러한 데이터 스트림 환경에서의 데이터 마이닝을 개방 데이터 마이닝이라고 한다[9].

순차패턴 분석과 같은 데이터 마이닝의 알고리즘들은 데이터 스트림 환경에서 수행되기 위하여 몇 가지 제약을 갖는다. 첫 번째는 개방 데이터 마이닝은 항상 최신의 데이터를 분석해야 한다. 앞에서 언급하였듯이 데이터 스트림은 지속적으로 입력이 되며 그 특징 또한 끊임없이 바뀔 수 있다. 따라서 계속해서 변화하는 데이터 스트림의 특징을 분석하기

위해 한 번 마이닝이 된 결과는 계속해서 입력되는 데이터 스트림 환경에 맞추어 새롭게 갱신이 되어야 한다. 두 번째 제약은 개방 데이터 마이닝의 알고리즘은 최대한 적은 양의 메모리를 사용하여야 한다. 데이터 스트림은 입력 속도가 일정하지 않기 때문에 갑자기 많은 양의 데이터 스트림의 입력이 발생 할 수도 있다. 이 때 데이터 마이닝 연산에서 너무 많은 양의 메모리를 사용한다면 입력되는 데이터 스트림을 저장할 메모리 공간이 부족하게 된다. 따라서 개방 데이터 마이닝은 최대한 적은 양의 메모리를 사용하면서 수행되어야 한다. 마지막 제약조건은 과거 데이터 스트림에 대한 데이터 마이닝이 가능해야 한다. 지속적으로 대량의 데이터 스트림이 입력이 되지만 디스크의 용량과 I/O 시간은 느리기 때문에 데이터 스트림은 디스크에 저장되기 어렵다[2]. 그러나 실시간으로 마이닝을 한 후, 그 결과에 대한 원인을 분석하기 위해서는 과거에 입력되었던 데이터 스트림에 대한 데이터 마이닝이 필요하다.

2.2 데이터 스트림의 순차패턴 분석

PrefixSpan 은 다수의 순차에서 빈발하게 등장하는 패턴을 찾아내는 알고리즘이다[16]. PreifxSpan 은 빈발 항목들을 찾아내는 Apriori 알고리즘을 기반으로 자주 등장하는 항목들을 찾아내어 이 항목들을 접두(Prefix)로 가지면서 빈발하게 등장하는 항목을 탐색한다. 그러한 항목을 찾았으면 첫 번째 발견된 접두 항목과 결합하여 하나의 패턴을 생성하고 생성된 패턴을 접두로 가지면서 빈발하는 항목을 재귀적으로 탐색해가며 접두 항목을 확장(Span)하는 방법으로 순차패턴을 탐색한다. 다른 순차패턴 분석 알고리즘은 순차패턴을 탐색하는 과정에서 패턴이 될 가능성이 있는 후보 패턴들을 생성하는 단점이 있으나 PrefixSpan 은 이러한 후보 패턴 생성 과정을 제거하여 사용되는 메모리의 양을 줄이고 패턴 분석의 속도를 높였다.

PrefixSpan 알고리즘으로 임의의 시간 범위를 가지는 데이터 스트림의 순차패턴 분석을 하려면 패턴 분석을 원하는 시간 범위에 속하는 순차 데이터 스트림들을 메모리에 가지고 있어야 한다. 그러나 순차패턴 분석을 하는 범위가 어제 1 시에서 2 시 사이나 어제 23 시부터 오늘 3 시까지와 같은 넓은 시간 범위를 가지게 되면 분석에 필요한 순차 데이터 스트림들을 모두 메모리에 유지하지 못하여 패턴 분석이 어려워진다.

MILE 은 PrefixSpan 을 기반으로 순차 데이터 스트림에 대한 패턴 분석을 하는 알고리즘이다[7]. MILE 은 최근에 입력되었던 일정한 개수의 데이터 스트림에 대한 순차패턴을 분석한다. 이 때 PrefixSpan 방법과는 다르게 중복된 패턴 탐색의 과정이 없으며 현재 탐색하고 있는 순차가 이전에 발견되었던 패턴으로 인식이 되면 현재 수행하던 패턴 분석을 마친다.

MILE 방법은 중복된 패턴 탐색이 없다는 장점 외에도 다수의 데이터 스트림에서도 순차패턴을 분석할 수 있는 장점을 가진다. 그러나 MILE 방법도 PrefixSpan 방법과 마찬가지로 과거 데이터 스트림에 대한 패턴 분석이 불가능하다. 분석된 패턴들을 디스크에 저장 할 수도 있지만 MILE 에서 순차패턴 분석의 단위는 최근 입력된 10 개, 혹은 최근 입력된 20 개와 같은 일정한 개수의 데이터 스트림이지 30 분, 1 시간과 같은 시간 범위의 순차패턴 분석 방법이 아니다. 또한 데이터 스트림은 입력 속도가 일정하지 않을 수 있다는 특징을 가지고 있기 때문에 MILE 과 같이 일정한 개수의 순차에서 패턴을 분석해 내면 생성된 패턴 결과들은 모두 다른 시간 범위를 가질 수 있다.

데이터 스트림의 패턴은 끊임없이 바뀔 수 있으므로 데이터 스트림에 대한 순차패턴 분석을 한 번의 분석 과정으로 끝나는 것이 아니라 끊임없이 새로 입력되는 데이터 스트림을 실시간으로 모니터링 해야 한다. SMDS 는 이러한 점을 고려하여 제안된 순차패턴 분석 방법이다[15]. SMDS 는 두 단계의 과정으로 순차패턴 분석을 실행한다. 첫 번째 단계에서는 입력된 순차 데이터 스트림들에서 빈발하게 등장하는 패턴을 분석한다.

분석하는 방법은 입력된 순차들에서 같은 차례로 등장하는 이벤트들을 그 등장횟수와 함께 모두 합하여 하나의 통합된 순차를 얻고, 그 통합된 순차에서 빈발하지 않는 이벤트를 지워 패턴을 생성한다. 두 번째 단계는 이러한 방법으로 얻어진 패턴은 기존에 생성된 패턴트리에 더해져서 저장이 된다. SMDS 는 이렇게 한 개의 패턴트리를 유지하고 있으며, 순차들에서 패턴이 생성 될 때마다 생성된 패턴으로 이전에 분석된 패턴트리를 갱신함으로써 현재 패턴을 유지한다.

SMDS 는 개방 데이터 마이닝의 조건 중에 실시간으로 마이닝을 해야한다는 조건을 만족시킨다. 그러나 MILE 과 마찬가지로 현재 패턴의 분석은 가능하지만 과거에 발생하였던 패턴에 대한 고려는 하지 않았다.

FP-stream 은 데이터 스트림에서 일정한 범위의 과거 순차패턴도 분석 할 수 있는 방법으로서 빈발 항목을 찾아내어 트리를 생성하는 FP-tree 를 기반으로 제안되었다 [11,12]. FP-stream 이 유지하고 있는 패턴트리의 각 노드에는 시간 별로 데이터 입력의 횟수를 표현하는 기울어진 시간 윈도우(Tilted-time window)를 가지고 있다. 기울어진 시간 윈도우는 최근 10 초, 최근 30 초, 최근 1 분, 최근 30 분과 같이 현재에서 과거의 다양한 시점까지의 해당하는 노드가 표시하는 이벤트의 입력 횟수를 표현한다. 따라서 다른 데이터 스트림에서의 순차패턴 분석 방법과는 다르게 현재에서부터 어떠한 과거 시점까지의 패턴 탐색이 가능하게 한다.

그러나 표현되는 빈발횟수는 항상 현재부터 과거의 어느 시점까지로 제한이 된다. 따라서 어제 1 시부터 어제 2 시까지와 같은 과거의 임의 시간 범위에 대한 패턴을 분석하기에는 한계가 있다.

3. 계층형 패턴트리를 이용한 임의 시간 범위의 순차패턴 분석 방법

본 장에서는 임의 시간 범위에 대한 순차패턴 분석 방법을 제안한다. 3.1 절에서는 본 논문에서 제안하는 패턴 분석 방법의 전체적인 특징을 나타내는 계층형 패턴트리에 관하여 설명한다. 그리고 3.2 절에서는 계층형 패턴트리를 구축하기 위해서 제일 작은 시간 범위를 가지는 패턴트리를 생성하는 방법을 설명하고, 3.3 절에서는 그러한 제일 작은 시간 범위의 패턴트리를 이용하여 더 넓은 시간 범위의 패턴트리를 생성하여 패턴트리를 계층적으로 구축하고 또한 디스크에 저장하는 방법을 설명한다. 그리고 3.4 절에서는 계층적으로 구축된 패턴트리들을 이용하여 임의 시간 범위에 대한 패턴 분석하는 방법을 설명한다.

3.1. 임의 시간 범위의 패턴 분석을 위한 계층형 패턴트리 구조

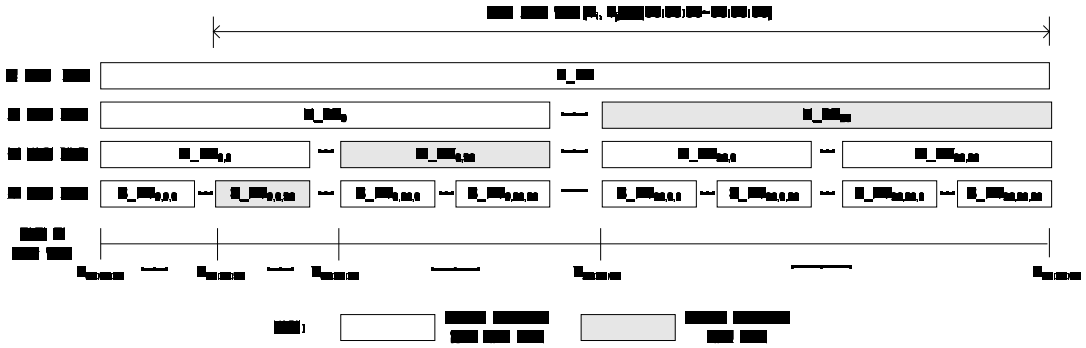
데이터 스트림의 순차들은 순차패턴 분석 알고리즘을 거쳐서 한 개의 패턴트리를 생성한다. 생성된 패턴트리는 시간 범위를 가지게 되는데, 그 시간 범위는 트리를 생성하는데 사용된 순차들의 생성 시간의 범위로 설정된다. 예를 들어 트리를 생성하는데 사용된 순차의 제일 작은 생성 시간이 1 시이고 제일 큰 생성 시간이 2 시면 그러한 순차들로 생성된 패턴트리의 시간 범위는 1 시부터 2 시가 된다.

패턴트리를 검색 할 때, 검색하려는 시간 범위가 생성된 패턴트리 한 개의 시간 범위보다 클 경우에는 짧은 시간 범위를 가지는 패턴트리 여러 개를 합쳐서 결과 패턴트리를 생성한다. 이 때 사전에 짧은 시간 범위의 패턴트리 뿐만 아니라 긴 시간 범위의 패턴트리를 생성하여 디스크에 저장하고 있으면 패턴트리 검색이 요청되었을 때 작은 시간

범위의 패턴트리 여러 개를 합치기 보다는 몇 개의 긴 시간 범위에 해당하는 패턴트리만을 결과로 넘겨주어 패턴 탐색 시간을 단축 시킬 수 있다.

패턴트리 탐색 시간의 단축을 위해 제안방법은 계층형 패턴트리 방법을 사용한다. 계층형 패턴트리 방법은 초, 분, 시, 일 범위로 다양한 패턴트리를 유지하는 방법으로서, 초 범위 패턴트리는 정해진 개수의 순차 데이터 스트림에서 얻어낸 기본 범위 패턴트리를 1 초 동안 모아 생성하고, 분 범위 패턴트리 한 개는 60 개의 초 범위 패턴트리로 만들어 진다. 이런 방법으로 시 범위, 일 범위 패턴트리도 생성한다. 만약 임의 범위를 가진 패턴트리 요청이 입력되면 생성되었던 패턴트리들을 읽어온 후, 트리들을 합병하여 결과 패턴트리를 생성한다. 예를 들어 1 시에서 2 시까지의 패턴트리를 구하는 질의가 입력되었을 경우, 시 범위 패턴트리 한 개를 검색하면 된다. 그러나 같은 1 시간의 범위지만 1 시 30 분부터 2 시 30 분까지 패턴트리를 구하는 질의가 입력이 되면, 분 범위 계층에서 1 분짜리 패턴트리 60 개를 가져온 후 합병하여 결과를 생성하고, 또한 1 시에서 1 시 40 분 20 초까지의 패턴트리를 검색하려면, 분 범위 20 개와 초 범위 20 개를 읽어들여 합치면 된다.

[그림 3-1]은 계층형 패턴트리의 모습을 나타낸다. 그림에서 검색하는 시간 범위는 00 시 00 분 59 초부터 00 시 00 분 00 초전까지이다. 한 계층적 패턴트리는 하루의 생성된 초, 분, 시, 일의 패턴트리를 가진다. 따라서 그 날의 패턴트리에 해당하는 것이 D_PT 이다. 또한 H_PT_i 는 i 시의 패턴트리를, $M\_PT_{i,j}$ 는 i 시 j 분의 패턴트리를, $S\_PT_{i,j,k}$ 는 i 시 j 분 그리고 k 초의 패턴트리를 나타낸다. 만약 임의의 시간이 입력되면 그 시간 범위에 해당하는 패턴트리들을 읽어낸다. 이 때 최대한 긴 시간 범위의 패턴트리를 가져오게 되므로 임의 시간의 순차패턴 분석의 속도를 높일 수 있다.



[그림 3-1] 임의 시간 범위에 대한 패턴트리를 읽는 계층형 패턴트리

이러한 계층형 패턴트리 방법은 과거에 임의 시간 범위에 대한 순차패턴 분석이 요청되었을 때, 과거 순차 데이터 스트림을 가지고 있지 않더라도 사전에 분석된 다양한 시간 범위의 패턴트리를 이용하여 빠르게 결과를 전해 줄 수 있다.

3.2. 기본 범위 순차패턴트리 구축 방법

기본 범위 순차패턴트리란 한 번의 순차패턴 분석의 결과로 생성된 짧은 시간 범위를 가지는 패턴트리다. 이러한 기본 범위 패턴트리가 1 초 동안 모여서 초 범위 패턴트리를 생성하고 초 범위 패턴트리는 분 범위 패턴트리를, 분 범위 패턴트리는 시 범위 패턴트리를 생성한다. 따라서 기본 범위 패턴트리는 순차패턴 분석 알고리즘으로 생성되는 유일한 패턴트리고, 계층별 패턴트리를 생성하게 되는 근원이 된다.

순차패턴 분석에서 언급되는 순차는 한 트랜잭션이 실행한 이벤트의 집합을 표현하며, 이벤트는 범주형 값으로 표현이 된다. 즉, 순차 < a, b, c >란 한 트랜잭션이 이벤트 a, b, c 가 순서대로 실행을 했다는 것을 의미한다. 또한 데이터 스트림은 끊임없이 입력이 발생되기 때문에 데이터 스트림에 질의를 하려면 반드시 질의에 사용할 데이터 스트림의 시간범위가 질의에 포함이 되어야 한다. 따라서 입력되는 순차들에도 입력된 시간 또는

생성된 시간을 의미하는 타임스탬프 값이 순차의 필드에 포함이 되어야 한다. 예를 들어 데이터 스트림에서의 한 트랜잭션이 2006 년 12 월 24 일 23 시 00 분 23 초 543 밀리초에 실행이 완료가 되었고 이벤트 a, b, c 가 순서대로 실행되었다면, 그 순차는 < 20061224230023543, a, b, c >로 표현이 가능하다. 이 시간 값은 간단히 Tn 으로 표현이 될 수 있다.

순차패턴 분석에 사용되는 아이디어는 짧은 순차패턴은 긴 순차패턴의 부분집합이 될 수 있다는 사실을 기반으로 하므로 짧은 순차패턴을 먼저 구한 후, 긴 패턴을 생성하다가 긴 패턴이 그 짧은 패턴을 가지고 있다고 판단이 되면 긴 패턴의 탐색을 멈추고 짧은 패턴을 포함한다고 표시하여 패턴 분석의 시간을 단축 할 수 있다. 이 방법을 이용하여 순차패턴 분석은 두 단계로 나뉘어 진다. 먼저 두 개의 이벤트만 가지는 짧은 순차패턴을 구한 다음, 구해진 짧은 패턴을 패턴트리에 보내준다. 패턴트리에서는 생성된 짧은 패턴이 다른 패턴의 부분 집합이 되는가 검사하게 되고, 그렇게 되면 다른 패턴에 입력된 짧은 패턴을 연결한다.

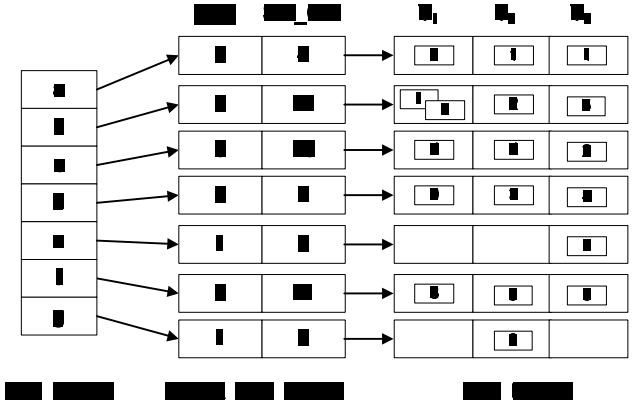
짧은 순차패턴을 찾으려면 먼저 입력된 순차들로부터 어떤 이벤트가 빈번하게 발생되었는지 알아내야 한다. 전체 순차에서 일정한 개수 이상의 순차에 등장하는 이벤트를 빈번한 이벤트라고 정의하고 빈번하다는 판단의 기준으로 Min_sup 를 정의한다. Min_sup 는 알고리즘이 사용되는 응용에 맞게 사용자가 정의한다. 만약 Min_sup 가 60%이면 전체 순차들에서 60% 이상의 순차에 등장하는 이벤트를 빈번한 이벤트라고 정의한다. 예를 들어 [표 3-1]에서 Min_sup 이 0.8 이면 순차의 개수는 3 이므로 2 개 이상의 순차에 등장하는 a, b, c, d, f 가 빈번한 데이터이다.

[표 3-1] 순차 데이터 스트림의 예

순차 이름	데이터
Seq_1	< T1, b, c, f, a, b, d >
Seq_2	< T2, f, c, d, b, a, g >
Seq_3	< T3, b, d, f, c, a, e >

데이터 스트림은 무한히 입력이 되기 때문에 데이터 스트림에 대한 질의를 하기 위해서는 연속적으로 입력되는 데이터 스트림을 일정한 개수로 나누어 질의를 해야한다. 그 질의의 종류는 일반적인 데이터베이스관리시스템에서의 selection 이나 projection 과 같은 연산이 될 수도 있고, sum 이나 average 같은 집계 연산이 될 수도 있다. 제안방법은 순차 데이터 스트림 패턴 분석을 다루고 있으므로 연속적으로 입력되는 순차 데이터 스트림들을 일정 개수로 나누기 위해 Win_size 를 정의한다. Win_size 만큼 입력된 순차 데이터 스트림은 패턴분석을 통하여 패턴 트리를 생성한다.

순차패턴 분석은 각 순차에 속한 이벤트들의 등장 횟수와 순서를 쉽게 알기 위하여 순차 정보 테이블에서 실행이 된다. [그림 3-2]은 [표 3-1]의 순차들이 순차 정보 테이블에 입력된 예이다.



[그림 3-2] 순차 정보 테이블

순차 정보 테이블은 이벤트 정보 테이블과 순서 테이블로 이루어지고 해시 인덱스로 검색된다. 이벤트 정보 테이블은 입력된 순차들 중에 해당 이벤트가 몇 번 등장하는지를 나타내는 Freq 와 각 순차에서 해당 이벤트의 Suffix 의 합을 나타내는 Suf_Cnt 로 이루어진다. Suffix 는 한 순차에서 특정 이벤트의 뒤에 등장하는 이벤트들의 개수이다. 예를 들어 순차 < Tn, a, b, c >에서 b 의 Suffix 는 c 한 개이다. 그러나 a 의 Suffix 는 b, c 두 개가 된다. 순차패턴 분석을 하려면 먼저 자주 나오는 빈발 데이터가 어떤 것인가 알아내야 한다. [그림 3-2]에서는 Win_size 가 3 이라고 가정하였고 또한 Min_sup 가 0.5 라고 가정하였다. 따라서 Freq 가 2 이상이면 빈번한 데이터라고 판단이 된다. 또한 패턴이 여러 개 있을 경우 긴 패턴은 짧은 패턴을 포함할 확률이 높기 때문에 짧은 패턴을 먼저 구하는 것이 순차패턴 분석을 빠르게 한다. 짧은 패턴 일수록 Suffix 가 적게 되므로 Suf_Cnt 가 작은 이벤트부터 순차패턴을 분석한다. 순서 테이블은 한 순차에 속하는 각 이벤트가 가지고 있는 Suffix 의 수를 나타내고 있다.

[알고리즘 3-1]은 입력된 순차 데이터 스트림을 순차 정보 테이블에 입력하고 이벤트별로 패턴을 탐색하는 알고리즘이다. 알고리즘에서 사용되는 이벤트는 이벤트의 값과 그 이벤트가 등장하는 타임스탬프를 가지는 구조로 되어있다. 예를 들어 이벤트 e 가 타임스탬프 T1, T2 를 가지는 순차에 나타났다면 그 이벤트는 $e_{T1, T2}$ 로 나타내 질 수 있다.

이러한 표기 방법은 같은 순차라도 등장하는 순차의 종류에 따라 다른 이벤트로 구별하기 위해서이다.

먼저 줄 01 에서 입력된 순차들을 순차 정보 테이블에 입력한 후 줄 02 에서 이벤트 정보 테이블에서 이벤트의 리스트를 받아온다. 그리고 이벤트의 개수만큼 줄 03 에서 줄 10 까지 반복문을 수행한다. Suf_Cnt 가 작은 이벤트가 다른 패턴의 부분집합이 될 확률이 높기 때문에, 줄 04 에서 분석의 순서는 빈번한 이벤트 중에서 Suf_Cnt 가 가장 작고 빈번하게 등장하는 이벤트부터 가져온다. 앞에서 설명한 바와 같이 빈번함의 기준은 Min_sup 이다. 만약 3 개의 순차가 입력되었고 Min_sup 가 0.5 이면 순차의 개수와 Min_sup 의 곱인 1.5 보다 많은 개수의 순차에 등장하는 이벤트가 빈번한 이벤트이다. 줄 05 에서 이벤트가 이미 트리에 있다면 다른 이벤트를 탐색한다. 그 이유는 이미 트리에 존재 한다는 것은 이미 탐색되었다는 의미이기 때문이다. 트리에 존재하는지 확인을 할 때는 이벤트의 값 뿐만 아니라 그 이벤트가 등장하는 순차의 타임스탬프의 값도 같아야 한다. 예를 들어 트리의 이벤트 $a_{T1, T2, T3}$ 와 $a_{T2, T3}$ 는 서로 다른 이벤트로 간주가 되어 트리에 동시에 존재할 수 있다. 이벤트가 트리에 없다면 줄 08 에서 트리에 추가시키고 줄 09 에서 그 이벤트로부터 시작하는 길이 2 이상의 순차패턴을 탐색한다. 선택된 이벤트로부터 시작하는 순차패턴이 있는가 결정하는 알고리즘은 [알고리즘 3-2]이다.

[알고리즘 3-1] 이벤트 별 패턴 탐색 알고리즘

Algorithm FindNewPattern (seq_list)

Input

seq_list : 입력된 순차들의 리스트

Variables

event_list : 패턴 탐색을 진행 할 이벤트 리스트

e : 이벤트

Begin

01: add_seqs(seq_list);

```
02:     event_list := get_event_list();
03:     while is_empty( event_list ) = FALSE
04:         e := get_freq_and_min_suffix_event( event_list );
05:         if exist_in_tree( e ) = TRUE then
06:             continue;
07:         end if
08:         add_pattern( e );
09:         FindPatternFrom( e );
10:     end while
End
```

[알고리즘 3-2]는 입력된 이벤트를 시작으로 하는 패턴을 찾는 알고리즘이다. 먼저 줄 01 에서 입력된 이벤트 e 를 시작으로 하는 이벤트들을 가져온다. 가져오는 이벤트는 e 가 등장하는 순차에서 e 의 Suffix 들이다. 만약 [표 3-1]의 순차들을 예로 들었을 때, e 가 $d_{T1, T2, T3}$ 이면 줄 01 에서 가져오는 Suffix 이벤트들을 살펴보면, 순차 Seq1 에서는 어떤 이벤트도 d 의 이후에 등장하지 않았고 Seq2 에서는 a, b, g, 그리고 Seq3 에서는 a, c, e, f 가 있다. 따라서 $a_{T2, T3}$, b_{T2}, c_{T3}, e_{T3}, f_{T3}, g_{T2} 가 줄 01 에서 얻어진다. 이러한 이벤트 리스트에서 이벤트를 줄 03 에서 한 개씩 가져와 줄 04 에서 빈발한지 검사한다. $d_{T1, T2, T3}$ 를 탐색하고 있는 도중, $d_{T1, T2, T3}$ 의 Suffix 이벤트 c_{T3} 는 입력된 순차 전체에서는 빈발한 이벤트이지만 $d_{T1, T2, T3}$ 의 Suffix 로는 T3 에만 등장하므로 $d_{T1, T2, T3}$ 의 Suffix 로는 빈발하지는 않는다. 따라서 $d_{T1, T2, T3}$ 의 Suffix 중에 빈발한 이벤트는 $a_{T2, T3}$ 만 선택이 되어 줄 07 으로 진행이 된다.

줄 07 에서는 선택된 suf_e 가 이미 트리에 존재하는지 검사한다. 이미 존재하면 줄 08 에서 단순히 트리에서 e 에서 suf_e 까지 포인터를 생성하고 suf_e 에 대한 탐색을 중단한다. suf_e 가 트리에 존재하지 않으면 줄 11 에서 e 와 suf_e 를 연결한 다음 줄 12 에서 패턴을 트리에 추가시켜 준다. 그리고 줄 13 에서 재귀적으로 suf_e 를 시작으로 새로운 패턴을 탐색한다. 예를 들어 $d_{T1, T2, T3}$,가 e 이고 $d_{T1, T2, T3}$,의 Suffix 로 가져온

$a_{T2,T3}$ 가 빈발하므로 둘을 연결한 < $d_{T1,\ T2,\ T3}$, $a_{T2,T3}$ >을 트리에 추가하고 $a_{T2,T3}$ 를 e 로 지정하여 재귀적으로 [알고리즘 3-2]를 호출하게 된다.

[알고리즘 3-2] 순차패턴 탐색 알고리즘

```
Algorithm FindPatternFrom ( e )
Input
    e           :       패턴 분석을 실행 할 이벤트
Variables
    event_list  :       입력된 이벤트의 Suffix들의 리스트
    suf_e       :       입력된 이벤트의 Suffix 이벤트
    pattern     :       찾아진 패턴

Begin
01:     event_list := get_suf_event_list ( e );
02:     while is_empty( event_list ) = FALSE
03:         suf_e := get_event( event_list );
04:         if is_frequent( suf_e ) = FALSE then
05:             continue;
06:         end if
07:         if already_exist_pattern( suf_e ) = TRUE then
08:             make_index( e, suf_e );
09:             continue;
10:         end if
11:         pattern := concate_event( e, suf_e );
12:         add_pattern_list( pattern );
13:         FindPatternFrom( suf_e );
14:     end while
End
```

[알고리즘 3-1]와 [알고리즘 3-2]를 [표 3-1]의 순차들로 예를 들어 설명해 보면 [알고리즘 3-1]에서 입력된 순차들의 빈번한 이벤트 중에 Suf_Cnt 가 작은 이벤트의 순서대로 선택한다. 따라서 선택된 이벤트는 a, d, c, f, b 의 순서대로 [알고리즘 3-2]의

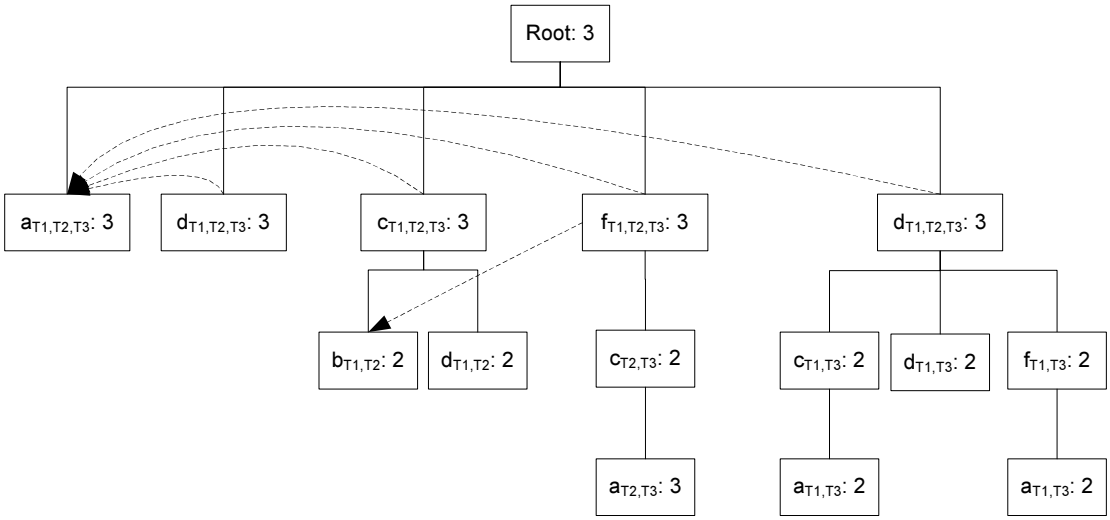
입력으로 들어가 패턴을 탐색한다. 먼저 $a_{T1, T2, T3}$가 트리에 없으므로 그 Suffix를 탐색하여 b_{T1}, d_{T1}, g_{T2} 를 얻어낸다. 그러나 선택된 Suffix 중에서는 빈번한 이벤트가 없으므로 다음 이벤트인 $d_{T1, T2, T3}$를 탐색한다.

이벤트 $d_{T1, T2, T3}$의 Suffix는 앞에서 살펴본 대로 $a_{T2, T3}$, b_{T1}, c_{T3}, e_{T3}, f_{T3}, g_{T2}가 된다. 이 중 빈번한 Suffix 는 $a_{T2, T3}$ 밖에 없다. 따라서 $d_{T1, T2, T3}$ 와 $a_{T2, T3}$ 를 연결한 후 트리에 추가한 후, 재귀적으로 $a_{T2, T3}$에 대한 패턴 탐색을 시작한다. 그러나 $a_{T2, T3}$의 Suffix는 g_{T2}, e_{T3} 밖에 없다. 따라서 $a_{T2, T3}$ 에 대한 탐색을 마치고 $d_{T1, T2, T3}$ 다음에 탐색될 이벤트인 $c_{T1, T2, T3}$의 탐색을 시작한다.

$c_{T1, T2, T3}$ 의 Suffix 는 $a_{T1, T2, T3}$, $b_{T1, T2}$, $d_{T1, T2}$, e_{T3}, f_{T1}, g_{T2} 가 있다. 먼저 $a_{T1, T2, T3}$ 를 탐색하려 한다. 그러나 이 Suffix 는 이미 트리에 존재한다. 따라서 $c_{T1, T2, T3}$에서 Suffix 인 $a_{T1, T2, T3}$까지 트리에서 포인터를 생성하고 다음 Suffix 를 탐색한다. 다음 Suffix 들 중에서 빈번한 이벤트는 $b_{T1, T2}$, $d_{T1, T2}$ 이다. 따라서 $c_{T1, T2, T3}$ 와 연결되어 트리에 입력이 되지만 재귀적으로 다시 패턴을 탐색 할 경우, 이 Suffix 로 시작하는 더 이상의 패턴은 찾지 못하고 다음 이벤트인 $f_{T1, T2, T2}$의 탐색이 시작된다.

이러한 방법으로 패턴을 탐색하게 된 결과가 [그림 3-3]에 나와 있다. 노드에 나타나는 점선 화살표는 [알고리즘 2-2]의 07 라인에서 판단하는 이미 분석된 패턴이 존재 할 경우 08 라인에서 존재하는 패턴으로 포인터를 생성해 준다. 예를 들어 $d_{T1, T2, T3}$의 경우 $a_{T1, T2, T3}$ 가 $d_{T1, T2, T3}$ 의 다음 패턴으로 나온다고 판단이 되고, $a_{T1, T2, T3}$ 가 이미 분석이 되어 트리에 존재 한다면 재귀적으로 $a_{T1, T2, T3}$에 대한 패턴 탐색하지 않고, $d_{T1, T2, T3}$에서 $a_{T1, T2, T3}$ 까지 포인터를 생성한다. 각 노드의 이벤트와 함께 있는 숫자는 이벤트가 등장하는 빈발횟수를 나타낸다. 예를 들어 “$a_{T1, T2, T3}$: 3”이라고 표기가 되어 있는 것은 $a_{T1, T2, T3}$ 이벤트가 3 개의 순차에서 등장을 하였다는 뜻이다. 또한 루트 노드의 빈발횟수는 해당

패턴트리가 몇 개의 순차로부터 생성되었는가를 나타낸다. [그림 3-3]은 [표 3-1]에 나와있는 세 개의 순차로부터 생성이 외었으므로 루트 노드의 빈발횟수는 3 이다.



[그림 3-3] 순차 데이터 스트림에서 추출된 이벤트 패턴트리

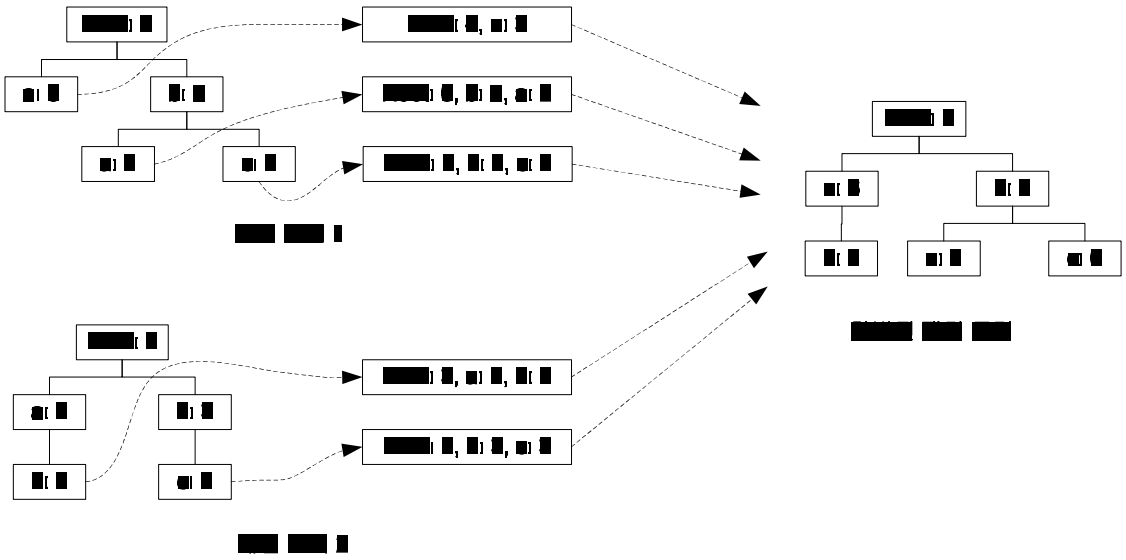
제안된 순차패턴 분석 방법은 재귀적으로 이벤트의 패턴을 탐색한다. 이 때 탐색되는 패턴에 속하는 부분 패턴이 이미 이전에 발견된 패턴이라면 재귀적 패턴 수행을 멈추고 발견이 된 부분 패턴으로 포인터를 생성한다. 이렇게 생성된 포인터의 모습이 [그림 3-3]에 나와있으며, 이러한 포인터로 인하여 패턴의 중복된 탐색이 없어지고, 패턴트리를 유지하는 메모리의 양이 절약되는 장점을 보인다.

생성된 패턴트리는 그 시간 범위와 함께 디스크에 쓰인다. 그러나 긴 시간 범위의 패턴트리를 요구하는 질의가 입력되면 짧은 패턴트리 여러 개를 합쳐야 한다. 다음 장은 작은 시간 범위를 가지는 다수의 패턴트리를 한 개의 긴 시간 범위의 패턴트리로 합치는 과정을 설명한다.

3.3 계층형 패턴트리 구축 방법

계층적으로 패턴트리를 구축하기 위해서는 먼저 1 초 동안 입력된 기본 범위 패턴트리로부터 초 범위 패턴트리를 생성하고, 60 개의 초 범위 패턴트리로부터 분 범위 패턴트리를, 그리고 분 범위 패턴트리 60 개로 시 범위 패턴트리를 생성한다.

[그림 3-4]은 짧은 시간 범위의 패턴트리들을 긴 시간 범위 패턴트리로 만드는 패턴트리 합병 과정을 나타낸다. 표현을 간단하게 하기 위하여 T1, T2, T3 와 같은 이벤트가 등장하는 순차의 타임스탬프는 생략하였다.



[그림 3-4] 이벤트 패턴트리의 합병

패턴트리의 합병 과정은 다수의 패턴트리에서 긴 패턴을 하나씩 가져와 합친다. 여기서 긴 패턴이란 루트 노드에서 리프 노드까지의 이벤트들과 그 이벤트들의 빈발횟수를 나타낸다. 예를 들어 [그림 3-4]의 패턴트리 1 에서의 긴 패턴은 < Root: 4, a: 3 >, < Root: 0, b: 4, a: 2 >, < Root: 0, b: 0, c: 3 >이 있다. 여기서 두 번째 긴 패턴에서 루트 노드의

빈발횟수가 0 인 것은 합병된 패턴트리의 루트 노드 빈발횟수를 두 번 더하지 않기 위해서다. 또한 같은 이유로 세 번째 긴 패턴의 루트 노드와 b 이벤트 또한 앞에서 이미 추가가 되었으므로 빈발 회수를 0 으로 하여 합병된 패턴트리에 더해진다. 같은 방법으로 여러 패턴트리에서 긴 패턴을 가져와서 트리에 합병 한다.

위와 같은 방법으로 같은 시간 범위 또는 다른 시간 범위의 패턴트리를 합병할 수 있다. 이 때 분 범위는 60 개의 초 범위로, 시 범위는 60 개의 분 범위로, 그리고 일 범위는 24 개의 시 범위 트리로 패턴트리를 얻을 수 있지만 초 범위는 몇 개의 기본 범위 패턴트리로 초 범위 패턴트리를 생성하는지 알 수 없다. 초 범위 패턴트리를 생성하려면 1 초 동안 생성된 기본 범위 패턴트리를 모아야 하지만 데이터 스트림의 입력 속도는 변할 수 있기 때문에 입력 속도가 빠르면 많은 기본 범위 패턴트리로부터 초 범위 패턴트리가 생성되고, 반대로 데이터 스트림의 입력속도가 느리면 적은 수의 기본 범위 패턴트리로부터 초 범위 패턴트리가 생성된다. 1 초에 수 천 이상의 기본 범위 패턴트리가 생성될 경우 기본 범위 패턴트리들을 초 범위 패턴트리로 합치는 연산은 높은 계산 비용을 필요로 하고 많은 기본 범위 패턴트리를 메모리에 유지하려면 메모리 사용량 또한 많아진다. 따라서 분 범위나 시 범위 패턴트리 합병과는 다르게 초 범위 패턴트리는 기본 범위 패턴트리가 입력되는 대로 해당하는 초 범위 패턴트리에 합쳐지고 그 초 범위 패턴트리의 생성시간이 1 초가 넘으면 초 범위 패턴트리의 합병을 완료한다.

계층적으로 구축된 패턴트리들의 합병이 완료되면 디스크에 쓰여진다. 순차에 있는 이벤트의 개수는 가변이기 때문에 생성된 패턴트리들도 가변크기를 갖는다. 그러나 하루에 생성되는 패턴트리의 개수는 일정하므로 가변크기를 가지는 패턴트리를 가리키는 고정크기의 인덱스 부분을 파일의 앞 부분에 생성한다. [그림 3-5]는 이러한 점을 고려한 파일의 구조이다. 한 파일은 하루에 생성된 모든 패턴트리의 정보를 가지고 있다. 따라서 파일의 헤더에는 파일이 생성된 날짜를 가리키는 Date 를 포함한다.

패턴트리의 인덱스 부분은 고정크기를 가진다. 인덱스는 그 날의 패턴을 나타내는 D_Idx, 시간 별 패턴트리의 인덱스를 나타내는 H_Idx_i, 분 별 패턴트리의 인덱스를 나타내는 $M\_Idx_{i,j}$, 초 별 패턴트리의 인덱스를 나타내는 $S\_Idx_{i,j,k}$ 가 있다. H_Idx_i 는 i 시의 패턴트리를 가리키고, $M\_Idx_{i,j}$ 는 i 시 j 분의 패턴트리를, $S\_Idx_{i,j,k}$ 는 i 시 j 분 k 초의 패턴트리는 가리킨다.

패턴트리의 크기는 가변적이다. 따라서 실제 패턴트리를 담고 있는 데이터 부분도 가변크기를 가진다. $S\_TP_{i,j,k}$ 는 i 시 j 분 k 초의 패턴트리를 나타내고 $M\_TP_{i,j}$ 는 i 시 j 분의 패턴트리를 H_TP_i 는 i 시의 패턴트리, D_TP 는 파일이 나타내는 날짜의 패턴트리를 나타낸다. 초 범위 패턴트리를 생성하면 데이터 부분에 초 범위 패턴트리를 입력하고 인덱스 부분에 입력된 패턴트리의 오프셋 값을 입력한다. 그리고 초 범위 패턴트리 60 개가 생성될 때마다 하나의 분 범위 패턴트리를 생성하여 60 개의 초 범위 패턴트리 뒤에 입력하고 또한 인덱스 부분의 분 범위 인덱스를 갱신한다. 비슷한 방법으로 60 개의 분 범위 패턴트리가 생성될 때마다 하나의 시 범위 패턴트리를 생성하여 파일에 입력한 후 시 범위 패턴트리에 대한 인덱스를 생성하고, 24 개의 시 범위 패턴트리가 생성될 때는 하나의 일 범위 패턴트리를 생성하여 파일에 입력한 후 일 범위 인덱스를 생성한다.

[그림 3-5] 이벤트 패턴트리의 파일 구조

[알고리즘 3-3]은 패턴트리를 합병하는 알고리즘이다. 입력되는 매개변수는 합병할 패턴트리들을 가지는 리스트이고 출력 값은 합병된 패턴트리다. 줄 01 에서 줄 08 까지 입력된 패턴트리 리스트가 빌 때까지 패턴트리를 한 개씩 가져와 수행한다. 줄 02 은

패턴트리 리스트에서 하나의 패턴트리를 가져오는 부분이다. 패턴트리를 가져왔으면 줄 03 에서 가져온 패턴트리에서 하나의 긴 패턴을 가져와서 가져온 긴 패턴을 줄 05 에서 합병될 트리에 추가를 시킨다. 그리고 줄 06 에서 다음 긴 패턴을 패턴트리에서 가져와서 다시 트리에 추가를 시켜준다. 더 이상 가져올 긴 패턴이 없으면 줄 06 에서 long_pattern 을 null 로 지정하고 줄 04 에서 long_pattern 이 없다고 판단하여 반복을 마친다.

[알고리즘 3-3] 패턴트리 합병 알고리즘

Algorithm MergePatternTree (pattern_tree_list)
Input
pattern_tree_list : 합병될 패턴트리를 가지는 리스트
Output
merged_tree : 합병된 패턴트리
Variables
pattern_tree : 합병할 패턴트리
long_pattern : 합병되는 패턴트리에 추가 할 패턴

Begin
```
01:   while is_empty( pattern_tree_list ) = FALSE
02:       pattern_tree := get_tree( pattern_tree_list );
03:       long_pattern := get_long_pat( pattern_tree );
04:       while is_empty( long_pattern ) = FALSE
05:           add_pattern( long_pattern, merged_tree );
06:           long_pattern := get_next_long_pat( merging_pattern, pattern_tree );
07:       end while
08:   end while
```
End

앞서 언급한 대로 분, 시, 일 범위 패턴트리는 일정한 개수의 패턴트리로부터 합병하지만 초 범위 패턴트리는 1 초 동안 입력된 기본 범위 패턴트리들로부터 합병한다. [알고리즘 3-4]은 초 범위 패턴트리를 생성하는 알고리즘이다. 기본 범위 패턴트리는 생성이 될 때마다

초 범위 패턴트리에 합쳐지고 갱신된 초 범위 패턴트리의 시간 범위가 1 초를 넘어가면 디스크에 입력이 된다. 입력되는 매개변수는 기본 범위 패턴트리와 기본 범위 패턴트리가 합쳐질 초 범위 패턴트리다. 줄 01 에서 입력된 두 패턴트리를 합병한다. 줄 02 에서 합병한 결과의 초 범위 패턴트리의 시간 범위가 1 초를 넘었는지 확인하게 되고, 넘었으면 줄 03 에서 디스크에 초 범위 패턴트리를 입력한다. 알고리즘의 반환 값은 초 범위 패턴트리가 디스크에 쓰여졌는지 여부를 나타낸다. 만약 알고리즘의 결과로 FALSE 가 나왔다면 이전에 합병한 초 범위 패턴트리에 다음 입력된 기본 범위 패턴트리를 합병한다.

[알고리즘 3-4] 초 범위 패턴트리 생성 알고리즘

Algorithm MakeSecondPatternTree (base_pattern_tree, sec_pattern_tree)
Input
base_pattern_tree : 초 범위에 합쳐질 기본 범위 패턴트리
sec_pattern_tree : 초 범위 패턴트리
Output
result : 디스크에 쓰여졌는지 여부

Begin
01: sec_pattern_tree := **MergePatternTree**(base_pattern_tree, sec_pattern_tree);
02: **if** over_second(sec_pattern_tree) = TRUE **then**
03: **WriteSecondTree** (sec_pattern_tree);
04: result := TRUE;
05: **else**
06: result := FALSE;
07: **end if**
08: **return** result;
End

초 범위의 패턴트리에 기본 범위 패턴를 입력한 후 1 초가 지나면 초 범위 패턴트리는 완성이 되어 디스크에 쓰여진다. [알고리즘 3-5]는 디스크에 초 범위 패턴트리를 디스크에 입력하는 알고리즘이다. 줄 01 에서 현재 패턴트리를 입력할 파일을 얻어오고 줄 02 에서 초

범위 패턴트리를 입력한다. 그리고 줄 03 에서 현재의 초 범위 패턴트리의 개수를 얻어온다. 이 때 줄 04 에서 생성된 초 범위 패턴트리의 개수가 60 의 배수라면, 줄 05 에서 최근 60 개의 초 범위 패턴트리를 가져와서 줄 06 에서 분 범위 패턴트리를 만든다. 비슷하게 줄 09 에서 줄 13 까지는 분 범위 패턴트리의 개수가 60 의 배수인지 확인하고, 60 의 배수이면 시 범위 패턴트리를 생성하고 줄 14 에서는 시 범위 패턴트리의 개수가 24 의 배수인지 확인하여, 24 의 배수이면 일 범위 패턴트리를 생성한다. 일 범위 패턴트리가 생성되면 현재 파일에 입력되는 것은 끝나게 되며 줄 18 에서 새로운 파일을 생성하여 새로운 초 범위 패턴트리의 입력을 받아들인다.

[알고리즘 3-5] 초 범위 패턴트리 입력 알고리즘

Algorithm WriteSecondTree(sec_pattern_tree)

Input

sec_pattern_tree	:	초 범위 패턴트리

Output

result	:	디스크에 쓰여졌는지 여부

Variables

cur_file	:	현재 생성되고 있는 파일
cur_sec_num	:	현재 생성되고 있는 초 범위 패턴트리의 개수
cur_sec_tree_list	:	최근 60개의 초 범위 패턴트리 리스트
new_min_tree	:	새로 생성된 분 범위 패턴트리
cur_min_num	:	현재 생성되고 있는 분 범위 패턴트리의 개수
cur_min_tree_list	:	최근 60개의 분 범위 패턴트리
new_hour_tree	:	새로 생성된 시 범위 패턴트리
cur_hour_num	:	현재 생성되고 있는 시 범위 패턴트리의 개수
cur_hour_tree_list	:	파일에 있는 24개의 시 범위 패턴트리 리스트
new_day_tree	:	새로 생성된 일 범위 패턴트리

Begin

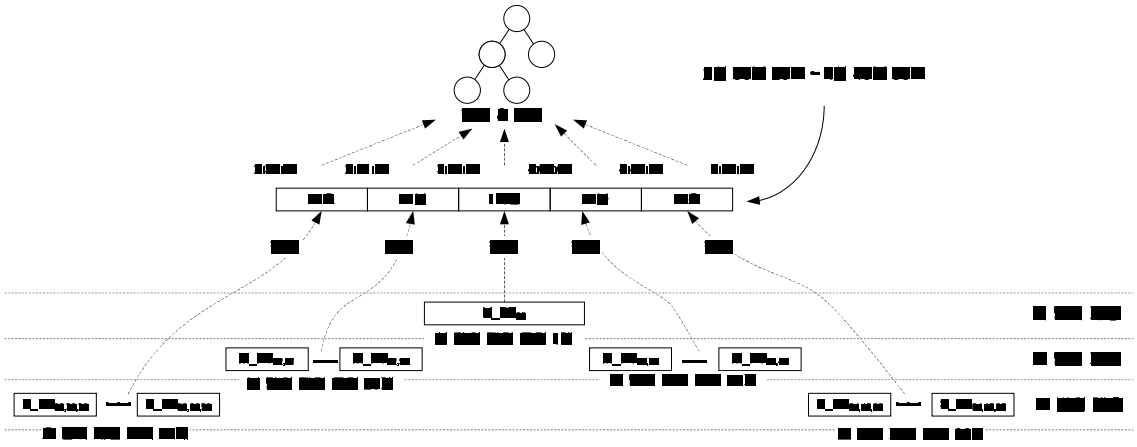
```
01:    cur_file := get_cur_file();
02:    insert_sec_tree( cur_file, sec_pattern_tree );
03:    cur_sec_num := get_cur_sec_num( cur_file );
04:    if cur_sec_num%60 = 0   then
```

```
05:         cur_sec_tree_list := get_cur_sec_list( cur_file );
06:         new_min_tree := MergePatternTree( cur_sec_tree_list );
07:         insert_min_tree( cur_file, new_min_tree );
08:         cur_min_num := get_cur_min_num( cur_file );
09:         if cur_min_num%60 = 0   then
10:             cur_min_tree_list := get_min_sec_list( cur_file );
11:             new_hour_tree := MergePatternTree( cur_min_tree_list );
12:             insert_hour_tree( cur_file, new_hour_tree );
13:             cur_hour_num := get_cur_hour_num( cur_file );
14:             if cur_hour_num%24 = 0   then
15:                 cur_hour_tree_list := get_hour_sec_list( cur_file );
16:                 new_day_tree := MergePatternTree( cur_hour_tree_list );
17:                 insert_day_tree( cur_file, new_day_tree );
18:                 create_new_file();
19:             end if //일 범위 패턴트리 생성 종료
20:         end if //시 범위 패턴트리 생성 종료
21:     end if //분 범위 패턴트리 생성 종료
22:     result := TRUE;
23:     return result;
End
```

계층적으로 생성된 패턴트리들은 파일에도 계층적으로 입력이 된다. 패턴트리의 크기는 가변이지면 하루에 생성되는 패턴트리의 개수는 고정 개수이기 때문에 파일의 구성은 고정크기의 패턴트리 인덱스가 먼저 쓰여지고 나오고 가변크기의 패턴트리가 나중에 나온다. 다음 절은 이러한 구성을 가지는 파일로부터 원하는 패턴트리들을 읽어내는 방법을 설명한다.

3.4 계층형 패턴트리를 이용한 임의 시간 범위의 순차패턴 분석 방법

디스크에는 초, 분, 시, 일 범위의 패턴트리들이 저장되어있다. 따라서 임의의 시간 범위를 가지는 패턴 분석이 요청되면 디스크에 입력된 초, 분, 시, 일 범위에 맞게 입력된 시간 범위를 나누어야 한다. 이 때 디스크에서 읽는 패턴트리의 개수를 줄이기 위해 최대한 긴 시간 범위를 가지는 시간 범위로 입력된 시간을 나눈다. [그림 3-6]은 입력된 시간 범위에 대한 패턴트리를 얻는 과정이다. H_PT_i 는 i 시의 시 범위 패턴를, $M\_PT_{i,j}$ 는 i 시 j 분의 분 범위 패턴트리를, $S\_PT_{i,j,k}$ 는 i 시 j 분 k 초의 초 범위 패턴트리를 가리킨다.



[그림 3-6] 임의 시간 범위의 패턴 분석 과정

[그림 3-6]에서는 2 시 30 분 20 초부터 4 시 45 분 35 초까지의 순차패턴을 요청하였다. 먼저 시간을 초, 분, 시에 맞게 나누어야 한다. 따라서 입력된 시간 범위는 2 시 30 분 20 초부터 40 초, 29 분, 1 시간, 45 분, 30 초로 나뉘어 지고 각 시간 범위에 해당하는 패턴트리를 디스크에서 읽는다. 트리를 읽을 때 같은 범위의 패턴트리를 합병하고, 각 시간 범위의 읽혀진 패턴트리들은 하나의 패턴트리로 합병된다. 하나로 합병된 패턴트리는 사용자에게 보내기 전에 정제의 과정을 거쳐야 한다.

정제는 패턴트리에서 빈발하지 않는 이벤트들을 삭제하는 과정이다. 이 과정은 다수의 패턴트리를 합병했을때 짧은 패턴트리에는 빈번하게 등장하였지만 긴 시간 범위로 보았을 때는 빈발하지 않은 패턴들을 제거해주기 위함이다. 패턴트리 루트 노드의 빈발횟수는 그 패턴트리를 이루고 있는 순차들의 개수이므로 이 수에 사용자가 정의한 Min_sup 값을 곱한 값보다 작은 빈발횟수를 가지는 이벤트를 패턴트리에서 지워준다. [그림 3-4]의 합병된 패턴트리에서 루트 노드의 등장 횟수가 7 이고 Min_sup 가 0.5 이라면, 패턴트리에서 등장 횟수 3.5 가 넘지 않는 < a, b >와 < b, a >는 빈발하지 않다고 판단이 되어 삭제가 된다.

[알고리즘 3-6]는 패턴트리를 정제하는 알고리즘이다. 입력 값은 정제할 패턴트리이다. 줄 01 에서 정제 할 기준 값인 yardstick 을 가져온다. yardstick 은 패턴트리 루트 노드의 빈발횟수에 min_sup 를 곱하여 계산이 된다. 줄 02 에서 루트 노드를 가져오고 줄 04 에서 루트 노드의 prefix 노드를 가져와서 정제를 시작한다. 줄 06 에서 가져온 이벤트 노드의 빈발횟수를 측정하여 yardstick 보다 작으면 패턴이 아니라고 판단하여 줄 07 에서 그 이벤트 노드와 이벤트 노드의 모든 자식 노드들을 삭제한다. 그리고 줄 08 에서 삭제된 이벤트 노드의 부모 노드를 가리키는 base_node 의 다음 prefix 노드를 가져와서 탐색한다. 09 에서 만약 이벤트 노드의 빈발횟수가 yardstick 보다 클 경우, 줄 11 에서 현재 이벤트 노드의 다음 prefix 노드를 가져와서 탐색을 진행한다. 만약 다음 prefix 노드가 없을 경우에는 searching_node 가 null 이 되고 이럴 경우에는 줄 05 에서 searching_node 가 비었다고 판단되어 반복문을 중단한다.

[알고리즘 3-6] 패턴트리 정제 알고리즘

Algorithm RefinePatternTree (pattern_tree)

Input

pattern_tree : 정제될 패턴트리

Output

refined_pattern_tree : 정제된 패턴트리

Variables

Min_sup	:	빈발한지 여부를 결정하는 기준 % 값. 전역 변수
yardstick	:	트리에서 삭제할 패턴의 기준 값
root_node	:	루트 노드
base_node	:	현재 정제 할 이벤트 노드의 suffix 노드
searching_node	:	현재 정제 할 이벤트 노드

```
Begin
01:     yardstick := get_yardstick( pattern_tree, Min_sup );
02:     root_node := get_root( pattern_tree );
03:     base_node := root_node;
04:     searching_node := get_next_prefix( base_node );
05:     while is_empty( searching_node ) = FALSE
06:         if is_frequent( searching_node, yardstick ) = FALSE then
07:             delete_pattern( searching_node );
08:             searching_node := get_next_prefix( base_node );
09:         else
10:             base_node := searching_node;
11:             searching_node := get_next_prefix( base_node );
12:         end if
13:     end while
End
```

[알고리즘 3-7]은 임의의 범위를 가지는 순차패턴 분석 알고리즘이다. [알고리즘 3-2]는 입력된 순차들에서 패턴트리를 생성하는 과정이지만, [알고리즘 3-7]은 이미 생성된 여러 시간 범위의 패턴트리들로부터 원하는 시간 범위의 패턴트리를 생성하는 과정이다. 알고리즘은 [그림 3-6]에서 나오는 것과 같이 5개의 시간 범위로 나뉘어져 진행이 된다. 입력된 시간 범위의 앞부분 초는 줄 05에서 줄 10까지, 앞부분 분은 줄 11에서 줄 16까지, 시간은 줄 17에서 줄 22까지, 뒷부분 분은 줄 23에서 줄 28까지, 뒷부분 초는 줄 29에서 에서 줄 34까지에서 각각 비슷한 방법으로 진행된다. 먼저 해당 시간 범위를 입력된 시간 범위에서 가져오고 해당 시간 범위에 속하는 패턴트리를 디스크에서 읽어와서 합병 후, merged_tree_list 합병된 트리를 추가시킨다. merge_tree_list는 요청된 시간 범위에 속한

여러 시간 범위의 패턴트리를 저장한 후 줄 36에서 담고있는 트리들을 합병하고, 줄 38에서 정제 후 사용자에서 결과 트리를 반환한다.

[알고리즘 3-7] 임의 시간 범위에 대한 순차패턴 분석 알고리즘

Algorithm ReadPatternTree (time_range)

Input

time_range	:	원하는 패턴트리의 시간 범위

Output

result_pattern_tree	:	결과로 얻어올 패턴트리

Variables

file_list	:	입력된 시간 범위에 속하는 파일 리스트
file	:	탐색될 파일
searching_time	:	한 파일내의 탐색할 시간 범위
sec_range	:	탐색 할 초 범위 시간 범위
min_range	:	탐색 할 분 범위 시간 범위
hour_range	:	탐색 할 시 범위 시간 범위
temp_tree_list	:	디스크에서 읽어온 트리 리스트를 담을 리스트
merged_tree	:	합병된 결과를 패턴트리
merged_tree_list	:	합병된 패턴트리들을 담을 리스트

Begin

```
01:     file_list := get_file_list( time_range );
02:     while is_empty( file_list ) = FALSE
03:         file := get_file( file_list );
04:         searching_time := get_file_time_range( file, time_range );
            //시간 범위의 앞부분 초 범위의 패턴트리 탐색
05:         sec_range := get_front_seconds( searching_time );
06:         if is_empty( sec_range ) != FALSE then
07:             temp_tree_list := read_sec_tree( file, sec_range );
08:             merged_tree := MergePatternTree( temp_tree_list );
09:             add_item( merged_tree_list, merged_tree );
10:         end if
            //시간 범위의 앞부분 분 범위의 패턴트리 탐색
11:         min_range := get_front_minutes( searching_time );
12:         if is_empty( min_range ) != FALSE then
```

```
13:             temp_tree_list := read_min_tree( file, min_range );
14:             merged_tree := MergePatternTree( temp_tree_list );
15:             add_item( merged_tree_list, merged_tree );
16:         end if
            //시간 범위의 시간 부분의 패턴트리 탐색
17:         hour_range := get_hours( searching_time );
18:         if is_empty( hour_range ) != FALSE then
19:             temp_tree_list := read_hour_tree( file, hour_range );
20:             merged_tree := MergePatternTree( temp_tree_list );
21:             add_item( merged_tree_list, merged_tree );
22:         end if
            //시간 범위의 뒷부분 분 범위의 패턴트리 탐색
23:         min_range := get_end_minutes( searching_time );
24:         if is_empty( min_range ) != FALSE then
25:             temp_tree_list := read_min_tree( file, min_range );
26:             merged_tree := MergePatternTree( temp_tree_list );
27:             add_item( merged_tree_list, merged_tree );
28:         end if
            //시간 범위의 뒷부분 초 범위의 패턴트리 탐색
29:         sec_range := get_end_seconds( searching_time );
30:         if is_empty( sec_range ) != FALSE then
31:             temp_tree_list := read_sec_tree( file, sec_range );
32:             merged_tree := MergePatternTree( temp_tree_list );
33:             add_item( merged_tree_list, merged_tree );
34:         end if
35:     end while
36:     result_pattern_tree := MergePatternTree( merged_tree_list );
37:     result_pattern_tree := RefinePatternTree( result_pattern_tree );
38:     return result_pattern_tree;
End
```

임의의 시간 범위에 해당하는 패턴트리를 찾는 [알고리즘 3-7]은 한 파일 뿐만 아니라 여러 파일에 걸쳐서도 패턴 분석이 가능하다. 이것은 패턴 분석의 범위가 하루에 국한되는 것이 아니라 여러 날에 걸쳐서 발생되는 순차패턴 분석도 가능하다는 의미이다.

4. 성능평가

본 장에서는 제안방법과 기존 방법의 순차패턴 분석 시간을 비교 평가 한다. 4.1 절에서는 평가하는 시스템의 성능과 제안방법과 비교할 방법, 그리고 입력되는 순차들의 구성에 대하여 설명한 후, 4.2 절에서 실험결과를 설명한다.

4.1 평가환경

본 논문에서 제안한 저장 방법은 C++로 개발되었으며, 시스템은 Intel Pentium 4 3.00GHz CPU, 3.24GB 의 램을 가지고 있고 Microsoft Windows XP 가 OS 로 설치되어 있다.

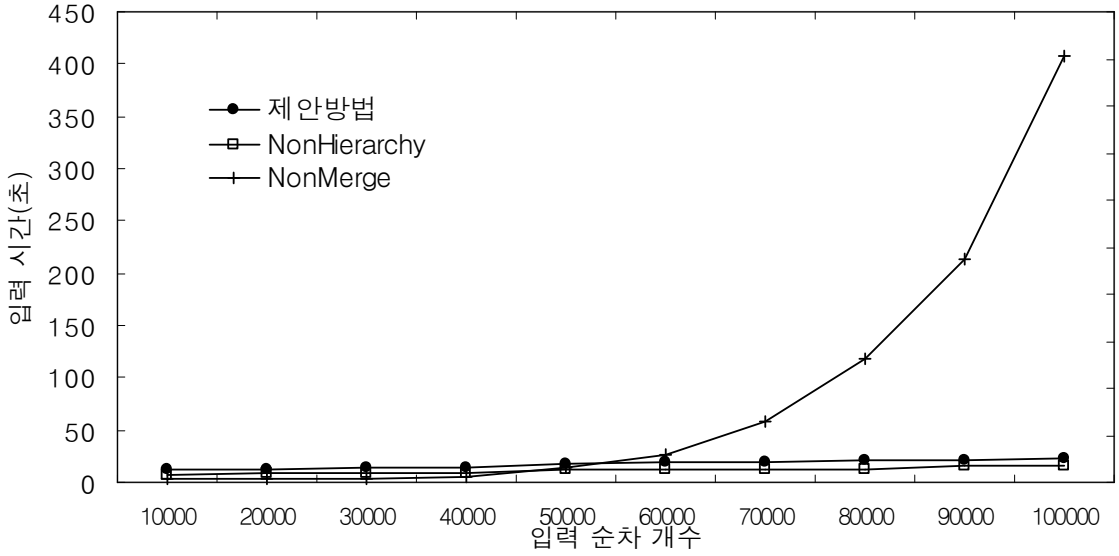
기존의 데이터 스트림에서 패턴 분석 방법으로는 임의 시간 범위를 가지는 과거 순차패턴 분석을 하기 어려웠다. 따라서 제안되는 패턴 분석 방법과 비교하기 위하여 두 가지 방법으로 비교할 방법을 구현한다. 첫 번째 방법은 제안방법의 패턴트리 합병의 장점을 보이기 위하여 트리의 합병이 없이 입력된 시간 범위에 속하는 모든 순차 데이터 스트림을 사용하여 요청된 시간 범위의 패턴 분석을 하는 방법으로 구현이 된다. 이 방법을 NonMerge 방법이라고 한다. 기존의 데이터 스트림에서의 순차패턴 분석 방법들은 트리 합병이 없으므로 모두 NonMerge 방법에 속한다. 비교할 NonMerge 방법은 MILE 을 기반으로 구현되었이다. 두 번째 방법은 제안방법에서 계층적 패턴트리의 장점을 보이기 위하여 생성되는 모든 기본 시간 범위의 패턴트리를 디스크에 저장한다. 그리고 원하는 시간 범위가 요청되면 디스크에서 시간 범위에 해당하는 기본 범위 패턴트리들을 읽고 합병하는 방법으로 구현이 된다. 이 방법을 NonHierarchy 방법이라고 한다. 기본 NonHierarchy 방법은 MILE 를 이용하여 구현이 되지만, 여러 순차패턴 분석 알고리즘을

통하여도 구현이 될 수 있다. 제안하는 논문의 순차패턴 분석을 사용하는 NonHierarchy, MILE 방법을 이용한 NH-MILE, SDMS 를 이용한 NH-SDMS, 그리고 FP-stream 방법으로 구현한 NH-FP 방법을 가지고 제안방법과 비교 평가한다.

입력되는 순차 데이터 스트림들은 일반적인 순차패턴 분석의 성능 평가에 널리 사용되는 IBM synthetic market-basket data generator 를 이용하여 생성한다. 총 100M 개의 순차는 총 1K 개의 이벤트를 가지고 평균 5 의 길이로 생성한다. 또한 총 패턴의 개수는 10000 개이고 평균 패턴의 길이는 최대 4 로 제한된다. 그리고 이벤트들간의 상관계수는 0.25, 평균 신뢰도는 0.75 이다.

4.2 성능평가

본 절에서는 제안방법과 기존 방법으로 구현한 NonMerge 와 NonHierarchy 의 성능을 비교한다. 비교하는 항목은 순차패턴 분석으로 인한 데이터 입력시간의 변화, 패턴트리 검색시간, 메모리 사용량이다. 먼저 [그림 4-1]은 순차패턴 분석으로 인한 데이터 스트림 입력시간의 변화를 측정한 결과이다. 실험은 입력되는 순차 데이터 스트림의 개수를 변화시켜 가며 입력시간의 변화를 특정하였다.

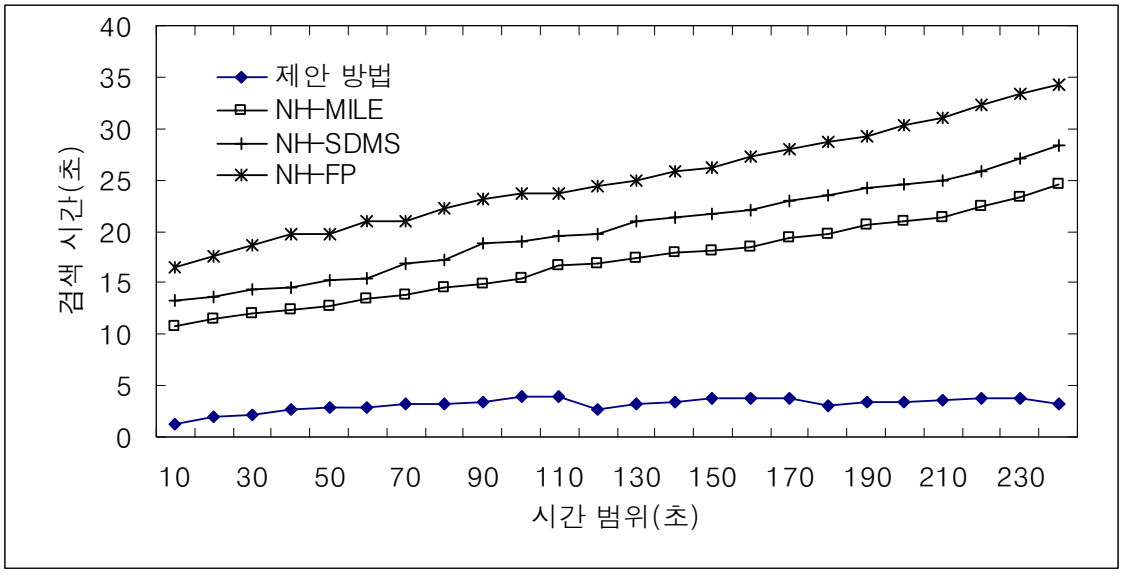


[그림 4-1] 패턴 분석으로 인한 입력시간 변화의 비교

계층적 트리 구조를 사용하지 않는 NonHierarchy 방법은 제안방법보다 약 70% 정도 빠른 입력시간을 보였다. 제안방법이 더 느린 입력시간을 보이는 것은 NonHierarchy 방법은 Win_size 만큼의 순차 데이터 스트림이 입력 될 때마다 기본 범위 패턴트리를 생성하지만 제안방법은 기본 범위 패턴트리 뿐만 아니라 1 초, 1 분마다 계층적으로 높은 시간 범위의 패턴트리를 생성하기 때문이다.

트리 합병을 사용하지 않는 NonMerge 방법은 입력량이 적을 때는 제안방법과 NonHierarchy 방법보다 빠른 데이터 입력시간을 보이다가 입력량이 커질수록 데이터 스트림이 입력되는 시간의 증가가 커지는 것을 보였다. NonMerge 방법은 순차 데이터 스트림이 입력되고 있을 때 패턴트리를 생성하지 않고, 패턴 분석이 요청되었을 때 시간 범위에 해당하는 순차 데이터 스트림을 메모리에서 읽어서 연산한다. 따라서 모든 순차 데이터 스트림을 메모리에 유지하고 있어야 하므로 입력량이 작을 때는 빠른 데이터 입력시간을 보이다가 입력량이 많아지면 사용할 수 있는 메모리의 양이 줄어들기 때문에 입력시간이 증가한다.

제안방법, NonMerge, 그리고 NonHierarchy 를 사용하였을 때의 데이터 입력시간의 변화를 비교한 결과, NonMerge 방법과 같이 트리 합병 방법이 사용이 되지 않으면 긴 시간 범위를 가지는 패턴트리 검색 요청이 입력되면 많은 양의 순차 데이터 스트림을 메모리에 저장하기 때문에 메모리 부족이 발생하여 데이터 스트림의 입력시간이 증가하는 결과를 발생시킨다.



[그림 4-2] 패턴 검색시간 비교

[그림 4-2]는 다양한 시간 범위에 대한 패턴 검색시간 비교이다. NonMerge 방법의 검색시간은 약 14 초에서 212 초까지의 검색시간을 가진다. 이러한 검색시간은 빠른 검색시간을 요구하는 데이터 스트림 환경에 적합하지 않기 때문에 그래프에 추가를 하지 않았다. 대신에 다양한 방법으로 구현된 NonHierarchy 방법을 사용해서 제안방법의 검색시간을 비교했다. 디스크에는 1 시간 분량의 계층형 패턴트리가 있으며, 검색에 사용되는 시간 범위를 임의로 정하여 50 번을 연산한 후 검색시간의 평균값을 트리에 표시하였다. 예를 들어 20 초 시간 범위에 대한 검색시간은 디스크에 있는 한 시간의 범위

안에서 임의로 20 초 시간을 검색하여 시간을 측정한다. 이러한 방법을 50 번 수행한 평균값을 그 시간 범위에 대한 검색시간으로 사용한다.

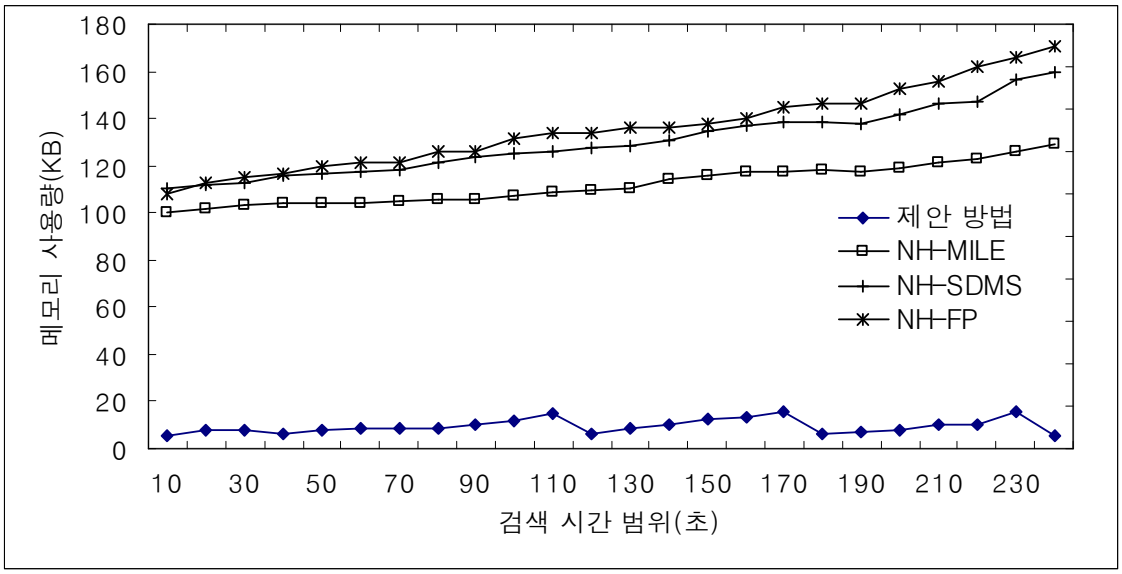
검색의 결과는 NonMerge 의 경우 가장 느린 검색시간을 보였다. 이는 패턴 분석이 요청되었을 때 메모리에 있던 순차 데이터 스트림을 한번에 모아서 패턴을 분석하기 때문이다. 예를 들어 20 초간의 패턴 분석의 경우 메모리에는 20 초 동안 입력되었던 약 20000 개의 순차를 유지하고 있으며, 20 초에 해당하는 패턴 분석의 요청이 입력되면 20000 개의 순차를 모두 읽어 패턴트리를 생성한다. 따라서 NonMerge 방법에서는 가장 느린 패턴 분석의 성능을 보였다.

NonHierarchy 방법은 Win_size 개의 패턴이 입력될 때마다 기본 범위의 패턴트리를 생성하여 디스크에 저장하고 패턴 분석이 요청되면 요청된 시간 범위에 포함되는 기본 범위 패턴트리들을 읽고 합병하여 결과를 얻어낸다. 따라서 시간 범위가 클수록 많은 양의 기본 범위 패턴트리를 읽어 합병을 해야 되므로 [그림 4-2]의 결과와 같이 시간 범위 증가에 따라 선형적인 패턴 분석 시간의 증가를 보였다. 그러나 200 초 범위의 패턴 분석을 하는데 소요되는 시간이 대략 2 초 정도이면 1 시간의 패턴 분석에는 36 초의 시간이 걸린다. 따라서 긴 시간 범위의 검색은 그만큼 패턴 분석 시간이 길어지는 단점이 있다.

제안방법의 검색시간은 NonMerge 나 NonHierarchy 방법보다 매우 낮은 시간을 보이고, 또한 시간 범위가 커질수록 검색시간이 커다란 변동을 보이지 않는 장점을 보인다. 이 것은 계층형으로 패턴트리를 구축해 놓았기 때문에 시간 범위가 커지면 작은 시간 범위 패턴트리 여러 개 대신에 한 개의 큰 시간 범위 패턴트리를 읽기 때문이다. 그리고 120 초, 180 초, 240 초 검색의 경우 그 전 시간 범위 때보다 검색시간이 다소 낮아지는 모습을 보인다. 이 것은 임의로 정한 시간 범위 중에 분 범위 패턴트리가 반드시 1 개, 2 개 또는 3 개 포함되기 때문이다. 시간 범위가 커질수록 선형적으로 검색시간이 증가하지 않는 점은 시간

범위가 늘어나면 여러 개의 초 범위 패턴트리 대신 분 범위 패턴트리를 탐색할 확률이 늘어나기 때문이다.

[그림 4-2]의 결과로 보면 NonMerge 와 같이 트리 합병을 하지 않으면 순차 데이터 스트림을 메모리에 담고 있다가 한꺼번에 패턴 분석을 하기 때문에 매우 높은 검색시간을 보이고, NonHierarchy 와 같이 계층적으로 트리를 구성하지 않으면 요청된 검색시간 범위가 늘어날수록 선형적으로 검색시간이 증가하는 것을 알 수 있다. 그러나 제안된 방법과 같이 패턴트리 합병과 계층형 트리 구조를 같이 사용하면 요청된 패턴 분석의 시간 범위에 상관없이 빠른 검색시간을 보였다.



[그림 4-3] 메모리 사용량의 비교

[그림 4-3]은 패턴 분석 시간 범위에 따라 달라지는 메모리 사용량을 나타낸다. 메모리 사용량을 측정할 때 사용되는 방법은 검색할 때와 동일하다. 시간 범위를 1 시간 내에서 임의로 정하여 검색 후 메모리 사용량의 최대치를 알아보고 이렇게 50 번을 측정하여 평균값을 그래프에 나타낸다.

NonMerge 방법은 대략 800KB 에서 10000KB 까지의 메모리를 사용한다. 따라서 검색과 마찬가지로 지나친 메모리 소비로 인하여 그래프에 포함되지 못하였다. NonMerge 방법은 모든 순차 데이터 스트림에서 한꺼번에 패턴트리를 생성한다. 따라서 시간 범위가 커질수록 많은 양의 순차 데이터 스트림을 메모리에 담고 있어야 되므로 메모리 사용량이 증가한다.

NonHierarchy 방법은 패턴 분석 요청이 입력되면 요청된 시간 범위에 해당하는 모든 기본 범위 패턴들을 디스크에서 읽어, 합병한 후 결과 패턴트리를 생성한다. 따라서 요청된 패턴 분석의 시간 범위가 커질수록 디스크에서 읽어야 하는 기본 범위 패턴트리의 개수는 많아지게 되어, [그림 4-3]과 같이 시간 범위가 커짐에 따라 선형적인 메모리 사용량의 증가를 나타낸다.

제안방법의 메모리 사용량의 변화는 검색시간의 변화와 비슷한 모습을 보인다. 이것은 검색할 때의 디스크에서 호출되는 패턴트리의 양의 변화에 따라 메모리 사용량도 변하기 때문이다. 예를 들어 120 초 범위의 검색 때는 반드시 분 범위 패턴트리 1 개를 읽어올 경우가 발생하고 180 초의 검색 시에는 분 범위 패턴트리 2 개를, 240 초 검색 시에는 반드시 분 범위 패턴트리 3 개를 읽어온다. 따라서 이러한 경우에는 낮은 메모리 사용량을 보인다.

트리 합병 방법을 사용하지 않거나 계층형 트리 구조를 사용하지 않는 NonMerge 와 NonHierarchy 방법 모두 요청된 패턴트리의 시간 범위가 크면 클수록 필요한 메모리 양이 선형적으로 증가됐다. 그러나 제안방법과 같이 트리 합병과 계층형 트리 구조를 사용하면 패턴 검색 요청의 시간 범위가 증가하여도 낮은 메모리 사용량을 보였다.

6. 결론 및 향후 연구

본 논문에서 임의의 시간 범위에 대한 검색을 위해 계층형 패턴트리를 이용한 순차패턴 분석 방법을 제안하였다. 입력되는 순차 데이터 스트림들에서 패턴트리를 생성할 때 중복되는 패턴 분석을 방지하여 패턴 분석의 시간을 낮춘다. 또한 생성된 패턴트리들을 초, 분, 시, 일 범위의 계층으로 구성하여 임의의 시간 범위에 대한 패턴 분석이 요청되면 최대한 큰 시간 범위의 패턴트리들만을 읽고 합병하여 사용자가 원하는 시간 범위의 패턴을 찾는다. 기존의 순차패턴 분석 방법은 임의 시간 범위에 대한 패턴 분석을 하려면 해당하는 모든 순차 데이터 스트림을 가지고 있어야 한다. 따라서 만약 사용자가 원하는 시간 범위가 커지면 가지고 있어야 하는 순차 데이터 스트림의 양이 많아지기 때문에 제안방법을 사용했을 때보다 데이터 입력시간이 20 배 이상 느려진다. 또한 제안방법은 이미 분석된 패턴트리들만을 읽어 결과 패턴을 생성하기 때문에 3 분 정도의 패턴을 분석할 때 기존의 방법으로 패턴을 분석할 때보다 2000 배 이상 빠른 검색시간과 0.003 배의 메모리 사용량을 보였다.

제안방법은 현재 입력되고 있는 순차 데이터 스트림에 대한 패턴분석 뿐만 아니라 과거에 입력되었던 순차 데이터 스트림의 패턴분석도 지원한다. 따라서 요일이나 계절에 따라 패턴이 바뀌는 쇼핑 트랜잭션 같은 응용에서 향후 패턴의 예측을 위한 과거 임의 시간 범위의 패턴분석을 가능하게 한다.

입력되는 순차 데이터 스트림에서 이벤트 패턴트리를 생성하려면 같은 이벤트를 여러 번 읽는 연산의 부담이 있다. 만약 순차 데이터 스트림의 입력 속도가 패턴트리 생성시간에 비해서 빨라지면 패턴분석을 위해 입력되는 순차 데이터 스트림이 대기해야 된다. 따라서 순차 데이터 스트림을 단 한번만 접근하여 빠르게 패턴트리를 생성하는 알고리즘의 개발이 필요하다.

참고 문헌

[1] A. Arasu, B. Babcock, S. Babu, J McAlister and J. Widom, “Characterizing Memory Requirements for Queries over Continuous Data Streams,” Symposium on Principles of Database Systems, pp. 221-232, 2002.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani and J. Widom, “Models and Issues in Data Stream Systems,” the Symposium on Principles of Database Systems, pp. 1-16, 2002.

[3] A. Arasu, S. Babu, and J. Widom, The CQL continuous query language: semantic foundations and query execution, Technical Report, Stanford University. 2003

[4] A. Arasu and J. Widom, Resource Sharing in Continuous Sliding-Window Aggregates, Technical Report, Stanford University, 2004.

[5] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik, "Monitoring Streams - A New Class of Data Management Applications," International Conference on Very Large Database, pp.215-226, 2002.

[6] J. H. Chang and W. S. Lee, “Finding Recent Frequent Itemsets Adaptively over Online Data Streams,” Special Interest Group on Data Mining and Knowledge Discovery, pp. 487-192, 2003.

[7] G. Chen, X. Wu, and X. Zhu, "Sequential Pattern Mining in Multiple Streams," International Conference on Data Mining, pp. 585-588, 2005.

[8] G. Das, K.I. Lin, H. Mannila, G. Renganathan, and P. Smyth, "Rule Discovery from Time Series," Knowledge Discovery and Data Mining, pp. 16-22, 1998.

[9] M.M. Gaber, S. Krishnaswamy, and A. Zaslavsky, "Ubiquitous Data Stream Mining," Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2004.

[10] H. Garcia-Moling, J.D. Ullman, and J. Widom, Database System Implementation, Prentice Hall International, 2000.

[11] C. Giannella, J. Han, J. Pei, X. Yan and P.S. Yu, “Mining Frequent Patterns in Data Streams at Multiple Time Granularities,” Next Generation Data Mining, MIT Press, 2003.

[12] J. Han, J. Pei and Y. Yin, “Mining Frequent Patterns without Candidate Generation,” The ACM Special Interest Group on Management of Data, pp 1-12, 2000.

[13] H.F. Li, S.Y. Lee, and M.K. Shan, "DSM-TKP: Mining Top-K Path Traversal Patterns over Web Click-Streams," Web Intelligence, pp. 326-329, 2005.

[14] H.F. Li, S.Y. Lee, and M.K Shan, "On Mining Webclick Streams for Path Traversal Patterns," World Wide Web Conference, pp. 404-405, 2004.

[15] A. Marascu and F. Masseglia, “Mining Sequential Patterns from Temporal Streaming Data,” In Proceedings of the first ECML/PKDD Workshop on Mining Spatio-Temporal Data, 2005.

[16] T. Oates, and P.R. Cohen, "Searching for Structure in Multiple Streams of Data," International Conference on Interetional Conference on Machine Learning, pp. 346-354, 1996.

[17] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.C. Hsu, "Mining sequential patterns by pattern-growth: The prefixspan approach," IEEE Transaction Knowledge Data Engineering, Vol. 16, No. 11, pp. 1424-1440, 2004.

[18] R. Srikant and R. Agrawal, “Mining sequential patterns: Generalizations and performance improvements,” International Conference on Extending Database Technology, pp. 3-17, 1996.

[19] Q. Zhao, and S.S. Bhowmick, Sequential Pattern Mining: A Survey, Technical Report Centre for Advanced Information Systems, School of Computer Engineering, Nanyang Technological University, Singapore. 2003.

[20] 백성하, 유병섭, 조숙경, 배해영, “다중 연속질의에서 슬라이딩 윈도우 집계질의 최적화를 위한 선형 자원공유 기법,” 정보과학회논문지. Vol. 33, No. 6, pp. 563-577, 2006.

[21] 신재진, 김호석, 김경배, 배해영, “RSP-DS: 데이터 스트림에서의 실시간 순차패턴 분석,” 한국멀티미디어학회 논문지, Vol. 9, No. 9, pp. 1118-1130, 2006.

감사의 글

졸업입니다. 학생으로써의 끝보다는 사회인으로써의 새로운 시작에 가슴이 벅차오릅니다.

데이터베이스 연구실을 지도해주시고 졸업까지 도와주신 지도 교수님이신 배해영 교수님께 깊은 존경과 감사를 드립니다. 그리고 졸업에 신경 써주신 유형선 교수님과 대학원 연구실을 처음으로 접하게 해주신 이필규 교수님, 또한 항상 인자하신 웃음의 이정현 교수님과 졸업논문을 심사해 주신 이균하 교수님, 이주홍 교수님, 그리고 훌륭하게 가르쳐주신 인하대학교 컴퓨터정보공학과의 모든 교수님께 감사를 드립니다.

졸업 후에도 바쁘신 와중에도 직접 찾아오셔서 연구실을 지도해 주신 조숙경, 김경배, 오영환 박사님께 감사를 드리고, 제가 석사 1, 2 차때 지도해주신 박순영, 김명근 박사님께도 감사를 드립니다. 그리고 논문을 지도해주고 기타 여러가지(?)를 가르쳐준 호석이형, 그리고 처음 들어와서 연구실에 적응할 때까지 굉장히 많은 가르침과 도움을 준 용일이형의 박사과정 졸업을 경축드리면서 감사를 전합니다. 그리고 드높은 카리스마와 힘을 가진 병섭이형, 강한 팔뚝을 가지신 상훈이형, 쇼핑에 있어서 천재적으로 박학다식한 동욱이형의 지도에 감사를 드립니다.

직장 생활을 하느라 바쁜 졸업생 형, 누나들에게도 감사합니다. 명호형, 현진이 누나, 희택이형, 호선누나, 영철이형, 일국형등 처음 연구실 왔을 때 있었던 멋진 형, 누나들처럼 되고 또한 더욱 노력하겠습니다. 나이는 똑같지만 먼저 사회에 진출한 정현이와 형선이는 기다려라. 졸업하면 막걸리 들고 구로로 찾아가겠다. 그리고 함께 졸업을 하고 싶었지만 운명이 우리 사이를 갈라놓았던 종현이형과 병윤이, 근형이, 비록 동기는 되지 못했지만 즐거웠던 연구실 생활에 고마움을 전합니다. 모두들 다시 대학원을 온다면 함께 하고 싶은 사람들입니다.

졸업을 하니 홀로 남겨진 동기 성하에게 미안합니다. 나 먼저 간다. 딱 4 년만 연구실 열심히 지켜라. 본드 적당히 마시고. 동기인 윤경이도 하루빨리 좋은 곳에 취직하고 좋은 곳으로 장가가기를 바랍니다. 많은 것을 가르쳐 드리진 못했지만 고집세고 못난 제 성질을 잘 받아주던 치수형과 승선이형한테 고마움을 전합니다. 남은 한 학기, 저처럼 긴장 풀다가 낭패보지 마시고 열심히 해서 저보다 훨씬 좋은 곳으로 가길 희망합니다. 연이도 빨리 졸업해서 좋은데 가길 바랍니다. 그리고 항상 연구실을 위해 수고해주시는 애리누나, 선화누나, 그리고 수고해 주셨던 신영이 누나에게도 감사의 말을 전합니다.

대학교에 생활의 많은 것을 가르쳐준 동아리 플라곤에게 조금만 감사합니다. 또한 누구보다도 제 멋대로 하게 믿고 돌봐준 가족들에게 가장 큰 감사를 드립니다.

졸업입니다. 아직 모르는 것도 많고, 배우고 싶은 것도 많습니다. 많이 배우고 싶어서 빨리 사회로 나가고 싶습니다. 언젠가는 잡지 1 면에 얼굴이 실리게 될 그 날까지 더욱 열심히 하겠습니다.