Journal of Korea Multimedia Society Vol. 9, No. 9, September 2006(pp. 1118-1130)

RSP-DS: 데이터 스트림에서의 실시간 순차 패턴 분석

신재진[†], 김호석[††], 김경배[†††], 배해영[††††]

요 약

데이터 스트림에 대한 기존의 패턴 분석 알고리즘은 대부분 속도 향상과 효율적인 메모리 사용에 대하여 연구되어 왔다. 그러나 기존의 연구들은 새로운 패턴을 가진 데이터 스트림이 입력되었을 경우, 이 전에 분석된 패턴을 버리고 다시 패턴을 분석하여야 한다. 이러한 방법은 데이터의 실시간적인 패턴 분석을 필요로 하는 실제 환경에서는 많은 속도와 계산 비용이 소모된다. 본 논문에서는 끊임없이 입력되는 데이터 스트림의 패턴을 실시간으로 분석하는 방법을 제안한다. 이 것은 먼저 빠르게 패턴을 분석하고 그 다음부터는 이전에 분석된 패턴을 효율적으로 갱신하여 실시간적인 패턴을 얻어내는 방법이다. 데이터 스트림이 입력되면 시간 기반 윈도우로 나누어 여러 개의 순차들을 생성한다. 그리고 생성된 순차들의 정보는 해시 테이블에 입력되어 정해진 개수의 순차가 해시 테이블에 채워질 때마다 해시 테이블에서 패턴을 분석해 낸다. 이렇게 분석된 패턴은 패턴 트리를 형성하게 되고, 이 후에 새로 분석된 패턴들은 이 패턴 트리 안의 패턴 별로 갱신하여 현재 패턴을 유지하게 된다. 새로운 패턴 추가를 위해 패턴을 분석할 때 이전에 이미 발견된 패턴이 Suffix로 나올 수 있다. 그러면 패턴 트리에서 이 전 패턴으로의 포인터를 생성하여 중복되는 패턴 분석으로 인한 계산 시간의 낭비를 방지한다. 그리고 FIFO방법을 사용하여 오랫동안 입력이 안 된 패턴을 손쉽게 제거한다. 패턴이 조금씩 바뀌는 데이터 스트림 환경에서 RSP-DS가 기존의 알고리즘보다 우수하다는 것을 성능 평가를 통하여 증명하였다. 또한 패턴 분석을 수행할 데이터 순차의 개수와 자주 등장하는 데이터를 판별하는 기준을 조절하여 성능의 변화를 살펴보았다.

RSP-DS: Real Time Sequential Patterns Analysis in Data Streams

Jae-Jyn Shin[†], Ho-Seok Kim[††], Kyoung-Bae Kim[†††], Hae-Young Bae[††††]

ABSTRACT

Existed pattern analysis algorithms in data streams environment have researched performance improvement and effective memory usage. But when new data streams come, existed pattern analysis algorithms have to analyze patterns again and have to generate pattern tree again. This approach needs many calculations in real situation that needs real time pattern analysis. This paper proposes a method that continuously analyzes patterns of incoming data streams in real time. This method analyzes patterns fast, and thereafter obtains real time patterns by updating previously analyzed patterns. The incoming data streams are divided into several sequences based on time based window. Informations of the sequences are inputted into a hash table. When the number of the sequences are over predefined bound, patterns are analyzed from the hash table. The patterns form a pattern tree, and later created new patterns update the pattern tree. In this way, real time patterns are always maintained in the pattern tree. During pattern analysis, suffixes of both new pattern and existed pattern in the tree can be same. Then a pointer is created from the new pattern to the existed pattern. This method reduce calculation time during duplicated pattern analysis. And old patterns in the tree are deleted easily by FIFO method. The advantage of our algorithm is proved by performance comparison with existed method, MILE, in a condition that pattern is changed continuously. And we look around performance variation by changing several variable in the algorithm.

Key words: Data Stream(데이터 스트림), Sequential Pattern(순차 패턴), Pattern Analysis(패턴 분석), Real Time(실시간)

※ 교신저자(Corresponding Author) : 배해영, 주소 : 인천광역시 남구 용현동 253 인하대학교 컴퓨터공학과 데이터베이스 연구실 하이테크관 1008호(402-751), 전화 : 032)860-8712, FAX : 032)862-9845, E-mail : jaejyn@dblab.inha.ac.kr
접수일 : 2006년 5월 8일, 완료일 : 2006년 7월 26일
[†] 인하대학교 컴퓨터정보공학과
(E-mail : jaejyn@dblab.inha.ac.kr)
[††] 인하대학교 컴퓨터정보공학과
(E-mail : hskim@dblab.inha.ac.kr)
[†††] 정회원, 서원대학교 컴퓨터교육과
(E-mail : gbkim@seowon.ac.kr)
[††††] 인하대학교 컴퓨터정보공학과
(E-mail : hybae@inha.ac.kr)
※ 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성.지원사업의 연구결과로 수행되었음

1. 서 론

최근에 주목을 받고 있는 재정 관리, 과학적 실험의 성능 측정, 네트워트 패킷 모니터링, 교통량 측정, 사용자 클릭 스트림 분석등의 응용들은 데이터 스트림이라고 하는 새로운 형태의 데이터를 다루고 있다. 기존의 데이터베이스에서의 데이터는 시간의 흐름에 관계없이 변화가 거의 없는 특징이 있다. 그러나 데이터 스트림은 기존의 데이터들과는 다르게 그 입력이 끊임없이 이루어지고, 시간에 따라서 데이터의 양과 입력 속도가 바뀔 수 있다. 또한 동시에 다수의 공급원으로부터 데이터가 발생하여 입력되어질 수 있는 특징을 가지고 있다[3].

이와 같은 새로운 특징을 갖는 데이터 스트림의 등장으로 기존의 데이터 마이닝은 데이터 스트림에 맞추어 새롭게 정의되어야 한다. 데이터 스트림은 끊임없이 입력되기 때문에 현재까지의 모든 데이터를 저장한다는 것은 불가능하다. 따라서 마이닝을 할 때도 모든 데이터가 아니라 최근의 데이터로만 마이닝 연산을 수행하여야 한다. 또한 데이터 스트림의 그 특징이 계속해서 바뀔 수 있기 때문에 한 번 마이닝한 결과는 얼마 지나지 않아 다시 마이닝을 수행하여야 한다. 하지만 빠른 속도로 입력되는 대용량의 데이터 스트림의 패턴을 매번 다시 분석하는데엔 많은 메모리와 계산 비용이 소모된다[10].

데이터 마이닝의 한 분야인 순차 패턴 분석이란 데이터들이 순차적으로 발생하였을 경우, 그러한 데이터들 사이에 어떠한 반복되는 순서를 찾아내는 방법이다[11]. 순차 패턴 분석의 대표적인 두 가지 방법으로는 GSP (Generalized Sequential Patterns)과 PrefixSpan 방법이 있다[1,4]. GSP 방법은 어떤 패턴이 빈번하게 발생하면 그 패턴의 일부분도 빈번하게 발생한다는 사실을 기반으로 순차 패턴을 찾는다. 그러나 GSP는 실제 패턴의 후보 패턴이 몇 번 나오는가 계산하는 단계는 실시간 응답을 원하는 데이터 스트림으로 적용하기에는 많은 시간이 걸린다는 단점이 있다. PrefixSpan은 후보패턴을 생성하지 않고 순차들의 패턴을 추출해내는 방법으로서 GSP보다 빠른 순차 패턴 분석이 가능하여 데이터 스트림의 패턴 분석에 적용이 가능하다.

이러한 기존의 순차 패턴 분석을 토대로 스트림에서의 많은 순차 패턴 분석에 관한 연구가 이루어져왔다. 불연속적인 데이터 스트림에서 이벤트 A가 발생한 후에 시간 T안에 이벤트 B가 발생한다는 법칙을 이끌어 내는 것에 대한 연구가 이루어 졌다[7]. 이러한 두 이벤트간의 법칙 발견은 순차 패턴 추출과 유사한 것이다. 다른 연구로는 이벤트 집합 x가 일어난 후 정확히 T시간 후에 이벤트 집합 y가 일어나는 순차 패턴 추출이 있다[8]. 그러나 정확한 시간을 명시하여 순차 패턴을 추출하는 것은 데이터의 종류가 끊임없이 바뀔 수 있는 데이터 스트림에서는 실용적이지 못하다. 스트림에서 순차 패턴을 분석하는 방법으로 StreamPath와 DSM-TKP (Data Stream Mining for Top-k Path Traversal Patterns) 가 제안되었다[6,9]. 하지만 이 방법들은 많은 긴 패턴들 사이에 공통된 짧은 패턴을 중복 탐색해야 하는 단점을 가지고 있다. MILE 방법은 PrefixSpan 을 확장하여 다수의 스트림에서의 패턴 분석 알고리즘이다[2]. MILE은 패턴 발견의 중복을 사전에 예방함으로써 패턴 발견 속도를 높였다.

앞에서 설명한 대로 데이터 스트림의 특징은 끊임없이 바뀔 수 있다. 따라서 기존의 방법들과 같이 한 순간의 패턴 분석만으로 끝나서는 안되고, 계속해서 변하는 패턴을 분석해야만 한다. 그러나 위의 알고리즘들은 한 순간의 패턴 분석에 대해서만 연구를 하였다. 따라서 사용자가 계속해서 실시간으로 분석된 패턴을 요구할 경우 데이터 스트림이 입력될 때마다 계속해서 패턴 분석을 실행해야 하는 단점이 있다.

본 논문에서는 계속해서 변하는 데이터 스트림의 패턴을 실시간으로 분석하는 방법을 제안한다. 제안하는 알고리즘은 입력되는 데이터 스트림를 시간 기반 윈도우로 나누어 여러 개의 순차들을 만든다. 이러한 순차들은 데이터를 키로 갖는 해시 테이블로 관리가 되며, 정해진 개수의 순차가 입력될 경우 매번 해시 테이블에서 패턴 분석이 이루어진다. 분석된 패턴들은 패턴트리를 형성한다. 그리고 이 후에 계속해서 분석된 패턴들은 패턴 트리로 입력이 되며, 오래된 패턴은 패턴 트리에서 삭제하는 방법으로 현재 패턴을 유지하게 된다.

2장에서는 제안하는 알고리즘의 기본이 되는 PrefixSpan과 MILE에 대하여 살펴본다. 그리고 3장에서는 하나의 스트림에서의 실시간 패턴 분석을 통하여 제안하는 알고리즘과 MILE의 성능을 비교 분석한 후, 4장에서 결론과 향후 연구에 대하여 설명한다.

2. 관련 연구

2.1 PrefixSpan

PrefixSpan은 순차 패턴을 발견해내는 한 방법이다. 정해진 개수의 순차가 들어오면 각 데이터들이 얼마나 많은 순차에서 등장하였는지 개수를 센다. 데이터의 개수가 사용자에 의해 정의된 min_sup보다 같거나 크면 자주 등장하는 데이터로 결정되고 패턴으로 선택된다. 그리고 자주 등장하는 항목으로 선택된 데이터들의 Suffix들 중에서도 자주 등장하는 데이터가 그 전에 Prefix로 등장하였던 패턴과 결합하여 패턴으로 결정된다. 이렇게 자주 등장하는 항목의 Suffix를 탐색하여 Prefix와 결합한 후 패턴으로 결정하는 방법이 PrefixSpan방법이다. PrefixSpan은 자주 등장하는 항목을 찾는데 있어서 후보패턴을 생성하지 않기 때문에 다른 순차 패턴 분석 알고리즘보다 월등한 성능을 보인다.

다음 알고리즘은 PrefixSpan의 알고리즘이다.

알고리즘1에서 처음에 진행패턴과 최종패턴은 모두 비어있고, min_sup=2와 3개의 순차 <b,c,g,a,b,a,d>, <b,c,h,f,a,b,d>, <f,c,d,b,e,a,g>가 주어졌다면,

2. 여기서 각 데이터는 a=3, b=3, c=3, d=3, e=1, f=2, g=2, h=1개의 순차에서 등장한다. 따라서 min_sup 보다 같거나 큰 빈번한 항목은 a, b, c, d, f, g가 선택 된다.

7. 1에서 선택된 각 데이터가 진행패턴과 합해져서 새로운 진행패턴을 생성해야 한다. 그러나 현재 빈발패턴은 비어있기 때문에 각 데이터 자체가 진행패턴, <a>, <b>, <c>, <d>, <f>, <g>이 된다.

8. 구해진 진행패턴이 최종패턴에 추가된다.

9. 진행패턴마다 진행패턴의 Suffix를 각 순차에서 구하면 <a>의 경우, <a,b,a,d>, <d,b>, <g>가 구해진다.

10. PrefixSpan (min_sup, 4에서 구한 Suffix들, 진행 패턴, 현재 패턴)을 재귀적으로 호출한다.

이렇게 재귀적인 실행을 통하여 결과로 나오는 패턴들은 같은 Suffix를 같은 parent로 갖는 트리형태로 표현될 수 있으며 최종 모습은 그림 1과 같다.

PrefixSpan 방법 후보 패턴을 생성하지 않는다는 점에서 다른 순차 패턴 분석 알고리즘보다 빠른 성능을 보인다. 그러나 그림 1에서 보이듯이 패턴 분석 과정에 있어서 중복되는 패턴 분석이 빈번히 발생한다. 이 점은 빠른 수행 시간을 요구하는 데이터 스트림 환경에서는 적합하지 못하다. 또한 패턴 트리에서 중복되는 패턴의 저장도 불필요한 메모리의 낭비가 발생하는 단점을 가진다.

2.2 MILE (MIning from muLtiple strEams)

PrefixSpan 방법은 이미 발견되었던 어떤 패턴이 다시 한 번 생성되는 단점이 있다. 예를 들어 그림

알고리즘 1 (PrefixSpan)

Input: min_sup, n개의 순차, 진행 패턴, 최종 패턴 집합

Output: 최종 패턴

```
진행 패턴=<>, 최종 패턴 집합=<>;
n개의 순차에서 min_sup 이상으로 등장한 빈번한 데이터를 탐색;
if 빈번한 데이터가 존재하지 않음 then
    함수를 종료;
end if
foreach 데이터 D ∈ 빈번한 데이터 do
    진행 패턴 = 진행 패턴 + D;
    최종 패턴 집합에 진행 패턴을 추가;
    각 순차에서 진행 패턴의 Suffix들을 탐색;
    call PrefixSpan( min_sup, 찾아진 Suffix들, 진행 패턴, 최종 패턴 집합 );
end for
```

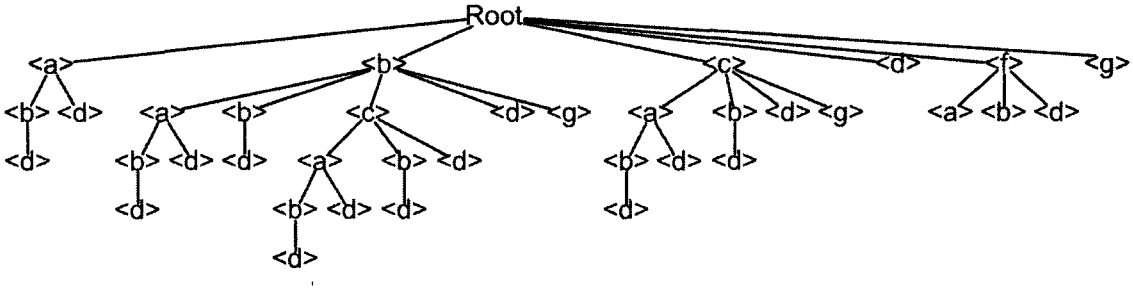


그림 1. PrefixSpan으로 생성한 패턴 트리

1의 경우 첫 레벨의 맨 왼쪽 패턴<a>와 그 자식 패턴 <a,b>, <a,b,d>, <a,d>들은 <b>의 Suffix로도 나오고, <b,c>의 Suffix로, 또 <c>의 Suffix에도 나온다. 패턴의 크기가 커질수록 이렇게 중복되는 패턴 탐색이 많아지게 되어 패턴 탐색 비용을 증가시키고, 패턴 트리를 저장하는 메모리의 크기도 늘어나게 된다.

MILE에서는 다수의 데이터 스트림 소스에서 입력을 받아 튜플 기반 윈도우로 데이터 스트림을 나누어 다수의 순차를 생성한다. 나누어진 순차들은 PrefixSpan방법과 같이 방법으로 패턴분석에 이용된다. 그러나 PrefixSpan과는 다르게 현재 생성하고 있는 패턴의 Suffix가 이미 패턴 트리에 생성이 되어 있다면 더 이상의 패턴 생성을 중단하고 이전 패턴으로 포인터를 형성하는 방법으로 패턴 트리를 구축한다. 이것은 중복되는 패턴 탐색을 사전에 방지하여 패턴 생성의 속도를 높이는 장점이 있다.

앞의 그림 1에서 쓰인 순차와 같은 순차 seq1=<b,c,g,a,b,a,d>, seq2=<b,c,h,f,a,b,d>, seq3=<f,c,d,b,e,a,g>가 각각 세 개의 데이터 스트림에서 구해졌다고 가정하자. 패턴 탐색의 중복 방지를 최대화하기 위하여 상대적으로 나중에 입력된 데이터부터 PrefixSpan방법으로 패턴을 생성해 나간다. 빈번한 데이터들 중에 가장 늦게 들어온 순서가 <d>-><a>-><g>-><c>-><b>-><f>가 된다면 이 순서대로 PrefixSpan을 실행하여 나간다. <c,a>의 패턴을 생성하고 있을 때 <a>를 Prefix로 가지는 <a,b>, <a,d> 패턴은 이미 구해져 있었다. 따라서 더 이상의 패턴 생성을 중단하고 <c,a>에서 <a>를 Prefix로 가지는 <b>, <d>로 포인터를 생성하게 된다. 이렇게 전체 패턴의 생성을 마치면 그림 2와 같이 패턴 트리가 생성된다. 그림 1과는 다르게 더 적은 수의 패턴 생성이 이루어진다. 패턴 <c,a,b,d>와 <c,a,d>의 경우 이전에 구해졌던 <a>로 시작하는 패턴들로의 포인터를 가지고 있다. 또한 패턴 <b,a,b,d>, <b,a,d>, <b,c,a,b,d>, <b,c,b,d>, <b,c,d>의 경우도 이전에 이미 구해졌던 패턴으로의 포인터를 가진다. 패턴의 길이가 길면 길수록, 많은 수의 중복 패턴 분석 방지가 발생하여 PrefixSpan보다 더 빠르게 전체 패턴을 분석 할 수 있다.

MILE은 PrefixSpan을 확장하여 단순한 순차 패턴 분석을 여러 데이터 스트림 사이의 패턴 분석에 적용하였다. 그러나 실제 상황에서 각각의 데이터 스트림의 패턴은 끊임없이 바뀔 수 있다. 따라서 각 데이터 스트림들의 패턴 주기가 다를 수 있다. 그러나 MILE은 튜플 기반 윈도우로 데이터 스트림을 나누어 분석한다. 이러한 방법은 임의로 패턴을 자르는 것이 된다. 따라서 계속해서 여러 데이터 스트림들을 동일한 튜플 기반 윈도우 패턴을 분석하는 것은 무리가 있다.

또, MILE은 PrefixSpan을 이용하여 효율적으로

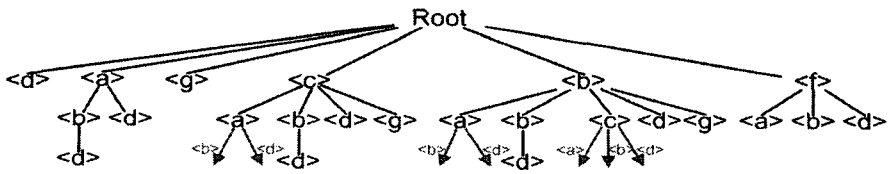


그림 2. MILE에 의해 만들어진 패턴 트리

데이터 스트림들 사이의 패턴을 생성한다. 분석된 패턴은 새로 입력되는 데이터 스트림들이 패턴에 만족하는지 결정할 수가 있다. 하지만 위와 마찬가지로 데이터 스트림의 패턴은 끊임없이 바뀔 수가 있다. 따라서 어느 정도 시간이 지나면 입력되는 데이터 스트림의 패턴들은 거의 이 전에 추출됐었던 패턴들과 다를 수 있다. 그렇다고 계속해서 입력되는 데이터 스트림들로 다시 패턴 트리를 생성하여 이전 패턴 트리와 비교하는 것은 부하가 많이 걸리는 작업이다.

3. RSP-DS (Real time Sequential Patterns analysis in Data Streams)

RSP-DS는 하나의 데이터 스트림에 대한 패턴 분석과 이에 대한 지속적인 갱신에 관한 알고리즘이다. 3.1에서는 RSP-DS에서 쓰이고 있는 용어들을 정의한다. 그리고 3.2에서 기존의 패턴의 갱신을 통하여 현재 패턴을 유지하는 방법을 설명하고 3.3에서는 오랫동안 등장하지 않는 패턴을 현재 패턴에서 제외하는 방법에 대하여 설명한다.

3.1 용어 정의

RSP-DS에서는 하나의 데이터 스트림에서 조금씩 변해가는 패턴을 효과적으로 추출하기 위해 시간 기반 윈도우에 의해 잘려진 다수의 순차 S_i, S_{i+1}, ... , S_{i-n}이 들어오면 S_{i-1}, S_{i-2}, ... ,$S_{i-(n-1)}$의 이전 순차와 같이 패턴 분석을 실행한다.

패턴 분석에 쓰이는 순차의 개수에 대한 용어는 다음과 같이 정의한다.

정의 1: old_seqs

패턴 분석에 사용될 이 전에 등장했었던 순차의 개수이다. 패턴의 등장은 순간적일 수도 있으며, 반대로 드물게 등장할 수도 있다. 패턴이 연속적으로 등장하는 순차들에서 전부 등장한다면 새로운 순차들만으로도 패턴을 분석해 낼 수 있지만, 어떠한 패턴이 드물게 입력되는 경우 이전의 순차들하고 같이 분석해 보지 않으면 그 패턴을 찾기 힘들다. 따라서 패턴을 분석할 때는 이 전에 등장했었던 순차들하고 새로 입력된 순차들을 같이 비교하여 패턴을 분석해 내야 한다.

정의 2: new_seqs

패턴 분석 시에 사용될 새로운 패턴의 개수이다. 패턴 분석 시에는 이 전에 등장한 순차들과 새로 들어온 순차들을 같이 비교해서 패턴 분석을 하여야 하므로 얼마나 많은 새로운 순차들로 패턴을 분석해 낼 것인가는 결정하는 것이 new_seqs이다. 패턴 분석 모두 old_seqs+new_seqs 개수의 순차에서 이루어 진다. 그리고 패턴 분석 후에는 이 중에서 가장 오래된 new_seqs개의 순차들을 제외시킨 순차들을 old_seqs로 정하고 새로 들어온 new_seqs개의 순차들로 다시 패턴을 분석해 나아간다.

3.2 패턴 분석과 패턴 추가

RSP-DS는 한 데이터 스트림에서의 패턴을 찾기 위해 MILE을 이용한다. MILE은 다수의 데이터 스트림으로부터 패턴을 분석해내는 알고리즘인 반면에 RSP-DS는 한 데이터 스트림을 일정한 개수의 순차들로 나누어, 그 순차들로부터 패턴을 분석해 나갈 것이다. (1)에서는 어떻게 한 개의 데이터 스트림에서 분석할 순차들을 얻어내는 것인가와 빠르게 패턴 분석을 하기 위하여 사용될 해시 테이블에 관해 설명한다. 그리고 (2)에서는 해시 테이블을 이용하여 패턴을 분석하는 알고리즘에 관해 설명한다.

(1) 자료 구조

어떤 패턴트리가 이전에 입력된 3개의 윈도우 <b,c,g,a,b,c,d>, <b,c,f,a,b,d>, <f,c,d,b,a,g>를 통하여 추출되었다고 하자. 이 때 순차 <b,d,f,c,a,e>순차가 새로 들어 왔다면, 이 전 순차들에서 제일 처음에 나온 순차 <b,c,g,a,b,c,d>를 제외하고 새로 입력된 순차 <b,d,f,c,a,e>를 포함시켜 다시 패턴을 분석하여야 한다. 즉 old_seqs는 2, new_seqs는 1이 되는 것이다.

그림 3은 old_seqs=2, new_seqs=1일때 패턴 분석을 하는 것을 나타낸다. t_1의 시간에서는 <b,c,g,a,b,c,d>, <b,c,f,a,b,d>, <f,c,d,b,a,g>을 가지고 패턴 분석을 한다. 그러나 t_2의 시간에서는 제일 오래된 순차 <b,c,g,a,b,c,d>를 제외하고 대신에 새로 입력된 순차 <b,d,f,c,a,e>를 포함하여 패턴 분석을 실시한다. 항상 가장 최근에 등장한 3개의 순차들로 패턴을 분석하고 있다.

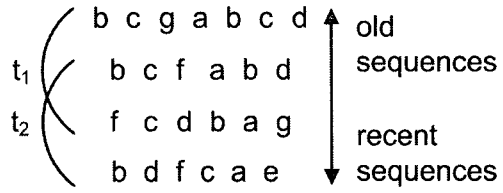


그림 3. 시간에 따라 패턴 분석에 사용되는 순차의 변화

새로운 패턴을 추출할 때 현재 순차를 가지고 패턴 트리를 다시 생성하여 이전 패턴 트리와 비교하는 것은 많은 시간을 필요로 하는 작업이다. 따라서 이전 패턴 트리에서 각각의 가지별로 갱신하여 새 패턴 트리를 생성할 수 있어야 한다. 한 패턴을 구축 할 때 현재 구축되고 있는 패턴의 Suffix가 이 전 패턴과 중복되고 있다는 것을 알고 있으면 이 전 패턴으로의 포인터를 생성하여 중복으로 패턴을 생성하는 것을 막을 수 있다. 예를 들어 패턴 <b,c,a>가 있다면 당연히 패턴 <c,a>또한 존재하여야 한다. 패턴 <c,a>가 패턴 <b,c,a>보다 일찍 발견이 되었다면 <b> 다음에 <c>가 패턴으로 발견되었을 때 단순히 <b>에서 <c,a>까지 포인터를 생성해 줄 수 있다.

이 때 중복 패턴 탐색의 제거를 최대화하기 위하여 어떤 패턴이 다른 패턴의 Suffix로 많이 등장하는지를 알아야 한다. 따라서 등장하는 각 데이터마다 얼마나 많은 Suffix를 가지고 있는 지 기억하여 제일 적은 Suffix를 가진 데이터 먼저 패턴 분석을 하여야 한다. 적은 수의 Suffix를 가진 데이터는 그 만큼 다른 패턴의 Suffix가 될 확률이 높기 때문이다.

패턴 분석에 사용되는 순차들은 빠른 패턴 분석을 위해 해시 테이블로 관리가 된다. 그림 4는 old_seqs는 2이고 new_seqs는 1인 해시 인덱스 이다. 따라서 마지막 1개의 순차 seq3는 새로 들어온 순차이다. 순차 seq_1=<b,c,f,a,b,d>, seq_2=<f,c,d,b,a,g>, seq_3=<b,d,f,c,a,e>로 이루어진 패턴 분석을 마친 해시 인덱스 이다. 해시 인덱스 가리키고 있는 Data Information Table(이하 DIT)는 전체 순차에 대한 통계 정보를 저장고, Order Information Table(이하 OIT)는 각 순차 안에 각 데이터들의 정보를 저장한다.

새로운 순차가 들어오면 OIT에서 제일 오래된 순차에 해당하는 열을 지우고, 새로운 열에 대한 열을 생성하여 기록한다. OIT는 한 순차 안에서의 한 데이터의 순서를 가리키는 nOrder와 이 데이터가 어떠한 패턴의 Prefix일 때 Suffix 데이터를 가리키는 포인

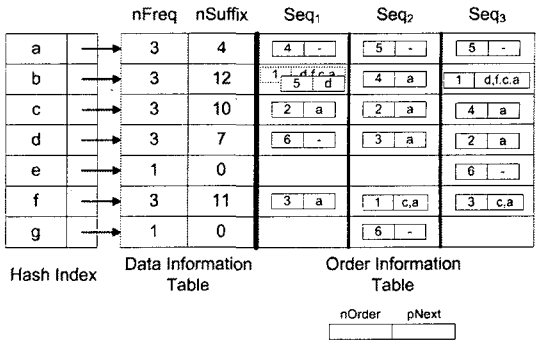


그림 4. 해시 테이블의 예

터 pNext로 구성된다. <b,a>란 순차가 패턴인지 판별하기 위해서 각 순체의 a와 b에 해당하는 nOrder를 살펴본다. Seq_1, Seq_2, Seq_3 세 순차 모두 a가 b보다 나중에 나오므로 패턴으로 판정된다. pNext를 이용하면 패턴 분석 시에 한 데이터가 다른 패턴으로 연결되는 것을 쉽게 파악할 수 있다. 따라서 중복된 패턴 분석을 간단히 방지 할 수 있게 된다.

DIT는 한 데이터가 얼마나 몇 개의 순차에서 등장하는 지를 나타내는 bFreq, 그리고 한 데이터가 각 순차에서 얼마나 많은 Suffix를 갖는지를 나타내는 nSuffix를 가지고 있다. bFreq는 한 데이터가 빈발한지 한 번에 파악할 수 있게 도와준다. 그리고 nSuffix는 다른 패턴의 Suffix로 등장 할 확률이 높은 짧은 길이의 데이터를 먼저 탐색함으로써 패턴 분석의 중복을 최대화할 때 사용된다. 패턴 분석의 중복이 빈번하면 그만큼 중복 탐색 방지도 많아져서 탐색 시간을 줄일 수 있다.

(2) 알고리즘

이 해시 테이블을 이용하여 새로운 패턴을 찾는 알고리즘은 다음과 같다.

그림 5는 알고리즘의 예제를 나타낸 해시 테이블이다. min_sup=2이고, Seq1=<a,b,c,d>과 Seq2=<d,b,c,e>는 이 전에 입력된 순차이고 Seq3=<a,e,b,c>는 새로 들어온 순차이다. 왼쪽 그림은 Seq3이 입력되었을 때의 모습을 나타낸 것이고, 오른쪽 그림은 패턴이 분석 된 후 수정된 해시 테이블을 나타낸 것이다. 오른쪽 그림에서 회색 바탕으로 된 곳이 수정이 일어난 부분이다. 해시 테이블이 수정되는 과정은 다음과 같다. 먼저 Seq3이 입력되면

알고리즘 2 (FindNewPattern)
입력: 패턴 트리 PT, 해시 테이블 HT, 새로 입력된 순차 s_new
출력: 갱신된 해시 테이블 HT

```
old_seqs 만큼 오래된 순차를 OIT에서 제거하고 DIT를 갱신
s_new의 데이터를 순서대로 해시 테이블에 입력하면서 OIT와 DIT를 갱신. 이 때 pNext는 모두 null로 함
foreach 데이터 D := s_new, nSuffix가 작은 D 순서대로 do
  if D의 nFreq >= min_sup then
    D를 PT에 추가
  end if
  foreach 데이터 pre_D := s_new에서 D의 Suffix들 중 nSuffix 가 큰 순서대로 do
    if pre_D의 nFreq >= min_sup then
      D와 pre_D를 연결하여 PT에 추가
      D의 nNext := pre_D
      if pre_D의 pNext != null then
        PT안의 패턴 <D, pre_D>에서부터 pre_D의 pNext로 시작하는 패턴으로 포인터를 생성, 이제 pNext는 탐색하지 않음
      end if
    end if
  end for
end for
```

nFreq와 nSuffix가 갱신된다. 그리고 Seq3에 있는 데이터들 중에 nSuffix가 작은 순서대로 분석을 진행한다. <c>와 <e>가 같은 nSuffix 값을 가지고 있으나 그 중 뒤에 등장한 <c>를 먼저 분석한다. <c>는 nFreq가 2이므로 패턴이 된다. 그리고 <c>의 Suffix가 없으니 탐색을 종료한다.

다음은 <e>를 조사한다. <e>의 nFreq도 2이므로 패턴이 된다. 그리고 각 순차에서 <e>의 Suffix들을 조사한다. 즉, Seq2에서 <>와 Seq3에서 <b,c>에서를 Seq3에 등장하는 데이터들의 nSuffix가 큰 기준으로 조사한다. 그러나 <b>, <c> 모두 한 번만 등장하므로 패턴으로 인식되지 못한다. 따라서 <e>만 패턴으로 인식되고 종료된다.

다음은 nSuffix가 5인 <b>를 조사한다. <b>의 nFreq는 3이므로 패턴으로 인식되고 각 순차에서 <b>의 Suffix들을 조사한다. Seq1의 <c,d>, Seq2의

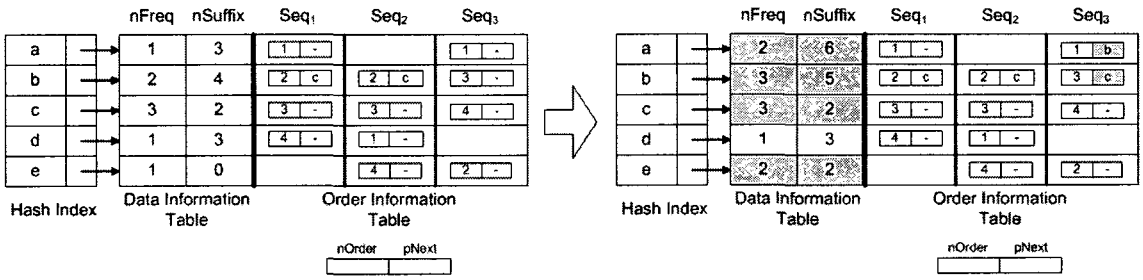


그림 5. FindNewPattern으로 인해 수정된 해시 테이블

<c,e>, Seq3의 <c>를 Seq3에 나오는 데이터의 nSuffix가 큰 기준으로 조사한다. <c>는 세 번 등장하므로 패턴이 된다. 따라서 <b>하고 연결되어 <b,c>가 패턴으로 판정된다. 그리고 <b>의 pNext는 <c>로 결정된다. Seq3에서 더 이상의 데이터가 없으므로 종료된다.

Seq3의 마지막 데이터인 <a>를 조사한다. <a>의 nFreq는 2이므로 패턴이 된다. 그리고 각 순차에서 <a>의 Suffix들을 살펴본다. Seq1에서는 <b,c,d>, Seq2에서는 <>, Seq3에서는 <e,b,c>를 Seq3에 등장하는 데이터 <e,b,c>의 nSuffix가 큰 기준으로 조사를 한다. <b>는 두 번 등장하므로 <a>하고 연결되어 <a,b>가 패턴으로 판별된다. 여기서 <b>의 pNext가 null이 아니고 <c>로 되었다. 따라서 <a,b,c>가 패턴으로 인식된다. 그리고 <c>의 탐색은 이루어지지 않는다. <e>는 한 번 등장하므로 패턴이 아니다.

이렇게 데이터 탐색은 nSuffix가 작은 순서대로, 탐색되는 데이터의 Suffix 탐색은 nSuffix가 큰 순서대로 탐색을 한다. 작은 순서대로 탐색하는 이유는 길이가 짧은 데이터를 먼저 탐색하여 다른 패턴의 부분집합을 먼저 구하기 위함이다. 또한 nSuffix가 큰 순서대로 탐색하는 것은 현재 탐색하고 있는 패턴의 가능한 긴 부분 집합 패턴을 찾기 위해서이다.

3.3 패턴 삭제

여태까지 패턴의 추가를 살펴보았다. 그러나 계속해서 패턴을 추가하기만 한다면 너무 많은 수의 패턴이 패턴 트리에 존재하게 되어 결국은 입력되는 모든 순차가 패턴으로 인식되는 최악의 상황이 일어날 수 있다. 따라서 새로운 패턴의 추가와 동시에 잘 나오지 않는 패턴의 삭제도 필요하다. (1)에서는 패턴 삭제에 관한 자료 구조를 설명하고 (2)에서는 그 알고리즘을 설명한다.

(1) 자료 구조

패턴의 삭제 방법은 버퍼 교체 정책 (Buffer Replacement Policy) [12]에서 응용을 할 수 있다. 버퍼 교체 정책과 패턴 삭제의 다른 점은, 버퍼 교체 정책에서는 사용 중인 Buffer의 양이 일정하게 유지되지만 패턴 삭제는 패턴의 개수가 일정하게 유지될 필요는 없다. 즉, 적은 수의 패턴이 자주 나오는 이상적인 경우에는 패턴의 개수가 적지만, 많은 수의 패턴이 드물게 등장하는 상황에는 많은 수의 패턴이 트리에 존재하는 것이 가능하다. 따라서 새로운 패턴의 입력된 시간을 가중치로 하여 일정시간 이상으로 패턴의 입력이 없었으면 패턴을 삭제하는 방법이 가능하다.

살펴볼 첫 번째 버퍼 교체 정책은 LRU (Least-Recently Used) 이다. 이것은 가장 접근이 오래된 블록을 지우는 방법으로 RSP-DS의 오랫동안 입력이 안된 패턴을 지우는 것과 유사하다. 그러나 LRU방법은 버퍼가 꽉 찼을 때 알고리즘이 실행되지만, RSP-DS는 매 시간마다 일정 시간이 지난 버퍼를 지워야 하는 차이가 있다. 따라서 LRU를 오래된 패턴 제거에 이용하면 매번 모든 패턴을 탐색하는 많은 비용이 소모된다. 다른 버퍼 교체 정책인 FIFO는 가장 오래 전에 들어온 블록을 지우는 것이다. 하지만 이 방법은 입력된 시간만 고려 할 뿐, 자주 접근되는 경우를 고려하지 않았다. 따라서 패턴 삭제에 이용할 경우 자주 입력되지만 오래 전에 인식이 되었을 패턴이 지워지게 된다는 단점이 발생할 수 있다. 또 다른 버퍼 교체 정책으로써 시계 방법 (Clock Algorithm)이 있다. 이 방법 또한 오랫동안 접근이 안 된 오래된 블록 한 개를 지우는 방법이다. 그러나 한 패턴이 아니라 일정 기간이 지난 모든 패턴들을 지워야만 하는 RSP-DS에는 적용하기가 어렵다.

이 중에서 많은 수의 패턴을 손쉽게 삭제 할 수 있는 FIFO를 간단하게 수정하여 패턴 삭제에 이용하겠다. 먼저 입력된 패턴의 시간 별로 FIFO를 생성한다. 그리고 그 패턴이 다음에 다시 입력되면 간단히 FIFO에서 찾아서 다시 FIFO에 입력을 한다. 물론, FIFO안에 있는 패턴들을 한 번에 접근할 수 있는 또 다른 인덱스가 구축이 되어 있어야 한다. 이러한 인덱스로는 패턴 트리를 이용하겠다. 그림 6은 이러한 변형된 패턴 트리와 FIFO 구조를 나타낸다.

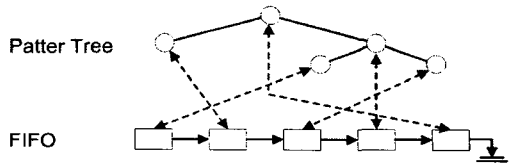


그림 6. 패턴 삭제를 위해 FIFO를 가진 패턴 트리

(2) 알고리즘

다음은 패턴 삭제를 고려해서 작성된 패턴 추가 알고리즘과 패턴 삭제의 알고리즘이다.

그림 7은 이미 트리에 존재하던 패턴이 입력되었을 때의 모습이다. 단순히 입력된 패턴을 트리에서 찾아서 해당되는 노드를 FIFO에 다시 입력을 해주면 된다.

그림 8은 패턴을 삭제할 때를 나타낸다. <a,c,b>가 빈번하지 않다고 해서 <a,c>가 빈번하지 않은 것은 아니다. 또 같은 Prefix를 공유하고 있는 패턴 <a,c,e>가 빈번 할 수도 있다. 따라서 <a,c,b>가 삭제 될 때 <a,c>는 삭제되면 안된다. 그러나 <a,c>가 빈번하지 않으면 <a,c,d>와 <a,c,e>도 빈번하지 않다. 따하서 <a,c>가 삭제 될 때는 그에 딸린 Suffix도 삭제를 해주어야 된다.

삭제 되야 할 짧은 패턴이 다른 패턴의 Suffix로 오는 경우와 Prefix로 오는 경우를 고려할 수 있다. <a,e,c>에서 Suffix로 오는 <c,e>가 삭제 될 경우 트리에서 <c,e>를 찾아서 삭제 한 다음 <e,c>를 참조하고 있던 <a>를 찾아가서 그 포인터를 제거해주면 된다. 또한 Prefix로 오는 <a,e>를 삭제 할 경우에는 어떠한 패턴이 빈발하지 않을 경우 그 패턴을 포함하는 모든 패턴이 빈발하지 않으므로 패턴 트리에서 <a,e>와 그 모든 하위 패턴 들을 삭제하면 된다. 그러나 하위 패턴 중에 다른 패턴을 가리키고 있는 포인터가 있을 경우, 가리켜지는 패턴은 또 다른 패턴의 Suffix로 참조되어질 수 있다. 따라서 포인터에 가리켜지는 패턴은 지우지 않는다.

그림 9는 새로운 순차가 들어 왔을 경우 그림 2에서 갱신된 트리의 모습을 나타낸 것이다. 새로운 패턴 추출에서 발견되지 않은 패턴은 트리에서 곧바로 제외 됐으며, 새롭게 추출된 패턴은 트리에 추가 하였다. <<>>안의 패턴들은 이 전 트리에서 나오지 않고 새롭게 추가된 패턴이다.

4. 성능 평가

본 논문에서 RSP-DS를 구현하기 위하여 사용된 시스템 환경은 다음과 같다. 개발에는 C++ 언어가 사용되었으며, 시스템은 Intel Pentium 4 3.00Ghz, RAM 1GB, HDD 150GB의 하드웨어 환경에 Red hat Linux 9.0 O/S를 가지고 있다. 제시하는 알고리즘의 성능은 크게 4개의 변수에 의해 변화할 수 있다. 다음은 그 변수들과 그에 대한 설명이다.

- min_sup: 자주 등장하는 데이터를 판단하는 기준. min_sup가 너무 크면 자주 등장하는 데이터의 개수가 줄어들어 계산 비용이 줄어든다. 그러

알고리즘 3 (AddPattern)
입력: 패턴 트리 PT, 새로 입력될 순차 s_new, FIFO
출력: 없음
1: **if** s_new ∈ PT **then**
2: FIFO에서 s_new를 찾아서, s_new를 FIFO의 맨 끝으로 이동시킴
3: **else**
4: s_new를 PT에 추가하고. 입력된 시간과 함께 FIFO에 입력
5: **end if**

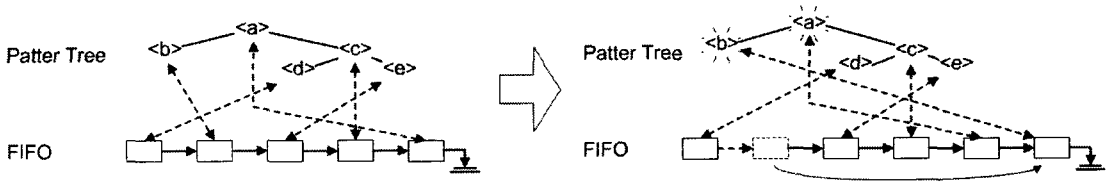


그림 7. AddPattern으로 인하여 수정된 패턴 트리와 FIFO

알고리즘 4 (DeletePattern)
입력: FIFO, pattern tree PT, Time Threshold t
출력: 없음
1: 패턴 집합 pat_set := FIFO에 있는 t가 넘는 입력시간을 가진 패턴들
2: **foreach** pat := pat_set의 한 패턴 **do**
3: **if** PT에 pat의 하위 패턴들이 있음 **then**
4: PT에서 pat와 pat의 하위 패턴들을 지움. 이 때 하위 패턴에 포인터가 있을 경우 포인터는 따라가서 지우지는 않음
5: **else**
6: PT에서 pat를 지움
7: **end if**
8: **end for**

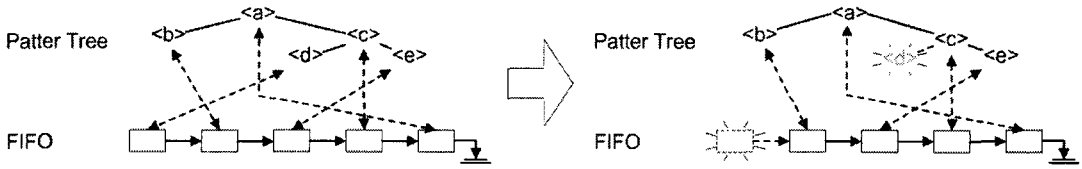


그림 8. DeletePattern으로 인하여 수정된 패턴 트리와 FIFO

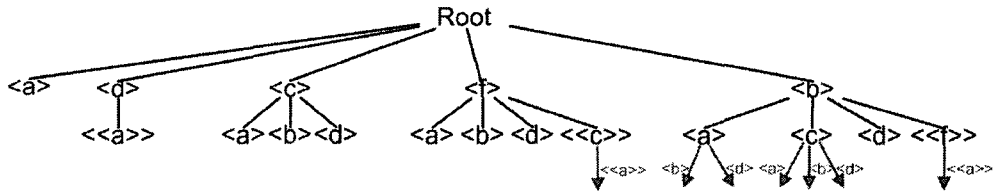


그림 9. RSP-DS의 결과로 나온 패턴 트리

나 너무 적은 수의 패턴을 발견하게 되어 원하는 결과를 얻지 못할 수 있다.

- dif_items: 스트림에 입력되는 데이터의 종류. 입력되는 데이터의 종류가 많으면 그 만큼 다양한 데이터가 입력된다는 것을 의미한다. 이 변수는 데이터 스트림 환경에 따라 달라지며, 사용자가 조절할 수 없다.
- new_seqs: 해시 테이블에 입력될 새로운 순차의 개수. 동일한 수의 순차로 패턴을 분석하기 위해 new_seqs개의 순차가 입력될 때마다 해시 테이블에서 제일 오래된 new_seqs개의 순차를 지운다. new_seqs의 크기가 크면 많은 양의 데이터가 입력될 때까지 패턴 분석이 지연되어, 패턴 분석의 주기가 길어지게 된다. 패턴 분석의 주기를 짧게 하기 위하여 new_seqs의 개수를 줄 일수도 있지만, 너무 자주 발생하는 패턴분석은 계산 비용을 증가 시킬 수 있다.
- old_seqs: 해시 테이블에 있는 과거 순차들의 개수. new_seqs와 비교하여 패턴을 분석해 낼 과거 순차들의 개수이다. old_seqs가 new_seqs보다 많이 크면 new_seqs와는 상관없이 어떠한 패턴이 자주 등장하는 데이터를 쉽게 구별 할 수 있다.

예를 들어 new_seqs가 3, old_seqs가 7, min_sup이 6이라고 하자. 그러면 new_seqs+old_seqs에서 5개 이상의 순차에 등장하는 데이터가 자주 등장하는 데이터로 결정된다. 이 때 n<3인, n개의 데이터 D가 해시 테이블에 존재 한다고 하면, 새로운 순차 3개 모두에 D가 존재한다 하더라도 n+3<6이 되어 자주 등장하는 데이터로 선택되지 못한다. 비슷하게 이미 해시 테이블에 n>=5인 데이터 D'가 존재 할 경우, 새로운 순차에 관계없이 자주 등장하는 데이터로 선택되게 된다.

data_rate: 시간당 순차의 입력 비율이다. 기준을 초단위로 가정하고, data_rate를 10으로 하면 1초에 10개의 순차가 들어온다는 뜻이다. 이 말은 time_stampe 단위가 1/10초라는 것과 동일하다. data_rate가 지나치게 높으면 패턴 분석하는 속도가 입력속도를 따라가지 못한다. 따라서 data_rate가 높을 때는 패턴 분석의 주기를 늘려 주어 패턴 분석이 너무 자주 일어나지 않도록 해야 한다.

그림 10은 시간 윈도우로 나누어진 한 데이터 스트림의 패턴 분석에 적용된 MILE과 RSP-DS의 트리 구축 시간을 비교한 것이다. 가로축은 초당 data_rate를 나타내고 세로축은 한 순차가 입력될 때부터 패턴분석이 이루어 질 때까지 시간의 평균을 나타낸 것이다. dif_item은 10, min_sup은 10, new_seqs는 10, old_seqs는 10으로 하여 실험을 진행하였다. 표 1은 data_rate에 따른 패턴 트리의 갱신 주기를 나타낸 것이다. MILE의 경우 10개의 순차가 입력될 때마다 트리를 새로 구축해야 한다. 또한 data_rate가 높아지면 갱신 주기가 짧아져서 패턴 계산 속도가 데이터의 입력 속도를 따라가지 못하게 된다. 따라서 해시 테이블에 입력을 대기하게 되는 순차의 개수가 많아지므로 한 순차 당 처리 시간은 급격히 높아진다. 그러나 RSP-DS의 경우에는 새로 들어오는 순차를 해시 테이블에 입력하고 이전 순차와 비교하여 기존의 패턴 트리를 갱신하기 때문에 빠른 처리 시간을 보였다.

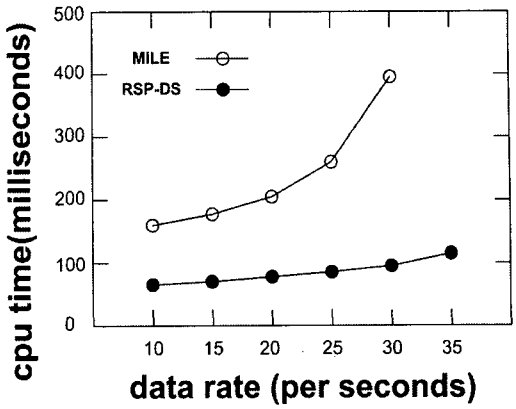


그림 10. RSP-DS와 MILE의 비교

그림 11은 다양한 min_sup값에 따른 RSP-DS의 성능 결과를 나타낸 것이다. 가로축과 세로축은 그림 10과 동일한 단위이다. dif_item은 10, new_seqs는 10, old_seqs는 10으로 하였다. min_sup값은 일반적으로 생각하기 위하여 new_seqs의 비율로 정하였다. min_sup가 높아질수록 적은 수의 데이터가 자주 등장하는 데이터로 선택이 되므로 계산 비용이 줄어들어 성능이 높아진다. 그러나 너무 적은 수의 min_sup을 선택하면 그만큼 분석되는 패턴의 개수도 줄어들게 되어 의도한 결과가 나오지 않을 수도 있다.

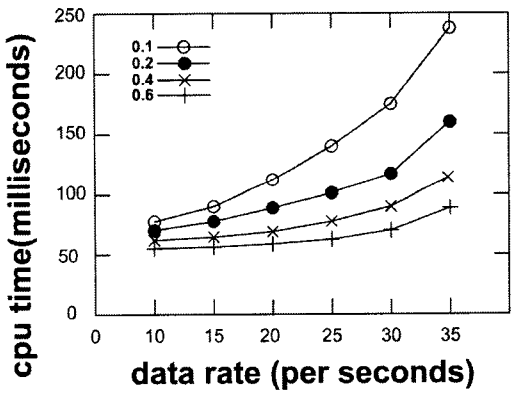


그림 11. 다양한 min_sup에 따른 성능 변화

표 1. 다양한 data_rate에 따른 패턴 트리의 갱신주기

Data_rate	10	15	20	25	30	35
갱신주기 (초)	1	2/3	1/2	2/5	1/3	2/7

그림 12는 다양한 new_seqs의 변화에 따른 성능의 결과이다. 가로축과 세로축은 그림 11과 동일하고, dif_item은 10, min_sup은 10, old_seqs는 10으로 실험을 하였다. 표 2는 new_seqs의 변화에 따른 갱신 주기를 나타낸 것이다. new_seqs의 개수가 많으면 그만큼 갱신 주기가 늘어난다. 따라서 data_rate가 높아져도 성능의 커다란 변화를 일으키지 않는다. 그러나 그만큼 느리게 패턴이 분석된다는 것을 의미한다. new_seqs를 줄이면 패턴 분석하는 순차의 개수가 줄어들어 패턴 분석 시간이 짧아 진다. 하지만 data_rate가 너무 높으면 패턴 분석의 주기가 너무 짧아진다. 따라서 데이터의 입력 속도를 따라가지 못해 오히려 패턴 분석 시간이 급격히 늘어나는 것을 볼 수 있다.

실험의 결과들을 통하여 RSP-DS 방법을 이용하면 상황에 따라 변하는 패턴을 빠르게 알아 낼 수 있다는 사실을 알아내었다. 특히 과학적 실험의 성능 측정, 네트워크 패킷 모니터링, 교통량 측정과 같이 지속적으로 모니터링을 하면서 패턴의 이상치를 발견해야만 하는 응용에서 효과적으로 쓰일 수 있다.

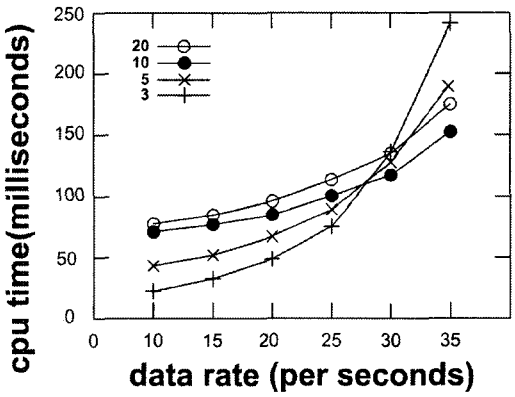


그림 12. 다양한 new_seqs에 따른 성능 변화

5. 결론 및 향후 연구

최근 데이터 스트림이라고 하는 새로운 형태의 데이터가 주목을 받고 있다. 데이터 스트림의 패턴은 끊임없이 변할 수 있다. 그러나 기존의 패턴 분석 알고리즘들은 단 한번의 패턴 분석만을 고려하였다. 이러한 방법으로 변하는 패턴을 인식하려면 계속해서 알고리즘을 처음부터 수행해야하는 단점을 가지고 있다. 제안하는 RSP-DS 방법은 발견된 패턴으로 패턴 트리를 구축하고 조금씩 변하는 패턴들을 이용하여 패턴 트리를 가지별로 갱신하여 실시간으로 패턴을 분석해내는 방법을 제안하였다. RSP-DS은 데이터 스트림 패턴 분석 알고리즘인 MILE을 기본으로 하여 작성되었다. 그러나 실험을 통하여 계속해서 변하는 패턴을 인식하는 과정은 MILE보다 높은 성능을 보인다는 것은 증명하였다. 이것는 MILE은 새로운 순차가 입력될 때마다 매번 트리를 다시 생성하지만, RSP-DS는 이 전에 생성된 트리를 수정하여 현재 패턴을 유지하기 때문이다. 그리고 빈발 항목을 판단하는 min_sup와 패턴 분석하는데 이용하는 순차의 개수에 따라 알고리즘의 성능이 조절될 수 있다는 것을 발견하였다. 따라서 이 두 변수의 적절한 선택으로 성능의 최적화를 이루어 낼 수 있다.

RSP-DS는 패턴이 일정한 시간 주기로 등장한다는 것을 가정하고 있기 때문에 향후에 패턴의 주기가 변하는 데이터 스트림에서의 패턴 분석이 연구가 되어져야 한다. 또한 패턴의 변화가 거의 발생하지 않은 환경에서 RSP-DS와 같이 계속하여 패턴을 분석하는 방법은 불필요한 연산을 초래한다. 따라서 데이터 스트림의 패턴이 변하고 있는지 빠르게 알아낼 수 있는 알고리즘의 연구도 이루어져야 할 것이다.

표 2. data_rate와 new_seqs의 변화에 따른 패턴 트리의 갱신 주기(초)

data_rate	10	15	20	25	30	35
new_seqs=3	3/10	1/5	3/20	3/25	1/10	3/35
new_seqs=5	1/2	1/3	1/4	1/5	1/6	1/7
new_seqs=10	1	2/3	1/2	2/5	1/3	2/7
new_seqs=20	2	4/3	1	4/5	2/3	4/7

참 고 문 헌

[1] 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 Trans. Knowl. Data Eng.*, Vol. 16, No. 11, pp. 1424-1440, 2004.

[2] G. Chen, X. Wu, and X. Zhu, "Sequential Pattern Mining in Multiple Streams," *ICDM*, pp. 585-588, 2005.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and Issues in Data Stream Systems," *PODS*, pp. 1-16, 2002.

[4] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," *EDBT*, pp. 3-17, 1996.

[5] H.F. Li, S.Y. Lee, and M.K Shan, "On Mining Webclick Streams for Path Traversal Patterns," *WWW*, pp. 404-405, 2004.

[6] 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.

[7] G. Das, K.I. Lin, H. Mannila, G. Renganathan, and P. Smyth, "Rule Discovery from Time Series," *KDD*, pp. 16-22, 1998.

[8] T. Oates and P.R. Cohen, "Searching for Structure in Multiple Streams of Data," *ICML*, pp. 346-354, 1996.

[9] H.F. Li, S.Y. Lee, and M.K. Shan, "On Mining WebClick Streams for Path Traversal Patterns," *WWW*, pp. 404-405, 2004.

[10] M.M. Gaber, S. Krishnaswamy, and A. Zaslavsky, "Ubiquitous Data Stream Mining," *PAKDD*, 2004.

[11] 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.

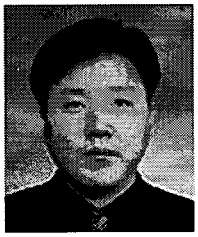
[12] H. Garcia-Moling, J.D. Ullman, and J. Widom, *Database System Implementation*, Prentice Hall International, Inc., New Jersey, 2000.

신 재 진

2005년 인하대학교 컴퓨터공학부 (공학사)
2005년~현재 인하대학교 컴퓨터정보공학부(석사과정)
관심분야 : DSMS, Data Stream, 공간데이터베이스

김 호 석

2001년 인하대학교 컴퓨터공학부 (공학사)
2003년 인하대학교 컴퓨터공학부 (공학석사)
2003년~현재 인하대학교 대학원 컴퓨터정보공학과(박사과정)
관심분야 : 공간데이터베이스, LBS, 유비쿼터스 컴퓨팅, RFID 미들웨어



김 경 배

1992년 인하대하교 전자계산공학과 (공학사)
1994년 인하대학교 대학원 전자계산공학과 (공학석사)
2000년 인하대학교 대학원 전자계산공학과 (공학박사)
2000년~2004년 한국전자통신연구원 선임연구원
2004년~현재 서원대학교 컴퓨터교육과 조교수
관심분야 : 이동 실시간 데이터베이스, 스토리지 시스템, GIS, VOD 등

배 해 영

1974년 인하대학교 응용물리학과 (공학사)
1978년 연세대학교 대학원 전자계산학과(공학석사)
1989년 숭실대학교 대학원 전자계산학과(공학박사)
1985년 Univ. of Houston 객원교수
1992년~1994년 인하대학교 전자계산소 소장
1982년~현재 인하대학교 컴퓨터공학부 교수
1999년~현재 지능형GIS연구센터 센터장
2000년~현재 중국 중경우전대학교 대학원 명예교수
2004년~2006년 인하대학교 정보통신대학원 원장
2006년~현재 인하대학교 대학원 원장
관심분야 : 분산 데이터베이스, 공간 데이터베이스, 지리정보 시스템, 멀티미디어 데이터베이스 등