DOI: 10.3745/KIPSTD.2008.15-D.6.741

한정된 메모리 공간에서 데이터 스트림의 빈발항목 최적화 방법

김 민 정[†] · 신 세 정[††] · 이 원 석[†††]

요 약

지속적으로 확장되는 데이터 스트림에 대한 데이터 마이닝 수행과정에서는 메모리 사용량을 가용한 범위 내로 제한하는 것이 중요한 요소이다. 본 논문에서는 데이터 스트림 환경에서 한정된 메모리 공간을 이용하여 빈발 항목집합을 탐색하는데 효과적인 프라임 패턴 트리(*Prime pattern tree: PPT)*구조를 제안한다. 프라임 패턴 트리는 기존의 전위 트리 구조와 비교하여 항목집합들을 하나의 노드로 관리함으로써 트리의 크기를 크게 줄일 수 있는 장점이 있다. 또한, 전지 임계값 S_δ에 따라 노드를 병합하거나 분리하여 동적으로 트리의 크기와 결과 집합의 정확도를 마이닝 수행 중에 조절 할 수 있다. S_δ값이 크면 한 노드에서 관리되는 항목집합의 수가 증가하게 되고, 출현 빈도수를 추정해야 하기 때문에, S_δ값이 작을수록 결과집합의 정확도가 높다. 이처럼 *PPT*에는 트리의 크기와 정확도의 trade-off 가 존재한다. *PPT*의 이러한 특성에 기반하여, 데이터 스트림에서 갑자기 데이터 집합에 변화가 생겨 빈발항목이 될 가능성이 높은 항목들이 많이 출현하는 경우에도 마이닝을 지속적으로 수행할 수 있도록 지원한다. 본 논문에서는 프라임 패턴 트리를 이전 연구에서 제안한 데이터 스트림에서 최근 빈발 항목 탐색 방법인 *estDec* 방법에 적용하여 한정된 작은 양의 메모리 공간을 이용하여 온라인 데이터 스트림에서 빈발항목을 탐색하는 방법을 제시한다. 또한, 가용 메모리 범위에서 최적의 메모리를 사용하여 최적의 마이닝 결과를 얻을 수 있도록 하는 메모리 사용량에 대한 적응적 방법을 제시한다. 끝으로, 여러 실험을 통한 효율성 검증을 통해 제안된 방법의 여러 특성을 확인한다.

키워드 : 데이터마이닝, 데이터스트림, 빈발 항목집합 탐색

Finding Frequent Itemsets Over Data Streams in Confined Memory Space

Min Jung Kim[†] · Se Jung Shin[††] · Won Suk Lee[†††]

ABSTRACT

Due to the characteristics of a data stream, it is very important to confine the memory usage of a data mining process regardless of the amount of information generated in the data stream. For this purpose, this paper proposes the *Prime pattern tree(PPT)* for finding frequent itemsets over data streams with using the confined memory space. Unlike a prefix tree, a node of a *PPT* can maintain the information necessary to estimate the current supports of several itemsets together. The length of items in a prime pattern can be reduced the total number of nodes and controlled by *split_delta* S_δ. The size and the accuracy of the *PPT* is determined by S_δ. The accuracy is better as the value of S_δ is smaller since the value of S_δ is large, many itemsets are estimated their frequencies. So it is important to consider trade-off between the size of a *PPT* and the accuracy of the mining result. Based on this characteristic, the size and the accuracy of the *PPT* can be flexibly controlled by merging or splitting nodes in a mining process. For finding all frequent itemsets over the data stream, this paper proposes a *PPT* to replace the role of a prefix tree in the *estDec* method which was proposed as a previous work. It is efficient to optimize the memory usage for finding frequent itemsets over a data stream in confined memory space. Finally, the performance of the proposed method is analyzed by a series of experiments to identify its various characteristics.

Keywords : Data Mining, Data Stream, Frequent Itemsets

1. 서 론

정보 기술의 발전으로 인해 컴퓨팅 기술 및 관련 기술의 발달로 발생되는 정보의 양과 속도가 증가 하고 있다. 이러한 정보들 중 가치 있는 지식을 얻기 위하여 데이터 마이닝이 다양한 분야에서 이용되고 있다. 기존의 데이터 마이닝 방법들은 지식 발견의 대상이 되는 데이터 집합이 마이닝 작업 시작이전에 명확히 정의되어 있다고 가정한다. 그러나 최근 들어 실시간 데이터 마이닝에 대한 필요성이 대두되면서 데이터 스트림(data stream)에 대한 마이닝 방법들[1, 2, 3, 4, 5, 6, 7, 8]이 제안되고 있다. 또한, *Moment* [17]와 *CFI-*

※이 논문은 2008년도 정부(교육과학기술부)의 재원으로 한국과학재단의 국가지정연구실사업으로 수행된 연구임(No.R0A-2006-000-10225-0).
† 정 회 원 : 삼성전자 무선사업부 GSM 단말 MMI 개발 연구원
†† 정 회 원 : 연세대학교 컴퓨터과학과 석·박사 통합과정
††† 종신회원 : 연세대학교 컴퓨터과학과 교수
논문접수 : 2008년 7월 7일
수 정 일 : 1차 2008년 8월 20일
심사완료 : 2008년 9월 5일

stream [18] 방법은 각각 *closed enumeration tree* (*CET*)와 *DIrect Update* (*DIU*) tree를 구성하여 스트림 슬라이딩 윈도우에서 폐쇄 빈발항목 집합을 탐색하는데 초점을 맞추고 있다.

데이터 스트림은 지속적으로 발생되는 데이터로 구성되는 무한 집합으로 정의된다. 사전 정의된 데이터 집합이 아닌 지속적으로 발생되는 데이터 스트림의 특성상 발생된 모든 데이터 객체를 별도로 저장하는 것은 한정된 저장 공간에서는 불가능하다. 이런 특성을 고려하여 데이터 스트림의 정보에 대한 지식을 추출하기 위해서는 다음과 같은 몇 가지 조건이 필요하다[9]. 첫째, 데이터 스트림의 각 트랜잭션 정보를 단 한번만 읽고 마이닝 결과를 생성해야 한다. 둘째, 데이터 스트림을 분석하기 위해 아무리 새로운 데이터가 계속적으로 무한히 생성된다 하더라도 물리적으로 한정된 메모리 공간으로 처리해야 한다. 셋째, 새롭게 생성된 데이터 객체는 가능한 빠르게 처리되어야 한다. 마지막으로 스트림에 대한 최신의 결과는 필요 시 즉시 제공되어야 한다. 이러한 요구조건을 만족시키기 위해 일반적으로 데이터 스트림에 대한 마이닝 방법들은 마이닝 결과에 다소의 오차를 포함한다.

데이터 스트림이 발생하는 환경은 데이터 웨어 하우스(data warehouse)등에서와 같이 대량의 새로운 트랜잭션들을 한꺼번에 추가되는 경우와 보안 및 감시 시스템에서와 같이 지속적으로 증가되는 각 트랜잭션들이 개별적으로 증가되는 경우로 구분할 수 있다. 전자를 오프라인 데이터 스트림(off-line data stream)이라 하고 후자는 온라인 데이터 스트림(on-line data stream)이라 한다. 오프라인 데이터 스트림에 대한 마이닝 방법은 새로 생성되는 트랜잭션이 많을 때 효율적으로 수행될 수 있으며 정해진 시간 간격당 업데이트가 일어나고 마이닝 결과를 특정 시점에서 얻고자 하며 이 작업이 빈번히 일어나지 않을 때 사용된다[2]. 반면, 온라인 데이터 스트림에 대한 마이닝 알고리즘은 최신 정보를 포함하는 마이닝 정보를 원하는 시점에서 즉시 구할 수 있어야 한다. 따라서 온라인 데이터 스트림을 마이닝 할 때 수행 시간과 마이닝 결과의 정확도 사이에 trade-off가 존재한다.

빈발 항목집합 마이닝을 위한 데이터 스트림은 지속적으로 발생되는 트랜잭션의 무한집합으로 다음과 같이 정의된다:

ⅰ) $I = \{i_1, i_2, \cdots, i_n\}$는 현재까지의 항목(item)의 집합이며 항목은 응용 도메인에서 발생한 단위 정보를 의미한다.

ⅱ) 2^I가 항목집합 I의 멱집합을 나타낼 때, $e \in (2^I - \{\varnothing\})$을 만족하는 e를 항목집합(itemset)이라 하고, 항목집합의 길이 $|e|$는 항목집합 e를 구성하는 항목의 수를 의미하며 임의의 항목집합 e는 해당 항목집합의 길이에 따라 $|e|$-항목집합이라 정의한다. 일반적으로 3-항목집합 $\{a, b, c\}$는 간단히 abc로 나타낸다.

ⅲ) 트랜잭션은 공집합이 아닌 I의 부분집합이며 각 트랜잭션은 트랜잭션 식별자 *TID*를 갖는다. k번째 순서로 데이터 집합에 추가되는 트랜잭션을 T_k라 나타내며 T_k의 *TID*는 k이다.

ⅳ) 새로운 트랜잭션 T_k가 추가되었을 때 현재의 데이터 집합 D_k는 현재까지 발생하여 추가된 모든 트랜잭션들 즉, $D_k = \langle T_1, T_2, \cdots, T_k \rangle$로 구성된다. 따라서 $|D|_k$는 현재 데이터 집합 D_k에 포함된 트랜잭션의 총 수를 의미한다. □

T_k를 현재 트랜잭션이라 할 때, 임의의 항목집합 e에 대한 현재 출현 빈도수를 $C_k(e)$라 정의하며 이는 현재까지의 k 트랜잭션에서 e가 포함된 트랜잭션의 수를 나타낸다. 이와 마찬가지로, 항목집합 e의 현재 지지도 $S_k(e)$는 현재까지의 트랜잭션의 총 수 $|D|_k$ 대비 항목집합 e의 출현 빈도수 $C_k(e)$의 비율로 정의한다. 항목집합 e의 현재 지지도 $S_k(e)$가 사전 정의된 최소 지지도 S_{min} 이상일 때, 항목집합 e를 현재 데이터 스트림 D_k에서의 빈발 항목집합이라 정의한다.

이러한 온라인 데이터 스트림에서 최근에 생성된 주요한 지식들 중 사전 정의된 최소 지지도 보다 출현 빈도수가 큰 항목집합들을 탐색하기 위하여 *estDec*[10]방법이 제안되었다. 이 방법은 데이터 스트림에서 발생된 모든 트랜잭션의 정보를 메모리상에 저장하지 않고 각 트랜잭션들이 출현 된 시간적 순서에 따라 가중치를 달리하여 최근에 최소 지지도 이상의 출현 빈도수를 가진 항목집합들에 대한 정보만을 메모리상의 전위 트리(prefix tree)[11, 12, 13, 14]로 관리한다. 생성된 지 오래된 트랜잭션은 새로운 트랜잭션들이 계속적으로 생성 되면서 전체적인 마이닝 결과에 크게 영향을 미치지 않기 때문에 시간적 가중치를 지속적으로 감쇠시켜 적어지도록 한다. 이렇게 시간적 순서로 매겨지는 가중치의 비율은 최신의 빈발 항목 집합의 범위를 결정한다. *estDec* 방법은 지연 추가(*delayed-insertion*)와 전지(*pruning*) 작업을 통해 메모리 상에 관리되는 전위 트리의 크기를 줄일 수 있다. 그러나 전위 트리에서는 출현 빈도수 관리 대상이 되는 개별 항목집합을 각각 별도의 노드로 관리하기 때문에 관리 대상 항목집합의 수가 증가될수록 트리의 크기가 커지며, 이는 마이닝 수행과정에서 메모리 사용량을 증가시킨다.

본 논문에서는 이러한 전위 트리 구조의 단점을 보완하여 동적으로 메모리상에서 관리되는 트리의 크기를 적응적으로 조절 할 수 있는 프라임 패턴 트리(*Prime Pattern Tree : PPT*)구조를 제안한다. 전위 트리의 한 노드가 하나의 항목집합을 표현하는데 반해, *PPT*는 전위 트리가 표현하는 모니터링 정보를 축약함으로써 한 노드에서 한 개 이상의 항목집합을 표현할 수 있으며 이는 전위 트리와 비교하여 메모리 사용량을 크게 줄일 수 있다. 프라임 패턴 트리의 가장 큰 특징은 하나의 노드로 함께 관리 될 수 있는 항목들을 결정하는 전지 임계값(*split_delta* : S_δ)을 마이닝 과정 중에 변화 시킴으로써 프라임 패턴 트리의 크기를 동적으로 조절 할 수 있다는 것이다. 이는 *estDec*방법의 단점을 보완하여 어느 시점에서 메모리 사용량이 급격히 늘어 날 때에도 마이닝 수행이 이루어 질 수 있도록 한다. 또한 프라임 패턴 트리를 *estDec* 방법에 적용하여 한정된 작은 양의 메모리 공간을 이용하여 온라인 데이터 스트림에서 빈발 항목집합을 탐색하는 *estDec+* 방법을 제시한다. *estDec+* 방법

에서는 감쇠율 기본값을 간단히 1로 설정한다. 보다 큰 감쇠율을 적용하는 경우의 감쇠율 적용 방법은 *estDec* 방법과 동일하다.

그러나 본 논문에서는 [10]에서 소개한 시간적 가중치를 결정하는 감쇠 기반(*decay-base*), b를 1로 하여 데이터 스트림을 이루는 각각의 트랜잭션들이 생성된 시간적 순서에 상관없이 동일한 가중치를 가질 수 있도록 한다.

본 논문의 구성은 다음과 같다. 2장에서는 데이터 스트림에서 빈발 항목집합 탐색과 관련된 연구를 기술하고 3장에서는 *PPT* 구조를 정의하고, *PPT*를 구성하는 방법에 대해 소개한다. 4장은 *PPT*를 이용하여 빈발 항목집합을 탐색하는 방법을 알아본다. 5장에서는 다양한 실험을 통해서 제안된 방법을 검증하며, 6장에서는 본 논문의 결론을 맺는다.

2. 관련 연구

일반적으로 빈발 항목집합 탐색은 한정적인 데이터 집합에서 특정 지지도 임계값(support threshold)보다 큰 지지도를 갖는 모든 항목집합을 파악하는 작업이다. 빈발 항목집합 탐색을 위한 가장 대표적인 것은 *Apriori*[15] 알고리즘으로 길이 n인 빈발항목집합을 탐색하기 위해 최대 n+1번의 트랜잭션 정보 검색 과정을 거친다. *Apriori*에서는 각 패스에서 빈발 항목집합들의 후보 항목집합을 구성하고 난 후에 각 후보 항목집합의 발생 빈도수를 계산하고, 사용자가 정의한 최소 지지도를 기초로 하여 빈발항목집합들을 결정한다. *Apriori* 알고리즘은 각각의 패스가 주어진 항목 개수를 가지는 모든 후보집합(Candidate)들을 생성하는 *Apriori-gen*에 대한 호출과 이 후보들에 대한 지지도를 계산하는 counting 단계로 구성되어 있으며 지지도 계산 단계에서는 전체 데이터베이스를 스캔한다. *DIC*[11]와 같은 알고리즘들이 트랜잭션 정보 검색 회수를 줄이기 위해 제안되었다. 빈발 항목집합 탐색을 위해 제안된 많은 데이터 구조 중에서 점진적인 마이닝을 위해 메모리 사용량을 줄이기 위한 한 방법으로 제시된 항목집합 트리(itemset tree)[16]는 전위 트리와 달리 기본적으로 트랜잭션 전체를 하나의 노드로 구성한다. 항목집합 트리를 생성하기 위해 다음의 두 단계를 거친다. 첫 번째 단계는 새로 들어온 항목집합에 대한 노드 생성단계이고, 두 번째 단계는 새로 들어온 항목집합에 의해 갱신될 수 있는 노드들의 출현 빈도수를 갱신한다. 새로운 항목집합에 대해 노드를 생성하기 위해 트리를 탐색 할 때 새로 들어온 트랜잭션으로 생성 된 항목집합들 중 항목집합 트리의 노드를 구성하고 있는 노드들과 비교하여 공통된 항목집합이 있으면 공통된 항목집합을 상위 노드로 하고, 나머지 항목집합을 하위 노드를 생성한다. 항목집합 트리의 노드는 정확한 출현 빈도수를 관리하기 때문에 두 번째 단계에서 전체 트리를 탐색하여 출현 빈도수를 갱신한다. 항목집합 트리는 대량의 데이터를 처리 할 때 메모리 사용량을 효과적으로 줄일 수 있으나 각 노드의 출현 빈도수를 갱신하기 위해 전체 트리를 재탐색해야 하므로 처리 시간이 오래 걸리며 생성된 모든 트랜잭션에 대한 정보를 메모리상에 저장하여야 하므로 동적으로 항목집합 트리의 크기를 결정 할 수 없다. 그러므로 실시간으로 마이닝 결과를 원하는 온라인 데이터 스트림을 대상으로 하는 빈발 항목집합 탐색 방법으로는 적합하지 않다.

데이터 스트림에 대한 빈발 항목집합 탐색 방법[1,2]은 데이터 스트림에 포함되는 이전의 모든 트랜잭션의 정보를 유지하는 것이 불가능하므로 마이닝 결과로 구해진 빈발 항목집합이나 출현빈도수에 다소의 오차를 포함할 수 있다. *Count Sketch* 알고리즘[1]은 데이터 스트림에서 빈발 단위항목을 찾는 방법으로 현재까지 생성된 트랜잭션에서 단위항목의 출현 빈도수를 추정하여 임계값 이상을 만족하는 단위 항목들의 집합을 생성한다. 반면에 *Lossy Counting* 알고리즘[2]은 데이터 스트림을 마이닝 하기 위한 방법으로 최소지지도와 최대허용오차(ε) 조건이 주어졌을 때 데이터 스트림에서 발생한 빈발 항목집합을 찾는다. 스트림을 구성하는 트랜잭션들에서 발생하는 빈발 가능한 항목들의 출현 빈도수와 이들 각각의 오차를 메모리에서 관리하며 새로 발생한 트랜잭션들은 메인 메모리에 유지되는 고정된 크기의 버퍼에 채워지며 동시에 처리된다. *Lossy Counting* 알고리즘[2]에서는 빈발 항목집합 탐색 과정에서 메모리 사용량을 일정 범위로 한정하기 위해서 출현 빈도수 관리 대상 항목집합들을 보조 저장 장치에 관리하며 메모리 상에서는 일괄 연산 수행을 위한 버퍼만 유지한다. 이 알고리즘에서는 버퍼의 크기와 한번에 일괄 처리될 수 있는 트랜잭션의 수가 비례하므로, 버퍼의 크기가 증가함에 따라 높은 효율을 보이지만 결과적으로 빈발 항목집합 탐색을 위해 필요한 메모리 사용 공간은 증가한다. 또한, 출현 빈도수를 갱신하거나 마이닝 결과를 얻기 위해서는 보조 저장 장치에서 관리되는 모든 항목집합들을 탐색해야 하며, 이는 마이닝 수행 시간을 증가시킬 수 있다. 따라서, 임의 시점에서 수시로 마이닝 결과를 구하고자 하는 온라인 데이터 스트림 환경에서의 마이닝에는 비효율적이다.

온라인 데이터 스트림을 대상으로 빈발 항목집합을 탐색하면서 출현 빈도수 관리 대상 항목집합을 메모리 상에서 관리될 수 있을 정도의 적은 수로 줄이기 위한 방법으로 이전 연구에서 *estDec* 방법[10]이 제안되었다. *estDec* 방법에서는, 데이터 스트림에서 발생한 항목집합들 중에서 사전 정의된 지연 추가 및 전지 임계값 S_{sig} ($S_{sig} \leq S_{min}$) 이상의 지지도를 갖는 항목집합들을 가까운 미래에 빈발 항목집합이 될 수 있는 주요 항목집합으로 간주하여 이 항목집합들만을 메모리상에서 관리한다. *estDec* 방법에서 데이터 스트림 D_k에 출현한 새로운 항목집합과 해당 항목집합의 출현 빈도수가 메모리상의 전위트리에서 관리되는 경우는 다음과 같은 두 가지 경우이다. 첫째, 새로운 트랜잭션 T_k에서 최초로 발생된 길이 1인 항목집합과 해당 항목집합의 출현 빈도수는 출현 빈도수 추정 과정을 거치지 않고 전위트리 P_k에 추가되어 관리된다. 둘째, T_k에 전위트리에서 관리되지 않는 길이 n ($n \geq 2$)인 새로운 항목집합이 발생되었을 때에는 해당 항목집합이 가까운 미래에 빈발 항목집합이 될 수 있

을 정도로 큰 지지도를 가질 때 전위트리 P_k에 추가된다. 즉, P_k에서 관리되고 있지 않은 새로운 n-항목집합 e는 자신의 모든 $(n-1)$-부분항목집합들이 P_k에서 관리될 때, e의 지지도를 이들 $(n-1)$-부분항목집합의 출현 빈도수로부터 추정하며, 추정값이 사전 정의된 지연추가 임계값 S_{ins} 이상일 때 P_k에 추가된다. (지연 추가과정).

한편, 이미 P_k에서 관리되고 있는 항목집합들에 대해 현재 항목집합의 지지도가 전지 임계값 S_{prn} 미만으로 감소할 때 해당 항목집합을 앞으로 빈발항목집합이 될 가능성이 상대적으로 낮은 비주요 항목집합으로 간주하여 빈발 항목집합의 안티모노톤(*anti-monotone*) 성질에 의해 해당 항목집합을 표현하는 노드와 그 노드의 모든 자손 노드들을 P_k로부터 제거한다(전지 과정).

전위 트리를 이용한 빈발 항목집합 탐색 방법에서는 탐색 과정 동안 전위트리가 메모리 안에서 관리되어야 하므로 전위 트리의 크기는 한정된 메모리 공간보다 작아야 한다. 그러나, 전위 트리의 크기는 데이터 스트림에 나타나는 S_{sig} 이상의 지지도를 갖는 주요 항목집합의 수에 의존하므로 주요 항목집합을 관리하는 전위 트리의 크기가 한정된 메모리의 크기보다 큰 경우 이후에 발생하는 새로운 주요 항목집합을 모니터링 할 수 없다는 단점을 갖는다. 이러한 단점으로 인해, 전위 트리의 크기가 한정된 메모리의 크기보다 큰 경우, *estDec* 방법을 이용한 빈발 항목집합 탐색 결과의 정확도는 감소한다.

3. 프라임 패턴 트리 구조

기본적으로 프라임 패턴 트리 (Prime pattern tree: *PPT*)는 트리를 구성하거나 탐색 할 때 전위 트리 구조를 기반으로 한다. 현재 데이터 집합 D_k가 주어질 때, *estDec*방법에서 메모리상에 관리되는 전위 트리 P_k는 다음과 같은 특징을 갖는다.

1. P_k는"*null*"값을 가지는 하나의 root노드를 가진다.
2. 어떤 항목집합 e=$i_1 \cdots i_k$에 대하여 $i_1, \cdots, i_k$는 사전적으로 정렬되어 있고 P_k의 경로 $path(e) = root \rightarrow i_1 \rightarrow \cdots \rightarrow i_k$ ($i_k \subset I, k \geq 1$)로 표현된다. 항목 e의 출현 빈도수는 $path(e)$의 마지막 노드의 출현 빈도수로 나타낸다.
3. 각각의 노드는 다음과 같이 3개의 필드로 구성된다: 단위 항목 i_k, 경로 $path(e) = root \rightarrow i_1 \rightarrow \cdots \rightarrow i_k$ ($i_k \subset I, k \geq 1$)로 표현되는 항목집합 $i_1 \cdots i_k$의 출현 빈도수, 각 노드의 자식 노드를 이어주는 링크.

이런 특징을 가지는 전위 트리의 크기는 생성 된 항목집합 중 가장 길이가 긴 항목집합에 의하여 결정된다. 전위 트리는 하나의 경로가 하나의 항목집합을 나타내므로 가장 길이가 긴 항목집합의 길이가 n일 때 최대 n의 깊이를 가지게 된다. 따라서 이런 전위 트리는 서로 다른 항목집합의 수가 많을 때 트리의 크기가 매우 커지는 단점을 가진다. 이런 단점을 보완하기 위해 본 논문에서는 서로 다른 항목집합의 개수나 항목집합의 길이에 영향을 많이 받지 않는 전위 트리를 기반으로 한 프라임 패턴 트리를 제안한다.

3.1 프라임 패턴 트리의 정의 및 구성

데이터 스트림에서 마이닝 대상이 되는 데이터 집합은 점진적으로 증가하므로 빈발 항목집합을 찾기 위해 모든 데이터 집합에 대한 정확한 정보를 메모리에 저장하기 어렵다. 이러한 이유로 데이터 스트림의 다음과 같은 특징을 이용하여 저장 공간을 최소화 할 수 있다. 첫째, 데이터 스트림에서 생성된 항목집합들 중 유사한 출현 빈도수를 가진 항목집합들이 있다. 유사한 지지도를 가지는 항목집합들의 지지도 차가 사전 정의된 분리 임계값(*split_delta* : S_δ)이내 일 때 이러한 항목집합들을 전위 트리 기반의 트리에서 하나의 노드로 관리하면 효과적으로 트리의 크기를 줄일 수 있다. 둘째, 데이터 스트림 환경에서 발생한 모든 항목집합에 대한 정확한 출현 빈도수를 관리하지 않고 그 노드까지의 경로로부터 생성 가능한 항목집합들 중 가장 큰 출현 빈도수와 가장 작은 출현 빈도수만을 저장하여 이 값들로 다른 부분항목집합들의 출현 빈도수를 추정하면 메모리 사용량을 줄일 수 있다. 이러한 특징을 이용하여 프라임 패턴 트리를 정의한다.

[정의 1] 프라임 패턴 (*Prime pattern*)

전위 트리상의 동일한 전위 노드(prefix node)를 공유하는 두 항목집합 e_1, e_2에 대하여 $e_1 = i_1 i_2 \cdots i_k$, $e_2 = e_1 i_{k+1} \cdots i_{k+n}$ ($i_k \in I$)이라 나타낼 때 e_1, e_2의 지지도 차가 $Supp(e_1) - Supp(e_2) \leq S_\delta$일 때 항목집합 $i_{k+1} \cdots i_{k+n}$ (n≥1)를 두 항목집합간의 *프라임 패턴*이라고 정의한다.

전위 트리의 부모-자식 관계에 있는 연속된 두 노드는 정의 1에 따라 두 노드에서 관리되는 항목집합들의 지지도 차이가 사전에 정의된 병합 임계값 S_δ Î (0, 1) 이하일 때 하나의 노드로 병합하여 관리하므로 필요한 노드 수를 S_δ 값에 따라 유동적으로 조절할 수 있는 특징을 갖는다. S_δ가 0인 경우는 같은 지지도를 갖는 두 연속적인 노드들이 하나의 노드로 병합되어 관리된다

[정의 2] 프라임 패턴 트리(*PPT*)의 구성

데이터 스트림 D_k의 k번째 새로운 트랜잭션 T_k가 생성되었을 때 프라임 패턴 트리는 다음과 같이 나타낸다.

1. *PPT*는 "*null*"값을 가지는 하나의 root노드를 가진다.
2. *PPT*의 1-레벨의 노드는 항목의 길이 $|e|$가 1인 단위 항목을 가지며, 2-레벨 이상의 노드는 $|e| \geq 1$인 프라임 패턴을 갖는다.
3. 두 항목 X, Y간의 *항목 결합 연산자* ⊎를 $X \uplus Y = \{z | z=xy, x \in X, y \in Y\}$로 정의할 때, *PPT*의 *root*로부터의 경로 $root \rightarrow e_1 \rightarrow \cdots \rightarrow e_{j-1} \rightarrow e_j \rightarrow e$에 해당하는 노드가 나타내는 항목집합들의 집합 E(e)은

E(*e*) ={*y*| *y*= $e_1 \cdots e_{j-1}\ e_j$ ↳ {*P*(*e*)-↗ }}같이 정의된다. 여기서 *P*(*x*)={*x*|*x* ⊄ *X*}는 항목집합 *X*을 구성하는 단위 항목들의 멱집합이다.

4. *PPT*의 각 노드에는 자신에 해당하는 정보 엔트리(*e, f, l, children_ptr*)를 갖는다. 여기서 *e*는 프라임 패턴이고 *f*는 자신이 나타내는 집합 E(*e*)의 항목집합들 중 최대 출현 빈도수, *l*은 최소 출현 빈도수를 의미하며 *children_ptr*는 자식 노드들을 연결한 링크이다.
5. 경로 *root* → e_1 → ⋯ → e_{j-1} → e_j → *e*에 해당하는 노드의 항목집합 *e*가 $e = i_1 \cdots i_n$ ($i_s \in$ I, s ≥1)일 때, $f = C(e_1 \cdots e_{j-1}e_j i_1)$, $l = C(e_1 \cdots e_{j-1}e_j e)$이며, $f - l / |D_k| \leq S_\delta$를 항상 만족한다.

정의 1의 프라임 패턴을 하나의 노드로 구성하는 *PPT*의 노드와 경로를 정의 2와 같이 나타낼 수 있다. *PPT*는 노드를 항목으로 구성하므로 *PPT*의 하나의 경로에 대하여 항목들의 집합을 생성 할 수 있다. 그러나 1-레벨의 노드들은 전위 트리와 같이 길이 1인 항목만으로 노드를 구성한다. 길이 1의 항목들은 다른 항목으로 출현 빈도수를 추정 할 수 없기 때문에 정확한 출현 빈도수를 관리한다.

(그림 1)은 전위 트리와 프라임 패턴 트리를 비교한 것이다. 전위 트리는 서로 다른 4개의 단위 항목으로 생성 가능한 항목들을 서로 다른 경로로 나타내기 때문에 16개의 노드(2^4)가 생성 되어야 하지만 *PPT*는 프라임 패턴을 하나의 노드로 표현 할 수 있으므로 (b)와 같이 8개의 노드만 생성한다. *PPT*의 크기는 프라임 패턴의 개수나 길이에 따라 달라질 수 있으므로 프라임 패턴을 결정하는 S_δ에 영향을 받는다.

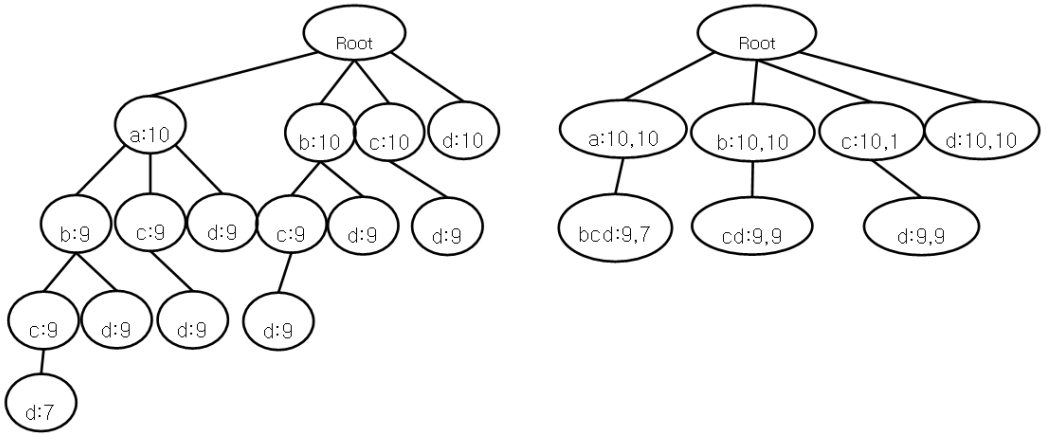
*PPT*는 전위 트리와 달리 항목집합으로 노드를 구성 할 수 있다. 이런 *PPT*에서 각각의 경로는 정의 2와 같이 정의되며 각 경로의 노드에서 관리되는 항목집합들의 부분집합들을 조합하여 모든 가능한 항목집합들을 생성 할 수 있다. 예를 들어 (그림 1)의 *PPT*에서 두 노드로 이루어진 경로 *a* → *bcd*가 있을 때 이 하나의 경로로 생성 될 수 있는 항목집합들의 집합 E(*a* → *bcd*)는 각 레벨의 노드들의 모든 가능한 조합 {*a*} ↳ {*b, c, d, bc, bd, cd, bcd*}으로 구성된다. 따라서 항목집합들의 집합 E(*a* → *bcd*)={*ab, ac, ad, abc, adb, acd, abcd*}을 나타내며 전위 트리에서 각각의 단일 경로들로 항목들을 표현할 때와 같이 모든 항목집합들을 표현할 수 있다.

3.2 프라임 패턴 트리의 노드 분리(split) 및 병합(merge)

데이터 스트림의 데이터 집합은 동적으로 변화되므로 동적으로 *PPT*를 구성하는 방법이 필요하다. *PPT*를 업데이트 할 때 함께 관리되는 항목들의 출현 빈도수 차를 고려하여 출현 빈도수 차가 S_δ이상일 때 하나의 노드를 두 독립된 노드로 나누어 주거나, 서로 독립적으로 존재 하던 노드들의 출현 빈도수가 S_δ이내 일 때 두 노드를 하나의 노드로 구성한다.

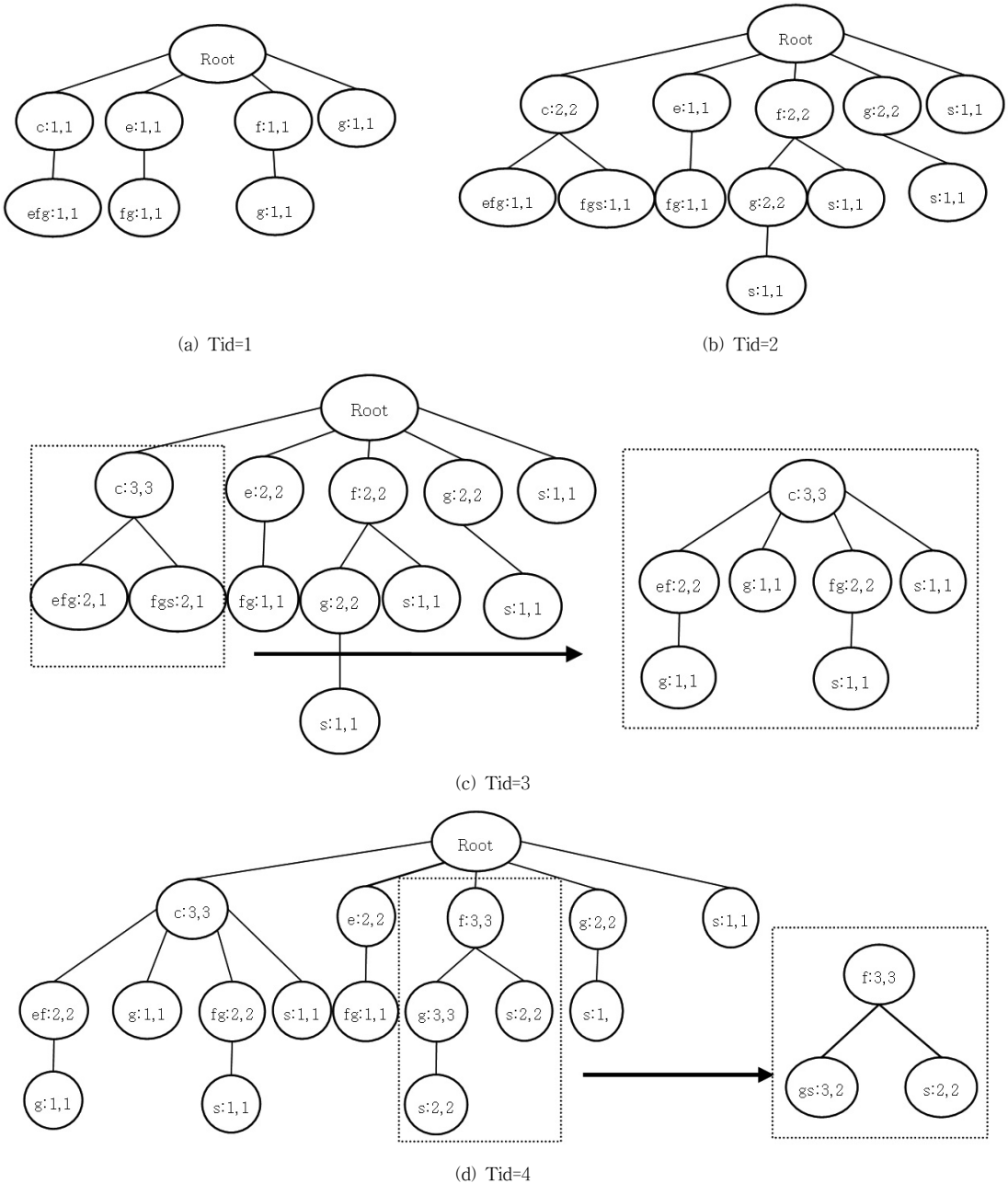
(그림 2)는 4개의 트랜젝션으로 *PPT*를 구성 하는 것을 보여 준다. (그림 2)의 (*a*)와(*b*)는 길이 1인 항목들은 1-레벨에서 관리되고 프라임 패턴이 하나의 노드로 구성되는 *PPT*의 기본적인 특성을 보여 준다. (그림 2)의 (*c*)는 *PPT*에서 하나의 노드가 분리되는 것을 보여 준다. *PPT*의 임의의 노드 *m*은 *f* 의 증가로 인해 *m*의 최소 항목집합과 최대 항목집합의 지지도 차가 S_δ 보다 클 때, 즉, $f - l / |D|_k > S_\delta$ 일 때, 여러 노드로 분리된다. 노드 *m*의 분리 과정에서는 각 리프 레벨 항목이 개별 노드로 분리되어 *m*의 자식 노드로 삽입된다. 예를 들어, *PPT*의 두 경로 *root* → *c* → *efg*, *root* → *c* → *fgs*에서 항목 *efg, fgs*에 대하여 각각 $f - l / |D|_k = 1/3 > S_\delta$ (*current* S_δ=0)이므로 *efg, fgs*는 더 이상 프라임 패턴이 아니다. 따라서 각각의 노드를 독립된 두 노드들로 분리한다. *PPT*는 전위 트리와 달리 하나의 경로로 항목들의 집합을 생성하므로 노드를 분리 할 때 (*c*)처럼 두 개의 노드로 분리 한 후 분리된 노드와 그 노드의 하위 트리(subtree)를 루트 노드를 제외한 상위 노드로 복사한다.

노드를 분리 하는 경우와 함께 (그림 2)의 (*d*)에서 보는 것과 같이 서로 독립적으로 존재하는 두 노드를 병합할 수도 있다. 노드를 병합하는 경우는 다음의 두 경우이다. 첫째, 새로운 트랜잭션으로 생성된 항목들을 *PPT*에 삽입시킬 때



(a) 전위 트리 (b) 프라임 패턴 트리

(그림 1) 전위 트리와 프라임 패턴 트리 구조 비교



(a) Tid=1 (b) Tid=2

(c) Tid=3

(d) Tid=4

(그림 2) 프라임 패턴 트리의 구성

이 항목과 기존에 *PPT*에 존재하던 항목들 간의 출현빈도수 차가 S_δ이내이면 기존의 노드의 항목으로 병합할 수 있다. 두 번째 같은 경로 위에서 서로 독립적으로 존재하던 부모-자식 관계의 두 노드 사이의 출현 빈도수가 S_δ이내일 때 하나의 노드로 병합한다. 예를 들어, (그림 2-(*d*))는 두 번째 경우로 한 경로 $root \rightarrow f \rightarrow g \rightarrow s$에서 두 노드의 출현 빈도수 차가 현재의 S_δ 이내일 경우 하나의 프라임 패턴으로 병합하여 하나의 노드로 관리한다.

*PPT*를 구성 할 때 프라임 패턴의 길이를 결정하는 전지임계값 S_δ는 동적으로 변화 될 수 있으며 S_δ에 의해 *PPT*의 노드가 분리 되거나 병합될 수 있으므로 유동적으로 *PPT*의 크기를 조절 할 수 있다.

예제 1. <표 1>의 트랜잭션 데이터 베이스를 이용하여 *PPT*를 구성한다.

〈표 1〉 트랜잭션 데이터 베이스

Tid	S_δ	항목
1	0	*c,e,f,g*
2	0	*c,f,g,s*
3	0	*c,e*
4	1	*f,g,s*

4. PPT를 이용한 빈발 항목집합 탐색

4.1 PPT의 프라임 패턴으로 생성될 수 있는 항목집합의 출현 빈도수 추정

*PPT*의 각 노드까지의 경로로 생성 될 수 있는 항목집합들 중 출현 빈도수가 가장 큰 항목집합의 출현 빈도수와 출현 빈도수가 가장 작은 항목집합의 출현 빈도수만을 관리한다. 따라서 두 개의 출현 빈도수만으로 다른 항목집합들의 출현 빈도수를 추정해야 한다. 길이가 긴 항목집합의 출현 빈도수는 길이가 짧은 항목에 비해 작은 값을 가진다. 따라서 항목집합의 길이 차에 따라 프라임 패턴으로 생성된 항목의 출현 빈도수가 결정된다.

[정의 3] 항목집합의 출현 빈도수 추정

*PPT*의 경로 $root \rightarrow e_1 \rightarrow \cdots \rightarrow e_{j-1} \rightarrow e_j \rightarrow e$을 가지는 노드에서 생성 가능한 항목집합들의 출현 빈도수를 추정할 때, 한 노드의 항목집합 $e=i_1 \oplus i_s$ ($i_s \in I$, s $\geq$1), f, l에 대하여 생성될 수 있는 항목집합 $e_1 \cdots e_{j-1} e_j i_1 \cdots i_c$ ($1 \leq c \leq s$)의 출현 빈도수는 $l \leq c(e_1 \cdots e_{j-1} e_j i_1 \cdots i_c) \leq f$을 만족하고 이 항목집합과, 이 노드로부터 생성 가능한 가장 출현 빈도수가 큰 항목집합과의 길이 차 υ에 대하여 두 항목집합 간의 출현 빈도수 차이를 함수 $F(\mathrm{x})$라 할 때 항목집합 $e_1 \cdots e_{j-1} e_j i_1 \cdots i_c$의 추정 출현빈도수는 $f - F(\upsilon)$으로 계산 할 수 있다.

출현 빈도수를 추정 하는 함수는 데이터 집합의 특성을 고려하여 다양한 형태로 정의할 수 있다. 본 논문에서는 항목집합의 길이 차에 따른 빈도수 차를 나타내는 함수 $F(x)$를 항목집합의 길이가 길어질수록 일정하게 출현 빈도수가 감소된다고 보는 비례 함수 $F_L(x)$와 항목집합의 길이가 길어질수록 출현 빈도수 길이에 반비례하여 감소되는 함수 $F_I(x)$라 하고 다음과 같이 표현한다.

$F_L(x)=\alpha x$ ($\alpha = \frac{f-l}{\mathrm{n}-1}$, x: 두 항목집합간 거리차, n: 항목집합의 길이 $|e|$)

$F_I(x)=\alpha \cdot \sum_{k=1}^{x-1} \frac{1}{k}$ ($\alpha = \frac{f-l}{\sum_{k=1}^{n-1} \frac{1}{k}}$, x: 두 항목집합간 거리차, n: 항목집합의 길이 $|e|$)

예제 2. *PPT*의 경로 $root \rightarrow e_1 \rightarrow \cdots \rightarrow e_{j-1} \rightarrow e_j \rightarrow e$을 가지는 노드의 항목집합 e가 $e=i_1 \oplus i_s$ ($i_s \in I$, $s \geq 1$)이고 이 경로로 생성 가능한 가장 출현 빈도수가 큰 항목집합 $e_1 \cdots e_{j-1} e_j i_1$와 추정 될 항목 $e_1 \cdots e_{j-1} e_j i_1 \cdots i_c$($1 \leq c \leq s$)사이의 항목 간 길이 차를 $\upsilon = |e_1 \cdots e_{j-1} e_j i_1 \cdots i_c| - |e_1 \cdots e_{j-1} e_j i_1|$라 할 때, 생성된 항목 $e_1 \cdots e_{j-1} e_j i_1 \cdots i_c$의 출현 빈도수는 정의 3과 두 함수 $F_L(x)$와 $F_I(x)$를 이용하여 다음과 같이 추정할 수 있다.

$$estimated\_C(e_1 \cdots e_{j-1} e_j i_1 \cdots i_c) = f - \mathrm{F_L}(\upsilon) = f - \frac{f-l}{\mathrm{n}-1} \cdot \upsilon - 1$$

$$estimated\_C(e_1 \cdots e_{j-1} e_j i_1 \cdots i_c) = f - \mathrm{F_I}(\upsilon) = f - \frac{f-l}{\sum_{k=1}^{n-1} \frac{1}{k}} \cdot \sum_{k=1}^{x-1} \frac{1}{k}$$

*PPT*의 노드에서 추정된 출현 빈도수는 정확한 출현 빈도수와 차이를 가진다. 그러나 *PPT*에서 하나의 노드로 구성 된 항목집합들은 프라임 패턴으로서 각 항목집합의 실제 출현 빈도수 차가 S_δ이내의 값을 가지므로 실제 출현 빈도수와 추정된 출현 빈도수 사이에 다음과 같은 오차 속성을 갖는다.

[속성 1] 출현 빈도수 추정에 따른 최대 오차 (*Error*)

예제 2에서 $e_1 \cdots e_{j-1} e_j i_1 \cdots i_c$의 추정 출현 빈도수 $estimated_C(e_1 \cdots e_{j-1} e_j i_1 \cdots i_c)$와 실제 출현 빈도수 $original_C(e_1 \cdots e_{j-1} e_j i_1 \cdots i_c)$의 출현 빈도수 추정에 따른 최대 오차 범위는 S_δ 이내이다.

증명.

$l \leq estimated\_C(e_1 \cdots e_{j-1} e_j i_1 \cdots i_c) \leq f$, $l \leq original\_C(e_1 \cdots e_{j-1} e_j i_1 \cdots i_c) \leq f$ 이고, *PPT*의 한 노드는 항상 $f-l \leq S_\delta$를 만족하므로 $Error(e_1 \cdots e_{j-1} e_j i_1 \cdots i_c) = |original\_C(e_1 \cdots e_{j-1} e_j i_1 \cdots i_c) - estimated\_C(e_1 \cdots e_{j-1} e_j i_1 \cdots i_c)| \leq S_\delta$ 이다. □

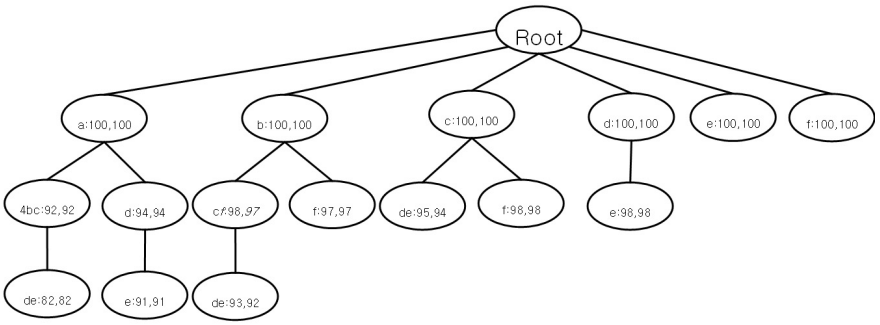
속성 1과 같이 *PPT*에서 각각의 항목집합들에 대해 추정된 출현 빈도수와 실제 출현 빈도수 사이의 오차는 S_δ에 의해 영향을 받는다. S_δ가 커질수록 하나의 노드에 관리되는 프라임 패턴의 길이가 길어지므로 생성 할 수 있는 다른 항목집합들이 많아지고, 각 항목집합들에 대하여 출현 빈도수 추정 과정을 거쳐야 하므로 오차가 커진다. 이런 *PPT*의 특징을 활용하면 동적으로 S_δ를 변화시켜 *PPT*의 크기와 오차를 조절 할 수 있다.

4.2 PPT를 이용한 메모리 사용량 조절

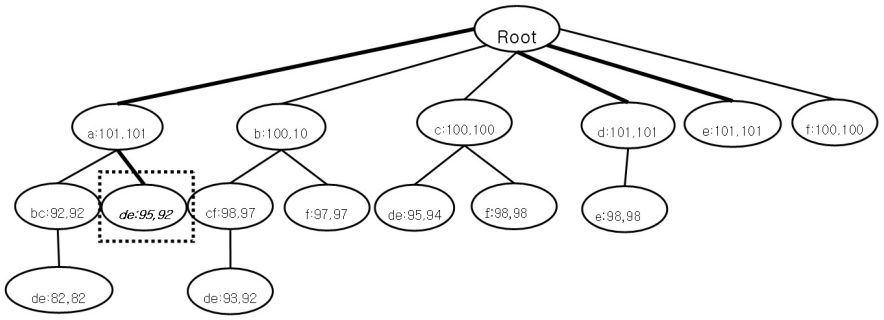
데이터 스트림에서 마이닝 대상이 되는 데이터 집합은 마이닝 수행 중에도 동적으로 변한다. 사용 가능한 메모리는 한정되어 있으므로 제한된 조건에서 효과적으로 빈발 항목집합을 탐색하기 위해 동적인 메모리 관리가 필수적이다. 데이터 스트림에 대한 마이닝을 제한된 메모리 공간에서 효율적으로 수행하기 위해서는 마이닝 수행 과정에서 요구되는 메모리 사용량을 데이터 스트림의 변화에 따라 동적으로 적응시킬 필요가 있다.

예제 3. 현재의 S_δ값 2를 3으로 변화 시킬 때 새로 들어온 트랜잭션 {*a,d,e*}에 의해 *PPT*는 (그림 3-(*a*)) 상태에서 (그림 3-(*b*))와 같이 갱신된다.

(그림 3)은 새로운 트랜잭션{*a,d,e*}에 의해 *PPT*가 갱신되는 과정을 보여 준다. 변화된 S_δ에 의해 같은 경로상의 두 노드가 하나의 노드로 병합된다. S_δ는 *PPT*를 구성하는데 주요한 기준이 되며, S_δ에 의해 프라임 패턴의 길이가 결정



(a) 갱신 전 *PPT* (S_δ = 2)



(b) 갱신 후 *PPT* (S_δ= 3)

(그림 3) S_δ의 변화에 따른 *PPT* 갱신

되며 위의 예제에서 볼 수 있듯이 *PPT*의 깊이를 결정한다. S_δ와 *PPT*의 크기 사이에 다음과 같은 속성을 갖는다.

[속성 2] S_δ에 의한 트리 크기 결정

S_δ의 크기에 따라 *PPT*의 크기가 결정 된다. S_δ가 크면 *PPT*의 노드 수가 적고 반대로 S_δ가 작으면 노드 수가 많아지므로 *PPT*의 크기는 커진다.

증명. 그림 1의 전위 트리와 *PPT*의 구조를 살펴보면 전위 트리는 모든 항목집합이 가능한 모든 경로로 표현 되어야 하기 때문에 길이가 가장 긴 패턴에 의해 트리의 깊이(depth)가 결정된다. 이것과 비교하면 *PPT*는 가장 긴 항목집합의 길이와 트리의 깊이 사이의 연관성을 갖지 않는다. 1-레벨의 노드들은 전위 트리처럼 단위항목으로 구성되지만 2-레벨 이상의 노드들은 항목집합으로 구성되기 때문이다. S_δ가 큰 값을 가지면 1-레벨을 제외한 나머지 항목집합들이 한 노드로 구성될 가능성이 높으며, 분리 될 가능성은 적으므로 트리의 크기는 작아진다. 반대로 S_δ가 작은 값이면 독립적으로 각각의 노드로 구성될 가능성이 높으므로 깊이가 커져 트리의 크기가 커질 수 있다. □

속성 1과 속성 2와 같이 S_δ는 *PPT*의 크기와 결과 집합의 정확도에 영향을 줄 수 있다. S_δ가 크면 *PPT*의 크기는 작아지지만 한 노드에서 생성되는 항목들의 추정된 출현 빈도수에 더 큰 오차 범위를 허용하므로 정확도는 낮아진다. 한편 S_δ의 값이 작을 경우 오차는 줄일 수 있으나 *PPT*의 크기가 커질 수 있다. 마이닝 수행과정에서 동적으로 메모리를 관리 하기 위해 새로운 노드가 *PPT*에 생성될 때 제한된 노드수 M_{node} 와 현재 생성된 노드수 C_{node} 를 비교하여 S_δ를 다음과 같이 변화시킨다.

$$\begin{cases} S_\delta^{new} : S_\delta^{old} + a, & (\text{when } C_{node} < M_{node}) \\ S_\delta^{new} : S_\delta^{old} - a, & (\text{when } C_{node} \geq M_{node}) \end{cases}$$

a는 한번의 메모리 적응 과정에 따른 S_δ의 변화량을 나타내며 사용자에 의해 정의된다. M_{node}를 가용 메모리 크기보다 작은 값으로 설정하면 M_{node}이상의 노드도 생성하면서 S_δ를 조절할 수 있으므로 효과적이다.

4.3 PPT를 이용한 빈발 항목집합 탐색

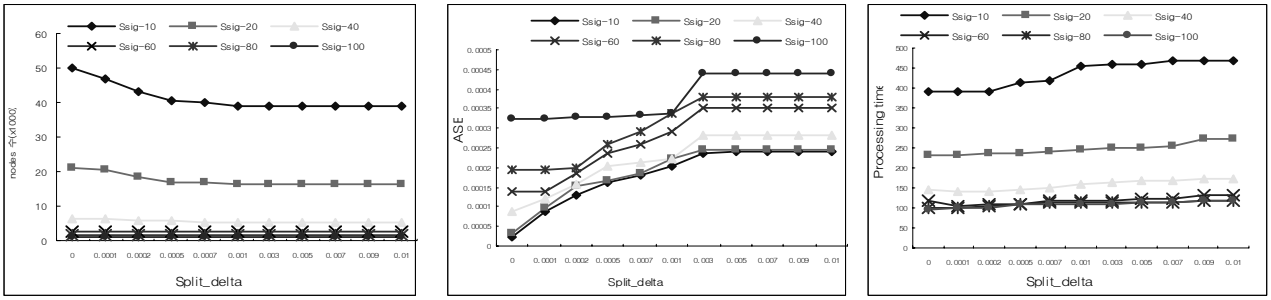
데이터 스트림 환경에서 빈발 항목집합을 탐색하기 위해 앞에서 제시한 *PPT*에 보다 의미 있는 정보만을 메모리 상에 저장하기 위해 *estDec*방법[10]의 지연 추가 및 전지 작업을 수행한다. *estDec* 방법과 같이 매개변수 갱신 단계, 출현 빈도수 및 노드 갱신 단계, 항목집합 추가 단계 및 빈발 항목집합 탐색 단계와 같이 네 단계로 구성된다. 데이터 스트림 D_{k-1}에 새로운 트랜잭션 T_k가 발생하였을 때, 빈발 항목집합 탐색 단계를 제외한 세 단계는 순차적으로 수행되며, 빈발 항목집합 탐색 단계는 실시간 마이닝 결과가 요청되었을 때 수행된다. (그림 4)는 *PPT*를 이용하여 빈발 항목집합을 탐색하는 과정을 보여준다. 각 단계에 대한 구체적인 내용은 다음과 같다.

```
Input: a data stream D_k
Output: a complete set of frequent itemsets L_k
  S_min : A minimum support
  S_sig : A given significant support
  PPT : A Prime pattern Tree that maintains a set of itemset entries e(f, l, links)'s

1:   PPT = ∅;
2:   for each new transaction T_k in D_k {
/* Parameter Updating Phase */
4:        |D|_k = |D|_k-1 +1;

 /* Count Updating Phase */
5:   for all itemset e s.t. e∈(2^T_k -{∅}) and e∈PPT {
6:                f_k(e) = f_k-1(e) + 1 or l_k(e) = l_k-1(e) + 1;
7:        if |e| >1 then
8:             if  f_k(e) -  l_k(e) > S_δ then
9:              node_split(e);
10:            if f_k-1(e) - l_k(e)  <  S_δ  then
11:             node_merge(e's parent node , e);
12:            if equal(e , e'sibling node) then
13:             eliminate duplication;
/* pruning */
14:       if  S_k(e)<S_sig and |e|>1 then
15:             Eliminate e and it's children nodes from PPT;
16:  }
      /*item - insertion phase */
17:   item_filtering(T_k);
18:   for all e s.t. e∈(2^T_k -{∅}) and e∉PPT {
19:     if |e| = 1 then
20:       Insert e into PPT and Initialize;
21:       if C_node > M_node ( or < M_node) then
22:        change  S_δ; //  S_δ± α
23:     else
24:       estimate frequency with estDec method[10]
25:       if estimated_frequency > S_ssig then
26:          if estimated_frequency – e-f(N_c) < S_δ then
27:             insert e as a node N_c's itemset;
28:          else
29:             insert e into PPT as a N_c's child node;
30:             if C_node > M_node ( or < M_node) then
31:              change  S_δ; //  S_δ± α
32:   }
     /* Frequent Itemset Selection Phase */
33:    L_k = ∅;
34:    for all e ∈ PPT {
35:       if S_k(e) ≥S_min then
36:          L_k=L_k ∪{e};
37:    }
38: }
```

(그림 4) PPT 탐색 과정



(a) 노드수 변화 (b) 정확도 변화 (c) 수행시간(μs)

(그림 5) Split_delta S_δ의 변화에 따른 비교

Step 1. 매개변수 갱신 단계: 트랜잭션 T_k가 생성될 때 전체 트랜잭션의 수를 하나 증가시킨다((그림 4)의 4번째 줄).

Step 2. 출현 빈도수 갱신단계: 새로 생성된 T_k의 항목집합들에 대하여 영향 받는 *PPT* 노드들의 출현 빈도수를 1씩 증가시킨다. 이때 갱신된 노드들 중 1-레벨 노드를 제외한 노드들은 출현 빈도수가 S_{sig}미만이면 이 노드들로 구성된 항목집합들은 빈발 항목집합이 될 가능성이 적으므로 *PPT*에서 전지시키며(6번째 줄), 새로 추가된 트랜잭션에 의해 탐색된 *PPT*경로의 노드들을 검사하여 노드를 분리, 병합하거나 중복된 노드들을 제거한다(7-13번째 줄).

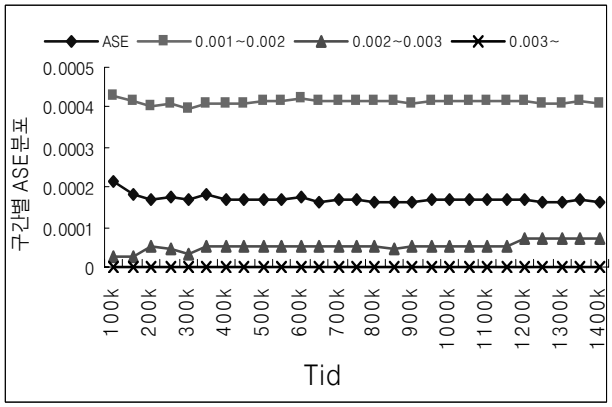
Step 3. 항목집합 추가 단계: 새로 들어온 트랜잭션 T_k에 의해 생성된 항목집합들 중에서 *PPT*에 포함되어 있지 않는 항목집합들을 새로 삽입한다. 이때 삽입 대상이 되는 길이 1인 항목집합들은 출현 빈도수 추정 과정을 거치지 않고 *PPT*의 1-레벨의 노드로 바로 삽입된다. 반면에 길이 2 이상의 항목집합들은 현재 시점에서 빈발 항목집합이 될 가능성이 있는 항목집합들만이 *PPT*의 노드로 삽입된다. 따라서 *PPT*의 2-레벨 이상의 노드들은 생성된 T_k에서 특정 항목집합에 대하여 기존에 생성된 노드들을 탐색하여 추정된 출현 빈도수가 S_{sig}이상의 값을 가질 때에만 *PPT*에 추가되고 그 외의 항목집합들은 무시된다(17-32 번째 줄). T_k의 특정 항목의 출현 빈도수를 추정하는 방법은 해당 항목집합의 부분항목집합들을 이용한다. n-항목집합 e의 출현 빈도수를 추정하기 위해 e의 (n-1)-부분 항목집합들의 출현 빈도수를 고려한다. $P_m(e)$이 항목집합 e를 구성하는 단위항목들 중에서 m개의 단위항목으로 구성되는 모든 항목집합들의 집합으로 정의되고 $P^C_m(e)$가 $P_m(e)$에 속하는 모든 항목집합들의 출현빈도 수 중에서 서로 다른 값들로 구성되는 집합이라고 할 때, 특정 항목집합 e의 출현빈도수 $C(e)$를 다음과 같이 계산한다. $C^{max}(e) = min(P^C_{n-1}(e))$, $C^{min}(e) = max\ (\{C^{min}(a_i \cup a_j) \mid \forall a_i, a_j \in P_{n-1}(e)$ and $i \neq j\})$,$C(e) = C^{max}(e)$. 그리고, 항목집합 추가 단계에서 새로운 항목집합들이 생성되어 *PPT*에 삽입 될 때 현재 시점에서의 메모리 사용량과 제한된 메모리 크기를 비교하여 *PPT*의 크기를 조절하기 위해 분리 임계값 S_δ를 동적으로 변화 시킨다(20-21, 30-31 번째줄).

Step 4. 빈발 항목집합 선택 단계: 마이닝 결과가 요구될 때, *PPT*의 모든 경로를 탐색하여 현재 시점에서의 빈발 항목집합을 찾는다.

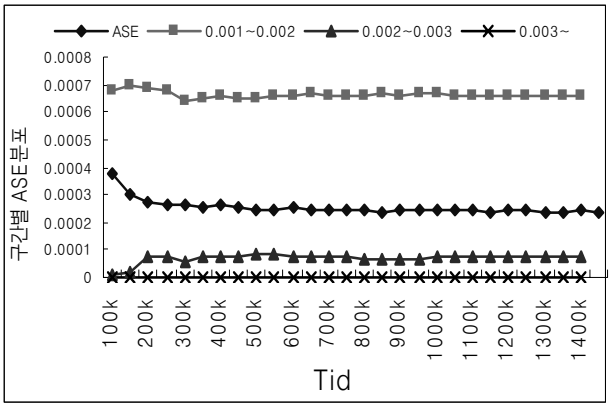
5. 실험 및 결과

본 장에서는 여러 실험을 통하여 본 논문에서 제안한 *PPT*의 성능을 검증한다. 본 실험에서 사용한 데이터 집합은 [15]와 같은 방법으로 생성되었고 N이 1K인 *T5.I4.D1400K* 인 데이터 집합을 생성 하였다. $|T|$, $|I|$, $|D|$ 및 N은 각각 트랜잭션의 평균 길이(average transaction size), 잠재적 최대 빈발항목의 평균 길이(average maximal potentially frequent itemset size), 트랜잭션의 총 수 및 데이터 집합을 구성하는 단위항목의 총 수를 나타낸다. [15]에서는 Synthetic data를 생성하기 위해 다음 트랜잭션의 크기를 $|T|$값을 평균으로 하는 포아송 분포로부터 선택한다. 다음으로 N값에 대하여 같은 확률 p를 갖는 이항분포를 따라 선택된 항목들을 트랜잭션에 할당하게 된다. 각 실험에서 데이터 집합의 트랜잭션들은 데이터 스트림 환경처럼 하나씩 차례로 탐색되며 모든 실험에서 지연 추가와 전지 작업을 위한 임계값 S_{sig}는 사전에 정의되는 최소 지지도 S_{min}값에 대한 상대적인 값으로 정의하고 임계값 S_{sig}가 p%로 표시되었을 때, 실제값은 $S_{min} \times (p/100)$이다. *PPT*구성을 위한 또 다른 임계값 S_δ는 S_{min}과는 독립적인 값으로 항목집합간의 지지도 차를 의미한다. 본 실험에서는 최소지지도 S_{min}은 0.001로 고정하고 결과의 정확도를 비교하기 위해 생성된 빈발 항목집합을 *Apriori* 알고리즘[15]의 결과 집합과 비교한다. *PPT*에서 각 항목집합의 출현 빈도수를 추정하기 위해 예제 1의 $F_1(x)$함수를 이용한다. 모든 실험은 500MB의 램(RAM)을 가진 1.8GHz 팬티엄 컴퓨터와 리눅스 7.3 환경에서 실험되었으며 모든 프로그램은 C언어로 구현되었다.

(그림 5)은 S_δ의 변화에 따른 노드수, *ASE*(*Average Support Error*), 수행시간의 변화를 보여준다. (그림 5)의 (*a*)는 S_δ가 커질수록 노드수가 줄어드는 것을 보여준다. S_δ가 클수록 한 노드에서 관리 될 수 있는 항목의 수가 늘어나기 때문에 *PPT*의 노드수는 작아지고 그 노드에서 추정하는 단위 항목의 출현 빈도수의 오차는 커진다. (*b*)는 이런 오차를 나타낸 것으로 [10]와 같은 방법으로 두 마이닝 결과 집합 $R_1 = \{(e_i, S_1(e_i)) \mid S_1(e_i) \geq S_{min}\}$ 과 $R_2 = \{(e_j, S_2(e_j)) \mid S_2(e_j) \geq S_{min}\}$가 있을 때 다음과 같이 계산된다.

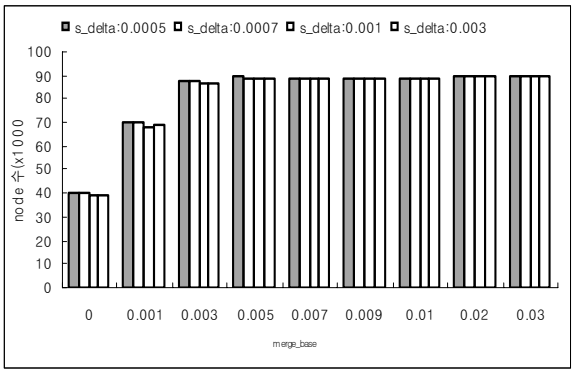


(a) S_δ = 0.0005 구간별 Error분포

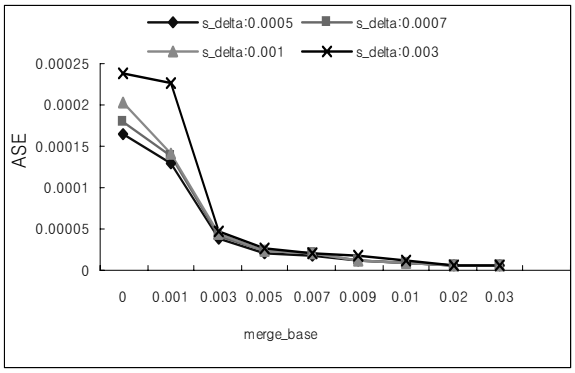


(b) S_δ = 0.0009 구간별 Error분포

(그림 6) 구간별 Error 분포



(a) *merge_base*변화에 따른 노드수



(b) *merge_base*변화에 따른 ASE변화

(그림 7) *merge__base* 변화에 따른 노드수, ASE 변화

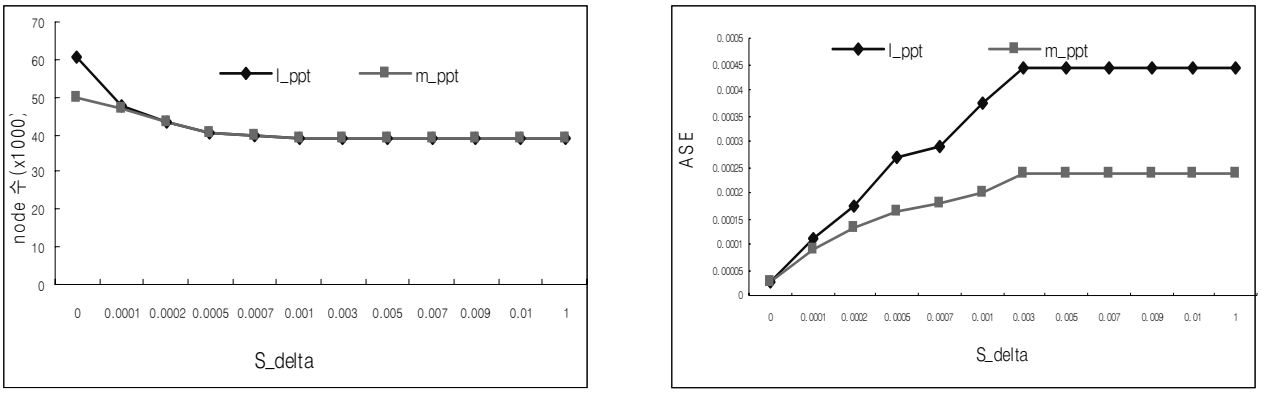
$$ASE\ (R_2|R_1) = \frac{\sum_{e_m \in R_1 - R_1 \cap R_2} S_1(e_m) + \sum_{e_m \in R_1 \cap R_2} \{|S_2(e_m) - S_1(e_m)|\} + \sum_{e_m \in R_2 - R_1 \cap R_2} S_2(e_m)}{|R_1|}$$

$|R_1|$은 결과 집합 R_1의 항목집합의 개수를 나타낸다. $ASE(R_2|R_1)$이 작을수록 R_2의 마이닝 결과 집합이 R_1의 결과 집합과 유사하다. 빈발 항목집합 마이닝 결과의 정확도 표현을 위해 $ASE(R_{PPT}|R_{Apriori})$를 사용하였다. R_{PPT} 와 $R_{Apriori}$는 각각 본 논문에서 제안한 방법과 *Apriori* 알고리즘에 의해 구해진 마이닝 결과를 나타낸다. (*c*)는 수행시간을 나타낸 것으로 하나의 트랜잭션이 추가 되었을 때 처리하는데 소모되는 시간의 변화를 나타낸 것이다. S_δ가 커질수록 한 노드에서 관리되는 항목의 수가 증가하게 되어 각 노드에 대한 탐색 시간이 길어지므로 전체적인 트리의 탐색 시간은 증가된다.

(그림 6)은 오차에 대한 *PPT*의 특성을 잘 보여 준다. 3장의 정의에 의해 *PPT*의 상위 노드는 하위 노드에 비해 정확한 출현 빈도수를 가지며, 하위 노드에서 생성된 항목집합들 중 프라임 패턴들로 생성된 항목집합들이 많기 때문에 비교적 높은 오차를 가진다. (그림 6)의 (*a*)와(*b*)에서 확인할 수 있는 것처럼 항목집합의 지지도가 낮은 하위 노드에서 생성된 항목집합들의 *ASE*가 지지도가 큰 상위 노드와 비교하여 크다는 것을 알 수 있다.

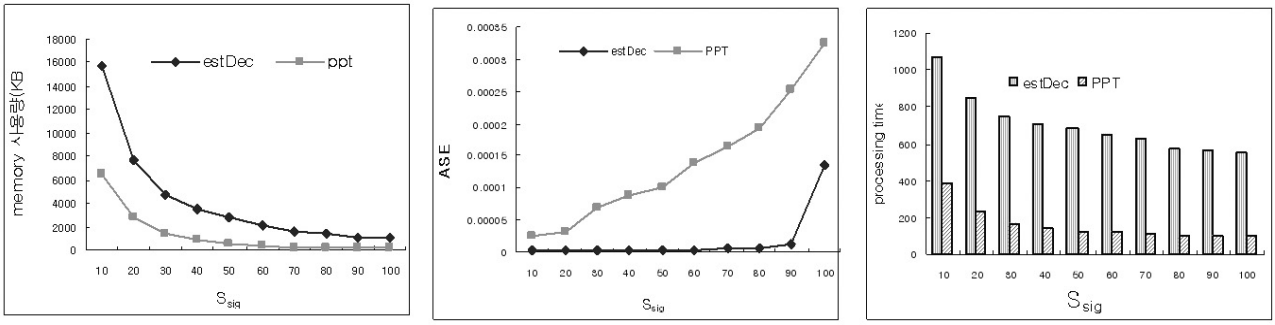
(그림 7)은 두 노드를 병합할 때 경계가 되는 값인 *merge_base*에 대한 노드수와 *ASE*변화를 보여주는 그래프이다. 두 노드를 병합할 때 하위 노드의 지지도 또는 지연 추가 과정을 거쳐 *PPT*에 삽입될 단위항목의 지지도가 *merge_base*이상이 될 경우만 한 노드로 병합되고 그렇지 않을 경우 병합될 대상의 지지도 차가 S_δ이내라도 각각 독립적으로 노드를 구성하게 된다. 따라서 *merge_base*가 높을수록 노드수는 증가하고 *ASE*는 감소한다.

(그림 8)은 예제 2에서 소개한 출현 빈도수를 구하는 두 종류의 함수에 대한 노드수, *ASE*, 출현 빈도수의 S_δ에 따른 변화를 보여준다. *l_ppt*는 한 노드에서의 단위항목의 위치에 따라 출현 빈도수가 순차적으로 감소하는 함수 $F_L(x)$를 이용하여 출현 빈도수를 계산 한 것이고 *m_ppt*는 각 단위항목의 위치에 따라 출현 빈도수가 반비례한 형태로 변화되는 함수 $F_I(x)$를 이용하여 출현 빈도수를 계산한 것이다. 두 함수 $F_L(x)$와 $F_I(x)$에 대하여 (그림 8)에서 보이는 것처럼 노



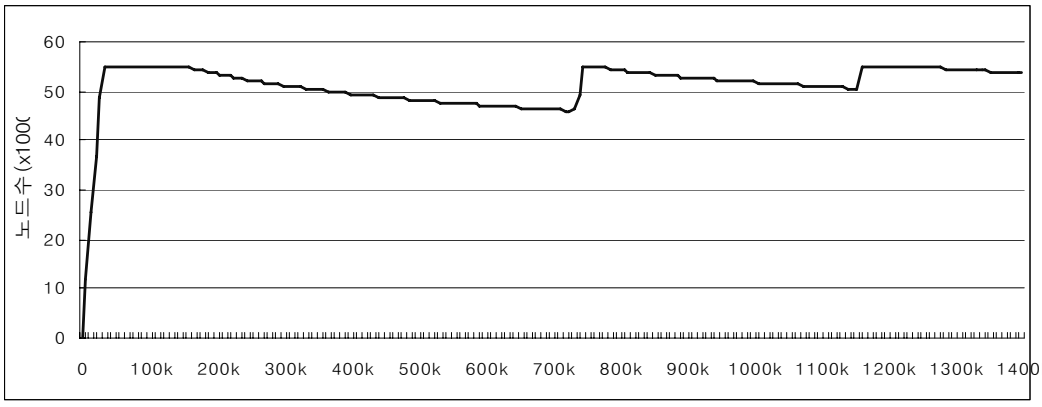
(a) 노드수 비교 (b) ASE 비교

(그림 8) 출현 빈도수 추정 방법의 변화에 따른 비교



(a) 메모리 사용량 비교 (b) *ASE* 비교(c) (c) 수행시간 비교(μS)

(그림 9) *estDec*방법과 *PPT*구조를 이용한 방법의 비교



(그림 10) *PPT*의 Memory adaptation

드수의 차이는 크지 않지만 정확도는 $F_I(x)$함수를 이용하여 출현 빈도수를 추정하는 방법이 더 효율적임을 알 수 있다.

(그림 9)는 *estDec*방법[10]와 *PPT*를 이용한 방법간의 노드수, *ASE*, 수행시간을 비교한 것이다. *PPT*구조를 이용하고 지연 추가와 전지 작업을 통해 (*a*)에서와 같이 노드수를 크게 줄일 수 있다. 그러나 *PPT*는 2-레벨 이상의 노드들은 프라임 패턴으로 구성되고 두 개의 출현 빈도수만을 관리함으로 나머지 항목들에 대해서는 출현 빈도수 추정과정을 거쳐야만 한다. 따라서 (*b*)에서와 같이 *ASE*는 *estDec*방법과 비교하여 더 큰 값을 가진다. 이는 *estDec*방법이 많은 수의 빈발 항목집합이 병합된 항목집합의 출현 빈도수 추정 과정에서 발생하는 오차 없이 정확히 모니터링 되기 때문이다. (*c*)는 두 방법의 수행시간을 비교한 것이다. 수행시간은 트리를 구성하고 탐색하는 시간으로 *PPT*를 이용한 방법은

*estDec*방법과 비교하여 노드수가 크게 줄기 때문에 전체적으로 트리를 탐색하는 시간은 줄어든다.

(그림 10)은 *PPT*구조를 이용한 메모리 사용량을 최적화 시키는 것을 보여주고 있다. 최대 생성할 수 있는 노드수는 55000개이고 최대 노드의 90%선을 기준으로 생성된 노드의 수를 검사하여 S_δ를 변화 시킨다. (그림 10)에서 보는 것처럼 S_δ가 0일 때 초기의 트랜잭션들로 생성된 노드수는 크게 증가 하지만 S_δ를 조절하면서 기준선인 50000와 최대 생성 가능한 55000개 사이에서 변화되고 있는 것을 볼 수 있다.

6. 결 론

데이터 스트림 환경에서 데이터 집합은 사전 정의되지 있지 않고 마이닝을 수행하는 중에 동적으로 변화된다. 따라서 데이터 스트림으로 구성된 데이터 집합에서 빈발 항목집합을 탐색 할 때 정확한 마이닝 결과를 얻기는 사실상 불가능하다. 끊임없이 생성되는 트랜잭션들의 모든 정보를 메모리상에 저장할 수 없기 때문이다. 이런 이유로 데이터 스트림 환경에서 마이닝을 수행할 때 어느 정도의 오차를 허용하면서 수행시간이나 메모리 사용량을 줄일 수 있는 방법에 대한 연구가 필요하다. 본 논문에서는 *estDec*[10]의 지연 추가와 전지 작업을 적용한 새로운 데이터 구조인 프라임 패턴 트리를 정의하여 메모리 사용량을 줄이고, 필요에 따라 메모리 사용량을 동적으로 조절 할 수 있는 방법을 제안 하였다. *PPT*의 각 노드는 1-레벨을 제외하고 프라임 패턴으로 노드를 구성한다. 그리고 최소한의 출현 빈도수만을 저장하여 메모리 사용량을 크게 줄인다. 이 방법은 메모리 사용량을 동적으로 조절하는 데에도 유용하다. *PPT*의 크기가 S_δ에 의해 결정됨으로 동적으로 S_δ를 변화시키면 제한된 메모리의 크기에 알맞게 *PPT*를 구성 할 수 있기 때문이다. 5장에서 다양한 실험을 통해 *PPT*의 성능을 검증 하였다. *PPT*는 메모리의 사용량을 크게 감소시킬 수 있다. 그러나 메모리 사용량을 감소시키면 감소시킬 수록 정확도는 떨어진다. 따라서 메모리의 사용량과 정확도 사이에서 적절하게 S_δ를 조절 할 수 있어야 한다.

결과적으로 *PPT*는 무한하게 증가되는 데이터와 메모리와 같은 제한된 컴퓨팅 자원 사이에서 가장 최적화 된 결과를 얻을 수 있도록 지원한다. 본 논문에서 제안한 방법은 데이터 집합의 크기에 상관없이 한정된 메모리의 크기에 맞게 마이닝 수행과정 중 메모리 사용량을 조절함으로써 데이터 스트림 환경에서 빈발 항목집합을 효율적으로 탐색 할 수 있는 방법을 제시하였다.

참 고 문 헌

[1] M. Charikar, K. Chen, and M. Farach-Colton, “Finding Frequent Items in Data Streams,” In *Proc. of the 29th Int'l. Colloq. Automata, Language and Programming*, 2002.

[2] G.S. Manku and R. Motwani, “Approximate Frequency Counts over Data Streams,” In *Proc. of the 28th Int'l Conf. on Very Large Data Bases*, 2002.

[3] M. Datar, A. Gionis, P. Indyk, and R. Motwani, “Maintaining Stream Statistics over Sliding Windows,” In *Proc. of the 13th Ann. ACM-SIAM Symp. Discrete Algorithms*, pp.635-644, 2002.

[4] S. Guha and N. Koudas, “Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation,” In *Proc. of the 18th Int'l Conf. on Data Engineering*, pp.567-576, 2002.

[5] G. Dong, J. Han, L.V.S. Lakshmanan, J. Pei, H. Wang, and P.S. Yu. Online Mining of Changes from Data Streams: Research Problems and Preliminary Results. In *Proc. of the Workshop on Management and Processing of Data* Streams, 2003.

[6] Wei-Guang Teng, Ming-Syan Chen, Philip S. Yu. A Regression-Based Temporal Pattern Mining Scheme for Data Streams, In *Proc. of the 29th Int'l Conf on Very Large Database*, Berlin, Germany, 2003

[7] Zhihong Chong, Jeffrey Xu Yu, Hongjun Lu, Zhengjie Zhang, and Aoying Zhou. False-Negative Frequent Items Mining from Data Streams with Bursting. In *Proc. of the 10th Int'l Conf on Database Systems for Advanced Applications*, pp.422-434, 2005.

[8] L. Qiao, D. Agrawal, and A.E. Abbadi, “RHist: Adaptive Summarization over Continuous Data Streams,” In. *Proc. of the 10th Int'l Conf. on Information and Knowledge Management*, pp.469-476, 2002.

[9] M. Garofalakis, J. Gehrke and R. Rastogi. “Querying and mining data streams: you only get one look”. In *the tutorial notes of the 28th Int'l Conf. on Very Large Databases*, 2002.

[10] J. H. Chang, W. S. Lee. “Finding recent frequent itemsets adaptively over online data streams.” In *Proc. of the 9th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, Washington, DC, 24-27, August, 2003.

[11] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur, “Dynamic Itemset Counting and Implication Rules for Market Basket Data,” In *Proc. of ACM SIGMOD Int'l Conf. Management of Data*, pp.255-264, 1997.

[12] M.J. Zaki, “Generating Non-Redundant Association Rules,” In *Proc. of the 6th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, pp.34-43, 2000.

[13] R.C. Agarwal, C.C. Aggarwal, and V.V.V. Prasad, “Depth First Generation of Long Patterns,” In *Proc. of the 6th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining,* pp.108-118, 2000.

[14] C.C. Aggarwal and P.S. Yu, “Online Generation of Association Rules,” In *Proc. of the 14th Int'l IEEE Conf. on Data Engineering*, pp.402-411, 1998.

[15] R. Agrawal, and R. Srikant. Fast algorithms for mining

association rules. In *Proc. of the 20th Int'l Conf. on Very Large Databases*, Santiago, Chile, Sept., 1994.

[16] A. Hafez, J. Deogun, and V. V. Raghavan. "The Item-Set Tree: A data Structure for Data Mining." In *Proc. of the 1st int'l Conf on data warehousing and knowledge discovery*, pp. 183-192, Aug., 1999.

[17] Yun Chi, Haixun Wang, Philip S. Yu, Richard R. Muntz "Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window." In *Proc. of the 4th IEEE int'l Conf. on Data Mining*, pp.59-66, 2004.

[18] N. Jiang, and L. Gruenwald, "CFI-Stream: Mining Closed Frequent Itemsets in Data Streams," In *Proc. of the 12th ACM SIGKDD int'l Conf. on Knowledge Discovery and Data Mining*, pp.592-597, 2006.

김 민 정

e-mail : minjung978.kim@samsung.com
2002년 숙명여자대학교 전산학과(학사)
2004년 연세대학교 대학원 컴퓨터과학과 (공학석사)
2004~현 재 삼성전자 무선사업부 GSM 단말 MMI 개발 연구원
관심분야 : 데이터마이닝, 데이터스트림 마이닝

신 세 정

e-mail : starofu@database.yonsei.ac.kr
2004년 연세대학교 컴퓨터과학과(공학사)
2004년~현 재 연세대학교 컴퓨터과학과 석·박사 통합과정
관심분야 : 데이터마이닝, 데이터스트림 마이닝

이 원 석

e-mail : leewo@database.yonsei.ac.kr
1985년 미국 보스턴대학교 컴퓨터공학과 (공학사)
1987년 미국 퍼듀대학교 컴퓨터공학과 (공학석사)
1990년 미국 퍼듀대학교 컴퓨터공학과 (공학박사)
1990년~1992년 삼성전자 선임연구원
1993년~1999년 연세대학교 컴퓨터과학과 조교수
1999년~2004년 연세대학교 컴퓨터과학과 부교수
2004년~현 재 연세대학교 컴퓨터과학과 교수
관심분야 : 분산데이터베이스, 미디에이터시스템, 데이터마이닝, 데이터스트림