개방 데이터 마이닝에 효율적인 이동 윈도우 기법

장 중 혁[†] · 이 원 석[††]

요 약

근래들어 구성 요소가 빠른 속도로 지속적으로 발생되는 무한 집합으로 정의되는 데이터 스트림에 대한 개방 데이터 마이닝 방법들이 활발히 제안되고 있다. 데이터 스트림에 내재된 정보들은 시간 흐름에 따른 변화의 가능성이 매우 높다. 따라서, 이러한 변화를 빠른 시간에 분석할 수 있다면 해당 데이터 스트림에 대한 분석에서 보다 유용한 정보를 제공할 수 있다. 본 논문에서는 개방 데이터 마이닝 환경에서 효율적인 최근 빈발 항목 탐색을 위한 **이동 윈도우 기법**을 제시한다. 해당 기법에서는 데이터 스트림이 지속적으로 확장되더라도 지연 추가 및 전지 작업을 적용하여 마이닝 수행과정에서의 메모리 사용량이 매우 작게 유지되며, 분석 대상 범위의 데이터 객체들을 반복적으로 탐색하지 않기 때문에 각 시점에서 마이닝 결과를 짧은 시간에 구할 수 있다. 더불어, 해당 방법은 데이터 스트림의 최근 정보에 집중한 분석을 통해 해당 데이터 집합의 변화를 효율적으로 감지할 수 있다.

키워드 : 이동 윈도우 기법, 개방 데이터 마이닝, 데이터 스트림의 가변성, 데이터 스트림, 실시간 분석

A Sliding Window Technique for Open Data Mining over Data Streams

Joong-Hyuk Chang[†] · Won-Suk Lee[††]

ABSTRACT

Recently open data mining methods focusing on a data stream that is a massive unbounded sequence of data elements continuously generated at a rapid rate are proposed actively. Knowledge embedded in a data stream is likely to be changed over time. Therefore, identifying the recent change of the knowledge quickly can provide valuable information for the analysis of the data stream. This paper proposes **a sliding window technique** for finding recently frequent itemsets, which is applied efficiently in open data mining. In the proposed technique, its memory usage is kept in a small space by delayed-insertion and pruning operations, and its mining result can be found in a short time since the data elements within its target range are not traversed repeatedly. Moreover, the proposed technique focused in the recent data elements, so that it can catch out the recent change of the data stream.

Key Words : SlidiNg Window Technique, Open Data Mining, Changeability of a Data Stream, Data Stream, Real-time Analysis

1. 서 론

분석 대상이 되는 데이터 집합의 특성 및 해당 데이터 집합이 발생되는 응용 환경의 요구 사항에 따라 데이터 마이닝 방법을 크게 두 가지로 구분할 수 있다[12]. 하나는 *폐쇄 데이터 마이닝(closed data mining)* 방법으로서 해당 방법에서는 지식 발견의 대상이 되는 데이터 집합이 마이닝 작업 시작 이전에 명확히 정의되는 것으로 가정하며 이러한 가정은 고정적인 데이터 집합에 내재된 정보들을 추출하는 것이 데이터 마이닝의 목적이 될 때 유효하다. 해당 방법들은 대용량의 데이터 집합에 대한 마이닝 결과를 얻는데 있어서 상당한 처리 시간을 요구한다. 다른 하나는 *개방 데이터 마이닝(open data mining)* 방법으로서 근래의 변화된 응용 도메인에서 발생되는 비한정적인 데이터 집합 즉, 데이터 스트림에 대한 실시간 분석 능력을 지원한다. 즉, 어느 시점에서나 해당 시점까지의 모든 트랜잭션들을 포함하는 현재 데이터 스트림에 대한 마이닝 결과를 얻을 수 있도록 지원한다. 개방 데이터 마이닝은 기본적으로 다음과 같은 요구사항을 만족해야 한다[13]. 첫째, 데이터 스트림의 각 구성요소들은 최대 한번 탐색되어야 한다. 둘째, 데이터 스트림 처리 과정에서 메모리 사용량은 무한히 증가되지 않고 한정적으로 유지되어야 한다. 셋째, 새로 발생된 구성요소들은 해당 구성요소를 포함한 전체 데이터 스트림에 대한 분석 결과를 빠른 시간에 얻을 수 있도록 가급적 빨리 처리되어야 한다. 이러한 요구사항들을 만족하기 위해서 데이터 스트림 처리 방법들은 일반적으로 분석 결과에 다소간의 오차를 허용한다.

데이터 스트림은 시간 흐름에 따른 변화의 가능성이 크며, 이러한 변화를 효율적으로 분석하기 위해서는 데이터

† 준 회 원 : 연세대학교 대학원 컴퓨터과학과 박사과정
†† 종신회원 : 연세대학교 컴퓨터과학과 교수
논문접수 : 2004년 11월 30일, 심사완료 : 2005년 4월 11일

스트림을 구성하는 각 구성 요소들의 중요성을 시간 축에 따라 차별화할 필요가 있다. 이를 통해서, 데이터 스트림의 최근 변화를 보다 효율적으로 분석함으로써 해당 데이터 스트림 분석시 보다 가치있는 정보를 얻을 수 있다. 이러한 목적에 부합하기 위해서, 과거에 발생한 중요성이 낮은 정보들의 최신 마이닝 결과에 대한 영향력은 효율적으로 감쇠되어야 한다. 이전의 연구에서 정보의 중요성 차별화 기법으로 *SWF*[2] 알고리즘 및 0 또는 1로 구성되는 데이터 스트림에 대해서 최근 일정 범위 동안의 1의 발생 빈도를 계산하기 위한 방법[3] 등이 제안되었다. 하지만, 이들 방법들은 일정 시점에서 마이닝 결과를 구하기 위해서 분석 대상 범위의 데이터들을 반복적으로 탐색하거나 매우 단순한 데이터 집합을 분석 대상으로 하는 것이다. 따라서 개방 데이터 마이닝 환경에는 적합하지 못하며, 특히 트랜잭션 데이터 스트림에서는 효율적으로 적용되는데 한계가 있다.

본 논문에서는 개방 데이터 마이닝 환경에서 트랜잭션 데이터 스트림을 대상으로 최근 빈발항목(recent frequent itemset) 집합을 탐색하는 이동 윈도우 기법을 제안한다. 본 논문에서 제안하는 방법은 현재 윈도우 범위에 속하는 트랜잭션들에 대해서만 분석 결과를 구함으로써 최근 빈발항목 집합을 구한다. 이때, 앞서 기술한 개방 데이터 마이닝에 대한 기본적인 요구 조건을 만족한다. 먼저, 출현빈도 수 관리 대상 항목의 수는 지연추가 및 항목전지 작업 등의 두 가지 핵심 기술에 의해 최소화되며, 따라서 마이닝 수행과정에서의 메모리 사용량이 매우 적은 정도로 유지된다. 또한, 현재 윈도우 범위에서 벗어나는 트랜잭션들의 영향력은 마이닝 수행과정에서 순차적으로 제거된다. 따라서, 각 시점에서 마이닝 결과를 구하고자 할 때 분석 대상 범위의 데이터들을 재탐색하지 않고 짧은 시간에 결과를 구할 수 있다. 더불어, 분석 대상을 최근 일정 범위의 데이터에 집중함으로써 데이터 스트림에서 나타나는 최근 변화를 효율적으로 감지할 수 있다.

본 논문의 구성은 다음과 같다. 먼저, 제 2장에서는 개방 데이터 마이닝 환경에서 빈발항목 탐색 및 정보의 중요성 차별화와 연관된 관련 연구들을 기술한다. 제 3장에서는 분석 대상이 되는 데이터 스트림을 정의하고, 본 논문에서 제안하는 방법에서 필요한 기본적인 개념을 정리한다. 제 4장에서는 개방 데이터 마이닝 환경에서 이동 윈도우 기법을 적용한 최근 빈발항목 탐색 방법을 상세히 기술하며, 제 5장에서는 논문에서 제안된 방법의 성능을 다양한 실험을 통해서 검증한다. 끝으로, 제 6장에서 논문의 결론을 맺는다.

2. 관련 연구

한정된 트랜잭션 데이터 집합을 대상으로 하는 폐쇄 데이터 마이닝에서 빈발항목(frequent itemset) 탐색 작업은 해당 데이터 집합에 출현한 항목(itemset)들 중에서 해당 항목의 지지도가 사전에 정의된 최소지지도 임계값보다 큰 항목들을 찾는 작업이며, 이들 항목들을 빈발항목이라 지칭한다. 빈발항목 탐색을 위한 가장 대표적인 것은 *Apriori*[6] 알고리즘으로 최대 빈발항목의 차수가 n인 경우(즉, n개의 단위 항목으로 구성되는 경우) 데이터 집합을 $n+1$번 탐색한다. 데이터 집합 탐색 횟수를 줄이기 위한 방법으로 *DIC*[4] 알고리즘 및 *Partition*[7] 알고리즘 등이 제안되었다. *Carma*[8] 알고리즘은 데이터 집합의 각 트랜잭션을 하나씩 차례로 탐색하며 다음과 같은 두 단계로 구성된다. 첫 번째 단계에서는 잠재적으로 빈발항목이 될 수 있고 자신의 모든 부분 항목이 이미 lattice에서 관리되고 있는 항목들이 후보 항목으로서 lattice에 추가한다. 두 번째 단계에서는 lattice에 존재하는 항목들에 대해서 해당 항목이 추가되기 이전의 트랜잭션들을 다시 탐색함으로써 정확한 출현빈도 수를 파악한다. 하지만, 이들 방법은 마이닝 결과를 구하기 위해서 분석 대상 데이터 집합을 반본적으로 탐색함으로 데이터 스트림을 대상으로 하는 개방 데이터 마이닝에는 적합하지 않으며, 또한 각 정보들의 중요성을 서로 동일한 간주함으로써 최근 빈발항목을 구할 수 없다.

최근들어 데이터 스트림을 대상으로 하는 개방 데이터 마이닝 환경에서 다양한 형태의 정보를 탐색하기 위한 방법들[1, 9, 10, 11]이 활발히 제안되고 있다. 이들 방법들 중에서 [1] 및 [9]에서 제안된 방법들이 데이터 스트림에 출현한 구성 요소의 빈발성을 분석하는데 초점을 맞추고 있다. 이들 방법들에서는 해당 방법들의 마이닝 결과 집합 또는 빈발항목으로 구해진 개별 항목의 지지도에 다소간의 오차를 포함한다. *Count Sketch*[9] 알고리즘은 데이터 스트림에서 단위 항목의 빈발성을 분석하는데 초점을 맞추고 있으며, 각 단위항목들의 출현빈도 수를 추정한다. 반면에 *Lossy Counting*[1] 알고리즘은 최대 허용오차 e 및 최소지지도가 주어졌을 때 트랜잭션 데이터 스트림에서 빈발항목 집합을 구하는데 목적을 두고 있다. 하지만, 이들 방법은 데이터 스트림을 구성하는 각 정보들의 중요성을 차별화하지 않고 서로 동일한 것으로 간주한다. 따라서, 해당 데이터 스트림에서는 과거에 발생한 정보들의 영향력이 크게 작용하고 최근 변화를 효율적으로 분석하지 못한다. 시간 변화에 따른 데이트 스트림의 가변성을 고려할 때 데이터 스트림에서 보다 의미있는 정보를 탐색하기 위해서는 시간 변화에 따른 각 정보들의 중요성을 차별화 할 수 있는 새로운 접근 방법을 필요로 한다.

SWF[2] 알고리즘은 폐쇄 데이터 마이닝 환경에서 한정된 데이터 집합에서 최근에 발생한 일정 개수의 트랜잭션들에서 빈번히 발생한 항목들을 탐색하기 위해서 윈도우 기법을 사용한다. 하나의 윈도우는 일련의 분할단위(partition)들로 구성되며, 각 분할단위는 다수의 트랜잭션들로 구성된다. 하나의 트랜잭션에서 발생한 항목들 중에서 빈발인 두개의 단위항목으로 구성되는 항목을 길이 2인 후보항목이라 지칭하며, 현재 윈도우에 발생한 길이 2인 모든 후보항목은 별도로 관리된다. 현재 윈도우 범위가 변경되었을 때, 해당 윈도우 범위를 벗어나는 과거 분할단위는 고려대상에서 제외되며 새로 발생된 트랜잭션들의 집합인 새로운 분할단위가 고려대상에 포함된다. 이와 동시에, 길이가 2인 후보항목 집합이 변화된 윈도우에 맞게 갱신된다. 이러한 갱신된 길이 2

인 후보항목 집합을 바탕으로 가능한 모든 후보 항목들이 조합되며, 변화된 윈도우 범위에 포함되는 모든 트랜잭션들을 재탐색하면서 후보항목들의 출현빈도 수를 계산함으로써 변화된 윈도우 범위에서의 빈발항목 집합을 구할 수 있다. 즉, 현재 윈도우 범위에 속하는 모든 트랜잭션들을 별도로 유지해야 하며, 또한 빈발항목 집합을 구하기 위해서 해당 트랜잭션들 전체를 재탐색해야 한다. 따라서, 해당 방법은 앞서 기술한 개방 데이터 마이닝의 기본적인 요구 조건을 만족하지 못한다.

한편, 0 또는 1로 구성되는 데이터 스트림에서 최근 발생한 N 개의 구성 요소들 중에서 1값을 갖는 구성 요소의 수를 추정하기 위해서 이동 윈도우 개념이 활용된다[3]. 해당 방법에서는 SWF 알고리즘과는 달리 현재 윈도우 범위에 속하는 구성 요소들을 별도로 유지하지 않는다. 하지만, 해당 방법은 0 또는 1로 구성되는 데이터 스트림에는 효과적이나 트랜잭션 데이터 스트림에서 항목들의 출현빈도 수를 관리를 위해 적용하는 것은 불가능하다.

3. 기본 정리

본 논문에서는 개방 데이터 마이닝 환경에서 최소지지도 $S_{min}\in(0,1)$, 중요항목 지지도 $S_{sig}\in(0,S_{min})$ 및 이동 윈도우의 크기 w가 주어졌을 때, 트랜잭션 데이터 스트림에서 최근 빈발항목(recent frequent itemset) 집합을 탐색하는 이동 윈도우 기법을 제시한다. 먼저, [12]에서 기술한 바와 같이 개방 데이터 마이닝을 위한 데이터 스트림 집합은 다음과 같이 정의된다.

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

ii) 2^I가 단위항목 집합 I의 멱집합을 나타낼 때, $e\in(2^I-\{\varnothing\})$을 만족하는 e를 항목(itemset)이라 한다. 항목의 길이 $|e|$는 항목 e를 구성하는 단위항목의 수를 의미하며 임의의 항목 e는 해당 항목의 길이에 따라 $|e|$-항목($|e|$-itemset)이라 정의할 수 있다. 한편, 일반적으로 항목 $\{a, b, c\}$는 간단히 abc로 나타낸다.

iii) k번째 순서로 데이터 집합에 추가되는 트랜잭션(transaction)을 T_k라 나타내며 이는 단위항목 집합 I의 부분집합이다.

iv) 새로운 트랜잭션 T_k가 추가되었을 때 현재의 데이터 집합 D_k는 현재까지 발생하여 추가된 모든 트랜잭션들로 구성된다. 즉, $D_k=<T_1, T_2, \cdots, T_k>$.

더불어 이동 윈도우 기법을 적용하는데 있어서 윈도우의 크기가 w로 표현될 때, 현재 윈도우 D_k^w는 최근에 발생한 w 개의 트랜잭션들로 구성된다. 즉, $D_k^w=<T_{k-(w-1)}, T_{k-(w-2)}, \cdots, T_k>$. 이때, $|D^w|_k$는 현재 윈도우 범위에 포함되는 트랜잭션의 총수를 의미하며, λ는 현재 윈도우 D_k^w 범위에서 가장 오래된 트랜잭션의 트랜잭션 아이디(TID)를 의미한다. 즉, $\lambda=k-(w-1)\,(k>w)$ 또는 $\lambda=1\,(k\le w)$. 하나의 트랜잭션 T_k가 새로 발생되었을 때, 항목 e의 현재 출현빈도 수 $C_k(e)$는 현재 윈도우 범위에 속하는 트랜잭션들 중에서 해당 항목을 포함하는 트랜잭션들의 수를 의미한다. 이때, 현재 윈도우의 트랜잭션의 총 수 $|D^w|_k$ 및 항목 e의 출현빈도 수 $C_k(e)$는 각각 다음과 같이 구해진다.

$$\text{i)}\ |D^w|_k=\begin{cases}k & \text{if } k<w\\ w & \text{otherwise},\end{cases}\quad \text{ii)}\ C_k(e)=\sum_{i=\lambda}^{k}V_i(e)$$

$$\text{where } V_i(e)=\begin{cases}1 & \text{if } e\subseteq T_i\\ 0 & \text{otherwise}\end{cases}$$

더불어, 해당 항목의 현재 지지도 $S_k(e)$는 현재 윈도우의 트랜잭션의 총 수 $|D^w|_k$ 대비 해당 항목의 현재 출현빈도 수 $C_k(e)$의 비율로 구해진다.

데이터 스트림에서 최근 빈발항목 집합을 정확히 구하기 위해서는 해당 데이터 스트림에 출현한 각 항목들의 출현빈도 수가 철저히 관리돼야 한다. 하지만, 해당 데이터 스트림에서 출현한 모든 항목들의 출현빈도 수를 정확히 관리하는 것은 지속적으로 확장되는 데이터 스트림의 특성으로 인해 거의 불가능하다. 만약, 모든 항목들을 관리한다면 메모리 사용량이 급격히 증가될 뿐만 아니라 마이닝 수행 시간도 크게 증가될 것이다. 이때, [12]에서 제안된 바와 같이 가까운 미래에 빈발항목이 될 수 있는 중요한 항목들의 출현빈도수만을 관리함으로써 메모리 사용량을 감소시킬 수 있다. 즉, 데이터 스트림에서 출현한 항목들 중에서 일정 임계값 이상의 지지도를 갖는 항목들을 중요항목이라 정의하고, 이들 중요항목들의 출현빈도 수만 관리한다. 이때, 기준이 되는 임계값을 ***중요항목 지지도 S_{sig}***라 한다. 한편, 출현빈도 수를 관리하는 항목들 중에서 해당 데이터 스트림의 확장으로 인해 지지도가 감소된 항목들을 비중요항목으로 간주할 수 있으며, 이들 비중요항목들의 출현빈도 수는 더 이상 관리하지 않는다. 이러한 두 가지 방법은 데이터 스트림을 대상으로 하는 개방 데이터 마이닝에서 메모리 사용량을 감소시킬 수 있는 핵심 기술들이다.

4. 이동 윈도우 기법을 적용한 최근 빈발항목 탐색

본 장에서는 온라인 데이터 스트림에 대한 개방 데이터 마이닝 환경에서 최근 빈발 항목 탐색을 위한 이동 윈도우 기법을 제안한다. 제안되는 방법에서는 사용자에 의해 정의된 윈도우 크기를 바탕으로 현재 윈도우 범위에 속하는 모든 트랜잭션들은 ***현재 트랜잭션 리스트 CTL***에 의해 관리된다. 윈도우 크기는 시간 단위 또는 관리되는 트랜잭션의 개수로 정의될 수 있다. 시간 단위로 윈도우 크기를 정의하는 경우에는 트랜잭션의 발생 시간을 고려하여 현재 윈도우

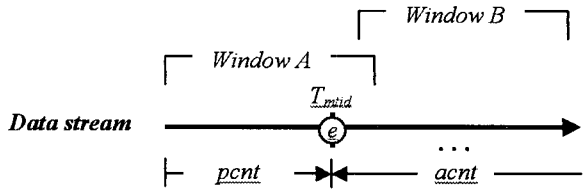
범위에 포함되는 유효 정보 여부가 결정되며, 개수 단위로 윈도우 크기가 정의되는 경우에는 트랜잭션의 발생 순서를 고려하여 현재 윈도우 범위에 포함되는 유효 정보 여부가 결정된다. 본 논문에서는 관리되는 트랜잭션의 개수로서 윈도우 크기를 정의하며, 시간 단위로 정의되는 경우에도 본 논문에서 제안하는 접근 방법을 유사하게 적용할 수 있다. *CTL*에 의해 관리되는 트랜잭션들은 데이터 스트림의 확장에 의해 현재 윈도우 범위가 변경되었을 때, 현재 윈도우 범위에서 벗어난 트랜잭션들의 영향력을 마이닝 결과로부터 제거하는 과정에 활용된다. 논문에서 제안되는 데이터 스트림에서 최근 빈발항목 탐색을 위한 이동 윈도우 기법은 데이터 발생 정도에 따라 다음과 같은 두 가지 상태로 구분된다.

i) **윈도우 초기화 상태(Window initialization state)** : 이 단계는 현재까지 발생된 트랜잭션의 수가 윈도우 크기 w보다 작은 경우이다. 따라서, 새로 발생되는 트랜잭션들이 현재 트랜잭션 리스트 *CTL*에 추가되기만 하며, 현재 윈도우 범위에 포함되는 트랜잭션의 수가 계속 증가된다.

ii) **윈도우 이동 상태(Window sliding state)** : 이 단계는 *CTL*이 완전 포화 상태일 때 시작된다. 새로 발생되는 트랜잭션이 *CTL*에 추가되며 가장 오래된 트랜잭션은 *CTL*로부터 제거된다.

논문에서 제안되는 방법에서는 현재 윈도우 내의 트랜잭션들에 출현한 모든 중요항목들은 ***모니터링 트리(Monitoring tree)***라 불리는 전위트리(prefix tree)[4, 5] 구조에 의해 관리된다. 해당 모니터링 트리의 각 노드들은 하나의 단위항목(item)을 포함하며, 해당 모니터링 트리의 루트 노드(root node)로부터 해당 노드에 이르는 경로(path)상에 존재하는 모든 노드들에 포함된 단위항목들로 구성되는 항목(itemset)을 표현한다. 또한, 각 노드는 해당 노드에 연관된 항목 e의 출현빈도 수 관리를 위해서(*pcnt, acnt, mtid*) 정보를 관리한다. 해당 항목 e의 최대 추정 출현빈도 수 *pcnt*는 해당 항목이 모니터링 트리에 추가되기 이전에 발생하고 현재 윈도우 범위에 속하는 트랜잭션들 중에서 해당 항목 e를 포함하는 트랜잭션들의 수를 나타낸다. 해당 항목의 실측 출현빈도 수 *acnt*는 해당 항목이 모니터링 트리에 추가된 이후에 발생하고 현재 윈도우 범위에 속하는 트랜잭션들 중에서 해당 항목 e를 포함하는 트랜잭션들의 수를 나타낸다. 끝으로, *mtid*는 해당 항목이 모니터링 트리에 추가되는 시점의 트랜잭션 아이디 *TID*를 나타낸다.

모니터링 트리에서 관리되는 중요항목은 서로 배타적인 두 그룹으로 구분될 수 있다. 하나는 (그림 1)의 '*Window A*'와 같이 현재 윈도우 범위 내에서 모니터링 트리에 추가된 항목들의 집합이다. 즉, 항목들의 *mtid* 값이 보다 큰 항목들의 집합이다. 이러한 그룹에 속하는 항목들은 현재 윈도우 범위에서 각 항목의 정확한 출현빈도 수 $C_k(e)$를 구할 수 없으며, 해당 항목의 추정 출현빈도 수를 모니터링 트리



(그림 1) 모니터링 트리에서 관리되는 항목 e의 각 세부 정보의 의미

의 연관된 노드에서 관리되는 해당 항목의 정보 (*pcnt, acnt, mtid*)로부터 다음과 같이 구할 수 있다.

$$\hat{C}_k(e)=pcnt+acnt \ (mtid > \lambda)$$

이때, *pcnt*는 해당 항목이 모니터링 트리에 추가되기 이전에 발생하고 현재 윈도우 범위 속하는 트랜잭션들에서 가질 수 있는 최대 출현빈도 수를 의미하므로 해당 항목의 추정 출현빈도 수 $\hat{C}_k(e)$는 해당 항목의 실제 출현빈도 수 $C_k(e)$보다 크거나 같은 값을 갖는다. 다른 하나의 그룹은 (그림 1)의 '*Window B*'와 같이 현재 윈도우 시작 이전에 모니터링 트리에 추가된 항목들의 집합이다. 즉, 항목들의 *mtid* 값이 λ보다 작은 항목들의 집합이다. 이 경우에는 각 항목들의 정확한 출현빈도 수 $C_k(e)$가 모니터링 트리의 연관된 노드상의 해당 항목의 정보 (*pcnt, acnt, mtid*)에서 관리되며 다음과 같이 구할 수 있다.

$$C_k(e)=acnt \ (mtid \le \lambda)$$

개방데이터 마이닝 환경에서 빈발항목 탐색을 위한 이동 윈도우 기법은 트랜잭션 확장 단계 (제 1단계), 출현빈도 수 갱신 단계 (제 2단계), 항목 추가 단계 (제 3단계), 트랜잭션 추출 단계 (제 4단계) 및 빈발항목 탐색 단계 (제 5단계)와 같은 다섯 단계로 구성된다. 각 단계의 기능은 각각 다음과 같다.

4.1 트랜잭션 확장 단계 및 출현빈도 수 갱신 단계

트랜잭션 확장 단계(그림 2의 3-5줄)에서는 현재 데이터 스트림에서 새로운 트랜잭션 T_k가 발생되었을 때 해당 트랜잭션이 *CTL*에 추가된다. 이때, 윈도우 초기화 상태에서는 현재 윈도우 범위에 속하는 트랜잭션의 총 수가 1만큼 증가된다.

이어서 출현빈도 수 갱신 단계(6-10줄)에서는 모니터링 트리에 관리되는 항목들 중에서 새로운 트랜잭션 T_k에 출현한 항목들의 출현빈도 수가 갱신된다. 즉, 새로운 트랜잭션을 구성하는 단위항목들에 생성되는 모니터링 트리상의 모든 연관된 경로가 순차적으로 탐색된다. 탐색 과정에서 해당 경로상에 위치한 하나의 노드에 대해서, 해당 노드에 연관된 항목 정보 (*pcnt, acnt, mtid*)는 다음과 같이 새로운 값으로 갱신된다.

```
S_min : A minimum support,  S_sig : A significant support
ML : A monitoring tree,     CTL : The current transaction list

1:    ML = ∅ ;
2:    for each new transaction T_k in D_k {

      // Transaction appending phase
3:      CTL_k = CTL_k-1 ∪ T_k;
4:      if (the current window is in the window initialization state)
5:        |D^w|_k = |D^w|_k-1+1;

      // Count updating phase
6:      for all itemset e s.t. e∈(2^T_k-{∅}) and e∈ML {
7:        ML.e.acnt=ML.e.acnt+1;
8:        if ML.e.acnt ≤ C_k^prn(e) and |e|>1    // Pruning
9:              Eliminate e and it's descendent nodes from ML;
10:     }

      // Itemset insertion phase
11:     T̄_k = ∅;
12:     for all itemset e s.t. e∈(2^T_k-{∅}) and |e|=1  {
13:       if e∉ML  {
14:           Insert e into ML;  ML.e.pcnt=0; ML.e.acnt=1; ML.e.mtid=k;
15:       }
16:       else if S_k(e)≥S_sig    { T̄_k=T̄_k∪{e}; }
17:   }
18:   for all itemset ē s.t. ē∈(2^T̄_k-{∅}) and e∈ML and |ē|=|e|+1 and e⊂ē {
19:     Estimate  C_k^max(ē) and C_k^min(ē);
20:     if C_k^max(ē)>C_k^upper(ē)  { C_k^max(ē)=C_k^upper(ē); }
21:     if C_k^min(ē)=0          { C_k^min(ē)=1; }
22:     if (C_k^max(ē)/|D^w|_k)≥S_sig  {
23:       Insert ē into ML;
24:       ML.ē.pcnt=C_k^max(ē)-1;  ML.ē.acnt=1;  ML.ē.mtid=k;
25:     }
26:   }

      // Transaction extracting phase
27:     read the oldest transaction T_old in the CTL;
28:     for all itemset e s.t. e∈(2^T_old-{∅}) and e∈ML {
29:       if ML.e.mtid ≤ λ
30:         ML.e.acnt=ML.e.acnt-1;
31:       if ML.e.acnt ≤ C_k^prn(e) and |e|>1    // Pruning
32:         Eliminate e and it's descendent nodes from ML;
33:     }
34:     Eliminate T_old from the CTL;
35:   }
```

(그림 2) 개방 데이터 마이닝 환경에서 빈발항목 탐색을 위한 이동 윈도우 기법

(*pcnt, acnt+1, mtid*)

현재 윈도우가 D_k^w가 윈도우 이동 상태일 때 항목의 *mtid*가 λ보다 작은 경우, 앞서 기술한 바와 같이 현재 윈도우 범위에서 해당 항목의 실제 출현빈도 수는 정확한 값을 구할 수 있으며 *acnt*로 표현된다. 따라서, 만약 갱신된 *acnt* 값이 $\lceil w \times S_{sig} \rceil$ 보다 작다면 해당 항목은 현재 윈도우 범위에서 비중요항목으로 간주될 수 있으며, 모니터링 트리로부터 전지할 수 있다. 반면에, 항목의 *mtid*가 λ보다 큰 경우에는 현재 윈도 범위에 속하는 *w*개의 트랜잭션들 중에서 단지 *w*-(*mtid*-λ)개의 트랜잭션들이 해당 항목의 *acnt* 값을 구하기 위해서 탐색되었다. 따라서, 만약 갱신된 *acnt* 값이 $\lceil \{w-(\text{mtid}-\lambda)\} \times S_{sig} \rceil$ 보다 작다면 해당 항목은 현재 윈도우 범위에서 비중요항목으로 간주될 수 있으며 모니터링 트리로부터 전지될 수 있다. 항목의 전지 여부를 판단하기 위해서 해당 항목의 *acnt* 값만을 이용함으로써 실제 탐색된 트랜잭션들의 출현 정보를 이용하여 전지 여부를 결정한다. 이 경우, 보다 엄격한 조건에서 항목의 전지 여부를 판단할 수 있다. 즉, 전지되는 모든 항목은 비중요항목임을 보장할 수 있으며, 중요항목이 전지되는 경우를 방지할 수 있다. 이러한 접근 방법을 바탕으로, 현재 윈도우가 D_k^w가 윈도우 이동 상태일 때 항목 정보 (*pcnt, acnt, mtid*)를 갖는 항목의 전지 여부를 판단하기 위한 항목 전지 임계값 $C_k^{prn}(e)$는 다음과 같이 구해진다.

$$C_k^{prn}(e) = \begin{cases} \lceil w \times S_{sig} \rceil & \text{if } mtid \le \lambda \\ \lceil \{w-(mtid-\lambda)\} \times S_{sig} \rceil & \text{otherwise} \end{cases} \quad (1)$$

만약 현재 윈도우가 윈도우 초기화 상태라면 현재의 윈도우의 크기는 k라 간주할 수 있으며 모니터링 트리에 관리되는 항목 e의 $mtid$ 값이 항상 λ보다 크다. 따라서, 윈도우 초기화 단계에서 항목 전지 임계값은 다음과 같이 구할 수 있다.

$$C_k^{prn}(e) = \lceil \{k-(mtid-1)\} \times S_{sig} \rceil \quad (2)$$

새로 발생된 트랜잭션에 의해 모니터링 트리의 연관된 노드를 방문하는데 있어서, 항목 e의 $acnt$ 값이 식 (1) 및 (2)에서 구해진 해당 항목의 전지 임계값 $C_k^{prn}(e)$보다 작으면 해당 항목은 비중요항목으로 간주될 수 있다. 따라서, 모니터링 트리상에서 해당 항목에 연관된 노드는 전지된다. 항목 전지 과정은 전위트리 구조에 기반한 기존의 일반적인 마이닝 방법[4,5]에서의 항목 전지 과정과 동일하다. 그러나, 만약 길이가 1인 1-항목(즉, 단위항목)이 전지된다면, 해당 항목이 추후에 다시 발생되어 이를 모니터링 트리에 추가할 때, 해당 항목의 출현빈도 수를 추정할 수 없다. 따라서, 모니터링 트리에 관리되는 1-항목은 $acnt$ 값이 전지 임계값보다 낮은 경우에도 전지되지 않는다. 이러한 일련의 과정을 ***항목 전지(pruning)*** 작업이라 한다. 현재 시점에서 전지된 항목들도 미래에 발생되는 트랜잭션들에서 빈번히 출현하여 높은 지지도를 갖는 것으로 추정될 경우 모니터링 트리에 추가될 수 있다.

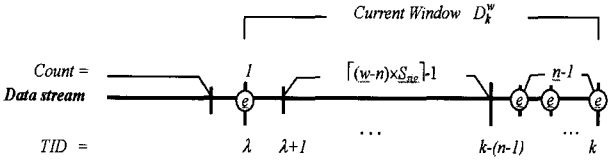
한편, 만약 윈도우 크기가 $1/S_{sig}$보다 작거나 같은 값으로 설정된다면, 현재 윈도우 범위에 속하는 트랜잭션들에서 단 한번 출현한 항목들의 지지도가 중요항목 지지도 S_{sig}보다 크거나 같은 값을 갖게 된다. 따라서, 현재 윈도우 범위의 트랜잭션들에 출현한 모든 항목들이 중요항목으로 간주되어 모니터링 트리에서 관리되어야 하며, 이러한 경우 모니터링 트리에서 관리되는 항목의 수가 지속적으로 증가될 수 있다. 그러므로 이러한 상황을 방지하기 위해서는 $w>(1/S_{sig})$ 관계를 항상 만족해야 한다.

4.2 항목 추가 단계

새로운 트랜잭션 T_k에 의해 유추되는 모든 항목들의 출현빈도 수 갱신 작업 및 항목 전지 작업이 종료되면 해당 트랜잭션에서 출현한 새로운 항목을 모니터링 트리에 추가하기 위한 항목 추가 단계(11-26줄)가 수행된다. 먼저, 해당 트랜잭션 T_k에 출현한 비중요 단위항목들이 제거된다. 왜냐하면 비중요 단위항목을 포함하는 항목은 해당 항목의 출현빈도 수 추정값이 중요항목 지지도 S_{sig} 보다 큰 값을 가질 수가 없기 때문이다 (즉, 중요항목이 될 수 없기 때문이다). 이를 위해서 해당 트랜잭션 T_k에 출현한 모든 단위항목들에 대해서 모니터링 트리상에서 해당 단위항목에 연관된 노드를 탐색하여 각 단위항목의 출현빈도 수를 검사한다. 만약 해당 단위항목의 지지도가 S_{sig}보다 작은 경우 해당 항목은 비중요 단위항목으로 간주되며, 항목 추가 과정에서 더 이상 고려되지 않는다. 이와 동시에 해당 트랜잭션 T_k에 출현한 단위항목들 중에서 모니터링 트리에 관리되지 않는 새로운 단위항목은 모니터링 트리에 추가된다. 이때, 단위항목의 경우는 별도의 출현빈도 수 추정을 필요로 하지 않는다. 따라서, 모니터링 트리에 관리되는 모든 단위항목의 현재 윈도우 범위에서의 출현빈도 수는 오차를 포함하지 않는 정확한 값을 갖는다. 새로 추가되는 단위항목에 연관된 노드의 항목 정보($pcnt$, $acnt$, $mtid$)는 다음과 같이 초기화된다.

$$pcnt_k = 0, \quad acnt_k = 1, \quad mtid = k$$

해당 트랜잭션에서 모든 비중요 단위항목들을 제거한 후 정제된 트랜잭션 $\overline{T}_k$를 얻을 수 있다. 이어서, 해당 트랜잭션에 출현한 새로운 중요항목들을 찾기 위해서 정제된 트랜잭션 $\overline{T}_k$를 이용하여 출현빈도 수 갱신 단계에서와 유사한 방법으로 모니터링 트리가 재탐색된다. 새로운 중요항목을 찾기 위한 이러한 과정에서 전위트리의 특성으로 인해 후보항목 생성을 위한 별도의 추가 작업 등을 필요로 하지 않는다. 즉, 정제된 트랜잭션을 이용한 모니터링 트리 탐색 과정에서 각 필요한 항목들의 생성 여부를 순차적으로 판단할 수 있다. 한편, 하나의 새로운 항목이 중요항목이 되기 위해서는 해당 항목의 모든 부분항목들이 현재 윈도우 범위에서 중요항목으로 모니터링 트리에서 관리되고 있어야 한다. 만약, 해당 항목의 부분항목들 중에서 단 하나라도 중요항목이 아닌 경우에는 해당 항목의 추정 지지도가 중요항목 지지도보다 낮은 값으로 구해지며, 특히 해당 부분항목이 모니터링 트리에서 관리되고 있지 않는 경우에는 $C_k^{max}(e)$가 0으로 추정되기 때문이다. 모니터링 트리에서 길이가 υ인 υ-항목 e ($\upsilon \ge 1$)에 연관된 노드가 탐색되었을 때, $\upsilon+1$ 개의 단위항목(즉, 해당 항목 e를 구성하는 모든 단위항목과 정제된 트랜잭션 $\overline{T}_k$에 존재하는 하나의 단위항목)으로 구성되는 모든 항목들의 출현빈도 수가 [12]에서 제안한 출현빈도 수 추정 방법에 의해 추정된다. 해당 $(\upsilon+1)$-항목의 추정 출현빈도 수가 중요항목 지지도 S_{sig}보다 크거나 같으면 해당 항목은 새로운 중요항목으로 간주되며, 이는 모니터링 트리에 추가된다. 이러한 과정을 ***지연추가(delayed-insertion)*** 작업이라 한다.



(그림 3) 새로 추가되는 항목 e의 출현빈도 수 상한값

길이가 $n(n\geq 2)$인 새로운 중요항목 e가 윈도우 이동 상태에서 식별되었을 때, 현재 윈도우 범위에서 해당 n-항목의 출변빈도 수 상한값 을 구할 수 있다. 다시 말해서, 해당 n-항목의 모든 부분항목들 및 해당 n-항목 e 자신을 모니터링 트리에 추가하기 위해서는 현재 윈도우 범위에 속하는 w 개의 트랜잭션들 중에서 n개의 트랜잭션들이 해당 n-항목 e를 포함해야 한다. 이때, 해당 n-항목 e의 출현빈도 수는 상대적으로 가장 큰 출현빈도 수를 갖는 상황을 찾을 수 있다. 즉, (그림 3)에서와 같이 해당 n 개의 트랜잭션들 중에서 하나의 트랜잭션은 현재 윈도우 범위의 첫번째 트랜잭션이고 나머지 $n-1$ 개의 트랜잭션은 가장 최근에 연속적으로 발생된 경우에 해당 n 개의 트랜잭션에 출현한 항목들의 출현빈도 수는 최대화된다. 이때, 트랜잭션 $T_{\lambda+1}$부터 $T_{k-(n-1)}$까지의 범위에서 해당 n-항목 e의 최대 가능 출현빈도 수는 $\lceil S_{sig}\times(w-n)\rceil$ 보다 작아야 한다. 왜냐하면 해당 n-항목은 현재 트랜잭션 T_k에서 새로 모니터링 트리에 추가돼야 하므로 해당 구간에서는 해당 n-항목의 지지도는 중요항목 지지도 S_{sig}보다 작아야 하기 때문이다. 만약 해당 n-항목의 지지도가 S_{sig}보다 크다면, 해당 n-항목은 현재 트랜잭션 이전 시점에서 모니터링 트리에 추가됐을 것이다. 이러한 사실에 바탕하여 해당 n-항목의 출현빈도 수 상한값 $C_k^{upper}(e)$는 다음과 같이 구해진다.

$$C_k^{upper}(e) = \lceil (w-n)\times S_{sig}\rceil - 1 + n \qquad (3)$$

한편, 현재 윈도우가 윈도우 초기화 상태라면 (즉, $k\leq w$), 현재 윈도우 범위에서 트랜잭션의 총 수가 k이므로 식 (3)에서 구해진 출현빈도 수 상한값 $C_k^{upper}(e)$는 다음과 같이 수정된다.

$$C_k^{upper}(e) = \lceil (k-n)\times S_{sig}\rceil - 1 + n \qquad (4)$$

중요항목 여부를 판단하기 위한 n-항목 e에 대해서, [12]에서 제안된 출현빈도 수 추정 방법에 의해 추정된 해당 n-항목의 최대 출현빈도 수 $C_k^{max}(e)$가 $C_k^{upper}(e)$ 보다 크다면, $C_k^{upper}(e)$가 해당 n-항목의 출현빈도 수 추정 값으로 활용된다. 이를 통해서 출현빈도 수 추정 과정에 따르는 오차를 최소화 하고 보다 정확한 값으로 구할 수 있다. 한편, 해당 n-항목이 현재 트랜잭션 T_k에서 출현한 정보는 실제 모니터링된 것으로 해당 n-항목에 연관된 노드의 항목 정보들 중에서 *acnt*에 의해서 관리된다. 따라서, 해당 n-항목의 추정 출현빈도 수 $min(C_k^{max}(e), C_k^{upper}(e))$ 값이 해당 n-항목에 연관된 노드의 항목 정보들 중에서 *pcnt*에 할당될 때 1만큼 감소돼야 한다. 결론적으로, 해당 n-항목이 모니터링 트리에 추가될 때 해당 n-항목에 연관된 노드의 항목 정보(*pcnt*, *acnt*, *mtid*)는 다음과 같이 초기화된다.

$$pcnt_k = min(C_k^{max}(e), C_k^{upper}(e)) - 1,\ acnt_k = 1,\ mtid = k$$

한편, 모니터링 트리에서 관리되는 n-항목($n\geq 2$)의 *pnct* 값에는 출현빈도 수 추정 과정에서 유발되는 일정 정도의 오차를 포함한다. 하지만, 해당 *pcnt* 값은 $min(C_k^{max}(e), C_k^{upper}(e))-1$로 설정되며, 이는 해당 *pcnt*를 추정하는 범위에서 해당 항목의 실제 출현빈도 수보다 크거나 같은 값이다. 따라서, 논문에서 제안된 방법에서는 출현빈도 수 추정 방법을 적용하더라도 각 개별 항목의 출현빈도 수는 해당 개별 항목의 실제 출현빈도 수보다 크거나 같은 값을 갖는다. 또한, 하나의 데이터 스트림에 대한 마이닝 결과로 구해진 최근 빈발항목 집합에는 *부정적인 오차(negative error)*가 존재하지 않는다. 즉, 하나의 데이터 스트림에 대한 최근 빈발항목 탐색 과정에서 매 시점의 현재 윈도우 범위에서 실제 빈발항목인 모든 항목들은 본 논문에서 제안하는 방법에서도 정확히 구할 수 있다.

4.3 트랜잭션 추출 단계

트랜잭션 추출 단계(27-34줄)는 현재 윈도우가 윈도우 이동 상태일 경우에만 수행된다. 이 단계에서는 *CTL*상의 가장 오래된 트랜잭션이 제거된다. 또한, 모니터링 트리에서 관리되는 각 항목들의 출현빈도 수에 있어서 제거되는 트랜잭션의 영향력을 소거하기 위해서 제거되는 트랜잭션을 이용하여 출현빈도 수 갱신 단계에서 기술한 것과 유사한 방법으로 모니터링 트리를 탐색한다. 이러한 과정에서 탐색되는 각 노드에 대해서, 해당 노드의 *mtid* 값이 λ(즉, 현재 윈도우의 시작 트랜잭션의 *TID*)보다 작거나 같은 경우 해당 노드의 *acnt* 값을 1만큼 감소시킨다. 만약 갱신된 *acnt* 값이 식 (1)과 같이 구해지는 해당 노드의 전지 임계값 $C_k^{prn}(e)$ 보다 작으면, 해당 노드(즉, 이와 연관된 항목)는 모니터링 트리로부터 전지된다. 길이가 1인 1-항목에 연관된 노드에 대해서는, 해당 노드의 *acnt*가 0인 경우에만 해당 노드가 전지된다. 반면, 탐색된 노드의 *mtid* 값이 λ보다 크거나 같다면, 해당 노드의 *acnt* 값은 변경되지 않는다. 왜냐하면, 이 경우에는 탐색된 노드에 연관되는 항목이 현재 시점에서 추출되는 트랜잭션에 출현한 상황은 실제로 모니터링돼서 *acnt* 값에 반영된 것이 아니기 때문이다. 이때, 해당 노드의 *pcnt* 값 또한 변경되지 않는다. 만약 현재 윈도우 크기가 매우 크게 설정되어 *CTL*에 유지되어야 할 트랜잭션의 수가 많다면, *CTL* 구조는 보조 기억장치에 유지될 수 있다. 논문에서 제안하는 방법에서는 *CTL*에 대한 접근이 *CTL*의 맨 처음이나 맨 끝에 위치한 하나의 트랜잭션만을 접근하기 때문에 *CTL*을 보조 기억장치에 유지하는 경우에도 수행 시간에 크게 영향을 미치지 않는다.

4.4 빈발항목 탐색 단계

빈발항목 탐색 단계는 현재 윈도우 범위에 속하는 트랜잭션들에 대한 마이닝 결과를 필요로 할 때 수행된다. 본 단계에서는 전위트리 구조에 기반한 기존의 마이닝 방법[4, 5]에서와 동일한 방법으로 모니터링 모니터링 트리를 탐색하

여 최신 빈발항목 집합을 구한다. 모니터링 트리의 연관된 노드에서 항목 정보 (*pcnt, acnt, mtid*)를 갖는 하나의 노드 e에 대해서, 만약 *mtid* 값이 λ보다 작거나 같다면, 현재 윈도우 범위에서 해당 항목의 실제 출현빈도 수 $C_k(e)$는 해당 *acnt*와 일치한다. 따라서, 이 경우에는 빈발항목 탐색 단계에서 항목의 지지도를 계산하는데 있어서 *acnt*만을 활용한다. 즉, $acnt/|D^w|_k$ 이 S_{min}보다 크다면 해당 항목은 최근 빈발항목으로 간주된다. 이 경우에 구해진 각 최근 빈발항목의 지지도 오차는 0이다. 반면 *mtid* 값이 λ보다 크다면, 빈발항목 탐색 단계에서 항목의 지지도를 계산하는데 있어서 *acnt* 뿐만 나이라 *pcnt* 정보를 활용해야 한다. 한편, *pcnt* 값은 해당 항목이 모니터링 트리에 추가될 때 구해진 값으로서, 데이터 스트림의 확장에 따른 윈도우 이동으로 인해 값을 현재 윈도우 상황에 맞도록 재계산해야 한다. 즉, 현재 윈도우를 기준으로 트랜잭션 T_λ부터 T_{mitd}까지의 범위에 속하는 트랜잭션에서 해당 항목이 동일한 확률로 출현했다고 가정하고 중요항목 지지도를 고려하야 다음과 같이 재계산한다.

$$pcnt = \lceil (mtid - \lambda) \times S_{sig} \rceil - 1 \qquad (\ mitd > \lambda)$$

이때, 해당 항목의 최대 가능 출현빈도 수$(anct+pcnt)/|D_w|_k$이 S_{min} 보다 크면 해당 항목은 최근 빈발항목으로 간주된다.

4.5 강제 전지 작업

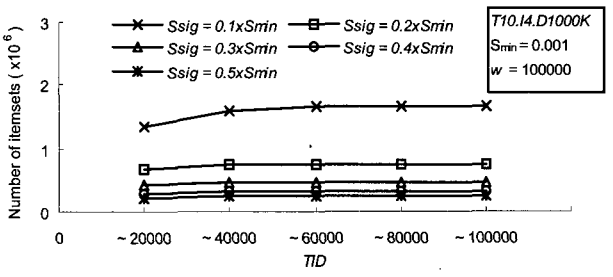
모니터링 트리에서 유지되는 모든 비중요항목들은 해당 모니터링 트리를 전체적으로 한번 탐색함으로써 전지할 수 있다. 이러한 과정은 ***강제 전지(force-pruning)*** 작업이라 지칭한다. 하지만, 강제 전지 작업에서는 모니터링 트리 전체가 탐색되어야 하므로 수행시간이 증가될 수 있다. 따라서, 마이닝 수행 과정에서의 메모리 사용량이 사전에 정의된 가용 메모리 공간을 초과한 경우에 수행하거나 또는 일정 주기마다 수행한다.

5. 실험 결과

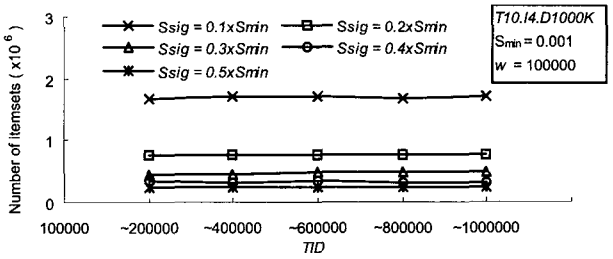
본 장에서는 두 가지 데이터 집합 *T10.I4.D1000K* 및 *T5.I4.D1000K-AB* 을 이용하여 제안된 방법의 성능을 평가한다. [6]에서와 같은 데이터 집합 명명법에 따라, 각 데이터 집합에서 세 가지 숫자는 각각 평균 트랜잭션 길이(T), 빈발항목 평균 길이(I) 및 트랜잭션의 총 수(D)를 의미한다. 데이터 집합 *T10.I4.D1000K* 생성에 활용된 단위항목의 수는 1,000개이며, [6]에서 제시된 데이터 생성 방법에 의해 생성되었다. 데이터 집합 *T5.I4.D1000K-AB*는 두 개의 연속적인 부분 데이터 집합 *TA* 및 *TB*로 나눠진다. *TA*는 단위항목 집합 A로부터 생성된 트랜잭션들의 집합을 의미하며, *TB*는 단위항목 집합 B로부터 생성된 트랜잭션들의 집합을 의미한다. 각 부분 데이터 집합은 500,000 개의 트랜잭션으로 구성되며 [6]에서 제시된 데이터 생성 방법에 의해 생성되었다. 각 부분 데이터 집합 생성에 활용된 단위항목의 수는 1,000개이다. 모든 실험에서 각 데이터 집합의 트랜잭션들은 온라인 데이터 스트림 형태를 구현하기 위해서 하나씩 차례로 접근된다. 모든 실험들은 512MB 메인 메모리를 가진 1.8GH 펜티엄 컴퓨터에서 실험되었으며 C언어로 구현되었다.

(그림 4) 및 (그림 5)는 데이터 집합 *T10.I4.D1000K*에 대해서 제안된 방법의 두 가지 상태(즉, 윈도우 초기화 상태 및 윈도우 이동 상태)에서의 메모리 사용량 변화를 보여준다. 최소지지도 S_{min} 및 윈도우의 크기 w는 각각 0.001 및 100,000으로 설정되었으며, 강제 전지 작업은 매번 1,000개의 트랜잭션이 처리될 때마다 수행되었다. 각 실험에서 일련의 트랜잭션들은 같은 수의 트랜잭션을 갖는 5개의 구간으로 구분된다. 메모리 사용량은 각 구간에서 모니터링 트리에 유지되는 항목 개수의 최대값으로 나타낸다. 윈도우 초기화 상태에서는 새로운 트랜잭션을 처리함에 따라 메모리 사용량이 다소 증가된다. 하지만, 윈도우 이동 상태에서는 새로운 트랜잭션이 발생되더라도 메모리 사용량이 거의 변화 없이 동일한 정도로 유지된다. 두 가지 상태 모두에서 S_{sig} 값이 증가되는 경우 메모리 사용량은 감소된다.

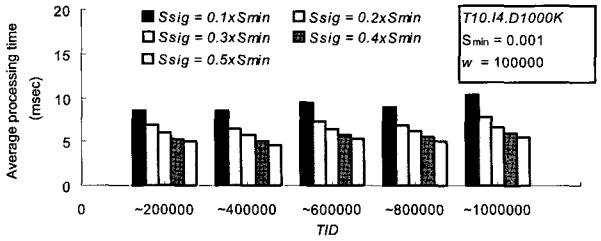
(그림 6)는 앞서와 같은 실험에서 각 구간별 평균 수행 시간을 보여준다. 각 구간은 같은 수의 트랜잭션을 갖는 5개의 구간으로 구분되며, 수행 시간은 하나의 트랜잭션이 생성된 시점부터 제 4단계까지 수행하는데 필요한 시간을 의미한다. (그림 6)에서 알 수 있듯이, 평균 수행 시간은 10 msec 이하로 유지된다. 평균 수행 시간은 지연 추가를 위해서 출현빈도 수를 추정해야 할 항목들의 개수뿐만 아니라 모니터링 트리의 크기에 영향을 받는다. (그림 5)에서 보듯



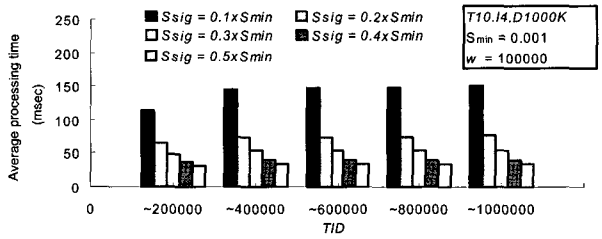
(그림 4) 윈도우 초기화 상태의 메모리 사용량



(그림 5) 윈도우 이동 상태의 메모리 사용량



(그림 6) 1 - 4단계의 평균 수행 시간



(그림 7) 5 단계의 평균 수행 시간

이 모니터링 트리의 크기는 S_{sig} 값에 반비례한다. 따라서, S_{sig} 값이 증가됨에 따라 평균 수행 시간은 감소된다. (그림 7)은 제안된 방법의 빈발항목 탐색 단계(제 5단계)의 평균 수행 시간을 (그림 6)에서와 동일하게 구분된 구간별로 보여준다. (그림 6)의 결과와 비교했을 때, 빈발항목 탐색 단계에서는 모니터링 트리의 상당 부분을 탐색해야 하므로 빈발항목 탐색 단계의 평균 수행 시간이 상대적으로 크다. 그러나, 빈발항목 탐색 단계의 수행 시간도 S_{sig} 값에 반비례한다.

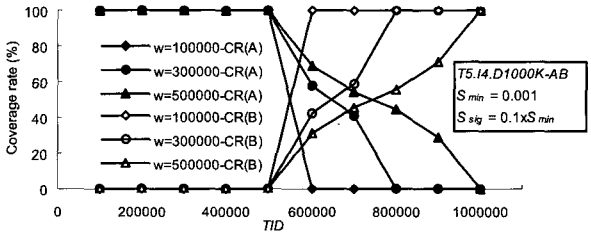
제안된 방법의 마이닝 결과의 정확도를 상대적으로 평가하기 위해서 ***평균 지지도 오차(ASE : average support error)*** 개념을 활용한다. 동일한 데이터 집합에 대한 서로 다른 두 개의 마이닝 결과 집합 $R_1=\{(e_i,\ S'_k(e_i)) \mid S'_k(e_i) \geq S_{min}\}$ 및 $R_2=\{(e_j,\ S''_k(e_j)) \mid S''_k(e_j) \geq S_{min}\}$에 대해서, 마이닝 결과 집합 R_1에 대한 마이닝 결과 집합 R_2의 평균 지지도 오차 $ASE(R_2|R_1)$는 다음과 같이 정의된다.

$$ASE(R_2|R_1) = \frac{\sum_{e_l \in R_1 - R_1 \cap R_2} S'_k(e_l) + \sum_{e_l \in R_1 \cap R_2} \{|S''_k(e_l) - S'_k(e_l)|\} + \sum_{e_l \in R_2 - R_1 \cap R_2} S''_k(e_l)}{|R_1|}$$

여기서, $|R_1|$는 마이닝 결과 집합 R_1에 속하는 항목의 수를 나타낸다. $ASE(R_2|R_1)$ 값이 작을수록 마이닝 결과 집합 R_2의 마이닝 결과 집합 R_1과 유사해짐을 의미한다. <표 1>에서는 데이터 집합 *T10.I4.D1000K*에 대한 제안된 방법의 평균 지지도 오차를 S_{sig} 값을 변화시켜 실험하였다. 실험에서는 각 시점에서 구해진 제안된 방법의 현재 윈도우 범위에 대한 마이닝 결과를 동일한 트랜잭션들에 대한 Apriori [6] 알고리즘의 마이닝 결과와 비교하여 평균 지지도 오차를 계산하였다. 또한, 실험에서는 100,000 개의 트랜잭션이 새로

<표 1> 데이터 집합 *T10.I4.D1000K*에서 마이닝 결과 정확도 (S_{min}=0.001, w=100,000)

S_{sig} ($\times S_{min}$)	ASE($\times 10^{-5}$)									
0.1	3.070	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.3	2.638	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.5	2.310	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.7	2.809	0.098	0.102	0.060	0.025	0.058	0.065	0.088	0.104	0.082
0.9	9.041	5.885	7.929	7.254	8.943	6.436	6.456	6.493	5.631	6.250
	100000	200000	300000	400000	500000	600000	700000	800000	900000	1000000
	TID									



(그림 8) 데이터 집합 *T5.I4.D1000K-AB*에 대한 변화 감지율

이 처리될 때마다 마이닝 결과를 구하고 평균 지지도 오차를 계산하였다. 본 실험에 ASE 값이 작을수록 논문에서 제안된 방법으로 구해진 마이닝 결과 집합이 Apriori 알고리즘에 의해 구해진 마이닝 결과 집합에 가까워짐을 의미한다. Apriori 알고리즘은 한정적인 데이터 집합을 대상으로 오차를 포함하지 않는 정확한 마이닝 결과를 구하는 방법으로서 제안된 방법에 의해 구해진 마이닝 결과가 이에 가까워진다는 것은 마이닝 결과의 정확도가 높아짐을 의미한다. <표 1>에서 알 수 있듯이 S_{sig} 값이 증가함에 따라 평균 지지도 오차는 다소 증가된다. 하지만, 전반적으로 평균 지지도 오차는 매우 작은 정도로 유지된다.

(그림 8)은 데이터 스트림에 내재된 정보의 변화에 대한 제안된 방법의 적응성을 보여준다. 본 실험에서는 데이터 집합 *T5.I4.D1000K-AB*이 사용되었으며, 강제 전지 작업은 매번 1,000개의 트랜잭션이 처리될 때마다 수행되었다. S_{min} 및 S_{sig} 값은 각각 0.001 및 $0.1\times S_{min}$로 설정되었다. 데이터 스트림의 변화에 대해 제안된 방법의 적응성을 평가하기 위해서 변화 감지율을 도입하였다. 상호 배타적인 두 개의 단위항목 집합 X_i 및 X_j ($X_i \cap X_j = \varnothing$)에 대해서, 단위항목 집합 X_i의 ***변화 감지율 $CR(X_i)$***은 해당 단위항목 집합 X_i로부터 유도된 최근 빈발항목의 비율을 나타내며 다음과 같이 구해진다.

$$CR(X_i) = \{\ |F(X_i)| \ /\ |F(X_i \cup X_j)|\ \} \times 100\ (\%)$$

여기서, $|F(X_i)|$은 단위항목 집합 X_i로부터 유도된 최근 빈발항목들의 수를 나타낸다. 윈도우 크기가 작게 설정될수록 제안된 방법은 보다 빠르게 데이터 집합의 변화를 감지한다. 즉, 윈도우 크기가 작게 설정되었을 때 서로 다른 두 개의 부분 데이터 집합으로 구성된 데이터 집합에서 부분 데이터 집합간의 전황을 보다 빠르게 감지한다. 윈도우 크

기를 조절함으로써 변화 감지 능력을 조절할 수 있다.

6. 결 론

데이터 스트림의 지속적인 확장성을 고려할 때, 해당 데이터 스트림에서 과거에 발생한 정보들은 현재 시점에서는 쓸모없는 정보들이거나 부정확한 정보일 가능성이 높다. 따라서, 최근에 발생한 정보들에 보다 집중하여 분석 결과를 구할 수 있다면 보다 의미 있는 정보를 얻을 수 있다. 본 논문에서는 온라인 데이터 스트림에서 최근 빈발항목을 탐색하기 위한 이동 윈도우 기법을 제안하였다. 제안된 방법은 데이터 스트림에서 나타나는 최신의 변화를 마이닝 결과에 효율적으로 반영할 수 있으며, 이를 통해서 데이터 스트림에 내재된 정보의 시간 흐름에 따른 변화를 효율적으로 분석할 수 있다. 제안된 방법에서 시간 변화에 따라 지속적으로 발생되는 정보들 중에서 유효 정보의 범위는 윈도우 크기에 의해 결정된다. 한편, 제안된 방법에서는 중요항목 지지도에 근거하여 감시해야 할 정도로 큰 지지도를 갖는 항목들의 출현빈도 수만 관리함으로써, 마이닝 수행 과정에서의 메모리 사용량 및 수행 시간을 단축한다. 응용 환경의 특성이나 요구 사항을 고려하여 중요항목 지지도를 조절함으로써 보다 높은 마이닝 정확도를 갖도록 설정하거나 또는 메모리 사용량을 보다 작은 정도로 유지하도록 할 수 있다.

참 고 문 헌

[1] G.S. Manku, R. Motwani, Approximate frequency counts over data streams, *Proceedings of the 28th International Conference on Very Large Databases*, Hong Kong, China, August, 2002.

[2] C.-H. Lee, C.-R. Lin, M.-S. Chen, Sliding-window filtering: An efficient algorithm for incremental mining, *Proceedings of the 10th International Conference on Information and Knowledge Management*, Atlanta, GE, pp.263-270, November, 2001

[3] M. Datar, A. Gionis, P. Indyk, R. Motawi, Maintaining stream statistics over sliding windows, *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, January, 2002.

[4] S. Brin, R. Motwani, J.D. Ullman, S. Tsur, Dynamic itemset counting and implication rules for market basket data, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Tucson, AZ, pp.255-264, May, 1997.

[5] R.C. Agarwal, C.C. Aggarwal, V.V.V. Prasad, Depth first generation of long patterns, *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Boston, MA, pp.108-118, September, 2000,

[6] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, *Proceedings of the 20th International Conference on Very Large Databases*, Santiago, Chile, September, 1994.

[7] A. Savasers, E. Omiecinski, S. Navathe, An efficient algorithm for mining association rules in large databases, *Proceedings of the 21st International Conference on Very Large Databases*, pp.432-444, 1995.

[8] C. Hidber, Online association rule mining, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Philadelphia, PA, pp.145-156, May, 1999.

[9] M. Charikar, K. Chen, M. Farach-Colton, Finding frequent items in data streams, *Proceedings of the 29th International Colloquium on Automata, Language and Programming*, 2002.

[10] D. Lambert, J.C. Pinheiro, Mining a stream of transactions for customer patterns, *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.305-310, 2001.

[11] G. Hulten, L. Spencer, P. Domingos, Mining time-changing data streams, *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.97-106, 2001.

[12] 장중혁, 이원석. 데이터 스트림에서 개방 데이터 마이닝 기반의 빈발항목 탐색. *정보처리학회논문지D*, 10-D(3), 2003.

[13] M. Garofalakis, J. Gehrke, and R. Rastogi, Querying and Mining Data Streams: You Only Get One Look, *The tutorial notes of the 28th Int'l Conference on Very on Large Data Bases*, 2002.

장 중 혁

e-mail : jhchang@amadeus.yonsei.ac.kr

1996년 연세대학교 컴퓨터과학과(학사)

1998년 연세대학교 대학원 컴퓨터과학과 (석사)

2001년~현재 연세대학교 대학원 컴퓨터과학과 박사과정

관심분야 : 데이터 스트림, 데이터마이닝, 멀티미디어 데이터 마이닝, 생물정보학

이 원 석

e-mail : leewo@amades.yonsei.ac.kr

1985년 미국 보스턴대학교 컴퓨터과학과 (학사)

1987년 미국 퍼듀대학교 컴퓨터공학과 (석사)

1990년 미국 퍼듀대학교 컴퓨터공학과 (박사)

1990년~1992년 삼성전자 선임연구원

1993년~1999년 연세대학교 컴퓨터과학과 조교수

1999년~2004년 연세대학교 컴퓨터과학과 부교수

2004년~현재 연세대학교 컴퓨터과학과 교수

관심분야 : 분산 데이터베이스, 멀티미디어 데이터베이스, 객체지향 시스템, 데이터마이닝