산업기술연구(강원대학교 산업기술연구소 논문집), 제25권 A호, 2005.
Journal of Industrial Technology, Kangwon Natl. Univ., Korea, No. 25 A, 2005.

효율적인 닫힌 빈발 시퀀스 마이닝

An Efficient Mining for Closed Frequent Sequences

김 형 근* 황 환 규**
Kim, Hyung-Geun Whang, Whan-Kyu

Abstract

Recent sequential pattern mining algorithms mine all of the frequent sequences satisfying a minimum support threshold in a large database. However, when a frequent sequence becomes very long, such mining will generate an explosive number of frequent sequence, which is prohibitively expensive in time. In this paper, we proposed a novel sequential pattern algorithm using only closed frequent sequences which are small subset of very large frequent sequences. Our algorithm extends the sequence by depth-first search strategy with effective pruning. Using bitmap representation of underlying databases, we can obtain a closed frequent sequence considerably faster than the currently reported methods.

키워드 : *순차 패턴 마이닝, 닫힌 순차 패턴*
Keywords : *Sequential pattern mining, Closed sequential pattern*

1. 서론

컴퓨터 시스템의 발달과 데이터베이스 시스템 사용의 증가로 컴퓨터에 저장되는 데이터의 양은 폭발적으로 증가하고 있다. 현재 컴퓨터에 저장되어 있는 대용량 데이터베이스에 숨어있는 정보를 발견하는 문제에 대한 관심이 높아지고 있다. 이렇게 감추어져 있지만 효용가치가 큰 패턴이나 정보를 찾아내는 연구를 데이터 마이닝(data mining)이라고 한다. 초기에는 연관 규칙 탐사[1]에 대해 연구가 활발히 이루어졌으며 최근에는 순차 패턴 탐사라는 새로운 데이터 마이닝 분야가 소개되어 많은 연구가 수행 중이다.

순차 패턴(sequential pattern)탐사[2]는 한 트랜잭션 안에서 발생하는 항목들 간의 연관 규칙에 시간의 변이를 추가한 것이다. 순차 패턴의 데이터베이스는 고객 번호와 트랜잭션이 발생한 시간, 그 트랜잭션에 포함되어 있는 항목에 대한 정보가 포함되어 있다. 이 때, 모든 고객 시퀀스 중 몇 퍼센트 이상 공통으로 나타나는 비율을 지지도라고 하며 사용자가 정의한 최소 지지도를 만족하는 모든 시퀀스를 찾는 것이 순차 패턴 마이닝이다. 최소 지지도를 만족하는 시퀀스를 빈발 시퀀스(frequent sequence)라고 하고, 빈발 시퀀스 중에서 동일한 지지도를 갖지만 다른 시퀀스에 포함되지 않는 시퀀스를 닫힌 빈발 시퀀스(closed frequent sequence)라고 하며, 빈발 시퀀스 중에서 지지도는 고려하지 않을 때 다른 시퀀스에 포함되지 않는 시퀀스를 최대 시퀀스라고 한다. 순차 패턴 탐사에서는 빈발 시퀀스를 효율적으로 찾는 것이 중요한

* 강원대학교 컴퓨터정보통신공학과 석사과정
** 강원대학교 컴퓨터정보통신공학과 교수, 공학박사
본 논문은 2005년도 두뇌한국21 사업에 의하여 지원되었음.

문제가 된다[2][3][4][5].

현재까지 이루어진 순차 패턴 탐사 방법은 긴 패턴(long pattern)을 찾는 문제나 최소 지지도가 매우 낮게 주어진 문제에는 성능이 급격히 저하된다. 만일 데이터베이스가 하나의 긴 시퀀스 $<(a_1)(a_2)\cdots(a_{100})>$을 갖는다면 최소 지지도가 1보다 큰 경우, 마이닝 과정 중에 $2^{100}-1$개의 빈발 시퀀스를 생성하게 된다. 하지만 이 시퀀스들 중 가장 긴 시퀀스만 알면 나머지 모든 시퀀스들은 서브셋으로 분해하여 계산해 낼 수 있다. 결국 가장 긴 빈발 시퀀스를 미리 알 수 있다면 가장 긴 시퀀스를 제외한 빈발 시퀀스들은 중복되는 정보로 마이닝 과정에 불필요하다[6].

연관규칙 탐사는 이와 유사한 문제를 해결하기 위해 닫힌 빈발 항목(closed frequent itemsets)을 제안했다[7]. 동일한 지지도를 갖는 빈발 항목 중 가장 큰 집합을 닫힌 빈발 항목이라고 한다. 순차 패턴 탐사의 닫힌 빈발 시퀀스도 같은 개념으로, 다음과 같은 관계를 가진다.

시퀀스의 총 수 -

{빈발 시퀀스} ≫ {닫힌 빈발 시퀀스} ≫ {최대 시퀀스}

포함 관계 -

{빈발 시퀀스} ⊇ {닫힌 빈발 시퀀스} ⊇ {최대 시퀀스}

순차 패턴 마이닝에서 닫힌 빈발 시퀀스를 최초로 제안한 Clospan 알고리즘[6]은 PrefixSpan 알고리즘[3]과 기본 구조가 유사하고 시퀀스 간에 프로젝션된 데이터베이스를 비교하여 탐색 공간을 줄인다. Clospan은 긴 패턴을 마이닝할 경우 반복적으로 데이터베이스를 프로젝션하고 프로젝션된 데이터베이스의 특징을 추출하기 위해서 시퀀스마다 한 번 더 프로젝션 하는 연산비용의 문제점을 안고 있다.

본 논문에서는 이 문제를 해결하기 위해 비트맵 표현을 이용한 SPAM 알고리즘[5]을 기본 구조로 깊이우선탐색(Depth First Search)방법으로 시퀀스를 확장하고 시퀀스 특성을 정의하여 탐색공간을 효율적으로 줄이는 다양한 가지치기 방법을 제안한다.

본 논문의 구성은 다음과 같다. 제 2 장에서는 순차 패턴 탐사에 대해 서술하고 관련연구에 대하여 알아보겠다. 제 3장에서는 제안하는 알고리즘의 특징을 설명한 다음 구체적인 예를 들어 알고리즘의 수행과정을 살펴본다. 다음 4장에서 실험을 통해 성능평가를 하며 마지막으로 제 5장에서 결론 및 향후 연구 과제를 제시한다.

2. 순차 패턴 탐사 알고리즘

2.1 문제 정의

순차 패턴 탐사에서 사용하는 데이터베이스는 고객별 트랜잭션의 집합이다. 각각의 트랜잭션은 고객 번호, 트랜잭션이 발생한 시간, 구입한 아이템으로 이루어진다. 한 트랜잭션 내에서 아이템은 집합 개념으로서 아이템의 개수는 고려하지 않는다. 표 1은 고객 아이디와 트랜잭션 시간으로 정렬된 데이터베이스를 나타낸 것이다.

표 1 CID와 TID로 정렬된 데이터베이스

고객 아이디	트랜잭션 시 간	아이템셋
1	1	{a, b, d}
1	3	{b, c, d}
1	6	{b, c, d}
2	2	{b}
2	4	{a, b, c}
3	5	{a, b}
3	7	{b, c, d}

모든 아이템의 집합을 $\{i_1\ i_2\ \cdots\ i_m\}$라고 하면 아이템 집합의 서브셋은 아이템셋이라고 한다. 시퀀스는 아이템셋을 트랜잭션 발생 순서대로 정렬한 리스트이며 $<s_1\ s_2\ \cdots\ s_n>$으로 나타낸다. s_i는 아이템셋 즉, 하나의 트랜잭션 안에 발생한 아이템 집합을 의미한다. 아이템셋 내의 아이템들은 편의상 일정한 순서로 나타낸다. 즉, {(a b d)}의 경우에 함께 구매된 아이템이지만 아이템셋 내에서 사전 편집 순서로 표현된다.(예를 들어, 알파벳 순서 a, b, d 로 나타난다.) 시퀀스의 크기는 시퀀스에 포함된 아이템셋의 개수를 말하고, 길이는 시퀀스에 포함된 아이템의 총 개수로서 길이가 k인 시퀀스를 k-시퀀스라고 부른다.

시퀀스 간에는 포함 관계가 존재할 수 있으며 정의 1과 같이 나타낸다.

정의 1 두개의 시퀀스 $s_a=<a_1, a_2, \cdots, a_n>$, $s_b=<b_1, b_2, \cdots, b_m>$에서 $a_1 \subseteq b_{i_1}$, $a_2 \subseteq b_{i_2}$,$\cdots$, $a_n \subseteq b_{i_n}$, $1< i_1 < i_2 <\cdots<i_n \le m$을 만족하는 $i_1, i_2, ..., i_m$ 이 존재할 때 s_b는 s_a을 포함하며, s_b를 s_a의 수퍼시퀀스(supersequence), s_a를 s_b의 서브시퀀스(subsequence)라고 하며, $s_a \sqsubseteq s_b$로 나타낸다.

예를 들어, <(a b) (c d)>는 <(g) (a b c) (b c d) (h)>에 포함되는데, (a b)는 (a b c)에, (c d)는 (b c d)에 포함되기 때문이다. 그러나 <(a) (b)>는 <(a b)>에 포함되지 않는다. 그 이유는 <(a) (b)>에서 아이템 a와 b는 각기 다른 트랜잭션에서 구입된 아이템인 반면, <(a b)>에서 a와 b는 함께

구입된 아이템이기 때문이다.

고객의 모든 트랜잭션은 트랜잭션 시간 순서로 정렬될 수 있다. 어떤 고객이 구입한 아이템을 트랜잭션 시간 순서로 정렬하여 나열한 것을 고객 시퀀스라고 부른다. 표 2는 표 1의 데이터베이스를 고객 시퀀스로 바꾼 것이다.

표 2 고객 시퀀스

고객 아이디	고객 시퀀스
1	<(a b d) (b c d) (b c d)>
2	<(b) (a b c)>
3	<(a b) (b c d)>

어떤 고객 시퀀스에 시퀀스 S가 포함되어 있을 때 그 고객은 시퀀스 S를 '지지한다'고 말한다. 시퀀스의 지지도는 전체 고객에 대한 그 시퀀스를 지지하는 고객의 비율로 나타낸다. 어떤 시퀀스의 지지도가 10%라면, 전체 고객 중에서 시퀀스를 포함하고 있는 고객이 10%라는 의미이다. 지지도가 높을수록 해당 패턴을 포함하고 있는 고객의 수가 많아진다. 시퀀스 S의 지지도는 정의 2와 같이 나타낸다.

정의 2 고객 시퀀스 D가 주어졌을 때, 시퀀스 S의 지지도

$$S_{support} = \frac{number\ of\ customers\ that\ contain\ S\ in\ D}{number\ of\ customers\ in\ D}$$

이때 시퀀스 S가 최소 지지도를 만족하면 빈발 시퀀스(frequent sequence)라고 부른다.

빈발 시퀀스 중에서 동일한 지지도를 갖지만 다른 시퀀스에 포함되지 않는 시퀀스를 닫힌 빈발 시퀀스라고 하며, 지지도를 고려하지 않을 경우에 다른 시퀀스에 포함되지 않는 시퀀스를 최대 시퀀스라고 한다. 순차 패턴 문제는 최소 지지도를 만족하는 모든 빈발 시퀀스를 찾는 것으로 요약된다.

문제 정의 고객 시퀀스 D와 최소지지도가 주어 졌을 때, 최소 지지도를 만족하는 모든 닫힌 빈발 시퀀스를 찾는 것이다.

표 2의 데이터베이스에서 최소지지도가 2보다 크게 주어졌을 때, 최소지지도를 만족하기 위해서는 시퀀스가 2명 이상의 고객에서 나타나야 한다. 최소 지지도를 만족하는 빈발 시퀀스 중에서 최대 시퀀스는 시퀀스 {<(a b) (b c d) : 2>}로서 고객 1과 3에 의해 지지된다. 닫힌 빈발 시퀀스는 {<(a b)> : 3, <(a b) (b c d)> : 2, <(b) (b c)> : 3}로 3개가 되며, 빈발 시퀀스는 총 30개가 나타난다. 표 3은 닫힌 빈발 시퀀스에서 유도할 수 있는 빈발 시퀀스를 보여 준다. 표 3에서 알 수 있듯이 닫힌 빈발 시퀀스는 빈발 시퀀스보다 현저히 적은 수로 나타나며 서브셋으로 나누면 모든 빈발 시퀀스와 그 지지도를 구할 수 있다.

예를 들어, 시퀀스 <(b) (b c) : 3>에서는 빈발 시퀀스 {<(b)>, <(c)>, <(b) (b)>, <(b) (c)>, <(b c)>, <(b) (b c)>, 지지도 = 3}를 계산해 낼 수 있다. 결국 닫힌 빈발 시퀀스에서 유도할 수 있는 빈발 시퀀스는 중복된 정보로서 닫힌 빈발 시퀀스가 발견된 후에는 탐사할 필요가 없다.

표 3 표 1의 빈발 시퀀스와 닫힌 빈발 시퀀스

닫힌 빈발 시퀀스 (총 3개)	빈발 시퀀스(총 30개)
<(ab):3>	<(a):3>,<(b):3>,<(ab):3>
<(ab)(bcd):2>	<(a)(b):2>,<(a)(c):2>,<(a)(d):2>,<(a)(bc):2>,<(a)(bd):2>,<(a)(cd):2>,<(ab)(b):2>,<(ab)(c):2>,<(ab)(d):2>,<(a)(bcd):2>,<(ab)(bc):2>,<(ab)(bd):2>,<(ab)(cd):2>,<(ab)(bcd):2>,<(b)(d):2>,<(bd):2>,<(b)(bd):2>,<(b)(cd):2>,<(bcd):2>,<(b)(bcd):2>,<(cd):2>,<(d):2>
<(b)(bc):3>	<(b)(b):3>,<(b)(c):3>,<(bc):3>,<(b)(bc):3>,<(c):3>

그래서 대용량 데이터베이스에 무수히 많은 빈발 시퀀스가 포함되어 있다면 그 수가 훨씬 적은 닫힌 빈발 시퀀스를 마이닝 하는 것이 더욱 효율적일 것이다. 제안하는 방법은 모든 빈발 시퀀스가 아닌 닫힌 빈발 시퀀스를 대상으로 마이닝을 수행한다.

2.2 관련 연구

순차 패턴 탐사의 대표적인 알고리즘으로 AprioriAll이 있다[2]. AprioriAll은 연관규칙 탐사에서도 사용한 바 있는 apriori-generation 알고리즘을 이용하여 잠재적인 빈발 시퀀스인 후보 시퀀스를 생성하고, 데이터베이스를 스캔하여 이들 후보 시퀀스의 지지도를 계산하여 빈발 시퀀스를 찾아내는 방법이다.

AprioriAll은 많은 수의 후보 시퀀스를 생성하는 단점이 있다. 후보 시퀀스의 수가 많으면 데이터베이스를 스캔하여 지지도를 구하는 데 오랜 시간이 걸리게 된다. 이 같은 단점을 해결하기 위해 고객 시퀀스에서 빈발 시퀀스를 전치(prefix)로 하여 이들의 후치(postfix)로 구성된 데이터베이스에서 후보 빈발 시퀀스를 찾는 PrefixSpan이 최근 제안되었다[3].

PrefixSpan은 후보 시퀀스를 찾는 횟수를 거듭함에 따라 데이터베이스 사이즈가 줄어들게 되어

효율적인 탐사를 진행할 수 있다. PrefixSpan의 주요 연산은 프로젝션 수행이다. 그래서 데이터베이스가 대용량화되어 찾아야 되는 시퀀스의 수가 기하급수적으로 많아지면 계속적인 프로젝션 수행 비용이 단점으로 작용하게 된다.

최근에는 데이터베이스를 비트맵으로 표현하여 지지도 계산을 효율적으로 수행하는 SPAM이 제안되었다[5]. SPAM은 다양한 길이의 빈발 시퀀스를 깊이우선 탐색 방법으로 시퀀스를 확장하는 중에 발견할 수 있는 장점이 있다. 하지만 빈발 시퀀스를 찾는 PrefixSpan과 SPAM은 찾아야하는 빈발 시퀀스가 기하급수적으로 많아지는 상황에서 실행시간이 오래 걸리거나 심지어 마이닝 수행을 완료하지 못하는 단점을 가지고 있다. 지금까지 방법은 빈발 시퀀스를 효율적으로 탐색하는 기법에 중점을 두었지만 이 방법은 긴 패턴이나 최소지지도가 낮을 때 무수히 많은 후보 시퀀스를 생성하여 성능이 떨어지는 단점이 있었다. 이 같은 단점을 해결하기 위해 빈발 시퀀스가 아닌 닫힌 빈발 시퀀스를 마이닝 대상으로 후보시퀀스를 대폭 줄여 성능향상을 시도한 Clospan이 제안되었다[6].

Clospan은 PrefixSpan과 같이 빈발 시퀀스를 전치로하여 이들의 후치로 구성된 데이터베이스를 구성하고 새로운 빈발 시퀀스를 탐색한다. 탐색 과정에서 프로젝션된 데이터베이스의 특징을 이용하여 기존에 발견된 후보 시퀀스와의 포함 관계를 찾는다. 이 관계를 통해서 모든 빈발 시퀀스보다 적은 수의 빈발 시퀀스를 탐색하여 닫힌 빈발 시퀀스를 발견한다. 하지만 Clospan은 가지치기에 이용하는 특성을 찾기 위해서 기존에 PrefixSpan보다 많은 프로젝션을 수행하므로 추가적인 비용이 큰 문제점이 되며, 마이닝 수행 후에 닫힌 빈발 시퀀스가 아닌 이들의 후보셋을 생성하게 되고 검사를 통하여 원하는 결과를 찾게 된다. 이 방법은 원하는 결과를 얻기 위해서 후처리 과정이 필요한 단점이 있다. 이 단점을 극복하기 위해서 후보를 생성하지 않고 닫힌 빈발 시퀀스를 찾는 BIDE가 최근에 제안되었다[8].

BIDE는 forward-extension event와 backward-extension event의 존재 유무를 이용하여 닫힌 빈발 시퀀스를 판단하기 때문에 기존에 발견한 닫힌 시퀀스를 유지할 필요가 없다. 그러므로 Clospan보다 훨씬 적은 메모리만으로 수행이 가능하고 Clospan의 구현상 가지치기 할 수 없었던 시퀀스도 가지치기 할 수 있어서 훨씬 적은 공간을 탐색하는 장점이 있다. 하지만 아이템셋이 하나의 아이템으로 구성된 경우만을 실험하였기 때문에 다양한 아이템으로 구성된 경우에는 다른 결과를 나타낼 수 있다. 두개의 이벤트 체크는 아이템셋이 다양한 아이템을 가질 경우에 더욱 복잡한 연산이 필요하고 고객 시퀀스가 길어질수록 닫힌 여부를 판단하기 위한 시간이 오래 걸리게 되는 문제점을 가지고 있다. 본 논문은 지금까지 연구된 전치 기반의 프로젝션 방법[3][6][8]이 갖는 문제점을 극복하기 위해서 비트맵을 이용한 새로운 알고리즘을 제안한다.

3. 닫힌 빈발 시퀀스를 찾는 알고리즘

본 장에서는 비트맵을 이용하여 닫힌 빈발 시퀀스를 탐색하는 알고리즘을 제안한다. 제 1절에서는 시퀀스 트리의 구조와 데이터베이스의 비트맵 표현을 설명하고, 2절에서는 후보 생성 방법을 설명한다. 3절에서는 검색 공간을 줄이기 위한 가지치기(pruning) 방법을 알아보고 4절에서는 닫힌 빈발 시퀀스를 찾는 알고리즘을 설명한다.

3.1 시퀀스 트리(sequence tree)와 비트맵 표현

시퀀스 트리는 데이터베이스에 나타나는 아이템을 확장하기 위한 기본적인 구조로 시퀀스 확장의 순서를 나타낸다[5][6][7][8].

데이터베이스의 아이템들 간에 사전편집 순서가 있다면, 아이템 a는 아이템 b보다 먼저 발생하며 a < b로 표기한다. 아이템간의 순서를 시퀀스간의 순서로 확장할 수 있다. 두 개의 시퀀스 s_a=<a_1, a_2, …, a_n>, s_b=<b_1, b_2, …, b_m>, $a_1 \le a_2 \le \dots \le a_n$, $b_1 \le b_2 \le \dots \le b_m$가 있다면, 다음 관계가 성립될 때 시퀀스 s_a는 시퀀스 s_b보다 먼저 발생하며 $s_a < s_b$로 나타낸다.

1. $0 \le i \le \min(n,m)$, for $r < i$, $a_r = b_r$ and $a_i \le b_i$일 경우
2. $n < m$, and $a_1=b_1$, $a_2=b_2$, ..., $a_n=b_n$일 경우

예를 들어, (a f) < (b f), (a b) < (a b c), (a b c) < (b c)의 시퀀스 순서를 가진다.

새로운 시퀀스는 아이템셋 확장과 시퀀스 확장으로 구분할 수 있다. 아이템셋 확장은 현재 시퀀스의 마지막 아이템셋에 하나의 아이템을 추가하는 것이고 시퀀스 확장은 현재 시퀀스에 하나의 아이템으로 구성된 트랜잭션을 추가하는 것으로 요약할 수 있다. 예를 들어, 시퀀스 <(a)>에서 아이템 b를 아이템셋 확장하면 시퀀스<(a b)>가 생성되며, 아이템 b를 시퀀스 확장하면 시퀀스 <(a)(b)>가 생성된다. (이후에 각각의 확장을 간략히 I-단계, S-단계라고 명명한다.)

그림 1은 위의 표 1에서 아이템 a를 루트로 하는 시퀀스 트리이다. 기존에 SPAM[5]의 시퀀스 트리와 주요한 차이점은 시퀀스 확장 순서이다. 빈발 시퀀스를 구할 경우에는 S-단계 확장 후에 I-단계 확장을 한다. 제안하는 알고리즘은 이와 반대로 I-단계 확장을 수행하고 S-단계 확장을 한다.

그 이유는 다음과 같다. 예를 들어, 데이터베이스에 하나의 시퀀스 <(a_1, a_2, a_3, a_4, a_5) (b_1, b_2, b_3, b_4, b_5)>가 있다면 최소지지도가 1보다 큰 경우, <(a_1)>를 깊이우선탐색을 이용하여 I-단계로 확장한 후에 아이템 b_1을 S-단계로 확장한다. 다음 확장할 시퀀스 <(a_1, a_2, a_3, a_4, a_5) (b_1)>에서 다시 I-단계를 재귀적으로 확장한 결과 시퀀스 <(a_1, a_2, a_3, a_4, a_5) (b_1, b_2, b_3, b_4, b_5)>가 닫힌 빈발 시퀀스가 된다. 결국 시퀀스 트리의 제일 왼쪽 가지를 제외한 모든 가지는 닫힌 빈발 시퀀스의 서브셋으로 확장할 필요가 없다. 하지만 S-단계를 먼저 수행하면 시퀀스 길이가 증가할 때마다 {b_1, b_2, b_3, b_4, b_5}에 대한 S-단계 확장을 반복적으로 수행해야 한다. 이 예에서 알 수 있듯이 I-단계를 먼저 수행하면 잠재적인 닫힌 빈발 시퀀스를 더욱 빨리 찾을 수 있다.

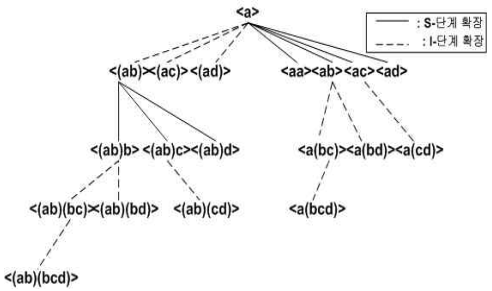


그림 1 시퀀스 트리

그림 1에서 보면 시퀀스 <(a)(b)>에서 확장한 시퀀스 <(a)(bc)>,<(a)(bd)>,<(a)(bcd)>는 시퀀스 <(ab)> 확장의 서브셋이므로 <(ab)>를 먼저 확장하면 가지치기 될 수 있다.

AprioriAll은 전체 데이터베이스를 스캔하여 지지도를 계산하므로 시퀀스의 길이가 증가할 때마다 계속적으로 이루어지는 데이터베이스 스캔은 연산 비용이 큰 문제가 된다. 제안하는 알고리즘은 이 문제를 해결하기 위해서 데이터베이스를 한번 스캔하여 비트맵으로 표현하고 이를 이용한 비트 연산으로 지지도를 효율적으로 계산한다

예를 들어, 시퀀스 <(a)>에서 아이템 b를 확장한 시퀀스 <(a b)>는 시퀀스 <(a)>를 포함하는 트랜잭션에서 아이템 b가 나타날 때 지지된다. 아이템 또는 시퀀스가 트랜잭션에 포함되어 있으면 해당 트랜잭션의 비트는 '1'로 설정하고, 그렇지 않으면 '0'으로 설정한다. 그러므로 시퀀스 <(a)>의 비트맵에서 '1'로 설정되어 있는 트랜잭션이 아이템 b에서도 '1'로 설정되어 있다면 <(a b)>는 그 트랜잭션이 나타난 고객에 의해서 지지된다. 결국 이 시퀀스의 지지도는 두 비트맵의 bitwise-And 연산 후에 값이 '1'로 설정된 고객 수가 된다[5].

최초로 데이터베이스 스캔할 때 길이 1인 아이템을 대상으로 각 고객이 가지는 트랜잭션 수만큼 bit를 가지는 비트맵을 구성한다. 시퀀스 s의 비트맵은 B(s)로 정의한다. 그림 2는 표 2에 대응되는 비트맵 표현이다. 그림 2에서 트랜잭션 4는 아이템 a, b, c를 포함하므로 각 아이템들에 대응되는 비트가 '1'로 설정된 것을 볼 수 있다.

CID	TID	a	b	c	d
1	1	1	1	0	1
	3	0	1	1	1
	6	0	1	1	1
2	2	0	1	0	0
	4	1	1	1	0
3	5	1	1	0	0
	7	0	1	1	1

그림 2 표2에 대응되는 비트맵 표현

3.2 후보 생성 방법(candidate generation)

이 절에서는 후보 시퀀스를 생성하는 방법과 비트맵을 이용하여 시퀀스의 지지도를 계산하는 방법을 I-단계와 S-단계로 나누어 설명한다.

(1) I-단계 확장

정의 3 (I-단계 확장의 지지도 계산) 시퀀스 s=(s_1, ..., s_k)에서 아이템 i를 I-단계로 확장하면 시퀀스 s_r=(s_1, ..., s_k ∪ {i})이 생성된다. 시퀀스 s_r의 지지도는 시퀀스 s의 비트맵 B(s)와 아이템 i의 비트맵 B(i)를 bitwise-AND 연산한다. 즉, B(s_r)= B(s) & B(i)이다.

(a)		b		(a b)
1		1		1
0		1		0
0	&	1	→	0
0		1		0
1		1		1
1		1		1
0		1		0

그림 3 I-단계 확장

정의 3은 시퀀스 s의 마지막 아이템셋에 아이템 i를 추가한 것으로 아이템 i가 시퀀스 s의 마지막 아이템셋 s_k와 같은 트랜잭션에서 발생할 때 해당 고객이 시퀀스 s_r를 지지한다는 의미이다.

I-단계로 확장할 아이템은 F_1={j_1, j_2, ..., j_m}에서 시퀀스 s의 마지막 아이템셋이 s_k={i_1, i_2,..., i_n}일

경우, $i_n < j_h$, $n \le h \le m$를 만족하는 j_h 이다.

예를 들어, 시퀀스 <(a)>에서 확장할 아이템은 $I_{\{a\}}$= {b, c, d}가 된다. 그림 3은 시퀀스 <(a)>에서 I-단계로 아이템 b를 확장한 예로서 정의 3을 이용해 B<(a b)>을 계산하면 고객 1, 2, 3에 '1'로 설정된 비트가 존재하므로 지지도가 3인 것을 알 수 있다.

⑵ S-단계 확장

정의 4 (S-단계 확장의 지지도 계산) 시퀀스 $s=(s_1,...,s_k)$에서 아이템 i를 S-단계로 확장하면 시퀀스 $s_r=(s_1,...,s_k,\{i\})$가 생성된다. 시퀀스 s_r의 지지도를 계산하기 위해서는 시퀀스 s가 나타난 트랜잭션 이후에 비트를 '1'로 설정하는 변환 과정이 필요하다. 변환된 비트맵은 $B(s_T)$로 나타내고 아이템 i의 비트맵 $B(i)$와 bitwise-AND 연산한다. 즉, $B(s_r) = B(s_T) \& B(i)$ 이다.

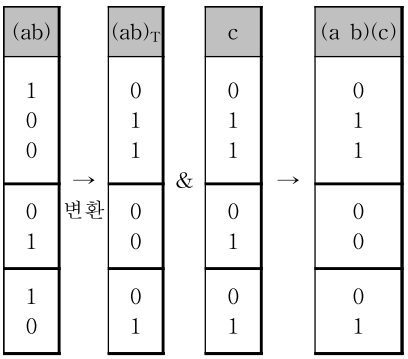


그림 4 S-단계 확장

정의 4은 시퀀스 s의 마지막 아이템셋에 아이템 i로 이루어진 아이템셋을 추가한 것으로 아이템 i가 시퀀스 s의 마지막 아이템셋 s_k가 나타나는 트랜잭션 이후에 나타나면 해당 고객이 시퀀스 s_r을 지지한다는 의미이다.

S-단계로 확장할 아이템은 $F_1=\{j_1, j_2, ..., j_m\}$의 모든 아이템이다. 예를 들어, 시퀀스 <(a)>에서 확장할 아이템은 $S_{\{a\}}=\{a, b, c, d\}$이다. 그림 4는 시퀀스 <(a b)>에서 S-단계로 아이템 c를 확장한 예로서 정의 4에 따라서 B<(a b)(c)>을 계산하면 고객 1, 3에 '1'로 설정된 비트가 있으므로 지지도는 2가 된다.

3.3 가지치기(pruning)

제안하는 알고리즘은 다음과 같은 Apriori 특성을 가지치기에 이용한다. 만일 시퀀스 s가 비빈발이면, 시퀀스 s의 수퍼셋 또한 비빈발이므로 시퀀스 s는 확장하지 않는다. 본 논문에서는 닫힌 빈발 시퀀스를 탐색할 경우에 적용할 수 있는 가지치기 방법을 추가적으로 제안 한다

⑴ 공통으로 나타나는 아이템

정의 5 (공통 아이템) 시퀀스 $s=(s_1,...,s_k)$가 주어졌을 때, 시퀀스 s의 마지막 아이템셋,

1. I-단계 : s_k에 공통으로 나타나는 아이템 또는
2. S-단계 : s_k 이후 트랜잭션에 공통으로 나타나는 아이템을 의미한다.

정의 5에서 시퀀스 s를 지지하는 모든 트랜잭션은 B(s)에 '1'로 설정된 부분이며, 확장할 아이템 i의 비트맵 $B(i)$와 bitwise-AND 연산한 결과 시퀀스 s_r의 비트맵 $B(s_r)$가 B(s)와 같다면 아이템 i는 I-단계의 공통 아이템이다. S-단계의 공통 아이템은 시퀀스의 마지막 아이템셋 다음 트랜잭션에 나타나므로, 변환된 비트맵 $B(s_T)$와 결과 시퀀스 s_r의 비트맵 $B(s_r)$을 비교하여 판단 한다.

보조 정리 1 (공통 아이템을 이용한 가지치기) 시퀀스 s가 주어지고 아이템 α가 공통 아이템으로 밝혀지면 같은 확장 타입의 확장 리스트 $I_{\{s\}}=\{i_1, i_2, ..., j_k\}$(또는 $S_{\{(a)\}}=\{i_1, i_2, ..., i_k\}$)에서 $\alpha < i_n$, $1 \le n \le k$을 만족하는 i_n은 확장 하지 않는다.

예를 들어, 그림 5에서 시퀀스 <(a)>에서 $I_{\{a\}}$= {b, c, d}의 아이템 b로 확장한 시퀀스 <(a b)>와 확장 전 시퀀스 <(a)>가 비트맵이 같을 경우에 <(a c)>,<(a d)>의 확장은 <(a b)> 확장의 서브셋이 되므로 하지 않는다. 즉, <(a c)>, <(a d)> 시퀀스들이 빈발이라면 <(a b c)>, <(a b d)>라는 같은 지지도의 수퍼셋이 시퀀스 <(a b)>의 확장에 이미 존재하기 때문에 닫힌 빈발 시퀀스를 찾는 문제에서는 확장할 필요가 없게 된다. 이런 관계 발견은 두 시퀀스의 비트맵을 직접 비교하지 않고 시퀀스 특성을 이용하여 쉽게 발견할 수 있다.

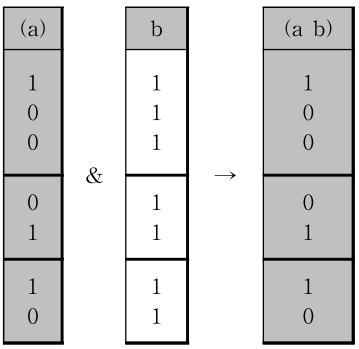


그림 5 I-단계에서 공통으로 나타나는 아이템

정의 6 (시퀀스 특성) 시퀀스 s와 비트맵 B(s)가 주어졌을 때, 시퀀스 특성은 지지도와 B(s)의 비트가 '1'인 트랜잭션의 수이고 표기는 P[s] = [sup(s) : trans(s)]로 한다.

정리 1 (비트맵의 동치) 두개의 시퀀스 s, s′가 주어지고, s가 s′의 서브 시퀀스라면,
B(s) = B(s′) ⇔
P[sup(s) : trans(s)] = P[sup(s′) : trans(s′)]

증명 : B(s) = B(s′) ⇒ P[s] = P[s′]는 자명하다. s ⊏ s′ 관계에서 sup(s') ≤ sup(s), trans(s') ≤ trans(s)인 것을 알 수 있고, B(s)에 '1'로 설정된 모든 비트가 B(s′)에서도 '1'일 경우만 sup(s′) = sup(s), trans(s′) = trans(s)이 성립한다. 그러므로 B(s) = B(s′) 이다.

시퀀스 특성은 비트맵을 구성하는 과정 중에 쉽게 계산할 수 있고 이를 이용하면 정리 1에서 알 수 있듯이 비트맵을 직접 비교하지 않아도 된다.
예를 들어, 그림 5의 가지치기도 두개의 시퀀스 특성이, P[(a)] = [3 : 3], P[(a b)] = [3 : 3], 서로 일치하는 것을 확인함으로서 쉽게 적용할 수 있다.
그림 6을 보면 시퀀스 <(a b)>의 변환된 비트맵 B<(a b)$_T$>에서 처음 '1'로 나타나는 비트에 해당하는 행이 B<(a b)(b)> 비트와 일치한다. 즉, b가 공통 아이템이므로 <(a b)(c)>, <(a b)(d)>의 확장은 <(a b)(b)> 확장의 서브셋이 되서 하지 않는다.

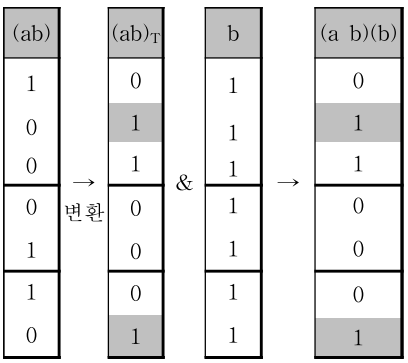


그림 6 S-단계에서 공통으로 나타나는 아이템

⑵ 포함관계를 이용한 가지치기

새로운 시퀀스를 생성할 경우에 기존에 발견한 시퀀스와 시퀀스 특성을 이용하여 포함관계를 밝힌다.

보조 정리 2 (포함관계를 이용한 가지치기) 시퀀스 s에서 시퀀스 s_r로 확장을 하면, 시퀀스 특성을 이용하여 기존에 발견된 시퀀스 s′와 포함 관계를 밝힌다. 포함 관계가 발견이 되면 시퀀스 s_r은 확장 하지 않는다.

확장하지 않는 이유는 다음과 같다.

1. 서브셋 관계라면 시퀀스 s_r의 확장은 s′의 확장과 동일한 지지도를 갖으며 s′의 확장에 포함된다.
2. 수퍼셋 관계라면 시퀀스 s_r의 확장은 s′의 확장과 동일한 지지도를 갖으며 s′의 확장을 하위노드로 보면 된다.

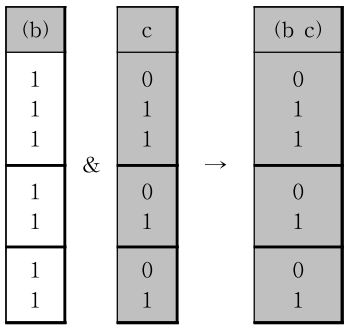


그림 7 I-단계에서 포함관계

예를 들어, 그림 7은 시퀀스 <(b)>에서 아이템 c를 I-단계로 확장할 경우에 두 시퀀스 <(c)>, <(b c)>의 특성이 P[(c)]=[3 : 4], P[(b c)]=[3 : 4] 일치하므로 보조 정리 2의 서브셋 관계에 의해서 <(c)>는 확장하지 않는다. <(c)>에서 확장되는 모든 시퀀스는 <(b c)>에서 확장한 시퀀스의 서브셋이기 때문이다.

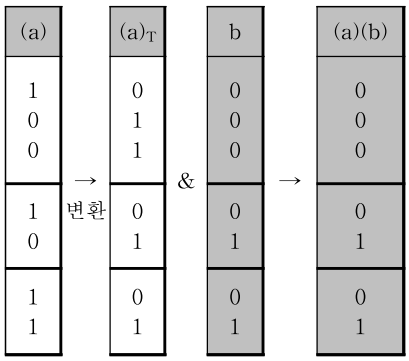


그림 8 S-단계에서 포함관계

그림 8은 S-단계로 확장할 경우로 표 1에 나타나지 않아서 임의로 비트맵을 가정했다. 시퀀스 <(a)>에서 아이템 b를 확장할 때 시퀀스 <(a)(b)>와 <(b)>의 특성이 일치하므로 위의 그림 7의 예제처럼 서브셋 관계에 의해서 <(b)>의 확장은 하지 않는다.

3.4 제안하는 알고리즘 설계 및 구현

제안하는 알고리즘의 전체 수행 과정은 알고리즘 1과 같다.

Algorithm 1: Find_Closed_Mining(D, min_support)

Input : Database DB and min_support
Output : The complete closed sequence set, FCS
FCS = ∅;
F_1 = Find_frequent_1(DB,min_support);
for each (sequence s ∈ F_1)
  call Frequent_Items(s, I_n, S_n);
return FCS;

알고리즘 1, Find_Closed_Mining은 제안하는 알고리즘의 기본 구조로서 처음에 빈발 1-아이템을 탐색하고 각 아이템에 대하여 알고리즘 2, Frequent_Items를 수행한다. 보조 정리 1과 2를 시퀀스 특성을 이용하여 구현한 알고리즘 2는 시퀀스 트리의 노드 순서로 확장하며 가지치기를 수행한다.

Algorithm 2: Frequent_Items (node n=(s_1, ...,s_k), I_n, S_n)

Input : node information n, the set of i-extension items I_n, the set of s-extension items S_n
Output : The complete closed sequence set.
I_{temp} = ∅;
S_{temp} = ∅;
for each ($i \in I_n$)
  if ((s_1,...,s_k ∪ {i}) is frequent) {
    if (!canPruning((s_1,...,s_k ∪ {i}))) {
      if (P[(s_1,...,s_k)] == P[(s_1,...,s_k ∪ {i})]) {
        Stop the I-extension by following item {i};
        Copy the I-extension list to I_{temp};
      }
      else
        I_{temp} = I_{temp} ∪ {i};
    }
  }
for each ($i \in I_{temp}$)
  Frequent_Items((s_1,...,s_k ∪ {i}), all items in I_{temp} greater than i, S_n)
for each ($i \in S_n$)
  if ((s_1,...,s_k,{i}) is frequent) {
    if (!canPruning((s_1,...,s_k,{i}))) {
      if (first '1' bit of B((s_1,...,s_k)$_T$)== B((s_1,...,s_k,{i}))){
        Stop the S-extension by following item {i};
        Copy the I-extension list to S_{temp};
      }
      else
        S_{temp} = S_{temp} ∪ {i};
    }
  }
for each ($i \in S_{temp}$)
  Frequent_Items((s_1,...,s_k,{i}), all items in S_{temp} greater than i, S_{temp})

알고리즘 3, canPruning은 포함관계를 이용한 가지치기를 구현한 부분으로서 서브셋, 수퍼셋 관계를 밝힌다. 이 과정에서 시퀀스 간에 비교하는 것은 시퀀스 특성과 포함관계이다. 포함관계를 밝히기 위한 연산은 비교되는 시퀀스가 많아질 경우 큰 오버헤드로 작용한다. 그러므로 포함관계를 비교하는 시퀀스의 수를 줄이기 위해서 먼저 시퀀스 특성을 비교하도록 구현하였다.

Algorithm 3: canPruning (node s=(s_1,..,s_k))

Input : node information s
Output : The result of pruning check
Check whether a discovered sequence s′ exists either s ⊆ s′or s′⊆ s;
  if (s⊆s′ and P[(s)] == P[(s′)])
    then prune s and modify the sequence tree, return SUB;
  else if (s′⊆ s and P[(s)] == P[(s′)])
    then prune the s′ and modify the sequence tree, return SUPER;
  else
    insert s into sequence tree, return 0;

제안하는 알고리즘의 수행 결과로 생성되는 시퀀스 트리를 기존의 Clospan의 결과 시퀀스 트리와 비교하면 그림 9(a),(b)와 같다.

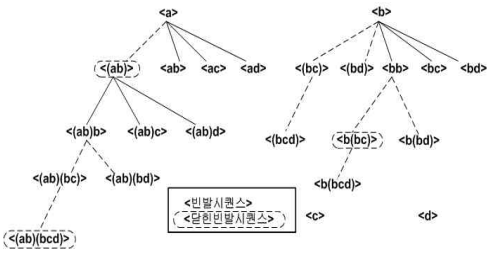


그림 9(a) Clospan의 결과 시퀀스 트리

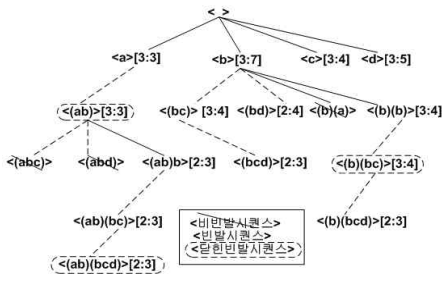


그림 9(b) 제안하는 알고리즘의 결과 시퀀스 트리

Clospan은 그림 9(a)에서 보듯이 23개의 시퀀스를 생성한 후 3개의 닫힌 빈발 시퀀스를 발견한다. 그 반면에 제안하는 알고리즘은 그림 9(b)에서 알 수 있듯이 17개의 시퀀스를 생성한 후에 같은 결과를 얻는다. 결국 제안하는 알고리즘은 Clospan보다 탐색 공간이 현저히 적은 것을 알 수 있다.

4. 실험 결과 및 분석

본 장에서는 제안된 알고리즘을 구현하여 기존에 빈발시퀀스를 마이닝 대상으로 하는 SPAM 알고리즘과 닫힌 빈발 시퀀스를 마이닝하는 Clospan과 BIDE 알고리즘의 실행시간을 비교하며, 이때 발생되는 시퀀스의 패턴 분포 및 닫힌 빈발 시퀀스의 개수 변화를 알아본다. 또한 실험용 인위 데이터 셋의 생성 파라미터 값을 변화시키며 성능에 미치는 영향을 알아본다. 4.1절에서는 성능 평가를 위한 실험 환경 및 데이터 생성 방법을 설명하고, 4.2절에서는 실험 결과 및 분석을 보인다.

4.1 성능 평가에 사용된 실험 환경 및 데이터

(1) 실험 환경

본 연구에서 제안된 알고리즘의 성능평가를 위한 환경은 Linux kernel 2.4, 3.2GHz CPU clock, 2GB 메모리이고, C언어로 구현하였다.

(2) 데이터셋

성능평가에 사용한 데이터셋은 인위데이터로 [2]의 방법에 따라 생성한다. 표 4는 데이터 생성에 사용된 인자들의 리스트를 나타낸 것이다.

표 4 실험 데이터 생성에 사용된 파라미터 목록

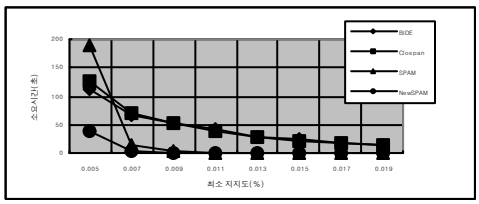
D	고객의 수(000s)
C	고객 당 트랜잭션의 평균 개수
T	트랜잭션 당 아이템의 평균 개수
S	잠재적인 최대 빈발 시퀀스의 평균 길이

4.2 실험 결과

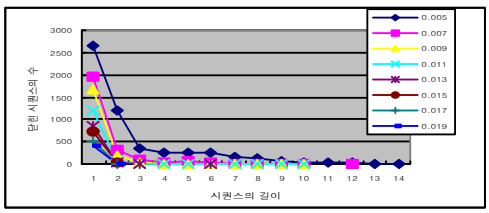
(1) 데이터셋 크기에 따른 성능 비교

실험 1, 2는 데이터셋의 크기를 증가시킬 경우에 각 알고리즘의 실행시간과 생성되는 시퀀스의 수 및 닫힌 빈발 시퀀스의 수를 비교한다.

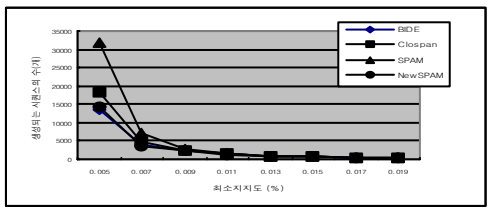
실험 1은 10MB 크기의 D1C10T5S8 파라미터로 생성한 인위데이터를 이용하여 Clospan, BIDE, SPAM과 제안하는 알고리즘, NewSPAM의 성능을 평가한다. 결과 그래프 1(a)을 보면 NewSPAM이 Clospan과 BIDE 보다 3~4배 성능이 향상 됐다. SPAM의 경우는 지지도가 낮아질수록 성능이 급격히 저하되는데 그 이유는 발견되는 빈발 시퀀스의 수가 지지도가 낮아지면서 급격히 많아지기 때문이다. 반면에 NewSPAM은 지지도가 낮아지면서 빈발시퀀스와 닫힌 빈발시퀀스의 개수 차이가 현저히 나타날 때 가장 성능이 우수하다. 하지만 상대적으로 높은 지지도에서는 SPAM이 가장 우수한 성능을 보이는데 NewSPAM의 가지치기 적용을 위한 연산이 오히려 오버헤드로 작용하기 때문이다. 실험 1(b)는 각 지지도별로 발견되는 닫힌 빈발 시퀀스의 수를 나타낸 것으로 지지도가 낮아질수록 닫힌 빈발 시퀀스가 많아지는 것을 확인할 수 있다. 실험 1(c)는 각 알고리즘별로 생성되는 빈발 시퀀스의 수를 나타낸 것으로 SPAM의 경우는 모든 빈발 시퀀스의 수가 되며, Clospan, BIDE와 NewSPAM은 닫힌 빈발시퀀스를 발견하기 위해 생성한 빈발 시퀀스의 수가 된다. 실험 1(c)의 결과를 보면 BIDE, NewSPAM, Clospan 순서로 많은 시퀀스를 가지치기 한다.



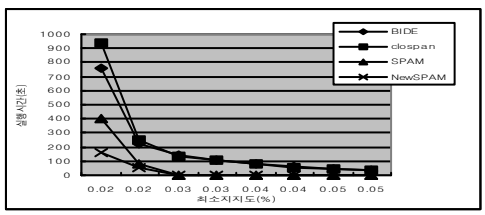
실험 1(a) D1C10T5S8 데이터셋의 성능평가



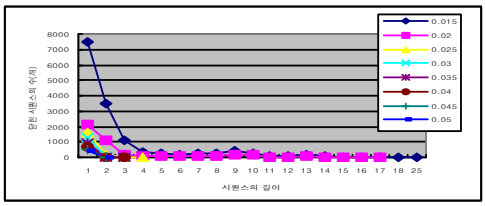
실험 1(b) D1C10T5S8 데이터셋의 패턴 분포



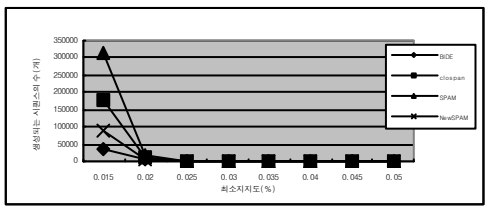
실험 1(c). D1C10T5S8 데이터셋의 생성된 시퀀스의 수



실험 2(a). D10C20T10S20 데이터셋의 성능평가



실험 2(b) D10C20T10S20 데이터셋의 패턴 분포



실험 2(c) D10C20T10S20 데이터셋의 생성된 시퀀스 수

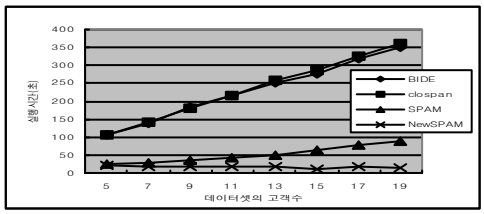
실험 2는 데이터셋의 용량을 40MB 크기로 증가 시켰을 경우로 용량이 커질수록 NewSPAM의 성능이 더 우수해진다. 데이터셋의 용량이 커질수록 닫힌 빈발 시퀀스의 수도 증가하기 때문이다.

실험 1, 2를 통하여 NewSPAM은 SPAM과 BIDE보다 최대 4배, Clospan보다 최대 5배 성능이 향상한 것을 확인했다.

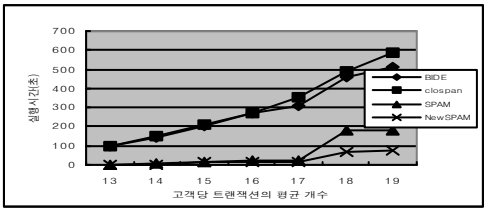
(2) 입력 파라미터 변화에 따른 성능비교

본 절에서는 데이터셋을 생성할 때 입력되는 파라미터들, 고객의 수(D), 고객 당 트랜잭션의 평균개수(C), 트랜잭션 당 아이템의 평균 개수(T), 잠재적인 최대 빈발 시퀀스의 평균 길이(S)를 변화시켜가며 NewSPAM과 SPAM, Clospan, BIDE의 실행시간을 비교한다.

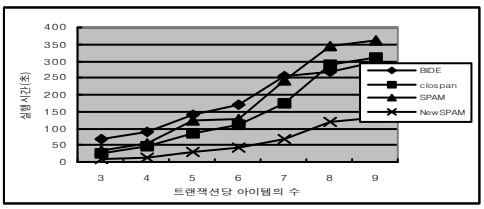
실험 3~6은 각 파라미터별로 실험한 결과로서 NewSPAM이 지지도가 상대적으로 낮아서 빈발 시퀀스와 닫힌 빈발 시퀀스 개수 차이가 현저히 커질 때 우수한 성능을 보인다. 결론적으로 데이터셋의 크기가 커질수록, 고객수가 많아질수록, 고객당 트랜잭션의 평균 개수가 많아질수록, 트랜잭션 당 아이템의 수가 많아질수록, 잠재적인 최대빈발 시퀀스의 길이가 작을수록 제안하는 알고리즘이 우수한 성능을 보인다.



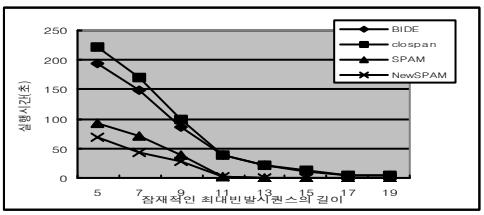
실험 3 데이터셋의 고객 수 변화에 따른 성능평가(D?C20T10S10 지지도=0.04)



실험 4 고객 당 트랜잭션의 평균 개수에 따른 성능평가(D10C?T10S10 지지도=0.03)



실험 5 트랜잭션 당 아이템의 수 변화에 따른 성능평가(D10C10T?S10 지지도=0.007)



실험 6 잠재적인 최대빈발시퀀스의 길이 변화에 따른 성능평가(D10C10T10S? 지지도=0.015)

5. 결론

지금까지 순차 패턴 마이닝은 모든 빈발 시퀀스를 찾는 문제로 정의되었지만 사용자가 임의로 정해주는 최소지지도가 낮거나 찾아야 되는 패턴이 긴 경우에는 기존에 제안된 알고리즘은 수행 시간이 오래 걸리거나 심지어 원하는 시간 안에 결과를 찾을 수가 없게 된다. 이런 상황에서는 빈발 시퀀스보다 훨씬 적은 수를 가지며, 빈발 시퀀스의 정보를 갖고 있는 닫힌 빈발 시퀀스를 찾는 문제로 재정의하여 순차 패턴 마이닝을 효율적으로 수행할 수 있다.

본 논문에서는 이 문제를 해결하기 위해 비트맵을 이용한 시퀀스 특성을 정의하고 I-단계, S-단계 확장 시에 적용할 수 있는 가지치기 방법을 제안한다. 제안하는 알고리즘의 성능 평가를 위해 기존의 알고리즘과 성능을 비교하고 다양한 데이터셋에 대하여 실험하였다. 빈발 시퀀스를 마이닝 대상으로 하는 SPAM과 비교 했을 때 최대 4배의 실행 시간 감소 효과를 볼 수 있으며, 닫힌 빈발 시퀀스를 마이닝 대상으로 하는 Clospan과 BIDE 보

다는 최대 7배의 실행 시간 감소 효과를 볼 수 있다. 최대 효과는 빈발 시퀀스와 닫힌 빈발 시퀀스 개수의 차이가 현저히 나타날 경우로서 연구동기와 부합되는 것을 확인할 수 있다. 또한, 각 데이터 셋의 특성을 변화시켜가며 실행시간을 측정해 본 결과, 제안하는 알고리즘은 데이터 셋의 크기가 커질수록, 고객수가 많아질수록, 고객 당 트랜잭션의 평균 개수가 많아질수록, 트랜잭션 당 아이템의 수가 많아질수록, 잠재적인 최대빈발시퀀스의 길이가 작을수록 특히 효율적이다.

본 논문에서는 닫힌 빈발 시퀀스를 효율적으로 탐색하는 방법을 제안하였으나 이후에는 데이터 셋에 트랜잭션이 주기적으로 추가되는 상황에서 마이닝을 효율적으로 수행하기 위한 닫힌 빈발 시퀀스 갱신 알고리즘에 대한 연구를 계속해서 진행할 예정이다.

참 고 문 헌

[1] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules in large databases", *In Proc ACM SIGMOD*, pp.207-216, May 1993.

[2] R.rikant and R.Agrawal, "Mining sequential Patterns : Generalizations and Performance Improvements", *In EDBT1996*, pp.3-17, Mar. 1996.

[3] J.Pei, J.Han, B.Mortazavi-Asl, H.into, Q.Chen, U.Dayal, and M.-C.Hsu., "PrefixSpan : Mining sequential petterns efficiently by prefix projected pattern growth", *In ICDE2001*, pp.215-226, Apr. 2001.

[4] M.J.Zaki, "SPADE : An efficient algorithm for mining frequent sequences", *Maching Learning*, pp.31-60, 2001.

[5] J.Ayres, J.E.Gehrke, T.Yiu, and J.Flannick, "Sequential pattern mining using bitmaps", *In Proc. 2002 ACM SIGKDD*, Jul. 2002.

[6] Xifeng Yan, Jiawei Han, Ramin Afshar, "CloSpan : Mining Closed Sequential Patterns in Large Datasets", *In Proc. 2003 SIAM Int.Conf. on Data Mining*, May 2003.

[7] M.J.Zaki and C.Hsiao, "CHARM : An efficient algorithm for closed association rule mining", *In Proc. 2002 SIAM Int. Conf. Data Mining*, pp.457-473, Apr, 2002.

[8] Jianyong Wang and Jiawei Han "BIDE : Efficient Mining of Frequent Closed Sequences", *In ICDE2004*, pp.79-90, Mar, 2004