정량 정보를 포함한 순차 패턴 마이닝 알고리즘
(Sequential Pattern Mining Algorithms with Quantities)

김 철 연 † 임 종 화 †† Raymond T. Ng ††† 심 규 석 ††††
(Chulyun Kim) (Jong-Hwa Lim) (Raymond T. Ng) (Kyuseok Shim)

요 약 순차 패턴을 찾는 것은 데이타마이닝 응용분야에서 중요한 문제이다. 기존의 순차 패턴 마이닝 알고리즘들은 아이템으로만 이루어진 순차 패턴만을 취급하였으나 경제나 과학분야와 같은 많은 분야에서는 정량 정보가 아이템과 같이 기록되어 있으며, 기존의 알고리즘이 처리하지 못하는 이러한 정량 정보는 사용자에게 보다 유용한 정보를 전달하여 줄 수 있다.

본 논문에서는 정량 정보를 포함한 순차패턴 마이닝 문제를 제안하였다. 기존의 순차패턴 알고리즘에 대한 단순한 확장으로는 모든 정량에 대한 후보 패턴들을 모두 생성하기 때문에 확대된 탐색 공간을 효율적으로 탐색할 수 없음을 보이고, 이러한 단순한 확장 알고리즘의 성능을 대폭 향상시키기 위하여 정량 정 보에 대해 해쉬 필터링과 정량 샘플링 기법을 제안하였다. 다양한 실험 결과들은 제안된 기법들이 단순히 확장된 알고리즘과 비교하여 수행시간을 매우 단축시켜 줄 뿐만 아니라, 데이타베이스 크기에 대한 확장성 또한 향상시켜줌을 보여 준다.

키워드 : 데이타마이닝, 순차패턴, 정량정보

Abstract Discovering sequential patterns is an important problem for many applications. Existing algorithms find sequential patterns in the sense that only items are included in the patterns. However, for many applications, such as business and scientific applications, quantitative attributes are often recorded in the data, which are ignored by existing algorithms but can provide useful insight to the users.

In this paper, we consider the problem of mining sequential patterns with quantities. We demonstrate that naive extensions to existing algorithms for sequential patterns are inefficient, as they may enumerate the search space blindly. Thus, we propose hash filtering and quantity sampling techniques that significantly improve the performance of the naive extensions. Experimental results confirm that compared with the naive extensions, these schemes not only improve the execution time substantially but also show better scalability for sequential patterns with quantities.

Key words : Data mining, Sequential Pattern, Quantity

1. 서 론

1.1 배경 및 문제점

순차패턴 문제는 데이타마이닝 분야에서 많은 연구가 진행된 분야이며, 구매분석, 통신사용 패턴분석 등의 다양한 응용영역들을 가지고 있다[1,2]. 기존의 연구들 의 순차패턴은 전형적으로 $<s_1,...,s_m>$의 형식을 가진다. 이때 $s_j = \{i_{j,1},...,i_{j,nj}\}$는 아이템의 집합을 의미하며, 각 아이템 $i_{j,k}$은 수량정보를 포함하고 있지 않다. 경영 데이타 분석, 과학적 실험 분석과 같은 많은 응용분야에서 이러한 수량정보가 데이타에 같이 기록되고 있음에도 불구하고, 기존의 순차패턴에서는 이러한 수량정보를 단순히 무시하고 있는 것이다. 예를 들면, 사용자 구매 시퀀스의 경우 사과라는 아이템만을 구매한 정보만 있는 것이 아니라, 사과를 3개 샀다는 수량적인 정보도 포함되어 있는 경우가 많다. 이러한 정량(quantity)적인 정보가 포함되어 있는 순차 패턴 마이닝의 결과를 얻는다면, 어떤 고객이 다음 방문 시에 구매할 것으로 예상되는 물품과 함께 그 구매 개

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성·지원 사업(IITA-2006-C1090-0603-0031)의 연구결과로 수행되었습니다.
† 학생회원 : 서울대학교 전기컴퓨터공학부
cykim@kdd.snu.ac.kr
†† 비 회 원 : 한국과학기술원 전산학과
jlim@kdd.snu.ac.kr
††† 비 회 원 : University of British Columbia Computer Science 교수
rng@cs.ubc.ca
†††† 정 회 원 : 서울대학교 전기컴퓨터공학부 교수
shim@ee.snu.ac.kr
논문접수 : 2006년 2월 1일
심사완료 : 2006년 6월 4일

수에 대한 정보도 얻을 수 있기 때문에, 마케팅 비용과 예상 이익을 비교하여 더 효과적인 마케팅을 할 수도 있을 것이며, 수량 정보를 이용하여 적당한 개수의 물품을 준비해 놓을 수도 있을 것이다. 웹 내비게이션 패턴에서도 다음으로 방문할 페이지와 함께 그 페이지에서 머무는 시간에 대한 정보도 제공해 주기 때문에 머무는 시간이 긴 페이지에서 사용되는 선전 배너나 링크에는 더 많은 광고료를 받는 등으로 이용을 할 수가 있을 것이다.

본 논문에서는 정량 정보를 포함한 순차패턴 문제를 제시하였다. 확장된 아이템은 아이템 번호 $i_{j,k}$와 정량 정보 $q_{i,k}$의 쌍인 $[i_{j,k}, q_{i,k}]$로 표기되며, 확장된 순차패턴은 이러한 확장아이템의 순차로 구성된다. 이러한 정량적인 데이타들이 추가된 마이닝의 경우에 기존의 알고리즘에 비해 탐색 공간(search space)이 늘어나게 되므로, 이러한 탐색 공간의 확장을 줄여서 수행 성능을 개선할 수 있는 기법들을 본 논문에서 제안하고 있다.

1.2 관련 연구

기존의 순차패턴을 찾는 알고리즘들은 패턴을 생성하는 순서에 따라 크게 두가지 부류로 분류할 수 있다. 모든 패턴들을 프리픽스(prefix)트리로 표현한다고 가정할 때, 패턴들을 너비우선 순서로 탐색하거나 깊이우선 순서로 탐색할 수 있다. 너비우선 알고리즘들은 널리 알려진 Apriori식의 알고리즘들로 [1-3] 등이 있으며, 깊이우선 알고리즘에는 대표적으로 Prefix-Span 알고리즘[4]이 있다.

이밖에 순차패턴 탐색의 다양한 확장의 연구가 진행되어 왔다. [3]에서는 아이템들의 계통정보를 고려한 순차패턴 알고리즘을 다루고 있으며, SPIRIT[5]에서는 사용자가 제시한 정규표현식 제약조건을 만족시키는 모든 순차 패턴을 검색한다.

연관규칙 탐색은 순차패턴 검색의 특별한 경우로 취급될 수 있다. 연관규칙 탐색 알고리즘들도 깊이우선 알고리즘들과 너비우선 알고리즘들로 분류될 수 있다. [6,7]에서 소개된 Apriori 알고리즘은 대표적인 너비우선 알고리즘이며, 해쉬 테이블을 사용하여 길이가 2인 패턴들에 대한 탐색을 보다 효율적으로 수행할 수 있다[8]. 깊이우선 알고리즘의 예로서는 Depth-Project[9]와 FP-Tree[10]가 있다. 이러한 알고리즘들이 모든 빈번한 아이템 집합을 검색하는 것과는 달리, 수행시간의 단축을 위해서 빈번한 닫힌(closed) 또는 최대(maximal) 아이템 집합만을 탐색하는 알고리즘이 있으며[11-13], MAFIA[14], 그리고 [15] 등이 이에 포함된다. 이때 빈번한 닫힌 아이템 집합이란 동일한 빈도수를 가지는 초집합(super set)이 존재하지 않는 빈번한 아이템 집합을 말하며, 빈번한 최대 아이템 집합이란 빈번한 초집합을 가지지 않는 빈번한 아이템 집합을 말한다. 본 연구에서도 빈번한 최대 아이템 집합에서 확장된 빈번한 최대 순차 패턴을 검색하여, 수행시간의 향상을 꾀하고 있다. 최근에는 최소한의 정보손실을 감수하면서 빈번한 패턴들을 압축하는 연구가 [16,17] 등에서 소개되고 있다.

2. 문제 정의

$I=\{i_1,i_2,...,i_n\}$를 모든 아이템의 집합이라 하자. 확장아이템은 아이템과 정량의 쌍$[i,n]$으로 표기되며, 이때 i는 I의 원소이며, i의 정량은 1부터 n 사이의 정수에 해당한다. n는 양의 실수로 쉽게 확장이 가능하나, 본 논문에서는 설명의 편의상 정수로 가정한다. 각 아이템마다 구분 능한 정량의 개수는 유한하며, 아이템 i의 정량 값들은 오름 차순으로 $N_i = \{n_{i1},n_{i2},...,n_{ik}\}$으로 표기된다. 전체 확장 아이템의 집합은 $EI = \{[i,n] \mid i \in I \wedge n \in N_i\}$이며, 한 확장아이템 집합(extended item set), eis은 EI의 부분집합으로, 단 동일한 아이템에 대한 중복된 확장아이템을 허용하지 않는다. eis와 EI의 부분집합 관계는 $eis \subseteq_e EI$ 으로 표기한다.

확장 아이템 집합 $eis_1=\{a_1,...,a_n\}$와 $eis_2=\{b_1,...,b_m\}$가 주어 졌을 때, 만약 모든 $a_j=[i,q_a]$에 대하여 $q_a \le q_b$를 만족하는 $b_k=[i,q_b]$가 존재하면, eis_1는 eis_2의 부분집합이라 불리며, $eis_1 \subseteq eis_2$으로 표기된다. 예를 들어 $eis_1=\{[a,3],[d,10]\}$,$eis_2=\{[a,4],[c,1],[d,10]\}$, $eis_3=\{[a,4],[c,1],[d,5]\}$이 주어졌을 때, eis_1과 eis_2는 $eis_1 \subseteq eis_2$ 관계가 성립하나, eis_1과 eis_3 사이에는 부분집합 관계가 성립하지 않는다. 이는 eis_3의 마지막 아이템의 정량이 5이므로 eis_1의 마지막 아이템의 정량보다 작기 때문이다.

순차 패턴은 확장 아이템 집합의 순차로 정의된다. 순차 패턴 s는 $<s_1s_2...s_j>$으로 표기되며, 이때 s_j는 확장아이템 집합이다. 즉, $1 \le j \le l$인 모든 j에 대하여 $s_j \subseteq_e EI$이다. 순차 패턴 안에서 확장 아이템 집합은 $(x_1x_2...x_m)$으로 표기되며, 이때 각 x_k는 EI의 원소이다. 오직 하나의 확장 아이템으로만 구성되는 확장 아이템 집합 (x)는 편의상 괄호를 생략하고 x로 표기될 수 있다. 순차 패턴에 속하는 모든 확장 아이템의 개수를 순차패턴의 길이로 정의한다. 순차패턴 $\alpha=<a_1...a_n>$와 $\beta=<b_1...b_m>$가 주어졌을 때, 만약 $1 \le j_1 < j_2 < ... < j_n \le m$에 대하여 $a_1 \subseteq b_{j1}$, $a_2 \subseteq b_{j2}$,...,$a_n \subseteq b_{jn}$인 정수 $j_1,j_2,...,j_n$들이 존재 한다면, α는 β의 부분패턴이라 불린다.

순차 데이타베이스 S는 순차 아이디 sid 와 순차 s로 구성되는 튜플의 집합이며, 순차 s의 정의는 순차패턴과 동일하다. 만약 순차패턴 α가 순차 s의 부분패

턴이면, 튜플 <sid,s>는 순차패턴 α를 포함한다고 말하며, 순차패턴 α의 지지도 $support_S(\alpha)$는 α를 포함하는 S에 속한 순차의 개수로 정의된다.

최소 지지도 한계 ξ가 주어졌을 때, $support_S(\alpha) \geq \xi$를 만족하는 순차패턴 α는 S에 속한 빈번한 순차패턴이라 불린다. 특히, 빈번한 초패턴(super pattern)이 존재하지 않는 빈번한 순차패턴은 최대 패턴으로 정의한다. 길이가 l인 빈번한 순차패턴은 편의상 l-패턴으로도 불릴 수 있다. 마지막으로 본 논문에서 제안하는 문제에 대한 형식적인 정의는 다음과 같다.

- 주어진 순차 데이타베이스 S와 최소 지지도 한계값 ξ에 대하여, $support_S(\alpha) \geq \xi$를 만족하는 모든 최대 순차패턴을 찾는다.

3. Apriori 방식의 확장

3.1 Apriori-QSP와 수행 예제

순차패턴 탐색을 위한 전형적인 Apriori 알고리즘의 각 단계에서는 직전 단계에서 발견한 빈번한 패턴을 사용하여 이번 단계의 후보패턴을 만들며, 데이타베이스에 대한 한번의 조회를 통해 각각의 후보패턴의 지지도를 구하여 빈번한 패턴의 여부를 판단하게 된다. 자세한 Apriori 방식의 알고리즘들에 대한 설명은 [1,3] 등에서 찾아볼 수 있다.

기존의 Apriori 알고리즘을 정량 정보를 처리할 수 있도록 확장하기 위해서는, 후보패턴의 생성과 지지도 계수의 방식을 수정하여야 한다. 이렇게 확장된 알고리즘을 Apriori-QSP라고 부르기로 한다. 각 단계는 후보패턴의 생성과 지지도 계수, 최대 패턴 여부 검사로 이루어져 있다. k번째 단계에서 k-패턴에 대한 후보패턴들은 apriori-gen 함수 내에서 k-1번째 단계에서 발견한 (k-1)-패턴들을 확장하여 만들어 진다. 두 개의 빈번한 (k-1)-패턴 α와 β가 주어졌을 때, α의 첫번째 확장 아이템을 삭제해서 얻어진 부분패턴이 β의 마지막 확장 아이템을 삭제해서 얻어진 부분패턴과 일치할 경우, α에 β의 마지막 확장 아이템을 덧붙여 k-패턴의 새로운 후보패턴을 생성하게 된다. 후보패턴이 모두 생성된 다음에는 순차 데이타베이스를 스캔하여 후보패턴들의 지지도를 계수한다. 후보패턴들중 지지도가 최소 지지도 이상인 패턴들은 빈번한 패턴들에 속하게 되고, 이를 사용하여 기존의 빈번한 패턴들 중, 새로운 패턴들로 인해 최대 패턴이 되지 못하는 패턴들을 삭제하게 된다.

순차 id	순차
1	<[d,3]([a,3][c,1])[f,1])>
2	<[a,2]([b,2][e,2][f,5])([d,2][f,2])>
3	<([b,2][d,1])[c,3]>
4	<([c,4][d,3])([a,2][c,4][f,5])>
5	<([d,5][f,3])([a,2][b,3])[e,1][c,4]>
6	<([c,6][f,5])[d,5]([a,3][c,4])>

(a) 순차 데이타베이스

아이템	정량	아이템	정량
a	2,3	b	2,3
c	1,3,4,6	d	1,2,3,5
e	1,2	f	1,2,3,5

(b) 아이템과 정량

그림 1 수행 예제

그림 1(a)의 순차 데이타베이스에서 최소 지지도가 3일 때의 예를 살펴보자. 각 단계에서 생성되는 후보패턴과 빈번한 패턴, 최대 패턴은 그림 2에 나타나 있다. 이 예에서는 총 201개의 후보패턴이 생성되며, 4개의 빈번한 최대패턴이 생성된다.

Apriori-QSP 알고리즘이 정량 정보를 가진 확장 아이템들을 처리할 수 있음에도 불구하고, 모든 정량에 대한 후보 패턴들을 생성하게 되기 때문에 알고리즘의 수행이 매우 비효율적이다. 이 장의 나머지 부분에서는 후보 생성을 최적화 할 수 있는 휴리스틱들을 소개하도록 한다.

3.2 해쉬 필터링 기법

만약 확장 아이템 $[i,n_{ii}]$의 n_{ii}가 아이템 i에 대한 정량의 최소값이라면 이러한 확장 아이템을 기본 확장 아이템이라고 부르기로 하자. 만약 순차 데이타베이스에서 정량 부분을 무시하고 기존의 순차패턴 탐색 알고리즘을 실행시켰을 때와 결과와 기본 확장 아이템만으로 구성된 빈번한 순차의 결과는 동일한 지지도를 갖는 결과를 보일 것을 쉽게 예상할 수 있다. 예를 들어 <dc>의 지지도와 <[d,1][c,1]>의 지지도는 동일할 것이다.

해쉬-필터링을 사용하는 Apriori-Hash 알고리즘은 우선 정량 부분을 무시한 기존의 방식의 후보패턴을 생성하며, 이의 지지도를 계수하여 빈번한 기본 확장 패턴들을 구한다. 이 단계를 필터링 단계라고 부른다. 이렇게 구한 기본 확장 패턴들을 모든 가능한 정량 값들의 조합으로 확장하여 후보패턴들을 생성된 뒤, 모든 빈번한 패턴들을 찾아낸다. 이 단계는 정량 계수 단계라고 불린다. 단, 정량 값들의 조합 중 기본 패턴이 되는 것은 후보패턴으로 다시 생성하지 않는다는 것을 주의하여야 한다. 마지막으로 기존의 빈번한 패턴들 중 최대 패턴이 되지 못하는 것들을 삭제하여 최대 패턴만을 남긴다.

그림 1(a)의 순차 데이타베이스에서 최소 지지도가 3일 때의 예를 살펴 보자. 각 필터링 단계와 정량 계수 단계에서 생성되는 후보패턴과 빈번한 패턴들은

길이	후보 패턴	빈번한 패턴	빈번한 최대 패턴
1	N/A	12개 <[a,2]>, <[b,2]>, <[c,1]>, <[c,3]>, <[c,4]>, <[d,1]>, <[d,2]>, <[d,3]>, <[f,1]>, <[f,2]>, <[f,3]>, <[f,5]>	<[b,2]>, <[f,2]>
2	198개 <[a,2][a,2[>, <[a,2][b,2]>, <([a,2][b,2])>, ... , <[d,3][f,5]>, <([d,3][f,5])>, <[f,5][d,3]>	13개 <([a,2][c,1])>, <[d,1][a,2]>, <[d,2][a,2]>, <[d,3][a,2]>, <[d,1][c,1]>, <[d,1][c,3]>, <[d,1][c,4]>, <[d,2][c,1]>, <[d,2][c,3]>, <[d,2][c,4]>, <[d,3][c,1]>, <[d,3][c,3]>, < [d,3][c,4]>	<[d,3][c,4]>
3	3개 <[d,1]([a,2][c,1])>, <[d,2]([a,2][c,1])>, <[d,3]([a,2][c,1])>	3개 <[d,1]([a,2][c,1])>, <[d,2]([a,2][c,1])>, < [d,3]([a,2][c,1])>	<[d,3]([a,2][c,1])>

그림 2 Apriori-QSP에 의해 생성되는 후보 패턴과 빈번한 패턴

길이	후보패턴(필터링단계)	빈번한 기본패턴	후보패턴(정량-계수 단계)	빈번한 패턴
2	35개 <aa>,<ab>, <(ab)>,<ba> ,...., <df>,<(df)>,<fd>	3개 <(ac)>,<da>,<dc>	12개 <([a,2][c,3])>,<([a,2][c,4])>,<[d,2][a,2]>,<[d,3][a,2]>,<[d,1][c,3]>,<[d,1][c,4]>,<[d,2][c,1]>,<[d,2][c,3]>,<[d,2][c,4]>,<[d,3][c,1]>,<[d,3][c,3]>,<[d,3][c,4]>	10개 <[d,2][a,2]>,<[d,3][a,2]>,<[d,1][c,3]>,<[d,1][c,4]>,<[d,2][c,1]>,<[d,2][c,3]>,<[d,2][c,4]>,<[d,3][c,1]>,<[d,3][c,3]>,<[d,3][c,4]>
3	1개 <d(ac)>	1개 <d(ac)>	2개 <[d,2]([a,2][c,1])>,<[d,3]([a,2][c,1])>	2개 <[d,2]([a,2][c,1])>,<[d,3]([a,2][c,1])>

그림 3 해쉬-필터링을 사용했을 때의 후보패턴과 빈번한 패턴

그림 3에 주어져 있다. 길이가 2인 패턴을 구하는 단계에서 35개의 후보패턴이 필터링 단계에서 생성되며, 12개의 후보패턴이 정량-계수 단계에서 생성되어 도합 47개의 후보 패턴이 생성된다. 해쉬-필터링 기법을 사용하지 않았을 경우 198개의 후보패턴이 생성되었을 경우와 비교했을 때 상당히 많은 수의 후보패턴이 생략되었음을 주목하여야 한다.

이러한 예제의 결과를 일반화하여 다음의 보조정리는 해쉬-필터링 기법이 최대한 ASP-QSP 만큼의 후보패턴들만을 생성해 낸다는 것을 보여주고 있다.

보조정리 3.1. 해쉬-필터링을 사용한 경우에 만들어지는 후보 패턴의 수는 이를 사용하지 않은 경우의 후보 패턴의 수보다 항상 작거나 같다.

증명. 해쉬-필터링을 사용한 경우에 만들어지는 후보 패턴의 집합을 C_1라고 하면 C_1은 필터링 단계에서 만들어지는 후보 패턴의 집합인 C_{filter}와 그 이후의 단계에서 만들어지는 후보 패턴의 집합인 C_{after}의 합집합이다. 즉 $C_1 = C_{filter} \cup C_{after}$이며, 필터링 단계에서 만들어지는 후보 패턴은 그 이후의 단계에서 다시 만들어 질 필요가 없으므로 $C_{filter} \cap C_{after} = \varnothing$이 된다. 또한 해쉬필터링을 사용하지 않은 경우의 후보 패턴의 집합을 C_2라고 하자. 여기서 $C_{filter} \subseteq C_2$이고 $C_{after} \subseteq C_2$이므로 $C_1 \subseteq C_2$가 된다. 또한 $|C_{filter}|+|C_{after}|=|C_1| \leq |C_2|$가 된다. □

3.3 정량 샘플링

다음의 최적화 기법은 정량 샘플링으로, 각 아이템의 정량마다 일정 간격(interval)를 두고, 그 간격에 해당하는 후보 패턴만을 만든 후[1], 그 후보 패턴들 중 빈번한 패턴들을 얻은 후, 그 다음 단계에서 간격 사이의 후보 패턴을 모두 만드는 방법을 사용한다. 즉, 샘플링 단계에서는 간격 개수에 따른 일부분의 (coarse-grained) 후보패턴을 만들어서 결과를 얻게 되고, 1단계에서 얻은 빈번한 패턴중에 다른 패턴의 진 부분패턴[2]인 것들을 지운 뒤, 남아 있는 패턴들을 정량-계수 단계에서 세밀한 후보 패턴을 만드는 데에 사용한다. 이때 샘플링 단계의 후보패턴중 하나의 빈번한 패턴으로부터 생성되는 세밀한 후보패턴들의 집합을 해당 패턴의 격자 단위(*grid cell*)라 부른다. 예를 들어 아이템 a의 정량이 {1,2,3,4,5,6}일 때, 샘플링 단계에서 [a,1]와 [a,4]가 생성되었을 경우, 정량-계수 단계에서 [a,1]의 격자단위는 [a,2], [a,3]이며, [a,4]의 격자단위는 [a,5], [a,6]이 된다.

그림 1(a)의 순차 데이타베이스에서 최소 지지도가 3, 샘플링의 인터벌 갯수가 2인 경우를 살펴보자. 그림 4(a)에서 보여 주는 바와 같이, 길이가 2인 패턴을 위해서 89개의 후보패턴이 샘플링 단계에서 생성됨을 알 수 있다. 이 후보패턴들 중 7개의 패턴이 빈번하며, 이 7개의 패턴들 중 <[d,1][c,1]>은 진 부분패턴 삭제에 의해 제거 된다. 그림 4(a)에서 밑줄이 쳐진 빈번한 패턴은 진 부분패턴 삭제에 의해 제거된 패턴을 의미한다. 보조정리 3.2는 이러한 진 부분패턴 삭제가 전체 최대 패턴의 검색에 아무런 문제가 없음을 보여주고 있다. 정량-계수 단계에서 4개의 후보패턴

1) 정량의 개수가 간격의 개수보다 작은 경우는 모든 정량을 다 생성한다.
2) 아이템 부분은 모두 같고 정량 부분은 모두 작은 부분패턴

이 추가로 생성되어 도합 93개의 후보패턴이 생성되게 된다. 길이가 3인 패턴을 위한 후보패턴과 빈번한 패턴은 그림 4(b)에 나타나 있다.

단계	후보패턴	빈번한 패턴
1	89개 <[a,2][a,2]>,<[a,2][b,2]>,<([a,2][b,2])>...<[d,3][f,3]>,<([d,3][f,3])>,<[f,3][d,3]>	7개 <([a,2][c,1])>,<[d,1][a,2]>,<[d,3][a,2]>,<[d,1][c,1]>,<[d,1][c,4]>,<[d,3][c,1]>,<[d,3][c,4]>
2	4개 <([a,2][c,3])>,<[d,2][a,2]>,<[d,2][c,4]>, <[d,3][c,3]>	3개 <[d,2][a,2]>,<[d,2][c,4]>,<[d,3][c,3]>

(a) 길이가 2인 후보패턴과 빈번한 패턴

단계	후보패턴	빈번한 패턴
1	2개 <[d,1]([a,2][c,1])>,<[d,3]([a,2][c,1])>	2개 <[d,1]([a,2][c,1])>,<[d,3]([a,2][c,1])>
2	1개 <[d,2]([a,2][c,1])>	1개 <[d,2]([a,2][c,1])>

(b) 길이가 3인 후보패턴과 빈번한 패턴

그림 4 정량-샘플링을 사용했을 때의 후보패턴과 빈번한 패턴

보조정리 3.2. 샘플링 단계에서 찾아진 빈번한 패턴 중에 진 부분패턴을 지워도 최대의 빈번한 패턴을 잃지 않는다.

증명. 샘플링 단계의 결과로 얻은 빈번한 패턴 α와 β가 있고, α가 β의 진 부분패턴이라고 하자. α를 지우지 않고 정량 계수 단계에서 α를 이용하여 후보 패턴을 만들었다고 하면, 그 후보 패턴들은 모두 β의 부분 패턴이 된다. 따라서 최대의 빈번한 패턴이 될 수가 없다. 따라서 샘플링 단계의 빈번한 패턴 중에서 진 부분 패턴들을 지워도 최대의 빈번한 패턴을 찾는 문제에 영향을 주지 않는다. □

각각의 단계에서 샘플링된 후보패턴이 이전단계에서 발견된 빈번한 패턴들의 샘플링된 패턴에 의해 생성되고, 지지도 계수를 통해 이중 빈번한 패턴이 찾아진다. 이러한 빈번한 패턴에 대해 보조정리 3.2에 따라 진부분패턴 삭제를 적용하여 탐색 공간을 축소한 후, 나머지 빈번한 패턴들을 확장하여 격자 단위를 구한 뒤 다시 빈번한 패턴들을 찾아낸다. 마지막으로 기존의 빈번한 패턴들중, 최대 패턴이 되지 못하는 패턴들을 제거한다.

다음의 보조정리는 정량 샘플링이 전체 생성되는 후보패턴의 개수를 감소시켜 줌을 보여준다.

보조정리 3.3. 정량 샘플링을 사용한 경우에 만들어지는 후보 패턴의 수는 정량 샘플링을 사용하지 않은 경우의 후보 패턴의 수보다 항상 작거나 같다.

증명. 정량 샘플링을 사용한 경우에 만들어지는 후보 패턴의 집합을 C_1라고 하면 C_1은 샘플링 단계에서 만들어지는 후보 패턴의 집합인 C_{level1}와 정량-계수 단계에서 만들어지는 후보 패턴의 집합인 C_{level2}의 합집합이다. 즉 $C_1=C_{level1} \cup C_{level2}$이며, 샘플링 단계에서 만들어지는 후보 패턴은 정량-계수 단계에서 다시 만들어 질 필요가 없으므로 $C_{level1} \cap C_{level2}=\varnothing$이 된다. 또한 정량 샘플링을 사용하지 않은 경우의 후보 패턴의 집합을 C_2라고 하자. 여기서 $C_{level1} \subseteq C_2$이고 $C_{level2} \subseteq C_2$이므로 $C_1 \subseteq C_2$가 된다. 또한 $|C_{level1}|+|C_{level2}|=|C_1| \leq |C_2|$가 된다. □

3.4 Apriori-ALL

마지막으로 앞서 이야기한 두가지 최적화 기법을 동시에 적용하는 방식에 대해서 소개하도록 한다. 이 알고리즘은 Apriori-ALL이라 불리우며, 3가지 단계로 이루어진다. 첫번째 단계는 필터링 단계이며, 이 단계에서 이루어지는 작업은 Apriori-Hash의 그것과 동일하다. 두번째 단계는 샘플링 단계이며, 이 단계에서 이루어지는 작업은 Apriori-Sampling의 그것과 유사하며, 단지Apriori-Hash의 결과 빈번한 기본 패턴들에 대해서만 샘플링을 수행한다는 점만 차이가 난다. 세번째 단계는 정량-계수 단계로 이는 Apriori-Sampling의 그것과 동일하다. Apriori-ALL도 다른 최적화 알고리즘에서와 마찬가지로 각 단계의 마지막에는 최대 패턴이 되지 못하는 패턴들을 삭제하여 준다.

4. PrefixSpan 방식의 확장

본 절에서는 깊이우선 탐색기법으로 확장된 알고리즘에 대해서 살펴보도록 한다.

깊이우선 탐색 기법의 확장된 알고리즘은 기존의 대표적인 깊이우선 탐색 알고리즘인 PrefixSpan 알고리즘에 기반을 두고 있다. PrefixSpan 알고리즘에 대한 자세한 사항은 [4]에 소개되어 있다. 본절의 나머지 부분에서는 깊이우선 탐색 기법에서의 확장에서 또한 해쉬-필터링 기법과 정량 샘플링 기법의 아이디어가 동일하게 적용될 수 있음을 보일 것이다.

```
Procedure PrefixSpan-QSP(α, DB, level)
begin
1. Fα := Apriori-ALL(α, DB, level)
2. F := { x | x := α • y for all y∈Fα }
3. Gα^level := { x | x는 y ∈ Fα^level의 부분정량 순차패턴 }
4. for each β ∈ Gα^level do {
5.     DB' := projection(α • β, DB)
6.     F := F∪PrefixSpan-QSP(α • β, DB', level)
7. }
8. return F
end
```

그림 5 PrefixSpan-QSP

4.1 PrefixSpan-QSP

PrefixSpan이 정량 정보를 처리할 수 있도록 하기 위해서 우선 확장 아이템 집합에 대한 데이타베이스 프로젝션과 프로젝션된 데이타베이스에서 빈번한 확장 아이템을 찾는 방식이 확장되어야 한다. 이러한 확장이 고려된 PrefixSpan 알고리즘을 PrefixSpan-QSP라고 부르기로 하며, 그림 5에 해당 알고리즘이 주어져 있다. PrefixSpan-QSP는 α, DB 그리고 level을 입력 매개변수로 가진다. DB는 프리픽스 α로 프로젝션된 데이타베이스를 가리키며, level은 PrefixSpan-QSP 알고리즘에서 찾아낼 α를 프리픽스로 가지는 빈번한 패턴의 길이(프리픽스 부분을 제외한)를 의미한다. 레벨-대-레벨(level-by-level) 프로젝션을 위해서는 level 값을 1로, 이중-레벨(bi-level) 프로젝션을 위해서는 level 값을 2로 설정하게 된다. Prefix-QSP에서는 level 만큼의 길이를 갖는 빈번한 패턴을 찾기 위해서 변형된 형태의 Apriori-ALL을 사용한다. 이를 위해 변형된 Apriori-ALL은 프로젝션된 데이타베이스 DB와 이에 대한 프리픽스 α, 그리고 패턴의 최대 길이 level을 인자로 갖는다. 즉, 변형된 Apriori-ALL에서는 level 길이의 빈번한 패턴을 검색하고 나면, 더이상 단계를 반복하지 않고 바로 함수를 종료하며, level 길이를 가지는 모든 빈번한 최대 패턴들을 리턴한다. 이 결과를 F_α로 부르고, 이 패턴들과 주어진 프리픽스 α를 결합하여 완성된 패턴들을 F에 저장한다. 그리고 이중에서 더 큰 패턴으로 확장될 가능성이 있는 모든 level 길이의 패턴, F_α^{level}을 가지고 가능한 정량 값을 조합하여 G_α^{level}을 생성한 후, G_α^{level}의 각 패턴을 이용하여 DB를 다시 프로젝션하고 PrefixSpan-QSP를 재귀적으로 호출하게 된다.

그림 1의 순차 데이타베이스에 대해 최소 지지도가 3, level이 1인 경우의 예를 살펴보도록 하자. 최초의 PrefixSpan-QSP의 호출에서 Apriori-ALL은 [a,2], [b,2], [c,4], [d,3], [f,5]를 리턴하며, 이를 바탕으로 G_{null}^{1}은 {[a,2], [b,2], [c,1], [c,3], [c,4], [d,1], [d,2], [d,3], [f,1], [f,2], [f,3], [f,5]}이 된다. 각각의 패턴에 대해서 데이타베이스를 프로젝션하고 PrefixSpan-QSP를 재귀적으로 호출하게 된다. 이러한 방식의 재귀적 호출이 깊이 우선 방식으로 이루어지며 모든 빈번한 패턴들을 탐색하게 된다. 전체적인 재귀적 호출은 그림 6에 나타나 있으며, 최종적으로 24번의 재귀적 호출이 이루어짐을 알 수 있다.

4.2 해쉬-필터링

3.2절에서 이미 너비우선 탐색에서의 해쉬 필터링의 아이디어를 소개한 바 있다. 기본적으로 PrefixSpan-Hash에서도 동일한 아이디어가 적용 가능하다. 즉,

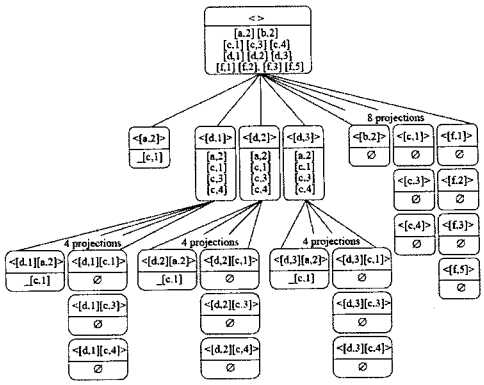


그림 6 PrefixSpan-QSP의 재귀적 호출 트리

우선 정량 정보를 무시하고 기존의 PrefixSpan 알고리즘에 의해 발견된 빈번한 패턴에 대해서만 정량의 조합을 확장하여 재귀적인 호출을 통해 전체적인 빈번한 패턴을 구하는 것이다. 첫번째 단계는 너비우선 탐색에서와 마찬가지로 필터링 단계로 부르며, 두번째 단계는 정량-계수 단계라고 부른다.

그림 7은 PrefixSpan-Hash의 그림 1의 순차 데이타베이스에 대해 최소 지지도가 3, level이 1인 경우의 재귀적 호출 트리를 보여준다. 색깔이 칠해져 있는 노드는 필터링 단계에서의 재귀적 호출을 의미하며, 색깔이 칠해져 있지 않는 노느는 정량-계수 단계에서의 재귀적 호출을 의미한다. 최초 루트 노드의 12개의 빈번한 패턴의 경우 필터링 단계에서 <[c,1]>과 <[f,1]>이 더 이상 패턴이 자라날 수 없음을 확인하였으므로 <[c,3]>, <[c,4]>,<[f,2]>,<[f,3]>,<f,5]>에 대해서는 재귀적 호출이 일어나지 않았음을 주목하라. 전체적으로 13번의 재귀적 호출이 일어났으며, 이는 PrefixSpan-QSP의 24번에 비해 많은 수의 재귀적 호출이 생략되었음을 알 수 있다.

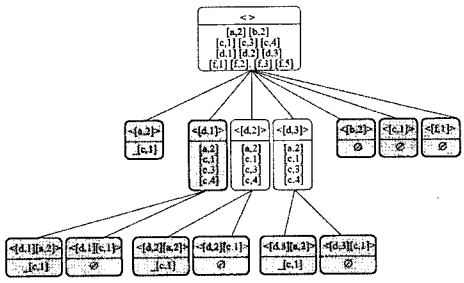


그림 7 PrefixSpan-Hash의 재귀적 호출 트리

4.3 정량 샘플링

3.3절에서 소개했었던 너비우선 탐색의 정량 샘플링

기법 또한 깊이우선 탐색 기법에 적용될 수 있다. PrefixSpan-QSP에서는 모든 정량에 대해 재귀적 호출을 불렀지만, 정량 샘플링 기법에서는 정량마다 일정 간격 (interval)을 두고, 그 간격에 해당하는 순차패턴에 대해서만 재귀적 호출을 부른 후, 이 중 한개 이상의 순차패턴을 재귀적 호출의 결과로 갖는 순차패턴에 대해서만, 그 다음 단계에서 간격 사이의 순차패턴에 대해 모두 재귀적 호출을 부르는 방법을 사용한다. 즉, 샘플링 단계에서는 간격 개수에 따른 일부분의 재귀적 호출을 통해서 결과를 얻게 되고, 샘플링 단계에서 1개 이상의 순차 패턴을 재귀적 호출의 결과로 갖는 순차 패턴 중에 다른 패턴의 호출의 결과와 동일한 결과를 갖는 패턴들을 보조정리 4.1에 의거해 삭제한 뒤, 남아 있는 패턴들을 정량-계수 단계에서 세밀한 재귀적 호출을 부르는 데에 사용한다.

보조정리 4.1. PrefixSpan-Sampling이 샘플링 단계에서 사용한 프리픽스 $\alpha \subseteq \lambda_1 \subseteq \lambda_2 \subseteq ... \subseteq \lambda_n \subseteq \beta$에 대한 재귀적 호출과정에 있다고 가정하자. 프릭픽스 Θ에 대한 재귀적 호출의 결과로 얻어진 빈번한 패턴들을 Θ의 포스트픽스, P_Θ라고 정의할 때, $P_\alpha = P_\beta$가 만족한다면 모든 i=1,...,n에 대한 프리픽스 λ_i의 재귀적 호출을 생략하여도 모든 최대 패턴을 찾아낼 수 있다.

증명. 주어진 데이타베이스에 대하여 만약 프리픽스 α, β에 대하여, $\alpha \subseteq \beta$가 성립한다면, $P_\alpha \supseteq P_\beta$이 성립하며, 이는 $P_\alpha \supseteq P_{\lambda 1} \supseteq P_{\lambda 2} \supseteq ... \supseteq P_{\lambda n} \supseteq P_\beta$가 성립함을 의미한다. 따라서 $P_\alpha = P_\beta$ 이라면 $P_\alpha = P_{\lambda 1} = P_{\lambda 2} = ... = P_{\lambda n} = P_\beta$가 성립하므로, λ_i에 의한 어떤 빈번한 패턴도 최대 순차 패턴이 될 수 없다. □

그림 1의 순차 데이타베이스에 대해 최소 지지도가 3, level이 1인, 샘플링의 인터벌의 개수가 2인 경우의 PrefixSpan-Sampling의 실행에 대해서 살펴보자. 전체적인 재귀적 호출에 대한 트리는 그림 8에 주어져 있다.

색깔이 칠해져 있는 노드는 샘플링 단계에서의 재귀적 호출을 의미하며, 색깔이 칠해져 있지 않는 노느는 정량-계수 단계에서의 재귀적 호출을 의미한다. 최초의 루트노드에서의 샘플링 단계에서는 <[a,2]>, <[b,2]>,<[c,1]>,<[c,4]>,<[d,1]>,<[d,3]>,<[f,1]>,<[f,3]> 패턴들에 대한 8번의 재귀적 호출이 일어난다. 이중, <[b,2]>,<[c,1]>,<[c,4]>,<[f,1]>,<[f,3]>의 재귀적 호출에서는 하나의 빈번한 패턴도 발견되지 않았으므로 정량-계수 단계에 해당 패턴의 격자 단위는 참여하지 않는다. 정량-계수 단계에 참여할 패턴은 <[d,1]>가 있으나, 이 경우는 <[d,1]>와 <[d,3]>의 포스트픽스들이 동일하므로, 보조정리 4.1에 의해 실제 정량-계수 단계에 참여하는 패턴은 모두 제거되게 된다. 전체적으로는 14번의 재귀적 호출이 발생하며, 이는 PrefixSpan-QSP의 24회와 비교하여 많은 수의 재귀적 호출이 생략되었음을 알 수 있다.

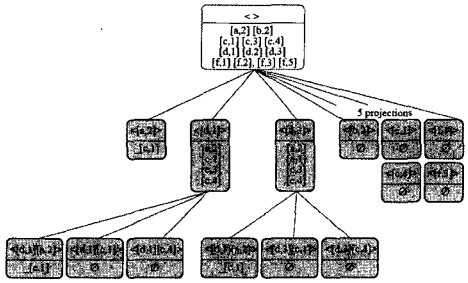


그림 8 PrefixSpan-Sampling의 재귀적 호출 트리

4.4 PrefixSpan-ALL

앞서 소개한 해쉬-필터링 기법과 정량 샘플링 기법은 Apriori-ALL의 경우와 마찬가지로 3단계로 적용될 수 있으며, 이러한 알고리즘을 PrefixSpan-ALL이라고 부른다. 첫번째 단계는 필터링 단계이며, 이 단계에서 이루어지는 작업은 PrefixSpan-Hash의 그것과 동일하다. 두번째 단계는 샘플링 단계이며, 이 단계에서 이루어지는 작업은 PrefixSpan-Sampling의 그것과 유사하며, 단지 PrefixSpan-Hash의 결과 재귀적 호출에서 빈번한 패턴들을 가지는 기본 패턴들에 대해서만 샘플링을 수행한다는 점만 차이가 난다. 세번째 단계는 정량-계수 단계로 이는 PrefixSpan-Sampling의 그것과 동일하다.

5. 실험 결과

모든 실험은 512MB 메모리의 Pentium-4 2.4 GHz 기계상에서 Linux 환경에서 실행되었다. 모든 알고리즘은 GCC 2.95.3 버젼을 사용하여 구현되었다. 본 실험은 합성 데이타와 실생활 데이타 집합에 대하여 최소 지지도의 변화에 따른 수행 시간을 측정하였으며, 실생활 데이타 집합에서 얻어낸 순차패턴들을 소개하였다.

5.1 합성 데이타 집합

합성 데이타 생성기를 구현하기 위하여 IBM Almaden 연구센터가 제공하는 합성 데이타생성기[3]를 수정하여 사용하였다. 수정 사항은 생성된 아이템 집합에 지수분포에 따라 정량의 값을 할당하는 과정이 추가 되었다. 데이타 생성기에 사용된 입력 매개변수는 표 1에 정리되어 있으며, 사용된 입력 매개변수의 설정은 표

3) www.almaden.ibm.com/cs/quest/syndata.html\#assocSynData

표 1 매개변수

$\|D\|$	순차의 개수
$\|S\|$	잠재된 빈번한 패턴의 평균 길이
N_S	잠재된 빈번한 패턴의 개수
N_I	잠재된 빈번한 아이템 집합의 개수
N	아이템의 개수
R	패턴 반복 수준
Q	정량의 평균값

표 2 합성 데이타 집합

Data Set	$\|D\|$	$\|S\|$	N_S	N_I	N	R	Q
data.default	100K	10	1K	10K	100K	0	30
data.rept_0.1	100K	10	1K	10K	100K	0.1	30
data.rept_0.3	100K	10	1K	10K	100K	0.3	30
data.rept_0.5	100K	10	1K	10K	100K	0.5	30
data.ncust_10	10K	10	1K	10K	100K	0	30
data.ncust_50	50K	10	1K	10K	100K	0	30
data.ncust_200	200K	10	1K	10K	100K	0	30
data.ncust_400	400K	10	1K	10K	100K	0	30

2에 정리되어 있다. 표기되지 않은 입력 변수값은 IBM 생성기의 표준입력값을 따른다.

5.1.1 Apriori 알고리즘의 결과

그림 9는 최소 지지도를 변화시켜 가면서 Apriori 계열의 알고리즘들에 대한 성능 비교를 보여준다. 실험 결과는 제안된 해쉬 필터링 기법과 정량 샘플링 기법이 단순히 확장된 Apriori-QSP 알고리즘의 성능을 획기적으로 향상시켜 줌을 확인시켜 주고 있으며, 특히 Apriori-All 알고리즘이 가장 뛰어난 성능을 보여 준다.

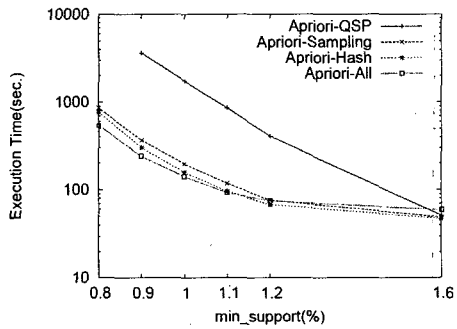


그림 9 Apriori 계열 알고리즘

5.1.2 PrefixSpan 알고리즘의 결과

그림 10은 PrefixSpan 계열 알고리즘들의 실행시간들을 비교하여 보여준다. 본 실험에는 레벨-대-레벨과 이중-레벨 구현이 모두 비교되었다. 그림 상에서 위의 4개의 결과가 레벨-대-레벨 구현의 것들이며, 아래의 4개의 결과가 이중-레벨의 결과이다. 이 결과를 통해 마찬가지로 제안된 해쉬 필터링과 정량 샘플링이 PrefixSpan-QSP의 성능을 향상시켜 줌을 보여주며, 이중 PrefixSpan-ALL의 성능이 레벨-대-레벨

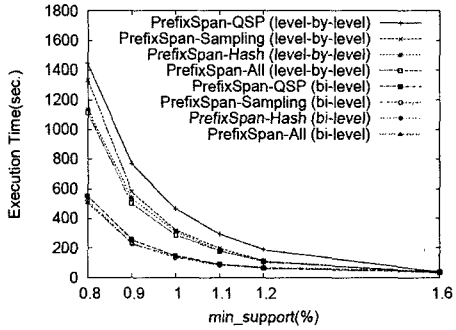
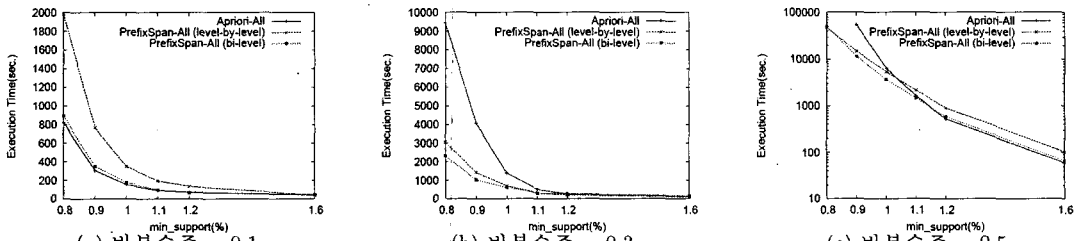


그림 10 PrefixSpan 계열 알고리즘

과 이중-레벨 구현에서 모두 가장 좋은 성능을 보임을 알 수 있다.

5.1.3 Apriori_ALL과 PrefixSpan-ALL의 비교

Apriori-ALL, PrefixSpan-ALL(레벨-대-레벨), PrefixSpan-ALL(이중-레벨)에 대해서 다양한 반복 수준으로 생성한 합성데이타를 가지고 성능을 비교하였다. 반복 수준을 크게 할수록 생성되는 빈번한 패턴의 평균길이가 길어지는 특성을 가지고 있다. 그림 11은 반복수준이 0.1, 0.3, 0.5 일때를 비교하여 보여주고 있다. 반복수준이 0.1일 때는 Apriori-ALL이 가장 좋은 성능을 보이며, PrefixSpan-ALL(이중-레벨)이 중간의 성능을 보인다. 반복수준을 높일수록, Apriori-ALL의 성능이 나빠지고, PrefixSpan-ALL(레벨-대-레벨)의



(a) 반복수준 = 0.1 (b) 반복수준 = 0.3 (c) 반복수준 = 0.5

그림 11 Apriori-ALL과 Prefix-ALL의 비교

성능이 상대적으로 좋아지는 것을 확인할 수 있다.

확장성. 그림 12에서는 최소 지지도가 1.0% 일때, 순차의 갯수를 바꿔가면서 각 알고리즘의 확장성을 시험하였다. 이 결과를 통해 세가지 알고리즘이 모두 데이타베이스의 크기에 대해 거의 선형적인 확장성을 가짐을 확인할 수 있다.

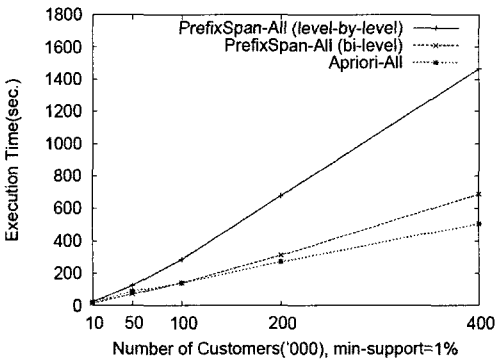


그림 12 순차 개수에 따른 확장성

5.2 실생활 데이타 집합

실생활 데이타 집합 실험을 위하여 온라인 마켓플레이스의 6개월치 매출 데이타를 분석하였다. 최소한 한명의 고객이 두번이상 구매한 아이템과 최소한 하나의 아이템에 대해서는 2 이상의 정량을 갖는 순차에 대해서 분석을 하였다. 결과적으로 사용된 데이타 집합에는 559,056개의 순차와 157,849개의 아이템이 포함되었으며 순차의 평균 길이는 3.0, 아이템의 평균 정량은 2.9였다.

최소지지도의 변화에 따른 각 알고리즘의 결과 그래프의 양상은 합성데이타의 그것과 크게 다르지 않음을 확인하였다. 발견된 순차 패턴의 예는 다음과 같다. 단, 데이타에 대한 권리는 이를 제공한 회사에 있으므로 해당 상품명을 약자로 표기하여 소개하도록 한다. 아래의 패턴은 0.01%의 최소지지도를 가지고 실험한 결과이다.

<([J,5][T,1])>: 이 패턴에 따라 5개의 J 상품과 하나의 T 상품을 하나의 패키지로 묶어 판매하는 마케팅 방식을 제안하였다.

<[B,1][U,3]>: 이 패턴을 사용하여 하나의 B상품을 구입한 고객에게 U 상품에 대한 3장의 쿠폰을 증정하는 정책을 수립하였다.

본 실험 결과를 통하여 정량 정보를 포함하지 않는 기존의 순차패턴에서는 제공할 수 없는 유용한 정보가 정량 정보를 포함한 순차패턴이 실생활에서 유용하게 활용될 수 있음을 확인할 수 있었다.

5. 결 론

논문에서는 기존의 순차패턴 탐색 알고리즘을 확장하여 정량 정보까지도 포함한 순차패턴 탐색 알고리즘을 제시하였다.

정량 정보가 있는 순차 패턴을 찾는 알고리즘으로 Apriori와 PrefixSpan 알고리즘의 초보적인 확장 알고리즘들은 생성해야 하는 후보 패턴의 수와 재귀적 호출의 수를 크게 늘리게 되어 결과적으로 수행 시간도 크게 증가하였다. 따라서 결과를 얻는 데에 필요한 후보 패턴의 수와 재귀적 호출의 수를 줄임으로써 수행시간을 개선할 수 있는 방법을 제안하였다. 즉, 해쉬 필터링 기법과 정량 샘플링 기법이라는 새로운 아이디어를 사용하여 최대의 패턴이 될 수 없는 후보패턴들과 재귀적 호출들을 미리 전지함으로써 알고리즘의 수행속도를 매우 개선하였다. 다양한 실험분석을 통해 본 논문이 제안한 기법들이 매우 효과적임을 확인할 수 있었다.

참 고 문 헌

[1] R. Agrawal and R. Srikant, "Mining Sequential Patterns," Proc. of ICDE, Taipei, Taiwan, Mar., 1995.

[2] H. Mannila, H. Toivonen and A. Inkeri Verkamo, "Discovery of Frequent Episodes in Event Sequences," Data Mining and Knowledge Discovery, Vol. 1, No. 3, 1997.

[3] R. Agrawal and R. Srikant, "Mining Sequential Patterns: Generalizations and performance improvements," Proc. of EDBT, Avignon, France, Mar., 1996.

[4] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal and M.-C. Hsu, "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth," Proc. of KDD. Apr., 2001.

[5] M. Garofalakis, R. Rastogi and K. Shim, "SPIRIT: Sequential pattern mining with regular expression constraints," Proc. of VLDB, Edinburgh, UK, Sep., 1999.

[6] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," Proc. of VLDB, Santiago, Chile, Sep., 1994.

[7] R. Srikant, and R. Agrawal, "Mining Generalized Association Rules," Proc. of VLDB, Zurich, Switzerland, Sep., 1995.

[8] J. Park, M. Chen and P. S. Yu, "An Effective Hash Based Algorithm for Mining Association Rules," Proc. of SIGMOD, San Jose, California, May, 1995.

[9] R. Agarwal, C. Aggarwal, and V. V. V. Prasad, "Depth-first generation of long patterns," Proc. of

KDD, Boston, MA, Aug., 2000.
[10] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," Proc. of SIGMOD, Dallas, TX., May, 2000.
[11] R. J. Bayardo, "Fast Subsequence Matching in Time-Series Databases," Proc. of SIGMOD, Seattle, Washington, Jun., 1998.
[12] N. Pasquier, Y. Bastide, R. Taouil and L. Lakhal, "Discovering frequent closed itemsets for association rules," Proc. of ICDT, Jerusalem, Israel, Jan. 1999.
[13] K. Gouda and M. J. Zaki, "Efficiently mining maximal frequent itemsets," Proc. of ICDM, San Jose, CA, Nov., 2000.
[14] D. Burdick, M. Calimlim and J. Gehrke, "MAFIA: a maximal frequent itemset algorithm for transactional databases," Proc. of ICDE, Heidelberg, Germany, Apr., 2001.
[15] J. Wang, J. Han and J. Pei. "CLOSET+: searching for the best strategies for mining frequent closed itemsets," Proc. of KDD, Washington, DC, Aug., 2003.
[16] F. N. Afrati, A. Gionis and H. Mannila, "Approximating a collection of frequent sets," Proc. of KDD, Seattle, Washington, Aug., 2004.
[17] D. Xin, J. Han, X. Yan and H. Cheng, "Mining Compressed Frequent-Pattern Sets," Proc. of VLDB, Trondheim, Norway, Aug., 2005.

김 철 연

1996년 서울대학교 컴퓨터공학과 졸업(학사). 1998년 서울대학교 인지과학 협동과정 졸업(석사). 2003~현재 서울대학교 전기컴퓨터공학부 박사과정. 관심분야는 데이타마이닝, 데이타베이스, Stream Data, 생물정보학

임 종 화

2000년 한국과학기술원 전산학과 졸업(학사). 2002년 한국과학기술원 전산학과 졸업(석사). 관심분야는 데이타마이닝

Raymond T. Ng

1984년 University of British Columbia (학사). 1986년 University of Waterloo (석사). 1992년 University of Maryland, College Park(박사). 현재 University of British Columbia 교수. 현재 IEEE TKDE, VLDB Journal의 Editorial Board ACM SIGMOD, VLDB, SIGKDD, ACM PODS를 비롯 다수의 국제 학술대회의 Program Committee Member임. 관심분야는 데이타마이닝, 생물정보학, 이미지 데이타베이스, 멀티미디어 시스템

심 규 석

1986년 서울대학교 전기공학과 졸업(학사). 1988년 University of Maryland, College Park(석사). 1993년 University of Maryland, College Park(박사). 1994년~1994년 Federal Reserve Board, Research Staff. 1994년~1996년 IBM Almaden Research Center, Research Staff. 1999년~2002년 KAIST 전산학과 조교수. 2002년~현재 서울대학교 전기컴퓨터공학부 부교수. 현재 VLDB저널과 IEEE TKDE저널의 Editorial Board, ACM SIGMOD, VLDB, ICDE, ACM SIGKDD, ICDM, EDBT를 비롯 다수의 국제 학술대회의 Program Committee Member임. 관심분야는 데이타마이닝, 데이타베이스, XML, Stream Data