논문 2006-43CI-5-3

점진적인 순차 패턴 갱신 알고리즘

(An Incremental Updating Algorithm of Sequential Patterns)

김 학 자*, 황 환 규*

(Hak Ja Kim and Whan Kyu Whang)

요 약

본 논문에서는 데이터베이스에 새로운 트랜잭션이 추가되었을 때 순차 패턴을 갱신하는 문제를 연구하였다. 트랜잭션이 순차적으로 증가되는 환경에서 기존에 발견된 빈발 시퀀스를 재사용하여 순차패턴을 갱신하는 효율적인 알고리즘을 제안한다. 본 논문에서 제안한 방법은 후보 집합의 개수를 효율적으로 줄임으로써 AprioriAll이나 PrefixSpan 알고리즘보다 좋은 성능을 보임을 실험으로 확인하였다.

Abstract

In this paper, we investigate a problem of updating sequential patterns when new transactions are added to a database. We present an efficient updating algorithm for sequential pattern mining that incrementally updates added transactions by reusing frequent patterns found previously. Our performance study shows that this method outperforms both AprioriAll and PrefixSpan algorithm which updates from scratch, since our method can efficiently utilize reduced candidate sets which result from the incremental updating technique.

Keywords: 데이터 마이닝, 순차 패턴, 순차 패턴 갱신

I. 서 론

컴퓨터 시스템의 발달과 데이터베이스 시스템 사용의 증가로 컴퓨터에 저장되는 데이터의 양은 폭발적으로 증가하고 있다. 기업의 일일 거래 데이터, 고객 데이터, 상품 데이터 혹은 각종 마케팅 활동에서의 고객 반응 데이터 등 모든 사용 가능한 근원 데이터를 기반으로, 데이터 속에 감추어진 지식이나 기대하지 못했던 경향 또는 새로운 규칙 등을 발견하고 이를 실제 비즈니스 의사결정 등에 유용한 정보로 활용하고자 하는 시도가 1990년대 초기부터 활발하게 이루어지고 있다. 이렇게 대규모 데이터베이스로부터 다양한 방법을 이용하여 데이터를 탐색하고 분석하여 예상하지 못했던 유용한 정보를 찾아내는 방법을 통칭하여 데이터 마이닝이라 한다[1].

데이터 마이닝의 여러 가지 기법 중에서 최초로 그리고 가장 많이 연구된 것으로 연관규칙(Association Rule) 탐사가 있다[2]. 연관규칙 탐사는 데이터베이스에서 한 트랜잭션 내에 있는 아이템 간의 연관관계를 찾아내는 것이며 이를 이용하여 한 장바구니에서 동시에 구매될 가능성이 큰 물품들을 찾아내 마케팅 전략이나 상점에서의 물품 배치 등에 응용할 수 있다.

연관규칙 탐사 알고리즘을 응용하여 순차적으로 발생한 아이템을 찾아내는 것이 순차 패턴 탐사로서 Agrawal과 Srikant에 의해 처음으로 소개되었다[3]. 순차 패턴의 예는 다음과 같다: "비디오 대여점에서 30%의 고객은 'Star Wars' ⇒ 'Empire Strikes Back' ⇒ 'Return of the Jedi' 의 순서로 비디오를 대여한다." 여기서 '30%'는 최소 지지도(minimum support)로서 전체 고객 중에 이 패턴을 포함하는 고객의 비율을 나타낸다. 순차패턴 탐사는 소비자의 구매 패턴을 분석하는 것 외에도 웹 페이지 접근 패턴 분석, 과학실험, 질병치료, 자연재해 분석, DNA 분석 등에 응용되고 있다.

* 정회원, 강원대학교 컴퓨터정보통신공학과
(Department of Computer and Telecommunication Eng., Kangwon National University)
※ 본 논문은 부분적으로 강원대학교 정보통신연구소의 지원을 받았음
접수일자: 2006년2월7일, 수정완료일: 2006년8월29일

데이터베이스가 대용량화됨에 따라 연산시간과 메모리 사용면에서 효율적인 마이닝 알고리즘의 개발이 중요해 졌다. 또한 데이터베이스는 지속적으로 트랜잭션이 추가되기 때문에 새로운 데이터베이스 상태를 반영하기 위해서는 발견된 규칙을 갱신해야만 한다. 순차 패턴 탐사에 있어서 마이닝 알고리즘에 대한 연구는 많이 진행되어 왔으나 갱신 알고리즘에 대한 연구는 미비하며 이것이 본 연구의 동기가 되었다.

순차 패턴을 갱신하는 한 가지 방법은, 갱신된 전체 데이터베이스를 대상으로 기존 알고리즘을 재실행하는 것이다. 이 방법은 상대적으로 구현이 간단한 편이지만 갱신 전에 이미 발견한 패턴들을 다시 발견하게 되므로 연산비용의 낭비가 있다[4].

본 논문에서는 갱신 전에 발견한 패턴을 재이용하고 새로운 패턴을 찾기 위한 후보 집합의 수를 줄임으로써 효율적으로 순차 패턴을 갱신하는 알고리즘을 제안한다. 본 논문과 유사한 논문으로 연관규칙을 갱신하기 위한 한 [4]의 방법이 있지만 본 논문은 응용 분야가 다른 순차 패턴에 대한 갱신 방법을 제안하며 [4]보다 효율적인 가지치기 방법을 사용하였다. 그 결과, 갱신된 전체 데이터베이스를 대상으로 순차 패턴 탐사 알고리즘을 적용하는 방법에 비해 연산속도 면에서 향상된 성능을 보임을 실험으로 확인할 수 있었다.

본 논문의 구성은 다음과 같다. 제 Ⅱ장에서는 순차 패턴 탐사에 대해 서술하고 Ⅲ장에서 순차 패턴 갱신에 대한 문제를 정의하고 갱신 알고리즘을 제시한다. 제 Ⅳ장에서는 갱신 알고리즘을 사용했을 때의 연산시간을 갱신 알고리즘을 사용하지 않았을 때와 비교하여 실험한다. 끝으로 Ⅴ장에서 결론으로 마무리한다.

Ⅱ. 순차 패턴 탐사 알고리즘

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

아이템셋은 아이템의 집합으로서 $(i_1\ i_2 \cdots i_m)$으로 나타낸다. 시퀀스는 아이템셋을 트랜잭션 발생 순서대로 정렬한 리스트이며 $<s_1\ s_2 \cdots s_n>$으로 나타낸다. S_j는 아이템셋을 의미한다. 시퀀스에 포함된 아이템셋의 개수를 시퀀스의 길이라고 하며, 길이가 k인 시퀀스를 k-시퀀스라고 부른다.

시퀀스는 다른 시퀀스를 포함할 수 있다. 정의 1은 시퀀스 간의 포함 관계를 나타낸 것이다.

정의 1. 시퀀스 $S1=<a_1, a_2, \cdots, a_n>$, $S2=<b_1, b_2, \cdots, b_m>$에서 $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i2}$, $\cdots$, $a_n \subseteq b_{im}$을 만족하는 $i_1<i_2<\cdots<i_n$이 존재할 때 S_2는 S_1을 포함한다.

예를 들어, <(c) (d e) (h)>는 <(g) (c h) (i) (d e f) (h)>에 포함되는데, (c)는 (c h)에, (d e)는 (d e f)에 (h)은 (h)에 포함되기 때문이다. 그러나 <(c) (e)>는 <(c e)>에 포함되지 않는다. <(c) (e)>에서 아이템 c와 e는 각기 다른 트랜잭션에서 구입된 아이템인 반면, <(c e)>에서 c와 e는 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.

어떤 고객 시퀀스에 시퀀스 S가 포함되어 있을 때 그 고객은 시퀀스 S를 '지지한다'고 말한다. 시퀀스의

표 1. 고객 번호와 트랜잭션 시간으로 정렬된 데이터베이스

Table 1. Database sorted by customer number and transaction time.

고객 번호	트랜잭션 시간	구입한 아이템
1	4	a
1	5	b, c
1	14	d
2	1	a, b
2	10	b, c
2	11	d, e, f
3	2	a
3	3	b
3	6	c
3	12	d
4	7	a
4	9	b, f
4	13	c
5	8	g

표 2. 고객 시퀀스

Table 2. Customer sequences.

고객 번호	고객 시퀀스
1	<(a) (b c) (d)>
2	<(a b) (b c) (d e f)>
3	<(a) (b) (c) (d)>
4	<(a) (b f) (c)>
5	<(g)>

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.

정의 2. 고객 시퀀스 D가 주어졌을 때, 시퀀스 S의 지지도

$$S_{support} = \frac{number\ of\ customers\ that\ contain\ S\ in\ D}{number\ of\ customers\ in\ D} \quad (1)$$

순차 패턴 기법을 적용하기 위해서는 사용자가 정의하는 최소 지지도를 입력해야 한다. 이 때 최소 지지도를 만족하는 시퀀스를 빈발 시퀀스(frequent sequence)라고 부른다.

정의 3. 최소 지지도가 σ로 주어졌을 때 빈발 시퀀스가 되기 위한 시퀀스 S의 조건

$$S_{support} \geqq \sigma \quad (2)$$

빈발 시퀀스 중에서 다른 시퀀스에 포함되지 않는 시퀀스를 최대 시퀀스라고 한다. 순차 패턴 문제는 최소 지지도를 만족하는 모든 빈발 시퀀스를 찾는 것인데, 빈발 시퀀스는 최대시퀀스의 부분집합이므로 결국 순차패턴 문제는 최대 시퀀스를 찾는 것으로 요약될 수 있다.

표 2의 데이터베이스에서 최소지지도가 25%로 주어졌을 때, 최소지지도를 만족하기 위해서는 시퀀스가 2명 이상의 고객에서 나타나야 한다. 그러한 시퀀스 중 최대 시퀀스는 4개로서 그 중 하나인 시퀀스 <(a) (bc) (d)>는 고객 1과 2에 의해 지지된다. 또, 시퀀스 <(a) (b) (c)>는 고객 3, 4에 의해, <(a) (b) (d)>는 고객 1, 2, 3에 의해 지지되고, <(a) (f)>는 고객 2, 4에 의해 지지된다.

순차 패턴 탐사의 대표적인 알고리즘으로AprioriAll이 있다[3]. Agrawal은 길이 k인 빈발시퀀스의 모든 부분집합은 반드시 빈발시퀀스라는 데 착안하여 apriori-generation 알고리즘을 개발하였다. 예를 들어 길이 3인 빈발시퀀스 <(a) (b) (c)>의 부분집합인 길이 2인 시퀀스 <(a) (b)>, <(b) (c)>, <(a) (c)>는 모두 빈발시퀀스라는 것이다. 따라서 <(a) (b)>, <(b) (c)>, <(a) (c)>를 구하고 나면 길이 3인 빈발시퀀스의 후보인 <(a) (b) (c)>를 생성할 수 있고, 이것이 실제로 빈발시퀀스인지를 확인하기 위해서 데이터베이스를 스캔하여 지지도를 구한다. 이렇게 길이 k인 빈발시퀀스를 찾기 위해 길이 (k-1)인 빈발시퀀스를 이용하여 상향식(bottom-up) 방식으로 진행한다.

이 때 만약 후보시퀀스의 수가 많으면 데이터베이스를 스캔하여 지지도를 구하는 데 오랜 시간이 걸린다. 이같은 단점을 해결하기 위해 고객 시퀀스에서 빈발 시퀀스를 전치(prefix)로 하여 이들의 후치(postfix)로 구성된 데이터베이스에서 후보 빈발 시퀀스를 찾는 PrefixSpan이 최근 제안되었다[5]. PrefixSpan은 후보 시퀀스를 찾는 횟수를 거듭함에 따라 데이터베이스 사이즈가 줄어들게 되어 효율적인 탐사를 진행할 수 있다.

III. 순차 패턴 갱신 알고리즘

본 장에서는 순차 패턴을 갱신하는 알고리즘을 제안한다. 제 1절에서는 순차 패턴 갱신에서 다루어야 할 문제에 대해 서술하고, 2절에서는 제안하는 갱신 알고리즘에 대해 자세히 기술하도록 하겠다.

1. 문제 정의

데이터 마이닝에서 다루는 데이터베이스는 지속적으로 트랜잭션이 추가되어 갱신된다. 데이터가 추가된 데이터베이스는 데이터가 추가되기 이전의 데이터베이스에서 발견한 순차 패턴 외에 새로운 패턴을 포함하고 있을 수 있다. 또 반대로 기존에 발견한 패턴이 더 이상 유효하지 않을 수도 있다. 그러므로 패턴이 신뢰할만한 것이 되기 위해서는 데이터베이스의 갱신에 따라 패턴 또한 주기적으로 갱신되어야만 한다[4].

간단한 순차 패턴 갱신 방법으로는 갱신된 데이터베이스를 대상으로 기존의 순차 패턴 탐사 알고리즘을 재실행하는 것이 있다. 이 방법은 간단하긴 하지만 몇 가지 단점이 있다. 갱신 후의 데이터베이스는 기존 데이터베이스에 데이터가 추가되는 것이므로, 갱신 전 데이터베이스에서의 순차 패턴과 갱신 후 데이터베이스의 순차 패턴은 상당 부분 중복될 것이다. 그러므로 기존에 이미 발견한 빈발 시퀀스를 이용하지 않고 또 다시 찾아내는 것은 연산 시간과 메모리의 낭비를 초래하게 된다.

순차 패턴 갱신 문제는 갱신 후의 데이터베이스에서 빈발 시퀀스를 찾는 것으로 요약될 수 있다. 문제를 좀 더 세분하면 다음의 경우로 나눌 수 있다.

1) 갱신되기 전 데이터베이스에서의 빈발 시퀀스가 갱신된 데이터베이스에서도 빈발 시퀀스이다.

2) 갱신되기 전 데이터베이스에서의 빈발 시퀀스가 갱신된 데이터베이스에서는 빈발 시퀀스가 아닐 수 있다.

3) 갱신되기 전의 데이터베이스에서 빈발 시퀀스가 아닌 시퀀스가 갱신된 데이터베이스에서는 빈발 시퀀스가 될 수 있다.

4) 추가된 아이템으로 인해 새로운 시퀀스가 생성될 수 있다.

위의 네 가지 경우에 대한 예는 다음과 같다. 아래 표 3은 갱신 전의 데이터베이스이다. 총 네 명의 고객이 있고 각 고객은 a~f의 아이템을 구입하였다. 표 4는 추가된 데이터베이스를 나타낸 것이다. 고객 3과 4는 갱신 전 데이터베이스에도 있던 고객이고 5는 새로 추가된 고객이다. 또한 표 4에서는 a~f 외에 아이템 g와 h가 추가되었다.

갱신 전 데이터베이스의 경우에 최소 지지도가 50%로 주어진다면 두 명 이상의 고객 시퀀스에 포함된 빈발 시퀀스를 찾아야 한다.

갱신 전 데이터베이스에서 최소 지지도를 만족하는 빈발 시퀀스는 <(a)>, <(b)>, <(c)>, <(f)>, <(b c)>, <(a)(b)>, <(a)(c)>이다. 최소 지지도가 동일하다고 할 때, 갱신 후의 데이터베이스에서의 빈발 시퀀스는 <(a)>, <(b)>, <(c)>, <(b c)>, <(e)>, <(g)>, <(a)(b)>, <(a)(c)>, <(b)(e)>, <(b)(g)>, <(e)(g)>이다. 갱신 전과 갱신 후의 빈발 시퀀스를 비교해 보면, 갱신 전에 빈발 시퀀스였던 <(a)>, <(b)>, <(c)>, <(b c)>, <(a)(b)>, <(a)(c)>는 갱신 후에도 최소 지지도를 만족시켜 빈발 시퀀스가 된다. 반면, 갱신 전에 지지도 50%로서 빈발 시퀀스였던 <(f)>는 갱신 후에는 지지도가 40%로 떨어져 더 이상 빈발 시퀀스가 아니다. <(b)(e)>는 갱신 전에는 지지도가 0%로 빈발이 아니었으나 갱신 후에는 지지도가 60%가 되어 빈발 시퀀스가 된다. <(g)>, <(b)(g)>, <(e)(g)>는 새로 추가된 아이템인 g로 인해 새롭게 발견된 시퀀스이며 최소 지지도를 만족하여 빈발 시퀀스가 된다.

표 3. 갱신 전 데이터베이스
Table 3. Database before update.

고객번호	트랜잭션 시간	구입한 아이템
1	1	a
1	4	b
1	6	c
1	7	f
2	2	a
2	3	b, c
2	5	d
2	10	f
3	8	b, c
4	9	e

표 4. 추가된 데이터베이스
Table 4. Added database.

고객번호	트랜잭션 시간	구입한 아이템
3	12	e
3	17	g
3	24	h
4	14	b
4	20	e, g
5	13	a
5	16	b, c
5	19	d
5	21	e
5	23	g

2. 갱신 알고리즘

가. 개관

본 절에서는 모든 빈발 시퀀스를 빠짐없이 찾으면서 기존 알고리즘을 재실행하는 것보다 연산 속도가 빠른 알고리즘을 제안한다.

제안하는 순차 패턴 갱신 알고리즘은 최소 지지도 이상이 되는 후보 집합을 생성할 때 AprioriAll 알고리즘과 마찬가지로 apriori-generation 알고리즘을 사용한다. 즉, 길이가 k인 시퀀스를 찾기 위해서 갱신 후의 데이터베이스에서 발견한 (k-1) 길이의 빈발 시퀀스로 후보를 생성한 후 데이터베이스를 스캔하여 각 후보 시퀀스들의 지지도를 구한 다음 빈발 시퀀스를 찾는다.

기존의 순차 패턴 탐사 알고리즘을 갱신 후의 전체 데이터베이스를 대상으로 적용한 방법과, 본 논문에서 제안한 갱신 알고리즘의 차이는, 데이터베이스를 스캔해야 하는 후보 집합의 개수이다. 제안하는 순차 패턴 갱신 알고리즘에서는 각 단계에서 생성되는 후보 시퀀스의 개수는 AprioriAll 알고리즘과 같으나, 가지치기(pruning) 방법을 사용하여 효율적으로 후보 집합의 개수를 줄일 수 있다. 가지치기(pruning)를 위해 필요한 정보는 갱신 전 데이터베이스에서의 빈발시퀀스와 이들 빈발시퀀스의 지지도, 그리고 갱신된 전체 데이터베이스가 아닌 추가된 데이터베이스에서 생성된 후보 시퀀스의 지지도이다. 이들 정보만으로도 갱신된 전체 데이터베이스에서 지지도를 구할 수 있고 따라서 빈발인지의

여부를 판단할 수 있다. 가지치기 후에도 여전히 후보 집합에 남아 있는 시퀀스들은 지지도 계산을 위해 전체 데이터베이스를 반드시 읽어야만 하는 시퀀스이다.

갱신 전 데이터베이스에서의 빈발 시퀀스를 유지하고 있을 때, k번째 단계에서 생성된 후보 시퀀스 중에서 추가된 데이터베이스와 갱신 전 빈발 시퀀스만으로 갱신 후의 빈발 시퀀스를 판단할 수 있는 경우는 다음과 같다.

1) 갱신 전 데이터베이스에서의 빈발 시퀀스는 시퀀스와 지지도를 유지하고 있으므로 추가된 데이터베이스에서의 지지도만 알면 이 둘을 더함으로써 갱신 후 데이터베이스에서의 지지도를 구할 수 있다. 갱신 후 데이터베이스에서의 지지도로 빈발 시퀀스인지 아닌지 판단할 수 있다.

2) 갱신 전 데이터베이스에는 나타나지 않았던 아이템이 포함된 후보 시퀀스는 갱신 전 데이터베이스에서의 지지도가 0이다. 따라서 추가된 데이터베이스에서의 지지도가 곧 갱신 후 데이터베이스에서의 지지도가 되므로 추가된 데이터베이스의 정보만으로 빈발인지 아닌지 판단할 수 있다.

3) 갱신 전에 빈발 시퀀스가 아니었던 시퀀스는 갱신 전 데이터베이스에서의 지지도에 대한 정보가 없다. 그런데 이들 시퀀스의 지지도는 갱신 전 데이터베이스에서 최소 지지도를 만족하기 위한 최소 개수에서 하나를 뺀 개수를 넘을 수 없다. 즉, 이 값이 갱신 전 데이터베이스에서 가질 수 있는 지지도의 최대값이다. 따라서 이 값을 갱신 전의 지지도로 가정하고, 이를 추가된 데이터베이스에서의 지지도와 더했을 때의 지지도가 갱신 후 데이터베이스에서 최소 지지도를 만족하기 위한 최소 개수 보다 작다면 갱신 후에는 빈발 시퀀스가 될 수 없다.

위의 세 가지 경우에서 1), 2)의 경우에는 각 후보 시퀀스의 갱신 후 데이터베이스에서의 지지도를 확실히 알 수 있고 따라서 빈발 시퀀스인지의 여부도 알 수 있다. 그러므로 각 단계에서 생성된 후보 시퀀스 중에서 1), 2)의 경우에 속하는 시퀀스를 제거할 수 있다. 빈발 시퀀스인 경우에는 지지도와 함께 빈발 시퀀스 집합에 포함시킨 후 후보 집합에서 제거한다. 빈발 시퀀스가 아닌 경우에는 단순히 후보 집합에서 제거한다. 3)의 경우에는 빈발 시퀀스가 아님이 확실한 시퀀스를 후보 집합에서 제거할 수 있다. 이렇게 후보 집합에 있는 시퀀스 중에서 제거 가능한 시퀀스를 모두 삭제한다.

이제 후보 집합에 남아 있는 시퀀스는 갱신 전 데이터베이스에서는 빈발 시퀀스가 아니었으나 갱신 후 데이터베이스에서 빈발 시퀀스가 될 가능성이 있는 시퀀스이다. 즉 3)의 경우에서, 최대값으로 가정한 갱신 전 데이터베이스에서의 지지도와 추가된 데이터베이스에서의 지지도를 더한 값이 갱신된 전체 데이터베이스에서 최소 지지도를 만족하기 위한 최소 개수보다 큰 경우이다. 이들 시퀀스들은 추가된 데이터베이스에서의 지지도는 정확히 알고 있으나 갱신 전 데이터베이스에서의 지지도는 가정한 값이므로, 실제 지지도를 알기 위해서는 반드시 갱신 전 데이터베이스를 스캔해야 한다. 그러나 이러한 시퀀스의 수는 최초에 생성된 후보 집합의 개수보다 상당히 줄어들게 되고 이로 인해 연산 비용에서의 이득을 얻을 수 있다. [4]에서 사용된 연관 규칙 갱신 알고리즘에서는, 위의 3)의 경우 무조건 전체 데이터베이스를 스캔하였으나, 본 알고리즘에서는 3)에서도 빈발 시퀀스가 아님이 확실한 시퀀스를 걸러내는 효율적인 가지치기 방법을 사용하였다.

나. 용어정의

본 논문에서 순차 패턴 갱신 알고리즘에 사용하는 기호는 다음과 같다.

- DB : 갱신 전의 데이터베이스
- db : DB에 더해지는 증가분의 데이터베이스
- UDB : 갱신 전 데이터베이스와 증가분의 데이터베이스가 더해진 갱신 후의 전체 데이터베이스
- S.DB : DB에서 시퀀스(또는 시퀀스 집합) S의 지지도
- S.db : db에서 시퀀스(또는 시퀀스 집합) S의 지지도
- S.UDB : UDB에서 시퀀스(또는 시퀀스 집합) S의 지지도
- DBmin_support : DB에서 최소지지도를 만족하기 위한 시퀀스의 최소 개수
- UDBmin_support : UDB에서 최소지지도를 만족하기 위한 시퀀스의 최소 개수
- L_k : DB에서의 빈발 시퀀스
- C_k : UDB에서의 후보 시퀀스
- L'_k : UDB에서의 빈발 시퀀스
- Old item : DB의 아이템 집합
- New item : db에만 있는 아이템 집합, 즉 {i|i∈db and i∉DB}

```
Input : Database DB, db, L_k, min_support
Output : The complete set of sequential patterns L'_k.
Method :
L'_k = ∅;
Scan db to generate C_1;
while (C_1 ≠ ∅) {
  Calculate L_1.UDB based on L_1.DB and C_1.db;
  for all S∈L_1
    if (S.UDB≥UDBmin_support)
      then L'_1 =L'_1∪S;
  for all S∈C_1
    if (S.db==S.UDB) //when S includes new
                     items
      then L'_1 =L'_1∪S;
  C_1 = C_1 - ( {L_1}
     ∪ {S|S∈C_1, S.db==S.UDB}
     ∪ {S|S∈C_1, (S.db+DBmin_support-1) <
                 UDBmin_support} );
  Scan DB to count the support of C_1;
  Calculate C_1.UDB based on C_1.DB and C_1.db;
  for all S∈C_1
    If (S.UDB≥UDBmin_support)
      then L'_1 =L'_1∪S;
}
for (k=2; C_k≠∅; k++) {
  Generate C_k from L'_k-1 by Apriori-generation;
  Scan db to find C_k.db;
  while (C_k≠∅) {
    for all S∈L_k
      if (S∈{L_k-1 - L'_k-1})
        then L_k =L_k - S;
    Calculate L_k.UDB based on L_k.DB and C_k.db;
    for all S∈L_k
      if (L_k.UDB≥UDBmin_support)
        then L'_k =L'_k∪S;
    for all S∈C_k
      if (C_k.db==C_k.UDB)
        then L'_k =L'_k∪S;
    C_k = C_k - ( {L_k}
       ∪ {S|S∈C_k, S.db==S.UDB}
       ∪ {S|S∈C_k, (S.db+DBmin_support-1) <
                   UDBmin_support});
    Scan DB to count the support of C_k;
    Calculate C_k.UDB based on C_k.DB and C_k.db;
    for all S∈C_k
      if (C_k.UDB≥UDBmin_support)
        then L'_k =L'_k∪S;
  }
}
```

그림 1. 제안하는 갱신 알고리즘
Fig. 1. Updating algorithm.

다. 갱신알고리즘

순차 패턴 갱신 알고리즘에서 각 단계마다 아래 과정을 수행한다.

(1) 후보 생성 및 지지도 계산

L'_{k-1}로부터 C_k를 생성한다. db에서의 C_k의 지지도를 계산한다.

(2) 가지치기 및 L'_k 생성

① L_k 갱신

• L_k 에 $(L_{k-1}-L'_{k-1})$에 속하는 서브 시퀀스, 즉 갱신 전에는 빈발이었으나 갱신 후에는 빈발이 아닌 시퀀스가 포함되어 있다면 L_k에서 미리 제거한다. 어떤 시퀀스가 빈발 시퀀스가 되기 위해서는 그 시퀀스의 모든 서브 시퀀스도 빈발 시퀀스여야 하기 때문이다. $(L_{k-1}-L'_{k-1})$에 속하는 서브 시퀀스를 포함하는 시퀀스는 L'_k에서도 빈발이 아닌 것이 확실하므로 L_k에서 제거할 수 있다.

• L_k의 시퀀스 중 C_k에도 속해있는 시퀀스가 있으면 DB에서의 지지도와 db에서의 지지도를 더해서 UDB에서의 지지도를 구한다. 갱신된 지지도가 UDBmin_support 이상인 시퀀스는 L'_k에 포함시키고 C_k에서 제거한다. 지지도가 UDBmin_support 보다 작은 시퀀스는 C_k에서 제거한다.

② C_k의 시퀀스 중에서 new item을 포함하면서 지지도가 UDBmin_support 이상인 시퀀스는 L'_k에 포함시키고 C_k에서 제거한다. New item을 포함하면서 지지도가 UDBmin_support 보다 작은 시퀀스도 C_k에서 제거한다.

③ C_k 중 갱신 후 빈발하지 않음이 확실한 시퀀스를 제거한다. 즉, DB에서의 지지도를 (DBmin_support-1)로 가정하고 이를 C_k에서의 지지도와 더했을 때의 지지도가 UDBmin_support 보다 작으면 C_k에서 제거한다.

④ C_k에 남아 있는 시퀀스들의 지지도를 갱신 전 데이터베이스 DB에서 구하여 UDB에서의 지지도가 UDBmin_support 이상인 시퀀스를 L'_k에 포함시킨다.

그림 1은 위에서 설명한 알고리즘의 의사코드이다.

IV. 실험 결과 및 분석

본 장에서는 갱신된 데이터베이스를 대상으로 본 논문에서 제안한 순차 패턴 갱신 알고리즘을 사용했을 때와 사용하지 않았을 때의 실행시간과 후보 집합의 개수를 실험을 통해 비교하고자 한다. 제 1절에서는 성능 평가를 위한 실험 환경 및 데이터 생성 방법을 설명하고, 제 2절에서 실험 결과를 보인다.

1. 성능 평가에 사용된 실험 환경 및 데이터

가. 실험환경

본 논문에서 제안된 알고리즘은 리눅스 커널 2.4 - GCC기반에서 수행하였다.

나. 데이터셋

실험에서 사용된 데이터는 인위데이터이다. 데이터셋 생성 방법은 다음과 같다. 먼저 [3]의 데이터 생성 방법에 따라 데이터를 생성한다. 이를 갱신 후의 데이터베이스인 UDB로 한다. UDB 중에서 일정 부분의 데이터는 db에 저장하고 남은 데이터는 DB에 저장한다. 실험에서 사용한 db는 UDB의 20% 비율이다.

표 5는 데이터 생성에 사용된 인자들의 리스트를 나타낸 것이다. 실험을 위해 총 7개의 데이터셋을 생성하였다. 표 6은 데이터셋 생성에 사용된 인자의 값과 데이터셋 사이즈를 나타낸 것이다.

표 5. 실험 데이터 생성에 사용된 파라미터 목록
Table 5. Parameters for generating experimental data.

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

표 6. 데이터셋
Table 6. Parameter settings.

데 이 터 셋	고객의 개수	크 기
D10C10T2.5S4	100,000	7MB
D10C10T5S4	100,000	14MB
D10C10T7.5S4	100,000	21MB
D10C15T2.5S4	100,000	10MB
D10C20T2.5S4	100,000	15MB
D20C10T2.5S4	200,000	14MB
D30C10T2.5S4	300,000	20MB

2. 실험 결과

가. 후보개수

그림 2~4에서는 지지도 계산을 위해 데이터베이스를 스캔해야 하는 후보 집합의 개수를 비교하였다. PrefixSpan은 AprioriAll이나 본 논문의 갱신 알고리즘에서 사용한 것과 같은 후보 생성과정이 없으므로 이 실험에서는 AprioriAll과 갱신 알고리즘만을 비교하였다.

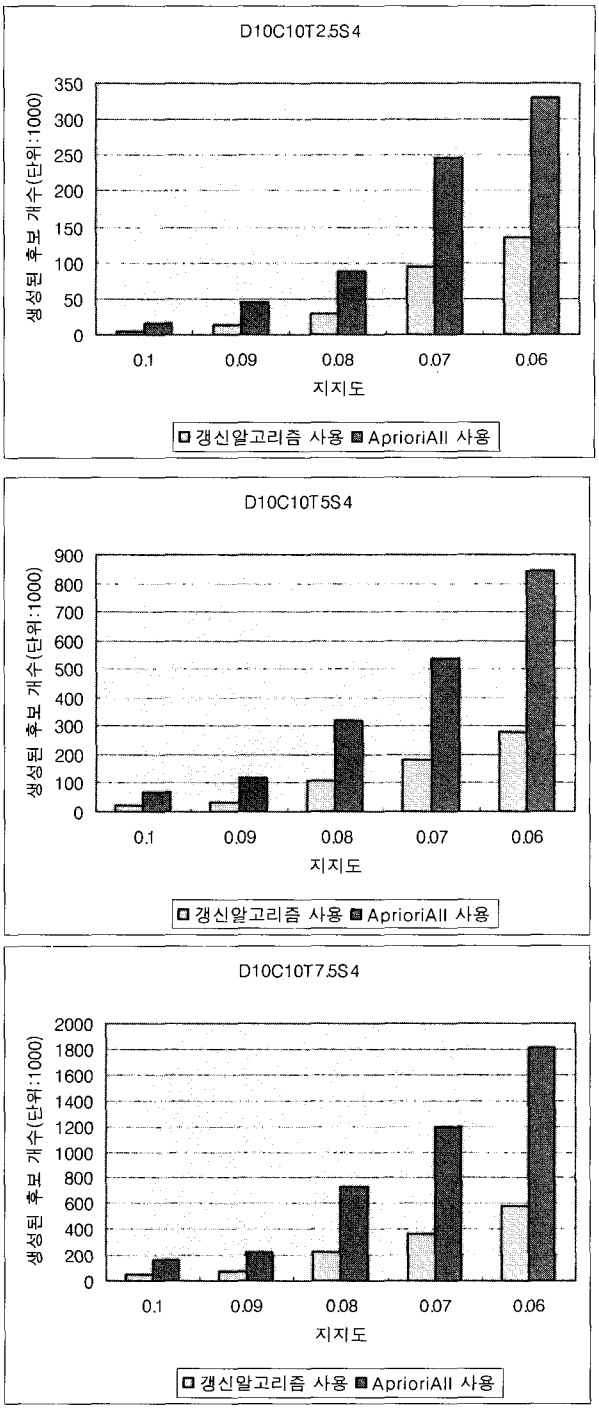
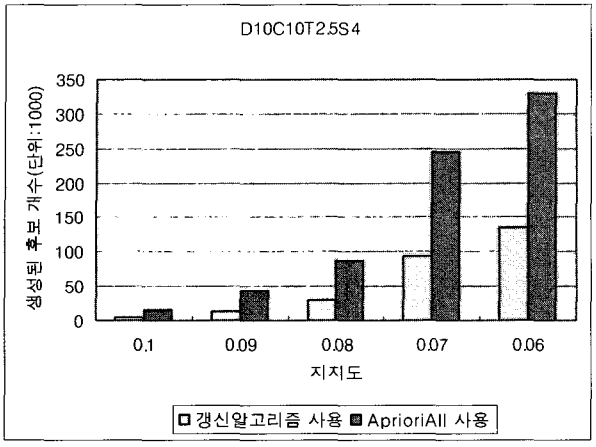
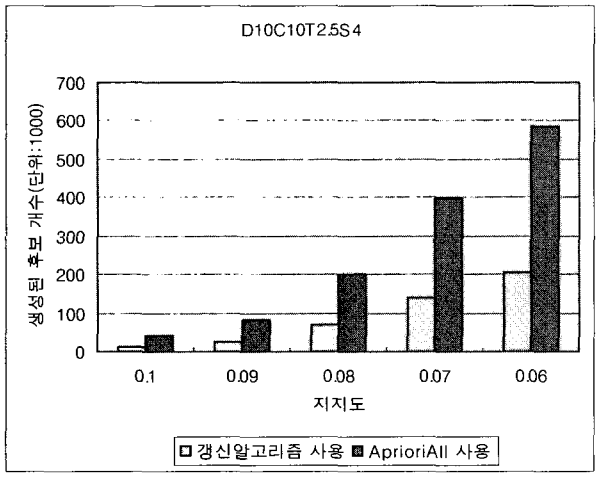


그림 2. 트랜잭션 당 아이템 개수(T)의 변화에 따른 후보 개수 변화
Fig. 2. Number of candidate set for number of items per transaction.

AprioriAll과 갱신 알고리즘은 각 단계에서 생성하는 후보 시퀀스의 개수가 같다. 그러나 갱신 알고리즘은 이렇게 생성된 후보 시퀀스 중에서 반드시 데이터베이스를 스캔해야 지지도를 알 수 있는 후보 시퀀스만 남기고 나머지는 제거하게 된다. 이 때 남겨진 후보 시퀀





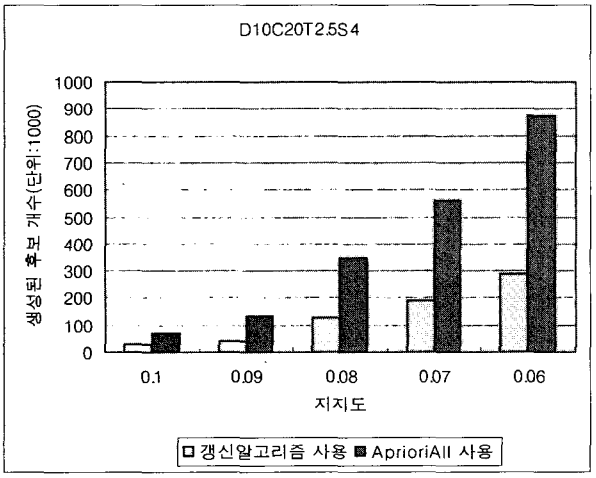
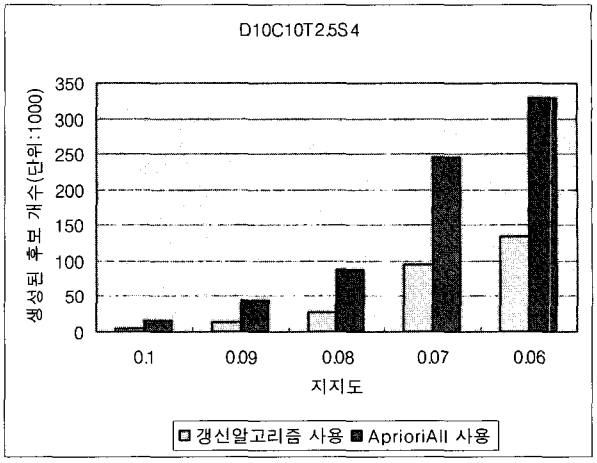
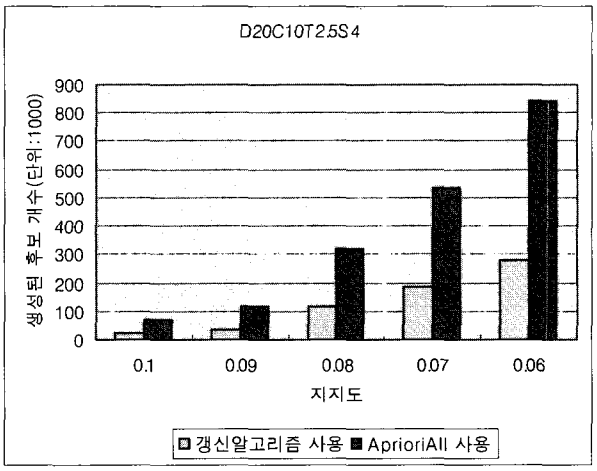


그림 3. 고객 당 트랜잭션 개수(C)의 변화에 따른 후보 개수 변화

Fig. 3. Number of candidate set for number of transactions per customer.





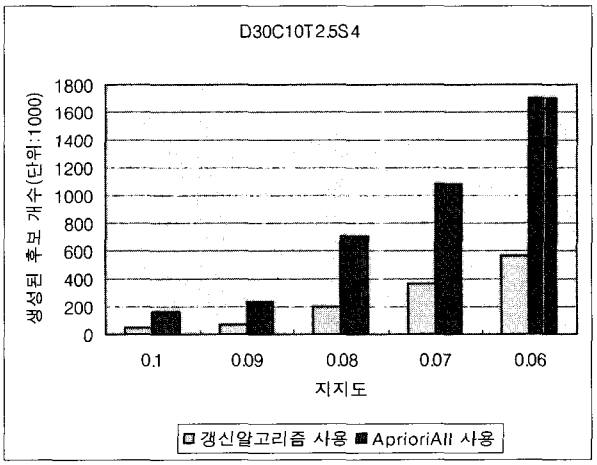


그림 4. 고객 수(D)의 변화에 따른 후보 개수 변화

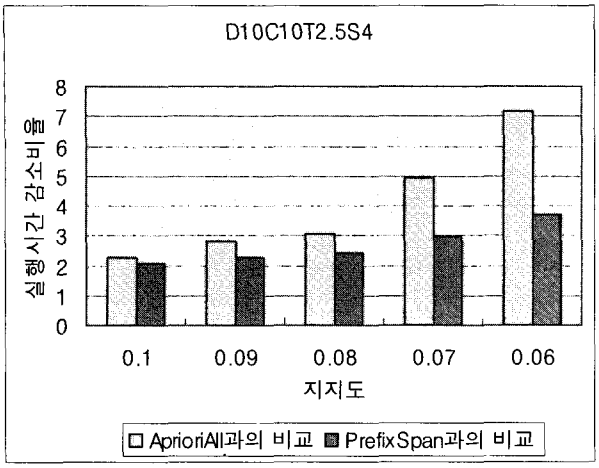
Fig. 4. Number of candidate sets for number of customers.

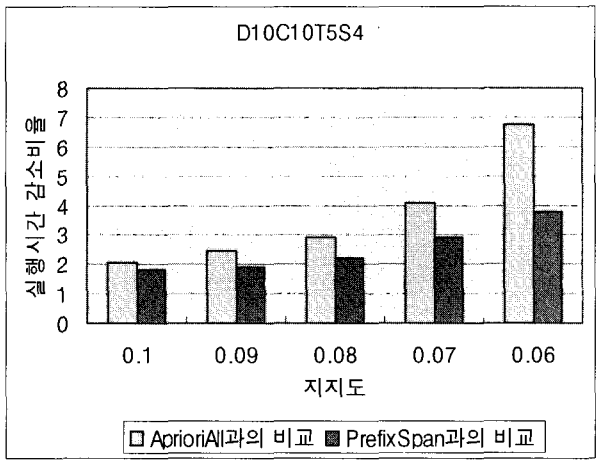
스의 개수와 최초에 생성된 후보 시퀀스의 개수를 비교하였다.

그림 2, 그림 3, 그림 4는 각각 T, C, D의 변화에 따른 후보 개수를 비교한 것이다. 각 데이터셋에서 지지도에 따라 2.5배에서 3.3배의 비율로 후보 개수가 줄어드는 것을 볼 수 있다.

나. 실행시간

본 절에서는 갱신 알고리즘을 사용했을 때와 그렇지 않을 때의 실행 시간을 비교하여 갱신 알고리즘의 시간





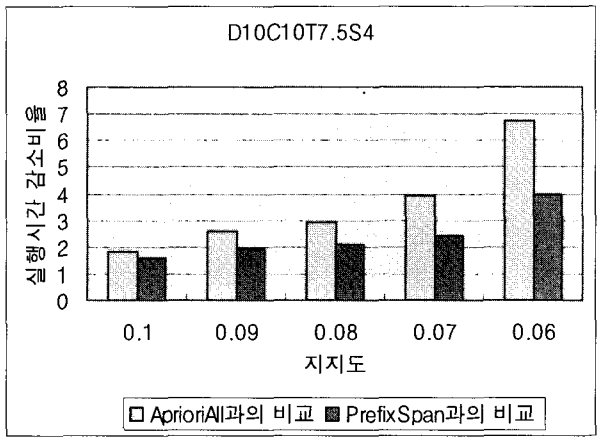
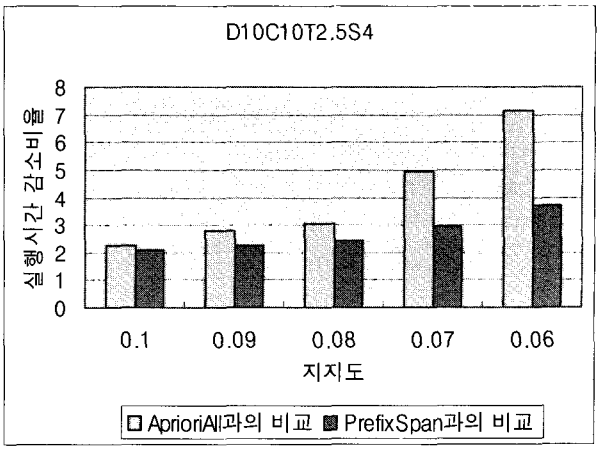
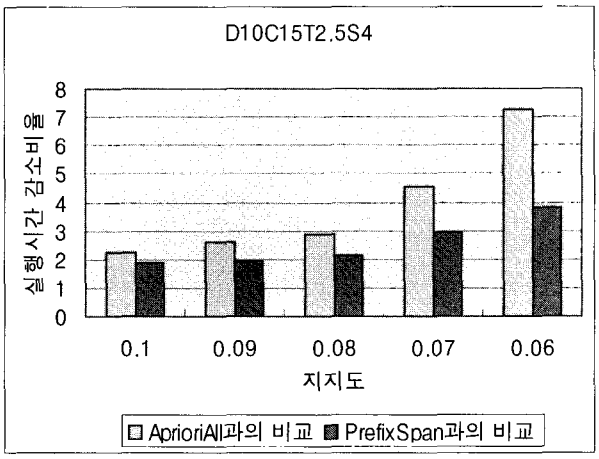


그림 5. 트랜잭션 당 아이템 개수(T)의 변화에 따른 실행시간 비교

Fig. 5. Execution times for number of items per transaction.





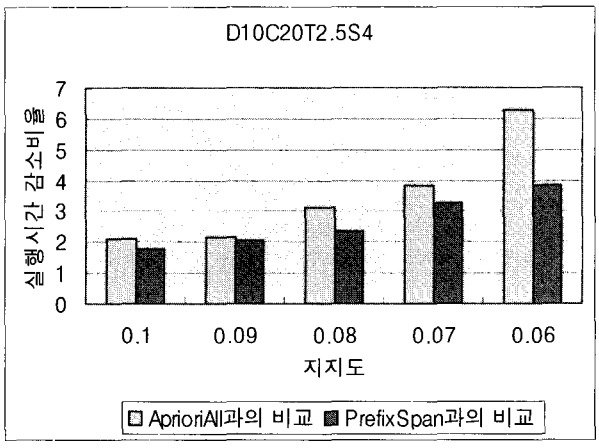


그림 6. 고객 당 트랜잭션 개수(C)의 변화에 따른 실행시간 비교

Fig. 6. Execution times for number of transactions per customer.

이득을 보여준다. 지지도는 0.1, 0.9, 0.8, 0.7, 0.6으로 차례로 변화시킨다.

그림 5는 트랜잭션 당 아이템의 평균 개수인 T를 2.5, 5, 7로 변화시켰을 때 갱신 알고리즘이 얻는 실행시간의 이득을 비율로 나타낸 그래프이다. 각 데이터셋에서 갱신 알고리즘을 사용했을 때 실행시간이 AprioriAll과 비교하면 2~7.1배 가량 감소하고, PrefixSpan과 비교하면 2~3.8배 가량 감소함을 볼 수 있다.

그림 6은 고객 당 트랜잭션의 평균 개수인 C를 10,

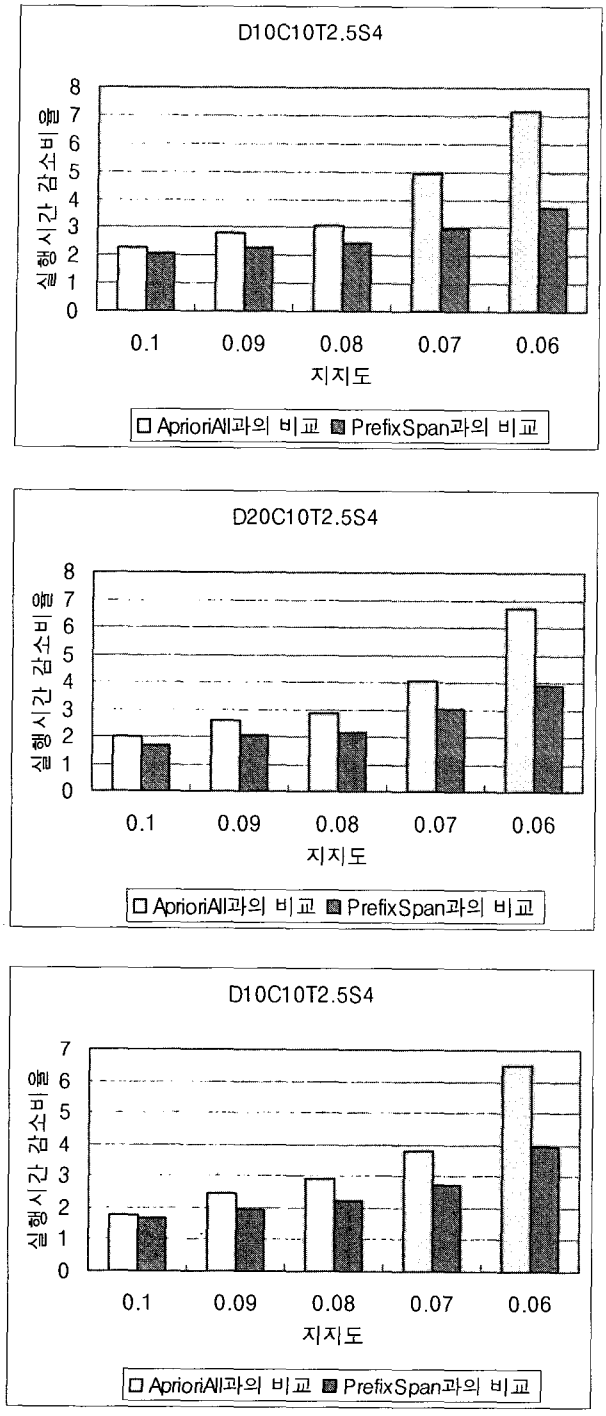


그림 7. 고객의 수(D)의 변화에 따른 실행시간 비교
Fig. 7. Execution times for number of customers.

15, 20으로 변화시켰을 때 실행시간의 비율을 나타낸 것이다. AprioriAll과 비교했을 때 2~7.1배 정도 실행시간이 감소했고, PrefixSpan과 비교했을 때는 1.7~3.8배 정도 실행시간이 감소했다.

그림 7은 고객의 수 D를 100,000, 200,000, 300,000으로 변화시켰을 때 실행시간의 비율을 나타낸 것이다. AprioriAll과 비교시 2~7.2배 정도 실행시간이 감소했고, PrefixSpan과 비교시 1.8~4배 가량 실행시간이 감소하였음을 볼 수 있다.

PrefixSpan은 후보 시퀀스의 지지도 계산을 위한 데이터베이스 스캔 과정이 필요 없기 때문에 AprioriAll보다 실행시간이 줄어든다. 따라서 PrefixSpan과 비교했을 때 갱신 알고리즘이 얻는 이득은 AprioriAll과 비교했을 때 보다 감소하나, 이 경우에도 갱신 알고리즘을 사용하는 것이 최대 3배의 이득을 보임을 확인하였다.

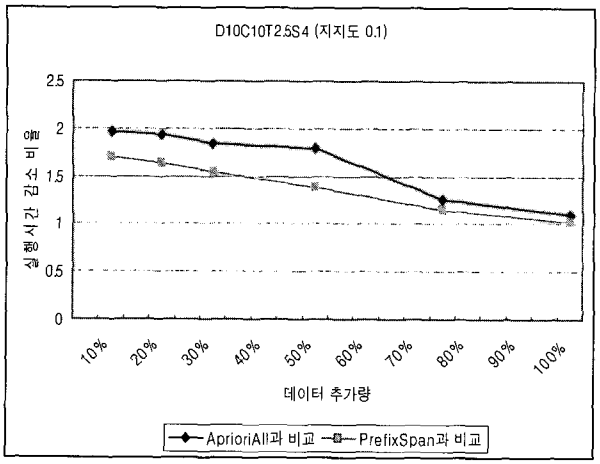
또한 그림 5, 그림 6, 그림 7의 결과를 보면 지지도가 작아짐에 따라 실행시간의 감소 비율이 점점 증가하였다. 지지도가 높으면 빈발 시퀀스의 수가 상대적으로 적기 때문에 어떤 알고리즘을 사용하든지 실행 시간이 큰 차이를 보이지 않는다. 반면 지지도가 낮아질수록 실행시간은 급격히 증가한다. AprioriAll의 경우 지지도가 낮으면 길이 1인 빈발 시퀀스의 개수가 많아지고 이로 인해 생성되는 길이 2인 후보 시퀀스의 개수도 급격히 증가한다. 실행 시간의 대부분은 후보 시퀀스의 지지도를 계산하는 데 소비되기 때문에 지지도가 작아질수록 실행시간은 지수적으로 증가하게 된다. PrefixSpan의 경우에도 마찬가지로 지지도가 작을 때 많은 수의 빈발 시퀀스가 존재하므로 전치(prefix)와 후치(postfix)로 구성된 중간 단계의 데이터베이스를 생성하는 데 많은 비용이 소비된다. 갱신 알고리즘을 사용하면 데이터베이스를 스캔하여 지지도를 계산해야 하는 후보의 개수가 줄어들게 되므로 지지도에 따른 실행시간의 증가율이 갱신 알고리즘을 사용하지 않을 때에 비해 작다. 따라서 지지도가 작아질수록 갱신 알고리즘이 얻는 실행 시간의 이득이 점점 커지게 된다.

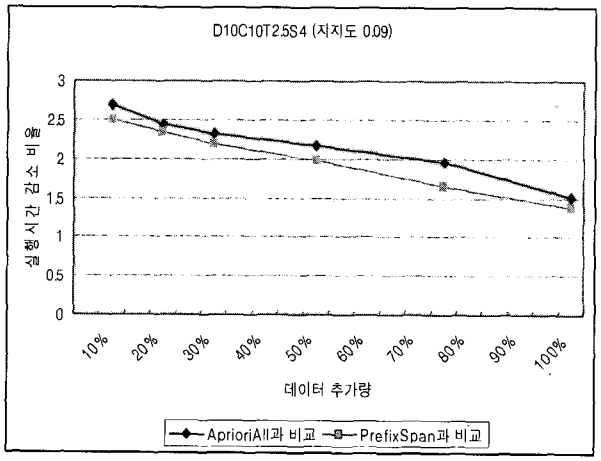
그림 2, 3, 4와 그림 5, 6, 7의 결과에서 데이터베이스를 스캔해야 하는 후보 시퀀스의 개수가 실행시간에 영향을 미치는 것을 확인할 수 있다. 또한 지지도에 따른 후보 개수의 차이보다 실행 시간의 차이가 큰 것으로 보아 지지도가 낮을수록 후보 개수의 감소량이 실행 시간의 큰 감소를 가져오는 것을 알 수 있다.

다. 데이터 추가량에 따른 실행시간

그림 8에서는 추가되는 데이터 양에 따른 실행시간의 변화를 측정하여 각각 AprioriAll, PrefixSpan과 비교하였다. 실험에 쓰인 데이터셋은 D10C10T2.5S4이고, 추가된 데이터의 양은 10%, 20%, 30%, 50%, 75%, 100%이다.

그림 8에서 볼 수 있듯이, 추가되는 데이터의 양이 많을수록 실행시간 감소 비율은 점차 감소하였다.





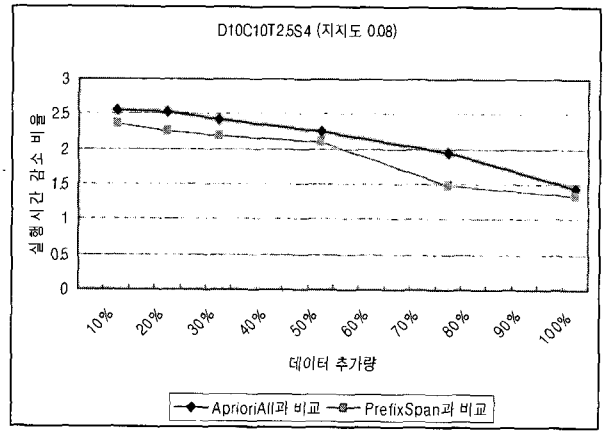


그림 8. 데이터 추가량에 따른 실행 시간 감소 비율
Fig. 8. Execution time reduction ratio for added data.

AprioriAll과 PrefixSpan 모두 데이터 크기에 따라 실행 시간이 선형적으로 증가한다. 반면 갱신 알고리즘에서는 추가되는 데이터의 양이 많으면 추가된 데이터베이스에서의 지지도를 구하는 시간이 증가하게 되므로 갱신 알고리즘을 사용하는 이득이 줄어들게 된다. 그러나 추가된 데이터의 양이 100%인 경우에도 여전히 실행 시간은 감소함을 볼 수 있다.

V. 결 론

순차 패턴 탐사는 데이터 마이닝의 중요 기법 중 하나이다. 데이터베이스는 계속해서 데이터가 추가되어 갱신되므로 효율적인 순차 패턴 탐사 알고리즘의 개발 못지않게 발견된 순차 패턴을 갱신하는 것도 중요한 문제이다.

본 논문에서는 순차 패턴 갱신 알고리즘에 의해 발견된 빈발 시퀀스를 갱신하는 방법에 대해 논의하였다.

데이터 마이닝의 연산 시간은 데이터베이스를 스캔하여 각 고객 시퀀스와 후보 시퀀스를 비교하는 연산이 크게 좌우하므로 후보 시퀀스를 줄여 전체 연산 시간을 빠르게 하는 것이 이 연구의 목적이다.

본 연구에서 제안하는 방법은 AprioriAll에서 후보 시퀀스 생성 방법으로 사용한 apriori-generation 알고리즘을 이용하여 후보를 생성한 후, 데이터베이스 갱신 전에 발견한 빈발 시퀀스와 지지도, 그리고 추가된 데이터베이스에서의 지지도 정보를 토대로 갱신 후 데이터베이스에서의 지지도를 계산한 후 빈발 시퀀스를 찾아내는 것이다.

제안하는 갱신 알고리즘의 성능 평가를 위해 갱신된 데이터베이스에 기존 알고리즘을 적용시켰을 때와 비교하여 실험한 결과, 갱신 알고리즘을 사용했을 때 전체 데이터베이스를 스캔해야 하는 후보 개수는 최대 3배의 감소를 보였고, 실행 시간 면에서는 최대 7배의 실행 시간 감소 효과를 볼 수 있었다. 실행 시간의 차이는 데이터베이스를 스캔해서 지지도를 구해야 하는 후보 시퀀스의 개수에서 비롯된다. 최근에 AprioriAll보다 실행 시간이 향상된 PrefixSpan 알고리즘과 비교했을 때도 최대 약 4배 이상의 성능향상을 보임으로써 특정 알고리즘과 관계없이 성능이 개선됨을 확인할 수 있었다.

또한 추가된 데이터 양에 따른 실행시간을 측정해본 결과, 추가된 데이터양이 적을수록 연산 속도 면에서의 이익이 큰 것을 볼 수 있었다. 따라서 본 논문에서 제안하는 알고리즘은 데이터베이스에 새로운 트랜잭션이 더해지는 상황이 빈번할 경우, 혹은 짧은 주기로 순차 패턴을 갱신해야 할 경우에 특히 효과적이다.

본 논문에서는 트랜잭션이 추가된 경우에 대해서만 다루고 있으나 데이터베이스에 갱신과 삭제가 일어났을 경우의 순차 패턴 갱신 알고리즘에 대한 연구도 계속해서 진행할 예정이다.

참 고 문 헌

[1] M.-S. Chen, J. Han, and P.S. Yu, "Data Mining: An Overview from a Database Perspective," IEEE Transactions on knowledge and Data Engineering, Vol. 8, No. 6, pp. 866-883, Dec. 1996.

[2] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules in Large Databases," Proc. of ACM SIGMOD Conference on Management of Data, Washington D.C., pp. 207-216, May 1993.

[3] R. Agrawal and R. Srikant, "Mining Sequential Patterns," Proc. of the 11th International Conference on Data Eng. (ICDE'95), Taipei, Taiwan, March, 1995.

[4] D. W. Cheung, Jiawei Han, Vincent T. Ng, C.Y. Wong, "Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique," Proc. of the 12th Int. Conf. on Data Eng. (ICDE'96), 1996.

[5] J. Pei, J. Han, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth," Proc. 2001 Int. Conf. on Data Eng. (ICDE'01), Heidelberg, Germany, April 2001.

저 자 소 개

김 학 자(정회원)
1998년 강원대학교 정보통신 공학과 학사 졸업.
2002년 강원대학교 컴퓨터정보 통신공학과 석사 졸업
<주관심분야 : 데이터베이스, 데이터 마이닝>

황 환 규(정회원)-교신저자
대한전자공학회 논문지 제 41권 CI편 제 2호 참조