효율적인 순차 패턴 갱신 알고리즘

김학자° 김형근 황환규
강원대학교 데이터베이스 연구실
{lucia76@orgio.net°, harby77@hanmail.net, wkwhang@cc.kangwon.ac.kr}

Efficient Update Algorithm of Sequential Pattern

Hakja Kim° Hyunggeun Kim WhanKyu Whang
Dept. of Information & Telecommunication, Kangwon National University

요 약

본 논문은 순차 패턴을 갱신하는 알고리즘을 제안한다. 갱신된 데이터베이스에서 새로운 순차 패턴을 찾는 비용을 줄이기 위해 갱신 전 데이터베이스에서 발견한 순차 패턴에 대한 정보와 추가되는 데이터베이스의 정보만으로 새로운 순차 패턴의 후보를 줄이는 방법으로, 갱신된 전체 데이터베이스를 대상으로 순차 패턴 마이닝 알고리즘을 재실행하는 방법에 비해 후보 셋이 줄어들고 이로 인해 연산 비용을 줄일 수 있는 장점이 있다.

1. 서 론

컴퓨터 시스템의 발달과 데이터베이스 시스템 사용의 증가로 컴퓨터에 저장되는 데이터의 양은 폭발적으로 증가하고 있다. 현재 컴퓨터에 저장되어 있는 대용량 데이터베이스에는 사용자가 미처 파악하지 못하는 중요한 정보가 포함되어 있을 수 있다. 이렇게 감추어져 있지만 효용가치가 큰 패턴이나 정보를 찾아내는 연구가 1990년대 초기부터 활발하게 이루어지고 있다. 대용량 데이터베이스에서의 지식 발견(knowledge discovery in database)이라고 정의되는 데이터 마이닝[1](data mining) 연구 중에서 연관 규칙 탐사[2]에 대해 가장 많은 연구가 이루어졌고 그 결과 새로운 패턴을 찾아내고 있는데[3] 그 중 하나가 순차 패턴 탐사이다.

순차 패턴(Sequential Pattern) 탐사[4]는 한 트랜잭션 안에서 발생하는 항목들 간의 연관 규칙에 시간의 변이를 추가한 것이다. 예를 들어 비디오 대여점에서 사용하는 데이터베이스에서의 순차 패턴의 예는 "비디오 대여점에서 30%의 고객은 ' Star Wars' ⇒ 'Empire Strikes Back' ⇒ 'Return of the Jedi ' 의 순서로 비디오를 대여한다." 는 것이다. 이렇게 소비자의 구매 패턴을 찾는 것 외에도 의료 분야에서 순차 패턴을 이용한 응용을 보면 환자들에 대한 질병과 투약에 관한 데이터를 이용해 일정 지지도 이상을 갖는 특정 질병에 대한 투약 과정을 순차 패턴으로 표현할 수 있다면 해당 질병 치료에 많은 도움을 주게 되고 의사는 더 좋은 진료와 치료를 할 수 있게 된다[3].

고객 번호와 트랜잭션이 발생한 시간, 그 트랜잭션에 포함되어 있는 항목에 대한 정보가 포함된 데이터베이스에서, 각 고객들의 트랜잭션을 시간 순서로 정렬한 것을 고객 시퀀스(customer sequence)라고 한다. 고객 시퀀스는 <아이템셋 (T1) 아이템셋 (T2) … 아이템셋 (Tn)>의 시퀀스이다. 시퀀스가 특정 고객의 고객 시퀀스에 속해 있다면 그 고객은 이 시퀀스를 지지한다고 말한다. 시퀀스의 지지도의 정의는 전체 고객에 대한 시퀀스를 지지하는 고객의 비율이다. 순차 패턴 탐사는 사용자가 정의한 최소 지지도를 만족하는 모든 시퀀스들 사이에서 최대 시퀀스를 찾는 것이다. 이러한 각각의 최대 시퀀스들을 순차 패턴(sequential pattern)이라 하고, 최소 지지도를 만족하는 시퀀스를 빈발 시퀀스(frequent sequence)라고 부른다. 순차 패턴 탐사에서는 이러한 빈발 시퀀스를 효율적으로 찾는 것이 중요한 문제가 된다.

발견된 순차 패턴은 데이터베이스의 현재 상태만 반영한다. 데이터베이스는 지속적으로 트랜잭션이 추가되어 갱신되므로 패턴 또한 갱신되어야만 한다. 패턴을 갱신하는 한 가지 방법은 갱신 후의 전체 데이터베이스를 대상으로 순차 패턴 알고리즘을 재실행 하는 것이다. 그러나 이 방법은 갱신 전에 이미 발견한 패턴들을 다시 발견하게 되므로 연산비용의 낭비가 있다[5].

본 논문에서는 갱신 전에 발견한 패턴을 재이용하고 새로운 빈발 시퀀스를 찾기 위한 후보 집합의 수를 줄임으로써 효율적으로 순차 패턴을 갱신하는 알고리즘을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서 순차 패턴 갱신의 문제를 개괄적으로 서술하고, 3장에서 구체적인 갱신 알고리즘을 예를 들어 설명한 다음 4장에서 결론으로 마무리한다.

2. 개 관

순차 패턴 갱신 문제는 갱신 후 데이터베이스에서 빈발 시퀀스를 생성하는 것으로 요약된다. 문제를 좀 더 세분하면 다음의 경우로 나눌 수 있다. ① 갱신 전의 빈발 시퀀스는 갱신 후에는 빈발 시퀀스가 아닐 수 있다. ② 갱신 전에 빈발 시퀀스가 아니었던 시퀀스가 갱신 후에는 빈발 시퀀스가 될 수 있다. ③ 추가된 데이터베이스로 인해 새로운 시퀀스가 생성될 수 있다.

예를 들어 표 1의 데이터베이스에 표 2와 같은 데이터베이스가 추가되었을 때 빈발 시퀀스의 변화는 다음과 같다.

최소지지도가 50%라면, 갱신 전 데이터베이스에서의 빈발 시퀀스는 <(I1)>, <(I2)>, <(I3)>, <(I6)>, <(I2 I3)>, <(I1)(I2)>, <(I1)(I3)>이고 갱신 후의 빈발 시퀀스는 <(I1)>, <(I2)>, <(I3)>, <(I2 I3)>, <(I5)>, <(I7)>, <(I1)(I2)>, <(I1)(I3)>, <(I2)(I5)><(I2)(I7)>, <(I5)(I7)>이다. 갱신 전에 빈발 시퀀스였던 <(I1)>, <(I2)>, <(I3)>, <(I2 I3)>, <(I1)(I2)>, <(I1)(I3)>은 갱신 후에도 빈발 시퀀스이다. 반면 갱신 전에 빈발 시퀀스였던 <(I6)>은 갱신 후에는 더 이상 빈발 시퀀스가 아니다. <(I2)(I5)>는 갱신 전에는 빈발이 아니었으나 갱신 후

에는 빈발 시퀀스가 된다. <(I7)>, <(I2)(I7)>, <(I5)(I7)>은 갱신 후에 새롭게 발견된 시퀀스이다.

표1. 갱신 전 데이터베이스

Customer-Id	Time	Items
C1	1	I1
C1	4	I2
C1	6	I3
C1	7	I6
C2	2	I1
C2	3	I2, I3
C2	5	I4
C2	10	I6
C3	8	I2, I3
C4	9	I5

표2. 추가된 데이터베이스

Customer-Id	Time	Items
C3	12	I5
C3	17	I7
C3	24	I8
C4	14	I2
C4	20	I5, I7
C5	13	I1
C5	16	I2, I3
C5	19	I4
C5	21	I5
C5	23	I7

제안하는 순차 패턴 갱신 알고리즘은 최소 지지도 이상이 되는 후보 집합을 생성하는 방법에 있어서 AprioriAll 알고리즘[4]과 유사하다. 즉, 길이가 k인 시퀀스를 찾기 위해서 (k-1) 길이의 빈발 시퀀스로 후보를 생성한 후 데이터베이스를 스캔하여 각 후보 시퀀스들의 지지도를 구한 다음 빈발 시퀀스를 찾는다. 그러나 갱신 알고리즘에서는 갱신 전 빈발 시퀀스와 추가된 데이터베이스를 이용하여 후보 셋의 집합을 줄일 수 있다.

k번째 단계에서 생성된 후보 셋 중에서, 추가된 데이터베이스와 갱신 전 빈발 시퀀스만으로 갱신 후의 빈발 시퀀스를 판단할 수 있는 경우는 다음과 같다.

① 갱신 전 빈발 시퀀스는 추가된 데이터베이스에서의 지지도만 알면 갱신 후 데이터베이스에서의 지지도를 구할 수 있다.

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

③ 갱신 전에 빈발 시퀀스가 아니었던 시퀀스의 지지도는 (갱신 전 데이터베이스에서 최소 지지도를 만족하는 최소 갯수-1)을 넘을 수 없다. 따라서 이 값을 갱신 전의 지지도로 가정하고, 추가된 데이터베이스에서의 지지도와 더했을 때의 지지도가 갱신 후 데이터베이스에서 최소 지지도를 만족하는 최소 개수 보다 작다면 갱신 후에는 빈발 시퀀스가 아니다. 따라서 이러한 시퀀스도 후보 셋에서 제거할 수 있다.

후보 셋에서 위의 경우에 해당하는 시퀀스를 제외한 나머지 시퀀스는 갱신 전에는 빈발이 아니었으나 갱신 후 데이터베이스에서 빈발 시퀀스가 될 가능성이 있는 시퀀스로서 실제 지지도를 알기 위해서는 갱신 전 데이터베이스를 스캔해야 한다. 그러나 이 수는 최초에 생성된 후보 셋의 개수보다 상당히 줄어들게 되고 이로 인해 연산 비용에서의 이득을 얻을 수 있다.

3. 순차 패턴 갱신 알고리즘

위의 관찰을 바탕으로 순차 패턴 갱신 알고리즘을 제안한다. 3.1절에서는 갱신 알고리즘에서 쓰이는 용어를 정리한다. 3.2절에서 후보 셋의 지지도 계산하는 방법을 서술하고, 3.3절에서 갱신 알고리즘을 설명한 다음 3.4절에서 구체적인 예를 보이겠다.

3.1 용어 정의

DB를 갱신 전의 데이터베이스라 할 때, db는 DB에 더해지는 증가분의 데이터베이스이다. UDB는 갱신 후의 전체 데이터베이스를 말한다. S를 시퀀스라 하면, $S.support_{DB}$, $S.support_{db}$, $S.support_{UDB}$는 각각 DB, db, UDB 데이터베이스에서의 지지도를 의미한다. DBmin_support, UDBmin_support는 각각 DB와 UDB에서 최소지지도를 만족하는 시퀀스의 최소 개수를 말한다. L_k는 DB에서의 빈발 시퀀스이고, C_k는 UDB에서의 후보 시퀀스, L'_k는 UDB에서의 빈발 시퀀스이다. Old item은 DB에서 한번이라도 나타났던 아이템이며, new item은 DB에는 나타나지 않았던 아이템, 즉 db에 추가된 아이템 중 old item이 아닌 아이템이다. <(Ii)s>는 시퀀스 <(Ii)>에 대한 지지도 s를 나타낸다.

3.2 C_k 지지도 계산

L'_{k-1}로부터 생성된 C_k의 지지도는 추가된 데이터베이스에서의 지지도이다. 순차 패턴의 지지도 계산 정의에 의하여 한 고객 시퀀스 내에 같은 패턴이 여러 번 나타나도 지지도는 1만 증가한다. 따라서 아이템이 추가된 고객의 이전 시퀀스에 이미 같은 패턴이 있다면 지지도를 증가시키지 않는다. 예를 들어 어떤 고객의 갱신 전 시퀀스가 <(I1)(I2 I4)>이고, 여기에 <(I1)(I5)>가 추가되었을 때 추가된 (I1)은 (I1)의 지지도에 영향을 미치지 않는다. 반면 갱신 전 시퀀스가 <(I2)(I3 I4)>이고 <(I1)(I6)>가 추가된다면 추가된 (I1)의 지지도는 1만큼 증가한다.

추가된 시퀀스가 이전 시퀀스와 결합되어 새로운 시퀀스가 생성되는 경우에도 지지도를 증가시킨다. 예를 들어, Ck에 <(I2)(I3)>가 있고 갱신 전 시퀀스가 <(I1 I2)>일 때, 추가되는 시퀀스가 <(I3)>이면 <(I2)(I3)>의 지지도를 증가시킨다. <(I2)(I3)>이 추가되었을 때도 마찬가지로 지지도를 증가시킨다.

3.3 갱신 알고리즘

(1) L'_{k-1}로부터 C_k를 생성하고 갱신이 일어난 고객의 시퀀스를 대상으로 C_k의 지지도를 계산한다.

(2) C_k중 L_k에 속한 시퀀스가 있으면 해당 시퀀스의 지지도를 갱신하여 UDBmin_support 이상인 시퀀스를 L'_k에 포함시키고 C_k에서 제거한다. 이 때 $(L_{k-1}-L'_{k-1})$에 속하는 시퀀스, 즉 갱신 전에는 빈발이었으나 갱신 후에는 빈발이 아닌 시퀀스는 L'_k에서도 빈발이 아니므로 이러한 시퀀스를 포함하는 시퀀스는 L_k에서 미리 제거한다. C_k에 속하지 않은 L_k 중에서 UDBmin_support 이상인 시퀀스도 L'_k에 포함시키고 C_k에서 제거한다.

(3) C_k에서 new item을 포함하는 시퀀스를 C_k에서 제거하고, 이 중 지지도가 UDBmin_support 이상인 시퀀스는 L'_k에 포함시킨다.

(4) 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.

(5) C_k에 남아 있는 시퀀스들의 지지도를 갱신 전 데이터베이스 DB에서 구하여 지지도가 UDBmin_support 이상인 시퀀스를 L'_k에 포함한다.

다음 절에서 표 1과 2로부터 L'1과 L'2를 생성하는 구체적인 예를 보이겠다.

3.4 예제

표2에서 C1을 구하면 다음과 같다.

C1	시퀀스	<(I1)>	<(I2)>	<(I3)>	<(I2 I3)>	<(I4)>
	지지도	1	2	1	1	1
	시퀀스	<(I5)>	<(I7)>	<(I5 I7)>	<(I8)>	
	지지도	2	3	1	1	

갱신 전 L1은 $<(I1)_2>$ $<(I2)_3>$ $<(I3)_3>$ $<(I2\ I3)_2>$ $<(I6)_2>$이고 이 중 C1에 속한 시퀀스인 $<(I1)_2>$ $<(I2)_3>$ $<(I3)_3>$ $<(I2\ I3)_2>$는 L1의 지지도와 C1에서의 지지도를 합산한다. 이 중 최소 지지도 이상인 것은 $<(I1)_3>$ $<(I2)_5>$ $<(I3)_4>$ $<(I2\ I3)_3>$이므로 L'1에 넣고 C1에서 제거한다. 이 과정을 마친 후의 L'1과 C1은 다음과 같다.

L'1	시퀀스	<(I1)>	<(I2)>	<(I3)>	<(I2 I3)>
	지지도	3	5	4	3

C1	시퀀스	<(I4)>	<(I5)>	<(I7)>	<(I5 I7)>	<(I8)>
	지지도	1	2	3	1	1

표1과 2에서 new item은 (I7)과 (I8)이다. 이 중 UDBmin_support 이상인 것은 $<(I7)_3>$이므로 L'1에 포함시키고 new item을 포함하는 모든 시퀀스를 C1에서 제거한다. 변경된 L'1과 C1은 다음과 같다.

L'1	시퀀스	<(I1)>	<(I2)>	<(I3)>	<(I2 I3)>	<(I7)>
	지지도	3	5	4	3	3

C1	시퀀스	<(I4)>	<(I5)>
	지지도	1	2

이제 C1에 남은 <(I4)>와 <(I5)>는 갱신 전 DB에서는 빈발 시퀀스가 아니었으므로 DB에서 최대 가질 수 있었던 지지도는 1이다. 따라서 <(I4)>는 C1에서의 지지도인 1을 더해도 UDB에서의 지지도가 2를 넘지 못하므로 UDB에서도 빈발 시퀀스가 아니다. 시퀀스 <(I5)>는 DB에서의 지지도를 1로 가정하고 C1의 지지도 2와 더하면 3이므로 L'1의 후보이다.

<(I5)>의 DB에서의 실제 지지도를 계산하기 위해 DB를 스캔한다. DB에서의 지지도가 1이므로 최종 지지도가 3이 되어 L'1에 포함된다. 아래 표는 최종 L'1을 나타낸 것이다.

L'1	시퀀스	<(I1)>	<(I2)>	<(I3)>	<(I2 I3)>	<(I5)>	<(I7)>
	지지도	3	5	4	3	3	3

다음으로, L'2를 구하기 위해 L'1으로부터 C2를 생성한다.

C2	<(I1)(I2)>	<(I1)(I3)>	<(I1)(I2 I3)>	<(I1)(I5)>
	1	1	1	1
	<(I1)(I7)>	<(I2)(I3)>	<(I2)(I2 I3)>	<(I2)(I5)>
	1	0	0	3
	<(I2)(I7)>	<(I3)(I2 I3)>	<(I3)(I5)>	<(I3)(I7)>
	3	0	2	2
	<(I2 I3)(I5)>	<(I2 I3)(I7)>	<(I5)(I7)>	<(I2)(I1)>
	2	2	3	0
	<(I3)(I1)>	<(I2 I3)(I1)>	<(I5)(I1)>	<(I7)(I1)>
	0	0	0	0
	<(I3)(I2)>	<(I2 I3)(I2)>	<(I5)(I2)>	<(I7)(I2)>
	0	0	1	0
	<(I2 I3)(I3)>	<(I5)(I3)>	<(I7)(I3)>	<(I5)(I2 I3)>
	0	0	0	0
	<(I7)(I2 I3)>	<(I7)(I5)>		
	0	0		

갱신 전 데이터베이스 DB에서 구한 L2는 $<(I1)(I2)>_2$ $<(I1)(I3)>_2$ 이다. C2에서 L2에 속한 $<(I1)(I2)>_2$ $<(I1)(I3)>_2$ 의 지지도를 갱신한다. 두 시퀀스 모두 UDB에서의 지지도가 3이므로 L'2에 포함시키고 C2에서 삭제한다.

New item인 (I7)을 포함하는 시퀀스 중 지지도가 UDBmin_support 이상인 $<(I2)(I7)>_3$과 $<(I5)(I7)>_3$을 L'2에 포함시키고 (I7)을 포함하는 모든 시퀀스를 C2에서 삭제한다.

마지막으로 UDBmin_support을 만족하려면 C2에서의 지지도가 2 이상이어야 하므로 1이하인 시퀀스는 모두 제거한다. C2에서 남은 $<(I2)(I5)>_3$ $<(I3)(I5)>_2$ $<(I2\ I3)(I5)>_2$ 시퀀스의 DB에서의 지지도를 구하여 UDBmin_support 이상인 것은 L'2에 넣는다. 최종 L'2는 $<(I1)(I2)>_3$ $<(I1)(I3)>_3$ $<(I2)(I5)>_3$ $<(I2)(I7)>_3$ $<(I5)(I7)>_3$ 이다.

Apriori-Gen[4] 알고리즘에 의해 L'2에서 C3는 생성되지 않으므로 마이닝은 종료된다.

이상의 예에서 최초 생성된 C1의 개수는 9개였으나 L'1을 구하기 위해 갱신 전 데이터베이스 DB에서 지지도를 구해야 했던 시퀀스는 <(I5)> 하나이다. 또, 최초에 생성된 C2의 개수는 30개였으나 L'2를 구하기 위해서 갱신 전 데이터베이스까지 읽어야 했던 시퀀스는 4개로 줄어들었다.

4. 결 론

데이터베이스는 지속적으로 트랜잭션이 추가되어 갱신되므로 순차 패턴 또한 주기적으로 갱신해야 하는데, 갱신된 트랜잭션은 기존 데이터베이스에 비하면 작은 비율이다. 따라서 많은 양의 갱신 전 데이터베이스에서의 연산 보다는 추가된 작은 양의 데이터베이스만으로 순차 패턴의 후보를 줄인 다음, 반드시 갱신 전 데이터베이스에서 지지도를 확인해야 하는 후보에 대해서만 지지도를 계산하는 방법을 제안하였다. 갱신된 전체 데이터베이스를 대상으로 순차 패턴 마이닝 알고리즘을 재실행하는 방법에 비해 후보 셋이 상당히 줄어들고 이로써 연산비용을 줄일 수 있다.

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", In Proceedings of ACM SIGMOD Conference on Management of Data, Washington D.C., pp. 207-216, May 1993.

[3] 박종수, 유원경, 홍기형, "연관 규칙 탐사와 그 응용", 한국정보과학회, 1998.

[4] R. Agrawal and R. Srikant, "Mining sequential patterns", In Proceedings of the 11th International Conference on Data Engineering, Taipei, Taiwan, March, 1995.

[5] David W. Cheung, Jiawei Han, Vincent T. Ng, C.Y. Wong, "Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique."