(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2017-0111608
(43) 공개일자 2017년10월12일

(51) 국제특허분류(Int. Cl.)
G06F 17/30 (2006.01)
(52) CPC특허분류
G06F 17/30539 (2013.01)
G06F 17/30227 (2013.01)
(21) 출원번호 10-2016-0037422
(22) 출원일자 2016년03월29일
심사청구일자 2016년03월29일

(71) 출원인
세종대학교산학협력단
서울특별시 광진구 능동로 209 (군자동, 세종대학교)
(72) 발명자
윤은일
서울특별시 성북구 종암로5길 7 (종암동)
양홍모
충청북도 청주시 상당구 수영로 350 (금천동, 장자마을1단지부영아파트) 104동 302동
(뒷면에 계속)
(74) 대리인
정부연

전체 청구항 수 : 총 13 항

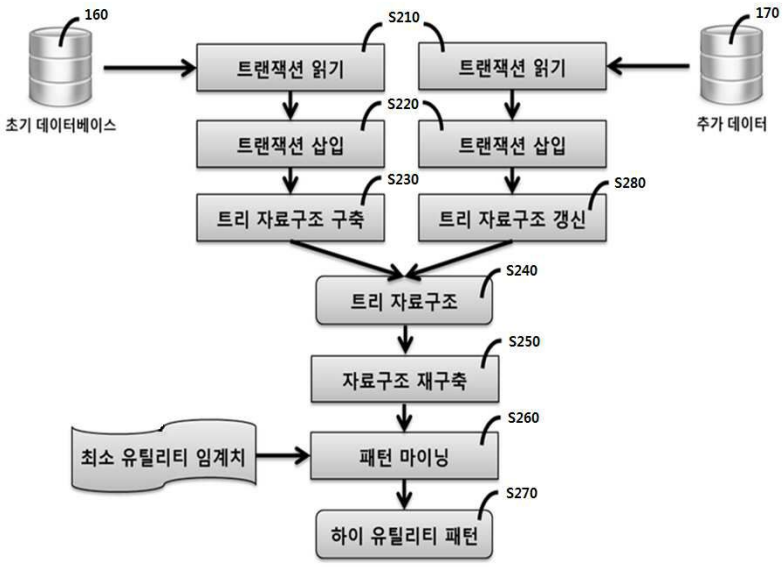
(54) 발명의 명칭 **정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법**

(57) 요 약

정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법은 초기 정적 데이터베이스에 대한 트리 자료구조를 구축하는 단계, 상기 정적 데이터베이스의 트랜잭션들을 빈도수 및 최대 아이템 수량 정보와 함께 상기 트리 자료구조에 저장하고 기 설정된 최적의 정렬순서에 따라 감소된 과추정 유틸리티를 계산하여 자료구조

(뒷면에 계속)

대 표 도 - 도2



를 재 구축하는 단계, 상기 초기 정적 데이터베이스에 동적 데이터베이스가 추가되면 상기 동적 데이터베이스에 대한 트리 자료구조를 갱신하는 단계, 및 사용자에 의한 마이닝 요청이 발생되면 상기 정적 데이터베이스 또는 상기 동적 데이터베이스에서 구축 또는 갱신된 자료구조에서 기 설정된 유틸리티 임계치를 만족하는 모든 하이 유틸리티 패턴들을 마이닝하는 단계를 포함한다. 따라서, 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법은 초기 정적 데이터베이스뿐만 아니라 추후 점진적으로 증가된 동적 데이터베이스까지 고려하여 하이 유틸리티 패턴들을 효율적으로 마이닝 할 수 있다.

(52) CPC특허분류
G06F 17/30345 (2013.01)
G06F 2216/03 (2013.01)

(72) 발명자
이강인
충청북도 청주시 흥덕구 모충로12번길 4 (개신동, 송학아파트)103동 905호
김동규
경기도 김포시 전원로 28, 114동 1304호 (장기동, 전원마을월드1단지아파트)
신동춘
경기도 안산시 상록구 안산천동로1길 9, 19동 301호(월피동, 한양아파트)

이 발명을 지원한 국가연구개발사업
과제고유번호 1345240021
부처명 교육부
연구관리전문기관 한국연구재단
연구사업명 이공학개인기초연구지원
연구과제명 실시간 이레이저블 패턴 마이닝 기술 개발
기 여 율 1/1
주관기관 세종대학교산학협력단
연구기간 2015.11.01 ~ 2016.10.31

명 세 서

청구범위

청구항 1

(a) 초기 정적 데이터베이스에 대한 트리 자료구조를 구축하는 단계;

(b) 상기 정적 데이터베이스의 트랜잭션들을 빈도수 및 최대 아이템 수량 정보와 함께 상기 트리 자료구조에 저장하고 기 설정된 최적의 정렬순서에 따라 감소된 과추정 유틸리티를 계산하여 자료구조를 재 구축하는 단계;

(c) 상기 초기 정적 데이터베이스에 동적 데이터베이스가 추가되면 상기 동적 데이터베이스에 대한 트리 자료구조를 갱신하는 단계; 및

(d) 사용자에 의한 마이닝 요청이 발생되면 상기 정적 데이터베이스 또는 상기 동적 데이터베이스에서 구축 또는 갱신된 자료구조에서 기 설정된 유틸리티 임계치를 만족하는 모든 하이 유틸리티 패턴들을 마이닝하는 단계를 포함하는 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법.

청구항 2

제1항에 있어서, 상기 (a) 단계는

(a1) 상기 초기 정적 데이터베이스로부터 상기 트랜잭션들을 하나씩 읽어오는 단계;

(a2) 해당 트랜잭션 아이템들을 정렬하는 단계; 및

(a3) 상기 정렬된 트랜잭션 아이템들을 트리 자료구조에 삽입하는 단계를 포함하는 것을 특징으로 하는 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법.

청구항 3

제2항에 있어서, 상기 (a) 단계는

(a4) 각 아이템 정보를 헤더 테이블에 트랜잭션 유틸리티를 기준으로 갱신하는 단계;

(a5) 상기 정렬된 트랜잭션 내 마지막 아이템에 대한 노드를 테일 노드 정보 목록(TIList) 내 엔트리와 연결하고 해당 엔트리의 빈도수 및 유틸리티 정보를 각각 1 및 상기 트랜잭션 유틸리티만큼 증가하여 갱신하는 단계;

(a6) 최소 아이템 유틸리티 테이블의 각 삽입 아이템에 대하여 상기 정적 데이터베이스 내 가장 작은 아이템 유틸리티 값 정보를 유지하도록 기 저장된 아이템 유틸리티 값과 현재 삽입되는 아이템 유틸리티 값 중에서 작은 값으로 갱신하는 단계; 및

(a7) 트리 자료구조 생성하는 단계를 더 포함하는 것을 특징으로 하는 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법.

청구항 4

제2항에 있어서, 상기 (a2) 단계는

(a21) 해당 트랜잭션 아이템들을 초기 정렬순서에 따라 정렬하는 단계를 더 포함하는 것을 특징으로 하는 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법.

청구항 5

제1항에 있어서, 상기 (c) 단계는

(c1) 새롭게 추가된 데이터로부터 트랜잭션을 하나씩 읽어오는 단계;

(c2) 현재 정렬순서에 따라 상기 트랜잭션 내 아이템들을 정렬하는 단계; 및

(c3) 상기 정렬된 트랜잭션 아이템들을 트리 자료구조에 삽입하는 단계를 포함하는 것을 특징으로 하는 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법.

청구항 6

제5항에 있어서, 상기 (c) 단계는

(c4) 각 아이템 정보를 헤더 테이블에 트랜잭션 유틸리티를 기준으로 기 구축된 자료구조를 갱신하는 단계;

(c5) 상기 정렬된 트랜잭션 내 마지막 아이템에 대한 노드를 테일 노드 정보 목록(TIList) 내 엔트리와 연결하고 해당 엔트리의 빈도수 및 유틸리티 정보를 각각 1 및 상기 트랜잭션 유틸리티만큼 증가하여 갱신하는 단계;

(c6) 최소 아이템 유틸리티 테이블의 각 삽입 아이템에 대하여 상기 동적 데이터베이스 내 가장 작은 아이템 유틸리티 값 정보를 유지하도록 기 저장된 아이템 유틸리티 값과 현재 삽입되는 아이템 유틸리티 값 중에서 작은 값으로 갱신하는 단계; 및

(c7) 트리 자료구조 생성하는 단계를 더 포함하는 것을 특징으로 하는 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법.

청구항 7

제5항에 있어서, 상기 (c2) 단계는

(c21) 상기 트리 자료구조의 헤더 테이블에 저장된 정보를 이용하여 과추정 유틸리티 내림차순으로 정렬하는 단계를 더 포함하는 것을 특징으로 하는 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법.

청구항 8

제1항에 있어서, 상기 (d) 단계는

(d1) 임시 자료구조(TmpTIList)을 생성하는 단계;

(d2) 테일 노드 정보 목록(TIList)의 각 엔트리를 시작으로 상기 트리 자료구조로부터 경로를 추출하는 단계;

(d3) 기 설정된 정렬순서에 따라 재정렬하고 기 설정된 임계값 이상으로 감소된 과추정 유틸리티를 계산하는 단계; 및

(d4) 재정렬된 경로를 해당 트리 자료구조에 다시 삽입하는 단계를 포함하는 것을 특징으로 하는 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법.

청구항 9

제8항에 있어서, 상기 (d) 단계는

(d5) 해당 경로의 마지막 아이템에 대한 노드 정보를 상기 임시 자료구조에 추가하는 단계;

(d6) 모든 경로에 대한 처리가 완료되면 원본 테일 노드 정보 목록을 제거하는 단계; 및

(d7) 상기 임시 자료구조가 새로운 테일 노드 정보 목록으로 생성되는 단계를 더 포함하는 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법.

청구항 10

정적 데이터베이스 또는 동적 데이터베이스로부터 입력받은 데이터를 스캔하고 상기 정적 데이터베이스 또는 상기 동적 데이터베이스 내의 트랜잭션을 읽어오는 트랜잭션 스캔부;

상기 정적 데이터베이스 또는 상기 동적 데이터베이스의 해당 트랜잭션 아이템들을 정렬하고 이를 트리 자료구조에 삽입하는 트랜잭션 삽입부;

상기 정적 데이터베이스 또는 상기 동적 데이터베이스에 대한 트리 기반의 자료구조를 구축하는 트리 자료구조 모듈;

사용자의 요청에 의해 상기 구축된 자료구조로부터 주어진 기 설정된 최소 유틸리티 임계치를 만족하는 모든 하이 유틸리티 패턴들을 마이닝하는 패턴 마이닝부; 및

상기 패턴 마이닝부에서 마이닝된 하이 유틸리티 패턴들이 기 설정된 기준에 의해 정상적으로 마이닝되었는지 여부를 확인하고, 상기 마이닝된 하이 유틸리티 패턴들을 사용자의 요청에 따라 기 설정된 분류별로 구분하여 디스플레이 기기로 출력하거나 다시 상기 패턴 마이닝부에서 하이 유틸리티 패턴들을 재 마이닝하도록 제어하는 하이 유틸리티 패턴부를 포함하는 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 장치.

청구항 11

제10항에 있어서, 상기 트리 자료구조 모듈은

상기 정적 데이터베이스로부터 입력받은 데이터에 대해 트리 기반의 자료구조를 구축하는 트리 자료구조 구축부;

상기 동적 데이터베이스로부터 입력받은 데이터를 트리 자료구조로 갱신하는 트리 자료구조 갱신부; 및

상기 트리 자료구조 구축부에서 구축된 자료구조를 재 구축하거나 상기 트리 자료구조 갱신부에서 갱신된 자료구조를 재 구축하는 자료구조 재 구축부를 포함하는 것을 특징으로 하는 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 장치.

청구항 12

제11항에 있어서, 상기 트리 자료구조는

각 아이템의 과추정 유틸리티(TWU) 정보를 저장하는 헤더 테이블;

트리;

테일 노드의 빈도수 및 트랜잭션 유틸리티 정보를 저장하는 테일 노드 목록 정보(TIList); 및

상기 트리 자료구조에 삽입되는 해당 트랜잭션 아이템에 대하여 상기 정적 데이터베이스 또는 상기 동적 데이터베이스 내 가장 작은 트랜잭션 아이템 유틸리티 값 정보를 유지하는 최소 아이템 유틸리티 테이블을 포함하는 것을 특징으로 하는 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 장치.

청구항 13

제11항에 있어서, 상기 재 구축부는

상기 정적 데이터베이스의 트랜잭션들을 빈도수 및 최대 아이템 수량 정보와 함께 상기 트리 자료구조에 저장하고 기 설정된 최적의 정렬순서에 따라 감소된 과추정 유틸리티를 계산하여 자료구조를 재 구축하는 것을 특징으로 하는 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 장치.

발명의 설명

기 술 분 야

[0001] 본 발명은 패턴 마이닝 기술에 관한 것으로, 보다 상세하게는, 초기 정적 데이터베이스 및 새로운 데이터가 점진적으로 추가된 동적 데이터베이스로부터 하이 유틸리티 패턴들을 효율적으로 마이닝 할 수 있는 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법에 관한 것이다.

배 경 기 술

[0002] 데이터 마이닝 기술은 질병 관련 중요 패턴 또는 리테일 분석과 같은 실세계 응용에서 데이터베이스에 대한 분석을 통해 중요한 의사 결정을 수행하기 위해 사용된다. 데이터 마이닝은 데이터베이스 내 지식 발견(KDD)의 핵심 작업으로 거대한 데이터베이스에서 중대한 잠재적으로 유용한 정보를 찾아내는 기술이다. 연관 규칙 마이닝, 분류 및 클러스터링 등은 데이터 마이닝을 위한 기술들이며, 그 중에 데이터베이스에서 연관 규칙을 찾아내는 연관 규칙 마이닝은 데이터 마이닝에서 중요한 연구 주제 중에 하나로서 연구되어 왔으며, 많은 알고리즘들이 개발되었다. 연관 규칙 마이닝을 위한 패턴 마이닝 기술은 의료 데이터베이스 내 질병과 관련된 중요 패턴 또는 리테일 상점과 같은 비즈니스에서 의사 결정을 위한 중요 패턴을 발견하기 위해 사용된다. 패턴 마이닝 분야의 가장 기본적인 주제인 빈발 패턴 마이닝은 아이템의 빈도수를 기반으로 하며, 이를 위한 가장 잘 알려진 알고리즘 중에 하나인 선험적(Apriori)는 레벨-와이즈 후보 생성 및 테스트 방법을 사용한다. 이러한 방법은 다중 데이터베이스 스캔 그리고 높은 계산 코스트를 필요로 함으로써 마이닝 성능 저하의 주요 원인이다.

[0003] 패턴 마이닝 분야에서 안티 모노톤 속성(Anti-monotone Property)은 효율적인 마이닝을 위한 근본적인 개념으로서, 마이닝 과정에서 이를 유지하는 것은 마이닝 성능 측면에 있어 중요한 요인들 중에 하나이다. 상기 속성은 최소 임계치가 주어졌을 때, 마이닝 과정에서 발견된 임의의 후보 패턴에 대하여 해당 패턴이 임계치를 만족하지 못한다면 그로부터 생성될 수 있는 모든 상위 패턴들(Super Patterns) 또한 유효하지 않으므로 해당하는 검색 공간을 효과적으로 제거하여 필요 연산을 감소시킬 수 있도록 한다. 하이 유틸리티 패턴 마이닝에서 상기 속성을 만족시키는 것은 쉬운 일이 아니며, 이를 위해 과추정 유틸리티 개념이 사용된다. 이를 통해 마이닝 과정에서 최소 유틸리티 임계치보다 작지 않은 과추정 유틸리티를 가지는 후보 패턴들을 먼저 추출하고, 그 후보들로부터 실제 하이 유틸리티 패턴들을 식별하는 과정을 거친다. 비록 과추정 유틸리티 개념을 적용함으로써 안티 모노톤 속성을 만족시킬 수 있으나, 수많은 후보 패턴들의 생성을 야기하여 패턴 식별 과정에서 많은 실행 시간이 소모된다. 특히, 트리 자료구조 기반의 분할 및 정복이 아닌 레벨 와이즈 후보 생성 및 검증 접근법을 적용하면, 수 많은 데이터베이스 스캔까지 동반하여 마이닝 성능이 더욱 악화되는 문제점이 있다.

[0004] 한국등록특허 제10-1443285호는 유용성 높은 패턴의 마이닝 방법에 관한 것으로, 정적 데이터베이스로부터 아이템 수량 및 중요도를 고려하여 유용성 높은 패턴들, 즉 하이 유틸리티 패턴들을 마이닝 하는데 정적 데이터베이스만을 고려하기 때문에 점진적으로 추가된 데이터를 추가적으로 반영하기 위해서는 갱신된 전체 데이터베이스에 대한 마이닝 과정을 수행함으로 비효율적일 뿐만 아니라 점진적으로 추가된 데이터를 추가적으로 반영하기에 적합하지 않다.

[0005] C.F. Ahmed, S.K. Tanbeer, B.-S. Jeong, and Y.-K. Lee, "Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases", IEEE Transactions on Knowledge and Data Engineering, 2009 는 정적 및 동적 데이터베이스로부터 하이 유틸리티 패턴들을 마이닝 할 수 있지만, 너무 높은 과추정 유틸리티 값을 사용함으로써 수많은 후보 패턴들을 생성하고 이로부터 실제 마이닝 패턴들을 식별하기 위해 많은 실행 시간을 필요로 한다.

[0006] C.-W. Lin, T.-P. Hong, and W.-H. Lu, "An Incremental Mining Algorithm for High Utility Itemsets", Expert Systems with Applications, 2012 는 동적 데이터베이스로부터 하이 유틸리티 패턴들을 마이닝 하는 방법에 관한 기술로서, 트리 자료구조 기반의 분할 및 정복 (Divide and Conquer)이 아닌 레벨 와이즈 후보 생성 및 검증(Level Wise Candidate Generation-and-Test) 접근법을 과추정 유틸리티 모델에 적용함으로써 수 많은 데이터베이스 스캔을 수행할 뿐만 아니라 많은 후보 패턴들을 생성함으로써 느린 실행 속도로 동일한 하이 유틸리티 패턴 집합을 마이닝 하기 때문에 비효율적이다.

선행기술문헌

특허문헌

[0007] (특허문헌 0001) 한국등록특허 제10-1443285호 (2014.09.22 등록)

비특허문헌

[0008] (비특허문헌 0001) C.F. Ahmed, S.K. Tanbeer, B.-S. Jeong, and Y.-K. Lee, "Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases", IEEE Transactions on Knowledge and Data Engineering, 2009

(비특허문헌 0002) C.-W. Lin, T.-P. Hong, and W.-H. Lu, "An Incremental Mining Algorithm for High Utility Itemsets", Expert Systems with Applications, 2012

발명의 내용

해결하려는 과제

[0009] 본 발명의 일 실시예는 초기 정적 데이터베이스뿐만 아니라 추후 점진적으로 증가된 동적 데이터베이스까지 고려하여 하이 유틸리티 패턴들을 효율적으로 마이닝 할 수 있는 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법을 제공하고자 한다.

[0010] 본 발명의 일 실시예는 테일 노드 정보를 이용하여 과추정 유틸리티를 효과적으로 감소시켜 훨씬 더 적은 수의 후보 패턴들을 생성함으로써 하이 유틸리티 패턴 마이닝 성능을 향상시킬 수 있는 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법을 제공하고자 한다.

[0011] 본 발명의 일 실시예는 과추정 유틸리티를 효과적으로 감소시켜 정적 및 동적 데이터베이스로부터 하이 유틸리티 패턴들을 이전의 기법들보다 더욱 효율적으로 마이닝 할 수 있는 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법을 제공하고자 한다.

과제의 해결 수단

[0012] 실시예들 중에서, 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법은, (a) 초기 정적 데이터베이스에 대한 트리 자료구조를 구축하는 단계, (b) 상기 정적 데이터베이스의 트랜잭션들을 빈도수 및 최대 아이템 수량 정보와 함께 상기 트리 자료구조에 저장하고 기 설정된 최적의 정렬순서에 따라 감소된 과추정 유틸리티를 계산하여 자료구조를 재 구축하는 단계, (c) 상기 초기 정적 데이터베이스에 동적 데이터베이스가 추가되면 상기 동적 데이터베이스에 대한 트리 자료구조를 갱신하는 단계, 및 (d) 사용자에 의한 마이닝 요청이 발생되면 상기 정적 데이터베이스 또는 상기 동적 데이터베이스에서 구축 또는 갱신된 자료구조에서 기 설정된 유틸리티 임계치를 만족하는 모든 하이 유틸리티 패턴들을 마이닝하는 단계를 포함한다.

[0013] 일 실시예에서, 상기 (a) 단계는 (a1) 상기 초기 정적 데이터베이스로부터 상기 트랜잭션들을 하나씩 읽어오는 단계, (a2) 해당 트랜잭션 아이템들을 정렬하는 단계, 및 (a3) 상기 정렬된 트랜잭션 아이템들을 트리 자료구조에 삽입하는 단계를 포함할 수 있다.

[0014] 상기 (a) 단계는 (a4) 각 아이템 정보를 헤더 테이블에 트랜잭션 유틸리티를 기준으로 갱신하는 단계, (a5) 상기 정렬된 트랜잭션 내 마지막 아이템에 대한 노드를 테일 노드 정보 목록(TIList) 내 엔트리와 연결하고 해당 엔트리의 빈도수 및 유틸리티 정보를 각각 1 및 상기 트랜잭션 유틸리티만큼 증가하여 갱신하는 단계, (a6) 최소 아이템 유틸리티 테이블의 각 삽입 아이템에 대하여 상기 정적 데이터베이스 내 가장 작은 아이템 유틸리티 값 정보를 유지하도록 기 저장된 아이템 유틸리티 값과 현재 삽입되는 아이템 유틸리티 값 중에서 작은 값으로 갱신하는 단계, 및 (a7) 트리 자료구조 생성하는 단계를 더 포함할 수 있다.

[0015] 상기 (a2) 단계는 (a21) 해당 트랜잭션 아이템들을 초기 정렬순서에 따라 정렬하는 단계를 더 포함할 수 있다.

[0016] 일 실시예에서, 상기 (c) 단계는 (c1) 새롭게 추가된 데이터로부터 트랜잭션을 하나씩 읽어오는 단계, (c2) 현재 정렬순서에 따라 상기 트랜잭션 내 아이템들을 정렬하는 단계, 및 (c3) 상기 정렬된 트랜잭션 아이템들을 트리 자료구조에 삽입하는 단계를 포함할 수 있다.

[0017] 상기 (c) 단계는 (c4) 각 아이템 정보를 헤더 테이블에 트랜잭션 유틸리티를 기준으로 기 구축된 자료구조를 갱

신하는 단계, (c5) 상기 정렬된 트랜잭션 내 마지막 아이템에 대한 노드를 테일 노드 정보 목록(TIList) 내 엔트리와 연결하고 해당 엔트리의 빈도수 및 유틸리티 정보를 각각 1 및 상기 트랜잭션 유틸리티만큼 증가하여 갱신하는 단계, (c6) 최소 아이템 유틸리티 테이블의 각 삽입 아이템에 대하여 상기 동적 데이터베이스 내 가장 작은 아이템 유틸리티 값 정보를 유지하도록 기 저장된 아이템 유틸리티 값과 현재 삽입되는 아이템 유틸리티 값 중에서 작은 값으로 갱신하는 단계, 및 (c7) 트리 자료구조 생성하는 단계를 더 포함할 수 있다.

[0018] 상기 (c2) 단계는 (c21) 상기 트리 자료구조의 헤더 테이블에 저장된 정보를 이용하여 과추정 유틸리티 내림차순으로 정렬하는 단계를 더 포함할 수 있다.

[0019] 일 실시예에서, 상기 (d) 단계는 (d1) 임시 자료구조(TmpTIList)을 생성하는 단계, (d2) 테일 노드 정보 목록(TIList)의 각 엔트리를 시작으로 상기 트리 자료구조로부터 경로를 추출하는 단계, (d3) 기 설정된 정렬순서에 따라 재정렬하고 기 설정된 임계값 이상으로 감소된 과추정 유틸리티를 계산하는 단계, 및 (d4) 재정렬된 경로를 해당 트리 자료구조에 다시 삽입하는 단계를 포함할 수 있다.

[0020] 상기 (d) 단계는 (d5) 해당 경로의 마지막 아이템에 대한 노드 정보를 상기 임시 자료구조에 추가하는 단계, (d6) 모든 경로에 대한 처리가 완료되면 원본 테일 노드 정보 목록을 제거하는 단계, 및 (d7) 상기 임시 자료구조가 새로운 테일 노드 정보 목록으로 생성되는 단계를 더 포함할 수 있다.

[0021] 실시예들 중에서, 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 장치는 정적 데이터베이스 또는 동적 데이터베이스로부터 입력받은 데이터를 스캔하고 상기 정적 데이터베이스 또는 상기 동적 데이터베이스 내의 트랜잭션을 읽어오는 트랜잭션 스캔부, 상기 정적 데이터베이스 또는 상기 동적 데이터베이스의 해당 트랜잭션 아이템들을 정렬하고 이를 트리 자료구조에 삽입하는 트랜잭션 삽입부, 상기 정적 데이터베이스 또는 상기 동적 데이터베이스에 대한 트리 기반의 자료구조를 구축하는 트리 자료구조 모듈, 사용자의 요청에 의해 상기 구축된 자료구조로부터 주어진 기 설정된 최소 유틸리티 임계치를 만족하는 모든 하이 유틸리티 패턴들을 마이닝하는 패턴 마이닝부, 및 상기 패턴 마이닝부에서 마이닝된 하이 유틸리티 패턴들이 기 설정된 기준에 의해 정상적으로 마이닝되었는지 여부를 확인하고, 상기 마이닝된 하이 유틸리티 패턴들을 사용자의 요청에 따라 기 설정된 분류별로 구분하여 디스플레이 기기로 출력하거나 다시 상기 패턴 마이닝부에서 하이 유틸리티 패턴들을 재 마이닝하도록 제어하는 하이 유틸리티 패턴부를 포함한다.

[0022] 일 실시예에서, 상기 트리 자료구조 모듈은 상기 정적 데이터베이스로부터 입력받은 데이터에 대해 트리 기반의 자료구조를 구축하는 트리 자료구조 구축부, 상기 동적 데이터베이스로부터 입력받은 데이터를 트리 자료구조로 갱신하는 트리 자료구조 갱신부, 및 상기 트리 자료구조 구축부에서 구축된 자료구조를 재 구축하거나 상기 트리 자료구조 갱신부에서 갱신된 자료구조를 재 구축하는 자료구조 재 구축부를 포함할 수 있다.

[0023] 일 실시예에서, 상기 트리 자료구조는 각 아이템의 과추정 유틸리티(TWU) 정보를 저장하는 헤더 테이블, 트리, 테일 노드의 빈도수 및 트랜잭션 유틸리티 정보를 저장하는 테일 노드 목록 정보(TIList), 및 상기 트리 자료구조에 삽입되는 해당 트랜잭션 아이템에 대하여 상기 정적 데이터베이스 또는 상기 동적 데이터베이스 내 가장 작은 트랜잭션 아이템 유틸리티 값 정보를 유지하는 최소 아이템 유틸리티 테이블을 포함할 수 있다.

[0024] 상기 재 구축부는 상기 정적 데이터베이스의 트랜잭션들을 빈도수 및 최대 아이템 수량 정보와 함께 상기 트리 자료구조에 저장하고 기 설정된 최적의 정렬순서에 따라 감소된 과추정 유틸리티를 계산하여 자료구조를 재 구축할 수 있다.

발명의 효과

[0025] 개시된 기술은 다음의 효과를 가질 수 있다. 다만, 특정 실시예가 다음의 효과를 전부 포함하여야 한다거나 다음의 효과만을 포함하여야 한다는 의미는 아니므로, 개시된 기술의 권리범위는 이에 의하여 제한되는 것으로 이해되어서는 아니 될 것이다.

[0026] 본 발명의 일 실시예에 따른 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법은 초기 정적 데이터베이스뿐만 아니라 추후 점진적으로 증가된 동적 데이터베이스까지 고려하여 하이 유틸리티 패턴들을 효율적으로 마이닝 할 수 있다.

[0027] 본 발명의 일 실시예에 따른 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법은 테일 노드 정보를 이용하여 과추정 유틸리티를 효과적으로 감소시켜 훨씬 더 적은 수의 후보 패턴들을 생성함으로써 하이 유틸리티 패턴 마이닝 성능을 향상시킬 수 있다.

[0028] 본 발명의 일 실시예에 따른 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법은 과추정 유틸리티를 효과적으로 감소시켜 정적 및 동적 데이터베이스로부터 하이 유틸리티 패턴들을 이전의 기법들보다 더욱 효율적으로 마이닝 할 수 있다.

도면의 간단한 설명

[0029] 도 1은 본 발명의 일 실시예에 따른 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 장치를 나타낸 도면이다.

도 2는 도 1에 있는 점진적 하이 유틸리티 패턴 마이닝 장치에서 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법을 설명하는 순서도이다.

도 3은 도 2에 있는 트리 자료구조를 설명하는 도면이다.

도 4는 도 2에 있는 정적 데이터베이스에 대한 트리 자료구조 구축 과정을 설명하는 순서도이다.

도 5는 도 2에 있는 동적 데이터베이스에 대한 트리 자료구조 갱신 과정을 설명하는 순서도이다.

도 6은 도 2에 있는 트리 자료구조 재 구축 과정을 설명하는 순서도이다.

도 7 내지 도 17은 본 발명의 과추정 유틸리티를 감소시킨 점진적 하이 유틸리티 패턴 마이닝 과정을 나타낸 도면이다.

발명을 실시하기 위한 구체적인 내용

[0030] 본 발명에 관한 설명은 구조적 내지 기능적 설명을 위한 실시예에 불과하므로, 본 발명의 권리범위는 본문에 설명된 실시예에 의하여 제한되는 것으로 해석되어서는 아니 된다. 즉, 실시예는 다양한 변경이 가능하고 여러 가지 형태를 가질 수 있으므로 본 발명의 권리범위는 기술적 사상을 실현할 수 있는 균등물들을 포함하는 것으로 이해되어야 한다. 또한, 본 발명에서 제시된 목적 또는 효과는 특정 실시예가 이를 전부 포함하여야 한다거나 그러한 효과만을 포함하여야 한다는 의미는 아니므로, 본 발명의 권리범위는 이에 의하여 제한되는 것으로 이해되어서는 아니 될 것이다.

[0031] 한편, 본 출원에서 서술되는 용어의 의미는 다음과 같이 이해되어야 할 것이다.

[0032] "제1", "제2" 등의 용어는 하나의 구성요소를 다른 구성요소로부터 구별하기 위한 것으로, 이들 용어들에 의해 권리범위가 한정되어서는 아니 된다. 예를 들어, 제1 구성요소는 제2 구성요소로 명명될 수 있고, 유사하게 제2 구성요소도 제1 구성요소로 명명될 수 있다.

[0033] 어떤 구성요소가 다른 구성요소에 "연결되어"있다고 언급된 때에는, 그 다른 구성요소에 직접적으로 연결될 수도 있지만, 중간에 다른 구성요소가 존재할 수도 있다고 이해되어야 할 것이다. 반면에, 어떤 구성요소가 다른 구성요소에 "직접 연결되어"있다고 언급된 때에는 중간에 다른 구성요소가 존재하지 않는 것으로 이해되어야 할 것이다. 한편, 구성요소들 간의 관계를 설명하는 다른 표현들, 즉 "~사이에"와 "바로 ~사이에" 또는 "~에 이웃하는"과 "~에 직접 이웃하는" 등도 마찬가지로 해석되어야 한다.

[0034] 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한 복수의 표현을 포함하는 것으로 이해되어야 하고, "포함하다"또는 "가지다" 등의 용어는 실시된 특징, 숫자, 단계, 동작, 구성요소, 부분품 또는 이들을 조합한 것이 존재함을 지정하려는 것이며, 하나 또는 그 이상의 다른 특징이나 숫자, 단계, 동작, 구성요소, 부분품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다.

[0035] 각 단계들에 있어 식별부호(예를 들어, a, b, c 등)는 설명의 편의를 위하여 사용되는 것으로 식별부호는 각 단계들의 순서를 설명하는 것이 아니며, 각 단계들은 문맥상 명백하게 특정 순서를 기재하지 않는 이상 명기된 순서와 다르게 일어날 수 있다. 즉, 각 단계들은 명기된 순서와 동일하게 일어날 수도 있고 실질적으로 동시에 수행될 수도 있으며 반대의 순서대로 수행될 수도 있다.

[0036] 여기서 사용되는 모든 용어들은 다르게 정의되지 않는 한, 본 발명이 속하는 분야에서 통상의 지식을 가진 자에 의해 일반적으로 이해되는 것과 동일한 의미를 가진다. 일반적으로 사용되는 사전에 정의되어 있는 용어들은 관련 기술의 문맥상 가지는 의미와 일치하는 것으로 해석되어야 하며, 본 출원에서 명백하게 정의하지 않는 한 이상적이거나 과도하게 형식적인 의미를 지니는 것으로 해석될 수 없다.

[0038] 도 1은 본 발명의 일 실시예에 따른 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 장치를 나타낸 도면이다.

[0039] 도 1을 참조하면, 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 장치(100)는 트랜잭션 스캔부(110), 트랜잭션 삽입부(120), 트리 자료구조 모듈(130), 패턴 마이닝부(140) 및 하이 유틸리티 패턴부(150)를 포함한다.

[0040] 먼저, 트랜잭션 스캔부(110)는 정적 데이터베이스(160) 또는 동적 데이터베이스(170)로부터 입력받은 데이터를 스캔하고 정적 데이터베이스(160) 또는 동적 데이터베이스(170) 내의 트랜잭션을 읽어(Read)온다.

[0041] 트랜잭션 스캔부(110)는 정적 데이터베이스(160)로부터 입력받은 데이터를 스캔하고 이에 해당하는 트랜잭션을 하나씩 읽어오며, 동적 데이터 베이스(170) 즉, 새로운 데이터가 점진적으로 정적 데이터베이스(160)에 동적으로 추가되면 추가된 해당 데이터만을 스캔하고 이에 해당하는 트랜잭션을 하나씩 읽어올 수 있다. 여기에서, 정적 데이터베이스(160)는 초기 데이터베이스로서 기존에 저장된 데이터를 의미하며, 동적 데이터베이스(170)는 초기 데이터베이스가 아닌 새롭게 추가된 새로운 데이터를 의미한다.

[0042] 트랜잭션 삽입부(120)는 정적 데이터베이스(160) 또는 동적 데이터베이스(170)의 해당 트랜잭션 아이템들을 정렬하고 이를 트리 자료구조에 삽입한다. 트랜잭션 삽입부(120)는 정적 데이터베이스(160)의 해당 트랜잭션 아이템들을 초기 정렬순서 즉, 초기 임의로 정해진 순서에 따라 정렬하고, 동적 데이터베이스(170)의 해당 트랜잭션 아이템들을 현재 정렬순서에 따라 트랜잭션 내 아이템들을 정렬할 수 있다. 여기에서, 초기 정렬순서는 사용자에 의해 트랜잭션 삽입부(120)에 기 설정된 정렬순서로서, 내림차순, 오름차순 또는 특정 기준 차순을 포함한 다양한 정렬방법이 이에 해당될 수 있다. 트랜잭션 삽입부(120)는 동적 데이터베이스(170)의 트리 자료구조를 갱신하기 위해 트리 자료구조의 헤더 테이블에 저장된 정보를 이용하여 과추정 유틸리티를 내림차순으로 정렬할 수 있다.

[0043] 도 1에서, 트리 자료구조 모듈(130)는 트리 자료구조 구축부(131), 트리 자료구조 갱신부(132) 및 자료구조 재구축부(133)를 포함하고, 이들을 통해 트리 기반의 자료구조를 구축한다.

[0044] 트리 자료구조 구축부(131)는 정적 데이터베이스(160)로부터 입력받은 데이터에 대해 트리 기반의 자료구조를 구축한다. 이하, 트리 기반의 자료구조에 대한 상세한 설명은 도 3에서 설명하고, 트리 기반의 자료구조 구축과정에 대한 상세한 설명은 도 4에서 설명한다.

[0045] 트리 자료구조 갱신부(132)는 동적 데이터베이스(170)로부터 입력받은 데이터를 정적 데이터베이스(160)를 통해 기 구축된 자료구조에 반영하여 보다 효율적으로 트리 자료구조를 갱신한다. 이하, 트리 자료구조 갱신과정에 대한 상세한 설명은 도 5에서 설명한다.

[0046] 자료구조 재 구축부(133)는 트리 자료구조 구축부(131)에서 구축된 정적 데이터베이스(160)에 대한 자료구조를 기 설정된 최적의 정렬순서에 따라 재 구축한다. 자료구조 재 구축부(133)는 자료구조 재 구축 과정 즉, 정적 데이터베이스의 트랜잭션들을 빈도수 및 최대 아이템 수량 정보와 함께 트리 자료구조에 저장하고 기 설정된 최적의 정렬순서에 따라 감소된 과추정 유틸리티를 계산하여 자료구조를 재 구축할 수 있다. 이때, 자료구조 재구축부(133)는 기 설정된 최적의 정렬순서에 따라 감소된 과추정 유틸리티를 계산하여 트리 자료구조에 저장함으로써 추후 마이닝 과정에서 이전 마이닝 기법을 사용한 것보다 감소된 수의 후보 패턴들 즉, 보다 신뢰성 있는 후보 패턴들을 추출할 수 있다. 감소된 과추정 유틸리티의 계산은 기 설정된 임계치 이상, 기 설정된 임계치 이하 또는 사용자에 의해 기 설정된 특정 임계치로 감소된 과추정 유틸리티를 계산할 수 있다. 여기에서, 기 설정된 임계치는 기 설정된 한계 값을 의미한다.

[0047] 자료구조 재 구축부(133)는 트리 자료구조 갱신부(132)에서 갱신된 동적 데이터베이스(170)에 대한 자료구조를 재 구축할 수 있다. 이하, 트리 자료구조 재구축 과정에 대한 상세한 설명은 도 6에서 설명한다.

[0048] 패턴 마이닝부(140)는 사용자의 요청에 의해 트리 자료구조 구축부(131)에서 구축된 자료구조로부터 주어진 기 설정된 최소 유틸리티 임계치를 만족하는 모든 하이 유틸리티 패턴들을 마이닝(mining)한다. 여기에서, 마이닝은 데이터 마이닝(data mining)이며 데이터베이스 내 지식 발견(KDD, Knowledge Discovery in Databases)의 핵심 작업으로, 거대한 데이터베이스에서 중대하거나 잠재적으로 유용한 정보를 찾아내는 것을 말한다. 하이 유틸리티 패턴은 실세계 데이터베이스 내의 각 아이템에서 서로 다른 중요도를 가지는 외부 유틸리티와 하나의 트랜잭션 내의 각 아이템과 연관된 비-바이너리 값을 가지는 내부 유틸리티의 곱으로 패턴의 유틸리티를 측정하고,

만약 패턴의 유틸리티가 사용자 정의 최소 유틸리티 값보다 작지 않은 패턴을 의미한다. 여기에서, 최소 유틸리티 임계치는 기 설정된 유틸리티 한계 값을 의미한다.

[0049] 패턴 마이닝부(140)는 테일 노드 정보를 이용하여 과추정 유틸리티를 효과적으로 감소시켜 기존의 마이닝 방법들 보다 훨씬 더 적은 수의 후보 패턴들을 추출할 수 있다. 패턴 마이닝부(140)는 트리 자료구조 구축부(131)에서 자료구조를 구축한 후뿐만 아니라 트리 자료구조 갱신부(132)에서 자료구조를 갱신한 후에도 사용자의 요청에 따라 마이닝 과정이 수행되어 하이 유틸리티 패턴들을 마이닝 즉, 추출 또는 검색할 수 있다.

[0050] 하이 유틸리티 패턴부(150)는 패턴 마이닝부(140)에서 마이닝된 하이 유틸리티 패턴들이 기 설정된 기준에 의해 정상적으로 마이닝 되었는지 여부를 확인할 수 있다. 이때, 하이 유틸리티 패턴부(150)가 마이닝된 하이 유틸리티 패턴들이 정상적으로 마이닝 되었다고 판단되면 이를 디스플레이 기기로 출력할 수 있으나 반대로, 마이닝된 하이 유틸리티 패턴들이 비정상적으로 마이닝 되었다고 판단되면 다시 패턴 마이닝부(140)에서 하이 유틸리티 패턴들을 재 마이닝 하도록 제어할 수 있다. 또한, 하이 유틸리티 패턴부(150)는 패턴 마이닝부(140)에서 마이닝된 하이 유틸리티 패턴들을 사용자의 요청에 따라 기 설정된 분류별로 구분하여 디스플레이 기기로 출력할 수 있다. 여기에서, 디스플레이 기기는 PC, 스마트폰 및 태블릿 PC 를 포함한 다양한 디스플레이 기기에 해당할 수 있다.

[0051] 하이 유틸리티 패턴부(150)는 패턴 마이닝부(140)에서 마이닝된 하이 유틸리티 패턴들을 사용자의 요청에 의해 다시 패턴 마이닝부(140)에서 재 마이닝 하도록 제어할 수 있다

[0052] 본 발명의 하이 유틸리티 패턴 마이닝 장치는 의료 또는 마켓 데이터베이스들과 같은 실세계 데이터의 특성을 반영하고 있으므로 중요한 역할을 수행할 수 있다. 더욱이, 웹 클릭 스트림 분석, 모바일 커머스 환경 계획 및 리테일 스토어들 등에서 크로스 마케팅 및 생체 유전자 데이터베이스 분석 등과 같은 다른 분야에서 사용될 수 있다.

[0053] 비록 하이 유틸리티 패턴 마이닝이 내부 그리고 외부 유틸리티라는 실제 데이터베이스의 특성을 반영하고 있지만 기존의 알고리즘들은 대부분 고정된 트랜잭션 데이터베이스를 대상으로 하였다. 그러나, 실세계 응용에서는 새로 생성된 데이터 또는 다른 데이터베이스 내 데이터의 통합 등에 의해 점진적으로 증가할 수 있으며 따라서 기존의 고정된 트랜잭션 데이터베이스를 위한 알고리즘들은 이를 반영하지 못한다. 이러한 문제를 다루기 위해, 점진적 데이터베이스에서 하이 유틸리티 패턴을 마이닝 하는 알고리즘들이 제안되었다. 한편, 데이터베이스에서 마이닝 하는 것은 안티모노톤의 속성(downward closure property)을 만족하지 않아 쉽지 않은 작업이다. 즉, 낮은 유틸리티를 가지는 패턴의 슈퍼 패턴이 하이 유틸리티 패턴이 될 수 있어 미리 패턴을 제거할 수 없다. 이 문제를 해결하기 위해, 각 후보 패턴을 포함하는 모든 트랜잭션들의 유틸리티 합을 이용한 유틸리티 과추정 기법이 제안되었으며, 그 속성을 유지하기 위해 기존의 연구들은 이 기법을 적용하였다. 그 결과, 종종 많은 수의 후보 패턴 생성을 야기하였고 이로 인해 마이닝 성능이 저하되었다. 그 이유는 그 후보 패턴의 슈퍼 패턴에 포함되지 않는 아이템들의 유틸리티들까지 그 합에 더했기 때문이다. 즉, 필요 이상으로 과추정 유틸리티를 계산하여 더 많은 수의 후보 패턴을 생성했기 때문이다. 과추정 기법 기반의 프레임 워크에서, 더 많은 수의 후보 패턴은 더 많은 실행 시간을 소비한다. 특히, 점진적 데이터베이스에서 하이 유틸리티 패턴을 마이닝 하기 위한 최신 알고리즘들 또한 과추정 유틸리티 기법을 적용함으로써 많은 시간을 필요로 한다. 이러한 이유로 우리는 과추정 유틸리티를 감소시킴으로써 생성되는 후보 패턴의 수를 줄여야 한다. 즉, 본 발명은 점진적 데이터베이스에서 하이 유틸리티 패턴을 마이닝 하기 위한 프레임 워크에서 마이닝 성능을 향상시키는 것을 목표로 한다. 한편, 데이터베이스를 스캔 하는 작업은 특히 데이터베이스의 크기가 클 때 많은 시간을 필요로 하는 작업이 될 수 있다. 따라서, 점진적으로 데이터가 증가할 때마다 전체 데이터베이스를 스캔 하는 작업은 효율적이지 못하다. 즉, 실세계 응용에서는 효율적인 패턴 마이닝을 수행하기 위해 전체 데이터베이스에 대한 추가적인 스캔 없이 새로운 정보를 데이터 구조에 반영할 필요가 있다. 이는 점진적 마이닝을 위해서는 한 번의 데이터베이스 스캔으로 구축되는 자료 구조가 필요하다. 또한, 패턴 마이닝에서 데이터 구조 내 아이템 정렬 순서는 효율적인 마이닝에 중요한 기준이 된다. 따라서, 본 발명의 또 다른 목표는 한 번의 데이터베이스 스캔으로 구축되는 데이터 구조, 추가적인 데이터베이스 스캔 없이 효율적인 마이닝을 위한 정렬 순서에 따라 그 구조를 재구축 하는 방법 및 점진적으로 추가된 정보를 효과적으로 반영하기 위한 방법에 대한 기술이다. 본 발명에서는 점진적 데이터베이스를 다루기 위한 새로운 효율적인 알고리즘을 제안한다. 또한, 제안된 알고리즘은 트리 구축 및 마이닝 과정에서 감소된 과추정 유틸리티의 사용을 통해 효과적으로 후보 패턴을 감소시키며, 이를 위한 새로운 자료 구조를 사용한다. 본 발명의 주요 기여는 다음과 같이 요약된다.

[0054] 먼저, 본 발명의 일 실시예는 점진적 데이터베이스에서 하이 유틸리티 패턴을 마이닝 하기 위한 한 번 스캔으로

구축되는 트리 구조와 HUPID-Tree(High Utility Patterns in Incremental Databases Tree)라 불리는 다른 트리 구조를 제안한다. 또한, 트리의 과추정 유틸리티를 감소시키기 위한 재 구축 방법을 새로운 자료 구조, 테일 노드 정보 목록(TIList, Tail-node Information List)(이하, TIList라 함)라 하는 대조 트리와 함께 제안하며 제안된 재 구축 방법의 적용을 통해 HUPID-Tree는 한 번의 데이터베이스 스캔으로 구축된다. 더욱이, 전체 데이터베이스에 대한 추가 스캔 없이 점진적으로 추가되는 데이터를 데이터 구조에 반영할 수 있다.

[0055] 또한, 본 발명의 일 실시예는 점진적으로 추가되는 데이터베이스 내 트랜잭션들의 정보가 반영된 트리로부터 하이 유틸리티 패턴들을 마이닝 하기 위한 새로운 알고리즘, HUPID-Growth(High Utility Patterns in Incremental Databases Growth)라 하는 알고리즘을 제안한다. 제안된 알고리즘을 TIList와 함께 적용함으로써 과추정 유틸리티 추정치를 감소시키며, 이를 통해 후보 개수와 실행 시간을 감소시킬 수 있다.

[0056] 또한, 본 발명의 일 실시예는 성능 비교를 위해 의료 및 실세계 데이터 셋들을 사용한 최신 점진적 하이 유틸리티 패턴 마이닝 알고리즘과의 성능 평가를 수행한다. 실험 결과는 제안한 알고리즘이 다른 알고리즘보다, 특히 데이터베이스가 많은 수의 긴 트랜잭션을 포함하거나 낮은 임계치가 설정됐을 때, 성능이 더 나은 것을 보여줄 수 있다.

[0058] 도 2는 도 1에 있는 점진적 하이 유틸리티 패턴 마이닝 장치에서 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법을 설명하는 순서도이다.

[0059] 도 2를 참조하면, 하이 유틸리티 패턴 마이닝 장치(100)는 트랜잭션 스캔부(110), 트랜잭션 삽입부(120) 및 트리 자료구조 구축부(131)를 통해 정적 데이터베이스(초기 데이터베이스)(160)에 대한 트리 자료구조를 구축한다. 트랜잭션 스캔부(110)는 정적 데이터베이스(160)로부터 입력받은 데이터를 스캔하고 이에 해당하는 트랜잭션을 하나씩 읽어온다(S210). 트랜잭션 삽입부(120)는 정적 데이터베이스(160)의 해당 트랜잭션 아이템들을 초기 임의로 정해진 순서에 따라 정렬하고 이를 트리 자료구조에 삽입한다(S220). 트리 자료구조 구축부(131)는 트랜잭션 삽입부(120)에서 정적 데이터베이스(160)로부터 입력받은 데이터에 대해 트리 기반의 자료구조를 구축(S230)하여 트리 자료구조를 생성한다(S240).

[0060] 자료구조 재 구축부(133)는 정적 데이터베이스(160)의 트랜잭션들을 빈도수 및 최대 아이템 수량 정보와 함께 트리 자료구조에 저장하고 기 설정된 최적의 정렬순서에 따라 감소된 과추정 유틸리티를 계산하여 자료구조를 재 구축한다본 발명에 관한 설명은 구조적 내지 기능적 설명을 위한 실시예에 불과하므로, 본 발명의 권리범위는 본문에 설명된 실시예에 의하여 제한되는 것으로 해석되어서는 아니 된다. 즉, 실시예는 다양한 변경이 가능하고 여러 가지 형태를 가질 수 있으므로 본 발명의 권리범위는 기술적 사상을 실현할 수 있는 균등물들을 포함하는 것으로 이해되어야 한다. 또한, 본 발명에서 제시된 목적 또는 효과는 특정 실시예가 이를 전부 포함하여야 한다거나 그러한 효과만을 포함하여야 한다는 의미는 아니므로, 본 발명의 권리범위는 이에 의하여 제한되는 것으로 이해되어서는 아니 될 것이다.

[0061] 한편, 본 출원에서 서술되는 용어의 의미는 다음과 같이 이해되어야 할 것이다.

[0062] "제1", "제2" 등의 용어는 하나의 구성요소를 다른 구성요소로부터 구별하기 위한 것으로, 이들 용어들에 의해 권리범위가 한정되어서는 아니 된다. 예를 들어, 제1 구성요소는 제2 구성요소로 명명될 수 있고, 유사하게 제2 구성요소도 제1 구성요소로 명명될 수 있다.

[0063] 어떤 구성요소가 다른 구성요소에 "연결되어"있다고 언급된 때에는, 그 다른 구성요소에 직접적으로 연결될 수도 있지만, 중간에 다른 구성요소가 존재할 수도 있다고 이해되어야 할 것이다. 반면에, 어떤 구성요소가 다른 구성요소에 "직접 연결되어"있다고 언급된 때에는 중간에 다른 구성요소가 존재하지 않는 것으로 이해되어야 할 것이다. 한편, 구성요소들 간의 관계를 설명하는 다른 표현들, 즉 "~사이에"와 "바로 ~사이에" 또는 "~에 이웃하는"과 "~에 직접 이웃하는" 등도 마찬가지로 해석되어야 한다.

[0064] 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한 복수의 표현을 포함하는 것으로 이해되어야 하고, "포함하다"또는 "가지다" 등의 용어는 실시된 특징, 숫자, 단계, 동작, 구성요소, 부분품 또는 이들을 조합한 것이 존재함을 지정하려는 것이며, 하나 또는 그 이상의 다른 특징이나 숫자, 단계, 동작, 구성요소, 부분품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다.

[0065] 각 단계들에 있어 식별부호(예를 들어, a, b, c 등)는 설명의 편의를 위하여 사용되는 것으로 식별부호는 각 단계들의 순서를 설명하는 것이 아니며, 각 단계들은 문맥상 명백하게 특정 순서를 기재하지 않는 이상 명기된 순

서와 다르게 일어날 수 있다. 즉, 각 단계들은 명기된 순서와 동일하게 일어날 수도 있고 실질적으로 동시에 수행될 수도 있으며 반대의 순서대로 수행될 수도 있다.

[0066] 여기서 사용되는 모든 용어들은 다르게 정의되지 않는 한, 본 발명이 속하는 분야에서 통상의 지식을 가진 자에 의해 일반적으로 이해되는 것과 동일한 의미를 가진다. 일반적으로 사용되는 사전에 정의되어 있는 용어들은 관련 기술의 문맥상 가지는 의미와 일치하는 것으로 해석되어야 하며, 본 출원에서 명백하게 정의하지 않는 한 이상적이거나 과도하게 형식적인 의미를 지니는 것으로 해석될 수 없다.

[0068] 도 1은 본 발명의 일 실시예에 따른 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 장치를 나타낸 도면이다.

[0069] 도 1을 참조하면, 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 장치(100)는 트랜잭션 스캔부(110), 트랜잭션 삽입부(120), 트리 자료구조 모듈(130), 패턴 마이닝부(140) 및 하이 유틸리티 패턴부(150)를 포함한다.

[0070] 먼저, 트랜잭션 스캔부(110)는 정적 데이터베이스(160) 또는 동적 데이터베이스(170)로부터 입력받은 데이터를 스캔하고 정적 데이터베이스(160) 또는 동적 데이터베이스(170) 내의 트랜잭션을 읽어(Read)온다.

[0071] 트랜잭션 스캔부(110)는 정적 데이터베이스(160)로부터 입력받은 데이터를 스캔하고 이에 해당하는 트랜잭션을 하나씩 읽어오며, 동적 데이터 베이스(170) 즉, 새로운 데이터가 점진적으로 정적 데이터베이스(160)에 동적으로 추가되면 추가된 해당 데이터만을 스캔하고 이에 해당하는 트랜잭션을 하나씩 읽어올 수 있다. 여기에서, 정적 데이터베이스(160)는 초기 데이터베이스로서 기존에 저장된 데이터를 의미하며, 동적 데이터베이스(170)는 초기 데이터베이스가 아닌 새롭게 추가된 새로운 데이터를 의미한다.

[0072] 트랜잭션 삽입부(120)는 정적 데이터베이스(160) 또는 동적 데이터베이스(170)의 해당 트랜잭션 아이템들을 정렬하고 이를 트리 자료구조에 삽입한다. 트랜잭션 삽입부(120)는 정적 데이터베이스(160)의 해당 트랜잭션 아이템들을 초기 정렬순서 즉, 초기 임의로 정해진 순서에 따라 정렬하고, 동적 데이터베이스(170)의 해당 트랜잭션 아이템들을 현재 정렬순서에 따라 트랜잭션 내 아이템들을 정렬할 수 있다. 여기에서, 초기 정렬순서는 사용자에 의해 트랜잭션 삽입부(120)에 기 설정된 정렬순서로서, 내림차순, 오름차순 또는 특정 기준 차순을 포함한 다양한 정렬방법이 이에 해당될 수 있다. 트랜잭션 삽입부(120)는 동적 데이터베이스(170)의 트리 자료구조를 갱신하기 위해 트리 자료구조의 헤더 테이블에 저장된 정보를 이용하여 과추정 유틸리티를 내림차순으로 정렬할 수 있다.

[0073] 도 1에서, 트리 자료구조 모듈(130)는 트리 자료구조 구축부(131), 트리 자료구조 갱신부(132) 및 자료구조 재구축부(133)를 포함하고, 이들을 통해 트리 기반의 자료구조를 구축한다.

[0074] 트리 자료구조 구축부(131)는 정적 데이터베이스(160)로부터 입력받은 데이터에 대해 트리 기반의 자료구조를 구축한다. 이하, 트리 기반의 자료구조에 대한 상세한 설명은 도 3에서 설명하고, 트리 기반의 자료구조 구축과정에 대한 상세한 설명은 도 4에서 설명한다.

[0075] 트리 자료구조 갱신부(132)는 동적 데이터베이스(170)로부터 입력받은 데이터를 정적 데이터베이스(160)를 통해 기 구축된 자료구조에 반영하여 보다 효율적으로 트리 자료구조를 갱신한다. 이하, 트리 자료구조 갱신과정에 대한 상세한 설명은 도 5에서 설명한다.

[0076] 자료구조 재 구축부(133)는 트리 자료구조 구축부(131)에서 구축된 정적 데이터베이스(160)에 대한 자료구조를 기 설정된 최적의 정렬순서에 따라 재 구축한다. 자료구조 재 구축부(133)는 자료구조 재 구축 과정 즉, 정적 데이터베이스의 트랜잭션들을 빈도수 및 최대 아이템 수량 정보와 함께 트리 자료구조에 저장하고 기 설정된 최적의 정렬순서에 따라 감소된 과추정 유틸리티를 계산하여 자료구조를 재 구축할 수 있다. 이때, 자료구조 재구축부(133)는 기 설정된 최적의 정렬순서에 따라 감소된 과추정 유틸리티를 계산하여 트리 자료구조에 저장함으로써 추후 마이닝 과정에서 이전 마이닝 기법을 사용한 것보다 감소된 수의 후보 패턴들 즉, 보다 신뢰성 있는 후보 패턴들을 추출할 수 있다. 감소된 과추정 유틸리티의 계산은 기 설정된 임계치 이상, 기 설정된 임계치 이하 또는 사용자에 의해 기 설정된 특정 임계치로 감소된 과추정 유틸리티를 계산할 수 있다. 여기에서, 기 설정된 임계치는 기 설정된 한계 값을 의미한다.

[0077] 자료구조 재 구축부(133)는 트리 자료구조 갱신부(132)에서 갱신된 동적 데이터베이스(170)에 대한 자료구조를

재 구축할 수 있다. 이하, 트리 자료구조 재구축 과정에 대한 상세한 설명은 도 6에서 설명한다.

[0078] 패턴 마이닝부(140)는 사용자의 요청에 의해 트리 자료구조 구축부(131)에서 구축된 자료구조로부터 주어진 기 설정된 최소 유틸리티 임계치를 만족하는 모든 하이 유틸리티 패턴들을 마이닝(mining)한다. 여기에서, 마이닝은 데이터 마이닝(data mining)이며 데이터베이스 내 지식 발견(KDD, Knowledge Discovery in Databases)의 핵심 작업으로, 거대한 데이터베이스에서 중대하거나 잠재적으로 유용한 정보를 찾아내는 것을 말한다. 하이 유틸리티 패턴은 실세계 데이터베이스 내의 각 아이템에서 서로 다른 중요도를 가지는 외부 유틸리티와 하나의 트랜잭션 내의 각 아이템과 연관된 비-바이너리 값을 가지는 내부 유틸리티의 곱으로 패턴의 유틸리티를 측정하고, 만약 패턴의 유틸리티가 사용자 정의 최소 유틸리티 값보다 작지 않은 패턴을 의미한다. 여기에서, 최소 유틸리티 임계치는 기 설정된 유틸리티 한계 값을 의미한다.

[0079] 패턴 마이닝부(140)는 테일 노드 정보를 이용하여 과추정 유틸리티를 효과적으로 감소시켜 기존의 마이닝 방법들 보다 훨씬 더 적은 수의 후보 패턴들을 추출할 수 있다. 패턴 마이닝부(140)는 트리 자료구조 구축부(131)에서 자료구조를 구축한 후뿐만 아니라 트리 자료구조 갱신부(132)에서 자료구조를 갱신한 후에도 사용자의 요청에 따라 마이닝 과정이 수행되어 하이 유틸리티 패턴들을 마이닝 즉, 추출 또는 검색할 수 있다.

[0080] 하이 유틸리티 패턴부(150)는 패턴 마이닝부(140)에서 마이닝된 하이 유틸리티 패턴들이 기 설정된 기준에 의해 정상적으로 마이닝 되었는지 여부를 확인할 수 있다. 이때, 하이 유틸리티 패턴부(150)가 마이닝된 하이 유틸리티 패턴들이 정상적으로 마이닝 되었다고 판단되면 이를 디스플레이 기기로 출력할 수 있으나 반대로, 마이닝된 하이 유틸리티 패턴들이 비정상적으로 마이닝 되었다고 판단되면 다시 패턴 마이닝부(140)에서 하이 유틸리티 패턴들을 재 마이닝 하도록 제어할 수 있다. 또한, 하이 유틸리티 패턴부(150)는 패턴 마이닝부(140)에서 마이닝된 하이 유틸리티 패턴들을 사용자의 요청에 따라 기 설정된 분류별로 구분하여 디스플레이 기기로 출력할 수 있다. 여기에서, 디스플레이 기기는 PC, 스마트폰 및 태블릿 PC 를 포함한 다양한 디스플레이 기기에 해당할 수 있다.

[0081] 하이 유틸리티 패턴부(150)는 패턴 마이닝부(140)에서 마이닝된 하이 유틸리티 패턴들을 사용자의 요청에 의해 다시 패턴 마이닝부(140)에서 재 마이닝 하도록 제어할 수 있다

[0082] 본 발명의 하이 유틸리티 패턴 마이닝 장치는 의료 또는 마켓 데이터베이스들과 같은 실세계 데이터의 특성을 반영하고 있으므로 중요한 역할을 수행할 수 있다. 더욱이, 웹 클릭 스트림 분석, 모바일 커머스 환경 계획 및 리테일 스토어들 등에서 크로스 마케팅 및 생체 유전자 데이터베이스 분석 등과 같은 다른 분야에서 사용될 수 있다.

[0083] 비록 하이 유틸리티 패턴 마이닝이 내부 그리고 외부 유틸리티라는 실제 데이터베이스의 특성을 반영하고 있지만 기존의 알고리즘들은 대부분 고정된 트랜잭션 데이터베이스를 대상으로 하였다. 그러나, 실세계 응용에서는 새로 생성된 데이터 또는 다른 데이터베이스 내 데이터의 통합 등에 의해 점진적으로 증가할 수 있으며 따라서 기존의 고정된 트랜잭션 데이터베이스를 위한 알고리즘들은 이를 반영하지 못한다. 이러한 문제를 다루기 위해, 점진적 데이터베이스에서 하이 유틸리티 패턴을 마이닝 하는 알고리즘들이 제안되었다. 한편, 데이터베이스에서 마이닝 하는 것은 안티모노톤의 속성(downward closure property)을 만족하지 않아 쉽지 않은 작업이다. 즉, 낮은 유틸리티를 가지는 패턴의 슈퍼 패턴이 하이 유틸리티 패턴이 될 수 있어 미리 패턴을 제거할 수 없다. 이 문제를 해결하기 위해, 각 후보 패턴을 포함하는 모든 트랜잭션들의 유틸리티 합을 이용한 유틸리티 과추정 기법이 제안되었으며, 그 속성을 유지하기 위해 기존의 연구들은 이 기법을 적용하였다. 그 결과, 종종 많은 수의 후보 패턴 생성을 야기하였고 이로 인해 마이닝 성능이 저하되었다. 그 이유는 그 후보 패턴의 슈퍼 패턴에 포함되지 않는 아이템들의 유틸리티들까지 그 합에 더했기 때문이다. 즉, 필요 이상으로 과추정 유틸리티를 계산하여 더 많은 수의 후보 패턴을 생성했기 때문이다. 과추정 기법 기반의 프레임 워크에서, 더 많은 수의 후보 패턴은 더 많은 실행 시간을 소비한다. 특히, 점진적 데이터베이스에서 하이 유틸리티 패턴을 마이닝 하기 위한 최신 알고리즘들 또한 과추정 유틸리티 기법을 적용함으로써 많은 시간을 필요로 한다. 이러한 이유로 우리는 과추정 유틸리티를 감소시킴으로써 생성되는 후보 패턴의 수를 줄여야 한다. 즉, 본 발명은 점진적 데이터베이스에서 하이 유틸리티 패턴을 마이닝 하기 위한 프레임 워크에서 마이닝 성능을 향상시키는 것을 목표로 한다. 한편, 데이터베이스를 스캔 하는 작업은 특히 데이터베이스의 크기가 클 때 많은 시간을 필요로 하는 작업이 될 수 있다. 따라서, 점진적으로 데이터가 증가할 때마다 전체 데이터베이스를 스캔 하는 작업은 효율적이지 못하다. 즉, 실세계 응용에서는 효율적인 패턴 마이닝을 수행하기 위해 전체 데이터베이스에 대한 추가적인 스캔 없이 새로운 정보를 데이터 구조에 반영할 필요가 있다. 이는 점진적 마이닝을 위해서는 한 번의 데이터베이스 스캔으로 구축되는 자료 구조가 필요하다. 또한, 패턴 마이닝에서 데이터 구조 내 아이템 정렬 순서는 효율적인

마이닝에 중요한 기준이 된다. 따라서, 본 발명의 또 다른 목표는 한 번의 데이터베이스 스캔으로 구축되는 데이터 구조, 추가적인 데이터베이스 스캔 없이 효율적인 마이닝을 위한 정렬 순서에 따라 그 구조를 재구축 하는 방법 및 점진적으로 추가된 정보를 효과적으로 반영하기 위한 방법에 대한 기술이다. 본 발명에서는 점진적 데이터베이스를 다루기 위한 새로운 효율적인 알고리즘을 제안한다. 또한, 제안된 알고리즘은 트리 구축 및 마이닝 과정에서 감소된 과추정 유틸리티의 사용을 통해 효과적으로 후보 패턴을 감소시키며, 이를 위한 새로운 자료 구조를 사용한다. 본 발명의 주요 기여는 다음과 같이 요약된다.

[0084] 먼저, 본 발명의 일 실시예는 점진적 데이터베이스에서 하이 유틸리티 패턴을 마이닝 하기 위한 한 번 스캔으로 구축되는 트리 구조와 HUPID-Tree(High Utility Patterns in Incremental Databases Tree)라 불리는 다른 트리 구조를 제안한다. 또한, 트리의 과추정 유틸리티를 감소시키기 위한 재 구축 방법을 새로운 자료 구조, 테일 노드 정보 목록(TIList, Tail-node Information List)(이하, TIList라 함)라 하는 대조 트리와 함께 제안하며 제안된 재 구축 방법의 적용을 통해 HUPID-Tree는 한 번의 데이터베이스 스캔으로 구축된다. 더욱이, 전체 데이터베이스에 대한 추가 스캔 없이 점진적으로 추가되는 데이터를 데이터 구조에 반영할 수 있다.

[0085] 또한, 본 발명의 일 실시예는 점진적으로 추가되는 데이터베이스 내 트랜잭션들의 정보가 반영된 트리로부터 하이 유틸리티 패턴들을 마이닝 하기 위한 새로운 알고리즘, HUPID-Growth(High Utility Patterns in Incremental Databases Growth)라 하는 알고리즘을 제안한다. 제안된 알고리즘을 TIList와 함께 적용함으로써 과추정 유틸리티 추정치를 감소시키며, 이를 통해 후보 개수와 실행 시간을 감소시킬 수 있다.

[0086] 또한, 본 발명의 일 실시예는 성능 비교를 위해 의료 및 실세계 데이터 셋들을 사용한 최신 점진적 하이 유틸리티 패턴 마이닝 알고리즘과의 성능 평가를 수행한다. 실험 결과는 제안한 알고리즘이 다른 알고리즘보다, 특히 데이터베이스가 많은 수의 긴 트랜잭션을 포함하거나 낮은 임계치가 설정됐을 때, 성능이 더 나은 것을 보여줄 수 있다.

[0088] 도 2는 도 1에 있는 점진적 하이 유틸리티 패턴 마이닝 장치에서 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법을 설명하는 순서도이다.

[0089] 도 2를 참조하면, 하이 유틸리티 패턴 마이닝 장치(100)는 트랜잭션 스캔부(110), 트랜잭션 삽입부(120) 및 트리 자료구조 구축부(131)를 통해 정적 데이터베이스(초기 데이터베이스)(160)에 대한 트리 자료구조를 구축한다. 트랜잭션 스캔부(110)는 정적 데이터베이스(160)로부터 입력받은 데이터를 스캔하고 이에 해당하는 트랜잭션을 하나씩 읽어온다(S210). 트랜잭션 삽입부(120)는 정적 데이터베이스(160)의 해당 트랜잭션 아이템들을 초기 임의로 정해진 순서에 따라 정렬하고 이를 트리 자료구조에 삽입한다(S220). 트리 자료구조 구축부(131)는 트랜잭션 삽입부(120)에서 정적 데이터베이스(160)로부터 입력받은 데이터에 대해 트리 기반의 자료구조를 구축(S230)하여 트리 자료구조를 생성한다(S240).

[0090] 자료구조 재 구축부(133)는 정적 데이터베이스(160)의 트랜잭션들을 빈도수 및 최대 아이템 수량 정보와 함께 트리 자료구조에 저장하고 기 설정된 최적의 정렬순서에 따라 감소된 과추정 유틸리티를 계산하여 자료구조를 재 구축한다(S250).

[0091] 패턴 마이닝부(140)는 사용자에 의한 마이닝 요청이 발생되면 정적 데이터베이스(160)에서 구축된 자료구조에서 기 설정된 최소 유틸리티 임계치를 만족하는 모든 하이 유틸리티 패턴들을 마이닝한다(S260). 패턴 마이닝부(140)는 테일 노드 정보를 이용하여 과추정 유틸리티를 효과적으로 감소시켜 기존의 마이닝 방법들 보다 훨씬 더 적은 수의 후보 패턴들을 추출할 수 있다.

[0092] 하이 유틸리티 패턴부(150)는 패턴 마이닝부(140)에서 마이닝된 하이 유틸리티 패턴들이 기 설정된 기준에 의해 정상적으로 마이닝 되었는지 여부를 확인할 수 있다(S270). 이때, 하이 유틸리티 패턴부(150)가 마이닝된 하이 유틸리티 패턴들이 정상적으로 마이닝 되었다고 판단되면 이를 디스플레이 기기로 출력할 수 있으나 반대로, 마이닝된 하이 유틸리티 패턴들이 비정상적으로 마이닝 되었다고 판단되면 다시 패턴 마이닝부(140)에서 하이 유틸리티 패턴들을 재 마이닝 하도록 제어할 수 있다. 또한, 하이 유틸리티 패턴부(150)는 패턴 마이닝부(140)에서 마이닝된 하이 유틸리티 패턴들을 사용자의 요청에 따라 기 설정된 분류별로 구분하여 디스플레이 기기로 출력할 수 있다. 하이 유틸리티 패턴부(150)는 패턴 마이닝부(140)에서 마이닝된 하이 유틸리티 패턴들을 사용자의 요청에 의해 다시 패턴 마이닝부(140)에서 재 마이닝 하도록 제어할 수 있다

[0093] 도 2에서, 정적 데이터베이스(160)에 동적 데이터베이스(새로운 데이터)(170)가 추가되면 동적 데이터베이스

(170)에 대한 트리 자료구조를 갱신할 수 있다.

[0094] 트랜잭션 스캔부(110)에 새로운 데이터(동적 데이터베이스)(170)가 점진적으로 정적 데이터베이스(160)에 동적으로 추가되면 추가된 해당 데이터만을 스캔하고 이에 해당하는 트랜잭션을 하나씩 읽어올 수 있다(S210).

[0095] 트랜잭션 삽입부(120)는 동적 데이터베이스(170)의 해당 트랜잭션 아이템들을 현재 정렬순서에 따라 트랜잭션 내 아이템들을 정렬하고 이를 트리 자료구조에 삽입한다(S220).

[0096] 트리 자료구조 갱신부(132)는 트랜잭션 삽입부(120)를 통해 동적 데이터베이스(170)에 대한 트리 자료구조를 삽입함으로써 동적 데이터베이스(170)를 정적 데이터베이스(160)에서 기 구축한 자료구조(S230)에 반영하여 동적 데이터베이스(170)에 대한 자료구조를 갱신(S280)하고 이에 따른 트리 자료구조를 생성(S240)하여 자료구조를 재 구축(S170)한다.

[0097] 패턴 마이닝부(140)는 사용자에 의한 마이닝 요청이 발생되면 동적 데이터베이스(170)에서 갱신된 자료구조에서 기 설정된 최소 유틸리티 임계치를 만족하는 모든 하이 유틸리티 패턴들을 마이닝한다(S260).

[0098] 하이 유틸리티 패턴부(150)는 정적 데이터베이스에서 수행한 같은 과정(S260)으로 하이 유틸리티 패턴을 디스플레이로 출력하거나 재 마이닝할 수 있다(S260).

[0100] 도 3은 도 2에 있는 트리 자료구조를 설명하는 도면이다.

[0101] 도 3을 참조하면, 트리 자료구조(300)는 헤더 테이블(310), 트리(320), TIList(330) 및 최소 아이템 유틸리티 테이블(340)을 포함한다.

[0102] 헤더 테이블(310)은 각 아이템의 과추정 유틸리티(TWU) 정보를 저장할 수 있다.

[0103] 트리(320) 내 동일한 아이템에 대한 노드들은 헤더 테이블(310)을 시작으로 링크를 통해 연결될 뿐만 아니라 각 트랜잭션의 마지막 아이템에 대한 노드인 테일 노드는 TIList(330)을 통해 연결될 수 있다.

[0104] TIList(Tail-node Information List)(330)은 테일 노드 정보 목록으로 테일 노드의 빈도수(Sup) 및 트랜잭션 유틸리티(TU) 정보를 저장하고 관리할 수 있다.

[0105] 최소 아이템 유틸리티 테이블(340)은 트리 자료구조에 삽입되는 해당 트랜잭션 아이템에 대하여 정적 데이터베이스 또는 동적 데이터베이스 내 가장 작은 트랜잭션 아이템 유틸리티 값 정보를 유지할 수 있다.

[0106] 트리(320) 내 각 노드는 최대 아이템 수량 및 유틸리티 정보를 저장하고 있으며, 초기에는 트랜잭션이 삽입될 때 최대 아이템 수량만 갱신되고 재 구축 과정에서 최대 아이템 수량 정보와 TIList를 이용하여 감소된 과추정 유틸리티 정보가 노드 유틸리티에 반영될 수 있다.

[0107] 이하, 트리 자료구조에 대한 자세한 설명은 도 7 내지 도 17에서 설명한다.

[0109] 도 4는 도 2에 있는 정적 데이터베이스에 대한 트리 자료구조 구축 과정을 설명하는 순서도이다.

[0110] 도 4를 참조하면, 먼저, 초기 데이터베이스(정적 데이터베이스)(160)로부터 트랜잭션을 하나씩 읽어오며(S410), 해당 트랜잭션 아이템들을 초기 정렬순서에 따라 정렬하고(S420), 이를 트리 자료구조에 삽입한다(S430). 삽입 과정에서 각 아이템 정보는 헤더 테이블(310)에 트랜잭션 유틸리티를 기준으로 반영하여 갱신되며(S440), 특히 정렬된 트랜잭션 내 마지막 아이템에 대한 노드는 TIList(330) 내 엔트리와 연결되고, 해당 엔트리의 빈도수 및 유틸리티 정보는 각각 1 및 트랜잭션 유틸리티만큼 증가되어 갱신된다(S450).

[0111] 또한, 최소 아이템 유틸리티 테이블(340)은 각 삽입 아이템에 대하여 데이터베이스 내 가장 작은 아이템 유틸리티 값 정보를 유지하기 위해 기존에 저장되어 있던 값과 현재 삽입되는 아이템 유틸리티 값 중에서 작은 값으로 갱신되어(S460) 트리 자료구조를 생성한다(S470).

[0113] 도 5는 도 2에 있는 동적 데이터베이스에 대한 트리 자료구조 갱신 과정을 설명하는 순서도이다.

[0114] 도 5를 참조하면, 초기 정적 데이터베이스(160)에 대한 트리 자료구조 구축 및 재 구축 과정이 완료된 후, 새로운 데이터(170)가 점진적으로 추가되면 동적 트리 갱신 과정을 통해 다음의 같이 기 구축된 자료구조가 갱신된

다. 본 발명은 기 구축된 자료구조를 갱신하기 위해 새롭게 추가된 데이터(170)로부터 트랜잭션을 하나씩 읽어오며(S510), 현재 정렬순서 즉, 트리 자료구조 헤더 테이블에 저장된 정보를 이용하여 과추정 유틸리티 내림차순으로 트랜잭션 내 아이템들을 정렬하고(S520), 삽입함으로써(530) 기 구축된 자료구조를 갱신한다(S540,S550,S560). 이때, 정적 데이터베이스의 트리 자료구조 구축 과정과 마찬가지로 헤더 테이블(310), TIList(330) 및 최소 아이템 유틸리티 테이블(340)은 아이템 및 트랜잭션 정보에 따라 갱신될 수 있다. 다만, 최소 아이템 유틸리티 테이블(340)은 각 삽입 아이템에 대하여 동적 데이터베이스 내 가장 작은 아이템 유틸리티 값 정보를 유지하도록 기 저장된 아이템 유틸리티 값과 현재 삽입되는 아이템 유틸리티 값 중에서 작은 값으로 갱신할 수 있다.

[0116] 도 6은 도 2에 있는 트리 자료구조 재 구축 과정을 설명하는 순서도이다.

[0117] 도 6을 참고하면, 구축 또는 갱신된 트리 자료구조를 최적의 정렬순서로 재 구축하는 과정이다. 본 발명은 먼저, TmpTIList라 하는 임시 자료구조를 생성하고(S610), TIList의 각 엔트리를 시작으로 트리 자료구조로부터 경로를 추출하며(S620), 이를 최적의 정렬순서에 따라 재정렬하고 감소된 과추정 유틸리틀 계산한다(S530). 재정렬된 경로는 다시 삽입하며(S540) 이때, 해당 경로의 마지막 아이템에 대한 노드 정보는 테일 노드 정보 목록(TIList)가 아닌 임시 자료구조(TmpTIList)에 추가된다(S650). 모든 경로에 대한 처리가 완료되면, 기존의 원본 TIList는 제거되고(S660), TmpTIList가 새로운 TIList가 된다(S670). 재 구축된 자료구조로부터 마이닝이 재귀적으로 수행될 때, 최소 아이템 유틸리티 테이블에 저장된 정보는 마이닝 과정에서 생성되는 후보 패턴들의 과추정 유틸리티를 감소시키는데 사용되며, 이를 통해 더 적은 수의 후보 패턴들이 생성되도록 한다.

[0119] 도 7은 본 발명의 샘플 데이터베이스를 나타낸 도면이다.

[0120] 도 7를 참조하면, 점진적 데이터베이스에서의 하이 유틸리티 패턴 마이닝의 정의를 설명한다. 유틸리티 마이닝에서, 각 아이템 item ip는 단위 이윤, 외부 유틸리티라 하는 eu(ip)를 가진다. 예를 들어, 표 1은 외부 유틸리티 테이블을 나타내며, 표 1의 이윤 테이블에서, eu(B) = 7이다. 트랜잭션 Ti 내 각 아이템 ip는 내부 유틸리티라고 부르는 iu(ip) 수량과 연관된다. 예를 들어, 도 7의 데이터베이스에서 iu(B, T1) = 1이다.

표 1

[0121]

Item	A	B	C	D	E	F
External Utility	5	7	3	2	1	3

[0122] 여기에서, Utility of an item ip in a transaction Ti는 $u(i_p, T_i)$로 나타내며 내부 유틸리티 $iu(i_p)$ 및 외부 유틸리티 $eu(i_p)$의 곱 $iu(i_p) \times eu(i_p)$로 정의된다. 예를 들어, 도 7과 표 1에서 $u(B, T_1) = 1 \times 7 = 7$이다.

[0123] *Utility* of a pattern P in T_i는 $u(P, T_i)$로 나타내며 $u(i_p, T_i)$, where $i_p \in P$ 그리고 $P \subseteq T_i$, 로 정의된다. 예를 들어, 도 7에서 $u(AC, T_3) = u(A, T_3) + u(C, T_3) = 5 + 3 = 8$이다.

[0124] *Utility* of a pattern P in database D는 $u(P)$로 나타내며 $u(P, T_i)$, where $P \subseteq T_i$ 그리고 $T_i \in D$, 로 정의된다. 예를 들어, 도 7에서 $u(AC) = u(AC, T_3) + u(AC, T_8) = 8 + 24 = 32$이다.

[0125] *Minimum utility threshold* ?는 데이터베이스의 유틸리티 총합에 대한 비율로 주어진다. 예를 들어, 도 7에서 데이터베이스의 총 유틸리티는 213이며, 만약 δ가 0.25이면 최소 유틸리티 $minutil = \delta \times u(D) = 0.25 \times 213 = 53.25$이다.

[0126] 만약, 패턴 P의 유틸리티 $u(P)$가 최소 유틸리티 $minutil$보다 작지 않으면, 즉 $u(P) \geq minutil$이면 P는 하이 유틸리티 패턴이라고 부른다.

[0127] *Transaction Utility* (*TU*) of T_i는 트랜잭션 하나의 전체 이윤을 의미하는 값이고 $TU(T_i)$로 나타내며 $u(i_p, T_i)$, where $i_p \in T_i$, 로 정의된다. 예를 들어, $TU(T_3) = u(A, T_3) + u(B, T_3) + u(C, T_3) + u(D, T_3) + u(E, T_3) = 5$

+ 21 + 3 + 4 + 1 = 34이다.

[0128] 하이 유틸리티 패턴 마이닝은 데이터베이스에서 모든 하이 유틸리티 패턴을 찾는 작업을 의미한다. 또한, 하이 유틸리티 패턴 마이닝에서 안티 모노톤 속성(즉, downward closure property)을 만족시키는 것은 중요한 이슈이다. 다음 정의들은 이를 위한 *TWDC*(*Transaction Weighted Downward Closure*)모델에 관한 정의들이다.

[0129] *Transaction Weighted Utility*(*TWU*) of a pattern P in D는 패턴 P를 포함하는 모든 트랜잭션들의 트랜잭션 유틸리티 합을 의미하고 *TWU*(P)로 나타내며 $\sum TU(T_i)$, where $P \subseteq T_i$ 그리고 $T_i \in D$, 로 정의된다. 예를 들어, $TWU(AC) = TU(AC, T_3) + TU(AC, T_8) = 15 + 42 = 57$이다.

[0130] 만약, 패턴 P의 *TWU* 값이 *minutil*보다 크거나 같으면 그 패턴을 하이 트랜잭션 가중화 유틸리티 패턴이라 정의한다. 예를 들어, 최소 유틸리티 임계치 *minutil*이 53.25 (25%)일 때, 비록 패턴 AC의 유틸리티 $u(AC) = 32$이지만 $TWU(AC) = 57$이므로 하이 트랜잭션 가중화 유틸리티 패턴이며 따라서 초기에 제거되지 않는다.

[0131] 도 8은 본 발명에서 제안된 정적 및 동적 데이터베이스에 대한 하이 유틸리티 패턴 마이닝의 전체과정을 나타낸 도면이다.

[0132] 도 8을 참조하면, 본 발명의 일 실시예는 점진적 데이터베이스에서 하이 유틸리티 패턴을 마이닝 하기 위한 알고리즘, HUPID-Growth를 제안하며, 데이터베이스 내 하이 유틸리티 패턴의 정보를 유지하기 위한 트리 구조, HUPID-Tree도 제안한다.

[0133] 또한, 본 발명의 일 실시예는 재 구축 및 과추정 유틸리티 감소를 가능케 하기 위한 자료 구조, TIList 가 제안된다. 제안하는 기법의 프레임 워크는 세 단계로 구성된다. 첫 번째 단계에서, 한 번의 데이터베이스 스캔으로 전역 HUPID-Tree가 구축되고 TWU 내림차순에 따라 트리 내 노드들을 재정렬함으로써 구축된 트리가 재 구축 된다. 이때, 이전 기술들과는 달리 트리 구축 단계가 아닌 재 구축 단계에서 노드 유틸리티, 즉 감소된 과추정 유틸리티가 계산된다. 또한, 트리 구축 후에 만약 원본 데이터베이스에 점진적으로 새로운 트랜잭션들이 추가되면 새로운 트랜잭션들 내 아이템들은 현재 구축된 트리의 정렬 순서에 따라 정렬되어 삽입되고 재 구축 과정이 수행될 수 있다. 두 번째 단계에서, 후보 패턴 수의 감소를 위한 유틸리티 추정치 감소 전략이 적용된 HUPID-Growth에 의해 HUPID-Tree에서 후보 패턴들이 생성된다. 마지막으로, 실제 하이 유틸리티 패턴들이 두 번째 단계의 후보 패턴들로부터 식별된다.

[0134] 제안하는 트리 구조는 HUPID-Tree(High Utility Patterns in Incremental Databases Tree)라 불리며, 점진적 데이터베이스 내의 트랜잭션들과 하이 유틸리티 패턴들의 정보를 유지하기 위해 사용된다. 또한, 트리 내 테일 노드들의 정보를 이용하여 효율적인 재 구축을 가능케 하기 위한 자료 구조, TIList(Tail-node Information List)라고 하는 구조도 사용된다. 트랜잭션들의 아이템 수량은 트리를 구축하는 데 이용되는데, 수량 정보의 활용을 위해 TIList가 사용된다. 즉, HUPID-Tree에 저장된 수량 정보 그리고 TIList를 통해 효과적으로 과추정 유틸리티를 감소시킨다. 또한, TIList는 지역 트리 내 과추정 유틸리티 감소를 위해 사용된다. 점진적 하이 유틸리티 패턴 마이닝을 위한 최신 알고리즘은 누적된 TWU 값이 노드 유틸리티로 사용되었다. 그 결과, 마이닝 과정에서 많은 후보들이 생성되었으며 이로 인해 마이닝 성능이 감소되었다. 다음에서, HUPID-Tree의 요소들을 먼저 정의한다. 이어서, TIList에 대해 설명한다. 마지막으로, 어떻게 초기 HUPID-Tree가 구축되는지 그리고 TIList를 사용하여 어떻게 초기 HUPID-Tree가 재 구축되는지를 자세히 설명한다.

[0135] HUPID-Tree에서, 각 노드 N은 N.name, N.max, N.nu, N.parent, N.nodelink, 그리고 자식 노드들의 집합으로 구성된다. N.name은 노드의 아이템 이름이다. N.nu는 노드의 과추정 유틸리티 값을 의미한다. N.parent와 N.nodelink는 각각 N의 부모 노드와 동일한 아이템 이름을 가지는 노드와 연결하는 데 사용된다. 특히, 점진적 하이 유틸리티 패턴 마이닝을 위한 이전 트리 구조들과는 달리 N.max를 노드 요소로써 사용한다. N.max는 최대 아이템 수량을 저장하는데 감소된 유틸리티 추정치를 계산하는 데 사용되기 때문이다. 또한, 재 구축을 위해서 사용되는 이 요소는 메모리의 효율성을 위해 오직 전역 트리의 노드 요소에만 포함된다. 즉, 지역 트리의 노드 요소에는 포함되지 않는다. 대신, 지역 트리에는 서포트를 저장하기 위한 N.sup가 포함된다. 헤더 테이블 역시 트리 탐색을 용이하게 하기 위해 HUPID-Tree 내에 포함된다. 헤더 테이블 내의 각 엔트리는 아이템 이름, TWU 값, 그리고 링크로 구성된다. 모든 엔트리들은 TWU 내림차순으로 정렬된다. 링크는 동일한 아이템 이름을 가진 마지막으로 발생한 노드를 가리킨다. 헤더 테이블 내의 링크 및 노드 링크를 사용함으로써 동일한 이름의 노드들을 효율적으로 탐색할 수 있다.

[0136] TIList는 전역 HUPID-Tree 내의 테일 노드들의 정보를 저장하며 TIList 내의 각 엔트리는 링크, 서포트 및 TU로

구성된다.

[0137] 여기에서, $T_i = \{i_1, i_2, \cdots, i_k\}$를 전역 HUPID-Tree에 삽입할 데이터베이스 내 트랜잭션이라고 하자. 그러면, HUPID-Tree에 마지막으로 삽입된 T_i 내의 아이템 i_k의 노드 N_{ik}를 *tail-node*라고 정의한다. 예를 들어, 도 7의 데이터베이스에서, 정렬 순서에 따라 정렬된 트랜잭션 $T_1 = \{A, B, E, F\}$가 전역 트리에 삽입된다고 하면, 마지막으로 삽입되는 아이템 F의 노드 N_F가 *tail-node*이다. TIList 내 각 엔트리의 *링크*는 각 *tail-node*를 가리키며, *서포트*는 해당 *tail-node*로부터 루트까지의 경로 내 아이템들로 구성된 데이터베이스 내 트랜잭션의 수를 의미한다. 또한, *TU*는 각 엔트리가 나타내는 트랜잭션들의 트랜잭션 유틸리티 합을 의미한다. 예를 들어, 데이터베이스 내에 $\{A, B, C, D\}$로 구성된 트랜잭션들이 두 개 존재하면, T_i 그리고 T_j, 해당 트랜잭션들에 대한 TIList 내 엔트리는 하나가 생성되며, 그 엔트리의 서포트 값은 2로 설정된다. 그리고 그 엔트리의 TU 값은 그 두 트랜잭션들의 트랜잭션 유틸리티들의 합, $TU(T_i) + TU(T_j)$, 으로 설정된다. TIList는 전역 HUPID-Tree의 재구축 및 과추정 유틸리티 감소를 위해 사용되며, 효율적인 점진적 하이 유틸리티 패턴 마이닝을 가능케 한다. 따라서, 메모리 효율성을 위해 전역 HUPID-Tree에만 포함된다. 이 자료 구조의 자세한 활용에 대한 내용은 HUPID-Tree 구축 및 재 구축 및 전역 트리에서 조건적 트리 생성에 대한 장에서 설명한다.

[0138] 도 9는 원본 데이터베이스와 초기 HUPID-Tree의 제조과정을 나타내는 도면이다.

[0139] 도 9를 참조하면, 데이터베이스에 대한 첫 번째 스캔에서, 각 트랜잭션 T_i는 브랜치로서 초기 HUPID-Tree에 삽입된다. 이때, 각 트랜잭션의 유틸리티, $TU(T_i)$가 계산되며, 해당 트랜잭션 내 아이템들의 TWU 값이 그 계산된 TU 값을 통해 헤더 테이블에 누적된다. 트랜잭션 $T_i = \{i_1, i_2, \cdots, i_k\}$ 내의 각 아이템 i_p $(1 \le p \le k)$ 및 i_p와 연관된 수량을 q_p라고 하면, i_p는 q_p 정보와 함께 $N_{ip}.name = i_p$인 하나의 노드, N_{ip} 로서 삽입된다. 맨 처음, 아이템 i_1이 루트 노드, N_R 의 자식 노드, N_{i1} 로서 추가된다. 만약 N_R이 자식 노드로서 N_i를 가지고 있지 않으면, N_R의 밑에 N_{i1}가 생성되며 $N_{i1}.max$가 q_1로 초기화된다. 그 후, 다음 아이템 i_2가 같은 방법으로 N_{i1}의 자식 노드로서 삽입된다. 특히, 마지막 아이템 i_k의 노드 N_{ik}는 *tail-node*로서 만약 TIList와 링크로 연결되지 않았다면, TIList에 추가된다. 즉, 해당 *tail-node*가 전역 트리에서 새로 생성되었다면, 트랜잭션 T_i의 *tail-node*에 대한 정보를 포함하는 엔트리가 TIList에 추가된다. 또한, TIList 내 *tail-node* N_{ik}에 대한 서포트와 TU는 각각 1 그리고 $TU(T_i)$ 만큼 증가한다.

[0140] 예를 들어, 도 7 및 표 1의 데이터베이스와 이윤 테이블을 각각 고려한다. 초기 HUPID-Tree의 구축 작업은 $T_1 = \{A, B, E, F\}$의 삽입을 시작으로 수행되며, 이 단계에서 루트 노드 N_R은 비어 있다. 먼저, T_1의 아이템 A를 삽입하기 위해 N_A가 N_R의 밑에 자식 노드로서 생성된다. 그리고, 노드가 새로 생성되었으므로 $N_A.max$는 A의 수량인 2로 설정된다. 그 후에, 같은 방법으로 T_1 내의 다른 아이템들도 삽입된다. 이때, 마지막 아이템 F의 노드 N_F는 *tail-node*이므로 TIList에 정보가 추가된다. 먼저 N_F에 대한 엔트리가 생성되고 링크로 연결되며, 서포트와 TU는 각각 1 그리고 $TU(T_1)$ 35로 설정된다. 도 9(a)는 T_1을 삽입한 결과이다. 초기 트리의 구축을 위해 트랜잭션 내 아이템들을 삽입할 때 이미 아이템 i_p 노드 N_{ip}가 존재하면, 노드를 새로 생성하지 않고 $N_{ip}.max$ 값만 갱신시킨다. $N_{ip}.max$는 최대 수량을 저장하는 요소로서 현재 저장된 max 값과 새로 삽입되는 i_p와 연관된 수량 q_p 중에서 큰 값으로 갱신된다. 즉, $N_{ip}.max = MAX(N_{ip}.max, q_p)$이다. That is, $N_{ip}.max = MAX(N_{ip}.max, q_p)$ 또한, 만약 새로 삽입되는 트랜잭션의 *tail-node* 정보가 이미 TIList에 존재하면 해당 엔트리의 서포트와 TU 값을 1 그리고 해당 트랜잭션의 TU만큼 증가시킨다.

[0141] 예를 들어, 도 7의 원본 데이터베이스 내의 나머지 트랜잭션들 및 도 9(a)를 고려한다. 먼저, $T_2 = \{C, E\}$ 내 아이템들을 차례대로 삽입하며, 이때 T_1과 마찬가지로 N_C와 N_E를 생성하고 $N_C.max$ 그리고 $N_E.max$를 각각 2 그리고 3으로 설정한다. 또한, *tail-node*인 N_E에 대한 엔트리를 TIList에 추가하고 링크로 연결한다. 그리고 서포트과 TU 값을 각각 1 그리고 $TU(T_2)$ 15로 설정한다. 도 9(b)는 T_1-T_2을 삽입한 결과이다. 이어서 $T_3 = \{A, B, C, D, E\}$를 삽입한다. 첫 번째 아이템 A에 대한 노드 N_A가 이미 N_R의 자식 노드로서 존재하므로 노드 생성 없이 $N_A.max$

값을 $MAX(N_A.max, q_A)$로 갱신한다. 이때, 저장돼 있는 $N_A.max$ 값이 q_A보다 크므로 값이 변경되지 않는다. 다음 아이템 B 또한 N_A가 이미 N_B를 자식 노드로 가지고 있으므로 새로 생성되지 않고 $N_B.max$ 값이 $MAX(N_B.max, q_B)$로 갱신된다. N_A와는 달리 q_B가 $N_B.max$보다 크므로 $N_B.max$는 3으로 갱신된다. T_3의 나머지 아이템들은 유사한 방법으로 트리에 추가되며, 마지막 아이템 E에 대한 노드인 N_E가 트리에 새로 생성된 *tail-node*이므로 이에 대한 엔트리가 TIList에 생성된다. 그리고 N_E에 대한 TIList의 엔트리의 링크는 그 생성된 N_E를 가리키고, 서포트 그리고 TU 값은 각각 1 그리고 $TU(T_3)$ 값으로 설정된다. 마지막으로, 원본 데이터베이스 내 트랜잭션 $T_4 = \{B, C, E\}$에 대한 작업이 수행된다. T_4 내의 아이템들에 대한 노드가 존재하지 않으므로 전역 HUPID-Tree에 N_B, N_C, 그리고 N_E들이 생성되고, $N_B.max$, $N_C.max$, 그리고 $N_E.max$가 각각 4, 2, 그리고 7로 설정된다. 또한, 새로 생성된 *tail-node* N_E에 대한 TIList 내 엔트리가 다른 트랜잭션들과 같은 방법으로 생성되어 갱신된다. 도 7의 원본 데이터베이스 내의 모든 트랜잭션들 (T_1-T_4)를 삽입한 결과로 구축된 전역 HUPID-Tree는 도 9(c)와 같다. TIList와 전역 트리에 저장된 수량 정보들은 재 구축 과정에서 과추정 유틸리티를 감소시키는 데 사용된다.

[0142] HUPID-Tree의 구조조정은 다음과 같다. 초기 전역 HUPID-Tree의 구축 후, 노드 유틸리티들, 즉 감소된 과추정 유틸리티들을 계산하기 위한 재 구축 과정이 수행된다. 이를 위해 TIList와 트리 내 각 노드의 *max* 요소에 저장된 값들이 사용된다. 본 발명에서 제안하는 재 구축 방법은 DGN을 기반으로 하며, 효율적인 수행을 위해 TIList 내 각 엔트리의 *tail-node*들에 대한 링크들을 사용한다. 제안하는 방법의 적용을 통해 감소된 과추정 유틸리티를 계산함으로써 검색 공간 그리고 후보 개수를 감소시킬 수 있다. 유틸리티 마이닝의 수행을 가능케 하기 위해서 초기 전역 HUPID-Tree는 재 구축되며, 이를 위해 먼저 TLIList 내 각 엔트리가 가리키는 *tail-node*를 기준으로 루트까지의 경로 P가 추출된다. 추출된 각 경로 P는 현재 트리의 TWU 내림차순으로 재정렬된다. 이와 동시에, P 내 각 아이템의 경로 유틸리티, 즉 감소된 과추정 유틸리티가 계산된다. 이때, TIList 내 P의 *tail-node* 대한 엔트리는 TIList에서 제거되며, 재정렬된 경로 내 마지막 아이템을 *tail-node*로 하는 새로운 엔트리가 임시 TIList, TIListTmp라 불리는 곳에 저장된다. 이때, TIListTmp는 트리 재 구축 시 사용되는 임시 자료구조로서 재 구축이 완료되면 제거된다. 예를 들어, 만약 추출된 경로 $P = \langle i_1, i_2, \cdots, i_k \rangle$가 TWU 내림차순에 따라 $P_{sorted} = \langle i_{1'}, i_{2'}, \cdots, i_{k'} \rangle$로 정렬돼 삽입되면, 새로운 *tail-node* $N_{ik'}$에 대한 엔트리가 TIListTmp에 추가된다. 그리고, TIList에 저장돼 있던 P에 대한 정보가 TIListTmp 내 P_{sorted}의 $N_{ik'}$에 대한 엔트리에 저장된다. 또한, 경로 P가 트리에서 추출되는 과정에서 P 내의 노드들이 트리에서 제거될 수 있다. 먼저, *tail-node*인 N_{ik}가 자식 노드를 가지고 있지 않고 TIListTmp에 N_{ik}를 *tail-node*로 하는 엔트리가 존재하지 않으면 N_{ik}는 트리에서 제거된다. 다음에, N_{ik}가 트리에서 제거됐을 때, 만약 N_{ik-1}가 자식 노드를 가지고 있지 않고 TIList 그리고 TIListTmp에 N_{ik-1}을 *tail-node*로 하는 엔트리가 존재하지 않으면 N_{ik-1}를 트리에서 제거한다. 유사한 방법에 따라, 나머지 N_{ik-2}부터 N_{i1}까지의 노드들을 처리한다. 즉, 자식 노드가 제거된 상태일 때, 현재 노드가 자식 노드를 가지고 있지 않고 *tail-node*가 아니면 트리에서 제거된다. 따라서, 어느 한 노드가 조건을 만족하지 않아 제거되지 않으면, 그 노드의 상위 노드들을 트리에서 제거되지 않는다. 한편, FP-Growth 기반의 상향식 마이닝에서 트리 내의 상위 아이템들을 조건적 아이템으로 하여 생성되는 패턴들은 하위 아이템을 포함하지 않는다. 예를 들어, 도 9(c)에서, C를 조건적 아이템으로 하여 상향식으로 경로를 추출하여 조건적 패턴 베이스를 생성하면, 그 조건적 패턴 베이스는 아이템 A 그리고 B만으로 구성된다. 즉, C 하위 아이템인 D, E, 그리고 F는 그 조건적 패턴 베이스로부터 생성되는 패턴에는 포함되지 않는다. 따라서, $\{i_1, i_2, \cdots, i_l\}$을 TWU 내림차순으로 정렬된 아이템 순서, $TWU(i_1) \geq TWU(i_2) \geq \cdots \geq TWU(i_l)$, 라고 하면, 아이템 i_p $(1 \leq p \leq l)$에 대한 노드의 유틸리티들 $N_{ip}.nu$는 하위 아이템들에 대한 유틸리티를 포함하지 않아도 된다. 하위 아이템들에 대한 유틸리티를 상위 아이템에 대한 노드 유틸리티에서 제거하기 위해서는 아이템 수량 그리고 빈도수 정보가 필요하며, 따라서 제안하는 알고리즘은 TIList에 빈도수 그리고 전역 HUPID-Tree에 수량 정보를 저장한다.

[0143] 여기에서, E_{tail}을 tail-node N_{tail}에 대한 TIList 내의 엔트리 그리고 $\langle i_1, i_2, \cdots, i_{tail} \rangle$를 전역 HUPID-Tree에서 N_{tail}을 시작으로 추출된 경로라고 하자. 그러면, 그 경로 내의 각 아이템은 $E_{tail}.TU$보다 크지 않은 유틸리티 값을 가진다.

[0144] TIList로 연결된 tail-node N_{tail}에 대한 엔트리 E_{tail}은 $\langle i_1, i_2, \cdots, i_{tail} \rangle$ 내 아이템들로만 구성된 트랜잭션들의

정보를 포함한다. 즉, $E_{tail}.sup$ 그리고 $E_{tail}.TU$는 각각 현재 데이터베이스 내의 $\{i_1, i_2, \cdots, i_{tail}\}$로만 구성된 트랜잭션들의 개수 및 그 트랜잭션들이 가지는 유틸리티의 합을 의미한다. 한편, 임의의 트랜잭션의 유틸리티는 그 트랜잭션을 구성하는 아이템들의 유틸리티 합을 의미한다. 즉, $\{i_1, i_2, \cdots, i_{tail}\}$로 구성된 $E_{tail}.sup$ 개의 트랜잭션들 중에서 임의의 트랜잭션 $T_i = \{i_1, i_2, \cdots, i_{tail}\}$ $(1 \le i \le tail)$의 유틸리티 $TU(T_i)$는 트랜잭션 내 각 아이템들의 유틸리티 합 $u(T_i, i_1) + u(T_i, i_2) + \cdots + u(T_i, i_{tail})$이다. 그러므로, 그 트랜잭션 내 각 아이템 i_p $(i \le p \le tail)$의 유틸리티는 $TU(T_i) \ge u(T_i, i_p)$이다. 따라서, $\{i_1, i_2, \cdots, i_{tail}\}$만으로 구성된 $E_{tail}.sup$ 개의 트랜잭션들에 대하여 $\sum_{i=1}^{E_{tail}.sup} TU(T_i) \ge \sum_{i=1}^{E_{tail}.sup} u(T_i, i_p)$이다.이때, $\sum_{i=1}^{E_{tail}.sup} TU(T_i) = E_{tail}.TU$이므로 $<i_1, i_2, \cdots, i_{tail}>$ 내 각 아이템의 $E_{tail}.sup$ 개의 트랜잭션들 내 유틸리티 합은 $E_{tail}.TU$보다 크지 않다.

[0145] 여기에서, $P = <i_1, i_2, \cdots, i_k>$를 전역 트리에서 추출된 경로, $P_{sorted} = <i_{1'}, i_{2'}, \cdots, i_{k'}>$를 TWU 내림차순으로 정렬된 경로 P, 그리고 $sup(P)$ 그리고 $TU(P)$를 각각 TIList에 저장된 *tail-node* N_{ik}에 대한 엔트리의 서포트와 TU 값이라고 하자. *Path utility* of an item i_p in P_{sorted}는 $pu(i_p, P_{sorted})$라고 나타내고 $MAX(N_{ip}.max \times eu(i_p) \times sup(P) + pu(i_{p-1}, P_{sorted}), TU(P))$로 정의된다. TIList 내 각 엔트리의 *tail-node*에 대한 링크를 통해 추출된 각 경로 $P = <i_1, i_2, \cdots, i_k>$가 헤더 테이블의 TWU 내림차순에 따라 정렬되면, 정렬된 경로 $P_{sorted} = <i_{1'}, i_{2'}, \cdots, i_{k'}>$ 내 각 아이템들의 경로 유틸리티를 해당 아이템에 대한 노드에 저장된 수량, 표 1의 외부 유틸리티, 그리고 TIList에 저장된 경로 P에 대한 빈도수 $sup(P)$를 이용하여 계산한다. 이때, 경로는 트리에 삽입된 데이터베이스 내 트랜잭션 그리고 경로의 서포트는 그 경로 내 아이템들로 구성된 데이터베이스 내 트랜잭션의 수를 의미한다. 따라서, 경로 내 각 아이템의 경로 유틸리티는 그 아이템에 대한 노드의 수량과 외부 유틸리티를 곱한 해당 아이템의 유틸리티에 TIList에 저장된 경로 서포트를 곱하고, 마지막으로 같은 방법으로 계산된 상위 아이템들의 유틸리티들을 더하여 계산한다. 이때, 만약 계산된 경로 유틸리티가 TIList에 저장된 해당 경로의 TU 값보다 크면 그 경로 유틸리티는 TU 값으로 설정된다. 이를 통해, 하위 아이템들의 유틸리티를 제거하여 과추정 유틸리티를 감소시킴으로써 검색 공간 및 후보 수를 감소시킬 수 있다.

[0147] 도 10은 초기 HUPID-Tree를 사용한 TIList의 구조조정 과정을 나타내는 도면이고, 도 11은 원본 데이터베이스에 대한 HUPID-Tree 구조조정 과정을 나타내는 도면이다.

[0148] 예를 들어, 도 9(c)의 초기 전역 HUPID-Tree에서 TIList의 첫 번째 엔트리로부터 추출된 도 10(a)의 첫 번째 경로 $P = <A, B, E, F>$를 고려하자. 맨 처음에, *tail-node*인 N_F는 자식 노드를 가지고 있지 않고 임시로 생성된 TIListTmp는 비어 있으므로 트리부터 제거된다. N_F의 부모 노드인 N_E 또한 N_F가 제거된 후 자식 노드를 가지지 않으며, TIList 그리고 TIListTmp 어디에도 포함돼 있지 않으므로 트리에서 제거된다. 그러나, N_B는 N_E가 제거된 후에도 자식 노드 N_C를 가지므로 트리에서 제거되지 않는다. 이에 따라, N_B 상위에 있는 모든 노드들 또한 제거되지 않는다. 즉, N_A는 제거되지 않는다. 추출된 경로를 TWU 내림차순에 따라 재정렬하면 $P_{sorted} = <E, B, A, F>$이다. P_{sorted}의 첫 번째 아이템 E의 경로 유틸리티는 상위 아이템이 없으므로 $pu(E, P_{sorted}) = MAX(N_E.max \times eu(E) \times sup(P), TU(P)) = MAX(6 \times 1 \times 1, 35) = 6$이고, 다음 아이템 B의 경로 유틸리티 $pu(B, P_{sorted})$는 $MAX(N_B.max \times eu(B) \times sup(P) + pu(E, P_{sorted}), TU(P)) = MAX(3 \times 7 \times 1 + 6, 35) = 27$이다. 같은 방법으로, $pu(A, P_{sorted}) = 35$ 그리고 $pu(F, P_{sorted}) = 35$이다. 그 후에, 재 정렬된 경로 P_{sorted}는 계산된 경로 유틸리티들 그리고 수량 정보들과 함께 전역 HUPID-Tree에 다시 삽입된다. 먼저, 아이템 E가 경로 유틸리티 $pu(E, P_{sorted}) = 6$ 그리고 수량 $N_E.max = 6$을 가지고 삽입된다. 루트 노드 N_R은 아이템에 대한 노드 N_E를 자식 노드로 가지고 있지 않으므로 N_E를 N_R 밑에 생성하고 $N_E.nu$ 그리고 $N_E.max$를 각각 6으로 설정한다. 나머지 아이템들도 같은 방법으로 다시 트리에 삽입되며, 정렬된 경로의 tail-node인 N_F에 대한 정보는 TIListTmp에 추가된다. 즉, N_F에 대한 엔트리가 TIListTmp에 추가되며, 해당 엔트리의 서포트와 TU는 각각 TIList에 저장된 P의 $sup(P) = 1$ 그리고 $TU(P) = 35$로 설정된다. 그 후, TIList에서 P의 정보가 제거된다. 도 10(b)는 추출된 첫 번째 경로가 위의 방법을 통해 다시 삽입된 결과이다. 또한, 도 11은 도 9(c)의 원본 데이터베이스에 대한 초기 전역 HUPID-Tree의 TIList에 존재하는 모든 엔트리에 대한 정보를 이용하여 재 구축한 결과이다. 한편, 재 구축이 완료되면

TIListTmp가 TIList로 되며, 반대로 TIList는 제거된다.

[0149] 점진적 데이터베이스와 HUPID-Tree의 업데이트는 다음과 같다. HUPID-Tree 구축 후, 점진적으로 데이터베이스가 증가하여 트랜잭션이 추가되면 제안하는 알고리즘은 추가된 트랜잭션 정보만을 이미 구축된 트리에 반영한다. 즉, 전체 데이터베이스에 대한 재 스캔을 통해 트리를 새로 구축하지 않고 추가된 정보를 반영한다. 새로 추가된 트랜잭션 정보를 구축된 트리에 반영하기 위해, 먼저 각 추가된 트랜잭션 내 아이템들을 현재 구축된 트리의 TWU 내림차순에 따라 재정렬하며, 트리 구축과 유사한 방법을 통해 트리에 삽입한다. 이때, 삽입되는 트랜잭션 내 각 아이템의 외부 그리고 내부 유틸리티를 이용하여 상위 아이템들의 유틸리티들만을 포함하는 경로 유틸리티들을 계산하고 삽입할 수 있다. 이를 통해, 점진적으로 추가된 트랜잭션들의 정보를 이미 구축된 트리에 반영하고 추가적인 재 구축 없이 마이닝을 수행하는 것이 가능하다. 그러나, 새로운 트랜잭션들의 추가에 따라 트리 내 아이템들의 TWU 값이 변화하며, 따라서 TWU 내림차순 순서가 바뀔 수 있다. 그런데, 기존 연구들의 실험 결과에 따르면 트리 기반의 하이 유틸리티 패턴 마이닝에서 트리 내 노드들을 TWU로 정렬했을 때 가장 빠른 수행 속도를 보인다. 따라서, 본 발명에서 점진적인 데이터베이스에 대한 효율적인 하이 유틸리티 패턴 마이닝을 위해 추가된 트랜잭션 내 아이템들을 현재 트리의 TWU 내림차순 순서에 따라 정렬하여 삽입하며, 이때 경로 유틸리티 계산 없이 노드의 수량 정보만을 갱신한다. 그 후, 트리 내 모든 노드 유틸리티 정보를 초기화하고 재 구축을 통해 감소된 과추정 유틸리티를 다시 계산한다. 이러한 원리는 초기 전역 HUPID-Tree 구축에도 적용된다.

[0150] 예를 들어, 도 11의 원본 데이터베이스에 대한 재 구축된 전역 HUPID-Tree 그리고 도 7의 점진적으로 추가된 데이터베이스 db_1+를 고려한다. 먼저, 첫 번째 트랜잭션 $T_5 = \{A, B, D\}$를 현재 구축된 트리 내 TWU 내림차순으로 정렬하며, 정렬된 결과는 $T_5' = \{B, A, D\}$이다. 현재, TWU 내림차순에 따라 정렬된 트랜잭션은 수량 정보와 함께 트리에 삽입된다. 루트 노드 N_R이 아이템 B에 대한 노드 N_B를 가지고 있지 않으므로 자식 노드를 생성하고 $N_B.max$를 q_B 1로 설정한다. 이때, 추가된 트랜잭션들의 정보를 반영 후 재 구축하기 위해 $N_B.nu$는 0으로 초기화한다. 나머지 아이템들, A 그리고 D, 또한 아이템 노드들, N_A 그리고 N_D, 을 생성하고, 최대 수량 값 max를 각각 2 그리고 1로 설정한다. 다음 트랜잭션 $T_6 = \{B, D, F\}$ 역시 같은 방법으로 삽입하며, 도 12(a)는 도 11의 재 구축된 HUPID-Tree에 경로 유틸리티 계산 없이 db_1+를 추가한 결과이다. 한편, 점진적으로 추가된 데이터베이스 내 트랜잭션 정보를 기존의 구축된 트리에 반영한 후 재 구축 없이 마이닝 작업을 수행하기 위해 트랜잭션 삽입 시, 트랜잭션 내 각 아이템의 경로 유틸리티를 계산하여 트리에 반영할 수 있다. 추가된 데이터베이스 db_1+ 내 첫 번째 트랜잭션인 $T_5 = \{A, B, D\}$는 삽입 전 $\{B, A, D\}$로 재정렬되고, 이때 각 아이템의 수량 정보를 이용하여 트랜잭션 내 경로 유틸리티를 계산하면 첫 번째 아이템 B의 경로 유틸리티는 상위 아이템이 없으므로 $u(B, T_5) = 1 \times 7 = 7$이다. 다음 아이템 A의 경로 유틸리티는 상위 아이템 B의 경로 유틸리티 그리고 자신의 트랜잭션 내 유틸리티의 합 $u(B, T_5) + u(A, T_5) = 7 + 10 = 17$이다. 마지막으로, 아이템 D의 트랜잭션 내 경로 유틸리티는 $u(B, T_5) + u(A, T_5) + u(D, T_5) = 7 + 10 + 2 = 19$이다. 이와 같은 방법으로 db_1+에 대하여 트랜잭션 내 경로 유틸리티들을 계산하여 도 11의 재 구축된 HUPID-Tree에 트랜잭션들을 삽입한 결과는 도 12(b)와 같다. 추가된 트랜잭션들을 트리에 삽입한 후 재 구축을 수행하기 위해 트리 내 노드 유틸리티들을 초기화하며, 다음에 TIList를 이용하여 바뀐 TWU 내림차순에 따라 트리를 재 구축한다. 도 13(a)는 db_1+가 추가된 도 12의 HUPID-Tree를 재 구축한 결과이며, 도 13(b)는 재 구축된 도 13(a)에 db_2+를 추가로 삽입한 후 재 구축한 결과이다.

[0151] HUPID-Growth를 통한 마이닝 방법은 다음과 같다. 하이 유틸리티 패턴 마이닝에서, 검색 공간 그리고 후보 패턴의 수를 줄이는 것은 중대한 문제인데 그 이유는 생성된 후보 패턴들 집합에서 실제 하이 유틸리티 패턴들을 식별하는 것은 매우 시간을 소비하는 작업이기 때문이다. 본 발명에서, 전역 HUPID-Tree 내 노드들의 과추정 유틸리티를 감소시키기 위한 방법과 함께 마이닝 과정에서 지역 HUPID-Tree 내 과추정 유틸리티를 감소시키기 위한 방법을 제안한다. 제안하는 마이닝 알고리즘, HUPID-Growth(High Utility Patterns in Incremental Databases Growth)의 적용을 통해 지역 트리에서 과추정 유틸리티를 감소시킴으로써 검색 공간 및 후보 개수를 더욱 줄일 수 있다.

[0152] HUPID-Growth의 과추정 유틸리티를 감소시키기 위한 전략 및 마이닝 과정에 대해 자세히 설명은 다음과 같다. 트리 기반의 점진적 데이터베이스에 대한 하이 유틸리티 패턴 마이닝은 먼저 전역 HUPID-Tree 내 헤더 테이블에 대한 상향식 탐색을 수행한다. 이때, 트리는 최소 유틸리티 임계치보다 작지 않은 유틸리티 추정치, 즉 TWU를 가지는 아이템의 링크를 따라 탐색된다. 이는 최소 유틸리티 임계치보다 작은 TWU를 가지는 아이템은

Transaction Weighted Downward Closure Property에 의해 어떠한 하이 유틸리티 패턴도 생성하지 못하기 때문이다. 그리고 트리 내 해당 아이템에 대한 노드들로부터 루트까지의 모든 경로들을 추출하여 조건적 패턴 베이스와 트리를 구축한다.

[0153] 예를 들어, 도 13(b)의 재 구축된 전역 HUPID-Tree 및 최소 유틸리티 임계치 $minutil$ = 63.9 (30%)를 고려하자. 먼저, 알고리즘은 전역 트리의 헤더 테이블을 가장 작은 TWU 값을 가진 아이템 F를 시작으로 상향식으로 탐색한다. 아이템 F는 최소 유틸리티 임계치보다 작은 TWU 값을 가지므로, $minutil > TWU(F)$, F를 위한 조건적 패턴 베이스 및 트리가 생성되지 않는다. 다음으로, 아이템 D는 F와는 달리 최소 유틸리티 임계치보다 작지 않은 TWU 값을 가지므로, $minutil < TWU(D)$, 헤더 테이블의 링크 및 트리 내 노드 링크를 따라 트리를 탐색하여 경로들을 추출하고, D를 위한 조건적 패턴 베이스를 생성한다. 헤더 테이블의 링크와 연결된 첫 번째 아이템 노드 {D: 1, 19}부터 루트 노드까지의 첫 번째 경로 <B, A>가 추출된다. 같은 방법으로 노드 링크를 따라 <B>, <B, E, C, A>, 그리고 <E, C>가 추출된다. 표 2는 이렇게 추출된 경로들로 구성된 아이템 D에 대한 조건적 패턴 베이스이다.

표 2

[0154]

Path	Utility	Support
<B, A>	19	1
<B>	25	1
<B, E, C, A>	76	1
<E, C>	8	1

[0155] 조건적 패턴 베이스의 생성에서 추출된 각 경로는 유틸리티를 가지며, 이 값은 헤더 테이블에서 선택된 아이템 i_p에 대한 노드 N_{ip}의 노드 유틸리티 값 $N_{ip}.nu$이다. 또한, 조건적 패턴 베이스를 생성하는 과정에서 경로 내 각 아이템들의 TWU 값은 그 아이템이 포함된 경로의 유틸리티 합을 통해 누적된다. 예를 들어, 표 2에서, 아이템 B는 <B, A>, <B>, 그리고 <B, E, C, A>에서 등장하므로 아이템 D에 대한 조건적 패턴 베이스에서의 TWU 값은 각 경로의 유틸리티를 합한 19 + 25 + 76 = 120이다. 같은 방법으로, 아이템 A, C, 그리고 E의 TWU 값은 각각 95, 84, 그리고 84이다. 전역 트리의 구축에서는 최소 유틸리티 임계치보다 작은 TWU 값을 가지는 unpromising 아이템들을 제거하지 않지만, 지역 트리, 즉 조건적 패턴 트리를 구축할 때는 이러한 unpromising 아이템들을 제거한다. 그 이유는 전역 트리에서 최소 임계치보다 작은 TWU를 가지는 어떤 아이템이 데이터베이스가 점진적으로 증가함에 따라 최소 임계치보다 작지 않은 TWU를 가지게 될 수 있기 때문이다. 즉, 점진적 데이터베이스에 대한 전역 트리에서는 이 때문에 제거 작업을 수행하지 않지만, 이러한 전역 트리로부터 생성되는 지역 트리에서는 이러한 아이템들이 TWU 기반의 downward closure property에 의해 어떠한 하이 유틸리티 패턴도 생성하지 못하기 때문에 효율적인 마이닝을 위해 unpromising 아이템들을 제거해도 되기 때문이다.

[0156] 여기에서, 전역 HUPID-Tree에서, 임의의 노드 N_{ik}의 서포트는 자기 자신 그리고 N_{ik}의 자손 노드들 중에서 tail-node인 노드들의 TIList 내 서포트들을 합한 값과 같다. 여기에서, Tail-node는 Definition 9에 따라 정렬된 트랜잭션 내 마지막 아이템에 대한 노드를 의미한다. 즉, 임의의 TWU 내림차순으로 정렬된 트랜잭션 {i_1, i_2, …, i_k}에서 i_k에 대한 노드 N_{ik}가 tail-node가 되며, 이때 N_{ik}는 TIList와 링크를 통해 연결되고 {i_1, i_2, …, i_k}로만 구성된 트랜잭션 개수 정보가 TIList 내 N_{ik}에 대한 엔트리 E_{ik}에 저장된다. 또한, 동일한 아이템들로 구성된 트랜잭션이 트리에 삽입되면, E_{ik}에 저장된 정보, 서포트 $E_{ik}.sup$ 그리고 TU $E_{ik}.TU$, 가 갱신된다. 즉, 현재 데이터베이스 내에 {i_1, i_2, …, i_k}로만 구성된 트랜잭션들이 몇 개이고 그 트랜잭션들의 트랜잭션 유틸리티들의 합은 얼마인지가 갱신된다. 다시 말해서, TIList 내 E_{ik}은 링크로 연결된 tail-node N_{ik}를 시작으로 루트까지의 경로 <i_1, i_2, …, i_k> 내 아이템들, 즉 i_k를 포함한 i_k의 상위에 존재하는 아이템들로만 구성된 트랜잭션 정보를 저장하고 있다. 그러므로 만약 N_{ik}의 자식 노드 N_{ik+1}이 tail-node이면, {i_1, i_2, …, i_k, i_{k+1}}들로만 구성된 트랜잭션의 개수 정보는 $E_{k+1}.sup$에 저장된다. 그 결과, N_{ik}를 포함한 N_{ik}의 자식 노드들 중에서 tail-node인 노드들의 TIList 내 서포트들의 합은 현재 데이터베이스 내 {i_1, i_2, …, i_k}를 포함하는 모든 트랜잭션들의 수를 의미한다.

[0157] 조건적 패턴 베이스가 생성되면, 각 경로 내 아이템들을 그 조건적 패턴 베이스의 TWU 내림차순으로 정렬하고 각 아이템의 감소된 과추정 유틸리티를 계산하며, 계산된 과추정 유틸리티 그리고 경로의 서포트와 함께 트리에 삽입하여 조건적 패턴 트리를 구축한다. 이때, 전역 트리 노드에는 서포트 정보가 포함돼 있지 않으므로 전역 트리로부터 선택된 아이템에 대한 조건적 패턴 트리를 구축하기 위해 경로를 추출할 때 각 경로의 서포트는 TIList에 포함된 각 *tail-node*의 서포트 정보를 이용하여 계산한다. 즉, 현재 선택된 조건적 아이템에 대한 노드로부터 루트까지 경로를 추출할 때 그 노드를 포함하여 자손 노드들 중에서 TIList에 저장된 모든 노드들의 서포트 합이 그 경로의 서포트이다.

[0158] 예를 들어, 도 13(b)의 재 구축된 전역 HUPID-Tree 및 도 13(b)로부터 생성된 표 2의 아이템 D에 대한 조건적 패턴 베이스를 고려하자. 표 2의 첫 번째 경로, <B, A>는 D에 대한 아이템 노드 {D: 1, 19}를 시작으로 추출된다. 그 노드는 자식 노드를 가지고 있지 않고, 서포트가 1인 TIList 내 엔트리에 연결돼 있으므로 도 14(a)에서 보이는 것과 같이 경로 <B, A>의 서포트는 1이다. 두 번째 경로 <B>는 노드 {D: 1, 25}로부터 추출되며, 그 노드와 자식 노드 {F: 3, 25} 중에서 TIList에 포함된 *tail-node*는 {F: 3, 25}이고 그 *tail-node*에 대한 서포트가 1이므로 도 14(b)에서 표현된 것과 같이 경로 <B>의 서포트도 1이다. 같은 방법으로, 나머지 경로들, <B, E, C, A> 그리고 <E, C>, 의 서포트는 각각 1이다.

[0159] 여기에서, 데이터베이스 D 내 아이템 i_p의 최소 아이템 유틸리티는 $u_{Min}(i_p)$로 나타내고 $MIN(u(i_p, T_d))$, where $T_d \in D$ 그리고 $i_p \in T_d$, 로 정의한다. 예를 들어, 표 1에서, 아이템 A를 포함하는 각 트랜잭션 내에서 A의 유틸리티는 $u(A, T_1) = 10$, $u(A, T_3) = 5$, $u(A, T_5) = 10$, 그리고 $u(A, T_8) = 15$이다. 따라서, A의 최소 아이템 유틸리티는 T_3에서의 유틸리티, $u(A, T_3) = 5$이다. 즉, $u_{Min}(A) = u(A, T_3) = 5$이다. 표 3은 D 내 각 아이템의 최소 아이템 유틸리티이다.

표 3

[0160]

Item	A	B	C	D	E	F
Min	5	7	3	2	1	9

[0161] 전역 트리의 재 구축과는 달리 지역 트리의 구축에서는 삽입되는 경로 내 각 아이템들의 대한 수량 정보를 알 수 없으므로 전역 트리에서 사용되는 과추정 유틸리티 감소 방법을 사용할 수 없다. 이 문제를 극복하기 위해, 마이닝 과정에서 지역 트리를 구축할 때 과추정 유틸리티를 감소시키기 위한 RLPUT(Reducing Local Path Utilities with TIList) 전략을 제안한다. 제안하는 RLPUT 전략은 전역 트리로부터 구축된 조건적 패턴 베이스에 대하여 표 1의 아이템 이윤 테이블, TIList를 통해 계산된 조건적 패턴 베이스 내 각 경로의 서포트, 그리고 표 3의 각 아이템에 대한 최소 아이템 유틸리티를 이용하여 조건적 패턴 트리 내 노드들의 과추정 유틸리티를 감소시킨다. 즉, 조건적 지역 트리를 구축할 때 각 아이템의 감소된 과추정 유틸리티는 하위 아이템들이 가질 수 있는 최소 유틸리티, 최소 아이템 유틸리티와 경로 서포트의 곱를 제외시킨 값이다. 전역 트리로부터 추출된 경로는 현재 데이터베이스 내 트랜잭션 중에서 경로 내 아이템들을 포함하는 트랜잭션을 의미하며, TIList를 이용하여 계산된 경로의 서포트는 그 트랜잭션들의 개수를 나타낸다. 그리고 각 아이템은 해당 아이템이 포함된 각 트랜잭션에서 그 아이템의 최소 아이템 유틸리티 이상의 유틸리티를 가진다. 즉, 최소 아이템 유틸리티 그리고 경로 빈도수의 곱으로 해당 경로가 포함된 트랜잭션들에서 각 아이템이 가질 수 있는 최소 유틸리티 값을 계산할 수 있다. 따라서, 전역 트리에서 추출된 경로를 통해 지역 트리를 구축할 때 경로의 유틸리티 및 서포트 그리고 최소 아이템 유틸리티를 이용하여 감소된 과추정 유틸리티를 계산한다. 이때, 지역 트리 구축에서는 전역 트리 구축과는 달리 경로 내 각 아이템의 실제 유틸리티를 알 수 없으므로 아이템 유틸리티를 누적시키는 방식이 아닌 경로의 유틸리티에서 최소 아이템 유틸리티와 경로 서포트를 곱한 값을 차례로 감소시키는 방식을 사용한다.

[0162] 여기에서, $P = \{i_1, i_2, \cdots, i_k\}$를 경로라 할 때 $u(P)$ 그리고 $sup(P)$를 각각 P의 유틸리티 그리고 서포트라고 하자. 그러면, P 내 아이템 i_p의 지역 경로 유틸리티는 $lpu(i_p, P)$라고 나타내고 $u(P) - u_{Min}(i_j) \times sup(P)$, where $1 \le p < j \le k$, 로 정의된다. 예를 들어, 표 2의 아이템 D에 대한 조건적 패턴 베이스에서, 첫 번째 경로 <B, A> 내 아이템 A의 지역 경로 유틸리티는 하위 아이템이 없으므로 $lpu(A, <B, A>) = u(P) = 19$이다. 즉, 경로 유틸리티와 같다. 다음 아이템 B의 지역 경로 유틸리티는 하위 아이템 A가 존재하므로 해당 아이템의 최소 아이템

유틸리티 $u_{Min}(A)$와 경로의 빈도수 $sup(P)$와 곱한 값, $u_{Min}(A) \times sup(P)$, 을 경로의 유틸리티에서 뺀 값 $lpu(B, \langle B, A\rangle) = u(P) - u_{Min}(A) \times sup(P) = 19 - 5 \times 1 = 14$이다.

[0163] 조건적 패턴 베이스 구축됐을 때, 이를 통해 조건적 패턴 트리, 즉 지역 트리를 구축하는 과정은 다음과 같다. 먼저, 조건적 패턴 베이스의 경로 내 아이템들을 TWU 내림차순으로 정렬한다. 그리고, 경로의 마지막 아이템부터 경로 내 모든 아이템들의 감소된 과추정 유틸리티, 즉 지역 경로 유틸리티를 계산한다. 마지막으로 전역 트리의 구축과 유사한 방법으로 TWU 내림차순으로 정렬된 경로를 계산된 지역 경로 유틸리티 그리고 경로의 서포트와 함께 지역 트리에 삽입한다. 조건적 패턴 베이스 내 모든 경로가 삽입되면 지역 트리의 구축이 완료된다. 예를 들어, 표 2의 아이템 D에 대한 조건적 패턴 베이스를 고려하며, 초기에 D에 대한 조건적 패턴 트리는 비어 있다. 이때, 그 조건적 패턴 베이스 내 아이템들의 TWU 값은 $TWU(A) = 95$, $TWU(B) = 120$, $TWU(C) = 84$, 그리고 $TWU(E) = 84$이며, 따라서 TWU 내림차순은 $\{B, A, C, E\}$이다. 첫 번째 경로 $\langle B, A\rangle$가 삽입되기 전, TWU 내림차순으로 정렬되어 $\langle B, A\rangle$가 되고, 그 경로 내 아이템들의 지역 경로 유틸리티가 계산된다. 그 경로 내 아이템 B 그리고 A의 지역 경로 유틸리티는 각각 $lpu(B, \langle B, A\rangle) = 14$ 그리고 $lpu(A, \langle B, A\rangle) = 19$이다. 다음에, TWU 내림차순으로 정렬된 경로 $\langle B, A\rangle$는 계산된 지역 경로 유틸리티 그리고 경로의 서포트와 함께 트리에 삽입된다. 이때, D에 대한 조건적 패턴 트리는 비어 있으므로 루트 노드 밑에 N_B가 자식 노드로서 생성되며, $N_B.sup$ 그리고 $N_B.nu$는 각각 $sup(\langle B, A\rangle) = 1$ 그리고 $lpu(B, \langle B, A\rangle) = 14$로 설정된다. 다음 아이템에 대한 노드 N_A 역시 N_B의 자식 노드로서 새로 생성되며, $N_A.sup = sup(\langle B, A\rangle)$ 그리고 $N_A.nu = lpu(A, \langle B, A\rangle)$로 설정된다. 도 15(a)는 첫 번째 경로 $\langle B, A\rangle$가 삽입된 D에 대한 조건적 패턴 트리이다. 다음으로, 두 번째 경로 $\langle B\rangle$가 조건적 패턴 트리에 삽입되며, 해당 경로는 오직 하나의 아이템으로 구성돼 있으므로 경로 내 유일한 아이템 B의 지역 경로 유틸리티 $lpu(B, \langle B\rangle)$는 그 경로의 유틸리티 $u(\langle B\rangle)$와 같은 값을 가진다. 즉, $lpu(B, \langle B\rangle) = u(\langle B\rangle) = 25$이다. That is, $lpu(B, \langle B\rangle) = u(\langle B\rangle) = 25$. 두 번째 경로 내 아이템 B가 트리에 삽입될 때, 이미 루트 노드가 B를 위한 자식 노드 N_B를 가지고 있으므로 새로 생성하지 않고, $N_B.sup$ 그리고 $N_B.nu$ 값을 각각 $sup(\langle B\rangle) = 1$ 그리고 $lpu(B, \langle B\rangle)$만큼 증가시킨다. 그 결과, $N_B.sup = 2$ 그리고 $N_B.nu = 39$이다. 도 15(b)는 두 번째 경로까지 삽입한 결과이다. 이어서, 세 번째 경로 $\langle B, E, C, A\rangle$가 삽입되며, TWU 내림차순으로 정렬되어 $\langle B, A, C, E\rangle$가 된다. 경로의 마지막 아이템은 그 경로의 유틸리티와 같은 지역 경로 유틸리티를 가지므로 $lpu(E, \langle B, A, C, E\rangle) = u(\langle B, A, C, E\rangle) = 76$이다. 나머지 아이템들, C, A, 그리고 B, 은 그 값에서 최소 유틸리티씩 감소된 값을 지역 경로 유틸리티로 가진다. 따라서, $lpu(C, \langle B, A, C, E\rangle) = u(\langle B, A, C, E\rangle) - u_{Min}(E) \times sup(\langle B, A, C, E\rangle) = 76 - 1 \times 1 = 75$, $lpu(A, \langle B, A, C, E\rangle) = u(\langle B, A, C, E\rangle) - (u_{Min}(E) + u_{Min}(C)) \times sup(\langle B, A, C, E\rangle) = 76 - (1 + 4) \times 1 = 72$, 그리고 $lpu(B, \langle B, A, C, E\rangle) = u(\langle B, A, C, E\rangle) - (u_{Min}(E) + u_{Min}(C) + u_{Min}(A)) \times 1 = 76 - (1 + 3 + 5) \times 1 = 67$이다. 이때, 임의의 아이템 i_p의 한 경로 P 내의 지역 경로 유틸리티 $lpu(i_p, P)$는 바로 뒤의 아이템 i_{p+1}의 지역 경로 유틸리티 $lpu(i_{p+1}, P)$에서 그 아이템의 최소 유틸리티 $u_{Min}(i_{p+1})$ 그리고 경로의 서포트 $sup(P)$ 곱을 뺌으로써 쉽게 계산할 수 있다. 예를 들어, 위에서 세 번째 경로 $\langle B, A, C, E\rangle$의 아이템 B의 지역 경로 유틸리티는 바로 뒤따라 오는 아이템 A의 지역 경로 유틸리티 $lpu(A, \langle B, A, C, E\rangle) = 72$에서 $u_{Min}(A) \times sup(\langle B, A, C, E\rangle) = 5$를 뺀 값이다. 즉, 경로의 마지막 아이템을 시작으로 첫 번째 아이템까지 순차적으로 최소 아이템 유틸리티와 경로의 서포트 곱을 감소시킴으로써 쉽게 지역 경로 유틸리티를 계산할 수 있다. 그 후, 계산된 지역 경로 유틸리티 그리고 경로의 서포트 값을 가지고 세 번째 경로 내 아이템들이 도 8(b)의 지역 트리에 삽입된다. 아이템 B 그리고 A에 대한 아이템 노드들은 루트부터 시작하여 모두 생성돼 있으므로 새로운 노드의 생성 없이 각 아이템의 지역 경로 유틸리티와 경로의 서포트 값을 가지고 삽입되며, 이때 기존의 노드가 가지고 있던 노드 유틸리티 그리고 서포트 값이 각각 그 값만큼 증가한다. 즉, $N_B.nu = 39 + 67 = 106$, $N_B.sup = 2 + 1 = 3$, $N_A.nu = 19 + 72 = 91$, 그리고 $N_A.sup = 1 + 1 = 2$이다. 도 15(c)는 세 번째 경로까지 삽입된 결과이다. 마지막 경로 $\langle E, C\rangle$ 역시 같은 방법으로 $\langle C, E\rangle$로 정렬되며, 지역 경로 유틸리티들, $lpu(C, \langle C, E\rangle) = 7$ 그리고 $lpu(E, \langle C, E\rangle) = 8$, 그리고 경로의 서포트 $sup(\langle C, E\rangle) = 1$을 가지고 도 15(c)의 트리에 삽입된다. 도 15(d)는 표 2의 아이템 D에 대한 패턴 베이스 내 모든 경로가 삽입되어 구축된 지역 트리이다.

[0164] 지역 트리를 생성하기 위해 전역 트리의 헤더 테이블에서 선택된 최소 유틸리티 $minutil$보다 작지 않은 TWU 값을 가지는 아이템 i_g는 길이가 1인 후보 패턴 $\{i_g\}$로서 생성된다. 전역 트리로부터 지역 트리가 생성됐을 때, 해당 지역 트리가 하나 이상의 $minutil$보다 작지 않은 TWU 값을 가지는 아이템을 포함하고 있다면, 전역 트리와

같은 방법으로 헤더 테이블을 상향식으로 탐색하며 새로운 지역 트리를 생성한다. 만약, 지역 트리의 헤더 테이블에서 선택된 아이템이 i_1이라고 하면, i_g에 대한 조건적 패턴 트리 내 i_1에 대한 모든 아이템 노드 N_{i1}을 탐색하여 경로를 추출하여 $\{i_g\ i_1\}$에 대한 조건적 패턴 베이스 그리고 트리를 생성하여 같은 과정을 *minutil*보다 작지 않은 TWU 값을 가지는 아이템이 없을 때까지 재귀적으로 반복한다. 이때, 전역 트리 내 노드들은 서포트 정보를 가지고 있지 않으므로 지역 트리를 구축하기 위해 TIList를 이용하여 각 경로의 서포트를 계산했지만, 지역 트리 내 노드들은 서포트 정보를 가지고 있으므로 TIList를 사용하지 않고 헤더 테이블에서 선택된 promising 아이템, *minutil*보다 작지 않은 TWU 값을 가지는 아이템에 대한 트리 내 아이템 노드의 서포트를 그 노드부터 루트까지의 경로에 대한 서포트 값으로 사용한다.

[0165] 예를 들어, 도 15(d)의 전역 트리로부터 생성된 아이템 *D*에 대한 조건적 패턴 트리를 고려하자. 마이닝은 헤더 테이블을 상향식으로 탐색하여 진행되므로 마지막 아이템 *E*부터 시작된다. 이때, 아이템 *E*는 *minutil* 63.9보다 작지 않은 TWU 값을 가지므로 링크를 따라 트리를 탐색하며 모든 경로들, <*B*, *A*, *C*> 그리고 <*C*>, 을 추출한다. 노드 {*E*: 76, 1}을 시작으로 첫 번째 경로 <*B*, *A*, *C*>는 유틸리티 u(<*B*, *A*, *C*>) 그리고 서포트 sup(<*B*, *A*, *C*>)가 각각 $N_E.nu$ = 76 그리고 $N_E.sup$ = 1이다. 또한, {*E*: 8, 1}을 시작으로 추출된 마지막 경로 <*C*>는 u(<*C*>) = $N_E.nu$ = 8 그리고 sup(<*C*>) = $N_E.sup$ = 1이다. 그리고 {*D*, *E*}가 후보 패턴으로서 생성된다. 도 16은 두 경로를 가지고 구축된 {*D*, *E*}에 대한 조건적 패턴 트리이다.

표 4

[0166]

Promising Item	Candidate Patterns	Actual Patterns
B	{B}	{B}
E	{E}, {E, B}	{E, B}
C	{C}, {C, E}, {C, B}, {C, B, E}	{C, B}, {C, B, E}
A	{A}, {A, B}, {A, E}, {A, E, B}, {A, C}, {A, C, E}, {A, C, B}, {A, C, B, E}	{A, B}, {A, E, B}, {A, C, B}, {A, C, B, E}
D	{D}, {D, E}, {D, E, C}, {D, E, A}, {D, E, A, C}, {D, E, B}, {D, E, B, A}, {D, E, B, C}, {D, E, B, C, A}, {D, C}, {D, C, A}, {D, C, B}, {D, C, B, A}, {D, A}, {D, A, B}, {D, B}	{D, E, B, A}, {D, E, B, C, A}, {D, C, B, A}, {D, A, B}, {D, B}

[0167] 위와 같이, *minutil*가 63.9일 때 전역 트리의 헤더 테이블 내 모든 promising 아이템들에 대한 재귀적인 마이닝 과정을 통해 표 4와 같이 총 31개의 후보 패턴들이 생성된다. 후보 패턴들이 생성되면 현재 데이터베이스에 대한 마지막 스캔을 통해 최종적으로 총 13개의 실제 하이 유틸리티 패턴들이 표 4와 같이 식별된다. 도 17은 점진적인 데이터베이스에서 하이 유틸리티 패턴들을 마이닝 하기 위한 제안하는 알고리즘이다. 먼저, 원본 또는 점진적으로 추가된 데이터베이스에 대한 한 번의 스캔으로 전역 HUPID-Tree를 구축한다(lines 1~10). 데이터베이스 내 모든 트랜잭션이 추가되면 TIList를 이용하여 트리 탐색을 통해 각 경로를 추출하고 현재 정렬 순서에 따라 아이템들을 재 정렬한다(lines 18~20). 재 정렬된 경로 내 각 아이템들의 감소된 과추정 유틸리티, 즉 경로 유틸리티를 계산하고 다시 트리에 삽입한다(lines 21~28). 트리가 재 구축되면, 사용자의 요청에 따라(lines 13~15) 헤더 테이블을 상향식 방법으로 탐색하여 헤더 테이블 내 각 promising 아이템에 대한 조건적 패턴 베이스를 구축한다(lines 33~39). 이후, 현재 prefix 아이템 집합을 후보 패턴으로 추가하고, 조건적 패턴 베이스를 이용하여 조건적 패턴 트리를 만들어 재귀적으로 마이닝 과정을 수행한다(lines 40~42). 마지막으로, 생성된 후보 패턴들의 집합에서 실제 하이 유틸리티 패턴을 식별한다(line 16).

[0169] 본 발명은 점진적인 데이터베이스에서 효율적으로 하이 유틸리티 패턴을 마이닝 하기 위한 알고리즘, HUPID-Growth을 제안하였다. 또한, 트랜잭션 및 하이 유틸리티 패턴 정보를 유지하기 위한 한 번의 데이터베이스 스캔으로 구축되는 새로운 트리 구조, HUPID-Tree도 제안하여 의료 센터 또는 리테일 마켓과 같은 실세계 응용의 데이터베이스에서 지속적인 운용을 통해 새로 생성된 데이터 또는 통합 분석을 위한 추가된 다른 데이터베이스 내 데이터로 인해 크기가 점진적으로 증가되는 것을 해결할 수 있다. 제안된 트리는 TIList라 불리는 제안된 새로운 자료 구조를 통해 효율적인 마이닝을 위한 아이템 순서로 정렬되어 재 구축된다. 더욱이, 본 발명은 과추정 유틸리티 추정치를 감소시켜 효율적인 하이 유틸리티 패턴 마이닝을 가능케 하기 위한 전략들을 개발하였다. 점진적 데이터베이스 내 하이 유틸리티 패턴을 마이닝 하기 위한 최신 알고리즘들과의 성능평가를 위한 의료 및

실제 데이터 셋들을 사용한 실험결과들을 통해 제안하는 알고리즘이 효과적으로 후보 패턴의 수를 감소시킴으로써 마이닝 성능을 향상시키는 것을 알 수 있다. 게다가, HUPID-Growth는 데이터베이스들이 많은 긴 트랜잭션들을 포함하고 있거나 최소 유틸리티 임계치가 작아질 때 비교 알고리즘들보다 월등히 더 나은 성능을 보일 수 있다.

[0170] 결과적으로, 정적 및 동적 데이터베이스에 대한 점진적 하이 유틸리티 패턴 마이닝 방법은 초기 정적 데이터베이스뿐만 아니라 추후 점진적으로 증가된 동적 데이터베이스까지 고려하여 하이 유틸리티 패턴들을 효율적으로 마이닝 할 수 있다. 테일 노드 정보를 이용하여 과추정 유틸리티를 효과적으로 감소시켜 훨씬 더 적은 수의 후보 패턴들을 생성함으로써 하이 유틸리티 패턴 마이닝 성능을 향상시킬 수 있다. 과추정 유틸리티를 효과적으로 감소시켜 정적 및 동적 데이터베이스로부터 하이 유틸리티 패턴들을 이전의 기법들보다 더욱 효율적으로 마이닝 할 수 있다.

[0172] 상기에서는 본 출원의 바람직한 실시예를 참조하여 설명하였지만, 해당 기술 분야의 숙련된 통상의 기술자는 하기의 특허 청구의 범위에 기재된 본 발명의 사상 및 영역으로부터 벗어나지 않는 범위 내에서 본 출원을 다양하게 수정 및 변경시킬 수 있음을 이해할 수 있을 것이다.

[0173] [참조문헌]

[0174] 1. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proceedings of the 20th international conference on very large data bases (VLDB 1994), pp 487-499

[0175] 2. Ahmed CF, Tanbeer SK, Jeong B-S, Choi H-J (2012) Interactive mining of high utility patterns over data streams. Expert Syst Appl 39(15):11979-11991

[0176] 3. Ahmed CF, Tanbeer SK, Jeong B-S, Lee Y-K, Choi H-J (2012) Single-pass incremental and interactive mining for weighted frequent patterns. Expert Syst Appl 39(9):7976-7994

[0177] 4. Ahmed CF, Tanbeer SK, Jeong B-S, Lee Y-K (2009) Efficient tree structures for high utility pattern mining in incremental databases. IEEE Trans Knowl Data Eng 21(12):1708-1721

[0178] 5. Barber B, Hamilton HJ (2003) Extracting share frequent itemsets with infrequent subsets. Data Min Knowl Disc 7(2):153-185

[0179] 6. Caldersa T, Dextersb N, Gillisc JJM, Goethalsb B (2014) Mining frequent itemsets in a stream. Inf Syst 39:233-255

[0180] 7. Cheung D.W.-L., Han J, Ng VTY,Wong CY (1996) Maintenance of discovered association rules in large databases: an incremental updating technique. In: Proceedings of the 12th international conference on data engineering (ICDE 1996), pp 106-114

[0181] 8. Cohen L, Avrahami-Bakish G, Last M, Kandel A, Kipersztok O(2008) Real-time data mining of non-stationary data streams from sensor networks. Inf Fusion 9(3):344-353

[0182] 9. Duonga H, Truonga T, Vob B (2014) An efficient method for mining frequent itemsets with double constraints. Eng Appl Artif Intell 27:148-154

[0183] 10. Erwin A, Gopalan RP, Achuthan NR (2008) Efficient mining high utility itemsets from large datasets. In: Advances in knowledge discovery and data mining (PAKDD 2008), pp 554-561

[0184] 11. Gigli G, Boss´e ´ E., Lampropoulos GA (2007) An optimized architecture for classification combining data fusion and data-mining. Inf Fusion 8(4):366-378

[0185] 12. Gionis A,Mannila H,Mielik¨ainen T, Tsaparas P (2007) Assessing data mining results via swap randomization. ACM Trans Knowl Discov Data 1(3)

[0186] 13. Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. In: Proceedings of the 2000 ACM SIGMOD international conference on management of data, pp 1-12

[0187] 14. H¨am¨al¨ainen W, Nyk¨anen Matti (2008) Efficient discovery of statistically significant

association rules. In: IEEE international conference on data mining (ICDM), pp 203-212

[0188] 15. Hong T-P, Lee C-H, Wang S-L (2011) Effective utility mining with the measure of average utility. Expert Syst Appl 38(7):8259-8265

[0189] 16. Hong T-P, Wang C-Y, Tseng S-S (2011) An incremental mining algorithm for maintaining sequential patterns using pre-large sequences. Expert Syst Appl 38(6):7051-7058

[0190] 17. Lee G, Yun U, Ryu K (2014) Sliding window based weighted maximal frequent pattern mining over data streams. Expert Syst Appl 41(2):694-708

[0191] 18. Lee D, Park S-H, Moon S (2013) Utility-based association rule mining: a marketing solution for cross-selling. Expert Syst Appl 40(7):2715-2725

[0192] 19. Li Y-C, Yeh J-S, Chang C-C (2008) Isolated items discarding strategy for discovering high utility itemsets. Data Knowl Eng 61(1):198-217

[0193] 20. Lijffijt J, Papapetrou P, Puolam¨aki K (2014) A statistical significance testing approach to mining the most informative set of patterns. Data Min Knowl Discov 28(1):238-263

[0194] 21. Lin M-Y, Tu T-F, Hsueh S-C (2012) High utility pattern mining using the maximal itemset property and lexicographic tree structures. Inf Sci 215:1-14

[0195] 22. Lin C-W, Hong T-P, Lu W-H (2011) An effective tree structure for mining high utility itemsets. Expert Syst Appl 38(6):7419-7424

[0196] 23. Lin C-W, Lan G-C, Hong T-P (2012) An incremental mining algorithm for high utility itemsets. Expert Syst Appl 39(8):7173-7180

[0197] 24. Liu M, Qu J-F (2012) Mining high utility itemsets without candidate generation. In: International conference on information and knowledge management (CIKM 2012), pp 55-64

[0198] 25. Liu J, Wang K, Fung BCM (2012) Direct Discovery of High Utility Itemsets without Candidate Generation. In: Proceedings of the 2012 IEEE international conference on data mining (ICDM2012), pp 984-989

[0199] 26. Liu Y, Liao W-K, Choudhary AN (2005) A two-phase algorithm for fast discovery of high utility itemsets. In: Advances in knowledge discovery and data mining (PAKDD 2005), pp 689-695

[0200] 27. Mallick B, Garg D, Grover PS (2013) Incremental mining of sequential patterns: Progress and challenges. Int Data Anal 17(3):507-530

[0201] 28. Palmieri F, Ciuonzo D (2013) Objective priors from maximum entropy in data classification. Inf Fusion 14(2):186-198

[0202] 29. Pisharath J, Liu Y, Ozisikyilmaz B, Narayanan R, Liao WK, Choudhary A, Memik G NU-MineBench version 2.0 dataset and technical report. http://cucis.ece.northwestern.edu/projects/DMS/MineBench.html

[0203] 30. Pyun G, Yun U, Ryu K (2014) Efficient frequent pattern mining based on linear prefix tree. Knowl Based Syst 55:125-139

[0204] 31. Pyun G, Yun U (2014) Mining top-k frequent patterns with combination reducing techniques. Appl Intell 41(1):76-98

[0205] 32. Ryang H, Yun U, Ryu K (2014) Discovering high utility itemsets with multiple minimum supports. Intelligent data analysis. (In Press)

[0206] 33. Shie B-E, Hsiao H-F, Tseng VS (2013) Efficient algorithms for discovering high utility user behavior patterns in mobile commerce environments. Knowl Inf Syst 37(2):363-387

[0207] 34. Shie B-E, Yu PS, Tseng VS (2012) Efficient algorithms for mining maximal high utility itemsets

from data streams with different models. Expert Syst Appl 39(17):12947-12960

[0208] 35. Shie B-E, Hsiao H-F, Tseng VS, Yu PS (2011) Mining high utility mobile sequential patterns in mobile commerce environments. In: Database systems for advanced applications (DASFAA 2011), pp 224-238

[0209] 36. Song W, Liu Y, Li J (2014) Mining high utility itemsets by dynamically pruning the tree structure. Appl Int 40(1):29-43

[0210] 37. Tanbeer SK, Ahmed CF, Jeong B-S, Lee Y-K (2009) Efficient single-pass frequent pattern mining using a prefix-tree. Inf Sci 179(5):559-583

[0211] 38. Tseng VS, Shie B-E, Wu C-W, Yu PS (2013) Efficient algorithms formining high utility itemsets from transactional databases. IEEE Trans Knowl Data Eng 25(8):1772-1786

[0212] 39. Tseng VS, Wu C-W, Shie B-E, Yu PS (2010) UP-Growth: an efficient algorithm for high utility itemset mining. In: Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining (KDD 2010), pp 253-262

[0213] 40. Vo B, Coenen F, Le Bac (2013) A new method for mining frequent weighted itemsets based on wit-trees. Expert Syst Appl 40(4):1256-1264

[0214] 41. Wen Y, Bein D, Phoha S (2014) Dynamic clustering of multimodal sensor networks in urban scenarios. Inf Fusion 15:130-140

[0215] 42. Wu C-W, Lin Y-F, Yu PS, Tseng VS (2013) Mining high utility episodes in complex event sequences. In: Knowledge discovery and data mining (KDD 2013), pp 536-544

[0216] 43. Wu C-W, Fournier-Viger P, Yu PS, Tseng VS (2011) Efficient mining of a concise and loss-less representation of high utility itemsets. In: The 11th IEEE international conference on data mining (ICDM 2011), pp 824-833

[0217] 44. Yeh J-S, Li Y-C, Chang C-C (2007) Two-phase algorithms for a novel utility-frequent mining model. In: Emerging technologies in knowledge discovery and data mining (PAKDD 2007), pp 433-444

[0218] 45. Yen S-J, Lee Y-S, Wang C-K (2014) An efficient algorithm for incrementally mining frequent closed itemsets. Appl Int 40(4):649-668

[0219] 46. Yin J, Zheng Z, Cao L (2012) USpan: an efficient algorithm for mining high utility sequential patterns. In: Knowledge discovery and data mining (KDD 2012), pp 660-668

[0220] 47. Yun U, Ryu K (2013) Efficient mining of maximal correlated weight frequent patterns. Int Data Anal 17(5):917-939

[0221] 48. Yun U, Lee G, Ryu K (2014) Mining maximal frequent patterns by considering weight conditions over data streams. Knowl Based Syst 55:49-65

[0222] 49. Yun U, Ryang H, Ryu K (2014) High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates. Expert Syst Appl 41(8):3861-3878

[0223] 50. Yu L, Huang W, Wang S, Lai KK (2008) Web warehouse - a new web information fusion tool for web mining. Inf Fusion 9(4):501-511

부호의 설명

[0224] 100: 하이 유틸리티 패턴 마이닝 장치

110 : 트랜잭션 스캔부 120 : 트랜잭션 삽입부

130 : 트리 자료구조 모듈 131 : 트리 자료구조 구축부

132 : 트리 자료구조 갱신부 133 : 자료구조 재 구축부

140 : 패턴 마이닝부 150 : 하이 유틸리티 패턴부

160 : 정적 데이터베이스　　170 : 동적 데이터베이스

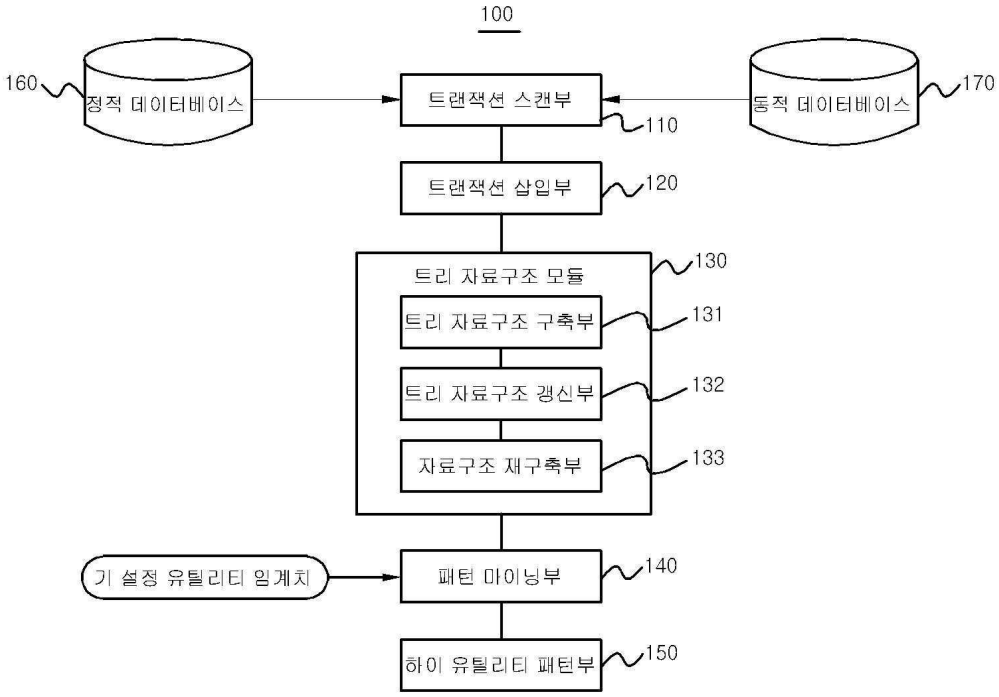
300 : 트리 기반 자료구조　　310 : 헤더 테이블

320 : 트리　　330 : TIList

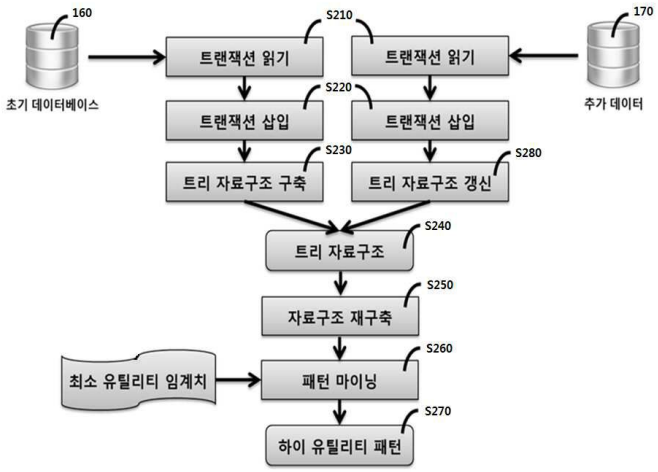
340 : 최소 아이템 유틸리티 테이블

도면

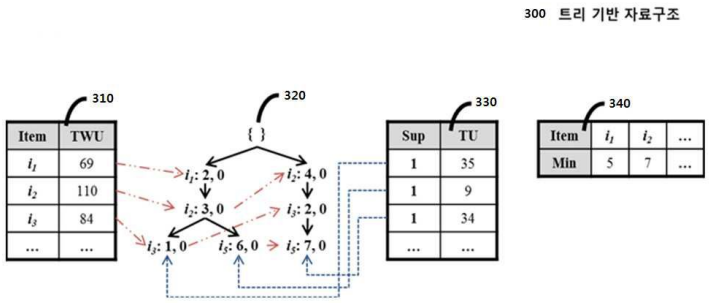
도면1



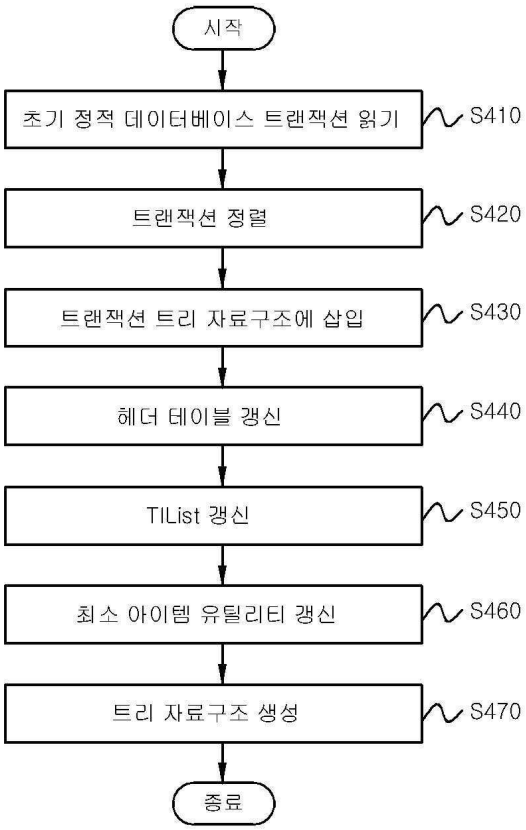
도면2


160
초기 데이터베이스
S210
트랜잭션 읽기
트랜잭션 읽기
170
추가 데이터
S220
트랜잭션 삽입
트랜잭션 삽입
S230
트리 자료구조 구축
S280
트리 자료구조 갱신
S240
트리 자료구조
S250
자료구조 재구축
S260
최소 유틸리티 임계치
패턴 마이닝
S270
하이 유틸리티 패턴

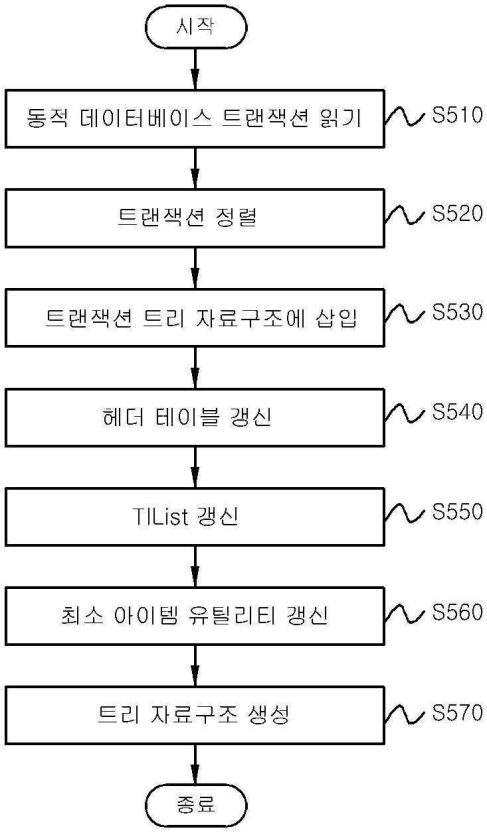
도면3


300 트리 기반 자료구조
310
Item TWU
i_1 69
i_2 110
i_3 84
... ...
320
{ }
i_1: 2, 0
i_2: 4, 0
i_2: 3, 0
i_3: 2, 0
i_3: 1, 0
i_5: 6, 0
i_5: 7, 0
330
Sup TU
1 35
1 9
1 34
... ...
340
Item i_1 i_2 ...
Min 5 7 ...

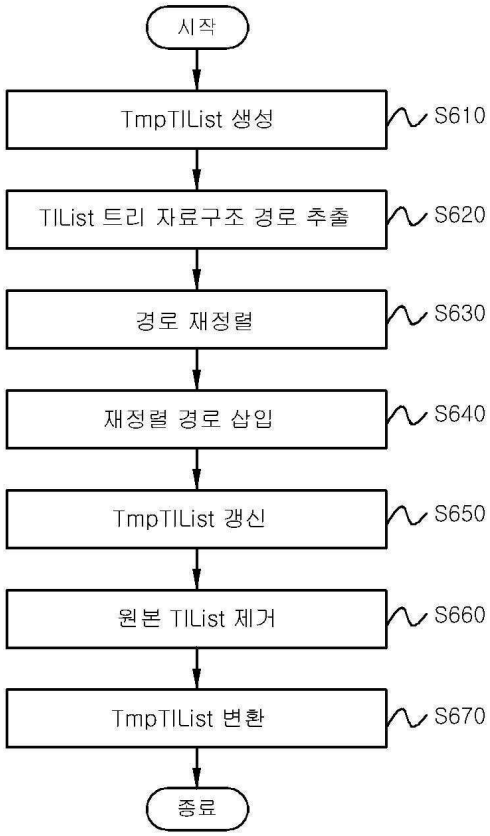
도면4


시작
초기 정적 데이터베이스 트랜잭션 읽기
S410
트랜잭션 정렬
S420
트랜잭션 트리 자료구조에 삽입
S430
헤더 테이블 갱신
S440
TIList 갱신
S450
최소 아이템 유틸리티 갱신
S460
트리 자료구조 생성
S470
종료

도면5


시작
동적 데이터베이스 트랜잭션 읽기
S510
트랜잭션 정렬
S520
트랜잭션 트리 자료구조에 삽입
S530
헤더 테이블 갱신
S540
TIList 갱신
S550
최소 아이템 유틸리티 갱신
S560
트리 자료구조 생성
S570
종료

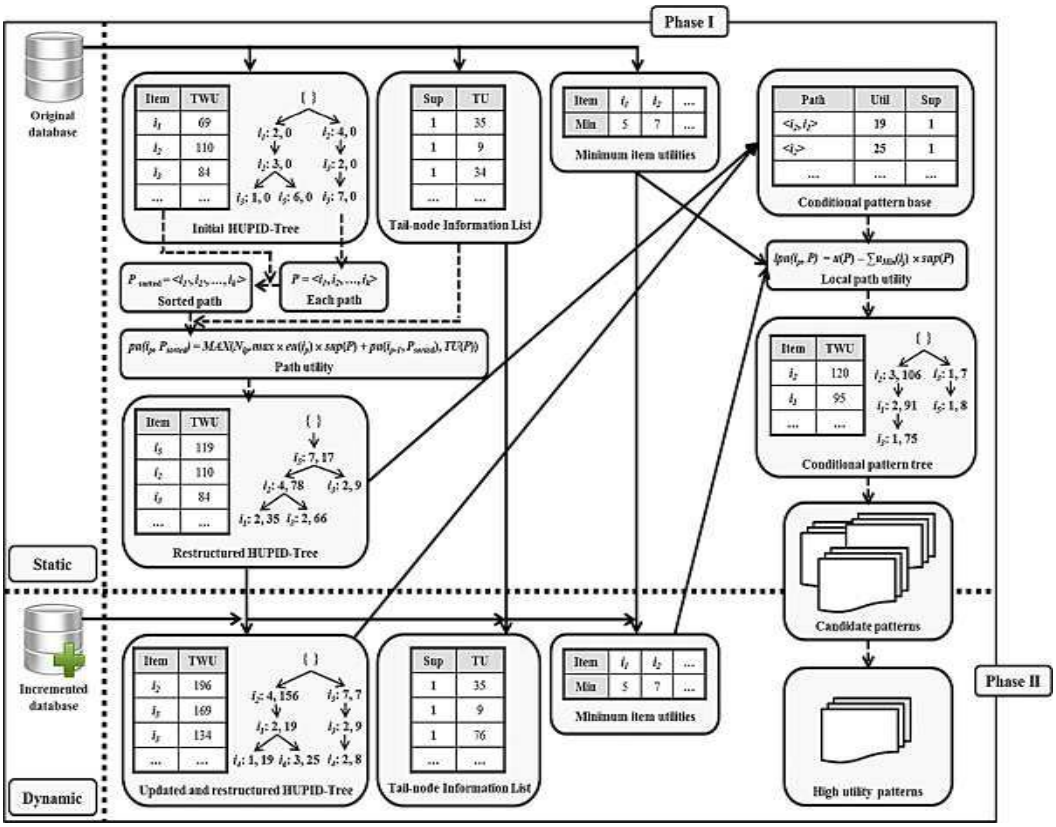
도면6



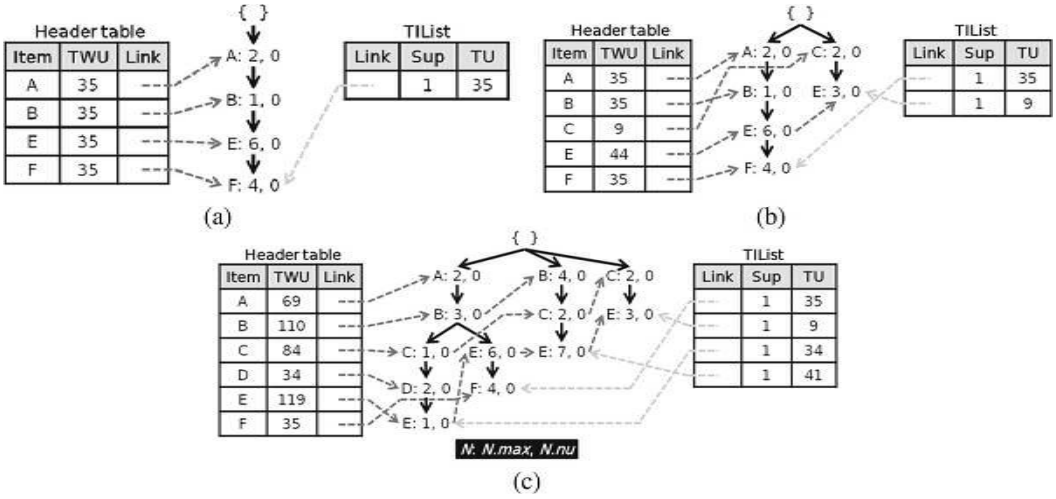
도면7

	TID	Transaction	TU
Original DB	T_1	(A, 2) (B, 1) (E, 6) (F, 4)	35
	T_2	(C, 2) (E, 3)	9
	T_3	(A, 1) (B, 3) (C, 1) (D, 2) (E, 1)	34
	T_4	(B, 4) (C, 2) (E, 7)	41
db_1^+	T_5	(A, 2) (B, 1) (D, 1)	19
	T_6	(B, 2) (D, 1) (F, 3)	25
db_2^+	T_7	(C, 1) (D, 2) (E, 1)	8
	T_8	(A, 3) (B, 2) (C, 3) (D, 1) (E, 2)	42

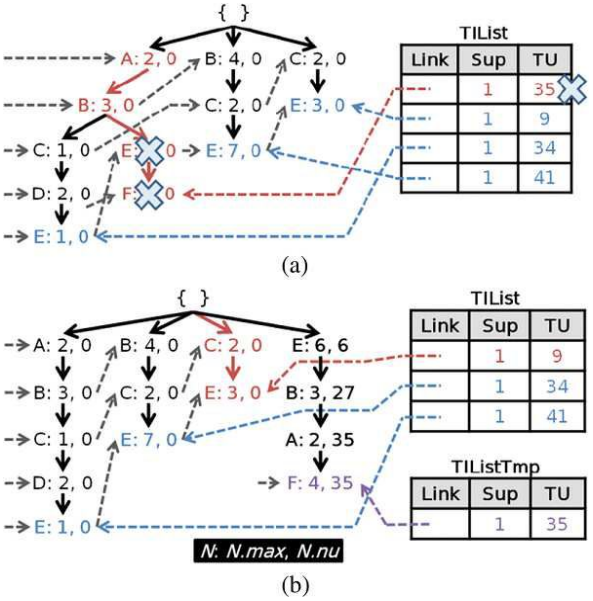
도면8


Phase I
Original database
Item TWU
Initial HUPID-Tree
Sup TU
Tail-node Information List
Item Min
Minimum item utilities
Path Util Sup
Conditional pattern base
Sorted path
Each path
Local path utility
Path utility
Item TWU
Conditional pattern tree
Restructured HUPID-Tree
Static
Incremented database
Candidate patterns
Phase II
Updated and restructured HUPID-Tree
High utility patterns
Dynamic

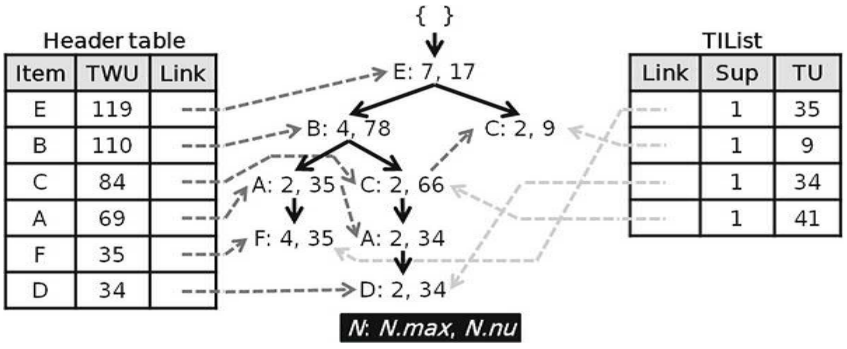
도면9


Header table
Item TWU Link
TIList
Link Sup TU
A: 2, 0
B: 1, 0
E: 6, 0
F: 4, 0
(a)
C: 2, 0
E: 3, 0
(b)
B: 4, 0
C: 1, 0
D: 2, 0
E: 1, 0
E: 7, 0
N: N.max, N.nu
(c)

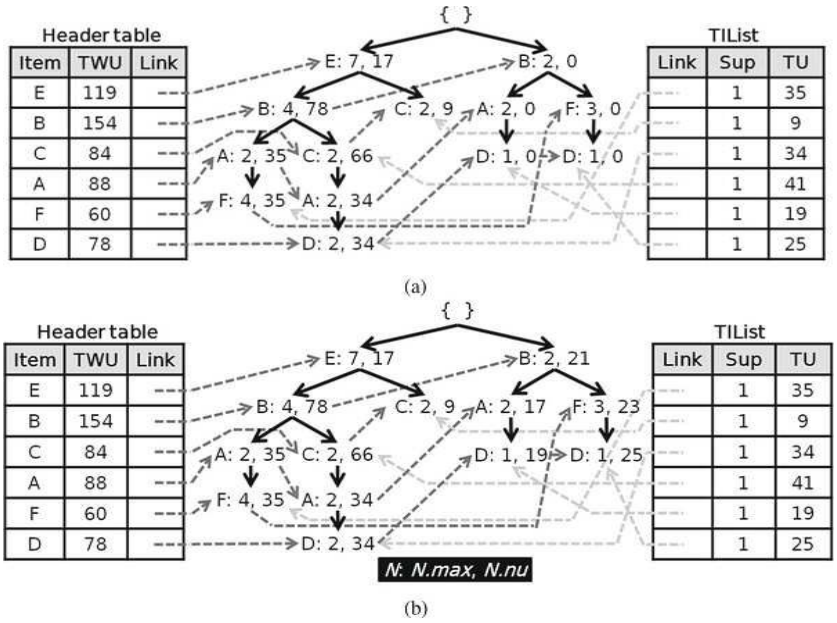
도면10



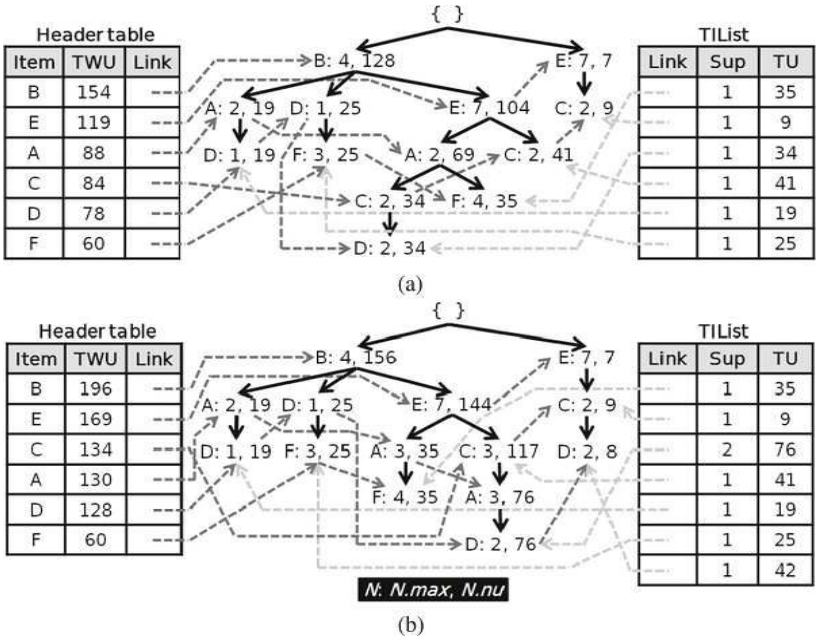
도면11



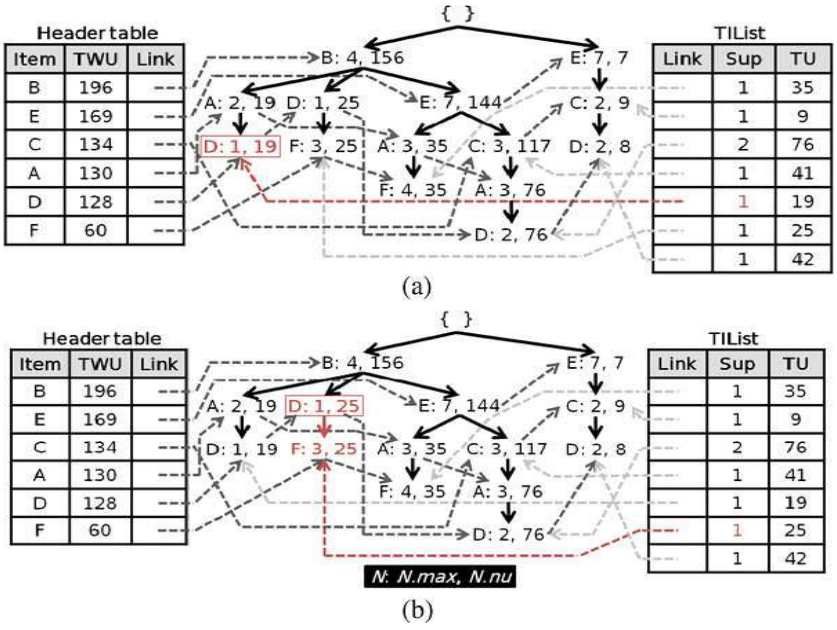
도면12



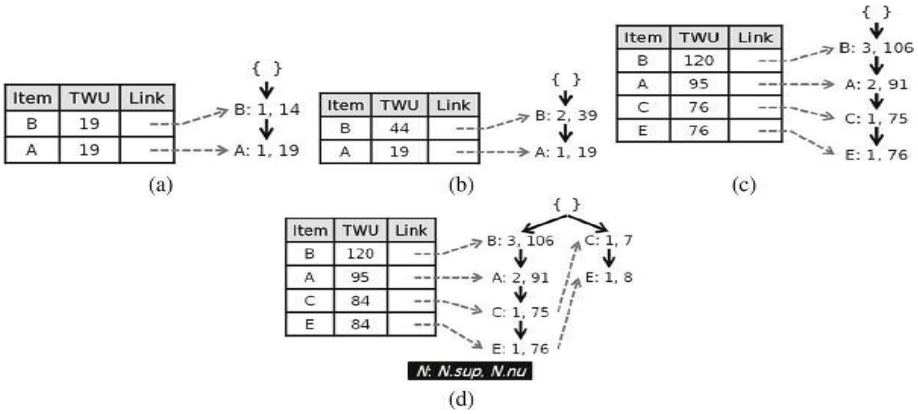
도면13


Header table
Item TWU Link
B 154
E 119
A 88
C 84
D 78
F 60
{ }
B: 4, 128
E: 7, 7
A: 2, 19
D: 1, 25
E: 7, 104
C: 2, 9
D: 1, 19
F: 3, 25
A: 2, 69
C: 2, 41
C: 2, 34
F: 4, 35
D: 2, 34
TIList
Link Sup TU
1 35
1 9
1 34
1 41
1 19
1 25
(a)
Header table
Item TWU Link
B 196
E 169
C 134
A 130
D 128
F 60
{ }
B: 4, 156
E: 7, 7
A: 2, 19
D: 1, 25
E: 7, 144
C: 2, 9
D: 1, 19
F: 3, 25
A: 3, 35
C: 3, 117
D: 2, 8
F: 4, 35
A: 3, 76
D: 2, 76
TIList
Link Sup TU
1 35
1 9
2 76
1 41
1 19
1 25
1 42
N: N.max, N.nu
(b)

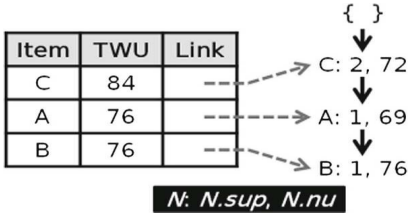
도면14


Header table
Item TWU Link
B 196
E 169
C 134
A 130
D 128
F 60
{ }
B: 4, 156
E: 7, 7
A: 2, 19
D: 1, 25
E: 7, 144
C: 2, 9
D: 1, 19
F: 3, 25
A: 3, 35
C: 3, 117
D: 2, 8
F: 4, 35
A: 3, 76
D: 2, 76
TIList
Link Sup TU
1 35
1 9
2 76
1 41
1 19
1 25
1 42
(a)
Header table
Item TWU Link
B 196
E 169
C 134
A 130
D 128
F 60
{ }
B: 4, 156
E: 7, 7
A: 2, 19
D: 1, 25
E: 7, 144
C: 2, 9
D: 1, 19
F: 3, 25
A: 3, 35
C: 3, 117
D: 2, 8
F: 4, 35
A: 3, 76
D: 2, 76
TIList
Link Sup TU
1 35
1 9
2 76
1 41
1 19
1 25
1 42
N: N.max, N.nu
(b)

도면15


Item TWU Link
B 19 --
A 19 --
{ }
B: 1, 14
A: 1, 19
(a)
Item TWU Link
B 44 --
A 19 --
{ }
B: 2, 39
A: 1, 19
(b)
Item TWU Link
B 120 --
A 95 --
C 76 --
E 76 --
{ }
B: 3, 106
A: 2, 91
C: 1, 75
E: 1, 76
(c)
Item TWU Link
B 120 --
A 95 --
C 84 --
E 84 --
{ }
B: 3, 106
C: 1, 7
A: 2, 91
E: 1, 8
C: 1, 75
E: 1, 76
N: N.sup, N.nu
(d)

도면16


Item TWU Link
C 84 --
A 76 --
B 76 --
{ }
C: 2, 72
A: 1, 69
B: 1, 76
N: N.sup, N.nu

도면17

Input: the original database ***DB***, an incremental database ***db***, a minimum utility threshold δ
Output: a set of high utility patters ***HUP***

```
/* a global HUPID-Tree Tree, the root node N_R, a tail-node information list TIList, a prefix itemset PFI */
01. For each transaction T do
02.     N := N_R
03.     Sort items in T according to the current sort order
04.     For each item i with a quantity value q in T do
05.         If N has no child node N_i such that N_i.name = i then
06.             Create a new child node N_i under N
07.             Set N_i.name := i, N_i.max := 0, N_i.nu := 0
08.         If N_i.max < q then N_i.max := q
09.         If i is the last item in T then
10.             Update N_i information in TIList with transaction utility of T, TU(T)
11.     If T is the last transaction of DB or db then
12.         Call Restructure_Tree(Tree)
13. If there is a mining request from a user then
14.     PFI := φ and HUP := φ
15.     Call Mine(Tree, PFI, HUP)
16.     Remove low utility patterns from HUP // Phase II
```

Procedure *Restructure_Tree(Tree)*

```
/* a temporal tail-node information list TIListTmp */
17. TIListTmp := φ
18. For each entry E in TIList do
19.     Extract a path P from Tree with E.link
20.     Sort P according to TWU descending order
21.     N := N_E
22.     For each node N_p for an item i_p in P do
23.         pu := MAX(N_p.max, eu(i_p) + E.sup + pu(i_{p-1}, P), E.TU)
24.         If N has no child node N_p such that N_p.name = i_p then
25.             Create a new child node N_p under N
26.             Set N_p.name := i_p, N_p.max := 0, N_p.nu := 0
27.         If N_p.max < q then N_p.max := q
28.         N_p.nu := N_p.nu + pu
29.         If N_p is the last node in P then
30.             Update N_p information in TIListTmp with E.TU
31.     Remove E from TIList
32. Remove TIList and Set TIListTmp as TIList
```

Procedure *Mine(Tree, PFI, HUP)*

```
/* a conditional pattern base CPB, a conditional pattern tree CPT, a conditional prefix itemset CPI */
33. For each item i in a header table of Tree such that i.twu ≥ minutil // bottom-up manner
34.     CPB := φ, CPT := φ, CPI := PFI ∪ {i}
35.     For each node N in Tree such that N.name = i do
36.         Extract a path P from N to N_R
37.         If Tree is a global tree then Calculate a path support of P, sup(P) with descendant nodes and
TIList
38.         Else sup(P) = N.sup
39.         Add P information to CPB with sup(P) and N.nu
40.     Create a conditional pattern tree CPT for CPI without unpromising items in CPB // using
Definition 12
41.     Add CPI to HUP as a candidate itemset
42.     Call Mine(CPT, CPI, HUP) // recursive call
```