KIPS Tr. Software and Data Eng.
Vol.4, No.11 pp.499~508 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2015.4.11.499

High Utility Itemset Mining Using Transaction Utility of Itemsets

Serin Lee† · Jong Soo Park††

ABSTRACT

High utility itemset(HUI) mining refers to the discovery of itemsets with high utilities which are not less than a user-specified minimum utility threshold, by considering both the quantities and weight factors of items in a transaction database. Recently the utility-list based HUI mining algorithms have been proposed to avoid numerous candidate itemsets and the algorithms need the costly join operations. In this paper, we propose a new HUI mining algorithm, using the utility-list with additional attributes of transaction utility and common utility of itemsets. The new algorithm decreases the number of join operations and efficiently prunes the search space. Experimental results on both synthetic and real datasets show that the proposed algorithm outperforms other recent algorithms in runtime, especially when datasets are dense or contain many long transactions.

Keywords : Data Mining, High Utility Pattern Mining, High Utility Itemsets, Transaction Utility

항목집합의 트랜잭션 유틸리티를 이용한 높은 유틸리티 항목집합 마이닝

이 세 린† · 박 종 수††

요 약

높은 유틸리티 항목집합 마이닝은 트랜잭션 데이터베이스에서 사용자가 지정한 최솟값 이상의 유틸리티를 갖는 항목집합들을 항목의 수량과 가중치값을 동시에 고려하여 찾아내는 것이다. 최근에 연구된 유틸리티-리스트 기반의 높은 유틸리티 항목집합 마이닝 알고리즘은 많은 후보 항목집합들을 피하기 위해 제안되었으며 비용이 높은 조인 연산을 수행한다. 본 논문은 유틸리티-리스트 구조에 항목집합의 트랜잭션 유틸리티와 공통 유틸리티 속성을 추가한 새로운 알고리즘을 제안한다. 이 새로운 알고리즘은 조인 연산의 수를 줄이고 탐색 공간을 효과적으로 가지치기한다. 생성 데이터와 실 환경 데이터상의 실험 결과를 통해 제안된 알고리즘이 다른 최근 알고리즘들에 비해 실행 시간 면에서 아주 우수하고, 특히 데이터가 조밀하거나 항목집합의 길이가 긴 경우에 더 효율적이라는 것을 보여준다.

키워드 : 데이터 마이닝, 높은 유틸리티 패턴 마이닝, 높은 유틸리티 항목집합, 트랜잭션 유틸리티

1. 서 론

데이터 마이닝은 대용량 데이터베이스로부터 유용하고 중요한 패턴이나 정보를 추출하는 프로세스이다. 이 기법은 인터넷과 소셜 네트워크 서비스의 사용 결과로 요즘 많은 관심을 받고 있는 빅 데이터와 기계 학습 분야의 기본 연산 기법으로써 중요한 역할을 하고 있다. 데이터 마이닝의 여러 기법들 중에서 연관규칙 탐사는 이미 마케팅 프로모션, 재고 예측 및 관리, 고객정보 분석과 같이 많은 양의 데이터를 관리하고 이용하는 분야에서 널리 상용화되고 있다. 이러한 연관 규칙 탐사는 주어진 최솟값 이상의 지지도를 갖는 빈발 항목집합들을 찾는 단계와 이 항목집합들로부터 연관 규칙을 생성하는 단계로 나뉘는데, 주로 빈발 항목집합을 찾는 단계에 많은 연구가 이루어져 왔다[1, 2].

사용자가 지정한 최소 지지도로 빈발한 항목집합들을 찾아내는 빈발 패턴 탐사(Frequent Pattern Mining)는 높은 유틸리티 항목집합 마이닝이 등장하기 이전에 항목의 존재 유무만으로 마이닝하는 초기 패턴 마이닝 기법이다. 이는 데이터베이스의 트랜잭션을 구성하는 각 항목의 중요도를 동일하게 간주하고 항목의 개수를 고려하지 않기 때문에 근본적으로 현실을 충분히 반영하지 못한다는 한계를 가진다. 이런 한계를 극복하기 위하여 항목의 중요도를 반영하는 가

※ 이 논문은 2014년도 성신여자대학교 학술연구조성비 지원에 의하여 연구되었음.
† 준 회 원 : 성신여자대학교 컴퓨터학과 석사과정
†† 종신회원 : 성신여자대학교 IT학부 교수
Manuscript Received : June 16, 2015
First Revision : August 3, 2015
Accepted : August 4, 2015
* Corresponding Author : Jong Soo Park(jpark@sungshin.ac.kr)

중치 빈발 패턴 탐사(Weighted Frequent Pattern Mining)[3, 4]로 연구를 확장하였으나, 이때까지만 해도 각 항목의 가중치만 반영하고 항목의 수량은 고려하지 않았다.

위 단계를 거쳐 연구는 각 항목의 가중치와 항목의 수량을 동시에 고려하는 유틸리티 패턴 마이닝(Utility Pattern Mining)으로 진전되어 패턴 마이닝은 많은 비즈니스 업무에 직접적으로 응용할 수 있는 실용성을 가지게 되었다. 유틸리티 패턴 마이닝[5-14]은 주어진 최소 유틸리티 이상의 값을 갖는 높은 유틸리티 항목집합들(High Utility Itemsets, HUI)을 찾아내는 기법으로, 현재 많은 조직들이 마케팅 전략과 추천 서비스 등의 핵심 기술로 적용하였고 이는 비즈니스의 확장과 높은 경제적 이익에 직접적인 영향을 줄 수 있어 그 중요성이 더욱 부각되고 있다.

본 논문에서는 높은 유틸리티 패턴을 효율적으로 탐사하기 위하여 최근에 제안된 유틸리티-리스트 구조 기반의 방법론을 더 개선시키는 방안들을 연구하였고, 그 결과로 새로운 유틸리티 패턴 탐사 알고리즘인 TUL-Miner(Transaction Utility List based HUI Miner)를 제안한다. 본 논문에서 기여하는 것은 아래와 같다:

- 효율적으로 HUI를 탐색하기 위해 확장된 유틸리티-리스트 구조와 해시 비트벡터 구조를 사용하는 TUL-Miner 알고리즘을 제안한다.
- 항목집합 트랜잭션 유틸리티(Transaction Utility, TU) 값을 이용하여 조인 연산의 횟수를 효과적으로 줄이는 두 가지 전략과 항목집합의 공통 유틸리티를 이용하여 유틸리티의 계산 시간을 단축시키는 한 가지 전략을 제안하였다.
- TUL-Miner의 성능을 평가하기 위하여 생성 데이터와 실 환경 데이터에서 높은 유틸리티 패턴을 탐사하는 실험을 하였고 그 실험 결과를 통해 제안된 알고리즘의 성능이 기존의 알고리즘들보다 개선되었음을 보여주었다.

본 논문의 구성은 다음과 같다. 2절에서는 문제를 정의하고, 이를 바탕으로 3절에서는 관련 연구를 소개한다. 그리고 제안된 알고리즘과 관련된 자료 구조 및 함수들은 4절에서 상세히 설명하였다. 5장에서는 제안된 알고리즘의 성능 측정을 위한 실험 과정과 결과에 대해 설명하고, 6절에서는 결론을 맺는다.

2. 문제 정의

이 절에서는 높은 유틸리티 항목집합 마이닝을 위한 기본 개념과 관련된 정의를 설명한다[5-14]. 항목들의 집합 $I=\{i_1, i_2, \cdots, i_m\}$에 대하여 각 항목 $i_p(1 \le p \le m)$는 수량 $q(i_p)$와 단위 가격 $pr(i_p)$를 가지며 이는 각각 Table 1과 2에서 찾을 수 있다. k개의 서로 다른 항목 $\{i_1, i_2, \cdots, i_k\}$ $(i_j \in I, 1 \le j \le k)$을 갖는 I의 부분집합 X는 항목집합 X라고 명명한다. 트랜잭션 데이터베이스 D는 트랜잭션들의 집합 $D=\{T_1, T_2, \cdots, T_n\}$이다. 트랜잭션 T_d는 고유번호 d로 구분되며 이를 Tid라고 지칭한다. T_d는 항목집합과 각 항목의 수량을 가지며, T_d에 속한 항목 i_p의 개수는 $q(i_p, T_d)$로 나타낸다.

Table 1. Price Table

Item	a	b	c	d	e	f
Price	4	2	1	3	1	5

Table 2. An Example Database

Tid	Transaction	count	TU
T_1	{a, b, c, d}	{1, 2, 3, 2}	17
T_2	{b, c, e}	{4, 5, 8}	21
T_3	{a, b, c, d}	{2, 1, 2, 1}	15
T_4	{a, b, d, e}	{2, 2, 1, 3}	18
T_5	{b, c, e}	{3, 7, 7}	20
T_6	{a, c, d, e, f}	{1, 1, 3, 5, 1}	24
T_7	{b, c, e}	{5, 2, 4}	15

정의 1. 한 트랜잭션에서 항목의 유틸리티 u(i_p, T_d)는 아래와 같이 정의한다.

$$u(i_p, T_d) = q(i_p, T_d) \times pr(i_p) \quad (1)$$

정의 2. 한 트랜잭션에서 항목집합의 유틸리티 u(X, T_d)는 아래와 같이 정의한다.

$$u(X, T_d) = \sum_{i_p \in X \wedge X \subseteq T_d} u(i_p, T_d) \quad (2)$$

예를 들어, u({ad}, T_1) = u({a}, T_1) + u({d}, T_1) = (4 + 6) = 10.

정의 3. 항목집합의 유틸리티 u(X)는 아래와 같이 정의한다.

$$u(X) = \sum_{X \subseteq T_d \wedge T_d \in D} u(X, T_d) \quad (3)$$

예를 들어, u({ad})는 Equation (2)의 합으로 나타낼 수 있으므로, u({ad}) = u({ad}, T_1) + u({ad}, T_3) + u({ad}, T_4) + u({ad}, T_6) = (1×4 + 2×3) + (2×4 + 1×3) + (2×4 + 1×3) + (1×4 + 3×3) = 10 + 11 + 11 + 13 = 45. 이와 마찬가지로 u({a}), u({abd}), u({abcd})을 계산하면 각 항목집합의 유틸리티가 각각 24, 42, 32임을 계산할 수 있다.

이 결과와 같이 유틸리티 항목집합 마이닝에서는 항목집합의 길이가 늘어날 때 유틸리티값이 증가하기도 하고 감소하기도 하여 downward closure 성질[1]을 적용한 가지치기 전략을 사용할 수 없다[5].

정의 4. 높은 유틸리티 항목집합을 찾는 문제는 사용자가 지정한 최소 유틸리티 minutil보다 크거나 같은 유틸리티값을 갖는 항목집합을 찾는 것이다. 예를 들어, minutil이 40일 때 항목집합 {ad}의 유틸리티는 Equation (3)을 따라 45이므로 이는 높은 유틸리티 항목집합이라고 할 수 있다.

정의 5. 한 트랜잭션의 트랜잭션 유틸리티 TU(T_d)는 아래와 같이 정의한다.

$$TU(T_d) = \sum_{i_p \in T_d} u(i_p, T_d) \quad (4)$$

예를 들어, TU(T_1) = u({a}, T_1) + u({b}, T_1) + u({c}, T_1) + u({d}, T_1) = 17.

정의 6. 한 항목집합의 트랜잭션 가중치 유틸리티(Transaction Weighted Utility, TWU) TWU(X)는 아래와 같이 정의한다.

$$TWU(X) = \sum_{X \subseteq T_d \wedge T_d \in D} TU(T_d) \quad (5)$$

따라서 TWU(X)는 항목집합 X가 포함된 모든 트랜잭션의 유틸리티로 해당 항목이 가질 수 있는 유틸리티의 최댓값이다. 그러므로 X의 실제 유틸리티는 항상 TWU(X)와 같거나 작다[6].

예를 들면, TWU({a}) = TU(T_1) + TU(T_3) + TU(T_4) + TU(T_6) = 17 + 15 + 18 + 24 = 74. 예시 데이터에 대한 다른 항목들의 TWU값은 Table 3에 계산되어있다.

성질 1. TWDC(Transaction-Weighted Downward Closure)[6] 성질은 TWU값에 항목집합이 하나 늘어날 때마다 해당 항목집합이 포함된 트랜잭션 수가 항상 항목을 추가하기 전보다 작거나 같다는 단조 감소 성질이 있음을 의미한다. 따라서 {ad}, {abd}, {abcd}의 TWU값은 74, 50, 32인데, TWU({a}) = 74인 것과 같이, {a}를 포함하는 모든 항목집합의 TWU값은 TWU({a})값보다 작거나 같다.

정의 7. 유틸리티-리스트(Utility-list) 구조[13]

유틸리티-리스트는 각 항목집합의 유틸리티 정보를 저장하기 위한 구조로 본 논문에서 제안하는 UTU-List 구조의 기본 형태이다. 유틸리티-리스트 구조는 세 개의 속성인 tid, iutil, rutil로 구성된다. 각 속성에는 항목집합 X를 포함하는 트랜잭션의 아이디, 한 트랜잭션에서의 항목집합의 유틸리티인 u(X, T_d), 그리고 한 트랜잭션에서 해당 항목집합 이후의 나머지 항목들 T/X의 유틸리티값으로 $\sum_{i \in (T_d/X)} u(i, T_d)$를 계산하여 각각의 속성에 저장한다.

정의 8. EUCS(Estimated Utility Co-Occurrence Structure) 구조[14]

EUCS는 유망하지 않은 항목집합의 가지치기를 위한 삼각 매트릭스 구조다. 이 구조에는 TWU가 minutil보다 작지 않은 항목들로 구성된 2-항목집합의 TWU값을 저장한다. EUCS의 데이터 (a, b, c)는 TWU({a, b}) = c를 의미하므로 Table 4의 TWU({b, c}) = 73은 (b, c, 73)으로 표현한다.

Table 3. 1-itemset Transaction-Weighted-Utility(TWU)

Item	a	b	c	d	e	f
TWU	74	91	112	89	98	24

Table 4. Estimated Utility Co-Occurrence Structure(EUCS)

Item	a	b	c	d
b	50			
c	51	73		
d	69	50	66	
e	37	59	75	52

3. 관련 연구

높은 유틸리티 항목집합 마이닝의 기초가 된 대표적인 연관 규칙 탐사 알고리즘은 각 항목의 빈도를 이용해 패턴을 마이닝했다. Apriori[1] 알고리즘은 빈발 항목집합을 찾기 위해 항목집합의 길이를 하나씩 증가시키는 'level-wise' 기법을 사용하였고 FP-Growth[2] 알고리즘은 재귀적인 트리 구축 방법으로 후보 생성 없이 빠르게 빈발 항목집합들을 얻음으로써 성능을 개선하였다. 이 두 알고리즘은 후보 항목집합의 수를 줄이기 위해 항목집합 지지도의 Downward Closure 성질을 이용한다.

이와 달리 높은 유틸리티 항목집합 마이닝에서는 항목집합의 확장에 따른 유틸리티값이 단조 증가나 단조 감소 성질을 갖지 않아서 Downward Closure 성질을 기반으로 하는 가지치기 전략을 적용할 수 없다[5]. Two-Phase[7] 알고리즘은 성질 1에서 설명한 TWDC 성질로 HUI의 한계를 극복한 알고리즘이다. 이 알고리즘은 TWU의 단조 감소 성질을 이용하여 Apriori의 'level-wise' 방식으로 후보 패턴을 찾은 후에 데이터베이스 재 스캔으로 후보의 실제 유틸리티를 계산하여 결과 패턴을 찾는다. IHUPTWU[9] 알고리즘은 Two-Phase의 두 단계 방식을 FP-Growth 알고리즘에 적용하여 재귀적인 트리 구축으로 빠르게 유용한 패턴을 찾는다. 이와 유사한 UP-Growth$^+$[12] 알고리즘은 많은 후보 생성을 억제하기 위해 유망하지 않은 항목집합을 가지치기 하는 전략들을 추가시켜 성능을 개선하였다. 그러나 이 알고리즘들은 실제 유틸리티값 대신에 TWU값이나 단조 감소 성질을 갖는 과대평가된(overestimated) 값을 사용하기 때문에, 데이터베이스 재 스캔을 통해 많은 후보 항목집합들의 실제 유틸리티값을 계산하는 데 큰 비용이 든다.

이를 해결하기 위해 제안된 HUI-Miner[13] 알고리즘은 정의 7의 유틸리티-리스트를 이용한다. 이 알고리즘은 항목집합을 반복적으로 조인하여 패턴을 확장시키며 실제 유틸리티 정보로 HUI를 탐사하기 때문에 많은 수의 후보를 생성하지 않으며 마지막에 데이터베이스를 다시 스캔하지 않아 실행 시간을 줄였다. FHM[14] 알고리즘은 HUI-Miner 알고리즘을 기반으로 정의 8에서 설명한 EUCS 구조를 이용해 조인 연산의 횟수를 줄여 성능을 향상시켰다.

4. TUL-Miner 알고리즘

본 장에서는 TUL-Miner 알고리즘이 사용하는 자료구조와 이 알고리즘의 효율적인 높은 유틸리티 항목집합 탐사 방법에 대해 설명한다.

4.1 UTU-List(Utility and Transaction-Utility List) 자료구조

제안된 알고리즘 TUL-Miner의 모든 항목집합은 Fig. 1의 UTU-List 구조를 갖는다. 이는 정의 7의 유틸리티-리스트[13]에 itu와 cutil 속성을 추가 시킨 형태이다. itu(itemset transaction-utility) 속성에는 Equation (4)로 계산한 항목집합의 트랜잭션 유틸리티인 TU값을 저장한다.

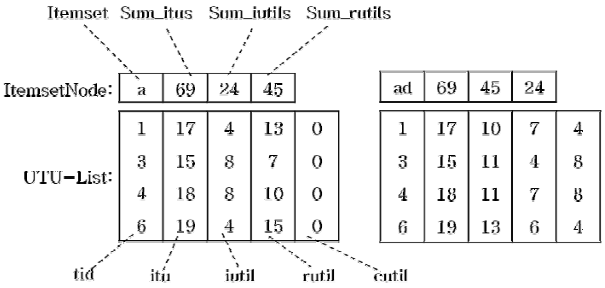


Fig. 1. Itemset Node and UTU-Lists of {a} and {ad}

정의 9. cutil(Common-utility)

이 속성은 조인 연산의 대상 유틸리티-리스트의 공통 유틸리티값이며 조인 연산 수행 시 iutil값을 구할 때 필요한 이 값에 빠르게 접근하기 위해 저장한다. 1-항목집합의 cutil은 모두 0을 갖고, (k+1)-항목집합은 마지막으로 확장되기 이전의 항목집합인 k-항목집합의 iutil값을 갖는다. 예를 들면, Fig. 1과 같이 {a}의 cutil은 모두 0이고 {ad}의 cutil은 {a}의 T_1, T_3, T_4, T_6의 iutil값인 4, 8, 8, 4가 복사된다.

4.2 항목집합 노드 구조

TUL-Miner의 항목집합 노드는 Fig. 1에서 보이는 것과 같이 항목집합인 Itemset과 UTU-List의 cutil을 제외한 각 열의 값들의 합으로 Sum_itus, Sum_iutils, Sum_rutils값을 갖는다. 이 값들은 항목집합이 유망한지를 판단하기 위한 정보로 사용된다. Sum_itus는 항목집합의 TWU값과 같으므로[7] TWDC 성질을 활용하면 정확하고 효과적인 가지치기를 할 수 있다.

4.3 TUL-Miner 알고리즘

본 논문에서 제안하는 TUL-Miner 알고리즘은 기존 알고리즘들[13, 14]과 비교하여 UTU-List의 사용과 TUL-Search에서 차이가 있다. TUL-Miner 알고리즘을 단계적으로 서술하기 위해 세 개의 소절들로 나누어서 설명한다. 첫 소절에서는 사전 작업으로 초기 UTU-List와 EUCS의 구축을 설명하고 두 번째 소절에서는 TUL-Search 알고리즘이 유망하지 않은 항목집합을 가지치기하는 전략을 소개한다. 마지막 소절에서는 빠른 조인 연산과 생성 항목집합이 유망하지 않으면 연산을 끝내는 내부 함수인 Utility- List Join 함수를 설명한다.

1) 초기 UTU-List와 EUCS 구축

높은 유틸리티 항목집합 마이닝의 사전 작업으로 초기 UTU-List와 EUCS를 구축한다. 이 구조들을 효율적으로 구축하기 위해 먼저 데이터베이스를 한 번 스캔하여 각 항목의 TWU값을 계산한 후에 TWU가 minutil보다 큰 항목들에 대해 TWU 비내림차순으로 항목들을 정렬한다. 이는 알고리즘의 특성상 항목집합의 나열에서 앞에 위치한 항목일수록 많은 항목집합들을 생성하므로 TWU가 작은 순서대로 정렬하면 그 항목들이 유망하지 않을 가능성이 높아져 효율적인 가지치기가 이루어지기 때문에 실행 속도가 빨라진다. 예시 데이터베이스로 이를 수행하면 1-항목집합은 a < d < b < e < c 순서로 정렬된다.

1-항목집합을 정렬한 후에는 데이터베이스를 다시 스캔하며 각 트랜잭션의 항목들을 같은 순서로 재배열하여 초기

Algorithm: TUL-Miner Algorithm
input : D, a transaction database; minutil.
output: the set of high-utility itemsets.
1 Scan D to calculate the TWU of 1-itemsets
2 I^* ← each item i such that TWU(i) ≥ minutil
3 Sort items in TWU ascending values on I^*
4 Scan D to build the UTU-List of each item i ∈ I^* and build the EUCS structure
5 TUL-Search(∅, I^*, minutil, EUCS)

Fig. 2. Algorithm TUL-Miner

UTU-List와 EUCS를 구축한다. 이때, TWU가 minutil보다 작은 항목이 있으면 DGN(Discarding Global Node utility) [11]전략을 따라 그 항목과 유틸리티를 트랜잭션에서 제외한다. 초기 UTU-List의 cutil값은 정의 9에 따라 모두 0으로 설정한다.

2) TUL-Search 알고리즘

사전 작업을 마치면 높은 유틸리티 항목집합을 탐색하기 위해 TUL-Search 알고리즘을 호출한다. TUL-Search는 초기 UTU-List로 항목집합들을 Fig. 3과 같이[13] 확장시켜 HUI를 찾아내는 재귀 함수이다.

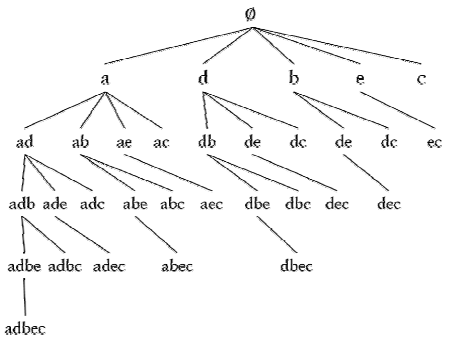


Fig. 3. Set-Enumeration Tree

Fig. 4의 TUL-Search 알고리즘은 항목집합 P에 x를 확장한 Px에 대해 높은 유틸리티 항목집합인지와 항목집합을 확장하기에 유망한지를 판단한다. Px의 유틸리티값을 검사해 minutil보다 크거나 같으면 높은 유틸리티 항목집합으로 출력하고 Px의 유틸리티값과 앞으로 증가될 수 있는 유틸리티값인 나머지 유틸리티값의 합을 검사해 minutil보다 작지 않으면 패턴 확장이 유망한 것이므로 그에 따른 연산을 수행한다. 이 연산은 항목집합 Px에 TWU값이 x보다 큰 y를 항목집합 P에 확장시킨 Py를 조인하여 확장된 항목집합 Pxy를 생성하고 가능한 모든 y에 대해 항목집합을 확장시킨 후 높은 유틸리티 항목집합들을 탐사하도록 재귀 호출한다.

THC(Tid-based Hash Check) 전략은 EUCS를 이용한 전략에서 유망하다고 판단된 항목집합들 중 HUI가 될 가능성이 없는 항목집합이 비용이 큰 Utility-List Join 연산을 호출하지 않도록 필터 역할을 한다. 이 전략은 Px와 Py의 tid들이 얼마나 동일한지를 파악하기 위해 해시 비트벡터 TrnxHashCheckList를 이용하여 Pxy의 Sum_itus의 추정치를 계산해 Pxy가 유망한지를 판단한다. 이 리스트의 길이는 1-항목집합의 UTU-List 중 최대 트랜잭션 수에 따라 충돌을 최소화할 수 있도록 결정된다. 리스트 비트 백터를 0으로 초기화시키고 setTHCL 함수에서 Px에 속한 tid들의 해시 주소 비트를 1로 설정한다. EUCS의 TWU({x, y})값이 minutil 이상이면 checkTHCL 함수에서 Py의 tid들에 같은 해시 함수를 적용해 각 tid의 해시 주소 비트값을 확인하고 비트가 0일 때마다 Py의 Sum_itus에서 해당 트랜잭션의 itu 값을 제외시킨다. Py의 UTU-List 안의 모든 tid들에 대해 검사가 끝날 때까지 그 값이 minutil값 이상이면 true를 반환하고 minutil보다 작아지면 false를 반환한다.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
0	1	0	1	1	0	0	0	0	0	0

Fig. 5. TrnxHashCheckList set by itemset {adb}

이 전략을 주어진 예제 데이터로 P가 {ad}일 때 각각 {b}와 {c}를 확장시킨 {adb}와 {adc}로 설명한다. 리스트의 길이는 초기 UTU-List의 최대 길이인 6보다 크면서 충돌이 적게 발생하도록 11로 정하고 setTHCL 함수에서 항목집합 {adb}의 tid인 {1, 3, 4}에 나누기 방법을 적용하여 Fig. 5와 같이 THCL의 길이 값 11로 나눈 나머지 값에 해당하는 인덱스의 비트를 1로 설정한다. EUCS의 TWU({b, c}) = 73으

Algorithm: TUL-Search Algorithm
input : P, an itemset; ExtensionsOfP, a set of extensions of P; minutil; EUCS.
output: the set of high-utility itemsets.

```
1   for i ← 0 to length(ExtensionsOfP)-1 do
2       Px ← ExtensionsOfP[i]
3       if Px_Sum_iutils ≥ minutil then    ▷ by the definition of high utility itemset
4           output Px
5       if (Px_Sum_iutils + Px.Sum_rutils) ≥ minutil then
6           ExtensionsOfPx ← ∅
7           setTHCL(Px)                    ▷ set TrnxHashCheckList according to the UTU-list of Px
8           for j ← i+1 to length(ExtensionsOfP)-1 do
9               Py ← ExtensionsOfP[j]      ▷ y after x such that TWU(x) < TWU(y)
10              if c ≥ minutil such that (x, y, c) ∈ EUCS and checkTHCL(Py) then
11                  Pxy ← Utility-List Join (Px, Py)
12                  if Pxy.Sum_itus ≥ minutil then    ▷ proceed when TWU(Pxy) is promising
13                      ExtensionsOfPx ← ExtensionsOfPx ∪ Pxy
14          TUL-Search(Px, ExtensionsOfPx, minutil)
```

Fig. 4. Algorithm TUL-Search

로 minutil 값인 40 이상이므로 함수 checkTHCL을 실행하여 {ac}의 UTU-List에 있는 tid인 {1, 3, 6}에 같은 방법으로 해당 인덱스 비트를 찾아 확인한다. 인덱스 1과 3은 비트가 1이지만 인덱스 6은 비트가 0이므로 {adc}의 Sum_itus 값 51에서 T_4의 itu값인 19를 빼면, 51-19 = 32가 되므로 minutil값 40보다 작아 false를 반환하고 항목집합 {adbc}를 생성하기 위한 조인 연산은 수행하지 않는다.

3) Utility-List Join 함수

THC 전략에서 필터 되지 않은 경우 Px와 Py는 Fig. 6의 Utility-List Join 함수로 항목집합 Pxy를 생성한다. 이 조인 연산을 빠르게 수행하기 위해 NSC 전략과 DTJ 전략을 추가하였다.

NSC(No Search by using Cutil) 전략은 새로운 항목집합의 유틸리티를 계산하기 위한 공통 유틸리티값의 탐색 시간을 절약하는 전략이다. Px와 Py를 조인한 항목집합 Pxy의 UTU-List에 tid가 추가될 때마다 iutil값은 다음과 같이 구할 수 있다.

$$Pxy.iutil = Px.iutil + Py.iutil - P.iutil \quad (6)$$

유틸리티-리스트 기반 알고리즘들[13, 14]에서는 공통 유틸리티인 P.iutil을 구하기 위해 Px와 Py의 공통 항목집합인 P를 탐색하고, P의 UTU-List에서 해당 tid를 탐색해야 한다. NSC전략은 이 시간을 단축시키기 위해 Px의 UTU-List에 저장한 cutil(Common-Utility)을 이용하여 이 값에 한 번에 접근하여 공통유틸리티의 존재여부 판단과 공통 항목집합에 접근하는 시간, 그리고 해당 tid를 탐색하는 비용을 절약한다. 따라서 Pxy의 iutil값은 아래의 식으로 구한다.

$$Pxy.iutil = Px.iutil + Py.iutil - Px.cutil \quad (7)$$

예를 들어, Fig. 7의 {ad}와 {ab}를 조인하는 경우 공통 트랜잭션 T_1에 대해 {adb}의 UTU-List는 iutil값 10+8-4 = 14를 갖는다. 그리고 T_1의 cutil은 {ad}의 T_1의 iutil값인 10이 된다.

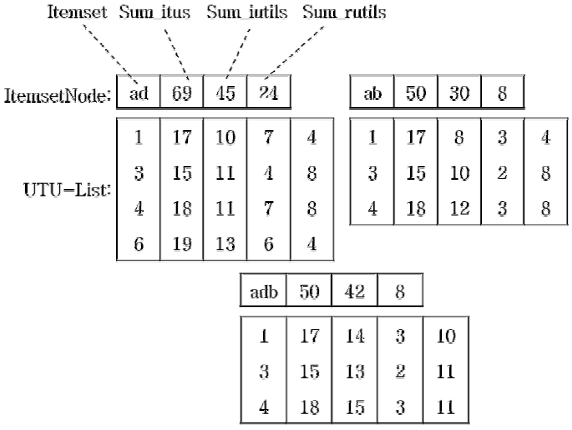


Fig. 7. UTU-Lists of {ad}, {ab} and {adb}

Function: Utility-List Join Function
input: Px, the extension of P with an item x; Py, the extension of P with an item y.
output: Pxy.

```
1   Pxy ← Px ∪ Py
2   rx ← Px.Sum_itus                ▷ TWU(Px)
3   ry ← Py.Sum_itus                ▷ TWU(Py)
4   i ← j ← 0
5   while i < Px.num and j < Py.num do
6       ex ← Px.utulist[i]
7       ey ← Py.utulist[j]
8       if ex.tid == ey.tid then
9           exy ← (ex.tid, ex.itu, ex.iutil + ey.iutil - ex.cutil, ey.rutil, ex.cutil)
10          Pxy.UTUList ← Pxy.UTUList ∪ {exy}
11          i++
12          j++
13      else if ex.tid < ey.tid then
14          rx -= ex.itu                ▷ minus the TU of the transaction not belong to Pxy
15          if rx < minutil then break  ▷ Pxy is definitely not a HUI
16          i++
17      else
18          ry -= ey.itu
19          if ry < minutil then break
20          j++
21  return Pxy
```

Fig. 6. Function Utility-List Join

Table 5. Characteristics of Datasets

Database	Size(KB)	#Trans	#Items	AvgLen	MaxLen	Type
Accidents	59663	340183	468	33.8	51	Dense
Kosarak	49859	990002	41270	8.1	2498	Sparse
T10I4D100K	6268	98424	1000	10.1	30	-
T40I10D100K	24905	100000	1000	39.6	78	-

DTJ(Decreasing TWU while Joining) 전략은 조인 연산으로 확장된 항목집합 Pxy의 TWU가 minutil보다 작아져 앞으로 HUI가 될 가능성이 없게 되면 조인 연산을 중도에 중지하는 전략이다. 이 전략은 Utility-List Join 함수의 line 2 - 3과 line 13 - 20에서 이루어진다. Px와 Py에 대해 조인 연산을 하는 도중에 각 항목집합의 TWU의 예상치가 minutil보다 작으면 연산을 중지하도록 하여 조인 결과 항목집합인 Pxy로 확장하지 않는다.

5. 실험 결과 및 분석

본 논문에서 제안하는 알고리즘 TUL-Miner가 높은 유틸리티 항목집합들을 효율적으로 탐사하는 것을 보여주기 위하여 유틸리티-리스트 기반의 최근 알고리즘인 HUI-Miner [13]와 FHM[14]의 성능을 비교 평가하였다. 참고문헌[13]은 유틸리티-기반 알고리즘이 재귀적으로 조건 트리를 구축하는 알고리즘들[5-12]과 비교하여 실행속도와 사용 메모리 관점에서 더 우수함을 보여주었고 본 연구에서 실행한 여러 실험으로도 비슷한 성능 평가를 얻을 수 있어서 본 논문에서는 유틸리티-기반 알고리즘들의 성능 평가만을 여러 관점에서 보여주고 설명하였다. 이 실험은 최소 유틸리티값을 변화시켰을 때 각 알고리즘의 실행 시간, 사용 메모리의 크기, 그리고 생성된 후보 항목집합들의 수를 측정하였다.

성능 평가에 사용된 실험 데이터는 IBM Quest Data Generator[1]로 생성한 데이터인 T10I4D100K와 T40I10D100K이고, 그리고 FIM Repository[15]에서 제공하는 실 환경 데이터인 Accidents와 Kosarak을 사용하였다[9-14]. 생성 데이터의 항목들의 수량은 1부터 10까지의 랜덤 수를 생성하였고, 항목가격은 1에서 1000까지의 단위 가격으로 로그 정규 분포를 따르도록 하였다[7-13]. Table 2를 보면 실 환경 데이터는 트랜잭션의 길이가 서로 유사해 평균길이 부근에 조밀(dense)한 분포를 가지거나 혹은 트랜잭션의 길이가 서로 상당히 달라 상당히 퍼져있는 희소(sparse)한 분포를 가진다. 그러나 생성 데이터 T10I4D100K와 T40I10D100K는 랜덤으로 생성된 데이터이므로 데이터의 형태가 상대적으로 고른 분포를 가진다.

본 실험은 MS Windows 7 64bit 운영체제의 Intel i7-4770 CPU 3.40GHz와 32GB 메모리의 환경에서 실행하였고, 모든 프로그램은 Microsoft Visual Studio 2010에서 C++로 구현하여 그 실험 결과를 측정하였다.

5.1 실행 시간 비교

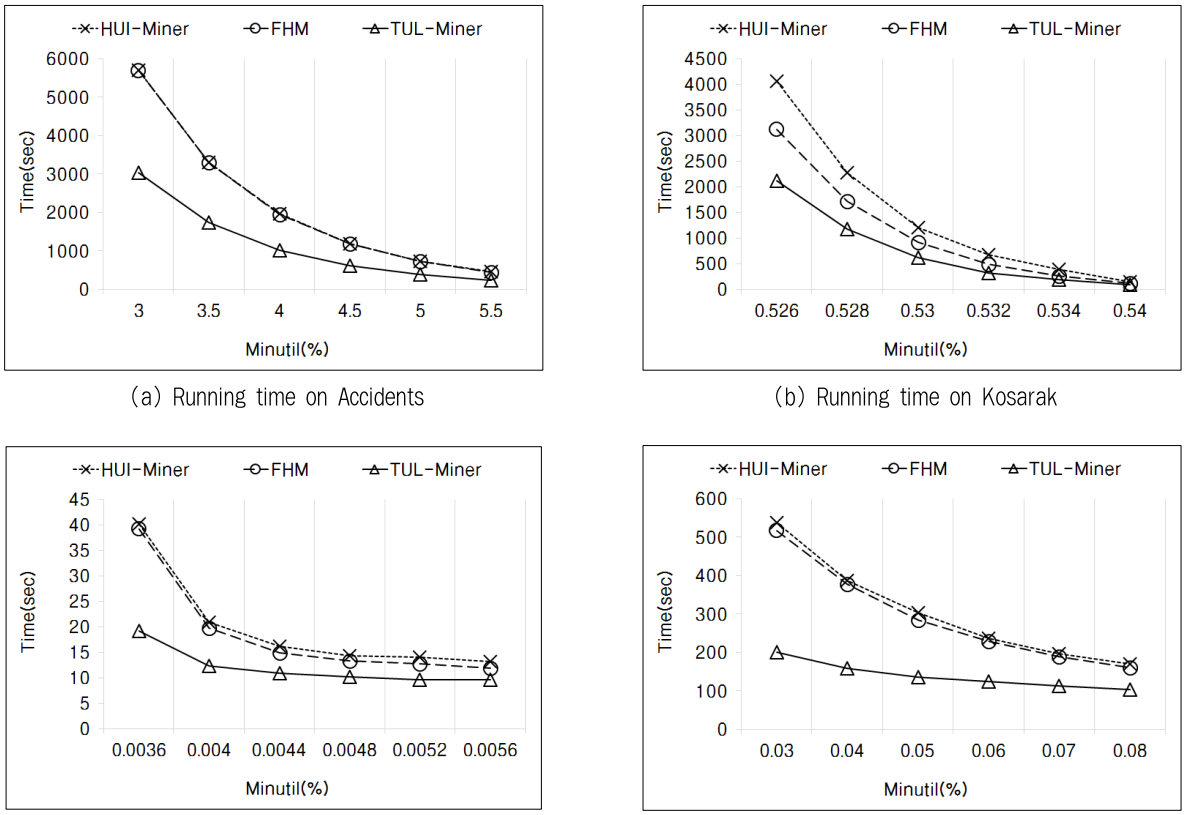
이 절에서는 각 알고리즘의 실 환경 데이터와 생성 데이터의 실행 시간을 비교한 실험 결과를 보인다. 먼저 조밀한 실 환경 데이터 Accidents를 대상으로 한 실험 결과 Fig. 8(a)에서는 HUI-Miner와 FHM 알고리즘의 성능이 거의 동일했고 TUL-Miner는 이 둘과 비교하여 Accidents 데이터에 대해 40-50% 정도로 실행 시간을 개선시켰다. Fig. 8(b)의 희소한 데이터 Kosarak에 대한 결과도 TUL-Miner의 속도 성능이 가장 우수했고 HUI-Miner와 비교하면 약 40~50%, FHM과 비교하면 약 25~35%의 성능 향상을 보였다.

생성 데이터 T10I4D100K에 대한 실험 결과 Fig. 8(c)는 HUI-Miner와 비교해 약 30~50% 정도로 실행 시간을 개선시켰고, FHM에 대해 20~50%로 성능을 개선시켰다. Fig. 8(d)의 데이터 T40I10D100K에 대한 실험에서는 FHM 알고리즘과 HUI-Miner의 실행속도가 거의 비슷했고 TUL-Miner는 이 두 알고리즘과 비교해 약 40~60%로 상대적 속도 향상이 나타났다. 제안하는 알고리즘은 트랜잭션의 평균 길이가 상대적으로 긴 데이터인 Accidents와 T40I10D100K에 대해 실행 속도 면에서 더 큰 성능 개선을 보여주었다.

5.2 후보 항목집합들의 개수 비교

이 절에서는 실 환경 데이터와 생성 데이터에 대해 minutil값에 따른 HUI-Miner, FHM, TUL-Miner 알고리즘의 후보 항목집합의 수와 실제 결과 항목집합인 HUI의 수를 비교한 실험 결과를 설명한다. 기존의 '후보 항목집합'의 의미는 패턴성장 방식의 HUI 마이닝 알고리즘[5-12]에서 과대평가된 유틸리티로 찾은 잠재적인 패턴들을 지칭하는 용어이므로 실제 유틸리티를 사용하여 결과를 찾는 유틸리티-리스트 기반 알고리즘은 '후보 항목집합'을 생성하지 않는다. 그러나 마이닝 중에 생성된 모든 항목집합들이 곧 유망한 항목집합이므로 이를 후보 항목집합 PHUI(Potential High Utility Itemsets)이라 부르고 그 개수 |PHUI|를 비교하여 가지치기 성능을 평가하고자 한다.

생성 데이터 T10I4D100K에 대한 실험 결과 그래프인 Fig. 9에서 minutil값이 0.015%일 때 실제 HUI는 64,609개인데, HUI-Miner가 생성하는 PHUI의 수는 14,202,178개, FHM이 8,230,643개, 그리고 TUL-Miner가 1,571,114개를 생성한다. Fig. 9에서 PHUI의 수가 가장 적은 TUL-Miner는 실제 HUI 개수의 약 4~6배이고, HUI-Miner의 |PHUI|가



(a) Running time on Accidents

(b) Running time on Kosarak

(c) Running time on T10I4D100K

(d) Running time on T40I10D100K

Fig. 8. Running Time

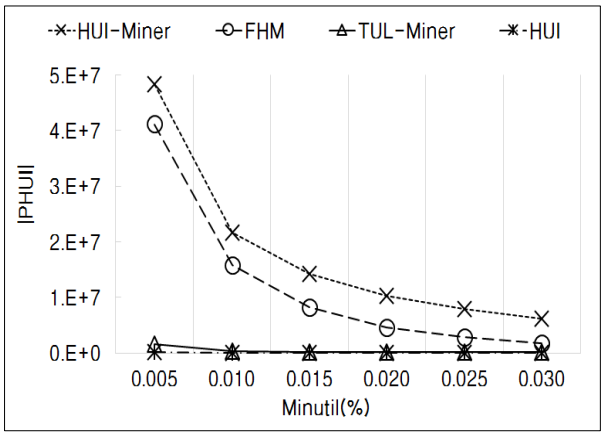


Fig. 9. Number of PHUI on T10I4D100K

TUL-Miner에 비해서 약 30~40배이며 FHM의 |PHUI|가 TUL-Miner에 비해서 약 10~30배이다. 이로서 TUL-Miner는 기존 알고리즘들과 비교해 항목집합의 가지치기가 확실히 효과적임을 알 수 있다.

5.3 사용 메모리 비교

이 절에서는 생성 데이터와 실 환경 데이터에 대해 각 알고리즘의 모든 실행 과정 동안의 최대 메모리(Peak memory) 값을 비교한 실험을 설명한다. T10I4D100K에 대해 minutil을 0.005%로 실험한 결과 TUL-Miner는 28.3MB이고 FHM은 23MB로 제안하는 알고리즘이 더 많은 메모리를 사용한다. 이는 TUL-Miner의 기반 구조인 UTU-List가 HUI-Miner나 FHM의 Utility-List 구조에 비해 두 필드 더 많이 갖기 때문에 한 항목집합마다 기존의 유틸리티-리스트 구조의 5/3 크기, 즉 66% 가량의 메모리 공간을 더 필요로 하게 되기 때문이다. 그러나 실험 결과 TUL-Miner의 상대적 메모리 사용량의 증가는 FHM의 약 23%로 상대적인 TUL-Miner의 메모리 효율이 리스트 구조의 증가 비율인 66%보다 작다.

5.4 실험 결과 분석

최근 알고리즘들과의 비교실험 결과에서 제안된 알고리즘 TUL-Miner의 성능이 전반적으로 우수함을 보였다. 먼저 시간 성능 측면에서, 희소한 데이터의 경우 EUCS의 2-항목집합의 TWU값을 검사하는 것만으로도 많은 가지치기가 수행되어 FHM 대비 속도 개선의 정도가 조밀한 데이터를 대상으로 한 실험보다 적었다. 조밀한 데이터의 경우와 트랜잭션의 평균 길이가 긴 데이터에 대한 실험에서는 EUCS의

가지치기 성능이 크지 않았으나 TUL-Miner의 모든 길이의 항목집합 TWU값을 이용한 가지치기 전략과 계산 시간단축 전략은 성능을 향상시키는 데 큰 역할을 하였다.

생성된 후보 항목집합들의 개수 비교 실험은 TUL-Miner의 후보 항목집합의 수가 큰 차이로 적어 제안하는 알고리즘의 가지치기 성능이 가장 우수함을 증명하였다. TUL- Miner 알고리즘의 가지치기 전략들이 효과적일 수 있었던 이유는 항목집합 트랜잭션 유틸리티값은 항목집합 길이가 길어질수록 그 값이 크게 감소하는 특성을 가지기 때문에 항목집합의 길이가 길어질수록 그 효과가 더욱 커지기 때문이다.

최대 사용 메모리에 대한 실험에서 TUL-Miner의 절대적 메모리 양은 최근 알고리즘 HUI-Miner와 FHM보다 더 많다는 결과를 보였으나 이는 TUL-Miner의 효과적인 항목집합의 가지치기로 TUL-Miner의 메모리 사용이 상대적인 증가 비율이 작아짐을 보여주었다.

6. 결 론

본 논문은 높은 유틸리티 항목집합을 찾기 위해 효과적인 알고리즘 TUL-Miner를 제안하였다. TUL-Miner는 많은 후보 생성과 계산을 방지하는 유틸리티-리스트 구조에 항목집합의 트랜잭션 유틸리티(TU)와 공통 유틸리티를 추가한 UTU-List구조와 해시 비트벡터 구조를 사용하여 유망하지 않은 경우 조인 연산을 하지 않음으로써 유틸리티-리스트 기반 알고리즘이 갖는 비용을 많이 줄였으며 유틸리티 계산의 단순화로 연산의 속도를 증가시켰다. 실험을 통해 TUL-Miner 알고리즘이 기존 유틸리티-리스트 기반 알고리즘들[13, 14]과 비교하여 속도 성능이 개선되었음을 보였고 특히 데이터가 조밀하거나 트랜잭션 평균 길이가 길수록 더 큰 성능 향상을 보였다. 또한 제안 알고리즘은 HUI를 탐사하는 동안 생성하는 항목집합들의 수를 크게 감소시켰고 메모리 사용에 있어서도 구조의 크기 증가율과 비교하면 사용 메모리의 증가율이 더 작아 제안 알고리즘이 효율적인 마이닝을 수행함을 보였다.

References

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Data Bases*, Santiago, Vol.1215, pp.487-499, 1994.

[2] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, Dallas, pp.1-12, 2000.

[3] W. Wang, J. Yang, and P. Yu, "Efficient mining of weighted association rules (WAR)," in *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Boston, pp.270-274, 2000.

[4] F. Tao, F. Murtagh, and M. Farid, "Weighted association rule mining using weighted support and significance framework," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, pp.661-666, 2003.

[5] H. Yao, H. J. Hamilton, and C. J. Butz, "A foundational approach to mining itemset utilities from databases," in *Proceedings of the Fourth SIAM International Conference on Data Mining. SIAM*, pp.482-486, 2004.

[6] Y. Liu, W. Liao, and A. Choudhary, "A fast high utility itemset mining algorithm," in *Proceedings of the 1st International Workshop on Utility-Based Data Mining, ACM*, Chicago, pp.90-99, 2005.

[7] Y. Liu, W. Liao, and A. Choudhary, "A two-phase algorithm for fast discovery of high utility itemsets," in *Proceedings of the 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer*, Vol.3518, pp.689-695, 2005.

[8] H. Yao and H. J. Hamilton, "Mining itemset utilities from transaction databases," *Data & Knowledge Engineering*, Elsevier, Vol.59, No.3, pp.603-626, 2006.

[9] 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*, Vol.21, No.12, pp.1708-1721, 2009.

[10] B.-S. Jeong, C. F. Ahmed, I. Lee, and H. Yong, "High utility pattern mining using a prefix-tree," *Journal of KIISE: Database*, Vol.36, No.5, pp.341-351, 2009. (in Korean)

[11] V. S. Tseng, C.-W. Wu, B.-E. Shie, and P. S. Yu, "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*, Washington, pp.253-262, 2010.

[12] V. S. Tseng, B.-E. Shie, C.-W. Wu, and P. S. Yu, "Efficient algorithms for mining high utility itemsets from transactional databases," *IEEE Transactions on Knowledge and Data Engineering*, Vol.25, No.8, pp.1772-1786, 2013.

[13] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, Maui, pp.55-64, 2012.

[14] P. Fournier-Viger, C.-W. Wu, S. Zida, and V. S. Tseng, "FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning," *Foundations of Intelligent Systems*, Springer, pp.83-92, 2014.

[15] Frequent Itemset Mining Dataset Repository. Available at [Internet] http://fimi.cs.helsinki.fi/data/.

이 세 린

e-mail : serin9@sungshin.ac.kr
2014년 성신여자대학교 IT학부(학사)
2014년～현 재 성신여자대학교 컴퓨터학과 석사과정
관심분야 : Data mining, Database

박 종 수

e-mail : jpark@sungshin.ac.kr
1981년 부산대학교 전기기계공학과(학사)
1983년, 1990년 한국과학기술원 전기 및 전자공학과(석사/박사)
1983년～1986년 국방부 군무설계기좌
1994년～1995년 IBM Watson 연구소 객원연구원
1990년～현 재 성신여자대학교 IT학부 교수
관심분야 : Data mining, Database, Transportation geography