점진적인 웹 마이닝을 위한 효율적인 후보패턴 저장 트리구조 및 알고리즘

An Efficient Candidate Pattern Storage Tree Structure and Algorithm for Incremental Web Mining

강희성* · 박병준**
Hee Seong Kang · Byung Jun Park

Abstract - Recent advances in the internet infrastructure have resulted in a large number of huge Web sites and portals worldwide. These Web sites are being visited by various types of users in many different ways.

Among all the web page access sequences from different users, some of them occur so frequently that may need an attention from those who are interested. We call them frequent access patterns and access sequences that can be frequent the candidate patterns. Since these candidate patterns play an important role in the incremental Web mining, it is important to efficiently generate, add, delete, and search for them.

This thesis presents a novel tree structure that can efficiently store the candidate patterns and a related set of algorithms for generating the tree structure, adding new patterns, deleting unnecessary patterns, and searching for the needed ones. The proposed tree structure has a kind of the 3 dimensional link structure and its nodes are layered.

Key Words : Tree, Incremental, Candidate, Pattern, Usage Web Mining

1. 서 론

웹 환경은 네트워크 환경의 양적 질적 팽배로 인해 더욱더 다양한 콘텐츠를 서비스 하게 되었고, 다양한 콘텐츠를 다수의 사용자가 이용하게 됨으로 인해 마이닝은 접근 패턴 분석에 더욱더 많은 리소스를 필요로 하게 되었다.

모든 접근 패턴을 분석하는 방법이 가장 이상적인 결과를 나타낼 수 도 있겠지만, 특정 빈도 이상의 패턴만으로 후보 패턴을 구성하고, 마이닝을 하는 것이 전체적인 수행시간을 단축시키고, 동일한 결과를 얻을 수 있기 때문에 현재의 마이닝은 후보 패턴을 이용하는 것과 새로운 후보패턴이 발생했을 경우 새로 발생한 패턴 들 만으로 마이닝 하는 점진적 증가 마이닝(Incremental Mining)[1][4]이 일반적이며, 이를 위해 FS-Tree 알고리즘 [1] FAP-Tree 알고리즘 [6] 등 다양한 트리구조를 이용한 알고리즘들이 기존의 논문에 소개되었다.

로그를 이용해 후보 패턴을 추출하고 이 후보 패턴을 트리구조에 저장하는 것은 검색과 삽입의 반복 작업으로 이루어지게 되는데, 기존의 트리형태는 후보 패턴이 증가할수록 성능이 저하되는 경향이 있다. 기존의 후보 패턴 추출 방법과 동일한 방법을 사용하고, 또한 기존의 마이닝 방법과 동일한 마이닝 방법을 사용 할 경우 추출한 후보 패턴을 생성 및 검색 하는 시간을 단축한다면 전체적인 마이닝 시간을 단축하는 효과를 보게 될 것이다.

본 논문에서는 후보 패턴의 구축 시간을 단축할 수 있는 노드와 계층 트리들로 구성된 3DFS-Tree(3D Frequent Sequence Tree)를 제안하며, 다른 트리들과의 성능 예측 및 비교 실험 결과를 도출할 것이다. 성능 비교 실험은 순수하게 트리의 성능을 테스트하기 위해 후보패턴이 되는 조건에 부합하는 시퀀스들을 가지고 생성, 점진적 증가 등에 대해 실행시간이 얼마나 차이가 나는지에 대해 실험 한다.

저자 소개
* 강희성: 광운대학 컴퓨터과학과 박사과정
** 박병준: 광운대학 컴퓨터과학과 교수 · 공박

2. 관련연구

본 장에서는 본 논문의 이해에 필요한 지지도와 비교 실험 대상인 FS-Tree 알고리즘에 대해서 설명한다.

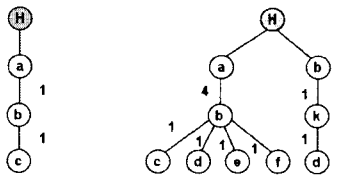
2.1 지지도 (Support Count)

마이닝 시스템은 2개의 인수를 가지고 있는데, 하나는 시스템 인수로서 최소한의 후보 패턴을 추출하기 위한 지지도를 최소 지지도 카운트(Minimum Support Count)라고 부르며 MSuppClink로 표현하고, 또 하나는 사용자 인수로서 사용자가 요구하는 후보 패턴을 추출하기 위한 값으로 최소 시퀀스 지지도 카운트(Minimum Sequence Support Count)라고 하며, MSuppCseq로 표현한다. 이 두 개의 값으로 후보 패턴을 추출해 내는데 Suppseq≥ MSuppCseq이면 빈발 시퀀스 패턴, Supplink(h) ≥ MSuppCseq이면 빈발 링크, Supplink(h) ≥ MSuppClink 이고 Supplink(h) < MSuppCseq이면 잠재적인 빈발 링크, MSuppClink, MSuppCseq 둘 다 만족 못하면 빈발하지 않은 링크라고 한다. 여기서 빈발 시퀀스 링크와 잠재적인 빈발 링크에 포함된 시퀀스를 후보패턴이라 한다[1].

2.2 FS-Tree 알고리즘

FS-Tree 알고리즘은 후보패턴의 조건을 만족하는 시퀀스를 이용해 트리를 구성하는 알고리즘으로 시퀀스내의 아이템 순서대로 하위노드로 연결되는 구조를 가지고 있다. 예를 들어 시퀀스 <a, b, c>가 입력된다면 그림1(a) 처럼 트리가 생성되고, 여기서 추가로 시퀀스 <a, b, d>, <a, b, e>, <a, b, f>, <b, k, d> 등이 발생하면 그림1(b) 같은 트리가 생성 된다. 중복되는 시퀀스 <a, b>는 한번만 표현되고 중복횟수만큼 카운트가 증가 하는 구조를 가지고 있다.

FS-Tree는 표현은 트리 형태로 표현되지만 한 노드에 여러 개의 노드가 연결되어 있어 트리 형태로 보일 뿐이며, 추가, 삭제, 검색 시 자료구조상의 트리가 갖고 있는 잇 점을 전혀 가지고 있지 않다. 본 논문에서 제안하는 3DFS-Tree는 표현, 구조 모두 트리의 장점을 살리고 있으며 이점이 FS-Tree와 차별화 되는 부분이다.



(a) (b)

그림 1 FS-Tree

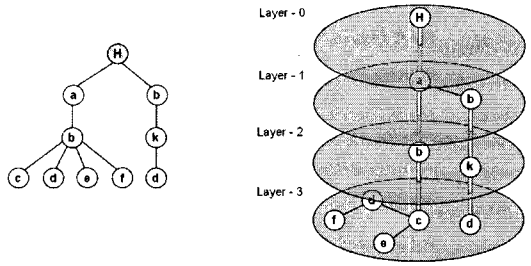
3. 3D Frequent Sequence Tree 알고리즘

3.1 3DFS-Tree 알고리즘의 소개 및 구성

3DFS-Tree는 관련연구에서 설명한 FS-Tree와 같이 입력된 시퀀스들을 저장하는 하는 트리구조를 갖고 있지만, 3DFS-Tree는 시퀀스의 추가, 삭제, 검색에 좀 더 효율적인 트리구조를 가지고 있다.

시퀀스 <a, b, c>, <a, b, d>, <a, b, e>, <a, b, f>, <b, k, d>가 입력됐을 때, FS-Tree는 그림2(a)와 같이 하나의 노드에 여러 개의 노드가 링크되는 형태로 저장이 되지만, 3DFS-Tree의 경우는 그림2(b)와 같이 하나의 노드는 최대 4개의 노드가 링크되는 구조를 가지고 있기 때문에, 노드수가 일정하지 않은 FS-Tree 보다 정형화된 구조를 가지며, 이 구조를 통해 더욱더 빠른 시퀀스의 추가, 삭제, 검색이 가능해지게 된다.

그림2(b)의 3DFS-Tree를 보게 되면 한 줄로 연결된 노드와 두 줄로 연결된 노드가 존재 하는데 이는 계층(Layer)에 따라 결정되어진다. 여기서 계층이란 시퀀스 < p1, p2, ···, pn > 이고 각 아이템은 pi 일때, 1≤ i ≤ n 이면, i가 계층이 된다. 예를 들어 시퀀스 <a, b, c>에서 각 아이템의 계층은 a는 1, b는 2, c는 3의 계층을 갖게 되는 것이다. 그림3(b)와 같이 3DFS-Tree는 각 시퀀스의 아이템들의 계층이 다르면 두 줄로 표시된 계층 링크(Layer Link)로 연결되며, 각 시퀀스의 아이템들의 계층이 같을 경우 한 줄로 표시된 노드 링크(Node Link)로 연결된다.



(a) FS-Tree (b) 3DFS-Tree

그림 2 Fs-Tree와 3DFS-Tree의 비교

3DFS-Tree는 최소 구성 요소인 노드(Node), 이 노드들을 트리로 구성한 계층 트리(Layer Tree)로 구성되어 있으며, 계층트리는 계층 링크 (Layer Link)로 모두 연결되어 있고, 이렇게 계층 링크 로 연결된 계층 트리들로 구성되어 있다.

계층트리는 비트 트리로 구성되어 있으며 이를 통해 균형잡힌 트리 구조를 유지 함으로 인해 높은 성능을 얻게 된다.

3.2 3DFS-Tree 알고리즘

3DFS-Tree 알고리즘은 입력된 시퀀스들의 추가, 삭제, 검색 알고리즘으로, 시퀀스를 추가하는 Insert3DFSTree(), 추가된 노드를 찾는 FindNode(), 계층 트리를 추가하는 Insert LayerTree(), 노드를 추가하는 InsertNode() 등으로 구성된다.

3.2.1 Node를 추가하는 알고리즘

```
# 의사코드 내의 Sequence p-P에서 p는 시퀀스의 첫 번째 아이템을 P는 서브시퀀스를 나타낸다.

Function Insert3DFSTree(Node N, Sequence p-P)
{
    Create Node D
    D = FindLayerTree(N.LayerDownLink, p)
    if(D is Not Null)
        N = D
        N.Count = N.Count + 1
    else
        N = InsertLayerTree(N, p)
    if(P is not Null) Insert3DFSTree(N, P)
}
```

그림 3 Insert3DFSTree()의 의사코드

Insert3DFSTree()는 노드와 시퀀스를 인수로 받고 노드 N이 시퀀스의 아이템 p를 하위 계층 링크의 노드로 가지고 있는지 아니면 N의 하위 계층 링크에 연결되어 있는 계층 트리에 포함되어 있는지를 FindLayer Tree()로 확인한다. 이와 같은 조건을 만족한다면 해당 노드 N의 카운트는 1이 증가하고, 만족하지 않는다면 InsertLayerTree()를 통해 p를 LayerTree에 추가하며, 추가된 노드는 다시 N으로 반환된다. 입력된 시퀀스내 아이템중 하나가 처리되었으므로, 서브시퀀

스 P와 노드 N을 가지고 다시 Insert3DFSTree()를 호출하며, 시퀀스의 길이만큼 Insert3DFSTree()를 재귀호출하게 된다.

3.2.2 계층 트리를 구성하는 비트 트리 알고리즘

계층 트리를 구성하는 트리 구조인 비트트리는 이진트리 형태를 띄고 있으나, 일반적인 이진트리에서 나타나는 좌우 노드의 불균형으로 인한 성능 저하를 없앤 트리 형태이다.

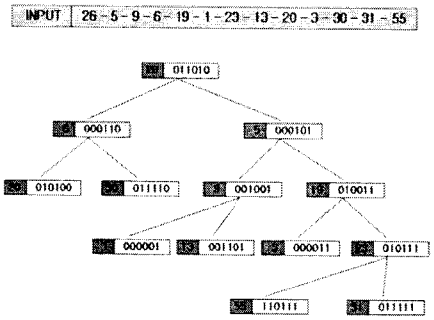


그림 4 비트트리(Bit Tree) 생성

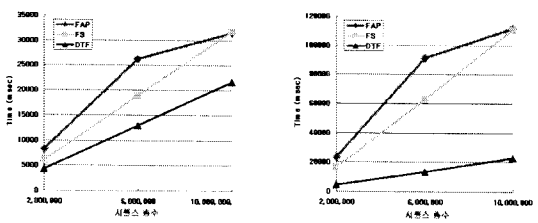
비트트리는 가장 마지막 자리의 비트 값이 0이면 왼쪽, 1이면 오른쪽이라는 방법으로 노드를 늘려 나간다. 해당 위치에 노드가 존재할 경우에는 그 다음 자리의 비트를 이용해 판단하므로 트리의 모양은 데이터의 총 비트수만큼 늘어나게 된다. 이런 방법은 검색과 삽입이 동시에 발생하는 3DFS-Tree에 가장 적합하며, 이를 통해 3DFS-Tree가 좋은 성능을 발휘 하게 된다.

4. 실 험

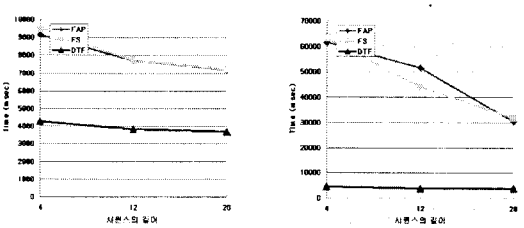
본 논문의 실험은 FS-Tree, 3DFS-Tree에 시퀀스를 추가, 검색하여 성능을 실험 하는 것으로, 후보 패턴의 조건(지지도)을 만족하는 시퀀스들이 존재 한다고 가정하였으며, 데이터는 시퀀스 생성기를 통해 랜덤하게 임의의 시퀀스를 생성하였으며 총 3개의 항목을 가지고 있다.

시퀀스 총수(사용자의 규모), 서로 다른 페이지 수(사이트의 규모), 시퀀스의 길이(유저가 보는 페이지 수)로 비교한다.

4.1 생성 및 점진적 증가 실험



(a) 서로 다른 페이지 50개 (b) 서로 다른 페이지 500개
그림 5 사이트 규모에 따른 생성시 걸리는 시간 성능 비교



(a) 서로 다른 페이지 50개 (b) 서로 다른 페이지 500개
그림 6 유저가 보는 페이지 수의 길이에 따른
점진적인 증가시 걸리는 시간 성능 비교

실험 결과에서 나타나듯이 3DFS-Tree는 사이트의 규모나 유저수, 그리고 유저가 보는 페이지 수에 관계 없이 후보패턴 트리의 생성 및 점진적인 증가시 기존의 트리 보다 안정적이고 좋은 성능을 나타내고 있다.

5. 결 론

본 논문에서는, 기존의 마이닝에서 사용하는 시퀀스의 트리구조를 개선하여 결과적으로 마이닝 속도를 향상 시키는 트리구조를 제시하였으며, 실험을 통해 알 수 있듯이 기존의 트리구조에 비해 데이터의 크기에 관계없이 안정적이고 좋은 성능을 보여주었다. 성능 면에서는 기존의 트리에 비해 높은 성능을 보이지만, 기존의 트리구조에 비해 가독성이 떨어지고 노드 구성이 복잡하여 구현이 어려운 점이 있으며, 전반적인 알고리즘의 이해가 필요 하다.

참 고 문 헌

[1] Maged El-Sayed, Carolina Ruiz, and Elke A.Rundensteiner, "FS-Miner : Efficient and Incremental Mining of Frequent Sequence Patterns in Web logs", WIDM, pp.128-135, 2004.

[2] R. Cooley, B. Mobasher, and J. Srivastava, "Web Mining : Information and Pattern Discovery on the World Wide Web", Proc. of the 9th IEEE International Conference on Tools with Artificial Intelligence, pp.558-567, Nov 1997.

[3] Ramakrishnan Srikant, and Rakesh Agrawal, "Mining Sequential Patterns : Generalizations and Performance Improvements", IBM Almaden Research Center 650 Harry Road, San Jose, CA 95120, pp.3-17, 1996.

[4] S.Parthasarathy, M. J. Zaki, M. Ogihara, S. Dwarkadas, "Incremental and Interactive Sequence Mining", CIKM, pp.251-258, 1999.

[5] Jaideep Srivastava, R. Cooley, M. Deshpande, P-T. Tan, "Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data", SIGKDD Explorations, Volume 1, Issue 2, pp.12-23, Jan 2000.

[6] Xidong Wang, Yiming Ouyang, Xugang Hu, Yan Zhang, "Discovery of User Frequent Access patterns on Web usage Mining", IEEE, pp. 765-769, 2003.