웹 사용 데이타와 하이퍼링크 구조를 통합한 웹 네비게이션 마이닝

(Web Navigation Mining by Integrating Web Usage Data and Hyperlink Structures)

구 흠 모 † 최 중 민 ††
(Heummo Gu) (Joongmin Choi)

요 약 웹 네비게이션 마이닝은 웹 접근 로그 데이타를 분석하여 웹을 항해하는 패턴을 발견하는 기법이다. 하지만 사용자들은 웹을 항해할 때 정상적인 계층적 경로를 따르지 않는 경우가 많기 때문에 웹 접근 로그 데이타에는 웹 항해 패턴 발견에 장애가 되는 잡음 정보가 많이 포함된다. 결과적으로 웹 접근 로그 데이타만을 이용한 기존의 웹 네비게이션 마이닝은 이런 잡음을 해결하기 위한 전처리 과정의 복잡성 등으로 인하여 웹 항해 패턴을 효율적으로 발견하는 데 좋은 성능을 보여주지 못했다.

이런 문제를 해결하기 위해 본 논문에서는 웹 접근 로그 데이타 외에 웹의 하이퍼링크 구조 정보를 함께 이용하여 웹 네비게이션 패턴을 효율적으로 발견하는 기법을 제시하였다. 웹 사이트의 계층적인 하이퍼링크 구조로부터 생성된 WebTree라 불리는 구조를 이용하여 웹 접근 로그 데이타에 포함된 비정상적인 경로에 대한 잡음을 효율적으로 제거하였다. 이 기법을 이용해 구현된 SPMiner(Sequence Pattern Miner) 시스템은 로그 데이타와 하이퍼링크 계층구조를 함께 이용함으로써 전처리의 오버헤드를 현저히 감소시켰고 결과적으로 효율적으로 네비게이션 패턴을 찾아주고 이를 추천에 이용할 수 있는 기반을 제시하였다.

키워드 : 웹 네비게이션 마이닝, 하이퍼링크 구조, WebTree, SPMiner

Abstract Web navigation mining is a method of discovering Web navigation patterns by analyzing the Web access log data. However, it is admitted that the log data contains noisy information that leads to the incorrect recognition of user navigation path on the Web's hyperlink structure. As a result, previous Web navigation mining systems that exploited solely the log data have not shown good performance in discovering correct Web navigation patterns efficiently, mainly due to the complex pre-processing procedure.

To resolve this problem, this paper proposes a technique of amalgamating the Web's hyperlink structure information with the Web access log data to discover navigation patterns correctly and efficiently. Our implemented Web navigation mining system called SPMiner produces a WebTree from the hyperlink structure of a Web site that is used to eliminate the possible noises in the Web log data caused by the user's abnormal navigational activities. SPMiner remarkably reduces the pre-processing overhead by using the structure of the Web, and as a result, it could analyze the user's search patterns efficiently.

Key words : Web navigation mining, hyperlink structure, WebTree, SPMiner

1. 서 론

웹의 급속한 증가로 인하여 정보의 양이 폭발적으로 늘어나면서 사용자는 자신이 필요로 하는 정확한 정보를 찾기가 매우 어려워졌다. 이에 따라, 사용자 측면에서는 원하는 정보를 쉽고 빠르게 찾기 위한 자동화된 툴이 필요하게 되었고, 관리자 측면에서는 웹 사이트를 효율적으로 운영하고 양질의 서비스를 제공하기 위한

· 본 연구는 한국과학재단 지역대학우수과학자 지원연구(과제번호 R05-2003-000-10223-0) 지원으로 수행되었음
† 비 회 원 : LG전자 정보통신사업본부
koomo@lge.com
†† 종신회원 : 한양대학교 컴퓨터공학과 교수
jmchoi@cse.hanyang.ac.kr
논문접수 : 2004년 2월 6일
심사완료 : 2005년 3월 21일

기법이 필요하게 되었다. 결과적으로 이 두 가지 측면을 모두 만족시키기 위하여 사용자들의 웹 사용 패턴을 분석하는 기법이 관심사로 떠오르게 되었다. 웹 사용자들의 항해 패턴을 분석하기 위해서 주로 이용하는 웹 접근 로그 데이타(Web access log data)는 특정 사이트 내에서 사용자들이 웹 문서를 탐색한 정보를 기록한 것이다. 이렇게 저장되는 정보의 양은 매우 방대하지만 단편적이기 때문에 로그 데이타 자체만으로는 유용한 정보를 찾기 어렵다. 이러한 로그 데이타를 분석하여 유용한 정보를 발견하기 위해 데이타 마이닝(data mining) 기술을 기초로 하는 웹 마이닝(Web mining)에 대한 연구가 이루어져왔다[1,2].

웹 네비게이션 마이닝(Web navigation mining)은 사용자가 웹을 항해하는 패턴을 발견하는 기법으로 웹 사용 마이닝(Web usage mining)이라고도 한다[3-5]. 웹 네비게이션 마이닝은 크게 전처리, 패턴 발견, 패턴 분석 등의 단계로 나누어진다. 웹 네비게이션 마이닝은 웹에 존재하는 실제의 데이타를 사용하는 것이 아니라 사용자와 웹 서버의 상호작용에 의하여 만들어진 로그 데이타를 주로 이용하는데 이러한 데이타는 웹 서버 접근 로그, 프락시 서버 로그, 브라우저 로그, 사용자 프로파일, 쿠키 등에 존재한다. 이렇게 발견된 패턴은 전문가의 분석을 통하여 사용자에게 더욱 편리한 서비스를 위하여 웹 페이지를 재구성하거나, 웹 서버 로드 밸런스, 사용자별 맞춤형 웹 페이지 구성, 관심 있는 자료에 대한 추천 등에 이용된다[6-9].

웹 접근 로그 데이타를 이용한 웹 네비게이션 마이닝에서의 어려운 점은 사용자들이 웹을 항해할 때 "뒤로" 버튼이나 "즐겨찾기"를 이용하는 등 정상적인 계층적 경로를 따르지 않는 경우가 많아 로그 데이타에 웹 항해 패턴 발견에 장애가 되는 잡음 정보(noisy information)와 모호한 정보가 많이 포함된다는 것이다. 결과적으로 웹 접근 로그 데이타만을 이용한 기존의 웹 네비게이션 마이닝은 웹 항해 패턴을 효율적으로 발견하는 데 좋은 성능을 보여주지 못했다.

이런 문제를 해결하기 위해 본 논문에서는 웹 접근 로그 데이타 외에 웹의 하이퍼링크 구조 정보를 함께 이용하여 웹 네비게이션 패턴을 효율적으로 발견하는 기법을 제시한다. 본 논문에서 제시하는 알고리즘은 먼저 웹 사이트의 계층적인 하이퍼링크 구조로부터 WebTree 구조를 생성하고, 웹 접근 로그 데이타를 분석하여 문서의 접근 빈도수를 구한다. 이 접근 빈도수에는 잡음이 포함될 수 있으므로 WebTree 구조를 참조하여 이 빈도수를 정상적인 네비게이션에 맞게 수정한다. 마지막으로 WebTree와 수정된 접근 빈도수를 병합한 WebTree+ 구조를 생성한다. 이 WebTree+가 사용자에게 유용한 문서에 대한 네비게이션 패턴의 추천에 사용된다. 이 기법의 유용성을 실험하기 위해 SPMiner(Sequence Pattern Miner)라는 웹 네비게이션 마이닝 시스템을 구현하였고 13개의 웹 사이트에서 얻은 실제 웹 접근 로그 데이타를 적용하여 성능을 평가하였다. SPMiner는 웹 접근 로그 데이타 외에 웹 사이트의 하이퍼링크 구조를 함께 이용함으로써 전처리의 오버헤드를 현저히 감소시켰고 결과적으로 효율적으로 네비게이션 패턴을 찾아주고 이를 추천에 이용할 수 있는 기반을 제시하였다.

본 논문은 다음과 같이 구성된다. 2장에서는 웹 네비게이션 마이닝에서 적용되는 패턴 분석이나 전처리 과정과 관련된 기존 연구를 분석하고 문제점을 알아본다. 3장에서는 SPMiner의 전체적인 시스템 구조와 본 논문에서 제안하는 WebTree+ 생성 알고리즘을 제안한다. 세부적으로는 웹 사이트에서 WebTree를 생성하는 알고리즘, 로그 데이타의 정제와 문서의 접근 빈도수 측정 방법, WebTree를 이용한 로그 데이타의 잡음 제거와 접근 빈도수 수정, WebTree와 수정된 접근 빈도수 정보를 병합한 WebTree+ 생성 방법 등을 제시한다. 4장에서는 WebTree+를 이용하여 웹 항해 패턴을 발견하는 방법과 발견된 정보를 온라인상에서 사용자에게 추천하는 방법을 알아본다. 5장에서는 SPMiner의 구현과 실험 결과를 알아보며, 마지막으로 6장에서는 결론 및 향후 연구 방향을 기술한다.

2. 관련 연구

이 장에서는 웹 네비게이션 마이닝에서 적용되는 패턴 분석 알고리즘이나 전처리 과정과 관련된 기존 연구를 분석하고 문제점을 지적하고자 한다.

웹 네비게이션 마이닝에서 '패턴'(pattern)이라 하면 웹을 항해할 때 거쳐가는 웹 페이지들을 접근 순서대로 나열한 것이며, 웹 로그 파일에 기록된 데이타 중에서 대개 웹 사이트의 하이퍼링크 계층 구조상에서 루트 노드부터 단말 노드까지의 경로를 가리킨다. 따라서 패턴 분석이라 하면 정형화되어 있지 않은 로그 파일에서 정확한 항해 패턴을 찾아내는 것이라고 할 수 있다. 대부분의 패턴 분석 알고리즘에서 발견된 패턴을 평가하는 방법에는 크게 지지도(support)와 신뢰도(confidence)가 있다. 예를 들어, $A \rightarrow B$라는 패턴이 있다면 이 패턴의 지지도는 전체 트랜잭션 중에서 A와 B의 페이지를 동시에 포함하는 트랜잭션의 비율을 말하며, 신뢰도는 A를 만족한 트랜잭션 중에서 B를 포함하는 트랜잭션의 비율을 말한다. 여기서 트랜잭션이란 한번에 발생할 수 있는 아이템이나 이벤트의 집합을 말한다. 본 논문에서 관심을 두고 있는 웹 네비게이션 마이닝에서의 지지도는 대부분 문서의 접근 빈도수(access frequency)로 나

타낸다.

패턴 분석에는 서로간의 연관관계를 찾아내는 연관 규칙(association rule)과 순차적으로 일어난 패턴을 찾아내는 순차 패턴(sequential pattern)으로 크게 나눈다. 연관 규칙을 이용하는 대표적인 알고리즘에는 Apriori 알고리즘과 FP-growth 알고리즘이 있다. Apriori 알고리즘[10]은 가장 작은 크기의 아이템셋(itemset)에서 집합의 크기를 하나씩 증가시켜 각 아이템셋에 대한 지지도를 구하고 최소 지지도 이상의 아이템셋을 찾아 나가는 것이다. 이 알고리즘은 특히, 어떤 아이템셋이 자주 발생하는 패턴이라면 그의 하위 아이템셋 또한 항상 자주 발생하는 패턴이라는 원리를 이용하였다. FP-growth 알고리즘[11]은 기존 Apriori 알고리즘의 문제점인 많은 양의 스캔과 후보 아이템 생성을 줄이기 위하여 FP-tree를 이용하였다. FP-tree는 트리 구조에 트랜잭션별로 경로를 만드는데, 트리에서 아이템이 있다면 빈도수를 증가시키고, 없다면 새로운 아이템 이름을 가지는 노드를 추가시키는 과정을 통해 생성되며, 빈도수가 가장 높은 패턴을 찾는다.

순차 패턴 분석의 알고리즘으로는 GSP 알고리즘, PrefixSpan 알고리즘, SPADE 알고리즘 등이 있다. GSP 알고리즘[12]은 Apriori 알고리즘에 추가적으로 아이템들의 순서를 고려하여 패턴을 찾는 방법이다. 후보를 생성할 때 Apriori는 아이템셋들의 집합을 만들지만 GSP는 순차 패턴들의 집합을 만든다. PrefixSpan 알고리즘[13]은 FP-growth 알고리즘과 같이 각 패턴의 빈도수를 구하고 미리 정해둔 최소 지지도 이상의 패턴들을 찾아내는데, 각각의 패턴을 접두사(prefix)로 하는 서브셋(subsets)을 만들어 분리하는 방법을 취한다. SPADE 알고리즘[14]은 기존의 알고리즘과는 달리 데이타베이스를 트랜잭션 단위가 아닌 아이템 단위의 데이타베이스로 변형하여 사용한다. 또한 연관규칙이 아닌 순차 패턴의 후보 패턴들을 찾아내기 위한 패턴의 길이를 확장시킬 때 모든 트랜잭션을 스캔하는 것이 아니라 아이템 별로 분류된 데이타베이스를 스캔한다.

전처리 과정(pre-processing)은 가공되지 않은 사용 데이타(raw usage data)를 추상적인 데이타로 변환하는 과정이며, 웹 데이타의 불완전성과 잡음으로 인해 웹 네비게이션 마이닝에서 가장 어려운 작업이다. 전처리 과정은 데이타 정제, 사용자/세션 식별, 페이지 뷰 식별, 경로 완성의 네 단계로 구분된다[8]. 데이타 정제 단계는 여러 서버들의 로그를 병합하거나 로그를 데이타 필드 형태로 파싱하는 것으로 사이트를 구체화 하는 단계이다. 사용자/세션 식별 단계는 웹 서버의 접근 시간에 따라 저장된 로그 데이타를 IP 등을 이용하여 사용자와 세션을 기준으로 구분하는 것이다. 하지만 프락시 서버를 사용하거나 IP를 공유하는 경우 사용자 식별이 어렵다. 페이지 뷰 식별 단계는 요청된 여러 문서를 하나의 페이지 뷰로 식별하는 것이다. 페이지 뷰는 실제 웹 브라우저를 통해 보는 한 페이지를 말하는데, 보통은 하나의 문서가 한 페이지를 구성하지만 프레임을 이용하는 경우에는 여러 문서가 하나의 페이지를 나타낼 수 있다. 경로 완성 단계는 웹 브라우저의 캐시(cache)에 저장된 페이지 참조를 추론하는 것이다. 네트워크의 오류, 웹 브라우저의 캐시, 관심 있는 문서의 즐겨찾기 등과 같은 웹의 특성으로 인한 잡음을 해결한다. 이와 같은 전처리 과정을 수행하면 시간을 기준으로 저장된 기존 웹 서버 로그 데이타는 사용자의 트랜잭션을 기준으로 변환된다.

앞에서 살펴본 여러 패턴 분석 방법의 가장 큰 문제점은 전처리 과정의 복잡성이다. 즉, 현재까지의 웹 네비게이션 마이닝은 웹 접근 로그 데이타를 트랜잭션 형태로 저장된 데이타베이스를 만들기 위하여 복잡한 전처리 과정을 거치고, 이 데이타베이스에서 기존 알고리즘을 이용하여 패턴을 분석하게 된다. 이는 전처리 과정의 복잡성과 시간의 비효율성을 초래한다. 또한 기존의 방법은 웹 구조를 반영하지 못하기 때문에 발견된 유용한 패턴들이 실제 웹 구조와 다른 형태로 나타날 수 있고, 이로 인해 패턴의 신뢰성이 떨어져서 실제 적용 후 잘못된 결과를 유발할 수 있다. 또한 마이닝 과정에서 발생하는 매우 방대한 양의 후보 아이템셋도 문제점으로 대두된다.

본 논문에서는 이러한 문제를 해결하기 위하여 웹 접근 로그 데이타 외에 웹의 하이퍼링크 구조를 추가로 이용하여 로그 파일의 전처리 과정의 복잡도를 현저히 줄임으로써 사용자의 탐색 패턴 분석을 효율적으로 수행할 수 있는 새로운 웹 네비게이션 마이닝 방법을 제안하는 것이다.

3. SPMiner: 웹 네비게이션 마이닝 시스템

그림 1은 본 논문에서 구현한 웹 네비게이션 마이닝 시스템인 SPMiner의 구조와 기능적 흐름을 보여준다. SPMiner는 크게 전처리(Preprocessing), 패턴 발견(Pattern Discovery), 패턴 분석 및 추천(Pattern Analysis and Recommendation)의 세 부분으로 이루어져 있다.

SPMiner의 입력 데이타는 특정 웹 사이트의 URL과 그 웹 서버에 저장된 웹 접근 로그 파일이다. SPMiner의 전처리 과정에서는 먼저 입력받은 URL로부터 하이퍼링크를 이용하여 WebTree를 만든다. 웹 서버의 로그 파일은 데이타 정제(data cleaning) 과정을 거친 후 각 참조 문서에 대한 접근 빈도수를 구하는데 이용된다. 이 두 결과를 병합하여 WebTree에 문서의 접근 빈도수를 매핑하는 과정을 거치면 WebTree+가 생성된다. 이런

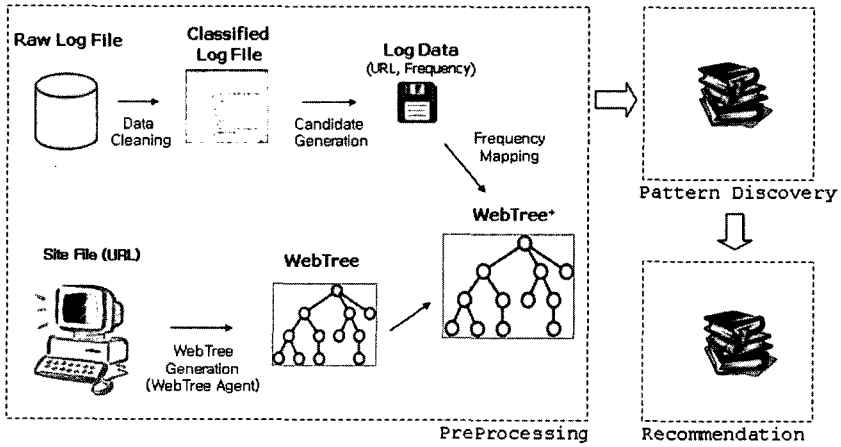


그림 1 SPMiner 시스템 구조

전처리 과정을 거쳐 얻어진 WebTree+는 패턴 발견 단계와 추천 단계에서 웹 사이트에 대한 네비게이션 패턴을 발견하고 온라인상에서 웹 사이트를 사용하고 있는 사용자에게 패턴을 추천해 줌으로써 웹 네비게이션의 효율을 높여준다.

3.1 WebTree 생성

웹 사이트의 구조는 웹 문서들 간의 연결 관계를 나타내는 하이퍼링크에 의해 표현된다. 이러한 웹 사이트 구조는 방향성 있는 그래프(directed graph)를 이용해 나타낼 수 있는데, 웹 문서는 그래프의 노드(node)로 표현되고, 문서 간의 하이퍼링크는 아크(arc)로 표현된다. 그림 2는 웹 사이트 구조를 그래프로 나타낸 한 예이다.

이러한 웹 사이트에 대한 하이퍼링크 그래프는 WebTree로 추상화시킬 수 있다. WebTree는 웹 사이트 그래프를 문서 노드와 그 문서의 자식 노드의 집합을 이용해서 정의한 선형적 표현이다. 즉, WebTree는 $WT=(d_1, C_1),(d_2, C_2),...,(d_n, C_n)$로 나타낼 수 있는데, 여기서 d_i는 문서, C_i는 d_i에서 하이퍼링크로 연결된 문서들의 집합을 각각 나타낸다. SPMiner에서 각 문서는 URL로 표현된다.

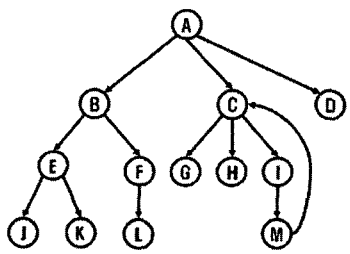


그림 2 웹 사이트 구조의 예

다음은 그림 2의 웹 사이트 구조를 WebTree로 표현한 것이다.

WT={(A,{B,C,D}), (B,{E,F}), (C,{G,H,I}), (D,{ }), (E,{J,K}), (F,{L}), (G,{ }), (H,{ }), (I,{M}), (J,{ }), (K,{ }), (L,{ }), (M,{C})}

본 논문에서는 효과적인 WebTree 생성을 위해 두 가지 가정을 한다. 첫째, 주어진 도메인이 아닌 다른 사이트에 있는 문서로 연결된 하이퍼링크는 탐색하지 않는다. 예를 들어, 작업 도메인이 http://cse.hanyang.ac.kr의 사이트라면 검색 중에 http://eecs.hanyang.ac.kr로 시작하는 URL로 연결된 링크는 탐색하지 않는다. 이는 에이전트가 무한정 웹을 떠돌아다니는 것을 방지한다. 둘째, 한번 탐색된 페이지는 다음 탐색에서 제외된다. 한번 탐색된 페이지가 다음 탐색에 다시 등장하는 경우는 사이클이 있는 경우이다. 예를 들어, 그림 2의 그래프에서 C, I, M 페이지들은 서로 사이클을 형성하므로 $C \rightarrow I$, $I \rightarrow M$을 탐색한 후에 $M \rightarrow C$의 하이퍼링크도 탐색한다면 무한루프에 빠질 수 있다. 따라서 이미 전 단계에서 탐색했던 C 문서는 다시 탐색하지 않는다. 이러한 가정을 기반으로 한 WebTree 생성 알고리즘의 의사코드(pseudo-code)는 그림 3과 같다.

이 알고리즘에서 WebTree는 WT 변수에 나타내었으며, 이것의 초기값은 null이고 이후 반복적으로 문서와 하이퍼링크를 통한 하위 문서(자식 노드) 집합의 쌍이 추가된다. 문서에서 하이퍼링크로 연결된 문서를 반복적으로 탐색하기 위해 새로 나타나는 문서 노드를 큐에 저장한 후 이 큐에서 하나씩 꺼내어 처리하게 된다. *DocQ*는 문서 노드를 저장하는 큐를 가리키고, **enqueue()**와 **dequeue()**는 각각 큐에 노드를 추가하고 꺼내는 함수를 나타낸다. L은 문서 d_k에서 하이퍼링크

로 연결된 모든 문서 노드의 집합을 나타낸다. (알고리즘에서 하이퍼링크의 연결은 $d_k \rightarrow d$로 표시하였다.) A는 이미 탐색된 문서를 저장하는 집합이며, 큐에서 꺼낸 각 문서는 처리되기 전에 A에 추가되어 다음번에 같은 문서가 나올 때는 다시 처리하지 않도록 해준다.

```
WebTree() {
  WT ← ∅; // WebTree
  A ← ∅; // 처리된 노드 저장
  DocQ ← ∅; // 처리될 노드를 저장하기 위한 큐
  d1 ← 웹 사이트의 URL; // 루트 노드(문서)
  k ← 1;

  repeat {
    A ← A ∪ {dk};
    L ← {d | dk → d};  // dk에서 하이퍼링크로 연결된 노드 집합
    Ck ← {c | c ∈ L, c ∉ A}; // 사이클을 이루지 않는 자식 노드 집합
    WT ← WT ∪ {(dk,Ck)};
    for each c ∈ Ck {
        enqueue(DocQ, c);
    }
    k ← k + 1;
    dk ← dequeue(DocQ);
  } until (dk = ∅);  // 모든 노드가 처리될 때까지 반복 수행

  return WT;
}
```

그림 3 WebTree 생성 알고리즘

3.2 웹 접근 로그 데이타 정제

웹 접근 로그 데이타는 사용자의 웹 사이트 문서에 대한 접근 사항을 기록하는 데이타이며, 일반적으로 CLF(Common Log Format)나 ECLF(Extended Common Log Format) 형식을 따른다. 로그 데이타는 웹 서버에 저장되며 모든 사용자 행동을 시간을 기준으로 저장한다. 이런 로그 데이타는 그 자체로는 큰 의미를 가지지 못하기 때문에 사용자의 탐색 패턴 정보를 알아내기 위해서는 먼저 전처리 과정을 통해서 마이닝을 수행할 수 있는 형태로 변환되어야 한다. 그림 4는 CLF형태의 웹 서버 로그 데이타의 한 예로서 여기에는 사용자를 구분할 수 있는 IP 주소, 접속시간, 접근한 페이지의 URL, 프로토콜, 문서의 상태 및 크기 등 다양한 종류의 데이타가 기록된다. SPMiner에서는 이 중에서 URL 부분만을 추출하여 WebTree+ 생성에 이용한다.

이렇게 로그 데이타 정제 과정을 통하여 얻어진 데이타를 이용하여 각 문서에 대한 접근 빈도수(access frequency)를 계산한다. 접근 빈도수는 문서의 지지도 값을 측정하기 위한 기초 데이타가 된다. 그림 5는 로그 파일로부터 필요한 URL 부분을 추출하고 각 URL에 대한 빈도수를 구하는 과정을 보여주고 있다.

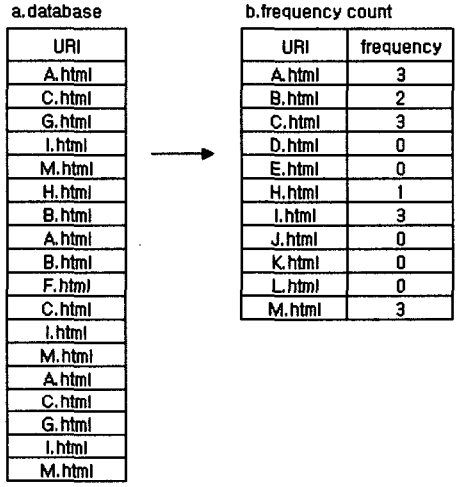


그림 5 로그 데이타 정제 후 각 문서의 접근 빈도수

3.3 문서의 접근 빈도수 수정 및 지지도 측정

웹 사이트 구조를 하이퍼링크에 의한 계층적인 구조로 본다면 이런 계층적 구조에 부합되는 정상적인 웹

IP Address	Auth ID	Date / Time	Method/URI/Protocol	Status	Size
61.97.17.223	- -	[19/Sep/2002:00:27:43 +0900]	"GET /A.html HTTP/1.1"	200	3206
61.97.17.223	- -	[19/Sep/2002:00:27:43 +0900]	"GET /C.html HTTP/1.1"	200	12154
61.97.17.223	- -	[19/Sep/2002:00:27:50 +0900]	"GET /G.html HTTP/1.1"	200	13458
61.97.17.223	- -	[19/Sep/2002:00:27:53 +0900]	"GET /I.html HTTP/1.1"	200	24857
61.97.17.223	- -	[19/Sep/2002:00:27:54 +0900]	"GET /M.html HTTP/1.1"	200	10808
61.97.17.223	- -	[19/Sep/2002:00:32:29 +0900]	"GET /H.html HTTP/1.1"	200	20171
61.97.17.223	- -	[19/Sep/2002:00:32:34 +0900]	"GET /B.html HTTP/1.1"	200	38207
203.200.5.189	- -	[19/Sep/2002:01:09:57 +0900]	"GET /A.html HTTP/1.1"	200	3206
203.200.5.189	- -	[19/Sep/2002:01:09:57 +0900]	"GET /B.html HTTP/1.1"	200	12154
203.200.5.189	- -	[19/Sep/2002:01:10:40 +0900]	"GET /F.html HTTP/1.1"	200	13458
166.104.226.206	- -	[19/Sep/2002:09:24:16 +0900]	"GET /C.html HTTP/1.1"	200	3206
166.104.226.206	- -	[19/Sep/2002:09:24:16 +0900]	"GET /I.html HTTP/1.1"	200	12154
166.104.226.206	- -	[19/Sep/2002:09:24:16 +0900]	"GET /M.html HTTP/1.1"	200	24857
203.252.115.231	- -	[19/Sep/2002:22:06:17 +0900]	"GET /A.html HTTP/1.1"	200	3206
203.252.115.231	- -	[19/Sep/2002:22:06:18 +0900]	"GET /C.html HTTP/1.1"	200	12154
203.252.115.231	- -	[19/Sep/2002:22:06:26 +0900]	"GET /G.html HTTP/1.1"	200	13458
203.252.115.231	- -	[19/Sep/2002:22:06:35 +0900]	"GET /I.html HTTP/1.1"	200	16948
203.252.115.231	- -	[19/Sep/2002:22:06:41 +0900]	"GET /M.html HTTP/1.1"	200	71897

그림 4 웹 접근 로그 데이타의 예

사이트 방문에서는 가장 상위 문서를 먼저 접속한 후 그 문서에서 하이퍼링크로 연결된 하위 문서들을 탐색해 나간다. 이 경우 상위 문서가 하위 문서보다 항상 접근 빈도수가 높게 나온다. 즉, 현재 문서에 대한 접근 빈도수는 부모 문서(상위 문서)의 빈도수보다 크지 않고, 모든 자식 문서의 접근 빈도수를 합친 것보다 작지 않다. 따라서 $freq_N$을 노드 N의 접근 빈도수라고 하고, *current*를 현재 노드, *parent*를 현재 노드의 부모 노드, *child*를 현재 노드의 자식 노드라 하면 다음의 식 (1)이 성립한다.

$$freq_{parent} \geq freq_{current} \geq \Sigma freq_{child} \quad (1)$$

하지만 웹의 특성과 네트워크의 불안정성 때문에 이런 정상적인 방문 외의 다른 형태가 빈번히 나타나며 이를 웹 로그 데이타의 잡음(noise)으로 처리할 수 있다. 웹 로그 데이타에 잡음이 포함되는 경우를 정리해 보면 다음과 같다.

첫째, 네트워크의 오류에 의한 경우이다. 서버와 사용자는 네트워크로 연결되어 원하는 정보를 주고받는다. 이때 네트워크의 불안정으로 데이타가 사라지는 경우가 발생하며, 그 결과 방문은 했지만 로그 데이타에 기록되지 못하는 경우가 생긴다. 둘째, 웹 브라우저의 캐시(cache)의 기능 때문에 발생하는 경우이다. 웹 브라우저의 캐시는 웹 페이지의 로딩 시간을 줄여주기 위해 사용자들이 탐색한 문서들을 임시로 저장한다. 보통 사용자들은 이전 문서로 돌아가기 위하여 하이퍼링크를 이용하지 않고 웹 브라우저의 '뒤로'(back) 버튼을 누른다. 이때 나타나는 문서는 캐시에 이미 저장되어 있기 때문에 웹 서버에 대한 새로운 접근이 발생하지 않고, 따라서 이전에 탐색하여 캐시에 저장된 문서는 여러 번 방문한 경우도 처음 방문 외에는 서버의 로그 데이타에 기록되지 않는다. 셋째, 관심 있는 문서에 대한 '즐겨 찾기'를 이용하는 경우이다. 사용자는 관심 있는 문서를 편리하게 찾기 위하여 웹 브라우저의 '즐겨 찾기' 기능을 많이 사용한다. 따라서 사용자는 특정 웹 문서를 바로 탐색할 수 있으며 이러한 경우 로그 데이타에는 기록되지만 상위 문서를 통해 온 것이 아니므로 상위 문서들은 로그 데이타에 기록되지 않는다.

이런 현상 때문에 현재 문서의 빈도수가 부모 문서의 빈도수보다 클 수도 있고, 또한 자식 문서들의 빈도수의 합보다 작을 수도 있어서, 접근 빈도수에 대한 식 (1)에서 부등호가 반대로 될 수가 있다. 하지만 식 (1)의 경우를 제외한 모든 경우는 로그 데이타의 잡음 때문에 나타나며, 이것은 웹 마이닝을 통한 문서 추천을 어렵게 만드는 요인이 된다. 따라서 이 문제를 해결하기 위해 본 논문에서는 이전 단계에서 구한 WebTree 구조를 활용하여 잡음 정보를 제거하여 항상 식 (1)이 만족되도록 빈도수를 수정하게 되는데, 다음과 같은 규칙을 재귀적으로 반복하여 적용한다.

- 현재 문서가 자식 문서들의 빈도수의 합보다 작지 않으면 정상적인 경우이므로 현재 문서의 빈도수를 그대로 유지한다.
- 현재 문서가 자식 문서들의 빈도수의 합보다 작으면 현재 문서의 빈도수를 하위 문서들의 빈도수의 합으로 수정한다.

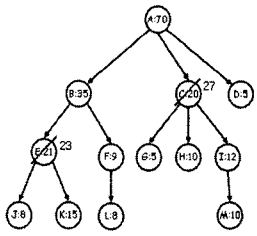
즉, $freq_{current} < \Sigma freq_{child}$이면 $freq_{current} = \Sigma freq_{child}$로 만든다.

접근 빈도수는 웹 사이트나 로그 파일 크기에 따라 달라질 수 있기 때문에 SPMiner에서는 각 페이지의 빈도수를 정규화시키고 이 값을 그 페이지의 지지도(support) 값으로 정한다. 웹 사이트의 도메인 URL에 해당하는 루트 페이지는 하위 문서보다 항상 빈도수가 크므로 이를 기준으로 각 문서의 빈도수를 정규화된 값으로 바꿀 수 있다. 즉, 현재 노드를 *current*, 루트 노드를 *root*라고 하면 현재 노드의 지지도 값은 다음과 같다.

$$Support(current) = freq_{current} / freq_{root}$$

그림 6은 WebTree를 이용하여 로그 데이타로부터 얻어진 각 문서의 빈도수를 수정하고 지지도를 구하는 과정을 예를 들어 보여주고 있다. 이 과정에서 문서 C와 문서 E의 빈도수가 하위 문서들의 빈도수의 합보다 작은 경우가 발생하여 각각 하위 문서 빈도수의 합인 27과 23으로 수정된 것을 볼 수 있다.

(a) WebTree (b) 빈도수 업데이트 & 지지도 측정



Node	Frequency	Update	Support
A	70	70	1.00000
B	35	35	0.50000
C	20	27	0.38571
D	5	5	0.07143
E	21	23	0.32857
F	9	9	0.12857
G	5	5	0.07143
H	10	10	0.14286
I	12	12	0.17143
J	8	8	0.11429
K	15	15	0.21429
L	8	8	0.11429
M	10	10	0.14286

[Update] [Support]

$C = \sum freq_{Children\ of\ C} = 27$ $Support(Node) = freq_{Node} / freq_{Root}$

$F = \sum freq_{Children\ of\ F} = 23$

그림 6 접근 빈도수 수정 및 지지도 계산

3.4 WebTree+ 생성

WebTree+는 WebTree에 로그 데이타에서 얻어진 노드의 지지도 값을 매핑한 것이다. 그림 7에 WebTree+ 생성 알고리즘을 의사코드로 나타내었다.

```
WT ← WebTree(); // WebTree 생성 모듈
d1 ← 루트 노드;

WebTreePlus() {
  WTplus ← Ø; // WebTree+
  N ← Ø; // 접근 문서 중에 WebTree에 없는 문서 저장
  for each (d, C) ∈ WT {
     WTplus ← WTplus ∪ {(d, C, 0, 0)}; // WTplus 초기화
  }
  for each (d, f) ∈ FreqCounts { // access frequency counts
     if (IsContained(d, WTplus)) // 접근된 문서가 WTplus에 있는지 검사
       AssignFreq(WTplus, d, f); // WTplus 수정
     else N ← N ∪ {d};
  }
  AdjustFreq(WTplus, d1); // 접근 빈도수 수정
  SupportNorm(WTplus, d1); // 지지도 계산
}

AdjustFreq(WTplus, d) {
   sumChildFreq ← 0; // 자식노드의 접근빈도수의 합 초기화
   C ← GetChildNodes(WTplus, d); // C는 d의 자식노드 집합
   f ← GetFreq(WTplus, d);  // f는 문서 d의 접근 빈도수
   if (C ≠ Ø) {
     for each c ∈ C {
        r ← AdjustFreq(WTplus, c); // 자식노드에 대한 recursive call
        sumChildFreq = sumChildFreq + r; // 자식노드의 빈도수를 합산
     }
   }
   // 현재 노드의 빈도수가 자식노드의 빈도수 합보다 작으면
   // 현재 노드의 빈도수를 자식노드 빈도수의 합으로 만든다.
   if (f < sumChildFreq) {
     f ← sumChildFreq;
     AssignFreq(WTplus, d, f);
   }
   return f;
}

SupportNorm(WTplus, d) {
   // max_frequency : WTplus에서 루트 노드의 접근 빈도수 값
   max_frequency = GetFreq(WTplus, d);
   for each (d, C, f, s) ∈ WTplus {
     s = f / max_frequency; // 지지도는 루트의 접근 빈도수에 대한 상대값
     AssignSupport(WTplus, d, s); // d 문서의 지지도를 s로 변경
   }
}
```

그림 7 WebTree+ 생성 알고리즘

여기서, *WTplus*는 이전 단계에서 생성된 WebTree 구조인 *WT*에다가 접근 빈도수와 지지도 데이타를 추가한 구조이다. 따라서 $WTplus = (d_1, C_1, f_1, s_1), (d_2, C_2, f_2, s_2), \ldots, (d_n, C_n, f_n, s_n)$로 표현될 수 있는데, 여기서 f_i와 s_i는 문서 d_i의 접근 빈도수와 지지도를 각각 나타낸다. 이 두 값은 WebTree+ 생성 알고리즘이 시작될 때 0의 값으로 초기화되며 알고리즘이 진행되면서 실제 값이 할당된다. 변수 *FreqCounts*는 3.2절에서 기술한 로그 데이타 정제 후에 얻어진 각 문서에 대한 접근 빈도수를 $FreqCounts = (d_1, f_1), (d_2, f_2), \ldots, (d_n, f_n)$의 형태로 저장하고 있다. *N*은 *FreqCounts*에 저장된 문서 중에서 *WTplus*에 매핑되지 못한 문서들을 저장한다.

WebTree+ 생성 알고리즘은 먼저 로그 데이타에서 구한 각 문서에 대한 빈도수, 즉 *FreqCounts*에 있는 데이타를 *WTplus*에 매핑시킨다. 이때, 웹 사이트에서 하이퍼링크로 연결되어 있지 않은 문서들은 매핑이 되지 않기 때문에 이런 문서들은 따로 *N*에 저장한다. 그런 다음 앞 절에서 기술한 빈도수 수정 알고리즘을 구현한 **AdjustFreq()** 함수를 호출하여 *WTplus*의 빈도수를 수정한다. 마지막으로 **SupportNorm()**을 이용해서 WebTree+ 노드의 접근 빈도수를 0~1사이의 지지도 값으로 정규화시킨다. WebTree+ 생성 알고리즘의 결과는 수정된 빈도수와 지지도가 추가된 *WTplus*와 매핑이 되지 않은 문서의 집합 *N*이다. *WTplus*는 다음 장에서 설명할 탐색 패턴의 발견과 추천에 필요한 자료가 되며, *N*은 부가적으로 웹 사이트 관리에 유용한 자료가 된다.

4. 탐색 패턴 발견 및 사용자를 위한 문서 추천

4.1 백트래킹을 이용한 탐색 패턴 발견

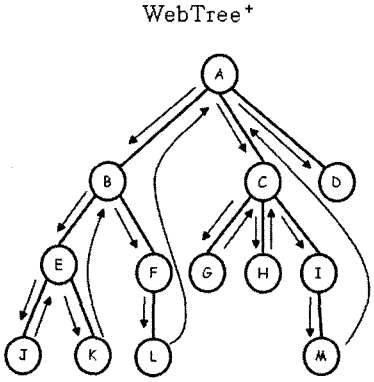
탐색 패턴 발견은 WebTree+에서 미리 지정한 최소 지지도(min_support)를 만족하는 패턴을 찾는 것이다. WebTree+에서는 사용자의 탐색 패턴을 찾기 위해 [15]에서 제시된 maximal forward reference(MFR) 알고리즘을 사용하였다. 이 알고리즘에서 MFR이라는 것은 페이지 방문 기록에서 백트래킹(backtracking)이 발생하기 바로 전의 노드를 가리킨다. 예를 들어, *A-B-A-C-D-C*라는 순서로 페이지를 방문했다면 *B*와 *D*가 MFR이 되고, *AB*와 *ACD*가 각각 탐색 패턴이 된다. 그림 8은 WebTree+로부터 흥미로운 패턴을 찾는 과정을 보여주고 있다.

가장 상위 문서인 *A* 노드부터 시작하여 탐색하면서 미리 지정해 둔 최소 지지도 이상의 경로들을 후보 패턴으로 선택하게 된다. 현재 문서의 지지도가 최소 지지도보다 작은 경우 그 하위 문서들은 검색하지 않는데 그 이유는 WebTree+에서는 상위 문서의 지지도가 하위 문서의 지지도보다 항상 크기 때문이다. 예를 들어, min_support = 0.2라고 한다면 그림 8에서 6번과 11번 패턴의 경우 상위 경로의 지지도가 각각 0.12857과 0.17143으로 최소지지도 0.2를 만족하지 못하므로 탐색을 하지 않는다. 이렇게 함으로써 패턴 탐색의 시간을 줄일 수 있다. 결과적으로 그림 8에서 추천할 패턴은 검은색으로 칠해진 4개의 경로가 된다. 참고로 추천 패턴의 경로는 경로상의 마지막 노드만 표시해도 유일하게 결정될 수 있기 때문에 위 예제에서의 추천 노드는 *B*, *C*, *E*, *K* (지지도 순위)가 된다.

4.2 지지도 재계산에 의한 문서추천

WebTree+에서 발견된 패턴들은 사용자가 웹 사이트를 항해하고 있을 때 좀더 유용한 정보를 추천할 때 사용된다. 사용자를 위한 문서 추천은 두 단계로 나누어지는데, 첫 번째는 사용자가 웹을 탐색할 때 현재 위치하고 있는 문서에서 하위 문서에 대한 흥미있는 패턴을 추천해주는 "패턴 추천" 단계이고, 두 번째는 현재 사용자의 탐색 문서에 대한 접근 빈도수와 지지도를 새롭게 계산하는 "지지도 재계산" 단계이다.

그림 9는 사용자 추가 탐색 자료에 의한 지지도 재계산 방법을 보여주고 있다. (a)는 추가적인 사용자의 탐색 데이타로서, 현재 탐색중인 페이지에 대한 정보이며, (b)는 사용자의 탐색에 따른 빈도수와 지지도의 변화를 보여주고 있다. 지지도의 재계산은 사용자의 탐색 중 한 트랜잭션이 완성될 때 이루어진다. (a)에서 *A*, *C*, *H*가 하나의 트랜잭션이므로 (b)의 ①의 과정을 수행하여 *A*,



탐색 패턴 발견
(*Min_support = 0.2*)

No.	Search	support	pattern
1	AB	0.50000	AB
2	ABE	0.32857	ABE
3	ABEJ	0.11429	-
4	ABEK	0.21429	ABEK
5	ABF	0.12857	-
6	ABFL	0.11429	-
7	AC	0.38571	AC
8	ACG	0.07143	-
9	ACH	0.14286	-
10	ACI	0.17143	-
11	ACIM	0.14286	-
12	AD	0.07143	-

(6,11) : 상위문서의 지지도가 최소 지지도를 만족하지 못하므로 탐색하지 않음

그림 8 탐색 패턴 발견 예제

Search	URL	Transaction	
1	A.html	-	
2	C.html	-	
3	H.html	A C H	①
4	I.html	-	
5	M.html	A C I M	
6	A.html	-	
7	D.html	A D	
8	C.html	-	
9	I.html	A C I	
10	A.html	-	
11	C.html	-	
12	I.html	A C I	
13	M.html	A C I M	
14	H.html	A C H	②
15	A.html	-	
16	C.html	-	
17	...	...	

(a) 사용자 추가 탐색 자료

Node	Frequency	Support	①		②	
			Frequency	Support	Frequency	Support
A	70	1.00000	71	1.00000	77	1.00000
B	35	0.50000	35	0.49296	35	0.45455
C	27	0.38571	28	0.39437	33	0.42857
D	5	0.07143	5	0.07042	6	0.07792
E	23	0.32857	23	0.32394	23	0.29870
F	9	0.12857	9	0.12676	9	0.11688
G	5	0.07143	5	0.07042	5	0.06494
H	10	0.14286	11	0.15493	13	0.16883
I	12	0.17143	12	0.16901	16	0.20779
J	8	0.11429	8	0.11268	8	0.10390
K	15	0.21429	15	0.21127	15	0.19481
L	8	0.11429	8	0.11268	8	0.10390
M	10	0.14286	10	0.14085	11	0.14286

(b) 지지도 재계산에 의한 문서 추천

그림 9 사용자 추가 탐색 자료에 의한 지지도 재계산

C, *H*에 대한 빈도수를 증가시키고 각 문서의 지지도를 새롭게 계산한다. 하지만 이 결과로 최소 지지도를 넘는 후보 패턴은 *A*, *B*, *C*, *E*, *K*로 변화가 없다. 그 후에 ②의 경우를 처리할 시점에서는 초기 빈도수보다 *A*는 7, *C*는 6, *D*는 1, *H*는 3, I는 4, *M*은 1이 증가된다. 이에 따라 지지도를 재계산하게 되면 기존의 *K*는 탈락하고 새롭게 *I*가 추가됨을 알 수 있다. 이런 방법으로 사용자들은 웹을 탐색하면서 동적으로 새로운 후보 패턴을 추천받게 되고 그 결과 사용자의 효율적인 웹 탐색에 도움을 줄 수 있다.

5. 구현 및 실험

이 장에서는 앞 장에서 논의한 알고리즘과 이론들을 바탕으로 한 SPMiner 시스템의 구현 상황을 기술하고, 실험을 통해 패턴 분석 결과의 정확성과 시스템의 성능을 알아보려고 한다.

5.1 시스템 구현

그림 10은 SPMiner의 메인 인터페이스이며 그림 11은 데이타 정제 후 빈도수 측정 결과를 나타낸 것이다. 여기서 도메인 URL은 http://islab.hanyang.ac.kr/ 이며 이 도메인에 대해 WebTree 버튼을 누르면 하이퍼링크 구조를 추출하여 WebTree를 생성하게 된다. 왼쪽 창에는 WebTree에 대한 XML 구조가 계층적으로 나타나 있다.

그림 12는 WebTree+의 실행 결과를 나타내고, 그림 13은 패턴 발견 결과를 보여주고 있는데 미리 주어진 최소 지지도(min_support =0.2)이상의 값을 찾아 URL과 지지도를 함께 보여준다.

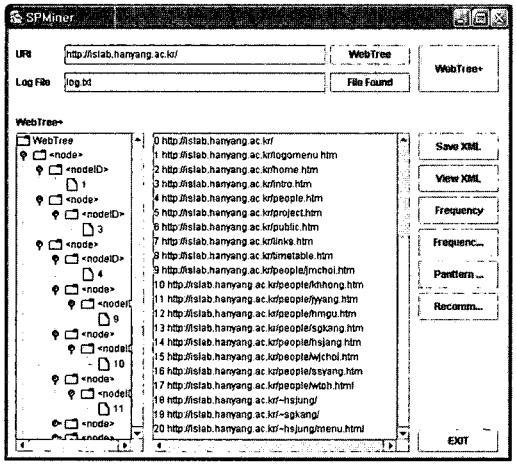


그림 10 SPMiner 메인 인터페이스

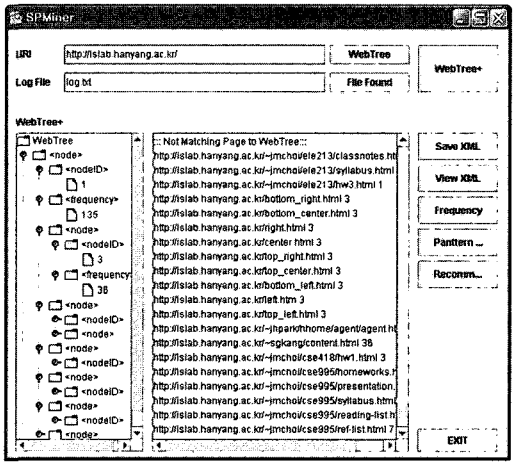


그림 11 로그 데이타 정제 후 빈도수 측정 결과

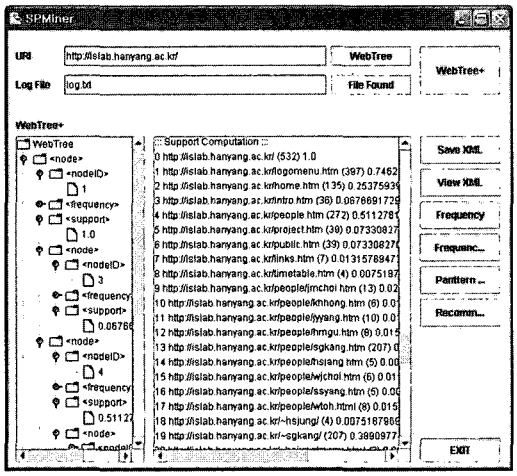


그림 12 WebTree+ 실행 결과

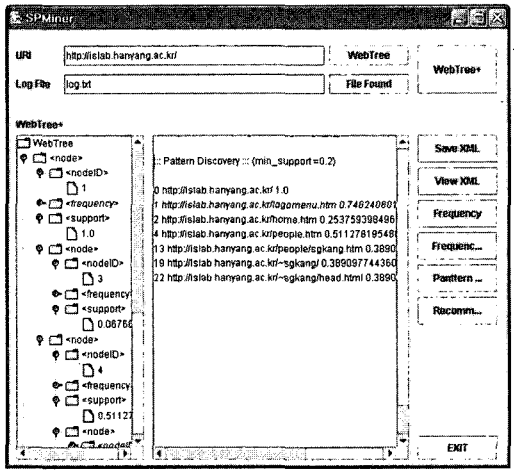


그림 13 WebTree+를 이용한 패턴 발견 결과

5.2 실험 결과

SPMiner의 성능 실험을 위해 한양대학교 컴퓨터공학과에 소속되어 있는 13개 대학원 연구실의 웹 사이트에서 얻은 로그 데이타를 사용하였다. 그림 14는 이 사이트들에 대한 하이퍼링크 문서를 추출하여 WebTree+를 생성할 때 얻은 결과들을 정리한 것이다. 1번과 3번 도메인의 경우 검색 링크수가 각각 992개와 741개가 검색되었는데 다른 도메인보다 큰 이유는 자바 문서나 여러 메뉴얼이 도메인 내에 포함되어 있었기 때문이다. 로딩 에러는 웹 서버상에 문서가 존재하지 않지만 하이퍼링크가 존재하는 경우이다.

그림 15는 최소 지지도가 0.2일 때 각 도메인에 대한 추천 패턴의 생성 결과를 나타낸다. 현재는 각 도메인 사이트에서 많이 접근되는 페이지에 대한 네비게이션 패턴을 생성하는데 그치고 있지만 추후에는 이런 후보 패턴을 바탕으로 실제 사용자가 웹 네비게이션 도중에 후보 패턴에 대한 추천을 받아서 다음 네비게이션할 페이지를 선택하는데 도움을 주도록 할 예정이다.

본 논문의 주된 목적은 기존 웹 사용 마이닝에서 사용하지 않았던 웹 사이트 구조(WebTree)를 이용하여 불필요한 웹 로그 데이타를 무시하고 이를 바탕으로 패턴 추출의 성능을 높이는 것이다. 따라서 여기서의 성능

No.	URL	검색링크수	로딩에러	특이사항
1	http://viplab.hanyang.ac.kr/	992	17	<a href="#gcc"> <a href="notices/index.php"> <a href="http://viplab.hanyang.ac.kr/%7Edrkim"> MySQL Reference Manual , JAVA api 외 다수의 문서
2	http://ailab.hanyang.ac.kr/	147	46	
3	http://para1.hanyang.ac.kr/	741	92	cgi-bin 관련 문서
4	http://osnn.hanyang.ac.kr/	72	4	다른 도메인 사용(oslab.hanyang.ac.kr)
5	http://commlab.hanyang.ac.kr/	48	1	
6	http://visionlab.hanyang.ac.kr/	38	0	
7	http://distcomp.hanyang.ac.kr/	54	8	
8	http://cns.hanyang.ac.kr/	16	0	<area shape="rect" coords="0,-1,54,17" href="indexEng.htm">
9	http://mslab.hanyang.ac.kr/	66	26	로그인이 필요한 페이지
10	http://pllab.hyu.ac.kr/	107	0	<a href=people.htm class=menu> <a href= "einstein.html"> <a href= "...">
11	http://islab.hanyang.ac.kr/	63	0	
12	http://aspen.hanyang.ac.kr/	11	1	
13	http://salab.hanyang.ac.kr/	16	5	다른 도메인 사용(selab.hanyang.ac.kr)

그림 14 13개 도메인에 대한 WebTree+ 생성의 실험 결과

No.	URL	패턴
1	http://viplab.hanyang.ac.kr/	/members/index.html /~ryul /~ryul/docs/index.html
2	http://allab.hanyang.ac.kr/	/index-k.htm /~ktkim/main.htm /~ktkim/opening.htm /~ktkim/frm_main_content.htm /~emchoi/cwb-data/data/Sun/index.htm
3	http://para1.hanyang.ac.kr/	/f-introduction.htm /menu.htm /~chieon /member.htm /~srkwon
4	http://oslab.hanyang.ac.kr/	/main.html /member.html /~dbkwon /~dbkwon/linuxnote.htm
5	http://commlab.hanyang.ac.kr/	/index3.htm /tail.htm /eindex.htm /professor/professor.htm /kindex.htm /class/class1_k.htm /class/class5_k.htm
6	http://visionlab.hanyang.ac.kr/	/top.htm /mid.htm /main.htm /professor.htm /undergraduate.htm /ktpark
7	http://distcomp.hanyang.ac.kr/	/menu.html /main.html /research/index.html /~sangjin/research/crypt/resource.html /~sangjin/research/crypt/protocols/reference.html
8	http://cns.hanyang.ac.kr/	/eng1.htm /eng2.htm /eng3.htm /index.htm
9	http://mslab.hanyang.ac.kr/	/index_up.html /index_menu.html /menu.html /main.html /index_right.html /~parksj/index.html /classes.html /~parksj/hp_up.htm
10	http://pllab.hyu.ac.kr/	/menu.htm /people.htm /~leehs /~leehs/menu.html /~leehs/text.html /~leehs/class/index.html /~leehs/class/c_index.html /~leehs/class/c/top.htm
11	http://islab.hanyang.ac.kr/	/logomenu.htm /home.htm /people.htm /sgkang.htm /~sgkang /~sgkang/head.html
12	http://aspen.hanyang.ac.kr/	/top.htm /right.htm /sogae.htm /ktosunim.htm
13	http://salab.hanyang.ac.kr/	/menu.htm /main.htm /graduate.htm

그림 15 각 도메인에 대하여 생성된 후보 패턴 (최소지지도= 0.2)

은 WebTree를 참조하여 웹 로그 데이타에 저장된 접근 링크 중에서 얼마나 많은 수의 링크를 무시할 수 있었나에 좌우된다고 볼 수 있다. 하지만 이러한 성능은 해당 사이트의 웹 로그 데이타의 크기와 접근 형태에 따라 달라질 수 있기 때문에 절대적인 수치로 나타내기가 어려우며 따라서 기존의 패턴 발견 알고리즘과의 비교 평가는 쉽지 않다. 다만 그림 14의 각 해당 사이트의 검색 링크수와 그림 15의 패턴의 수를 비교하면 어느 정도의 성능 평가가 가능하다. 즉, 검색 링크수에 비해 패턴의 수가 현저히 적은 이유가 WebTree에 의한 노이즈 제거로 그만큼 불필요한 접근 링크를 차단할 수 있었기 때문이라고 해석할 수 있다.

물론 이 성능은 최소지지도 값에 따라 달라질 수 있기 때문에 최소지지도 값을 얼마로 설정하는지가 중요할 수 있다. 그림 16은 최소지지도 변화에 따른 패턴의 변화를 나타낸 것이다. 여기서 패턴수는 13개의 연구실에서 구한 후보 패턴을 모두 합친 수이다. 최소 지지도가 0.05부터 0.25까지는 패턴수의 변화가 큰 것을 볼 수 있으며 그 이후는 완만한 변화를 나타낸다. 이 결과는 최소 지지도를 정하는데 도움을 줄 수 있다. 즉, 패턴수의 변화가 급격하게 변하다가 완만한 변화를 이루는 시점에서 최소 지지도를 선택하게 되면 추천 후보 패턴수를 줄이면서도 접근 빈도수가 높은 페이지에 대한 추천 가능성을 높여주는 효과를 갖게 된다.

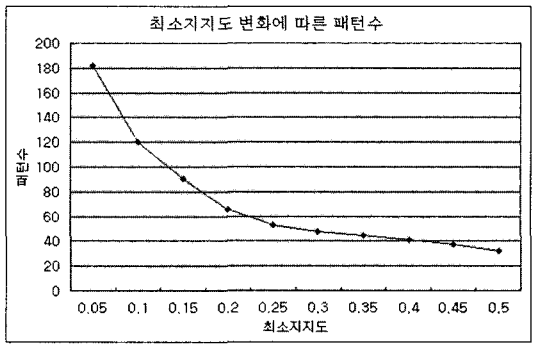


그림 16 최소 지지도 변화에 따른 패턴수

6. 결론 및 향후 연구 방향

본 논문은 웹 네비게이션 마이닝에서 웹 접근 로그 데이타 외에 웹 사이트 구조를 이용함으로써 효과적으로 사용자 탐색 패턴을 발견할 수 있는 SPMiner 시스템을 제안하였다. SPMiner는 로그 데이타의 전처리 과정을 줄여주고 온라인 상황에서 사용자들에게 능동적으로 흥미로운 패턴들을 추천하게 된다. 기존의 전처리 과정에서 로그 데이타의 잡음을 해결하는 복잡한 방법을 본 논문에서는 웹 사이트 구조와 빈도수 수정을 통하여 효율적이 되도록 하였다. 웹 사이트 구조를 생성하기 위하여 WebTree 생성 알고리즘을 만들었고, WebTree와

로그 데이타를 병합하는 WebTree+ 생성 알고리즘을 제안하였다.

현재 시스템의 문제점 및 향후 연구 방향은 다음과 같다. 첫째, 본 논문에서 사용한 html 페이지 이외의 javascript, JSP, ASP, PHP 등 다양한 형태의 웹 페이지에 대한 패턴 발견 및 추천이 가능한 시스템 개발에 대한 연구이다. 특히 html 이외의 문서들은 웹 브라우저에서 html 형태의 페이지로 변환되어 디스플레이 되지만 정확한 하이퍼링크를 찾기에는 그 구조가 복잡한 경우가 많다. 또한 html 구조라고 하더라도 논문에서 가정했던 트리의 모양이 아니라 일반적인 그래프 구조와 같이 복잡한 모양을 가질 수 있으며, 따라서 복잡한 구조를 가진 웹 사이트에 대한 정확한 WebTree 생성에 대한 연구가 필요할 것이다. 둘째, 현재 시스템에서는 온라인상에서 사용자에게 추천하는 과정에서 사용자의 탐색 정보를 받아 지지도를 재계산할 때, 매 트랜잭션마다 지지도를 수정해야 하였다. 이에 따라 지지도 재계산에 걸리는 수행 시간을 줄이기 위한 연구가 필요하다. 셋째, SPMiner에서 사용된 실험 데이타는 각 웹 사이트의 로그 파일 전체를 사용자 구분 없이 전처리한 것이기 때문에 후보로 추천된 패턴은 사용자에 관계없이 해당 페이지의 지지도에 의해 구해졌다. 추후에는 로그 파일의 데이타를 사용자별로 구분하여 각 문서의 지지도 이외에 사용자의 로그파일이나 다른 여러 정보를 함께 사용함으로써 각 개인 사용자에 대한 정확한 패턴 추천을 해주어야 할 것이다. 이것은 개인화된 추천 시스템에 적용하기 위해 꼭 필요한 기능이 된다. 넷째, 대형 포털 사이트와 같은 규모가 방대한 사이트의 웹 로그 데이타를 가지고 성능 평가를 함으로써 작은 규모의 데이타에서 찾을 수 없었던 문제점을 파악하고 새로운 기법을 제시할 필요가 있다.

참 고 문 헌

[1] R. Agrawal, T. Imielinski, A. Swami, "Database mining: A performance perspective," *IEEE Trans. on Knowledge and Data Engineering*, Special issue on Learning and Discovery in Knowledge-Based Databases, vol.9, no.6, pp. 914-925, 1993.

[2] O. Etzioni, "The World Wide Web: Quagmire or gold mine," *Comm. of the ACM*, vol.39, no.11, pp. 65-68, 1996.

[3] S. Madria, S. Bhowmick, W. Ng, E. Lim, "Research issues in Web data mining," Proc. 1st Int. Conf. on Data Warehousing and Knowledge Discovery, pp. 303-312, 1999.

[4] M. Spiliopoulou, "Data mining for the Web," Proc. 3rd European Conf. on Principles of Data Mining and Knowledge Discovery, pp. 588-589, 1999.

[5] R. Kosala, H. Blockeel, "Web mining research: A survey," *ACM SIGKDD Explorations*, vol. 2, issue 1, pp. 1-15, 2000.

[6] M. Spiliopoulou. "Web usage mining for Web site evaluation," *Comm. of the ACM*, vol.43, no.8, pp. 127-134, 2000.

[7] A. Nanopoulos, Y. Manolopoulos, "Finding generalized path patterns for Web log data mining," *Lecture Notes in Computer Science*, vol. 1884, pp. 215-228, 2000.

[8] J. Srivastava, R. Cooley, M. Deshpande, P. Tan, "Web usage mining: Discovery and applications of usage patterns from Web data," *ACM SIGKDD Explorations*, vol. 1, issue 2, pp. 12-23, 2000.

[9] J. Borges, M. Levene, "A fine grained heuristic to capture Web navigation patterns," *ACM SIGKDD Explorations*, vol. 2, issue 1, pp. 40-50, 2000.

[10] R. Agrawal, A. Srikant, "Fast algorithms for mining association rules," Proc. VLDB'94, pp. 487-499, 1994.

[11] J. Han, J. Pei, Y. Yin, "Mining frequent patterns without candidate generation," Proc. ACM SIGMOD, pp. 1-12, 2000.

[12] A. Srikant, R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," Proc. 5th Int. Conf. on Extending Database Technology, 1996.

[13] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, M. Hsu. "PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth," Proc. Int. Conf. on Data Engineering, pp. 215-224, 2001.

[14] M. Zaki, "SPADE: An efficient algorithm for mining frequent sequences," *Machine Learning*, vol.42, no. 1-2, 2001.

[15] M. Chen, J. Park, P. Yu, "Data mining for path traversal patterns in a Web environment," Proc. 16th Int. Conf. on Distributed Computing Systems, pp. 385-392, 1996.

구 흠 모

2001년 한양대학교 전자컴퓨터공학부 졸업(학사). 2003년 한양대학교 대학원 컴퓨터공학과 졸업(석사). 2003년~2004년 (주)에이치씨아이. 2005년~현재 LG전자 정보통신사업본부 단말연구소. 관심분야는 Data Mining, SyncML(DS, DM)

최 중 민

1984년 서울대학교 컴퓨터공학과 졸업(학사). 1986년 서울대학교 대학원 컴퓨터공학과 졸업(석사). 1993년 뉴욕주립대(Buffalo) 전산학과 졸업(박사). 1993년~1995년 전자통신연구원(ETRI) 선임연구원. 1995년~현재 한양대학교 컴퓨터공학과 교수 관심분야는 인공지능, 에이전트, 웹 정보처리, 시맨틱 웹