Efficiently Mining Tree Traversal Patterns in a Web Environment

Xuemin Lin & Xiaomei Zhou
Computer Science and Engineering
The University of New South Wales
Sydney, NSW 2052, Australia
{lxue, xmei}@cse.unsw.edu.au

Yuh-Chi Lin
Computer Science
The University of Western Australia
Perth, WA 6907, Australia
yuh@cs.uwa.edu.au

Abstract

In this paper, we firstly propose a novel data mining problem in a WWW environment. Secondly, we outline the algorithm for solving the problem. Our initial implementation of the algorithm suggests the algorithm is very efficient and scalable in practical. Therefore it can be applied to a very large database.

Keywords: Data Mining, Access Patterns Discovery, Algorithms, WWW.

1 Introduction

In the last several years, the internet and the World Wide Web (WWW) have grown exponentially so that huge amount of information is now available in the internet for users to access. Managing and organizing internet resources, therefore, becomes critical. Many internet search engines have been prototyped. To make internet resources search more efficiently, mining user access patterns in a web environment emerges as one of the hot topics in the research area of database applications.

Currently, the research in computing user patterns follows two directions, 1) a *relational database* oriented approach [4, 5], and 2) a *topological* oriented [3] approach. In a relational database oriented approach, we view the input records of user access patterns as a table in a relational database, and output the discovered *association* rules by applying the algorithms in [1, 8]. In a topological oriented approach, we view a web environment as the one with linked documents - a directed graph, and view a user access to the web environment as a *walk* [2] in the directed graph along arcs. Then, we are interested in discovering walk patterns with a specific topology which are frequently used by users.

In this paper, we shall restrict ourself to a topological oriented approach. To the best of our knowledge, the paper [3] is the first topoligical oriented approach, where an efficient algorithm has been proposed to compute frequently used *path traversal* patterns. In this paper, we shall investigate a more general problem - how to compute frequently used *tree traversal patterns*. As an outcome, a novel and efficient algorithm will be proposed for this purpose. Because of the space limitation, we are only able to outline the algorithm; the interested readers please refer our full paper [6] for details.

The rest of the paper will be organized as follows. In the second section, we specify the problem and its applications. The third section outlines our algorithm. This is followed by the conclusions and our preliminary implementation results.

2 Preliminary

A user may access the internet as many times as he (she) likes, and each time may access as many web sites as he (she) wants. The detailed records of web access from the users in a local system may be recorded in a system log. These records will enable the local system to learn the web access patterns from its local users and then build up a good system for local users to access the internet [7]. This is one of the applications of our research in this paper.

An access to the web from a user may be retrived from the system log in form of $< UID, (s_1, d_1), (s_2, d_2), ...(s_n, d_n) >$, where domain UID gives the user ID, (s_i, d_i) indicates that the user travels from document s_i to document d_i, and $d_i = s_{i+1}$. By recording the starting point of an access as null, we should be able to identify different accesses from the same user, and different accesses from different users.

To illustrate a partial ordering in each user access, a user access can be converted into a *traversal* tree as follows. Gradually, add a new vertex d_i into the tree together with the link (s_i, d_i). For example, with respect to the access
$< (null, A), (A, B), (B, C), (C, B), (B, A),$
$(A, D), (D, E) >$,

0-7803-4886-9/98/ $10.00 © 1998 IEEE

the corresponding traversal tree is illustrated in Figure 1. A traversal tree is a *rooted tree* [2], that is, tree with notion of parents and children. In this paper, we are interested only in rooted trees; for terminology simplicity, a rooted tree is abbreviated into "tree"

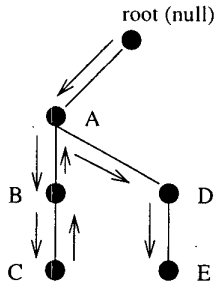

Figure 1: A Traversal Tree

Now, we can assume that a database (table) D of user access patterns has been obtained such that each row (tuple) is a traversal tree with a domain (attribute) TID to identify it. The problem of mining frequent traversal trees (MFTT) can be described as follows. Given a minimum support $s\%$, find the set T of all trees such that:

- for each tree $t \in T$, there are at least $s\%$ of tuples in D which contain t, and
- no two trees in T which has the inclusion relationship.

3 The Algorithm for Solving MFTT

A naive way to solve MFTT is to firstly find out all vertices of the trees in the database; secondly enumerate all subtrees generated from these vertices; and thirdly for each subtree t, count how many tuples in D contain t. Clearly, in practice this approach is too expensive to be applicable, as the number of generated subtrees is larger than $n!$ where n is the number of vertices involved and usually is very large. Inspired by the results in [1, 3, 8], in this paper we will propose an efficient algorithm for solving MFTT. Regarding the space limitation, we are only able to outline our algorithm; the details may be found in the full paper [6].

We need another terminology to describe our algorithm. Given a database D of user access patterns and a minimum support $s\%$, we say that a tree t is *frequent* with respect to $s\%$ and D if there are at least $s\%$ tuples from D containing t.

The algorithm follows a data mining framework in [1, 3]. We shall iteratively expand the size of the set of frequent trees. As MFTT is more complicated than the problems in [1, 3], new techniques are developed to make the iterations efficient. The algorithm consists of the following three steps.

Step 1: Use the approach in [3] to calculate the frequent *paths*, and store them in L_1. Goto Step 2.

Step 2: Iteratively generate the set of frequent trees with k *branches*, starting the iteration from $k = 2$ as follows. (Note: a path can be regarded as a frequent tree with 1 branch). Let L_k store the frequent trees with k branches, and C_k store the candidates of frequent trees with k branches. For $k \geq 2$:

- We first generate C_k from L_{k-1} using a similar operation to the join operation in relational databases based on the fact that if a tree t in L_k, then each subtree, with $k-1$ branches, of t should be in L_{k-1}. This will prevent us from generating too many candidates in C_k. The detailed description of this new join operation may be found in the full paper [6]. An illustrative example is depicted in Figure 2.
- After obtaining C_k, we organize the candidates in C_k in a hash-like tree H_k in order to speed up our test of whether a candidate in C_k is in L_k. Our new hash technique is detailed in [6].
- Scan the database D. For each tuple $d \in D$, enumerate all k-branched subtrees of d, hash each subtree into the hash-like tree H_k, and then add 1 into the count of a candidate in C_k if the candidate is contained by a subtree of d. Note that in order to reduce the chances of generating unnecessary subtrees of d, we suggested in [6] that 1) the enumeration should be carried out in the same time while doing hash, and 2) only *maximal* k-branched subtrees will be generated. For example, suppose the join result tree in Figure 2 is also a tuple in D. When in the iteration $k = 4$, instead of generating two subtrees as depicted in Figure 3(a) and 3(b), we generate only one subtree as depicted in Figure 3(a) to do counting for the candidates in C_k. An efficient counting technique is also presented in [6].
- Store the candidates, with count greater than $s\% \times |D|$, of C_k into L_k; and continue the next iterations.

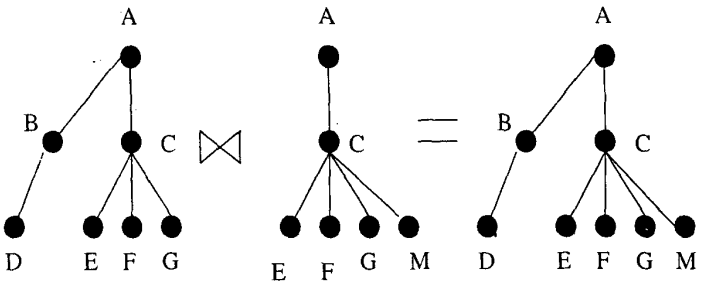

Figure 2: Generate Candidates Using a Join-Like Operation

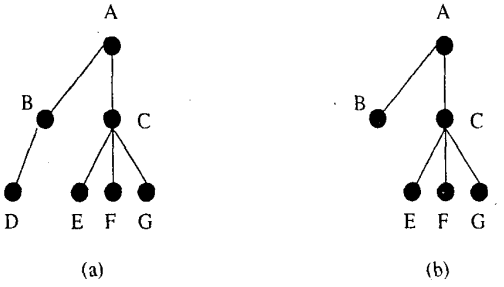

Figure 3: Enumeration

The iteration will be terminated if either no candidates are generated or no more frequent trees are generated.

Step 3: Remove the frequent trees which are contained in another frequent tree.

The proof of the algorithm correctness may be found in the full paper [6].

4 Conclusions

In this paper, we specified a new data mining problem for enhancing web organization. We also provided an efficient algorithm to solve this problem. Our preliminary implementation suggests that this algorithm runs fast in practice. We initially implement the algorithm on a pentum-pro workstaion with main memory size 128 MB running the Windows NT server 4.0 System and using Javaview. For a database where there are 200,000 tuples on average, each tuple is a tree with 20 vertices on average, and totally there are more than 1000 vertices, the algorithm takes just a few minutes to complete.

References

[1] R. Agrawal and R. Srikant, Fast Algorithm for Mining Association Rules in Large Database, *Proceedings of the 20th International Conference on Very Large Data Bases*, 478-499, 1994.

[2] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, Macmillan, 1977.

[3] M.-S. Chen, J. S. Park, and P. S. Yu, Data Mining for Path Traversal Patterns in a Web Environment, *Proceedings of the 16th International Conference on Distributed Computing Systems*, 385-392, 1996.

[4] D. W. Cheung, B. Kao, and J. Lee, Discovering User Access Patterns on the World-Wide Web, *1st Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD-97)*, 1997.

[5] B. Mobasher, N. Jain, E.-H. Han, and J. Srivastava, Web Mining, Pattern Discovery from World Wide Web Transactions, *Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'97)*, 1997.

[6] X. Lin and X. Zhou, Efficiently Mining Tree Traversal Patterns in a Web Environment, Manuscript, 1998.

[7] D. Ngu and X. Wu, SiteHelper: A Localized Agent that Helps Incremental Exploration of the World Wide Web, *6th International WWW Conference*, 1996.

[8] J. S. Park, M.-S. Chen, and P. S. Yu, An effective Hash Based Algorithm for Mining Association Rules, *ACM SIGMOD*, 175-186, 1995.