基于改进的Wap算法的Web序列模式的研究

王 慧 张骏温
(北京交通大学计算机与信息技术学院 北京 100044)

摘 要 序列模式挖掘是Web日志挖掘中的一个重要范畴。针对Wap算法中递归构建大量条件树的这一缺陷,提出了一种改进算法NGCWAP。NGCWAP算法采用前序遍历号和后序遍历号来跟踪频繁序列分布在哪些后缀树集中,避免了条件树的构建,从而减少了内存消耗。通过实验验证了改进算法的正确性和高效性。

关键词 数据挖掘,Web日志挖掘,Wap算法,频繁序列
中图法分类号 TP313.13 **文献标识码** A

Research on the Web Sequence Pattern Based on the Improved Wap Algorithm

WANG Hui ZHANG Jun-wen
(Department of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China)

Abstract Sequential pattern mining is an important mining area of the Web log mining. Wap algorithm must construct a large number of conditional trees during mining, so this paper proposed an improved algorithm which name is NGCWAP to solve that problem. The NGCWAP algorithm uses the pre-order traversal number and post-order traversal number to trace the sub-trees in which candidates are located, which avoids construction of the conditional tree, thus the algorithm reduces memory consumption. The experiment results show accuracy and efficiency of the improved algorithm.

Keywords Data mining, Web log mining, Wap algorithm, Frequent sequences

1 引言

Web中有价值的信息不仅仅存在于网页的内容描述和网页之间的链接结构中,用户在浏览站点的过程中也显示了一些有用的模式。这些模式反映了用户的兴趣爱好,而Web日志记录着用户每次的浏览行为,如果采用一些挖掘技术对Web日志进行挖掘,就可以发现这些有用的模式,最后将这些模式用于Web站点的管理中。

序列模式挖掘是Web日志挖掘中的一个重要研究课题。它主要研究"一些项跟随另一项发生"的规律,其不仅考虑了项之间的关联,还将项之间的次序关系纳入了研究范围。目前序列模式挖掘算法主要分为两类,第一类是以类Apriori算法[1,2]为代表的有候选集产生的算法,第二类是以Wap算法[2]为代表的无候选集产生的算法。第一类算法在挖掘过程中需要产生大量的候选集,统计每个候选集的支持度需要多次扫描数据库,因此效率比较低。第二类算法采用树结构来存储访问序列,相比第一类算法效率有所提高。其中,Wap算法在挖掘过程中需要构建大量的条件树,存储这些条件树需要占用大量内存,所以效率还有待提高。

因此,本文在深入研究Wap算法后,提出了一种改进算法——NGCWAP算法,并通过实验验证了算法的效率。

2 相关概念和定义

定义1(频繁访问模式)[3] 如果访问序列S的支持度大于等于给定的最小支持度阈值,记为$Support(S)\geqslant min_sup$,则序列$S$在WASD是频繁的。设$S$的长度为$L$,则$S$称为$L$阶频繁序列。

定义2(序列前缀,后缀)[3] 两个序列$S=\langle e_1e_2e_3\cdots e_n\rangle$和$A=\langle e_1'e_2'e_3'\cdots e_m'\rangle$,其中$m\leqslant n$,如果对所有的$i\leqslant m$,都有$e_i'=e_i$,则$A$为$S$的一个前缀,$B=S-A=\langle e_{m+1}e_{m+2}\cdots e_n\rangle$为$S$以$A$为前缀的后缀。

3 基于Wap改进算法

NGCWAP(Not Generate Conditional Web Access Pattern Tree)算法借鉴了Wap算法中采用树来存储序列的这一思想,但改变了挖掘方向,Wap算法优先发现频繁的后缀序列。但在NGCWAP算法中优先寻找频繁的前缀序列。例如,对于频繁序列P1→P2→P3→P4,Wap算法发现频繁序列的顺序为P4,P3→P4,P2→P3→P4,P1→P2→P3→P4,而在NGCWAP算法中挖掘顺序为P1,P1→P2,P1→P2→P3,P1→P2→P3→P4。

NGCWAP算法的大体思想是:首先发现频繁m阶前缀序列,然后在m阶前缀序列的末尾事件的后缀子树中,找到频繁事件e_i,将e_i附加到m阶前缀序列的后面,构成频繁$m+1$阶前缀序列。对于一个频繁序列,序列的前部分位于树的上部分,余下的序列只能在树的下部分找到,因此,只要知道序列的下一个频繁后缀事件分布在哪部分子树中,就没必

到稿日期:2011-03-21 返修日期:2011-05-15 本文受核高基重大专项项目(2009ZX01045-005-001)资助。
王 慧(1987-),女,硕士,主要研究方向为计算机应用,E-mail:wh1987519@163.com;张骏温(1966-),男,博士,副研究员,硕士生导师,主要研究方向为计算机应用、信息安全、信息系统工程监理。

要去再次构建条件树，但主要问题是怎样确定给定事件的后缀树。

文献[4]通过位置码来解决上述问题，每个节点的位置码是在父节点的位置码后添加 1，兄弟节点则在最近左兄弟节点的位置码上附上 0，因此，当树比较大时，该位置码会呈指数增长。文献[5]以宽度优先的方式进行编码，算法中假定每个祖先节点的孩子节点的个数小于 100，实际情况孩子的节点个数很有可能大于 100，因此，本算法通过前序遍历号 pre_index 和后序遍历号 aft_index 来确定任意两个节点的子孙关系，进而找到一个特定事件的后缀树，避免了条件树的构建。

NGCWAP 算法包括构建 NGCWAP_tree 和 NGCWAP_mine 两部分。第一步构建 NGCWAP_tree，来存储 WASD 中的访问序列，然后在该 NGCWAP_tree 上采用 NGCWAP_mine 挖掘出所有的频繁序列。

3.1 构建 NGCWAP_tree

NGCWAP_tree 中每个节点包含的属性为 label，count，pre_index，aft_index，anc_index，des_count，每个标识的具体含义如表 1 所列。

表 1 每个节点包含的属性

字段	含义
label	节点标识
count	节点支持度计数
pre_index	前序遍历中的次序
aft_index	后序遍历中的次序
anc_index	序列中最邻近祖先节点的序号
des_count	节点的子孙节点的个数

由前面可知，前序遍历号和后序遍历号是用来识别子孙关系，des_count 是为了减少 NGCWAP_mine 算法中节点之间的比较次数，anc_index 用来辅助统计最近祖先节点的子孙节点个数。构建 NGCWAP_tree 的过程与构建 Wap_tree 的过程类似，只是多了两次遍历，其构建流程图如图 1 所示。

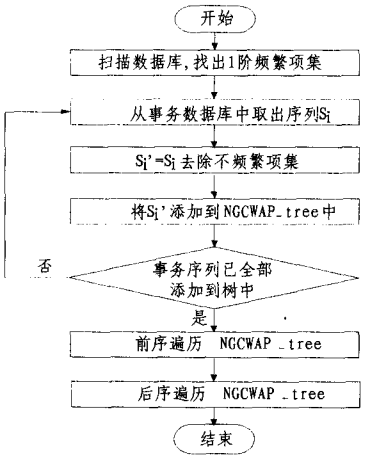

图 1 构建 NGCWAP_tree 流程图

构建算法描述如下：

算法 1 构建 NGCWAP_tree

输入：Web 访问序列数据库(WASD)，最小支持度 Min_Sup，1 阶频繁项集集合 L

输出：NGCWAP_tree

中间变量：

current_pointer：指向当前节点的指针，初始指向根节点。

nodes_count：节点计数，初始值为 0。

Map ancMap：key 存放 1 阶频繁项集，value 用来记录每个节点的最近祖先节点号，初始全为 0。

开始：

(1) 建立根节点 T，所有属性项设置为 0；

(2) 从 WASD 中依次取出访问序列 S，进行如下操作：

(2.1) 从 S 中提取频繁子序列 $S'=S1S2\cdots Sn$；

将 S 中不属于 L 集合的事件去除，将 current-node 指向 T 的根的节点；

清空 ancMap，设为初始值 0；

(2.2) For $k=1$ to n，n 是序列 S'的长度，进行操作：

(2.2.1)

IF current-node 孩子节点中有标识为 Si 的节点，则

Si. count++；

ancMap. set("Si", Si. aft_index)；

将 current-node 指向 Si 节点；

ELSE 新建一个标识为 Si 的节点，计数为 1；

//节点的最近祖先号暂时存储为最近祖先的插入序号，用 aft_index 临时存储节点的插入序号

nodes_count++；

Si. aft_index=nodes_count；

Si. anc_index=ancMap. get("Si")；

ancMap. set("Si", Si. aft_index)；

将 current-node 指向新节点；

(3) 前序遍历已经建立好的树，从 T 的 root 开始，遍历顺序为：根，左子树，右子树；

先将 nodes_count 置 0；

每遍历一个节点

nodes_count++；

pre_index=nodes_count；

将 Si. anc_index 由原来最近祖先的插入号改为最近祖先的先序号；

将当前节点加入到相同标识的链表中；

直到遍历所有节点

(4) 后序遍历已经建好的树，遍历顺序为：左子树，右子树，根；

先将 nodes_count 置 0；

每遍历到一个节点 Si

nodes_count++；

Si. aft_index=nodes_count；

//更新最近祖先节点的孩子节点个数

Si. anc_index. des_count+=Si. des_count+1；

直到遍历所节点

(5) 返回建立好的 NGCWAP_tree。

例如，给定样本数据库，如表 2 所列。经过构建算法构建后的 NGCWAP_tree 如图 2 所示。

表 2 样本事务数据库

事务 Id	用户访问序列
1	P1→P2→P4→P1→P3
2	P5→P1→P5→P2→P3→P1→P3
3	P2→P1→P2→P6→P1→P5→P3
4	P1→P6→P2→P1→P3→P6→P3

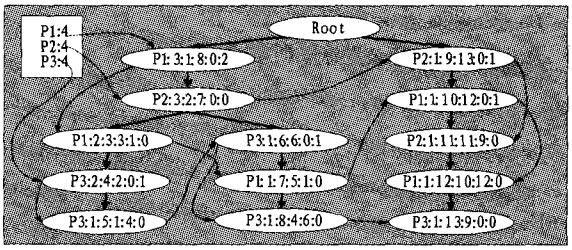

图 2 构建后的 NGCWAP_tree

3.2 采用 NGCWAP_Mine 算法进行挖掘

构建完 NGCWAP_tree，其后续的工作只需在该树上进行，不再依赖原始的序列数据库。在 Wap_mine 算法中优先发现拥有相同后缀的频繁模式，挖掘方向是从后向前，而 NGCWAP_mine 算法中改变了挖掘方向，即首先去发现拥有相同前缀的频繁模式，然后在模式后面加上频繁项集来发现所有的频繁序列，在介绍算法前先引入以下定义和性质：

定义 3(第一节点) 从树的根到该节点之间不存在与该节点具有相同标识的节点，称为第一节点。

定义 4(第一节点集) 已知树 T，e 在树 T 中的第一节点集是指树 T 中标识为 e 且为第一节点的节点组成的集合。

定义 5(后缀树集) 已知序列 S，S 的后缀树集是指以 S 中最后一个事件的孩子节点为根的子树组成的集合。

性质 1 在序列数据库中，已知频繁序列 S，e 是 S 后缀树集中的事件，如果 e 为频繁事件，则序列 Se 为频繁序列。

性质 2 节点 e 在当前正在挖掘的后缀树集中的支持度等于 e 在后缀树集中的第一节点集的支持度之和。

性质 3 设 α 为节点在前序遍历中的序号，β 为节点在后序遍历中的序号，对于任意两个结点 $P1[\alpha1,\beta1]$，$P2[\alpha2,\beta2]$，根据 $P1$，$P2$ 的前序遍历号和后序遍历号，可以判断 $P1$ 与 $P2$ 的祖孙关系。

NGCWAP_mine 挖掘过程为：

首先，根据建树过程中挖掘出的频繁事件，找到 1 阶频繁序列。

其次，对每个频繁事件和当前要挖掘的后缀树集，根据该事件的链接队列，找到在后缀树集中的第一节点集，计算该事件在当前后缀树集中的支持度，如果大于最小支持度，则将该事件附加到最新频繁序列表中，并且将第一节点集的孩子节点作为当前后缀树集，挖掘下一个频繁事件。

算法的伪代码描述如下。

算法 2 NGCWAP_Mine(T,L,Min_Sup,F,R)

输入：NGCWAP_tree T，链接头表 L，最小支持度 Min_Sup，m 阶频繁序列 F，后缀子树的根集 R(R 初始为 T 的根 root，F 初始为空集)

输出：$m+1$ 阶频繁序列 F'

中间变量：C：ei 在后缀树集中的支持度

算法开始：

(1)如果 R 是空集，返回；

(2)对 L 链接头表中的每个事件 ei，在 T 中寻找 ei 的后缀树集；

(2.1) 对 ei-queue 中每个事件 a

If Is_Descendants(a,R)

$C=C+a$.event_count；

将 a 节点的孩子节点放入 R' 中；

int N=节点 a 的子孙个数 des_count；

跳过 ei-queue 中事件 a 后 N 个事件；

(2.2) 如果 C 大于等于 Min_Sup

将 a 附加到 F 的后面，添加到 F' 中，输出 F'；

递归调用算法 NCWAP_Mine(T,L,Min_Sup,F',R')；

算法：Is_ Descendants(ei,R)

输入：待判断节点 ei，根节点集合 R(可为单个节点)

输出：true 或 false

开始：For each r in R

IF ei.pre_index>r.pre_index && ei.aft_index<r.aft_index

Return true；

Return false；

举例，对图 2 中的 NGCWAP_tree 进行挖掘，以头表中的第一个频繁事件 $P1$ 为例，最小支持度设为 3，最初后缀树集 R={Root}。

首先，在以 R 为根的后缀树集中查找 $P1$ 的第一节点集 P={($P1$:3:1:8:0:2),($P1$:1:10:12:0:1)}，R'={($P2$:3:2:7:0:0),($P2$:1:11:11:9:0)}，如图 3 所示。具体查找过程为：跟踪头表中 $P1$ 的链接找到 $P1$-队列，将队列中的所有节点与 R 中的节点进行比较，如果发现是 R 的子孙节点，则加入到 P 中。首先找到符合条件的节点为($P1$:3:1:8:0:2)，将该节点加入到 P 中，观察其子孙节点个数为 2，可知 $P1$ 队列的后两个节点为其子孙节点，可跳过比较，随后找到符合条件的节点($P1$:1:10:12:0:1)，将其加入到 P 中，最后得到 P={($P1$:3:1:8:0:2),($P1$:1:10:12:0:1)}，支持度计数为 4>3，故将 $P1$ 附加到 F 序列后面，F={$P1$}，并添加到 F'={$P1$}。

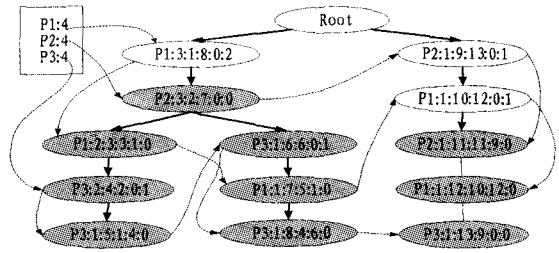

图 3 P1 的后缀树集

然后，在以 R' 为根的后缀树集上递归地调用算法，来发现所有以 $P1$ 为前缀的频繁序列 F'={$P1$, $P1\rightarrow P1$, $P1\rightarrow P1\rightarrow P3$, $P1\rightarrow P2$, $P1\rightarrow P2\rightarrow P1$, $P1\rightarrow P2\rightarrow P1\rightarrow P3$, $P1\rightarrow P2\rightarrow P3$, $P1\rightarrow P3$}。

NGCWAP 算法中不再构建大量的条件树，仅仅存储构建好的 NGCWAP_tree，这样大大地减少了内存空间的耗费，而且减少了因构建条件树所花费的时间，但 NGCWAP 算法却增加了前序遍历和后序遍历，时间上的优势可能不是很大。因此，针对 NGCWAP 算法的性能，本文在下节进行了详细的实验验证。

4 实验结果及分析

实验环境：英特尔 2.53GHz Cpu，2GB 内存，Windows Xp 操作系统，Java 开发语言，MyEclipse 6.0 开发工具。实验测试数据采用 IBM 数据生成器生成的标准数据集，该生成器在序列模式挖掘研究中已经得到广泛使用。

本论文分别从算法准确性和拓展性等方面进行比较。

4.1 准确性

表 3 3 个算法挖掘出的频繁路径

事务数据库	频繁路径(长度大于 2)			
	Wap		NGCWAP	
	频繁项集	支持度	频繁项集	支持度
	P2-P1	5	P1-P1	5
	P1-P2-P1	4	P1-P1-P3	4
	P1-P1	5	P1-P2	4
	P1-P2	4	P1-P2-P1	4
	P1-P5	3	P1-P2-P1-P3	4
P1-P2-P5-P1-P3	P2-P5	3	P1-P2-P3	4
P5-P1-P4-P2-P3-P1-P3	P1-P3	4	P1-P5	3
P2-P1-P2-P6-P1-P5-P3	P2-P1-P3	4	P1-P3	4
P1-P6-P2-P1-P3-P6-P3	P1-P2-P1-P3	4	P2-P1	5
P2-P1-P4-P1-P5	P1-P1-P3	4	P2-P1-P3	4
	P2-P3	4	P2-P5	3
	P1-P2-P3	4	P2-P3	4
	P5-P3	3	P5-P3	3

(下转第 239 页)

sing nearest neighbor condensation [J]. IEEE Transactions on Neural Networks, 2010, 2(1): 351-357

[11] Ferrandiz S, Boulle M. Bayesian instance selection for the nearest neighbor rule [J]. Machine Learning, 2010, 81(3): 229-256

[12] Marchiori E. Class conditional nearest neighbor for large margin instance selection [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2010, 32(2): 364-370

[13] Nikolaidis K, Goulermas J Y, Wu Q H. A class boundary preserving algorithm for data condensation [J]. Pattern Recognition, 2011, 44(3): 704-715

[14] Barandela R, Ferri F J, Sanchez J S. Decision boundary preserving prototype selection for nearest neighbor classification [J]. International Journal of Pattern Recognition and Artificial Intelligence, 2005, 19(6): 787-806

[15] Pawlak Z. Rough sets [J]. International Journal of Information and Computer Sciences, 1982, 11: 341-356

[16] 苗夺谦，李道国. 粗糙集理论、算法与应用[M]. 北京：清华大学出版社，2008：34-66

[17] Parthalain N M, Shen Q. Exploring the boundary region of tolerance rough sets for feature selection [J]. Pattern Recognition, 2009, 42(5): 655-667

[18] Blake C L, Merz C J. UCI Repository of machine learning databases[EB/OL]. http://www.ics.uci.edu/~mlearn/MLRepository.html, 1996

(上接第 208 页)

人工生成少量的数据，设定支持度为 3，分别执行算法，其挖掘结果如表 3 所列。

由表可知，两个算法挖掘出的频繁序列是相同的，验证了 NGCWAP 算法的挖掘结果是准确的，因而算法是可行的。

4.2 同一数据集不同支持度下算法执行的时间变化和内存变化

采用生成器生成 5 万行的测试用例，分别设置不同的最小支持度，观察两个算法执行时间和内存占用情况，结果如图 4 和图 5 所示。由图 4 可知，对于同一数据集，随着最小支持度的增大，两个算法的执行时间都随之降低。这主要是因为当最小支持度增大时，满足条件的频繁序列减少，算法将花费较少的时间去挖掘频繁序列。Wap 和 NGCWap 算法抛弃了“候选-测试”的方式，直接在树上挖掘频繁序列，NGCWAP 算法虽然避免了条件树的构建，但在构建 NGCWAP_tree 过程中，增加了两次遍历，故时间跟 Wap 算法相差不大。

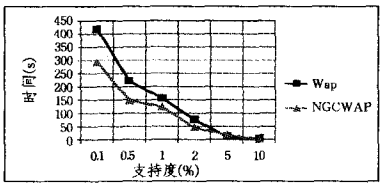

图 4 算法在不同支持度下的执行时间曲线图

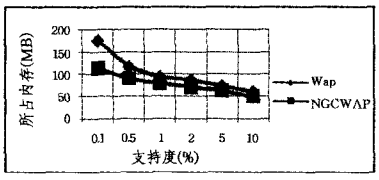

图 5 算法在不同支持度下的内存消耗曲线图

由图 5 可以看出，NGCWAP 算法在内存消耗上优于 Wap 算法，尤其当支持度比较小时更明显，这是因为 Wap 算法在挖掘的过程中需要保存大量的物理条件树，导致该算法占用了大量的内存。

4.3 同一支持度不同数据集下算法执行的内存变化

采用数据生成器分别生成 0.1 万，1 万，2 万，4 万，8 万，10 万行 6 组测试数据，最小支持度设为 3%，Wap 算法和 NGCWAP 算法的内存变化如图 6 所示。由图 6 可知，随着事务量的增大，两个算法的内存消耗呈线性增长。这主要是因为数据量增加，导致了树的规模增大，但 NGCWAP 算法的内存消耗总体小于 Wap 算法，尤其当数据量比较大时，更能体现 NGCWAP 算法的优势。

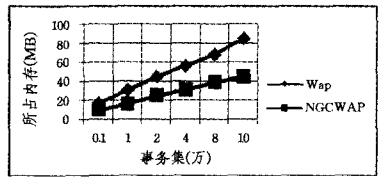

图 6 算法在不同数据集下的内存消耗曲线图

由本节实验可知，NGCWAP 算法相对于 Wap 算法，优势主要集中在内存消耗方面，其执行时间相差不是很大。总体来看，NGCWAP 算法的性能优于 Wap 算法。

结束语 在 Wap 算法的基础上提出了一种改进算法 NGCWAP，该算法改变了序列模式挖掘的方向，采用前序遍历号和后序遍历号优先发现频繁序列的前缀，能够有效避免条件树的构建。实验表明，NGCWAP 算法在内存消耗上优于 Wap 算法，尤其当数据量比较大时更明显。NGCWAP 算法虽然在挖掘过程中增加了两次遍历，但最后花费的时间与 Wap 算法相差不是很大。总体来看，改进后的算法在序列模式挖掘中是高效的。

参 考 文 献

[1] Agrawal R, Srikant R. Mining Sequential Patterns [C]// Proceedings of 11th International Conference on Data Engineering. 1995

[2] Agrawal R, Srikant R. Fast algorithms for mining association rules in large databases[C]// Proceedings of the 20th International Conference on Very Large Databases. Santiago, Chile, 1994: 487-499

[3] Pei Jian, Han Jia-wei. Mining Access Patterns Efficiently from Web Logs[C]//PADKK '00 Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining. 2000

[4] 李朋轩. 基于 Web 挖掘的电子商务推荐技术的研究[D]. 哈尔滨：哈尔滨工程大学，2009

[5] Ezeife C I, Lu Yi. Mining Web Log Sequential Patterns with Position Coded Pre-Order Linked WAP-Tree [J]. Journal Data Mining and Knowledge Discovery, 2005, 10(1): 6-36

[6] Liu Li-zhi, Liu Jun. Mining web log sequential patterns with layer coded breadth-first linked WAP-tree[C]// Proceedings 2009 ISECS International Colloquium on Computing. Communication, Control, and Management. 2009