第 34 卷 第 4 期 计 算 机 工 程 2008 年 2 月
Vol.34 No.4 Computer Engineering February 2008
·软件技术与数据库· 文章编号：1000—3428(2008)04—0060—03 文献标识码：A 中图分类号：TP311.12

# Web 日志挖掘中 GITC 算法的改进

郭 维
（安徽理工大学计算机科学与技术系，淮南 232001）

**摘 要：**GITC 算法和 Tree-DM 算法都是基于交集关系的挖掘算法。文章分析这 2 个算法的性能特点，提出一种 GITC 算法的改进算法：GI 算法。该算法利用适当的数据结构来保存支持数信息，省去了扫描原数据库来统计支持数耗费的大量时间，并解决了 Tree-DM 算法在二次求交、冗余求交等方面存在的问题。经过实验验证，较 GITC 算法而言，GI 算法可以更高效地挖掘用户频繁访问模式。
**关键词：**Web 日志挖掘；频繁访问模式；交集关系

# Improvement of GITC Algorithm on Web Log Mining

GUO Wei
(Dept. of Computer Science and Technology, Anhui University of Science and Technology, Huainan 232001)

**【Abstract】**The GITC algorithm and the Tree-DM algorithm are both based on the intersection relation. The paper analyzes the performance of both algorithms deeply, and puts forward an improved algorithm named GI. It stores the information of support number in appropriate data structure so as to spare a mass of time of getting the support number of each candidate by scanning the original database. It also solves the problem of getting the intersections repeatedly and redundantly in the Tree-DM algorithm. Experimental results show that the GI algorithm can discover user frequent access patterns more effectively than GITC.
**【Key words】** Web log mining; frequent access pattern; intersection relation

Web日志挖掘中用户频繁访问模式的发现，主要是将适当的挖掘算法(例如：类Apriori算法)应用于经过数据预处理后得到的最大前向引用路径集。当前，如何降低挖掘算法的时间复杂度和空间复杂度已经成为Web日志挖掘研究的重点之一。基于交集关系的原理[1]，文献[2]中提出一种以求交作为基本算元的用于Web日志挖掘的GITC算法。文献[3]中同样提出一种基于交集关系的挖掘关联规则的Tree-DM算法。本文结合Tree-DM算法的优点，对GITC算法进行改进，得到GI算法。

## 1 相关问题的描述

**定义 1** 已知 2 个用户访问模式分别为$X=\{X_1,X_2,\cdots,X_m\}$和$Y=\{Y_1,Y_2,\cdots,Y_n\}(m\leq n)$，若$X_1=Y_k,X_2=Y_{k+1},\cdots,X_m=Y_{k+m-1}(1\leq k\leq m,k+m-1\leq n)$，则称$X$是$Y$的子访问模式(即$Y$支持$X$)。记为$X\subset^s Y$，其中$\subset^s$右上角的$s$是代表序列(sequence)的意思。

**定义 2** 已知访问模式 $X$ 的支持数为 $t$，用户访问模式数据库 $T$ 容量为 $q$，所以 $X$ 的支持度 $sup(X)=(t/q)\times 100\%$。

如果 $sup(X)\geq min\_sup$($min\_sup$ 是最小支持度阈值)，那么 $X$ 就是频繁访问模式。

**定义 3** 在用户访问模式集$T=\{T_1,T_2,\cdots,T_m\}$中，其中$T_i(1\leq i\leq m)$为$T$中的一个用户访问模式，如果$T_k\not\subset^s T_i(1\leq i\leq m,\ i\neq k)$，那么$T_k$就是用户访问模式集T中的一个最大访问模式。

**定义 4** 已知 2 个用户访问模式分别为$X$和$Y$，以及一个访问模式集合$Z=\{Z_1,Z_2,\cdots,Z_n\}$。如果$Z_i\subset^s X$，同时$Z_i\subset^s Y(1\leq i\leq n)$，那么集合$Z$中的所有最大访问模式构成的集合称为访问模式$X$和$Y$的交集结果的集合$L$。

如 $X$={ABCDEF},$Y$={ACDF}，则 Z={{A},{C},{D},{F},{CD}}。于是，$X$ 和 $Y$ 的交集结果的集合 $L$={{A},{CD},{F}}。

## 2 GITC 算法与 Tree-DM 算法简述

GITC 算法主要基于如下 2 条原理：

(1)对 2 个访问模式求交，则求交结果中的访问模式将被这 2 个访问模式同时支持。

(2)通过求交，隐藏起来的访问模式可以被挖掘出来。例如：隐藏在{ABCD}和{BCDE}中的{BCD}会被挖掘出来。

算法简述：合并事务数据库 $T$ 中相同的访问模式，得到集合 $X$；对集合 $X$ 中的所有两两访问模式求交，得到交集结果集合 $Y$。接着将集合 $X$ 与 $Y$ 合并，且消除冗余访问模式后得到候选集合 $Z$。最后扫描原数据库 $T$，统计候选集合 $Z$ 中各个候选的支持数，完成挖掘。实例如下：

事务数据库 $T$：

{ABC},{ABC},{ABC},{ABCD},{ACDE},{CDE},{CDE},{BDE},{DE},{ABD},{B},{C}

(1)合并 $T$ 中相同的访问模式后，得到数组 $A$：

$A$={{ABC},{ABCD},{ACDE},{CDE},{BDE},{DE},{ABD}}

(2)对 $A$ 中的两两访问模式求交，过程见表 1。

(3)将求交结果与数组 $A$ 合并，且消除冗余访问模式，得到字符串数组 $B$：(删除 1 阶访问模式)

$B$={{ABC},{ABCD},{ACDE},{CDE},{BDE},{DE},{ABD},{AB},{CD},{BD}}

(4)扫描原数据库 $T$ 统计 $B$ 中各个候选的支持数，结果如下：(括号中的数字为相应访问模式的支持数)

{ABC4},{ABCD1},{ACDE1},{CDE3},{BDE1},{DE5},{ABD1},

**作者简介：**郭 维(1980－)，男，硕士，主研方向：数据挖掘
**收稿日期：**2007-03-23 **E-mail：**guowei2003128@21cn.com

{AB5},{CD4},{BD2}

**表 1　GITC 算法求交过程**

| | ABC | ABCD | ACDE | CDE | BDE | DE | ABD |
|---|---|---|---|---|---|---|---|
| ABC | - | - | - | - | - | - | - |
| ABCD | ABC | - | - | - | - | - | - |
| ACDE | A,C | A,CD | - | - | - | - | - |
| CDE | C | CD | CDE | - | - | - | - |
| BDE | B | B,D | DE | DE | - | - | - |
| DE | Φ | D | DE | DE | DE | - | - |
| ABD | AB | AB,D | A,D | D | BD | D | - |

(5)基于以上(4)中的结果可以直接查询，得到满足最小支持数阈值为 3 的最大频繁访问模式和部分非最大频繁访问模式组成的集合 *R*：

*R*={{CDE},{ABC},{CD},{DE},{AB}}

最后将集合 *R* 转换为相应的用户频繁访问模式集：

{{CD},{DE},{AB},{BC},{CDE},{ABC}}(已删除 1 阶访问模式)。

Tree-DM 算法：每扫描数据库中的一行，都要将该行事务对应的项目树与项目森林中已有的项目树求交，生成相应的事务树；再将新生成的事务树与事务森林中已有的事务树求交，来修改事务森林；所有求交过程结束后，可在事务森林中查询得到需要的挖掘结果。

## 3　GITC 算法改进

### 3.1　GITC 算法与 Tree-DM 算法分析

通过分析可以发现，在 GITC 算法执行过程中，扫描原数据库来统计支持数耗费的时间占据了 GITC 算法总运行时间的大部分。

Tree-DM 算法同样将求交作为基本算元，分析发现该算法存在如下问题：

(1)Tree-DM 算法不仅需要原数据库中不同项集间求交，而且需要将每次得到的项集间的新求交结果，与以前得到的所有求交结果都要进行二次求交，求交次数较多。

(2)Tree-DM 算法没有解决大量重复项集间的冗余求交问题，对于较大的数据库来说，如果不进行优化，这可能会成为严重的算法瓶颈。

(3)Tree-DM 算法需要构造比较复杂的数据结构即项目树和事务树。

(4)Tree-DM 算法挖掘结果中没有保留支持数为 1 的项集，但是随着原数据库的增大，这些项集的支持数很可能会增加，并成为可能的频繁结果，忽略它们是不正确的。

### 3.2　GI 算法

本文提出一种不需扫描原数据库来统计支持数的 GITC 算法的改进算法——GI(Getting the Intersections of each two access patterns)算法，它有效解决了上述 Tree-DM 算法中存在的 4 个问题：

针对问题(1)，用简化的一次扫描求交结果的过程来代替功能类似的反复二次求交过程(详见 GI 算法实例)；针对问题(2)，采用 GITC 算法的原理，即求交前先合并相同访问模式，来避免相同访问模式间的求交；针对问题(3)，直接采用普通数组作为基本数据结构，降低算法数据结构的复杂性；针对问题(4)，采用类似 GITC 算法的方法，来保留所有 1 阶访问模式。

GI 算法如下：

输入：已经存储在数组 *A* 中的事务数据库 *T*

输出：所有支持数对应的最大访问模式集

**Step1** 合并 *A* 中相同的访问模式，并累计支持数：

```
bool existed1=false;
for (int i=0;i<数组 A 的长度;i++)
{for (int j=0;j<数组 B 的长度;j++)
{existed1=false;
if (Ai.sequence==Bj.sequence)
{number=Bj.sup+1;
B.RemoveAt(j);
B.Add(Ai.sequence,number);
existed1=true;
}
}
if ((!existed1)&&(Ai.sequence的长度＞1))
B.Add(Ai.sequence,1);
}
```

**Step2** 求交：

```
bool existed2=false;
for (int k=0;k<数组 B 的长度-1;k++)
{for (int n=k+1;n<数组 B 的长度;n++)
{G=Bk.sequence∩Bn.sequence;
for (int m=0;m<数组 C 的长度;m++)
{if(G=Cm.sequence)
{existed2=true;
if(Bk.sequence 不出现在 Cm 中)
将 Bk.sequence 和 Bk.sup 插入 Cm 对应的存储空间;
if(Bn.sequence 不出现在 Cm 中)
将 Bn.sequence 和 Bn.sup 插入 Cm 对应的存储空间;
}
}
if (!existed2)
将 G,Bk.sequence,Bk.sup,Bn.sequence,Bn.sup 插入数组 C 中;
}
}
```

**Step3** for (int g=0;g<数组 C 的长度;g++)

```
{统计 Cg.sequence 的支持数和缩减项;
D.Add(Cg.sequence,支持数);
E.Add(缩减项);
}
```

**Step4** for (int e=0 ;e<数组 E 的长度;e++)

```
for (int h=0;h<数组 A 的长度;h++)
{if (Ah=Ee)
A.RemoveAt(h);//从 E 中删除所有缩减项
}
```

**Step5** A∪C

### 3.3　GI 算法实例

原数据库

{ABCD},{ABCD},{ABCD},{ABCDE},{ABCDE},{ABCDE},{DEFH},{DEFH},{DEFH},{DEFH},{CAB},{CAB},{CAB},{CAB},{CAB},{BEF},{BEF},{BEF},{BEF},{DEFI},{DEFI},{DEFI},{ABE},{ABE}

(1)合并数据库中相同访问模式，并累计其支持数：(括号内的数字就是其左边访问模式的支持数)

{ABCD3},{ABCDE3},{DEFH4},{CAB5},{BEF4},{DEFI3},{ABE2}

(2)对(1)结果中的两两元素间求交，同时保留求交得到每个交集结果的两个访问模式及其支持数，过程见表 2。

(3)接着扫描一边求交结果，将求交结果中相同访问模式的附加信息组织如下：(删除 1 阶访问模式)

ABCD:(ABCDE3+ABCD3);

AB:(CAB5+ABCD3),(ABE2+ABCD3),(CAB5+ABCDE3),(ABE2+ABCDE3),(ABE2+CAB5);

DE:(DEFH4+ABCDE3),(DEFI3+ABCDE3);

EF:(BEF4+DEFH4),(DEFI3+BEF4);

DEF:(DEFI3+DEFH4);

BE:(ABE2+BEF4)。

**表 2　GI 算法过程**

| | ABCD3 | ABCDE3 | DEFH4 | CAB5 | BEF4 | DEFI3 | ABE2 |
|---|---|---|---|---|---|---|---|
| ABCD3 | - | - | - | - | - | - | - |
| ABCDE3 | ABCD (ABCDE3+ABCD3) | - | - | - | - | - | - |
| DEFH4 | D | DE (DEFH4+ABCDE3) | - | - | - | - | - |
| CAB5 | C,AB (CAB5+ABCD3) | C,AB (CAB5+ABCDE3) | Ø | - | - | - | - |
| BEF4 | B | B,E | EF (BEF4+DEFH4) | B | - | - | - |
| DEFI3 | D | DE (DEFI3+ABCDE3) | DEF (DEFI3+DEFH4) | Ø | EF (DEFI3+BEF4) | - | - |
| ABE2 | AB (ABE2+ABCD3) | AB,E (ABE2+ABCDE3) | E | AB (ABE2+CAB5) | BE (ABE2+BEF4) | E | - |

(4)消除每个访问模式附加信息中的冗余访问模式及其支持数

ABCD:(ABCDE3+ABCD3)；

AB:(CAB5+ABCD3+ABE2 +ABCDE3)；

DE:(DEFH4+ABCDE3+DEFI3)；EF:(BEF4+DEFH4+DEFI3)；

DEF:(DEFI3+DEFH4)；

BE:(ABE2+BEF4)。

(5)计算支持数和缩减项(如果某个求交得到的访问模式在其附加信息中出现，那么该访问模式就是缩减项)

ABCD:(ABCDE3+ABCD3)支持数为 6；缩减项：ABCD(因为 ABCD 存在其附加信息(ABCDE3+ABCD3)中出现)

AB:(CAB5+ABCD3+ABE2 +ABCDE3)支持数为 13；缩减项：无

DE:(DEFH4+ABCDE3+DEFI3)支持数为 10；缩减项：无

EF:(BEF4+DEFH4+DEFI3)支持数为 11；缩减项：无

DEF:(DEFI3+DEFH4)支持数为 7；缩减项：无

BE:(ABE2+BEF4)支持数为 6；缩减项：无

(6)结果整理

原数据库

{ABCD3},{ABCDE3},{DEFH4},{CAB5},{BEF4},{DEFI3},{ABE2}

1)从原数据库中删除缩减项{ABCD}后得到集合 *A*

{ABCDE3},{DEFH4},{CAB5},{BEF4},{DEFI3},{ABE2}

2)求交结果

*B*{ABCD6},{AB13},{DE10},{EF11},{DEF7},{BE6}

3)将集合 *A* 和 *B* 放在一起，得到所有支持数对应的最大访问模式集

{ABCD6},{ABCDE3},{DEFH4},{CAB5},{BEF4},{DEFI3},{ABE2},{AB13},{DE10},{EF11},{DEF7},{BE6}

以下同 GITC 算法一样。

## 4　实验结果

为了评价 GI 算法的性能，本文在主频为 Intel Pentium III 733 MHz,128 MB 内存和 Windows 2000 操作系统的 PC 机上对该算法进行了测试，并与 GITC 算法进行了比较。所有的程序代码由 VC++6.0 编写，数据库系统是微软的 SQL Server 2000。所采用的数据为从 26/Jan/2004:00:00:14至26/Jan/2004:23:59:21 合肥工业大学网站服务器上的数据，经过数据预处理后，就构成了拥有 3 569 个事务的事务数据库。由于 GITC 算法与 GI 算法的最后一步，即在所有支持数对应的最大访问模式集上，查询需要的挖掘结果的过程完全相同，因此本实验就不考虑该过程。实验结果见表 3。

**表 3　挖掘时间比较**　s

| | GITC 算法 | GI 算法 |
|---|---|---|
| 挖掘时间 | 64.653 | 2.754 |

从表 3 中可以看出：相对于 GITC 算法来说，GI 算法在很大程度上减少了挖掘所需时间。这是因为 GI 算法用适当的数据结构来保存每个求交结果的支持数信息，省去了扫描原数据库来统计支持数耗费的大量时间。

## 5　结束语

本文对 Tree-DM 算法进行了深入分析，并结合了 GITC 算法的优点，提出了 GI 算法。GI 算法解决了 GITC 算法中扫描原数据库来统计支持数耗费大量挖掘时间的问题，也解决了 Tree-DM 算法在二次求交、冗余求交和复杂数据结构等方面存在的问题。且 GI 算法的挖掘结果中可以包含支持数为 1 的最大访问模式，这为以后的增量挖掘打下基础。实验证明，较 GITC 算法而言，GI 算法可以明显提高挖掘效率。

## 参考文献


[1] 白石磊, 毛雪岷, 王儒敬, 等. 一种快速挖掘频繁项目集算法[J]. 模式识别与人工智能, 2003, 16(4): 465-469.

[2] 郭　维. 基于交集关系的 Web 日志挖掘研究[D]. 合肥: 合肥工业大学, 2006.

[3] 陈嶷瑛, 武　强, 李文斌. 基于事务树操作的关联规则挖掘算法[J]. 计算机工程, 2006, 32(14): 40-42.


(上接第 56 页)


[2] NetScreen Technologies Inc.. Administrator’s Guide NetScreen-Security Manager[EB/OL]. (2004-07-06). http://www.juniper.net/software/ security-manager.

[3] Check Point Software Technologies Ltd. Check Point Smart Center Guide[EB/OL]. (2002-05-15). http://support.checkpoint.com/kb.

[4] Beigi M S. Policy Transformation Techniques in Policy-based Systems Management[C]//Proc. of IEEE International Workshop on Policies for Distributed Systems and Network. Yorktown: [s. n.], 2004.

[5] Mont M, Baldwin A. Power Prototype: Towards Integrated Policy Based Management[C]//Proc. of IEEE/IFIP Network Operations and Management Symposium. Honolulu: [s. n.], 2000.

[6] 曾旷怡. 一种基于域内的访问控制策略提炼模型及其实现[J]. 计算机工程, 2006, 32(11): 136-137.

[7] 王焕然. 路由器操作管理及基于策略的网络管理的研究与实现[D]. 北京: 清华大学, 2004.

[8] Niemann T. Lex 和 Yacc 简明教程[EB/OL]. (2006-10-29). http://www. epaperpress.com.

[9] Andreasson O. Iptables 指南 1.1.19[EB/OL]. (2006-11-07). http://www.man.ChinaUnix.net.