极大频繁子树挖掘及其应用*)

杨 沛[1] 谭 琦[2]

(华南理工大学计算机应用研究所 广州 510640)[1] (华南师范大学计算机学院 广州 510631)[2]

摘 要 极大频繁子树挖掘在Web挖掘、HTML/XML文档分析、生物医学信息处理等领域有着重要的应用,可用于解决这些领域的自同构问题。本文提出了一种极大频繁子树挖掘算法(MFTM)。MFTM基于最右路径扩展技术,在搜索过程中,采用覆盖定理进行裁剪,压缩搜索空间,从而极大地加快了算法的收敛速度。性能实验表明,极大频繁挖掘等算法是有效和可伸缩的。

关键词 频繁子树挖掘,Web挖掘,信息抽取

Maximum Frequent Tree Mining and its Applications

YANG Pei[1] TAN Qi[2]

(Research Institute of Computer Application, South China University of Technology, Guangzhou 510640)[1]

(Institute of Computer Science and Engineering, South China Normal University, Guangzhou 510631)[2]

Abstract A novel algorithm called Maximum Frequent Tree Mining (MFTM) is presented to discover maximum frequent sub-trees from forest. MFTM uses the right-most path expansion technique. The Overlay Theorem is proposed to reduce the search space and accelerate the convergence speed. We conduct detailed experiments to test the performance and scalability of the methods. The experiments demonstrate that MFTM is effective and scalable. MFTM can be applied to solve the isomorphic problems in the domains such as Web mining, HTML/XML data analysis, bioinformatics, and so on.

Keywords Frequent tree mining, Web mining, Data extraction

1 引言

频繁子树(图)挖掘是指在森林(图数据库)中寻找频繁出现的子树(图)。频繁子树(图)挖掘在Web挖掘[1]、半结构化数据处理[2]、生物化学信息分析[3]等领域有着日趋重要的应用。

Inokuchi[4]等提出AGM算法,用于在图数据库中寻找频繁子图。Kuramochi[5]在AGM的基础上提出了FSG算法。AGM和FSG都是利用了Apriori的反单调性质。此外,文[6]介绍了SUBDUE系统,文[7]介绍了GBI系统。无论SUBDUE或是GBI,采用的都是贪婪搜索策略,所以得到的可能不是频繁子图的完全集。近一两年来,T. Asai等[8]提出了FREQT算法,用于在森林中寻找频繁子树。M. J. Zaki[9]提出了TREEMINER算法,利用等前缀类扩展(Prefix Equivalence Class Extension)来寻找频繁子树。与FREQT不同的是,TREEMINER挖掘的是广义的频繁子树。FREQT和TREEMINNER都是采用了类似于Apriori逐级连接机制,通过频繁k-子树的连接产生候选$(k+1)$-子树,并对候选$(k+1)$-子树进行频繁度测试,从而得到频繁$(k+1)$-子树。

以上算法都试图找出频繁子树(图)的完全集,因而挖掘的输出结果也是相当庞大的。如何从庞大的挖掘结果中筛选出我们感兴趣的知识,这是一个需要解决的问题。另一方面,上述算法的搜索空间非常巨大,导致算法的时间和空间复杂度很高,在实际应用中会受到较大的限制。为了能快速、高效地从森林中挖掘到极大频繁子树,在前期工作的基础上[10],我们提出了一种极大频繁子树挖掘算法(MFTM)。MFTM基于最右路径扩展技术,在搜索过程中,采用覆盖定理进行裁剪,压缩搜索空间,从而极大地加快了算法的收敛速度。

2 问题定义

本文针对的是有序树。树用$T=(V,E)$表示,其中,V是节点的集合,E是边的集合。对树T进行深度优先遍历,得到树T的前序序列。树T的前序序列中的最后一个节点称为树T的最右节点,用$\mathrm{rml}(T)$表示。数据库D是由多棵互不相交的树构成的森林。

如果树$S=(V_s,E_s)$满足条件:(1)$V_s\subseteq V$;(2)$e=(v_x,v_y)\in E_s$当且仅当在树T中,v_x是v_y的父节点,则称S为T的子树,记为$S\subseteq T$。大小为k的子树称为k-子树。

用$\delta(S,T)$表示子树S在树T中是否出现。如果出现,则令$\delta(S,T)=1$,否则令$\delta(S,T)=0$。在数据库(森林)D中,子树S的支持度表示为:$\mathrm{support}(S)=\sum_{T\in D}\delta_T(S,T)$。设用户自定义的最小支持度阈值为$\beta$,若$\mathrm{support}(S)\geqslant\beta$,则称$S$为频繁子树。

设v_0是树S的根节点,v_m是树S的最右节点。树S中的一个有限点边交替序列$(v_0,e_1,v_1,e_2,\cdots,e_m,v_m)$,使得对$1\leqslant i\leqslant m$,$e_i$的端点是$v_{i-1}$和$v_i$,且满足$v_i$是$v_{i-1}$的最右子节点(即$v_i=\gamma_{v_{i-1}}$),则称序列$(v_0,e_1,v_1,e_2,\cdots,e_m,v_m)$为树$S$的最右路径,记作$\mathrm{rmp}(S)$。

设S_k是k-子树,S_k的前序序列$\Theta(S_k)=(v_1,v_2,\cdots,v_k)$。设$S_{k+1}$是$(k+1)$-子树,$S_{k+1}$是将节点$v_{k+1}$嫁接到$S_k$的某个节点$v_i(1\leqslant i\leqslant k)$上(即将$v_{k+1}$作为$v_i$的子节点)得到。如果有$\Theta(S_{k+1})=(v_1,v_2,\cdots,v_k,v_{k+1})$,则有$v_i$必是位于$S_k$的最右路径上(即$v_i\in\mathrm{rmp}(S_k)$)。反之,如果有$v_i\in\mathrm{rmp}(S_k)$,则

*)国家自然科学基金(60003019)资助。杨 沛 博士,研究方向:Web挖掘、机器学习、信息抽取。

必有 $\Theta(S_{k+1})=(v_1,v_2,\cdots,v_k,v_{k+1})$。基于最右路径扩展产生的频繁子树具有唯一性和完备性，如果 S_{k+1} 是 S_k 从其最右路径上扩展得到，则简记为 $S_{k+1}=S_k\oplus v_{k+1}$。如图 1，红色标出的节点序列$(v_1,v_3,v_4,v_6,v_7))$构成了树 S 的最右路径。v_c 是待扩展节点，它嫁接到 v_3 节点上，构成一个新的子树。

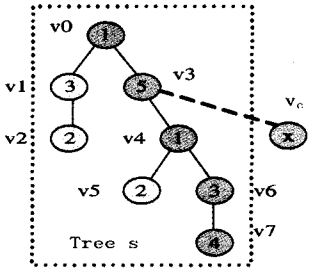

图 1　最右路径扩展

设 S 是 T 的子树，即 $S\subseteq T$，将 S 的最右节点在树 T 中的位置简记为 $p(S,T)$。子树 S 可能在树 T 上不同的地方多次出现，将最小的位置记为 $p_{\min}(S,T)$。

频繁子树 S 的文档集定义为：$D(S)=\{T_i\mid T_i\in D,\delta(S,T_i)=1\}$。由频繁子树 S 递推扩展产生的所有频繁子树的集合记为 $\Omega(S)$。由 S 扩展产生的极大频繁子树记为 $T_{\max}(S)$，满足 $|T_{\max}(S)|=\max\limits_{T_i\in\Omega(S)}|T_i|$。

3　极大频繁子树挖掘

3.1　覆盖定理

定义（覆盖关系）

对于任意两棵频繁子树 S_x 和 S_y，若满足以下关系：

$|T_{\max}(S_x)|\geqslant|T_{\max}(S_y)|$

则称 S_x 覆盖 S_y，简记为 $S_x\succ S_y$。

如果 S_x 覆盖 S_y，则可将 S_y 删除。理由是，相对 S_x 而言，S_y 不可能产生比 S_x 更大的极大频繁子树。在搜索过程中，可利用覆盖关系对待扩展节点进行裁剪，压缩搜索空间，从而加快算法的收敛速度。

定理（覆盖定理）

对于任意两棵频繁子树 S_x 和 S_y，若满足以下关系：

1) $|S_x|\geqslant|S_y|$；

2) $D(S_x)\supseteq D(S_y)$；

3)对于 $\forall T_i\in D(S_y)$，均有 $p_{\min}(S_x,T_i)\leqslant p_{\min}(S_y,T_i)$；

则有：$S_x\succ S_y$

证明：对于 $\forall T_y\in\Omega(S_y)$，均可构造这样的频繁子树 T_x：

$\because D(S_x)\supseteq D(S_y)$；且对于 $\forall T_i\in D(S_y)$，均有 $p_{\min}(S_x,T_i)\leqslant p_{\min}(S_y,T_i)$，

$\therefore$可将 S_x 的最右叶子节点嫁接到 T_y 上，从而得到一棵新的频繁子树 T_x，显然有 $|T_x|=|T_y|+1$，$D(T_y)\subseteq D(T_x)$。

下面比较 T_x 和 T_y 的支持度：

$$\text{support}(T_x)=\sum_{T\in D(T_x)}\delta(T_x,T_i)\geqslant\sum_{T_i\in D(T_y)}\delta(T_x,T_i)=\sum_{T_i\in D(T_y)}\delta(T_y,T_i)=\text{support}(T_y)$$

同时，T_x 可由 S_x 扩展得到，因此有 $T_x\in\Omega(S_x)$。

综上所述，对于 $\forall T_y\in\Omega(S_y)$，总可以在 $\Omega(S_x)$ 中找到一棵比 T_y 更大的频繁子树 T_x。因此，有 $|T_{\max}(S_x)|\geqslant T_{\max}(S_y)|$，即有 $S_x\succ S_y$。　（证毕）

3.2　极大频繁子树挖掘算法(MFTM)

输入：树数据库 D，最小支持度阈值 min_sup

输出：极大频繁子树集 Ω

1)扫描原始数据库 D 中所有树的根节点，得到频繁 1-子树集 L_1，并将 L_1 中所有的频繁 1-子树添加到频繁子树队列 freqtree_vec。

2)如果频繁子树队列 freqtree_vec 为空，则返回极大频繁子树集 Ω。否则，转步骤 3。

3)利用覆盖定理对频繁子树队列 freqtree_vec 进行裁剪，删除被覆盖的频繁子树。

4)从频繁子树队列 freqtree_vec 中取出一个元素 F_k（F_k 为频繁 k-子树），若 F_k 不可扩展，则转步骤 5；否则，对 F_k 进行扩展，得到一个或多个频繁$(k+1)$-子树，并分别将它们添加到频繁子树队列 freqtree_vec 中。转步骤 2。

5)将 F_k 添加到极大频繁子树集 Ω 中。转步骤 2。

3.3　时间复杂度分析

MFTM 的算法分析比较复杂。为简便计，设每个节点扩展之后平均得到 b 个频繁子节点。这样，通过挖掘算法最终得到的极大频繁子树是一棵完全 b 叉树 T。假设该完全 b 叉树 T 的深度为 d，节点总树为 n。下面利用分治法来推导 MFTM 的时间复杂度。

如图 2，v_0 是树 T 的根节点，$v_i(i=1,\cdots,b)$是 v_0 的子节点，$t_i(i=1,\cdots,b)$是以 v_i 为根的频繁子树。设以 v_0 为根的频繁子树数为 $T(n)$，则有以 v_i 为根的频繁子树数为 $T(\frac{n-1}{b})$。以 v_0 为根的频繁子树可通过将 t_i 进行组合得到，构造过程如下：以 v_0 作为根，分别从以 v_i 为根的频繁子树集中各取一棵频繁子树 t_i（共有 $T(\frac{n-1}{b})$种可能）或者不取（一种可能），因为有 b 个子树，所以一共取 b 次，并分别按次序嫁接到 v_0 上，得到 v_0 为根的频繁子树。

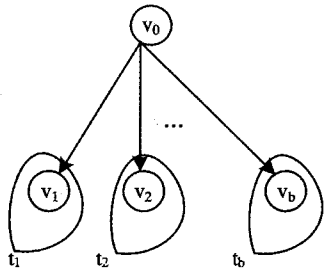

图 2　时间复杂度分析

因此，根据组合原理，可推得以下递推公式：

$$T(n)=[T(\frac{n-1}{b})+1]^b$$

又因为 $T(1)=1$，所以有如下递推方程：

$$\begin{cases}T(n)=[T(\frac{n-1}{b})+1]^b & (n\geqslant b+1)\\ T(1)=1\end{cases}$$

可推得 $T(n)=O(2^{b^d})=O(2^n)$（推导过程略）。

可见，极大频繁子树挖掘过程产生的频繁子树的数量是非常巨大的，是指数级的。指数级的算法在实际中一般很难应用。如果根据覆盖定理在挖掘过程中进行裁剪，一般可将 b 减少到 2 以下（大多数情况可将 b 减少为 1），这样就极大地降低了 MFTM 的时间复杂度。假设 b 等于 2 的概率为 p（初步实验结果表明 $p<10\%$），b 等于 1 的概率为 $1-p$，这样 MFTM 的时间复杂度为：

$$T(n)=O(2^{b^d})=O(2^{2^{dp}\times 1^{d(1-p)}})=O(2^{2^{dp}})$$

4　实验分析

实验在 PC 机上进行，CPU 是 Pentium 4 3.0GHz，内存

为 1GB。

4.1 可伸缩性

可伸缩性是评价一个数据挖掘算法的关键技术指标之一。可伸缩性实验采用五个不同的数据库。五个数据库的网页的总数分别为：200、500、1000、2000、5000。实验结果如图3所示。横坐标表示网页数，纵坐标表示 CPU 运行时间，以秒为单位。支持度阈值分别设为 5%和 2%，进行了两组实验。

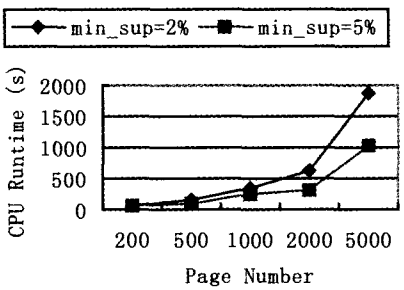

图 3 可伸缩性实验

算法运行时间和支持度阈值成反比关系。用户设定的支持度阈值越小，挖掘得到的频繁子数也越多，算法运行时间也越长。随着支持度阈值的调低(从 5%调低到 2%)，根据网页数的不同，算法的运行时间分别增加了 26%、55%、32%、98%、42%。

4.2 MFTM 应用于兴趣网页的发现

通过对大量电子商务网站的分析可以发现，购物网站(较为典型，但也不局限于购物网站)往往符合如下规律：(1)一致性规则——网站建设者总是试图使得网站的所有网页具有一致或相似的布局、外观和色调；(2)大同小异原则——同一类型的网页，如商品信息，在不同网页之间，其结构大体相同，而内容稍有差异。这种网页一般都是通过网页模板产生的。从数据库中读取不同的商品记录，填进模板中，从而产生不同的网页。

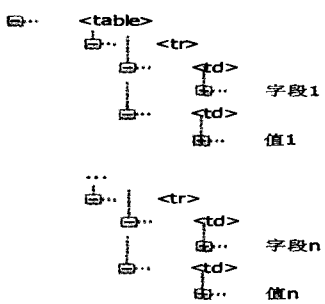

图 4 兴趣信息块

图 4 是一个典型的包含商品信息的 HTML 片断，我们称之为兴趣信息块。对于包含兴趣信息块的网页，我们称之为兴趣网页。元素〈table〉表示一个记录，每个〈table〉包含多个〈tr〉元素。每个〈tr〉元素包含两个〈td〉元素，第一个〈td〉表示字段，第二个〈td〉表示字段的值。对于同一模板产生的不同网页，字段是相同的，而字段值是不同的。

兴趣网页之间具有很高的结构相似性，而极大频繁子树挖掘正好可以用来挖掘这种结构的相似性，从而从大量的网页中筛选出兴趣网页。通过对挖掘到的极大频繁子树集进行模式匹配，可以从中筛选出包含兴趣信息块的兴趣网页。实验采用的网页数据都是从一些比较流行的手机网站上收集的。采用查准率(Precision)和查全率(Recall)作为算法的评价指标。其中，

$$\text{Precision}=\frac{\text{正确抽取到的兴趣网页数}}{\text{抽取到的网页数}}$$

$$\text{Recall}=\frac{\text{正确抽取到的兴趣网页数}}{\text{总的兴趣网页数}}$$

从表 1 中可以看出，算法平均的抽取精度为 98.9%，平均的查全率为 96.2%。实验结果表明，系统具有较高的抽取精度和查全率。

表 1 查准率—查全率

Web site	Totol pages#	Interest pages#	Extracted#	Correctly extracted#	P	R
www.imobile.com.cn	100	24	24	24	100%	100%
www.18900.com	100	37	36	35	97%	94.6%
www.3533.com	100	11	11	11	100%	100%
www.enet.com.cn	100	17	17	17	100%	100%
www.mobile8848.com	100	40	38	38	100%	95%
www.xieheng.com	100	26	24	24	100%	92.3%
mobile.okwit.com	100	32	31	30	96.8%	93.8%
www.showji.com.cn	100	21	21	21	100%	100%
www.imobile.com.au	100	35	33	32	97%	91.4%
www.younet.com	100	43	43	43	100%	100%
Total	1000	286	278	275	98.9%	96.2%

结论 通过极大频繁子树挖掘，可以从海量的网页集合中筛选出兴趣网页，并找出兴趣信息块。下一步，我们将兴趣信息块进行信息抽取[11,12]，发现隐含的数据库模式。同时，极大频繁子树挖掘还可应用于网页的分类和聚类。

极大频繁子树挖掘可以应用到很多领域，如 Web 挖掘、半结构化数据处理、生物化学信息分析等。这些海量信息都是具有结构性的，采用常规的数据挖掘方法，如关联分析、序列模式挖掘等，并不能有效地从这些海量信息中获取到结构性的知识。通过极大频繁子树挖掘，可以解决这些领域的自同构问题。因此，极大频繁子树在这些领域的应用会日趋广泛。

参考文献

1 Cooley R, Mobasher B, Srivastava J. Web Mining: Information and Pattern Discovery on the World Wide Web. In: 8th IEEE Intl Conf on Tools with AI, 1997

2 Li Q, Moon B. Indexing and querying XML data for regular path expressions. In: 27th Int'l Conf. on Very Large Data Bases, 2001

3 Shapiro B, Zhang K. Comparing multiple RNA secondary strutures using tree comparisons. Computer Applications in Biosciences, 1990, 6(4): 309～318

4 Inokuchi A, Washio T, Motoda H. An apriori-based algorithm for mining frequent substructures from graph data. In: 4th European Conference on Principles of Knowledge Discovery and Data Mining, September 2000

5 Kuramochi M, Karypis G. Frequent subgraph discovery. In: 1st

IEEE Int'l Conf. on Data Mining, November 2001
6 Cook D, Holder L. Substructure discovery using minimal description length and background knowledge. Journal of Artificial Intelligence Research, 1994, 1: 231～255
7 Yoshida K, Motoda H. CLIP: Concept learning from inference patterns. Artificial Intelligence, 1995, 75(1): 63～92
8 Asai T, Abe K, Kawasoe S, et al. Effecient substructure discovery from large semi-structured data. In: 2nd SIAM Int'l Conference on Data Mining, April 2002
9 Zaki M J. Efficiently mining frequent trees in a forest. In: SIGKDD'2002 Edmonton, Alberta, Canada
10 杨沛,郑启伦,彭宏,等. PFTM:一种基于投影的频繁子树挖掘算法. 计算机科学, 2005, 32(2): 206～209
11 Golgher R D, Silva P, Laender A. Automatic Web news extraction using tree edit distance. In: WWW-04, 2004
12 Liu B, Grossman R, Zhai Y H. Mining Web Pages for Data Records. In: WWW-05, 2005

(上接第 133 页)

AVICap 提供了预览(Preview)模式和叠加(Overlay)模式显示视频。在预览模式中视频帧先从捕获硬件传到系统内存,接着采用 GDI 函数在捕获窗口显示;在叠加模式中,叠加视频的显示不经过 VGA 卡,叠加视频的硬件将 VGA 的输出信号与其自身的输出信号合并,形成组合信号显示在计算机的监视器上。可以利用宏 capPreview 启动和停止预览模式,同时需要用宏 capPreviewRate 设置预览播放频率。如:

```
capPreviewRate(hWnd,66); //预览播放频率
capPreview(hWnd,TRUE); //启动预览模式
```

可用宏 capOverlay 来启动和停止叠加模式,如:

```
CAPDRIVERCAPS CapDrvCaps;
capDriverGetCaps(hWnd, &CapDrvCaps, sizeof(CAPDRIVERCAPS));
if(CapDrvCaps.fHasOverlay) //检查设备是否支持叠加模式
capOverlay(hWnd,TRUE); //启动叠加模式
```

(9)利用宏 capFileSetCaptureFile 指定为保存视频剪辑所需的文件名,利用宏 capFileAlloc 为文件分配内存缓冲区。其用法如下:

```
Char szCaptureFile[]="MyCap.AVI";
capFileSetCaptureFile(hWnd,szCaptureFile);
capFileAlloc(hWnd,1024L*1024L*5));
```

(10)利用宏 capCaptureSequence 启动视频捕捉,利用宏 capFileSaveAs 将捕捉的视频数据保存到文件中。视频启动和保存的代码如下:

```
Char szNewName[]="NewFile.AVI";
capCaptureSequence(hWnd);
capFileSaveAs(hWnd,szNewName);
```

上面介绍了视频捕捉的主要流程,为了更好地控制视频流捕捉过程,还需要用到其他的 AVICap 宏。如宏 capDlgVideoSource、宏 capDlgVideoFormat;宏 capDlgVideoDisplay、分别用来显示视频对话框、格式和显示。图 1 是本算法的主要流程图。

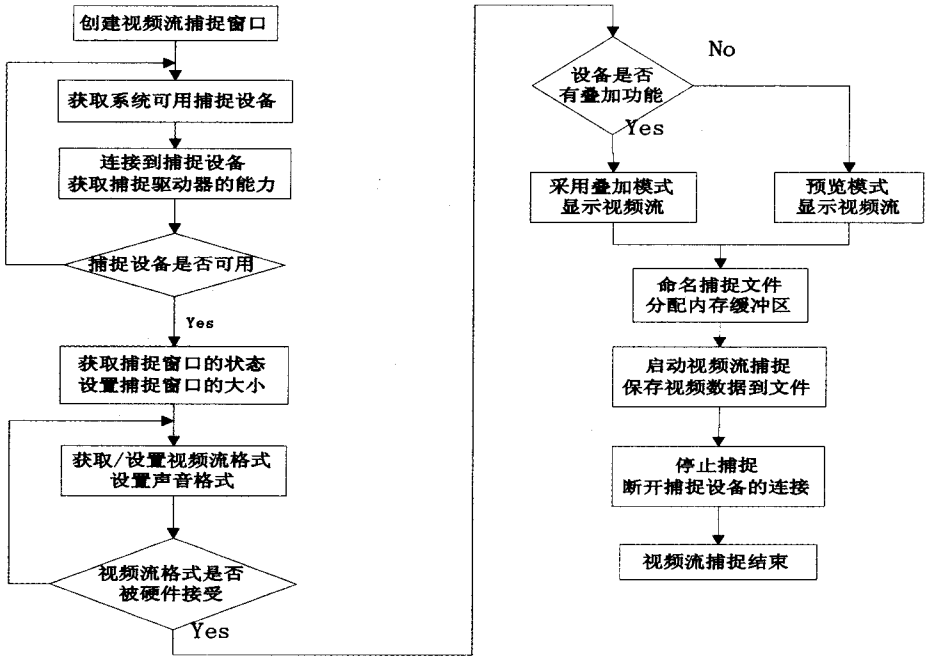

图 1

结束语 本文综述了 VFW 技术中的 AVICap 视频捕捉模块的技术原理,用 Visual C++编程实现了 AVI 格式视频流的捕捉算法,并且应用在桌面视频会议、可视电话等多媒体应用中。提出的算法对于从电视机、录象机等外部设备获取数字视频并进行编辑、存储、传输等具有重要的实用价值。

参 考 文 献

1 朱仲杰,蒋刚毅,郁梅,等. 一种基于 MPEG-2 的立体视频编码中的视差匹配快速算法[J]. 电路与系统学报,2003,8(1)
2 查锦发,陈莘萌. 音、视频数据捕捉方法[J]. 计算机工程, 2003, 29(12)
3 刘炜玮. Visual C++ 视频/音频开发实用工程案例精选[M]. 人民邮电出版社, 2004
4 王洪涛. 深入剖析 Visual C++编程技术及应用实例[M]. 人民邮电出版社,2003