PFTM:一种基于投影的频繁子树挖掘算法*)

杨 沛 郑启伦 彭 宏 李颖基

(华南理工大学计算机科学与工程学院 广州510640)

摘 要 频繁子树在Web挖掘、XML文档分析、生物信息处理等领域有着重要的应用。提出了一种新的基于投影的频繁子树挖掘算法(PFTM),通过对数据库和候选节点集进行投影,并采用递推式候选节点集更新技术来有效地压缩搜索空间,以高效地从森林中挖掘出频繁子树。PFTM不需要产生候选子树。性能对比实验表明,PFTM是有效和可扩展的,而在算法效率上,PFTM要比FREQT平均高出40%左右。

关键词 频繁子树挖掘,关联规则,序列模式

PFTM:A Frequent Subtrees Mining Algorithm Based on Projection

YANG Pei ZHENG Qi-Lun PENG Hong LI Ying-Ji

(School of Computer Science and Engineering,South China University of Technology,Guangzhou 510640)

Abstract Mining frequents subtrees is very useful in domains such as Web mining,XML data analysis,bioninformatics,and so on. A novel algorithm called PFTM to discover all frequent subtrees in a forest based on projection is presented. The process partitions both the projected database and the set of candidate nodes. No candidate subtree need to be generated by PFTM. An novel method of recursive updating the set of candidate nodes is applied to greatly reduce the search space. We contrast PFTM with FREQT. We conduct detailed experiments on synthetic datasets to test the performance and scalability of these two methods. Experiments show that PFTM outperforms the FREQT by an average of 25 percent.

Keywords Frequent subtrees mining,Association rule,Sequential patterns mining

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都是基于最右路径扩展技术(Rightmost Path Expansion)的。同时,两者都是采用了类似于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。

Apriori[10]性质是指频繁项集的所有非空子集都必须也是频繁的。各种基于Apriori的频繁子树(图)算法利用Apriori性质压缩搜索空间,虽然效果明显,但存在两个主要缺点[11]。一个是在连接步产生大量候选项集,且其连接具有一定的盲目性,会产生很多在数据库中实际不存在的候选子树,对这些实际上不存在的候选子树的频繁度测试会耗费大量的时间。另一个缺点是要重复扫描数据库,I/O开销巨大。

是否能够不产生候选子树呢?在序列模式挖掘方面,J. W. Han和J. Pei等独辟蹊径,先后提出了不需要产生候选序列的FreeSpan[12]和PrefixSpan[13]算法。通过对数据库和序列模式进行投影,压缩搜索空间,从而大大提高了算法效率。而树也可以用一个序列来表示,因此,PrefixSpan的思想可以借鉴,并加以利用。

此外,FREQT的一个主要缺点是,在对候选子树进行频繁度测试时,需要重复多次地扫描最右路径上节点的子节点集,此前逐次扫描结果不能加以利用,从而导致算法效率降低。

为此,本文提出了一种新的基于投影的频繁子树挖掘算法,在最右路径扩展的基础上,利用递推式的候选节点集更新技术来压缩候选节点集,从而压缩搜索空间,并利用C-矩阵进行候选节点支持度的快速计数,以达到高效地挖掘出频繁子树完全集的目的。

2 频繁子树及最右路径扩展

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。树T中包含的节点的个数表示树T的大小,用$|T|$表示。边$e=(v_x,v_y)\in E$表示节点v_x是v_y的父节点。节点v的子节点集用F_v表示。节点v的最右子节点用r_v表示,显然有$r_v\in F_v$。

*)基金项目:广东省科技攻关项目(C10201、A1020103)资助。

森林 D 是 $m(m\geqslant 0)$棵互不相交的树的集合。

对树 T 进行深度优先遍历，得到树 T 的前序序列，用 $\Theta(T)$表示。节点 v 在树 T 的前序序列中的索引位置(简称为节点 v 在树 T 中的位置)用 ρ 表示，ρ 满足$0\leqslant\rho\leqslant|T|-1$。每个节点 v 对应一个标签，用 l 表示。不同的节点的标签可能相同。数据库 D 中所有标签的集合用 L 表示。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 $rml(T)$表示。

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 $S\prec T$。大小为 k 的子树称为 k-子树。

用 $\delta_T(S)$表示子树 S 在树 T 中是否出现。如果出现，则令 $\delta_T(S)$为1，否则令 $\delta_T(S)$为零。在数据库(森林)D 中，子树 S 的支持度表示为：$support(S)=\sum_{T\in D}\delta_T(S)$。设用户自定义的最小支持度阈值为 β，若 $support(S)\geqslant\beta$，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 $rmp(S)$。为简便计，将 $rmp(S)$中包含的边剔除，只保留节点序列。$rmp(S)$中包含的节点的个数称为树 T 的最右路径长度，记作$|rmp(S)|$。

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 $\Theta(S_{k+1})=(v_1,v_2,\cdots,v_i,v_{k+1})$，则有，$v_i$ 必是位于 S_k 的最右路径上(即 $v_i\in rmp(S_k)$)。反之，如果有 $v_i\in rmp(S_k)$，则必有 $\Theta(S_{k+1})=(v_1,v_2,\cdots,v_k,v_{k+1})$。基于最右路径扩展产生的频繁子树具有唯一性和完备性[3,9]。如果 S_{k+1}是 S_k 从其最右路径上扩展得到，则简记为 $S_{k+1}=S_k\oplus v_{k+1}$。

设 S 是 T 的子树，即 $S\prec T$，将 S 的最右节点在树 T 中的位置简记为 $\rho^*_{T|S}$。子树 S 可能在树 T 上不同的地方多次出现，因此，可能存在多个的 $\rho^*_{T|S}$。

3 基于频繁子树的投影数据库

定义1(候选节点) 设 S 是 T 的子树(即 $S\prec T$)，S 的最右节点在树 T 中的位置为 $\rho^*_{T|S}$，节点 u 位于 S 的最右路径上(即 $u\in rmp(S)$)，u 在 $rmp(S)$中的索引位置用 $\tau(0\leqslant\tau\leqslant|rmp(S)|-1)$表示，设有节点 $v_c\in F_u$，且满足位置关系 $\rho(v_c)>\rho^*_{T|S}$，则称 v_c 为(子树 S 在 T 上基于 $\rho^*_{T|S}$的)候选节点。候选节点用三元组(l,ρ,τ)表示。

候选节点 $v_c(l,p,\tau)$的直观含义是，可以将标签为 l 的节点嫁接到频繁子树 S 的最右路径的第 τ 个位置上，而 ρ 则记录了节点在树 T 中的位置。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 是一个候选节点。

定义2(候选节点集) (子树 S 在 T 上基于的 $\rho^*_{T|S}$的)所有候选节点 v_c 的集合称为(子树 S 在 T 上基于的 $\rho^*_{T|S}$的)候选节点集，记为 Ω。

S 可能在 T 中不同的位置多次出现，因此，不同的 $\rho^*_{T|S}$对应有不同的 Ω。

定义3(投影子树) 树 T 上基于子树 S 的投影子树用二元组$(T_id,\sum_{i\geqslant 1}(\rho_i^*,\Omega_i))$表示，记为 $\pi_{T|S}$，其中，T_id 是指向树 T 的指针，(ρ_i^*,Ω_i) 为$(\rho^*,\rho^*$对应的候选节点集)值对。

定义4(投影数据库) 所有基于子树 S 的投影子树 $\pi_{T|S}$的集合构成投影数据库 $D|_S$。即有 $D|_S=\{\pi_{T_i|S}|\pi_{T_i|S}$为投影子树，$T_i\in D,S\prec T_i,1\leqslant i\leqslant m\}$。

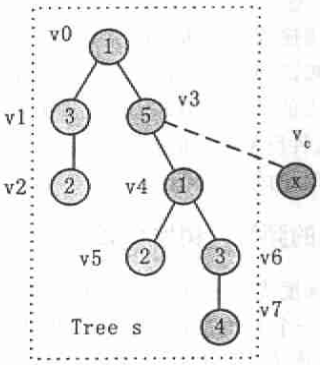

图1 right-most path expansion

3.1 C-矩阵

设 S_k 为频繁 k-子树，要由 S_k 递推得到频繁$(k+1)$-子树，则只需要扫描基于 S_k 的投影数据库 $D|_{S_k}$即可。通过扫描 $D|_{S_k}$中所有的候选节点集，可以得到一个候选节点的频繁1-项集，记为 $L_1(S_k)$。将 $L_1(S_k)$中任一候选节点 $u(u\in L_1(S_k))$嫁接到 S_k 上，即可得到一颗新的频繁$(k+1)$-子树 $S_{k+1}=S_k\oplus u$。

为了加快扫描速度，采用一个二维矩阵 C 进行候选节点的支持度计数。矩阵中的行表示频繁子树最右路径上的索引位置(即 τ)，列表示候选节点的标签(即 l)。$C[i][j]$表示嫁接到频繁子树最右路径的 τ 位置上的标签为 l 的候选节点的支持度计数。逐个处理投影数据库 $D|_{S_k}$中的每一个候选节点 $v_c(l,\rho,\tau)$，即可构造基于 S_k 的 C-矩阵。

如果在同一候选节点集 Ω_c 中，存在两个(或多个)候选节点，不妨设为 v_1和 v_2，满足如下三个条件：(1)$\tau_1=\tau_2$；(2)$l_1=l_2$；(3)$\rho_1\neq\rho_2$，支持度计数时则只需要计算一次。这对应于频繁子树 S 在树 T 中多次出现，但对 S 的支持度计数只贡献了一次的情况。

3.2 候选节点集递推式更新

设 S_{k-1}为频繁$(k-1)$-子树，S_{k-1}对应的频繁1-项集为 $L_1(S_{k-1})$。将 $L_1(S_{k-1})$中任一候选节点 $u(u\in L_1(S_{k-1}))$嫁接到 S_{k-1}上，即可得到相应的频繁 k-子树 $S_k=S_{k-1}\oplus u$。那么，S_k 的候选节点集应该怎么构造呢?根据 Apriori 反单调性质，频繁项集的所有非空子集都必须也是频繁的，因此，非 $L_1(S_{k-1})$中的候选节点将被裁减掉。与此类推，可以归纳出以下的候选节点集的递推式更新公式：

$$\begin{cases}\Omega(S_0)=\Phi\\ \Omega(S_k)=\rho(\Omega(S_{k-1}))+f(u)(k\geqslant 1)\end{cases}\quad(1)$$

其中，S_{k-1}为频繁$(k-1)$-子树，S_k 为 k-子树，u 为节点，$u\in L_1(S_{k-1})$，$S_k=S_{k-1}\oplus u$。显然，u 变成了 S_k 的最右节点。

(1)P 为筛选函数。对于 $\Omega(S_{k-1})$中的任一候选节点 $v_c(v_c\in\Omega(S_{k-1}))$，如果同时满足两个条件：(1)$\rho(v_c)>\rho(u)$；(2)$v_c\in L_1(S_{k-1})$，则 v_c 将被筛选出来，添加到 $\Omega(S_k)$中。否则，v_c 将被裁减掉。

(2)$f(u)$为扩展函数。设有 $u(l,\rho,\tau)$。因为 u 变成了 S_k 的最右节点，而 u 是嫁接到 S_k 的最右路径的第 τ(从0开始计算)个位置上，所以 S_k 的最右路径长度变成了 $\tau+1$。根据如下方法进行最右节点的扩展：扫描 u 的子节点集 F_u，对于任一节

点 $v\in F_u$，构造一个新的候选节点 $v(l_v,\rho_v,\tau+1)$，并将 $v(l_v,\rho_v,\tau+1)$ 添加到 $\Omega(S_k)$ 中。$v(l_v,\rho_v,\tau+1)$ 的直观含义是指，可以将节点 v 嫁接到节点 u 上，而节点 u 是位于 S_k 的最右路径的第$(\tau+1)$个位置上。

候选节点集递推式更新的实质是利用 S_k 之前的逐次扫描得到的历史知识，来压缩 S_k 的搜索空间，避免对 S_k 的最右路径上所有节点的子节点集进行反复多次的扫描。随着树数据库的增大，这种反复多次的扫描所耗费的时间会急剧地增加，从而导致算法的效率降低。

4 基于投影的频繁子树挖掘算法

对树进行深度优先遍历，如果从子节点回退到父节点，则在序列中添加一个-1作为标记，这样得到的一个序列用 Γ 表示。原始数据库 D 是序列 Γ 的集合。如图中，$T1$的编码为 $\Gamma=\{1,2,5,-1,6,-1,-1,3,4,-1,-1\}$。

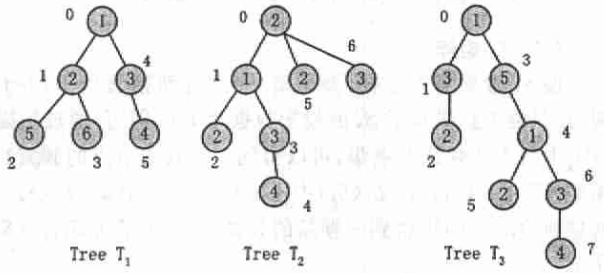

图2 Forest of 3 trees

4.1 举例说明算法 PFTM

1. 扫描原始数据库 D，得到频繁1-项集为 $L_1=\{1:3,2:3,3:3,4:3\}$。设支持度阈值为3，得到4棵频繁1-子树及其支持度计数，下面以频繁子树 $S_1=\{1\}$为例进行说明。

2. 构造基于频繁子树 S_1的投影数据库 $D|_{S1}$，得到(用(l,ρ,τ)表示候选节点)：

1)$T1,\rho^*=0,\Omega:(2,1,0),(3,4,0)$；

2)$T2,\rho^*=1,\Omega:(2,2,0),(3,3,0)$；

3)$T3,\rho_1^*=0,\Omega_1:(3,1,0);\rho_2^*=4,\Omega_2:(2,5,0),(3,6,0)$。

3. 使用 C-矩阵进行频繁度计数。

扫描所有的候选节点集，得到频繁1-项集 $L_1(S_1)=\{(2,-,0):3;(3,-,0):3\}$。候选节点$(3,-,0)$虽然在 $T3$中出现了两次，但只计算一次。

4. 将候选节点$(2,-,0)$嫁接到子树 S_1上，得到新的频繁子树 $S_2=\{1,2,-1\}$，supp$=3$。

5. 构造新的投影数据库：

1)$T1,\rho^*=1,\Omega:(3,4,0)$；2)$T2,\rho^*=2,\Omega:(3,3,0)$；3)$T3,\rho^*=5,\Omega:(3,6,0)$。

6. 使用 C-矩阵进行频繁度计数。

扫描所有的候选节点集，得到频繁1-项集 $L_1(S_2)=\{(3,-,0):3\}$

7. 将候选节点$((3,-),0):3$嫁接到子树 S_2上，得到频繁子树 $S_3=\{1,2,-1,3,-1\}$，supp$=3$。

8. 构造新的投影数据库：

1)$T1,\rho^*=4,\Omega:(4,5,1)$；2)$T2,\rho^*=3,\Omega:(4,4,1)$；3)$T3,\rho^*=6,\Omega:(4,7,1)$。

9. 使用 C-矩阵进行频繁度计数。

扫描所有的候选节点集，得到频繁1-项集 $L_1(S_3)=\{((4,-,1):3\}$

10. 将候选节点$((4,-),1)$嫁接到子树 S_3上，得到频繁子树 $S_4=\{1,2,-1,3,4,-1,-1\}$，supp$=3$。

11. 构造新的投影数据库：

1)$T1,\rho^*=5,\Omega$:null；2)$T2,\rho^*=4,\Omega$:null；3)$T3,\rho^*=7,\Omega$:null。

12. 使用 C-矩阵进行频繁度计数。

扫描所有的候选节点集，得到频繁1-项集 $L_1(S_4)=\{\phi\}$。步骤4至此结束。

13. 其它与此类似进行递归。

4.2 基于投影的频繁子树挖掘算法 PFTM

输入 树数据库 D，最小支持度阈值 min_sup
输出 频繁子树完全集
算法逻辑：
(1)扫描原始数据库 D，得到频繁1-项集 L_1。
(2)for each $v(l,-,0)\in L_1$ do {
　构造频繁1-子树 S_1，并输出 S_1及其支持度计数；
　构造基于 S_1的投影数据库 $D|_{S_1}$；
　call ProjFtm$(S_1,1,D|_{S_1})$;}

Procedure ProjFtm$(S_k,k,D|_{S_k})$
参数：S_k：频繁 k-子树；
　k：频繁子树的大小；
　$D|_{S_k}$：基于频繁子树 S_k 的投影数据库。
算法逻辑：
(1)扫描投影数据库 $D|_{S_k}$中所有的候选节点集，并使用 C-矩阵进行候选节点的支持度计数，从而得到基于 S_k 的频繁1-项集 $L_1(S_k)$。
(2)如果基于 S_k 的频繁1-项集为空，即 $L_1(S_k)=\Phi$，则返回。
(3)否则，逐个处理频繁1-项集 L_1中的每个节点。对于节点 $v(l,-,\tau)\in L_1(S_k)$。对于节点 $v(l,-,\tau)$：
(i)将 $v(l,-,\tau)$嫁接到频繁 k-子树 S_k 上作为新的最右节点，得到一棵频繁$(k+1)$-子树 S_{k+1}，并输出 S_{k+1}及其支持度计数。
(ii)构造基于 S_{k+1}的投影数据库 $D|_{S_{k+1}}$，递归调用 $ProjFtm(S_{k+1},k+1,D|_{S_{k+1}})$。

5 实验分析

为了对 PFTM 算法进行性能测试，我们将 PFTM 和 FREQT 进行了对比实验。实验是在 PC 机上进行，CPU 是 Pentium 1.8GHz，内存为512MB。我们利用了和文[9]同样的随机树生成器来构造原始数据库。随机树生成器模拟的是用户对网站的浏览行为，而挖掘到的频繁子树则反映了用户的兴趣模式。随机树生成器的主要控制参数包括：节点标签总数 N，树的最大深度 D，最大子节点树 F，数据库中的树的总数 T，树的平均深度 avg_depth，平均子节点数 avg_fanout。缺省参数取：$N=20,D=20,F=20$。

5.1 性能对比

性能对比实验采用三个不同的数据库。数据库 $D11$的控制参数为：$T=5000,avg\_depth=12.0196,avg\_fanout=1.36076$。$D11$的文件大小为460kB。数据库 $D12$的控制参数为：$T=10000,avg\_depth=11.9916,avg\_fanout=1.35322$。$D12$的文件大小为912kB。数据库 $D13$的控制参数为：$T=100000,avg\_depth=8.08645,avg\_fanout=1.30104$。$D13$的文件大小为6018kB。

实验结果表明，PFTM 和 FREQT 得到的频繁子树完全集是一样的。在运行效率方面，从图3可以看出，PFTM 平均要比 FREQT 高出40%左右。实验也进一步验证了以上的分析，FREQT 算法需要重复多次地扫描子树最右路径上节点的子节点集，这导致了算法效率的降低。而 PFTM 利用历史知识来压缩扫描空间，能够有效地提高算法的效率。同时，还可以看出，无论 PFTM 或是 FREQT，算法运行时间和支持度阈值都构成一定的线性比例关系。随着支持度阈值的调低，算法的运行时间不会急剧增加。在图3 $D12$中，当设定支持度阈值 $min_sup=8\%$时，挖掘到的频繁子树个数为6797，PFTM

的运行效率要比 FREQT 高出36%。当设定支持度阈值降到 $min\_sup=0.5\%$ 时，挖掘到的频繁子树个数为90672，PFTM 的运行效率要比 FREQT 高出43%。这说明，随着数据库中需要挖掘的频繁子树个数的增多，PFTM 对 FREQT 的效率优势会更为明显。

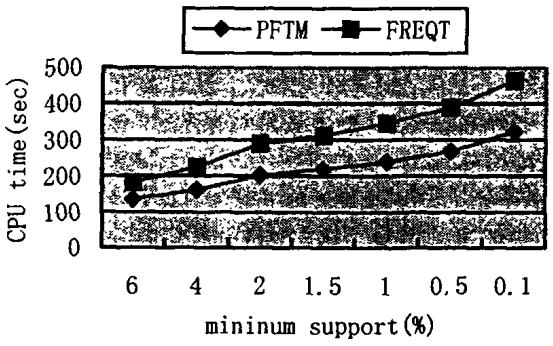

图3 性能比较(D11)

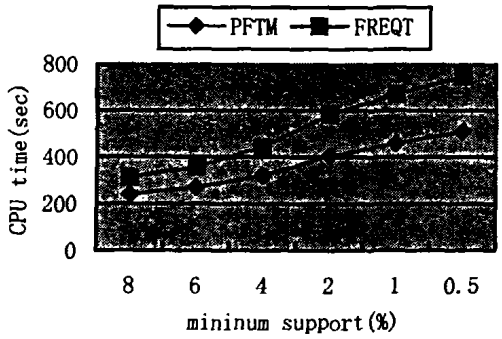

图3 性能比较(D12)

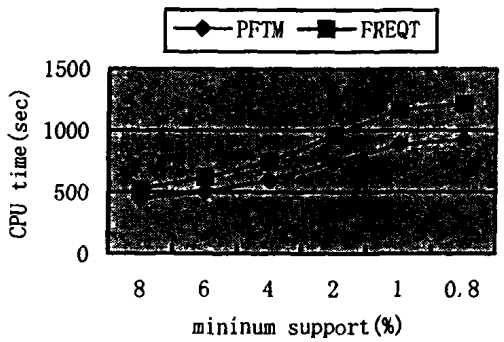

图3 性能比较(D13)

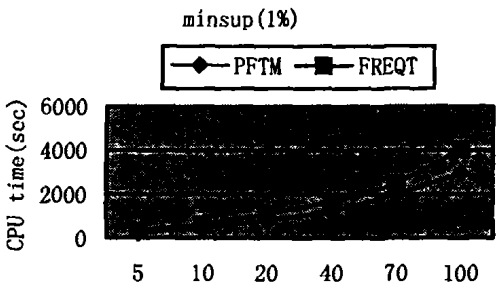

图4 可伸缩性比较

5.2 可伸缩性对比

可伸缩性实验采用六个不同的数据库。缺省参数为：$N=20$，$D=20$，$F=20$。数据库 $D21\sim D26$的树的总数 T 分别为：5000，10000，20000，40000，70000，100000。实验结果如图4所示。从图中可以看出，PFTM 和 FREQT 都呈现了很好的可伸缩性。算法运行时间和数据库中树的总数都构成一定的线性比例关系。在运行效率上，PFTM 也要比 FREQT 平均要高出42%左右。

结论 本文提出了一种新的基于投影的频繁子树挖掘算法，通过对数据库和候选节点集进行投影，并采用递推式候选节点集更新技术来有效地压缩扫描空间，从而高效地从森林中挖掘出频繁子树。性能对比实验表明，PFTM 是有效和可扩展的。

频繁子树挖掘可以应用到很多领域，如 Web 使用挖掘、Web 结构挖掘、XML 文档处理、生物信息处理等。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。因此，频繁子树在这些领域的应用会日趋广泛。

致谢 感谢 T. Kudo 博士提供 FREQT 的源码。感谢 M. J. Zaki 博士提供 TreeMiner 及随机树产生器的源码。

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 Conf. on Principles of Knowledge Discovery and Data Mining, Sep. 2000

5 Kuramochi M, Karypis G. Frequent subgraph discovery. In: 1st IEEE Int'l Conf. on Data Mining, Nov. 2001

6 Cook D, Holder L. Substructure discovery using minimal description length and background knowledge. Journal of Arti_cial 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, Arimura H, Satamoto H, Arikawa S. Effecient substructure discovery from large semi-structured data. In: 2nd SIAM Int'l. Conf. on Data Mining, April 2002

9 Zaki M J. Efficiently mining frequent trees in a forest. In SIGKDD'2002 Edmonton, Alberta, Canada

10 Agrawal R, Srikant R. Fast algorithms for mining association rules. In VLDB'94, Santiago, Chile, Sept. 1994. 487～499

11 Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. In SIGMOD'00, Dallas, TX, May 2000

12 Han J, Pei J, Mortazavi-Asl B, et al. FreeSpan: Frequent pattern-projected sequential pattern mining. In: Proc. 2000 Int. Conf. Knowledge Discovery and Data Mining (KDD'00), Boston, MA, Aug. 2000. 355～359

13 Pei J, Han J, Mortazavi-Asl B, et al. PrefixSpan: Mining Sequential Patterns Efficiently by PrefixProjected Pattern Growth. In: Proc. 2001 Int. Conf. Data Engineering (ICDE'01), Heidelberg, Germany, April 2001. 215～224