动态数据库中的频繁子树挖掘算法

郭 鑫[1] 董坚峰[1,2] 周清平[1]
(吉首大学信息管理与工程学院 张家界 427000)[1] (武汉大学信息资源研究中心 武汉 430072)[2]

摘 要 针对动态数据库随时间发生改变的特性,提出了一种新的在动态数据库中挖掘频繁子树的算法,引入树的转变概率、子树期望支持度和子树动态支持度等概念,提出了动态数据库中的支持度计算方法和子树搜索空间,从而解决了数据动态变化的频繁子树挖掘问题。随着子树搜索的进行,算法定义裁剪公式和混合数据结构,能有效地减少子树搜索空间和提高频繁子树的同构速度。实验结果表明,新算法有效可行,且具有较好的运行效率。

关键词 数据挖掘,有序树,频繁子树,支持度,动态数据库
中图法分类号 TP311 **文献标识码** A

Mining Frequent Subtrees from Dynamic Database

GUO Xin[1] DONG Jian-feng[1,2] ZHOU Qing-ping[1]
(School of Information Management and Engineering, Jishou University, Zhangjiajie 427000, China)[1]
(Center for Studies of Information Resources of Wuhan University, Wuhan 430072, China)[2]

Abstract On account of dynamic database's characteristic which is changing over time, a new algorithm aiming to mine frequent subtree from dynamic database was proposed. It put forward the support algorithm and subtree-searching space involving some concepts such as tree change probability, subtree expectation support and subtree dynamic support. The problem of mining frequent subtree from dynamic database was investigated. With the process of the subtree-searching, algorithm definition pruning expressions and mix data structure could reduce subtree-searching space and improve frequent subtree isomorphism speed efficiently. The experimental result showed that the new algorithm is effective and workable and has a better operating efficiency.

Keywords Data mining, Ordered tree, Frequent subtree, Support, Dynamic database

1 引言

结构化数据挖掘问题是数据挖掘领域中的热点研究问题,树模式挖掘作为其中的一种,已经引起了广大专家学者的关注。在现实世界中存在着许多实际应用,例如,酶作为化合物的一种,包括水解酶、氧化酶等不同种类,将不同种类的酶分子转化成树型结构并组成一个树数据库,运用已有的频繁子树挖掘算法可以得到不同种类的酶分子相同或相似的拓扑结构,这样便于发现不同种类酶之间的功能异同点。然而,在某些情况下,事物的结构可能会随着环境的变化而发生改变,酶就是其中一种,其结构随着外界温度的变化很容易发生改变,并且在不同的温度下,酶分子结构的改变程度也不同。在一定的时间段内,不同的酶分子可能会转变为多种不同形式的结构,在这种情况下,如何发现不同酶分子之间的特性已成为一个新的研究问题。由于现有的树挖掘算法只能挖掘静态数据库,即只能挖掘某时间点上的数据,并不能挖掘某时间段内数据的共有特性,因此传统的树挖掘算法已经无法满足现实变化的需要。

近年来,国内外的专家学者对传统的频繁子树挖掘算法进行了大量的研究,早在2002年,Zaki等人就提出了TreeMiner[1]算法;2004年,Y. chi提出了树挖掘混合算法HybridTreeMiner[2];同年,朱永泰提出了ESPM[3]算法;2006年,赵传申等提出FTPB[4]算法;2009年,李云等提出了无序树挖掘算法UnorderTreeMiner[5];文献[6]与文献[7]提出了特殊树结构挖掘算法,这些算法在静态数据库挖掘过程中具有较好的运行效率。在动态数据库挖掘方面,2006年,Backstrom L研究了社会网络随时间变化的改变规律[8];2007年,Tantipathananandh C研究了如何挖掘动态图中的社团结构及变化模式[9];2009年至2010年,邹兆年分别提出了不确定图挖掘[10]和挖掘图的演变模式[11]两个算法。然而这些算法只能寻找在某领域中数据随时间变化的规律,并不能解决在一定时间段内数据共同特性的挖掘问题。

本文主要研究在动态数据库中挖掘频繁子树,以解决在给定有效时间段内挖掘子树模式的问题。存在的解决方法有:在有效时间段内设置n个时间点,分别在每个时间点上产生一个树数据库,然后运用已有的树挖掘算法对每个时间点上的数据库进行挖掘,汇总得到频繁子树,并根据用户给定的最小支持度阈值淘汰非频繁的子树,最终可以得到在此时间段内所有的频繁子树,显然这个方法时间复杂度太高,在实际应用中并不可行。因此本文提出一种新的在动态数据库中挖

到稿日期:2010-06-17 返修日期:2010-11-12 本文受国家自然科学基金项目(70573082),教育部重点研究基地重大项目(08JJD870225)资助。
郭 鑫(1984—),男,硕士,助教,主要研究方向为数据挖掘、并行计算,E-mail:jianghai079@126.com;董坚峰(1977—),男,博士生,讲师,主要研究方向为计算机信息系统工程、知识挖掘;周清平(1966—),男,博士,教授,主要研究方向为量子计算。

掘频繁子树的算法，该算法能有效地解决在时间段内数据挖掘的问题。提出树的转变概率、包含集转变概率、子树期望支持度和子树动态支持度等概念，提出子树裁剪公式和一个线性表哈希表混合数据结构，其能有效地减少子树同构次数，在一定程序上提高了算法的运行效率。大量实验结果表明，本文的算法具有较好的可行性、有效性及运行效率。

2 问题定义

考虑到树的结构化特性对算法设计的复杂性影响较大，故本文将研究重点集中在常见的有序标号树。在详细介绍动态数据库中频繁子树挖掘算法之前，首先给出有关概念和问题的定义。

定义 1(有序标号树[1]) 有序标号树 $T=(V,E)$是一个具有标号和根结点的有向无环图，其中 V 表示树结点集合，$E=\{(x,y)|x,y\in V\}$表示边集合，用 $L=\{l_0,l_1,\cdots,l_n\}$ 表示一个标号集，结点 V 与 L 存在对应关系 $l:V\to L$，l_i 称为结点标号。有序是指树中结点按照从左至右的顺序形成兄弟关系。

定义 2(转变树) 若树 $T=(V,E)$表示时刻 t 下的有序标号树，则在时刻 t'下存在另一个树 $T'=(V',E')$，当满足条件：(1)$V'=V$；(2)$E'=E-E_1$ 或 $E'=E\cup E_2$，其中 $E_1\subseteq E$，$E_2\subseteq(V\times V)-E$时，则称 T'为时刻 t'下由 T 转变得到的树，记为 $T\to T'$，其中 V'和 T'分别是树 T'的顶点集和边集。

从定义 2 可以看出，若 T 在某时间段内改变自身结构，如增加一些边或减少一些边，则树结构 T'为 T 转变得到的树。如图 1 所示，在时间段 $t1\to t5$ 内，树结构随时间发生变化，在每一时刻所产生的树结构不同。在现实环境中，树的转变可能是有规律的，也可能是无规律的，如何表示树的转变规律是本文需要解决的问题。本文只考虑在树结点数不变的情况下，树的边发生改变，因此，树的转变问题可以转化为边的转变问题。

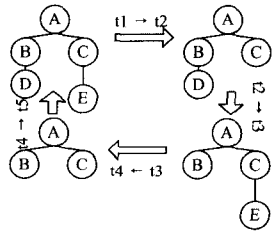

图 1 转变树产生图

定义 3(边的转变概率) 树 $T=(V,E)$为时刻 t 下的有序标号树，定义 $P:E\to(0,1]$为边集中每条边转变的可能性函数，称函数 P 是边的转变概率，包含函数 P 的树 $T^*=(V,E,P)$称为动态树。

由上述定义可知，动态树是一种树边带权值的树结构，边的权值即为边转变概率。对于同一棵动态树，在不同的时刻所产生的转变树可能不同，因此，在某时间段内，动态树 T^*是一组转变树集，有如下定义。

定义 4(转变树包含集) 给定转变树 T 和动态树 T^*，若满足 $T^*\to T$，则称 T^* 包含 T，动态树 T^* 包含全部转变树的概率分布为 $S(T^*)=\{T_1,T_2,\cdots,T_n\}$，称之为转变树包含集，其中$\{1,2,\cdots,n\}$表示时间段内 n 个时间点。

定义 5(动态树数据库) 令 $TDB^*=\{T_1^*,T_2^*,\cdots,T_n^*\}$表示一个动态树数据库，采用元组$(TID,String)$表示树结构，$TID$ 是树在数据库中的 ID，$String$ 是动态树的字符串表示，在某时间段内，每个动态树都对应一个转变树包含集。

根据上述定义，动态数据库中的频繁子树挖掘问题可以定义为：在给定动态数据库、最小支持度阈值和有效时间段的情况下，运用动态数据库中的频繁子树挖掘算法，找出所有的频繁子树模式。

3 动态数据库中的频繁子树挖掘算法

3.1 子树动态支持度

在传统的频繁子树挖掘算法中，树的支持度定义一般采用相对支持度和绝对支持度两种形式，然而这样的定义只适用于在静态数据库中挖掘频繁子树。动态数据库挖掘具有与传统数据库挖掘不同的特点：动态性，在有效的时间段内，数据库中的树结构随时间变化而改变，每个时刻产生的树结构可能相同，也可能不同，转变树的产生是不确定的，因此传统的支持度定义方式在动态数据库挖掘中没有意义，本文需要根据树的动态变化特征重新定义动态支持度。

由定义 4 可知，转变树包含集是在有效时间段内不同时刻所产生的转变树集合，转变树的边出现可能性为边转变概率，在某时刻，转变树为所有边出现可能性皆为 1 的特殊树。如何计算在某时刻转变树出现的概率，本文给出以下定义。

定义 6(树的转变概率) 给定一个动态树 $T^*=(V^*,E^*,P^*)$和一个转变树 $T=(V,E)$，在时刻 t 下转变树 T 出现的概率为：

$$P((T^*\Rightarrow T)|t)=\prod_{e\in E}P(e)\cdot\prod_{e\in E^*-E}(1-P(e))$$

本文假设转变树中不同边的存在与否是相互独立的，其中 $P(e)$表示边的转变概率。令 $S(T^*)=\{T_1,T_2,\cdots,T_n\}$为转变树包含集，根据概率统计可得如下定理。

定理 1 对于动态树 T^*，函数 $P((T^*\Rightarrow T)|t)$定义了在样本空间 $S(T^*)$上的概率分布。

定义 7 包含集转变概率。给定动态树数据库 TDB^*和转变树包含集 $S(T^*)$，在有效时间段 $t_e=(t\to t')$内 $S(T^*)$出现的概率为：

$$P((TDB^*\Rightarrow S(T^*))|t_e)=\prod_{i=1}^{n}P(T_i^*\Rightarrow T_i)$$

令 $S(TDB^*)$表示动态树数据库下所有 $S(T^*)$的集合，可得到如下定理。

定理 2 对于动态树数据库 TDB^*，函数 $P((TDB^*\Rightarrow S(T^*))|t_e)$定义了在样本空间 $S(TDB^*)$上的概率分布。

根据以上分析，在有效时间段 t_e 内，子树 T'在动态树数据库 TDB^* 中的支持度是一个概率分布：

$$\begin{pmatrix} s_1 & s_2 & \cdots & s_n \\ P(s_1|t_e) & P(s_2|t_e) & \cdots & P(s_n|t_e) \end{pmatrix}$$

式中，$s_1,s_2,\cdots,s_n$ 为子树 T'在转变树包含集 $S(T^*)$中的概率，$P(s_i|t_e)$为支持度 s_i 对应的概率。

定义 8(子树期望支持度) 给定动态树数据库 TDB^*，在有效时间段 t_e 内子树 T'在 TDB^* 中的期望支持度为：

$$ES_{TDB^*}(T'|t_e)=\sum_{i=1}^{n}S_i\cdot P(S_i|t_e)$$

或为：

$$ES_{TDB^*}(T'|t_e)=\sum_{q\in w}S_q(T')\cdot P((TDB^*\Rightarrow q)|t_e)$$

式中，q 为转变树包含集 $S(T^*)$，w 为动态树集 $S(TDB^*)$。子树 T'在动态树 T^* 中出现，当且仅当 T'子树同构于至少一个 $S(T^*)$所包含的子树，因此在时间段 t_e 内，T'在 T^* 中出现的概率为：

$$P((T'\subseteq T^*)|t_e)=\sum_{k\in S(T^*)}P(T^*\Rightarrow k)\cdot\Phi(k,T')$$

式中，k 为 $S(T^*)$ 中的转变树，$\Phi(k,T')$ 为布尔函数，若子树 T' 在 k 中出现则函数值为 1，否则为 0。综上所述，子树期望支持度公式可定义为：

$$ES_{TDB^*}(T'|t_e)=\frac{1}{|TDB^*|}\sum_{i=1}^{|TDB^*|}P((T'\subseteq T^*)|t_e)$$

定义 9（子树动态支持度） 对于给定的动态数据库 TDB^*，子树 T' 在有效时间段 t_e 内的期望支持度为 $ES_{TDB^*}(T'|t_e)$，则称 $ES_{TDB^*}(T'|t_e)$ 为子树 T' 的动态支持度。

根据定义，子树在动态数据库 TDB^* 中的转变概率满足 Apriori 性质，同时子树在 TDB^* 中的动态支持度也满足相同性质，即对于任意两个子树 T_1 和 T_2，若 T_1 是 T_2 的子树，则 $ES_{TDB^*}(T_1|t_e)\geqslant ES_{TDB^*}(T_2|t_e)$。因此根据 Apriori 的逆反性质：不频繁模式的任何超都是不频繁的，可以有效地降低频繁子树挖掘算法的复杂度。

3.2 频繁子树挖掘算法基本思想

在给定动态树数据库 TDB^* 的情况下，TDB^* 中的子树直接父模式关系构成一个有向无环图。子树扩展方式可以分为结点扩展和边扩展两种方式，有向无环图的顶点为初始搜索结点或边，子孙结点为上层结点或边的扩展模式，此有向无环图可称之为子树的搜索空间。在边或结点的扩展过程中，搜索空间可以划分为 n 个互不相交的子搜索空间。

频繁子树挖掘算法的关键在于如何确定子树搜索空间，子树搜索空间的质量将直接影响到挖掘算法的性能。TreeMiner算法是一个具有较好性能的传统频繁子树挖掘算法，该算法基于叶子结点扩展的子树生成策略，采用深度优先搜索的方式形成一个子树搜索空间，且满足无冗余、无遗漏等特点，但是，子树生成策略所需满足的条件较多，算法定义的候选生成规则较复杂，并且此候选生成策略只能应用于传统的频繁子树挖掘中，对动态数据库挖掘并不适合。动态数据库中的频繁子树挖掘具有时效性，树结构随时间发生改变，考虑到转变树结点是连接边发生改变而形成的，树中每条边都对应着一个转变概率，因此本文吸取 TreeMiner 算法的优势并改进子树生成策略，采用最右边扩展的方式，深度优先搜索子树形成子树搜索空间，以枚举得到全部的频繁子树。

在给定的时间段 t_e 内进行深度优先搜索，对于每个子搜索空间，算法从第一层开始搜索，不断进行边的扩展生成候选子树 T，同时计算 T 在 TDB^* 的支持度 $ES_{TDB^*}(T|t_e)$，若 $ES_{TDB^*}(T|t_e)\geqslant$minsup，则 T 是频繁的，将 T 加入频繁子树集中并继续深度优先搜索 T 的所有超模式；若 $ES_{TDB^*}(T|t_e)<$ minsup，则 T 是非频繁的，可根据 Apriori 的逆反性质，停止对 T 的搜索并返回到搜索空间中上一次访问过的父结点。搜索结束条件为：当深度优先探索返回到该子空间的第一层且所有的子树模式都已被访问时，搜索停止。本文研究的树结构是有序标号树，具有有向无环的特性，生成的子树搜索空间中不会产生冗余子树，且边的扩展也能保证不会遗漏子树。

在动态支持度公式中，计算子树 T' 的动态支持度 $ES_{TDB^*}(T'|t_e)$ 需要计算 T' 在动态树中的概率 $P((T'\subseteq T^*)|t_e)$，基于子树搜索策略，设有两个子树 T_1 和 T_2，且 T_2 是 T_1 的子树，显然 $P((T_2\subseteq T^*)|t_e)$ 与 $P((T_1\subseteq T^*)|t_e)$ 相比，前者要优先得到，根据这一特性，定义裁剪公式：

$$ES_{TDB^*}(T_1|t_e)\leqslant\frac{1}{|TDB^*|}\sum_{i=1}^{k}P((T_1\subseteq T^*)|t_e)+\frac{1}{|TDB^*|}\sum_{i=k+1}^{|TDB^*|}P((T_2\subseteq T^*)|t_e)$$

式中，$k=\{0,1,\cdots,|TDB^*|\}$，公式右边是子树 T_1 动态支持度的上限，若上限小于最小支持度阈值，显然 T_1 是非频繁的，因此可以利用先验知识来判断子树是否频繁，从而有效地裁剪搜索空间，在一定程度上提高了算法的运行效率。

3.3 算法设计

算法利用哈希表快速匹配的特性，设计一个线性表连接哈希表的混合数据结构，线性表中每一个元素指向一个哈希表结构，相应的哈希表中存储频繁子树信息，如：线性表第 k 个元素指向第 k 个哈希表，同时在第 k 个哈希表中存储频繁 k 子树。利用该结构既能方便存储搜索得到的频繁子树，又能较快地获得已产生的频繁子树，实现在候选子树生成过程中的子树剪枝操作。

在时间段 t_e 内，算法首先扫描动态数据库 TDB^* 得到频繁一子树，将得到的频繁一子树放到对应的哈希表结构中。然后根据子树生成策略依次对每个频繁一子树进行扩展，生成候选二子树，同时计算候选子树的动态支持度，若大于用户给定的最小支持度阈值，则将此子树放入哈希表结构中。在计算 $k(k\geqslant 3)$ 子树动态支持度之前，应根据裁剪公式判断该子树是否频繁，如果非频繁则停止搜索，如果不能判断则根据动态支持度公式来进行判断。然后按照同样的策略，深度搜索其它子树，直到产生所有的频繁子树，算法结束。算法的伪代码如下。

算法 1 DynamicTreeMine

输入：动态树数据库 TDB^*，最小支持度 s，时间段 t_e；

输出：TDB^* 中所有频繁子树。

1)初始化线性表哈希表混合数据库结构 LH=NULL；

2)扫描动态数据库得到频繁一子树并放入 LH 中；

3)FOR 每个 $k(k=1)$ 子树 T DO

4) IF k 子树，$k\geqslant 3$ THEN 根据裁剪公式计算上限；

5) IF 上限$<s$ THEN Continue；

6) 根据公式计算动态支持度 ES；

7) IF $ES\geqslant s$ THEN 将 k 子树加入到 LH 中；

8) 对 k 子树进行扩展得到超子树；

9) FOR 每个超子树($k=k+1$) DO

10) 返回第 4)步执行；

11)END FOR

12)输出 LH；

算法第 4)步至第 10)步实际上是一个递归过程，对应于深度优化搜索候选子树，所产生的频繁子树将保存在 LH 中，并最终输出出来。

4 实验与结果分析

为验证算法的可行性及执行效率，我们采用 C 语言实现算法并进行大量实验，实验环境为 Intel(R) Core(TM) 2 Quad CPU，4GB 内存，操作系统为 Ubuntu Linux。

实验数据采用通用的树数据库生成器生成模拟数据，该生成器共有 8 个参数来控制树的生成，分别为：标号集大小 S，子节点生成概率 p，基本树的个数 L，基本树的高度 I，基本树每个节点的扇出 C，数据库大小 N，树的最大高度 H，每个节点的扇出 F，树高度遵循期望为 $I(H)$、标准差为 1 的高斯分布。为适应动态数据挖掘的需要，修改树生成器源码并增加第 9 个参数：边的转变概率 P，以模拟边随时间发生的改变。

实验分为三组，第一组实验考察在数据库动态变化的情况下，算法的可行性以及与传统算法相比的性能优势；第二组实验在不同的支持度阈值下进行，以验证算法的执行效率；第三组实验考察边的转变概率 P 发生改变时，算法的执行时间，进一步从另一个角度来验证算法的运行效率。

在第一组实验中，数据库生成器参数设置为 S10 p0.2 L6 I4 C3 N1000 H6 F6 P0.5，在给定的有效时间段内，设置 n 个时间点并产生相应的转变树数据库，分别采用 TreeMiner 算法和 DynamicTreeMine 算法来挖掘频繁子树，实验结果如图 2 所示。实验结果说明，在动态数据挖掘中，本文算法具有较好的可行性，且执行效率要好于传统算法。

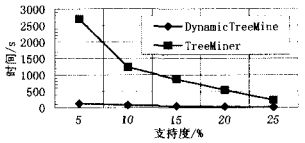

图 2　TreeMiner 算法与 DynamicTreeMine 算法的对比

第二组实验采用树数据生成器生成 6 个树数据库：D1 至 D6，参数设置如表 1 所列。

表 1　树数据生成器设置表

DATA SET	PARAMETER
D1	S10 p0.3 L6 I4 C3 N1000 H4 F3 P0.5
D2	S10 p0.3 L6 I5 C5 N1000 H4 F3 P0.5
D3	S10 p0.3 L6 I6 C5 N1000 H4 F3 P0.5
D4	S10 p0.3 L6 I7 C6 N1000 H4 F3 P0.5
D5	S10 p0.3 L6 I8 C7 N1000 H4 F3 P0.5
D6	S10 p0.3 L6 I8 C8 N1000 H4 F3 P0.5

实验结果如图 3、图 4 所示，该实验可以说明，在不同的动态树数据库下，本文算法运行效率较好，运行时间随最小支持度的增加而显著减少，因为随着支持度的增加，产生的频繁子树显著减少，算法所需进行的子树同构测试也将显著减少。

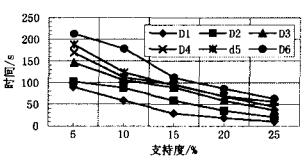

图 3　最小支持度变化与运行时间的关系

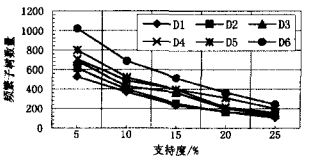

图 4　最小支持度变化与频繁子树数量的关系

第三组实验在支持度(5%)保持不变的情况下，对表 1 设置进行修改，前面 8 个参数设置保持不变，修改边的转变概率 P 依次为：0.5，0.6，0.7，0.8，0.9。实验结果如图 5 所示，可以说明，随着边的转变概率的增加，算法运行时间也在增加，因为在边转变概率增加的同时，转变树的边也将增多，频繁子树的模式也将增加，因此算法时间也相应地增加。

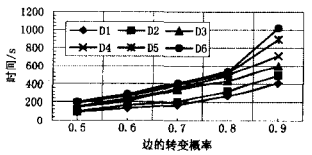

图 5　边的转变概率与运行时间的关系

结束语　本文在传统树模式挖掘算法的基础上提出了一种新的频繁子树挖掘算法，提出了边的转变概率、包含集转变概率、子树期望支持度和子树动态支持度等概念，及将子树的期望支持度作为子树动态支持度来判断子树是否频繁，很好地解决了在有效时间段内，当树数据库随时间发生改变时，如何挖掘频繁子树的问题。算法基于最右边扩展，深度优先生成候选子树，形成了一个子树搜索空间，同时设计了一个线性表哈希表混合数据结构来存储频繁子树。在搜索过程中，首先根据裁剪公式计算动态支持度上限，若上限小于最小支持度阈值，则该子树是非频繁的，从而有效地裁剪了搜索空间，否则按照动态支持度公式计算动态支持度，并与最小支持度进行比较，判断该子树是否频繁，最后将产生的所有频繁子树保存到混合数据结构中。大量实验结果表明，该算法有效可行，且具有较好的运行效率。下一阶段的工作可以考虑将更多的数据类型应用于动态数据挖掘中，提供一个通用的动态数据挖掘的解决方法。

参 考 文 献

[1] Zaki M J. Efficiently mining frequent trees in a forest: Algorithms and applications[J]. IEEE Transaction on Knowledge and Data Engineering: Special Issue on Mining Biological Data, 2005, 17(8): 1021-1035

[2] Chi Y, Yang Y, Muntz R R. HybridTreeMiner: An efficient algorithm for mining frequent rooted trees and free trees using canonical forms[C]// Proceedings of 16th International Conference on Scientific and Statistical Database Management. Washington, DC: IEEE Computer Society, 2004: 11

[3] 朱永泰，王晨，洪铭胜，等. ESPM—频繁子树挖掘算法[J]. 计算机研究与发展，2004(10)：1720-1726

[4] 赵传申，孙志挥，张净. 基于投影分支的快速频繁子树挖掘算法[J]. 计算机研究与发展，2006，43(3)：456-462

[5] Li Yun, Guo Xin, Yuan Yun-hao. A efficient algorithm to mine unordered trees[C]// ICIS 2009: Proceedings of 8th IEEE/ACIS International Conference on Computer and Information Science. Shanghai, 2009: 331-336

[6] 杨沛，谭琦. 极大频繁子树挖掘及其应用[J]. 计算机科学，2008，35(2)：150-153

[7] 朱颖雯，吉根林. 一种高效的最大频繁 Embedded 子树挖掘算法[J]. 计算机科学，2007，34(12)：17-179

[8] Backstrom L, Huttenlocher D, Kleinberg J, et al. Group formation in large social networks: Membership, growth, and evolution[C]// Proceedings of the 12th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. New York: ACM Press, 2006: 44-54

[9] Tantipathananandh C, Berger-Wolf T Y, Kempe D. A framework for community identification in dynamic social networks [C]// Proceedings of the 13th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. New York: ACM Press, 2007: 717-726

[10] 邹兆年，李建中，高宏，等. 从不确定图中挖掘频繁子图模式[J]. 软件学报，2009，20(11)：2965-2976

[11] 邹兆年，高宏，李建中，等. 演变图上的连接子图演变模式挖掘[J]. 软件学报，2010，21(5)：1007-1019