基于最大频繁 Induced子树的 GML文档结构聚类

朱颖雯[1], 吉根林[2]

(1. 三江学院 计算机基础部, 江苏 南京 210012; 2 南京师范大学 数学与计算机科学学院, 江苏 南京 210097)

[摘要] 提出了一种基于最大频繁 Induced子树的 GML文档结构聚类新算法 TBCClustering. 通过挖掘 GML文档集合中的最大频繁 Induced子树构造特征空间, 并对特征空间进行优化; 采用 CLOPE聚类算法聚类 GML文档, 可自动生成最小支持度与聚类簇的个数, 无需用户设置; 不仅减少了特征的维数, 而且得到了较高的聚类精度. 实验结果表明算法 TBCClustering是有效的, 且性能优于 PBClustering算法.

[关键词] GML结构聚类, 最大频繁 Induced子树, 闭合频繁 Induced子树

[中图分类号] TP 391 [文献标识码] A [文章编号] 1672-1292(2008)04-0050-06

Clustering GML Documents by Structure Based on Maximal Frequent Induced Subtrees

Zhu Yingwen[1], Ji Genlin[2]

(1. Department of Computer Elementary Training, Sanjiang University, Nanjing 210012, China;
2. School of Mathematics and Computer Science, Nanjing Normal University, Nanjing 210097, China)

Abstract This paper presents an algorithm TBCClustering for clustering GML document structure based on maximal frequent subtree patterns. During the maximal frequent subtree mining process, it optimizes characteristic spaces, gets the minimum support automatically, chooses some subtree pattern to form the optimistic clustering features, and uses CLOPE algorithm to cluster documents by clustering features without giving the number of cluster. Not only the dimensions of features are reduced, but also the higher clustering precision is obtained. Experiment results show that TBCClustering is more effective and efficient than PBClustering.

Key words GML clustering by structure, maximal frequent subtrees, closed frequent subtrees

GML(Geography Markup Language)文档聚类与 XML文档聚类相似, 可分为基于文本内容[1,2]聚类与基于文档结构[3-9]聚类. GML文档结构聚类是一种树型聚类, 目前其相似度度量方法主要有两种: 一种是用编辑距离[3,4]来度量两篇 GML文档的相似度; 另一种是用 Jaccard系数[5,6]来度量 GML文档的相似度; 还有一些其它的相似度度量方法[7-9]. 相似度确定后, 通常使用层次聚类算法 HAC[10]或 ROCK[11]对文档进行聚类. 这两种层次聚类方法都需要事先确定聚类的簇的个数, 且在聚类过程中, 需对整个数据集进行两两相似度计算, 使得聚类的时间性能很差, 当聚类文档个数增加或者样本特征维数太大时尤为明显.

文献[7]提出了 XML结构聚类算法 PBClustering, 采用频繁路径作为聚类特征, 利用层次聚类算法 HAC对聚类特征进行聚类. 随着用户定义的最小支持度的改变, 特征个数变化, 聚类精度不稳定. 如何确定聚类簇的个数以及最小支持度成为该算法的瓶颈. 为此, 本文提出了一种基于最大频繁 Induced子树的 GML文档结构聚类算法 TBCClustering以解决此问题. 通过挖掘 GML文档集合中的最大频繁 Induced子树构造特征空间, 并对特征空间进行优化, 去除重复特征; 挖掘过程中自动生成较好的最小支持度, 无需用户设置; 最后采用 CLOPE算法[12]聚类 GML文档, 聚类过程中自动生成簇的个数. 实验结果表明, TBCClustering不仅能够得到维数较小的特征空间, 而且在聚类时间和精度上也高于 PBClustering算法.

收稿日期: 2008-06-18
基金项目: 国家自然科学基金(40771163)资助项目.
通讯联系人: 朱颖雯, 助教, 研究方向: 数据挖掘. E-mail: zhu_ying_wen@163.com

1 GML文档预处理

定义 **1** 有序标号树及其长度[13]. 令 $L = \{l_0, l_1, \quad\}$ 为一个标号集合, 则集合 L 上的有序标号树(简称树) $\boldsymbol{T} = (V, E, L, v_0)$ 是一个具有标号和根节点的有向无环图, 并且具有以下性质:

(1) $\boldsymbol{T}$ 中每一个节点 $v \quad V$ 用 L 中的元素标记为 $L(v)$.

(2) 除根节点 v_0 外所有节点有惟一的父节点并与其形成父子关系 $E \quad V^2$.

(3) 对于树中的每一个节点, 其所有子节点按照从左到右的顺序形成兄弟关系.

树 $\boldsymbol{T}$ 的长度定义为树 $\boldsymbol{T}$ 中节点的个数, 记为 $|\boldsymbol{T}|$.

定义 **2** 树的先序序列表示[13]. 设有序标号树 $\boldsymbol{T}$ 对应的先序序列 S 初始为空, 根据深度优先遍历顺序, 每访问一个节点将该节点标号加入到 S 中, 当从一个孩子节点返回到它的父节点时同时添加 - 1到 S 中.

GML文档预处理包含以下 3步:

(1) 将每篇 GML文档解析成 DOM树.

(2) 最小化 DOM树得到 GML文档的最小结构[14], 并将每个 DOM树中的元素节点(不包含值及属性节点)的标号用数字表示, 得到有序标号树的集合. 所谓最小化 DOM树, 就是去除它的重复节点, 重复节点一般包括两种, 一种是叶子节点, 另一种是非叶子节点.

(3) 将每棵有序标号树用先序序列表示. 例如, 对于图 1所示的有序标号树, 其先序序列表示为 1 2 3 - 1 4 - 1 5 6 7 - 1 - 1 - 1 - 1.

2 特征提取及表示

定义 **3** Induced子树[15]. 给定有序标号树 $\boldsymbol{T} = (V, E, L, v_0)$, $\boldsymbol{T} = (V, E, L, v_0)$. 若满足以下条件, 则称树 $\boldsymbol{T}$ 是树 $\boldsymbol{T}$ 的 Induced子树.

$V \quad V$, $E \quad E$.

L 保持 $\boldsymbol{T}$ 中 V 的标号.

若节点 $v_1, v_2 \quad V$, $v_1, v_2 \quad V$, 且先序遍历 $\boldsymbol{T}$ 时 v_1 在 v_2 之前, 则先序遍历 $\boldsymbol{T}$ 时, v_1 也在 v_2 之前.

若节点 $v_1, v_2 \quad V$, $v_1, v_2 \quad V$, 且在 $\boldsymbol{T}$ 中 v_1 是 v_2 的双亲, 则在 $\boldsymbol{T}$ 中 v_1 是 v_2 的双亲.

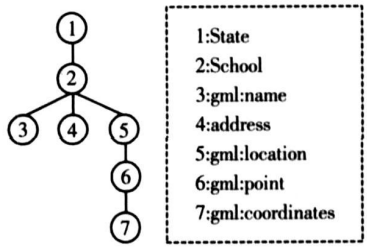

图 1 有序标号树

Fig.1 Ordered labeled tree

若树 $\boldsymbol{T}$ 是树 $\boldsymbol{T}$ 的 Induced子树, 那么树 $\boldsymbol{T}$ 就是树 $\boldsymbol{T}$ 的超树. 本文以下讨论的子树均为 Induced子树.

定义 **4** 支持度与频繁子树. 给定有序标号树数据库 TDB 以及子树 $\boldsymbol{T}$, $\boldsymbol{T}$ 的支持度定义为 $\mathrm{Support}(\boldsymbol{T}) = |P(\boldsymbol{T})| / |TDB|$, 其中 $P(\boldsymbol{T})$ 是 TDB 中包含 $\boldsymbol{T}$ 的树的集合, $|P(\boldsymbol{T})|$ 是 T 的支持数. 当 $\mathrm{Support}(\boldsymbol{T}) \quad minsup$ ($0 \quad minsup \quad 1$) 时, 称 $\boldsymbol{T}$ 是 TDB 中的频繁子树. 其中, $minsup$ 为用户指定的支持度阈值. 若 $\boldsymbol{T}$ 频繁且它的任何超树均不频繁则称 $\boldsymbol{T}$ 为最大频繁子树. 若 $\boldsymbol{T}$ 频繁且它的任何超树的支持度均小于 $\boldsymbol{T}$ 的支持度则称 $\boldsymbol{T}$ 为闭合频繁子树.

闭合频繁子树的超树可能是频繁的, 但最大频繁子树的超树一定是不频繁的. 当闭合频繁子树 $\boldsymbol{T}$ 的支持度等于 $minsup$ 时, $\boldsymbol{T}$ 就是一棵最大频繁子树.

GML文档经过预处理, 可得到一个有序标号树数据库, 从而将 GML文档的结构聚类问题转化为树的聚类问题. 一般来说属于同类的 GML文档对应的有序标号树必含有较多的相同子树结构, 而不同类的树则含有较少的相同子树结构. 因此, 可以将最大频繁子树作为聚类特征, 但需解决以下 3个问题:

(1) 如何选择一个合适的最小支持度挖掘有序标号树中的最大频繁子树?

(2) 当最小支持度较低, 同类树中含有较多相似的最大频繁子树, 如何去除重复特征从而得到最优特征?

(3) 如何用特征表示一个 GML文档有序树?

2.1 自动获得最小支持度

一个合适的最小支持度应该可以在 GML文档对应的有序标号树数据库中挖掘出代表每类文档的最

大频繁子树,例如,若一个 GML文档集合中有 4个类,每个类分别含有 50、100、150以及 200篇文档,当 *minsup*为 100时,含有 50篇文档的类将因为 *minsup*过大而得不到任何最大频繁子树,所以最适合的 *minsup*应为 50.然而聚类前,用户不知道每个类的文档个数,所以这个参数用户无法给出.设有序标号树数据库 *TDB*中出现的标号节点集合 $L = \{l_0, l_1, \cdots, l_n\}$,每个标号节点即一棵长度为 1的子树 t_i ($i = 1 \sim n$),满足条件 $P(t_i) = TDB$ 的那个最小的支持度可以近似为最合适的支持度.自动获得最小支持度的算法见算法描述 1.

算法 **1**　GetBestMinsupport

输入:有序标号树数据库 *TDB*,离群点出现概率 o ($0 \le o < 1$).

输出:最小支持度 *minsup*.

步骤:

```
(1) C = ∅;
(2) L = GetLFromTDB(TDB);        //得到 TDB 中所有标号节点 t_i 及 P(t_i)
(3) L = SortLbasedSupport(L);    //对 t_i 根据 |P(t_i)| 从大到小排序
(4) for(i = 1; i <= |L|; i++)
(5)     if(P(t_i) != TDB)
(6)     {C = C ∪ P(t_i);
(7)     if(|C|/|TDB| >= 1 - o) {minsup = |P(t_i)|/|TDB|; break;}}
```

2.2　最优特征提取

给定最小支持度 *minsup*,根据 CMTreeMiner算法[15]可以得到整个有序标号树数据库 *TDB*中的所有最大频繁子树以及闭合频繁子树.若将整个最大频繁子树集合选为聚类的特征,可能含有一些冗余信息.例如,对于包含 4个类的 GML文档集合(50/100/150/200),当 *minsup*为 50,有可能含有 200篇文档的类中挖掘得到了很多的最大频繁子树,而且这些最大频繁子树之间含有许多共同的结构,则认为这些特征是重复的,要将之去除.

算法 **2**　GetBestFeature

输入:有序标号树数据库 *TDB*, *minsup*, 特征优化阈值 θ.

输出:聚类特征集合 F.

步骤:

```
(1) M = ∅; tempM = ∅; tempTDB = ∅;
(2) CMTreeMiner(TDB, minsup, M);      //调用 CMTreeMiner算法得到最大频繁子树集合 M
(3) F = M;
(4) for(i = 1; i <= |M|; i++)
(5) {Ti = GetElement(M, i);           //得到 M 集合中的一棵树 Ti
(6)     for(j = i + 1; j <= |M|; j++)
(7)     {Tj = GetElement(M, j);
(8)     tempTDB = Ti ∪ Tj;
(9)     CMTreeMiner(tempTDB, 1, tempM);
(10)    Sim = GetMaxSubtreeLength(tempM);     //得到 Ti 与 Tj 中最大相同子树结构长度
(11)    if(Sim/max{|Ti|, |Tj|} >= θ)          //两个特征 Ti 与 Tj 很相似
(12)    if(|Ti| > |Tj|) { Delete(F, Ti); P(Tj) = P(Tj) ∪ P(Ti); i--; break; }
(13)    else { Delete(F, Tj); P(Ti) = P(Ti) ∪ P(Tj); j--; }}
(14)}
```

2.3　特征表示

若聚类特征集合 *F*中含有 *n*棵树 $f_1, f_2, \cdots, f_n$,则 GML文档的聚类特征用一个 *n*维向量 $(s_1, s_2, \cdots, s_k, \cdots, s_n)$ 来表示,其中, $s_k = \begin{cases} 1 & \text{若该文档对应用有序标号树包含 } f_k \\ 0 & \text{否则} \end{cases}$. $k = 1, 2, \cdots, n$.

若一个 GML文档对应的有序标号树 ***T***不包含 F中的任何树,则依次挖掘 ***T***与 f_i 中含有的最大共同子

树结构, 选择含有最大共同子树结构的 f_i, 并给 s_i 置 1. 若找不到这样一个特征 f_i, 则认为该文档是离群点.

3 TBCClustering 算法描述

传统的聚类方法如划分聚类或层次聚类都是建立在两两比较的局部准则函数上的. K-Means通过比较元素与最近簇的距离, 把它分到相似度最大的簇, 而凝聚的层次聚类是通过合并相似度最大的两个簇完成, 大量的两两相似度或距离的计算非常耗时, 当数据是高维时, 时间性能很差, 还需要事先确定聚类的个数, 而这在聚类过程中是很难确定的. 故 TBCClustering算法采用 CLOPE聚类算法[12] 聚类 GML文档. 聚类过程是通过一个评价聚类效果的全局准则函数实现的, 不需要计算两棵树间的相似度, 也不需要事先确定聚类个数, 聚类时没有维数的约束, 大大提高了聚类的时间性能, 而且聚类效果很好.

设文档 d_i 是簇 C_i 的一个元素, 由一组特征向量 $(s_1, s_2, \cdots, s_n)$ 表示, $|d_i|$ 表示该元素对应的特征向量中含有 1的个数, $S(C_i)$ 是簇中所有元素含有 1的个数 (见式 (1)), $W(C_i)$ 是簇里含有 1的特征的个数, $H(C_i)$ 代表簇的高度 (见式 (2)), 值越大, 代表簇内元素之间有很多的重复元素, 内聚性高. 对于一个聚类 $C = \{C_1, \cdots, C_k\}$, 评价 C 的好坏的全局准则函数如式 (3) 所示. 其中, r 是一个控制簇的内聚程度的参数, r 越大, 簇内元素的相似性越大, 簇之间的差异性越大. 聚类过程中把文档 d_i 分到能够使全局准则函数取得最大值的簇或自成一簇.

$$S(C_i) = \sum_{d_j \in C_i} |d_j|, \tag{1}$$

$$H(C_i) = S(C_i) - W(C_i), \tag{2}$$

$$\mathrm{Profit}_r(C) = \frac{\sum_{i=1}^{k} \frac{S(C_i)}{W(C_i)^r} |C_i|}{\sum_{i=1}^{k} |C_i|}. \tag{3}$$

算法 3 TBCClustering

输入: GML文档集合 $D = \{d_1, d_2, \cdots, d_m\}$, 参数 o ($0 \le o < 1$)、、r.

输出: 聚簇集合 C.

步骤:

```
(1) TDB = Pre-Process(D);          //GML文档预处理, 得到有序标号树数据库 TDB
(2) minsup = GetBesMinsupport(TDB, o);          //获得最小支持度
(3) F = GetBestFeature(TDB, minsup, );          //提取最优特征
(4) D = TDB_to_Vector(TDB, F);          //将每篇文档用特征 F表示
(5) CLOPE(D, r);          //调用 CLOPE算法进行文档聚类
    /* 阶段 1: 初始化 */
    a  classnum = 0;
    b  for(i = 1; i <= |D|; i++)          //对于每个文档 di
    c  {class(di) = -1;          //开始时文档没有类别
    d  计算 di 自成一簇时的全局函数值 Profitnew;
    e  Profitold = 0; oldid = -1;
    f  for(j = 1; j <= classnum; j++)          //对于 C中的每个簇 Cj
    g  {计算文档 di 分到 Cj 簇的全局函数值 Profitj;
    h        if(Profitj > Profitold) {Profitold = Profitj; oldid = j;}}
    i  if(Profitnew > Profitold)
    j        {classnum++; Cclassnum = {di}; C = C  Cclassnum; class(di) = classnum; }
    k  else {Coldid = Coldid  {di}; class(di) = oldid;}
       }
    /* 阶段 2: 迭代 */
    repeat step b ~ k until all class(di) not changed;
```

4　实验结果与分析

为了验证本文提出的 TBCClustering算法的有效性,本文使用 C++ 及 STL实现了 TBCClustering算法,并与 PBClustering算法[7] 以及 TBCClustering-HAC算法进行了比较实验. HAC是用来进行 XML文档聚类最多的凝聚型层次聚类算法,将 TBCClustering算法中的 CLOPE聚类换成 HAC聚类即得到了 TBCClustering-HAC算法,其中相似度比较采用欧氏距离度量,设有两个 n维文本向量 $\boldsymbol{x}$与 $\boldsymbol{y}$,分别用 $(x_1, x_2, \cdots, x_n)$和 $(y_1, y_2, \cdots, y_n)$表示,它们间的距离可以定义为:

$$\mathrm{dis}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} \left(\frac{x_i}{\sqrt{\sum_{i=1}^{n} x_i^2}} - \frac{y_i}{\sqrt{\sum_{i=1}^{n} y_i^2}} \right)^2 . \tag{4}$$

实验环境为 PC/Pentium 4/1 73 GHz/1 25GB,操作系统为 Windows XP. 利用程序自动生成 GML文档作为实验数据源,得到了 7个不同的数据集 Dataset1 ~ Dataset7. 图 2,表 1,表 2与表 3是 3个算法在 Dataset1 ~ Dataset7上的总执行时间比较以及分步执行时间比较. Dataset1 ~ Dataset7上 3个算法的聚类结果完全正确. 其中, PBClustering算法需手动设置支持度与聚簇个数 k,而本文提出的 TBCClustering算法与 TBCClustering-HAC算法则不需要. 参数取值为 $o = 0.02$, $s = 80\%$, $r = 2$. 支持度的选取对于算法的时间性能以及聚类结果都有着不可忽视的影响,支持度取的太小,会导致产生过多的特征,增加聚类时间,导致本来应分到一起的文档集合分为更小的簇. 当支持度增加到一定程度后,会导致本应产生的代表一个簇的特征不再频繁,也就造成聚类结果的不准确. 特别是对于分布不均匀的数据,支持度用户很难确定,但是通过特征优化,可以去除重复的特征,增加聚类的准确性. 可以看出,本文的算法有绝对的优势.

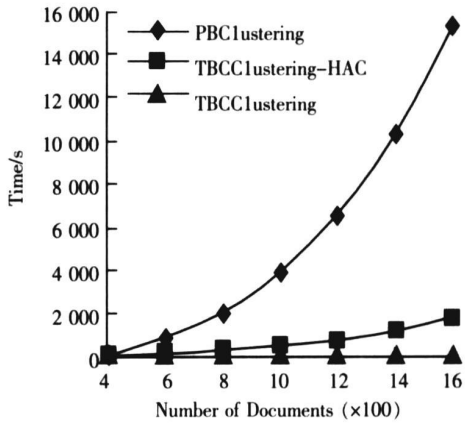

图 2　3个算法在 Dataset1~Dataset7上的执行总时间比较

Fig.2　Peformance comparison based Dataset1~Dataset7

表 1　TBCClustering算法分步执行时间 ($o = 0.02$, $s = 0.8$, $r = 2$) (单位: s)

Table 1　Peformance of algorithm TBCClustering

数据集	Dataset1	Dataset2	Dataset3	Dataset4	Dataset5	Dataset6	Dataset7
自动获得的 minsup	100	150	200	250	300	350	400
特征维数	4	4	4	4	4	4	4
特征提取时间	11.266	15.14	19.437	23.906	30.484	34.812	39.906
聚类时间	0.031	0.031	0.047	0.047	0.062	0.094	0.094

表 2　TBCClustering-HAC算法分步执行时间 ($o = 0.02$, $s = 0.8$) (单位: s)

Table 2　Peformance of algorithm TBCClustering-HAC

数据集	Dataset1	Dataset2	Dataset3	Dataset4	Dataset5	Dataset6	Dataset7
自动获得的 minsup	100	150	200	250	300	350	400
特征维数	4	4	4	4	4	4	4
特征提取时间	11.406	15.531	19.484	23.984	30.578	35.469	39.875
聚类时间	26.657	90.291	212.515	415.031	717.093	1 138.61	1 719.34

表 3　PBClustering算法分步执行时间 (单位: s)

Table 3　Perfomance of algorithm PBClustering

数据集	Dataset1	Dataset2	Dataset3	Dataset4	Dataset5	Dataset6	Dataset7
设定的 minsup	100	150	200	250	300	350	400
特征维数	76	75	75	75	75	75	75
特征提取时间	5.734	7.515	9.844	11.75	14.203	15.5	18.953
聚类时间	247.141	827.328	1 934.78	3 770.36	6 450.34	10 254	15 282.4

5 结语

针对 GML文档结构聚类提出了 TBCClustering算法, 通过挖掘 GML文档集合中的最大频繁 Induced子树构造特征空间, 并对特征空间进行优化, 挖掘过程中自动生成较好的最小支持度, 无需用户设置; 最后采用 CLOPE算法聚类 GML文档, 聚类过程中自动生成簇的个数. 实验表明该算法有很好的性能. 本文工作为 GML文档结构聚类提供了一种新方法.

[参考文献] (References)

[1] Guillaume D, Murtagh F. Clustering of XML documents[J]. Computer Physics Communications, 2000, 127(2/3): 215-227.

[2] Doucet A, Ahonen-Myka H. Naive Clustering of a Large XML Document Collection[C] // Proc 1st Annual Workshop of the Initiative for the Evaluation of XML retrieval(INEX). Germany: ACM Press, 2002: 81-88.

[3] Nierman A, Jagadish H V. Evaluating structural similarity in XML documents[C] // Proceedings of the 5th International Workshop on the Web and Database(WebDB). Madison, 2002: 61-66.

[4] Zhang K, Shasha D. Simple fast algorithms for the editing distance between trees and related problems[J]. SIAM Journal on Computing, 1989, 18(6): 1 245-1 262.

[5] Wang L, Cheung D W, Mamoulis N, et al. An Efficient and Scalable Algorithm for Clustering XML Documents by Structure [J]. IEEE TKDE, 2004, 16(1): 82-96.

[6] Leung H P, Chung F L, Chan S C F. On the use of hierarchical information in sequential mining-based XML document similarity computation[J]. Knowledge and Information Systems, 2005, 7(4): 476-498.

[7] Leung H P, Chung F L, Chan S C F, et al. XML document clustering using common Xpath[C] // 2005 International Workshop on Challenges in Web Information Retrieval and Integration. Tokyo: IEEE Computer Society Press, 2005: 91-96.

[8] Nayak R, Xu S. XCLS: a fast and effective clustering algorithm for heterogenous XML documents[C] // Proceeding of the 10th Pacific-Asia Conference on Knowledge Discovery and Data Mining. Singapore: ACM Press, 2006.

[9] Chehreghani M H, Rahgozar M, Lucas C, et al. Clustering rooted ordered trees[C] // Computational Intelligence and Data Mining. Honolulu, Hawaii: IEEE Press, 2007: 450-455.

[10] Francesca F D, Gordano G, Ortale R, et al. A general framework for XML document clustering[R]. Consiglio Nazionale delle Ricerche Istituto di Calcolo e Reti ad Alte Prestazioni, 2003.

[11] Guha S, Rastogi R, Shim K. ROCK: A robust clustering algorithm for categorical attributes[C] // Proceedings of ICDE99. Sydney: IEEE Computer Society Press, 1999.

[12] Yang Y, Guan X, You J. CLOPE: a fast and effective clustering algorithm for transaction data[C] // Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Edmonton: ACM Press, 2002.

[13] Zaki M J. Efficiently mining frequent trees in a forest: algorithms and application[J]. IEEE Transaction on Knowledge and Data Engineering: Special Issue on Mining Biological Data, 2005, 17(8): 1 021-1 035.

[14] Dalamagas T, Cheng T, Winkel K J, et al. Clustering XML documents using structural summaries[C] // Proceedings of the EDBT Workshop on Clustering Information over the Web. Heidelberg: Springer Berlin, 2004: 547-556.

[15] Chi Y, Xia Y, Yang Y, et al. Mining closed and maximal frequent subtrees from databases of labeled rooted trees[J]. IEEE Transactions on Knowledge and Data Engineering, 2005, 17: 190-202.

[责任编辑: 严海琳]