高技术通讯 2013 年 第23 卷 第4 期:337 ~ 344

doi:10.3772/j.issn.1002-0470.2013.04.002

MRSM:挖掘具有代表性的极大频繁子图①

杨 艳②*** 屈 松* 刘 勇③***

(*黑龙江大学计算机科学技术学院 哈尔滨 150080)

(**黑龙江省数据库与并行计算重点实验室 哈尔滨 150080)

摘 要 基于随机化思想,提出了一种新的挖掘具有代表性的极大频繁子图的算法——MRSM 算法。该算法在第一步挖掘极大频繁子图过程中,采用基于随机化的方法,利用已挖掘到的结果,提高算法的效率;在第二步聚类过程中,综合考虑了频繁模式在支持度和结构上的相似性,使得聚类的质量更好。在真实和模拟数据集上的实验结果证实了MRSM 算法的有效性。

关键词 数据挖掘,极大频繁子图,代表模式,随机算法

0 引 言

频繁模式挖掘是数据挖掘领域中的重要课题。频繁模式挖掘从最早的关联规则挖掘[1]到频繁子树挖掘[2]和频繁子图挖掘[3-6],数据结构越来越复杂,应用也越来越广泛,尤其是现在海量数据大都以复杂的图或网络的形式存在,比如在生物化学、计算机、社会学等领域用图或网络来建模,因此频繁子图挖掘受到越来越多的关注。本文基于随机化思想,提出了一种新的极大频繁模式挖掘算法,该算法利用已挖掘出的部分结果,在减小搜索空间同时去掉重复结果,从而提高了算法的效率。

1 相关工作

目前已有很多有效挖掘频繁子图的算法,但是这些算法大部分主要集中在挖掘频繁子图的全集,在很多实际应用中不能有效处理大规模的数据集,而且挖掘的结果庞大,很大一部分非常相似,用户并不一定对所有结果都感兴趣,而是倾向于更小、更具代表性的集合。

为了解决这样的问题,最近几年,研究者已经提出了一些挖掘具有代表性的频繁子图集合的方法,这些方法一般分为两步,第一步先挖掘出频繁子图集合,第二步运用聚类算法从频繁子图集合中得到有代表性的子图集合。在第一步中,为了处理大规模的数据集,往往采用随机化方法。随机方法在频繁模式挖掘领域的研究越来越深入,从而有效地解决了传统方法在大规模数据集上的效率瓶颈问题。但是因为随机性,势必造成结果的重复发现,比如一个频繁子图,它是有 n 个顶点的完全图,将有 $n!$ 条路径从空模式游走到该图,那么可以通过一条路径多次或通过多条路径游走到该图,这样势必造成很多重复和无用的子图同构计算。但是如果该图通过某条路径第一次被发现,则另 $n!-1$ 条路径也可以确定。那么该图再次通过某条路径被发现时,该路径上的所有中间模式的支持度均不用计算,因为所有的中间模式都是该图的子图,都是频繁的。本文提出的基于随机化的方法根据这样的发现,充分利用已经挖掘到的结果,提高随机算法的效率。算法第二步运用聚类方法,综合考虑频繁子图在支持度上的相似性和结构上的相似性,使得聚类的质量更好,获得的模式更具代表性。所得到的簇在支持度上的误差平方和更小,而结构上的误差平方和则相差无几。

近年来,出现了很多挖掘频繁子图的算法[3-5],也包括挖掘极大频繁子图的算法[6]。这些算法主要用于挖掘频繁子图的全集。从频繁子图集合中发

① 国家自然科学基金(60973081),黑龙江省自然科学基金(F201011),黑龙江省高校科技创新团队建设计划项目(2013TD012),黑龙江省教育厅科学技术研究面上项目(11551352,12531476)和哈尔滨市青年科技创新人才研究(2012RFQXG096,2012RFQXS094)资助项目。

② 女,1973 年生,博士,教授,CCF 会员;研究方向:数据挖掘,数据库;E-mail:yangyan@hlju.edu.cn

③ 通讯作者,E-mail:liuyong001@hlju.edu.cn

(收稿日期:2011-08-22)

现具有代表性的模式集合的算法主要见于文献[7-10]等。其中的 ORIGAMI[7] 是典型的两步算法,它挖掘具有正交代表性的极大频繁子图集合,该算法首先用随机游走的方法从图数据库中挖掘出极大频繁子图集合,然后从集合中聚类得到具有代表性的模式集合,但是算法第一步的效率不高,第二步聚类时没有考虑频繁子图在支持度上的相似性。RING[8] 算法挖掘具有代表性的频繁子图集合,也分为两步,第一步采用和 ORIGAMI 相似的方法得到具有代表性的频繁模式集合,从而获得频繁模式的近似分布,第二步采用深度优先搜索方法得到代表模式集合,该算法也没有考虑频繁模式在支持度上的相似性。还有一些采用蒙特卡罗马尔可夫的方法,比如文献[9,10]等。它们从所有的频繁模式或极大频繁模式集合中均匀的采样。

2 问题定义

现在定义文中所用到的一些概念及问题描述。

标号图:标号图可以用五元组来表示 $G=(V, E, \Sigma_E, \Sigma_V, L)$,其中 V 表示图中顶点集合,$E \subseteq V \times V$ 代表图中边的集合。Σ_V、Σ_E 分别代表节点标号的集合和边标号的集合,L 是标号函数,它是标号与顶点或边之间的映射函数:$V \to \Sigma_V$ 和 $E \to \Sigma_E$。为了便于讨论,在下文中均不考虑边标号。

图 1 所示为四个标号图。

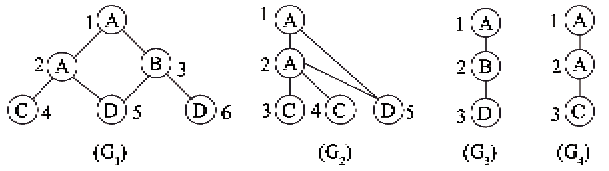

图 1 图数据库

子图同构: 给定两个图 $G=(V, E, \Sigma_E, \Sigma_V, L)$ 和 $G'=(V', E', \Sigma_{E'}, \Sigma_{V'}, L')$,一个从 G 到 G' 的子图同构是一个单射函数 $f: V \to V'$,满足(1) $\forall u \in V$, $L(u)=L'(f(u))$;(2) $\forall (u,v) \in E$, $(f(u), f(v)) \in E'$ 且满足 $L((u,v))=L'((f(u), f(v)))$。

如果存在一个从 G 到 G' 的子图同构,则 G 称为 G' 的子图,G' 称为 G 的超图。记为 $G \subseteq G'$。

频繁子图:给定一个图数据库 $D=\{G_1, G_2, \cdots, G_n\}$ 和一个图模式 p,p 在 D 中的支持集定义为 D 中包含 p 的图的 ID 集合,记为 $D_{supp}(p)=\{i \mid p \subseteq G_i, G_i \in D\}$。$|D_{supp}(p)|$ 称为 p 在 D 中的支持度,记为 $supp(p; D)$。支持度度量具有反单调性质:如果 $p_1 \subseteq p_2$,则 $supp(p_1; D) \geqslant supp(p_2; D)$。对于用户给定的一个最小支持度阈值 $minsup$,如果 $supp(p; D)/|D| \geqslant min\_sup$,称 p 在 D 中是频繁的。

极大频繁子图:若频繁子图 p 的所有超图都不频繁,则称 p 为极大频繁子图。

图在结构上的相似度测量:两个图 G_1、G_2 在结构上的相似度最常用的测量函数是通过计算二者的最大公共子图的规模,可用以下公式计算:

$$sim_{mc}(G_1, G_2)=\frac{|G_{mc}|}{\max(|G_1|, |G_2|)} \quad (1)$$

其中 G_{mc} 为图 G_1、G_2 的最大公共子图,$|G_{mc}|$、$|G_1|$、$|G_2|$ 表示图的规模大小,既图的顶点个数。

图在支持度上的相似度测量:对于两个图 G_1、G_2,其对应的支持集分别为 $D_{supp}(G_1)$ 和 $D_{supp}(G_2)$。这两个图在支持度上的相似度由以下的公式计算:

$$fre\_sim(G_1, G_2)=1-\frac{||D_{supp}(G_1)|-|D_{supp}(G_2)||}{max\_D_{supp}-min\_D_{supp}}$$

公式中的 max_D_{supp}、min_D_{supp} 为频繁模式集合中,支持集规模的最大值和最小值。

图的相似度测量:我们期望挖掘到的代表性模式不仅仅在结构上具有代表性,还要在支持度上具有代表性。因此在计算图之间的相似性时要综合考虑在结构上和支持度上的相似性。为了更好地描述图之间的相似性,我们提出了新的相似度测量函数,其计算公式如下:

$$sim(G_1, G_2)=\frac{str\_sim(G_1, G_2) \times fre\_sim(G_1, G_2)}{str\_sim(G_1, G_2)+fre\_sim(G_1, G_2)} \times 2 \quad (2)$$

问题描述:给定一个图数据库 D,一个最小支持度阈值 $minsup \in [0,1]$ 和图相似度阈值 α,从图数据库 D 中挖掘出具有代表性的极大频繁子图集合问题可形式化定义如下:(1)从数据库挖掘出极大频繁子图的样本集合 M。(2)从集合 M 中挖掘出具有代表性的模式集合 R,使得 M 中的任一模式 p 均能被 R 中某模式 p' 代表,如果 p 和 p' 满足 $sim(p, p') \geqslant \alpha$。

3 极大频繁子图挖掘算法

3.1 算法的搜索空间

频繁子图挖掘算法会在搜索的过程中形成一个

由频繁模式组成的图状的搜索空间,该空间从空模式开始,按边的个数递增的顺序形成层次图,而图的最底层由极大频繁模式组成。以图1所示的图数据库为例,设 *minsup* 为50%,则频繁子图构成的层次图如图2所示,其中加框的为极大频繁子图。图中每个模式代表一个频繁子图。假设两个频繁子图 P_1 和 P_2,如果 P_2 可以通过 P_1 中的某个顶点通过一条边扩展得到,则代表这两个频繁子图的模式之间就有一条边。大部分挖掘算法采用深度优先或广度优先搜索。当数据量很大时,广度或深度优先算法由于需要巨大的搜索空间而变得极其低效。但是采用随机的方法可以避免以上的问题,因为随机的方法只与当前的模式有关,不需要太大的内存空间。

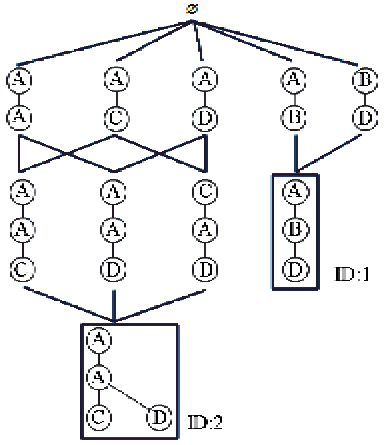

图2　搜索空间图

因为是在图上的随机游走,对于图上的某一个极大频繁子图,可通过多条路径或通过一条路径多次游走达到此图。这样势必造成大量重复和无用的计算,而这些是可以避免或减少的。当发现一个极大频繁子图时,它的所有子图均可以确定,也就是由空模式到该极大频繁子图的所有路径均已确定。如何利用这些路径呢?可以存储这些路径,使下次随机游走时不重复走这些路径,但是存储这些路径要耗费大量的存储空间。也可以为这些极大频繁子图建立索引,而建立索引不需要多少存储空间,因为极大频繁子图的数量相比频繁子图的数量要小得多。在挖掘极大频繁子图过程中,对子图扩展时产生的中间模式,可利用索引技术进行子结构查询,判断是否为已发现极大频繁子图的子图。如果是,则说明该模式是频繁的,就不用再计算支持度。这样可减少重复和无用的计算,提高算法的效率。

3.2　算法使用的数据结构

在算法中用到 *Edge _Map* 和 *Max _Map* 两种数据结构。用 *Edge _Map* 存储顶点标号对应的频繁边的信息,它包括三部分:顶点标号,包含该顶点标号的所有频繁边,频繁边的 *Occ _List*。*Occ _List* 用于记录那些包含该边的图的ID。用 *Max _Map* 为已挖掘到的极大频繁子图的每一条边建立索引,记录所有包含该边的极大频繁子图的ID。表1和表2分别为 *Edge _Map* 和 *Max _Map* 的实例。

表1　Edge _Map

顶点标号	边	*Occ _List*
B	B-D	1,3
	B-A	1,3

表2　Max _Map

边	*Occ _List*
A-A	2
A-C	2
A-D	2

例1:表1为图1所示数据库的 *Edge _Map* 的一部分,以标号为 *B* 的顶点为例,有两条频繁边含有标号 *B*,分别出现在ID为1、3的图中。*Max _Map* 是动态的,随着极大频繁子图的个数变化而变化。假设此时只发现一个极大频繁子图,如图2中所示的ID为2的图,那么对应此时 *Max _Map* 如表2所示。边 *A-A*,*A-C*,*A-D* 分别出现在图2中ID为2的极大频繁子图中。

3.3　算法描述

本节,详细介绍挖掘极大频繁子图的随机化算法。首先介绍最常用的极大频繁子图的随机化产生方法 Basic _Max _Gen,ORIGAMI[7]、RING[8]、RAM[11]等采用与之相似的思想。但是由这种方法挖掘极大频繁子图集合会造成重复和无用的计算,为此我们提出了新的方法 Optimal _Max _Gen,它可以利用已挖掘到的结果提高效率。每运行一次 Basic _Max _Gen 或 Optimal _Max _Gen 即可得到一个极大频繁子图,所以运行若干次后就可以得到极大频繁子图的样本集合。

在运行算法之前,首先扫描图数据库,找出所有频繁边,并构造 *Edge _Map*。Basic _Max _Gen 算法(如算法1所示)从空模式开始,每次从空模式开始随机的增加一条边形成一些中间模式直到极大。假设中间模式为 S_k,它有 k 条边,算法1中第1行,先判断该模式是否为极大频繁模式,即 is _max 函数,

如果 S_k 的所有可能扩展都失败,那么该模式为极大的,则返回该模式,算法结束。如果不是,则扩展 S_k,算法第 3 行的函数 RandomGrow 为随机扩展函数,它扩展 S_k 形成 $k+1$ 条边的频繁模式的候选 S_{k+1}。RandomGrow 首先从 S_k 中随机的选择一个顶点,再由该顶点的标号从 *Edge _ Map* 中随机的选择一条频繁边,将边加入到 S_k 中,如果每次都增加一个顶点,则扩展后的模式为一棵自由树。否则扩展后的模式为带环连通图。如果 S_{k+1} 不频繁则记录扩展失败的顶点和边,以防下次再从此处扩展,并重新扩展 S_k(算法第 5 行)。如果 S_{k+1} 频繁,则递归调用 Basic _ Max _ Gen 从 S_{k+1} 开始扩展(算法第 6 行)。

算法 1 Basic _ Max _ Gen

输入:图数据库 D,模式 S_k,支持度阈值 *minsup*

输出:一个极大频繁子图

功能:由 S_k 扩展形成一个极大频繁子图

Basic _ Max _ Gen (D, S_k, *minsup*)

1. if is _ max (S_k) = *true*
2. return S_k
3. S_{k+1} = RandomGrow (S_k, *Edge _ Map*)
4. if Support (S_{k+1}) < *minsup*
5. goto 3
6. Basic _ Max _ Gen (D, S_{k+1}, *minsup*)

基于 3.1 节的分析,Basic _ Max _ Gen 算法会造成大量重复和无用的子图同构计算,为了减少重复和无用的计算,我们对已挖掘到的极大频繁子图建立索引,对扩展后的模式先在已挖掘到的极大频繁子图集合中查询,如果该模式是某个极大频繁子图的子图,则可以确定该模式频繁,不用需要扫描图数据库确定其支持度。为此,我们给出了另一个更高效的算法 Optimal _ Max _ Gen(如算法 2 所示)。

算法 2 Optimal _ Max _ Gen

输入:图数据库 D,模式 S_k,支持度阈值 *minsup*

输出:一个极大频繁子图

功能:由 S_k 扩展形成一个极大频繁子图

Optimal _ Max _ Gen (D,S_k,*minsup*)

1. if is _ max (S_k) = *true*
2. return S_k
3. S_{k+1} =RandomGrow (S_k, *Edge _ Map*)
4. if Query (S_{k+1}) = *true*
5. Optimal _ Max _ Gen (D, S_{k+1}, *minsup*)
6. if Support (S_{k+1}) < *minsup*
7. goto 3
8. Optimal _ Max _ Gen (D, S_{k+1}, *minsup*)

假设中间模式为 S_k,它有 k 条边,算法 2 中第 1 行,先判断该模式是否为极大频繁模式,如果是则输出 S_k,否则扩展 S_k,对扩展后的模式 S_{k+1} 在已挖掘到的极大频繁子图集合中查询,Query 为查询函数,它判断给定的模式是否为某个极大频繁子图的子图。如果是,则说明模式 S_{k+1} 是频繁的,递归调用 Optimal _ Max _ Gen 从 S_{k+1} 开始扩展(算法 2 第 5 行)。如果不是,则需要计算 S_{k+1} 的支持度。如果 S_{k+1} 不是频繁的,则重新扩展 S_k(第 7 行)。如果频繁则递归调用 Optimal _ Max _ Gen 从 S_{k+1} 开始扩展(算法 2 第 8 行),直到找到一个极大的频繁模式为止。

至此,我们讨论了两种极大频繁子图的随机产生算法,为了产生极大频繁子图的集合,需要运行极大频繁子图产生算法若干次。算法 3 为 Maximal _ Generator 的伪代码,它从图数据库中挖掘极大频繁子图集合。算法首先扫描图数据库,找出所有频繁边,并构造 *Edge _ Map*。初始化 *Max _ Map*,因为初始时并没有挖掘到极大频繁子图,所以 *Max _ Map* 中并无信息。算法 3 第 4 行从频繁边集中随机的选择一条边,然后调用 Optimal _ Max _ Gen 算法,产生一个极大频繁子图。如果产生的极大频繁子图是新的未曾被发现过的,就将该图并入结果集中,然后进行扫描,更新数据结构 *Max _ Map*。*Max _ Map* 有助于提高 Query 函数的查询效率,3.4 节给出了详细的讨论。算法第 8 行判断是否中止程序,由用户根据运行的次数或挖掘到的极大频繁子图的个数来决定。在以下算法伪码中根据运行的次数来控制。第 9 行输出挖掘到的结果。

下面给出算法 Maximal _ Generator 的伪代码:

算法 3 Maximal _ Generator

输入:图数据库 D,支持度阈值 *minsup*,运行次数 *miniter*

输出:极大频繁子图集合 M

功能:从图数据库 D 中挖掘极大频繁子图集合 M

Maximal _ Generator (D,*minsup*,*miniter*)

1. 产生频繁边集 F_K;
2. 构造 *Edge _ Map*,初始化 *Max _ Map*,$M = \Phi$,*iter* = 0
3. while(true)
4. 随机的选择一条边 $e \in F_K$;
5. M' = Optimal _ Max _ Gen (D,e,*minsup*)
6. 如果 M' 为新的极大频繁子图,则 $M = M \cup \{M'\}$,扫描 M',更新 *Max _ Map*
7. *iter* = *iter* + 1
8. If *iter* ≥ *miniter*
9. return M

3.4 子图查询

子图查询函数 Query 在极大频繁子图挖掘过程

中起到很重要的作用,因为在实验中发现,由于随机算法的随机性,同一个频繁子图可能发现很多次,需要进行很多重复和无用的子图同构计算。为了减少这些重复和无用的计算,采用子图查询方法。

子图查询用于判定中间模式是否为已发现极大频繁子图的子图。若是某个极大频繁子图的子图,就不用再扫描图数据库确定其支持度,因为频繁子图的子图一定是频繁的。因此可减少子图同构计算。

在子图查询过程中我们利用由极大频繁子图建立的边索引结构(即 Max_Map)提高查询的效率。在极大频繁子图挖掘过程中每发现一个新的极大频繁子图就扫描各边,维护全局数据结构 Max_Map。算法 4 为 Query 函数的伪代码,它接受中间模式 g,然后判断 g 是否是某个极大频繁子图的子图,如果是,Query 函数返回真,否则返回假。算法 4 第 1 行,计算图 g 的所有边在 Max_Map 中的索引的交集,第 2 行判断 S 是否为空,若为空,则返回假,否则将 g 与 S 中的每一个模式 G 进行子图同构测试,SubIsomorphism 函数为子图同构测试函数。如果有模式包含 g,则返回真,否则返回假。

下面给出算法 Query 的伪代码:

算法 4 Query

输入:图 g,极大频繁模式集合 M,Max_Map

输出:真或假

功能:判断 g 是否为 M 中某个图的子图

Query (g, M,Max_Map)

1. $S = \sum_{e \in g} \cap e.Occ\_List$
2. if ($s = \varnothing$) return *false*
3. for *each* $G \in S$
4. if (SubIsomorphism (g,G) = *true*)
5. return *true*
6. return *false*

3.5 支持度计算

在支持度计算时我们采用 VF2[12] 子图同构算法,该算法比其它子图同构算法有更高的效率。但是子图同构仍然是频繁子图挖掘过程中最耗时的部分,因此要减少子图同构次数。

定理 1:如果图 q 是通过频繁子图 p 插入边 e 得到,且满足 $|D_{supp}(p) \cap Occ\_List(e)| < minsup$,则 q 不频繁。

证明:因为图模式 q 是 p 的超图,所以包含 q 的模式必在 p 的支持集中,可知 $D_{supp}(q) \subseteq D_{supp}(p)$。又因为图 q 是通过频繁子图 p 插入边 e 得到的,所以包含 q 的模式必是 p 的支持集中含有边 e 的模式,即 $D_{supp}(q) \subseteq (D_{supp}(p) \cap Occ\_List(e))$。因为 $|D_{supp}(p) \cap Occ\_List(e)| < minsup$,所以 $|D_{supp}(q)| < minsup$,因此模式 q 不频繁。证毕。

在频繁子图挖掘过程中,可以用 Occ_List 来估算支持度。对一个中间模式 p 的支持集和新插入边 e 的 Occ_List 求交集,既 $D_{supp}(p) \cap Occ\_List(e)$。由定理 1 知,如果集合的大小小于支持度阈值,则该中间模式肯定不是频繁的,就不用再计算其支持度。如果大于支持度阈值,只需对 p 和集合中的每个图 $g \in D_{supp}(p) \cap Occ\_List(e)$ 进行子图同构测试,而不用对图数据库中所有的图进行子图同构测试,这样可以减少子图同构次数。

4 挖掘具有代表性的模式

我们的目的不仅仅是快速的获得极大频繁子图的样本集合,而且要从中获得更小的更有代表性的模式集合。因为在挖掘到的极大频繁子图集合中有很多模式是非常相似的,甚至仅仅有一两条边的差异,再者集合的规模往往随着图数据库的规模增长或支持度阈值的减小而变得异常巨大,对用户而言变得难以分析。

为了从极大频繁子图集合 M 中获得一个更小的更有代表性的集合,我们采用聚类的方法。

4.1 图的相似度测量

对聚类算法返回簇,我们希望簇的内部尽可能地相似,这种相似不仅仅是结构上的相似,也要考虑支持度上的相似。因为我们期望挖掘到的代表性模式不仅仅是在结构上的具有代表性,还要在支持度上具有代表性。

如图 3 所示的三个频繁模式 p_1、p_2、p_3,它们两两在结构上均只有一条边的差异。根据第 2 节给出的公式(1)计算它们在结构上的相似度:

$str\_sim(p_2,p_1) = 0.5$

$str\_sim(p_2,p_3) = 0.5$

可以看出,p_2 与 p_1、p_3 的结构上的相似度是一样的。但是如果考虑模式在支持度上的相似性,那么很明显 p_1 比 p_3 更像 p_2,因为 p_1 的支持度和 p_2 的支持度一样大。因此在计算模式之间的相似度时,综合考虑模式在结构上和支持度上的相似性,所得的相似度才能更好地描述模式之间的相似性。用公式(2)再次计算它们的相似度如下:

$sim(p_2, p_1) = 0.667$

$sim(p_2, p_3) = 0.4$

可以看出 p_1 比 p_3 更像 p_2。

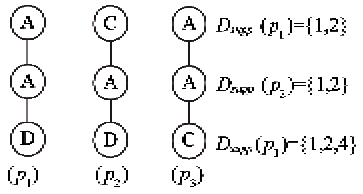

图 3 三个相似的频繁子图

4.2 聚类

对于 M 中的任意两个模式 p 和 p'，计算它们的相似度。相似度测量函数 sim 由公式(2)给出，如果不满足 $sim(p, p') \geqslant \alpha$($\alpha$ 为相似度阈值，由用户给出)，则置其相似度为 0。这样我们得到一个 $|M| \times |M|$ 大小的相似度矩阵，然后采用 k-means 或其它聚类方法从集合 M 中获得代表模式集合。可以用聚类算法返回的簇的质心做为代表模式代表整个簇。其中 k-means 方法需要用户指定 k 的大小。

MRSM(maximal frequent representative subgraphs mining)算法为挖掘代表性极大频繁子图的结构框架，该算法也可以挖掘极大频繁自由树。算法第 1 行挖掘出极大频繁子图集合，然后计算相似度矩阵，第 3 行用聚类方法挖掘出具有代表性的模式集合。算法中采用 k-means 方法。其伪代码如算法 5 所示。

算法 5 MRSM

输入：图数据库 D，支持度阈值 $minsup$，相似度阈值 α，运行次数 $miniter$，代表模式个数 k

输出：代表模式集合 R

功能：从图数据库 D 中挖掘出具有代表性的极大频繁子图集合

MRSM ($D, minsup, \alpha, miniter$)

1. M = Maximal _ Generator ($D, minsup, miniter$)　2. 计算相似度矩阵 $A|_{M \times |M|}$

3. R = k-means (A, k)

4. return R

5 实 验

实验的运行环境为 Intel P4 3.2GHz 处理器，512M 内存，160G 硬盘，操作系统为 Windows XP SP3，所有算法均由 Microsft Visual C + + 设计实现。实验采用的真实的化学化合物数据集来自文献[3]，可以从文献[13]下载，它包括 421 个图，平均每个图有 39 个顶点、42 条边。模拟数据集由数据模拟器进行模拟，该模拟器可从文献[14]下载。

实验从算法的执行效率与挖掘结果集规模两个方面比较算法 Basic _ Max _ Gen 和 Optimal _ Max _ Gen。大部分的随机算法如 ORIGAMI[7]，RING[8]，RAM[13]等采用与 Basic _ Max _ Gen 相似的思想。

5.1 频繁子图挖掘算法实验对比

在真实数据集上，设定支持度阈值为 23.8%，我们分别运行 Basic _ Max _ Gen 和 Optimal _ Max _ Gen 算法 10 ~ 800 次，观察其运行时间随运行次数的变化。

图 4 显示两种算法的运行时间随运行次数的变化情况。从中可以看出 Optimal _ Max _ Gen 算法明显要比 Basic _ Max _ Gen 算法效率要高。随着运行次数的增加，二者渐渐的拉开了距离。这是因为随着运行次数的增加，重复和无用的计算越来越多。

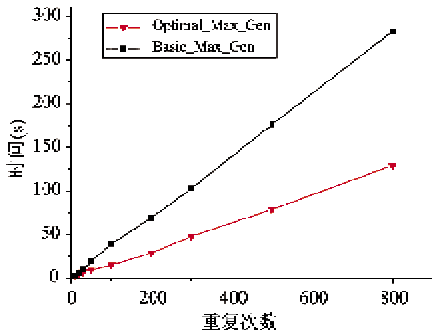

图 4 运行时间随运行次数的变化

为了验证算法的综合效率，我们在不同的支持度阈值上挖掘极大频繁子图，且让每个算法都运行 800 次，然后对比算法的运行时间和挖掘的极大频繁子图个数。其实验结果如图 5 所示。

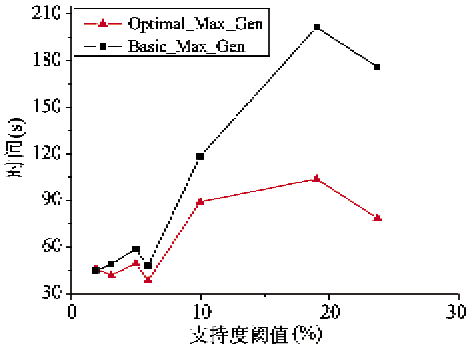

图 5 运行时间随支持度阈值的变化

从上图可以看出在整个支持度变化区间 Optimal _ Max _ Gen 算法要比 Basic _ Max _ Gen 算法高效，随着支持度阈值的增大，表现的更为明显。而二

者挖掘的极大频繁子图个数则几乎一致。

5.2　模拟数据集上的实验结果与分析

模拟数据集由模拟数据产生器 GraphGen[14] 产生。一个典型的模拟数据集通常用类似“D1000T30L200I30V10E10”的名称来表示生成的数据,它表示生成的图总数为1000(D1000),图的平均规模为30条边(T30),频繁子图的种子个数为200(L200),频繁子图的平均规模为30条边(I30),节点的标号个数为10(V10),边的标号个数为(E10)。

在模拟数据集 D1000T30L200I30V10E10 上,设定支持值为5%,分别运行 Basic _ Max _ Gen 和 Optimal _ Max _ Gen 算法1000~10000次,然后对比两个算法运行时间随运行次数的变化情况,实验结果如图6所示。

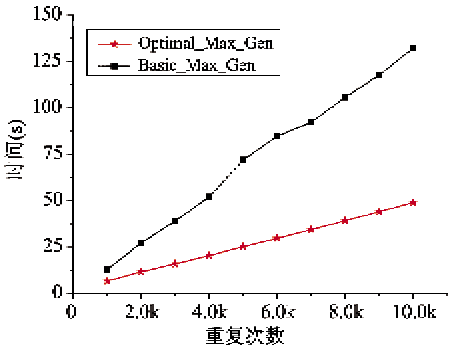

图6　在模拟数据集上的运行结果

由图6可知 Optimal _ Max _ Gen 比 Basic _ Max _ Gen 更加高效。图7显示了在不同的支持度阈值上,两种算法都运行6000次,运行时间变化情况。从图中可以看出,当支持度阈值在2%到5%之间的时候,Optimal _ Max _ Gen 比 Basic _ Max _ Gen 提高了3~5倍,随着支持度阈值的增加,两种算法的运行时间趋于一致。

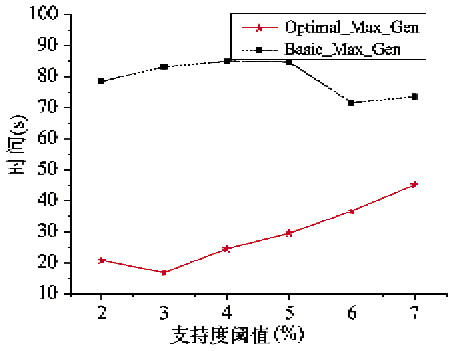

图7　运行时间随支持度阈值的变化

5.3　聚类结果分析

接下来用 k-means 方法对极大频繁子图集合聚类分析,挖掘出具有代表性的模式集合。在真实数据集上,当支持度为2%时,有285个极大频繁子图。我们用误差的平方和(Sum of the Squared Error,SSE)作为度量聚类质量的目标函数。其计算如下:

$$SSE = \sum_{i=1}^{k} \sum_{x \in C_i} dist(c_i, x)^2$$

公式中 $dist$ 用 fre_sim、str_sim 来替代,分别计算在支持度和结构上的误差平方和。误差平方和越小,簇的质量越好。其中的 c_i 表示第 i 个簇的质心。可以用均值来替代。支持度和结构上的误差平方和分别用 SSE_F 和 SSE_S 来表示。实验结果如图8和图9所示。

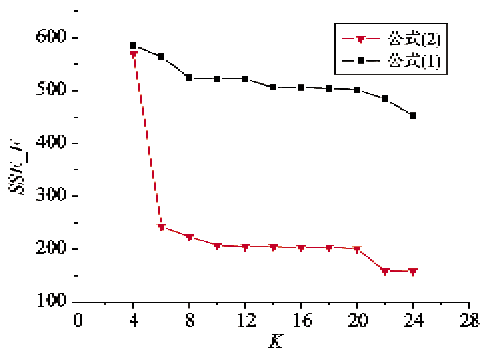

图8　SSE_F 随着 k 的变化

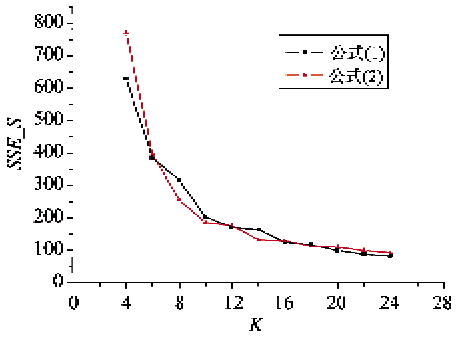

图9　SSE_S 随着 k 的变化

在聚类时,我们分别用公式(1)和公式(2)作为模式之间的相似度测量函数。k 为簇的个数。从图中可以明显的看出随着 k 的增加,SSE_F 和 SSE_S 越来越小,当 $k=12$ 时,SSE_F 和 SSE_S 的曲线均出现拐点。从图中可以得知,聚类时综合考虑结构和支持度上的相似性比只考虑结构上的相似性所获得的簇的质量更好,因为由公式(2)计算的相似度所得到的簇在支持度上的误差平方和比由公式(1)

计算相似度所得到的更小,而结构上的误差平方和则相差无几。

6 结 论

本文提出的新的基于随机化思想的挖掘极大频繁模式的算法利用已挖掘到的部分结果,能够高效地挖掘极大频繁自由树和极大频繁子图。实验结果也证实了这样的结论。并且我们提出了一种新的相似度测量函数,综合考虑模式在结构上和支持度上的相似性,使得聚类的质量更好,所获得的模式更具代表性。

参考文献

[1] Agrawal R, Srikant R. Fast algorithms for mining association rules in large database. In: Proceedings of the 20th International Conference on Very Large Databases, Santiago, Chile, 1994. 487-499

[2] Zaki M J. Efficiently mining frequent trees in a forest. In: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, Canada, 2002. 71-80

[3] Yan X F, Han J W. gSpan: Graph-based substructure Pattern mining. In: Proceedings of the IEEE International Conference on Data Mining, Maebashi City, Japan, 2002. 548-551

[4] Huan J, Wang W, Prins J. Efficient mining of frequent subgraphs in the presence of isomorphism. In: Proceeding of the IEEE International Conference on Data Mining, Melbourne, USA, 2003. 549-552

[5] Nijssen S, KoK J N. A Quickstart in frequent structure mining can make a difference. In: Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, USA, 2004. 549-552

[6] Huan J, Wang W, Prins J, et al. SPIN: mining maximal frequent subgraphs from graph databases. In: Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, USA, 2004. 286-295

[7] Chaoji V, Hasan M A, Salem S, et al. ORIGAMI: a novel and effective approach for mining representative orthogonal graph patterns. *Statistical Analysis and Data Mining*, 2008, 1(2):67-84

[8] Zhang S J, Yang J, Li S R. RING: an integrated method for frequent representative subgraph mining. In: Proceedings of the IEEE International Conference on Data Mining, Miami, USA, 2009. 1082-1087

[9] Hasan M A, Zaki M J. Output space sampling for graph patterns. In: Proceedings of the 35th International Conference on Very Large Databases, Loyn, France, 2009. 730-741

[10] Hasan M A, Zaki M J. Musk: uniform sampling of k maximal patterns. In: Proceedings of the SIAM International Conference on Data Mining, Sparks, USA, 2009. 650-661

[12] Cordella L P, Foggia P, Sansone C, et al. An improved algorithm for matching large graphs. In: Proceedings of the 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, Munster, Germany, 2001. 149-159

[13] gSpan. http://www.cs.ucsb.edu/~xyan/software/gSpan.htm, 2002

[11] Zhang S J, Yang J. RAM: randomized approximate graph mining. In: Proceedings of the 20th International Conference on Scientific and Statistical Database Management, Hong Kong, China, 2008. 187-203

[14] Cheng J, Ke Y P, Ng W. GraphGen: A graph synthetic generator. http://www.cse.ust.hk/graphgen/, 2006

MRSM: a new algorithm for mining maximal frequent representative subgraphs

Yang Yan[*,**], Qu Song[*], Liu Yong[*,**]

([*]School of Computer Science and Technology, Heilongjiang University, Harbin 150080)

([**]Key Laboratory of Database and Parallel Computing Heilongjiang Province, Harbin 150080)

Abstract

A new algorithm for maximal frequent representative subgraph mining (MRSM), called the MRSM algorithm for short, is proposed based on the randomized strategy. The new algorithm uses the mined patterns to improve its efficiency in the stage of mining maximal frequent subgraphs, and in the stage of clustering, it comprehensively considers the similarity in both structure and support of frequent patterns to improve its clustering performance. The extensive experiments on real and synthetic datasets verified the effectiveness and efficiency of the new algorithm, and showed that it can extract high-quality representative patterns.

Key words: Data mining, maximal frequent subgraph, representative pattern, randomized algorithms