ISSN 1000-9825, CODEN RUXUEW
Journal of Software , Vol.18, No.10, October 2007, pp.2469 - 2480 http://www.jos.org.cn
DOI: 10.1360/jos182469
E-mail: jos@iscas.ac.cn
Tel/Fax: +86-10-62562563

一种高效频繁子图挖掘算法 ?

李先通 李建中+, 高 宏

(哈尔滨工业大学 计算机科学与技术学院 ,黑龙江 哈尔滨 150001)

An Efficient Frequent Subgraph Mining Algorithm

LI Xian-Tong, LI Jiang-Zhong +, GAO Hong

(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

+ Corresponding author: Phn: +86-451-86415827, E-mail: lijzh@hit.edu.cn, http://db.hit.edu.cn

Li XT, Li JZ, Gao H. An efficient frequent subgraph mining algorithm. Journal of Software , 2007,18(10): 2469- 2480. http://www.jos.org.cn/1000-9825/18/2469.htm

Abstract : With the successful development of frequent item set and frequent sequence mining, the technology of data mining is natural to extend its way to solve the problem of structural pattern mining — Frequent subgraph mining. Frequent patterns are meaningful in many applications such as chemistry, biology, computer networks, and World-Wide Web. In this paper we propose a new algorithm GraphGen for mining frequent subgraphs. GraphGen reduces the mining complexity through the extension of frequent subtree. For the best algorithm before, the complexity is $O(n^3 \cdot 2^n)$, n is the number of frequent edges in a graph dataset. The complexity of GraphGen is $O\left(2^n \cdot \frac{n^{2.5}}{\log n}\right)$, which is improved $O(\sqrt{n} \cdot \log n)$ times than the best one. Experiment results prove this theoretical analysis.

Key words : frequent pattern mining; subgraph isomorphism; subtree isomorphism; frequent subgraph; spanning tree

摘 要: 由于在频繁项集和频繁序列上取得的成功 ,数据挖掘技术正在着手解决结构化模式挖掘问题——频繁子图挖掘 .诸如化学、生物学、计算机网络和 WWW 等应用技术都需要挖掘此类模式 .提出了一种频繁子图挖掘的新算法 .该算法通过对频繁子树的扩展 ,避免了图挖掘过程中高代价的计算过程 .目前最好的频繁子图挖掘算法的时间复杂性是 $O(n^3 \cdot 2^n)$,其中 ,n是图集中的频繁边数 .提出的算法时间复杂性是 $O\left(2^n \cdot \frac{n^{2.5}}{\log n}\right)$,性能提高了 $O(\sqrt{n} \cdot \log n)$ 倍.实验结果也证实了这个理论结果 .

关键词: 频繁模式挖掘 ;子图同构 ;子树同构 ;频繁子树 ;生成树
中图法分类号 : TP311 文献标识码 : A

? Supported by the National Natural Science Foundation of China under Grant No.60473075 (国家自然科学基金); the Key Program National Natural Science Foundation of China under Grant No.60533110 (国家自然基金重点项目); the National Basic Research Program of China under Grant No.2006CB303000 (国家重点基础研究发展计划 (973)); the Program for New Century Excellent Talents in University (NCET) under Grant No.NCET-05-0333 (国家教育部新世纪创新人才计划)

Received 2006-09-08; Accepted 2006-11-14

由于图具有广泛的应用性,所以针对图挖掘的研究也越来越引起人们的重视.很多领域内的数据都可抽象为图,如给定物质的化学结构能被抽象为无向标号图(其中的节点对应着原子,而边则对应着原子间的化合键),生物学、XML、计算机网络和WWW中的数据也可抽象为图,用于进一步的研究.

图挖掘可以粗略地分为频繁子图挖掘、图聚类与图分类等等.频繁子图挖掘的目的是找到在图集中频繁出现的子图.由于频繁子图挖掘结果可作为图聚类、图分类等其他图挖掘研究的基础,也使得频繁子图的挖掘工作具有更加深远的影响.目前,频繁子图的挖掘结果已经应用于一些应用领域内,如在PTE(predictive toxicology evaluation challenge)项目中的应用,其目的是找到频繁出现的与有毒物质具有相同子结构的物质.Borgelt和Berthold在文献[1]中证明了使用图描述和采用频繁子图挖掘技术对这个问题研究的有效性.其他的基于频繁子图挖掘的应用包括录像索引、提高结构化数据库的存储效率、高效索引技术和WEB信息管理等.

但是随着图和图集规模的扩大,现有频繁子图挖掘算法在效率上已不能满足需求.在PTE项目中,图的规模还比较小(20个节点与边),节点度的平均值也较小(≤4),节点标号集却相对较大(>60).但在蛋白质研究领域内抽象出的图则大得多,虽然节点标号集相对较小(20个标号左右),但图的尺寸更大(100~1 000个节点或边),每个节点的度也更大(在某些例子中达到6~20).因此,要设计适合更大、更复杂图集的挖掘算法,提高频繁子图的挖掘效率势在必行.

初期的频繁子图挖掘算法,将工作重点集中在近似技术(SUBDUE[2])和小规模数据库上(inductive logic programming).

近期的频繁子图挖掘算法可以分为两类:一类是广度优先算法(broad first search,简称BFS).这类算法采用了类Apriori性质去枚举重复出现的子图.这类算法包括AGM[3,4]和FSG[5].AGM在图集中搜索所有"诱导"子图.图G的诱导子图G′的节点为V(G′)?V(G),G′的边为V(G′)中节点在图G中的所有边.FSG则利用边增长的方式查找所有图集中的频繁连通子图.因为采用了Apriori性质,即在扩展边的过程中,通过两个n条边候选图进行乘积以得到(n+1)条边的候选图,这样做会生成大量重复候选子图,降低了算法的整体效率.

频繁子图挖掘算法的另一类是具有更好执行效率的深度优先算法(depth first search,简称DFS).这类算法包括gSpan,CloseSpan和FFSM等.所有算法均通过逐步扩展频繁边得到频繁子图,但每个算法对图的扩展过程略有不同.Yan和Han在文献[6]中提出的算法gSpan是典型的深度优先算法[7],频繁子图G是通过对其父亲子图G′增加一条新的频繁边而得到的.与枚举所有潜在子图不同,gSpan方法保存一个频繁子图G的子图同构列表,这样做的好处是能减少制约效率的同构测试次数.不过,gSpan并不能避免子图同构测试,边的扩展算法复杂性也很高,达到了$O(2^n)$,使总的时间复杂性高达$O(2^n\cdot n^2)$.在文献[8]中,Yan等人又提出了挖掘频繁闭图集的CloseSpan算法,虽然它在边的扩展与结果集的剪枝中采用了一些优化方法以减少计算时间,但总体的时间复杂性依然为$O(2^n\cdot n^2)$.Huan和Wang在文献[9]中提出了另一种频繁模式挖掘算法FFSM,将图的描述巧妙的转化为标准邻接矩阵(canonical adjacent matrix,简称CAM).这样,FFSM不但将子图同构问题转化成了对矩阵的操作,也将图扩展过程巧妙的转化为矩阵的联接操作与矩阵的扩展操作.但是,由于其图扩展的过程通过矩阵的联接与扩展完成,所以扩展边的时间复杂性为$O(m^2n\cdot n^2)$?,图同构测试的时间复杂性为$O(m^2)$,总的时间复杂性降低至$O(m^4n\cdot n^2)$.具有m个节点的完全图包含$m(m-1)/2$条边,因此,其时间复杂性转化为用频繁边数表示的情况为$O(n^3\cdot n^2)$.

在表1中,我们给出了上述3种算法时间复杂性的比较,其中,n表示边数,m表示节点数,第2列给出了这些挖掘算法对频繁图中边进行扩展时的时间复杂性,第3列给出了这些算法进行子图同构测试的时间复杂性.由于在频繁子图的挖掘过程中,每次对边进行扩展都需要进行同构测试,因此,总的时间复杂性是边扩展的复杂性与同构测试复杂性的乘积,在第4列中给出.

从表1中可以看出,在频繁子图挖掘算法中,两个对算法时间复杂性影响最大的问题是子图同构问题(子图同构问题已被证明是NP完全问题)和有效的图扩展方法.表1中的数据显示,以上3种算法对边的扩展过程均采

?算法本身是按先深搜索的方式对边进行扩展,其复杂性为$O(2^n)$.但由于在扩展过程中,使用了矩阵的联接操作与扩展操作形成新的CAM,使复杂性达到$O(m^2n\cdot n^2)$.

用先深搜索方式 ,但对子图同构测试的方式却不尽相同 ,FFSM 效率更高 ,它能回避图与图之间直接的同构测试 , 通过标准邻接矩阵的比较确定子图同构问题 ,达到相对较好的时间复杂性 $O(m^2)$.因此 ,无论提高扩展边的效率还是子图同构测试的效率 ,都会对降低整体算法的时间复杂性起到积极的作用 .虽然近期也有部分挖掘技术研究成果 [10-12],但这个问题仍然没有很好的解决 .

Table 1 Complexity of graph mining algorithms

表 1 图挖掘算法的时间复杂性分析

	扩展边的复杂性	同构测试复杂性	总复杂性
gSpan	$O(2^n)$	$O(2^n)$	$O(2^n \cdot n!)$
CloseSpan	$O(2^n)$	$O(2^n)$	$O(2^n \cdot n!)$
FFSM	$O(m^2 n \cdot n!)$	$O(m^2)$	$O(n^3 \cdot n!)$

本文提出了一种新的频繁子图挖掘算法 GraphGen, 基本思想是将图挖掘过程分为两步 :第 1 步是生成频繁子树 ,对边的扩展采用深度优先的方法 , 复杂性为 $O(2^n)$, 但由于挖掘结果是树 , 其同构测试时间复杂性为 $O\left(\frac{n^{2.5}}{\log n}\right)$[10].第 2 步是对第一步中生成的频繁子树进一步扩展以形成频繁图 ,边的扩展方式为广度优先 , 复杂性为 $O(n^2)$, 进行子图同构测试的时间复杂性为 $O(2^n)$, 这部分总的时间复杂性为 $O(n^2 \cdot n!)$.由于这两部分是顺序完成的 ,因此 , 算法总体的时间复杂性为 $O\left(2^n \cdot \frac{n^{2.5}}{\log n}\right)$.

本文第 1 节给出与频繁子图挖掘的相关术语及定义 .第 2 节给出频繁子图挖掘算法 (先给出图描述的标准化编码 ,然后给出图挖掘算法).第 3 节给出实验结果 .第 4 节对全篇进行总结 .

1 基本概念

在本文中 ,我们将问题集中于简单无向连通标号图 .通过修改 ,本文提出的算法也适用于复杂无向连通标号图 ,即含有环或两节点间有多条边的情况 .

标号图作为一种通用的数据结构 ,能用于复杂模式的模型化工作 .我们在这部分给出标号图的定义 , 同时给出的还有子图同构与频繁子图支持度的定义 .基于这些定义 , 我们能对图进行准确的数学描述 .

定义 1.1(标号图). 一个标号图是一个五元组 , $G=\{V, E, \Sigma E, \Sigma V, L\}$. 其中 , V 代表图中节点的集合 , $E \subseteq V \times V$ 代表图中边的集合 . ΣV, ΣE 分别代表节点标号的集合与边标号的集合 .L 是标号函数 , 用于完成标号向节点和边的映射 :$V \to \Sigma V$ 与 $E \to \Sigma E$.

定义 1.2(图的同构). 图的同构是一个双射 $f: V(G) \to V(G')$. 对于图 $G=\{V, E, \Sigma V, \Sigma E, L\}$ 与图 $G'=\{V', E', \Sigma V', \Sigma E', L'\}$, 若它们是同构的 ,则满足如下条件 :

$\forall u \in V, L(u)=L'(f(u))$

$\forall u,v \in V, ((u,v) \in E) \Leftrightarrow ((f(u), f(v)) \in E')$, 且

$\forall (u,v) \in E, L(u,v)=L'(f(u),f(v))$.

定义 1.3(子图同构). 给定标号图 G 与 G',若 G 中存在子图 G''与图 G 同构 ,则称 G 与 G' 是子图同构的 , 记为 $G \subseteq G'$.

子图同构问题是 NP 完全问题 .如何减少或降低子图同构的计算开销 ,是图挖掘工作研究人员的主要研究内容之一 .

定义 1.4(支持度). 给定一个图的集合 GD, 图 G 的支持度记为 SUP_G, 计算方法为 GD 中与 G 存在子图同构的图 G' 的个数与整个图集中图的个数的比值 , 表示如下 :

$$SUP_G = \frac{|\{G' \in GD \mid G \subseteq G'\}|}{|GD|}.$$

一个图是否频繁 , 与该图在图集中的出现次数相关 , 同时也与预先定义的最小支持度阈值有关 .设预先给定

的最小支持度阈值为 min_sup,则频繁图的定义如下 :

定义 1.5(频繁图与频繁树). 给定一个图集 GD ,GD ={ G_i |i=0,1, …n}, 且给定最小支持度阈值为 min_sup, 我们称图 G 是频繁的 ,当且仅当 G 的支持度不小于最小支持度阈值 ,即 $SUP_G \geq min\_sup$.相应地 ,当图 G是频繁的且其中无回路时 ,我们称 G为频繁树 .

频繁子图挖掘问题是给定一个最小支持度阈值 min_sup,在图集 GD 中找到所有支持度不小于 min_sup 的子图 .在以下章节中 ,使用大写字符表示集合 ,如 “G ” ,E“ ” ,V“ ”等 ,而用小写字符表示元素 ,如 “g ” ,e“ ” ,v“ ”等 .

2 挖掘频繁子图

采用先深搜索的办法挖掘频繁子图 ,会在搜索的过程中形成一个树状的搜索空间 (这个特点可以参照频繁项集中 FP 树的定义),这棵树的根节点为空 ,由根至叶按边个数递增的顺序形成树 ,非根节点分别代表不同的频繁子图 .整个挖掘过程是由根至叶的遍历过程 .图 1 显示了这个树状的搜索空间 .

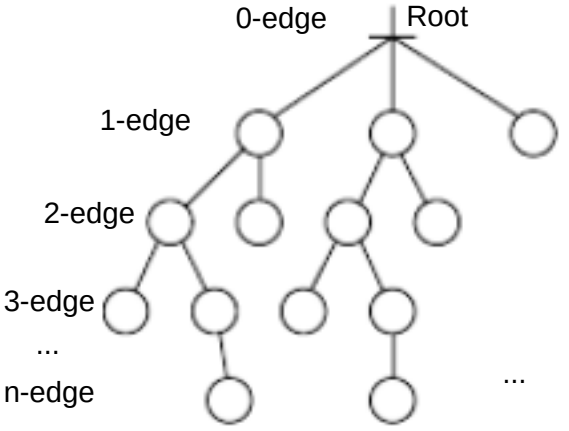


Fig.1 Search space

图 1 搜索空间

图的标准化编码是对图的一种形式化描述 .将图与标准化编码进行一一对应 ,可有效的避免同构图的不同遍历顺序引起的重复候选集的计算开销 .确定了图的标准化编码 ,我们就可以只计算具有标准化编码的遍历 ,降低算法的复杂性 .

2.1 图的标准化编码

标号图的每一条边 ,都存在 5 个基本元素 ,即两个节点标识、两个节点标号和边标号 ,我们可直接使用该五元组对边进行描述 ,将边 e=(v_i, v_j) 表示为 (v_i,l_i,v_j,l_j,l_e), 其中 ,v_i,v_j 为节点标识 ,l_i,l_j 为节点标号 ,l_e 为边标号 .例如图 2(a) 所示标号图 ,边 (v_1,v_2) 可表示为 (v_1,B,v_2,A,b), 边 (v_0, v_3) 可表示为 (v_0,A,v_3,B,c).

节点标识之间存在一种线性顺序 :若在图的遍历过程中先遍历 v_i, 后遍历 v_j.则 v_i 和 v_j 之间的顺序关系为 v_i ? v_j.同样地 ,我们规定节点标号与边标号之间也存在一种序关系 ,即线性的字母序 .由此 ,我们可定义边之间的线性关系 .例如图 2(a) 中 , 我们可通过 v_0 ? v_1 来确定 (v_0, v_1) ? (v_1,v_2), 但图 2(a) 与图 2(b) 中 , 边 (v_0,v_1) 中各项编码均相同 , 仅边的标号不同 , 我们可通过边的标号来确定其大小 , 由于 a<b, 所以 $(v_0,v_1)_a$? $(v_0,v_1)_b$, 其中 , 括号的下标表示图 (a) 或图 (b).

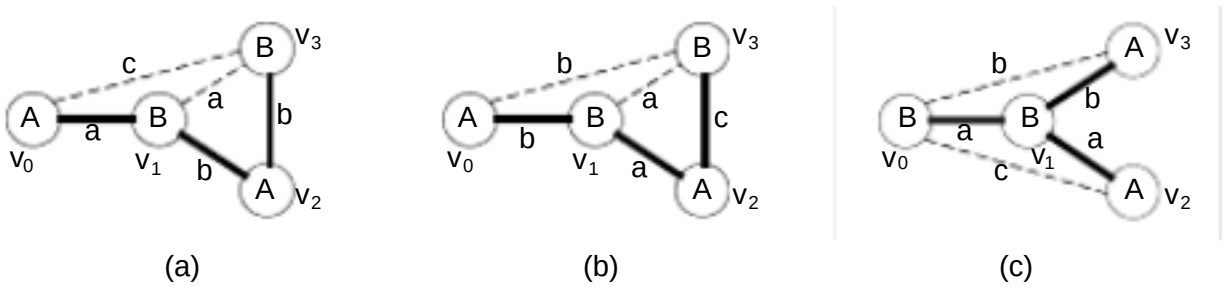


Fig.2 Representation of labeled graph

图 2 标号图的表示方法

定义 2.1(边的线性顺序). 给定标号图的任意两条边 $e_a=(a_1,a_2,a_3,a_4,a_5)$, $e_b=(b_1,b_2,b_3,b_4,b_5)$, 其线性顺序由下

列条件决定 :

1) $e_a=e_b$,当且仅当 $a_i=b_i$,i=1,2, …,5;

2) e_a? e_b,当且仅当 ?k,1≤k≤5,使 $a_j=b_j$ (1≤j<k),且 a_k? b_k;

3) e_b? e_a,其他情况 .

例如 ,图 2 所示为同一个图的 3 种不同遍历结果 .图 2(a)中的边 $(v_1,v_3)_a$可描述为 (v_1,B,v_3,B,a),图 2(b)中的边 $(v_1,v_3)_b$可描述为 (v_1,B,v_3,B,a). 由于这两个五元组完全相等 ,因此 ,这两条边的顺序关系是相同的 .然而 ,虽然图 2(a)中的边 $(v_0,v_1)_a=(v_0,A,v_1,B,a)$与图 2(b)中的边 $(v_0,v_1)_b=(v_0,A,v_1,B,b)$前 4 个元素具有相同的顺序关系 ,但第 5 个元素的顺序关系为 a? b,依照定义 2.1,边的顺序关系为 $(v_0,v_1)_a$? $(v_0,v_1)_b$. 同理 ,图 2(a)中的边 $(v_0,v_3)a=(v_0,A,v_3,B,c)$与图 2(b)中的边 $(v_0,v_3)_b=(v_0,A,v_3,B,b)$之间的大小关系为 $(v_0,v_3)_b$? $(v_0,v_3)_a$,而图 2(a)中的边 $(v_2,v_3)_a=(v_2,A,v_3,B,b)$与图 2(b)中的边 $(v_2,v_3)_b=(v_2,A,v_3,B,c)$之间的大小关系为 $(v_2,v_3)_a$? $(v_2,v_3)_b$.

同样地 ,我们可给出图的线性顺序定义

定义 2.2(图的线性顺序). 给定两个标号图的 DFS遍历编码 $g_1=\{e_{11},e_{12},\cdots e_{1n}\}$, $g_2=\{e_{21},e_{22},\cdots e_{2m}\}$, 其中 ,$e_{ij}$ 表示图 g_i 中 DFS遍历第 j条边的编码 .图 g_1、图 g_2的线性顺序由下列条件决定 :

1) $g_1=g_2$, 当且仅当 m=n,且 $e_{1i}=e_{2i}$, 其中 ,1≤i≤m;

2) g_1? g_2,当且仅当下列条件之一成立 :

a) ?k,1≤k≤min(n,m),使 $e_{1j}=e_{2j}$ (1≤j<k), 且 e_{1k}? e_{2k};

b) n<m,且 $e_{1i}=e_{2i}$ (1≤i≤n)

3) g_2? g_1,其他情况 .

例如 ,表 2 的第 2 列~第 4 列分别对应图 2 中的 3 种遍历结果图 2(a)~图 2(c). 通过定义 2.1 与定义 2.2 给出的线性顺序 ,我们得到 g_a? g_b? g_c.下面我们证明对任意图 ,其DFS编码值都存在最小值 ,且这个最小值是唯一的 .

Table 2 DFS coding of graphs

表 2 图的 DFS 编码

Edge	图 2(a)	图 2(b)	图 2(c)
0 (	v_0,A,v_1,B,a) (	v_0,A,v_1,B,b)	(v_0,B,v_1,B,a)
1 (	v_1,B,v_2,A,b) (	v_1,B,v_2,A,a)	(v_1,B,v_2,A,a)
2 (	v_2,A,v_3,B,b) (	v_2,A,v_3,B,c)	(v_1,B,v_3,A,b)
3 (	v_3,B,v_1,B,a) (	v_3,B,v_1,B,a)	(v_2,A,v_0,B,c)
4 (	v_3,B,v_0,A,c) (	v_3,B,v_0,A,b)	(v_3,A,v_0,B,b)

定理 2.1. 图的最小 DFS 编码是唯一的 .

证明 :先证明其存在性 .设图 g为具有 n条边的标号图 ,共有 k种遍历方法 ,其编码的集合为 $G_{DFS}=\{g_1,g_2,\cdots g_k|g_i=\{e_{i1},e_{i2},\cdots e_{in}\}(1\le i\le k)\}$. 由定义 2.2 知,? g_i,g_j∈G,g_i和 g_j之间均存在线性顺序关系 .因此 ,将G中的编码逐个比较 ,必然能找到 g′∈G,且其线性顺序具有最小值 ,所以图的最小 DFS编码存在 .

再证明其唯一性.设最小DFS编码不唯一.图g的不同遍历产生的DFS编码为$G_{DFS}=\{g_1,g_2,\cdots g_k|g_i=\{e_{i1},e_{i2},\cdots e_{in}\}(1\le i\le k)\}$. 设 g_i 与 g_j均为最小 DFS编码 ,且 $g_i\neq g_j$.由定义 2.2 知,两个不同的 DFS 编码值 ,其相互关系仅有 3 种情况 ,若 $g_i\neq g_j$,则必有 g_i ? g_j或 g_j? g_i之一成立 .在不影响结果的前提下 ,不妨设 g_i ? g_j成立 ,这与题设 g_i与 g_j均为最小 DFS编码矛盾 ,而导致这种矛盾的原因是假定最小 DFS编码不唯一 .所以 ,最小 DFS编码是唯一的 .

综上所述 ,最小 DFS 编码存在 ,并且唯一 . □

定理 2.1 确定了图与最小 DFS 编码的一一对应关系 .在进行图挖掘的过程中 ,由于同构图的遍历顺序不同 ,会形成不同的 DFS 编码 .通过最小 DFS 编码与图的一一对应 ,我们在挖掘过程中可以放弃对其他 DFS 编码的继续操作而只考虑最小 DFS 编码 ,这样做既会保证得到完整的结果集 ,又可以避免扩展重复候选集所带来的额外计算开销 .

接下来 ,我们从频繁子图的生成过程着手 ,进一步分析挖掘过程中树与图之间的关系 .

定义 2.3. 在图的扩展过程中 ,如果一条扩展边引入了一个新的节点 ,则称该扩展边为外边 ,由 e_o表示 .若该扩展边未引入新的节点 (扩展边的两个节点已存在于该图中), 则称其为内边 ,用 e_w表示 .

图集中 ,节点与边的频率超过最小支持度阈值时 ,我们称其为频繁节点与频繁边 .由此设定 ,当外边 (内边)的频率不低于最小支持度阈值时 ,称其为频繁外边 (频繁内边).

例如 ,图 2(a)的遍历过程如表 2 第 2 列所示 ,其中 ,用粗实线描述的边表示扩展过程中的外边 ,而用虚线表示的边则为内边 .因此 ,我们可以将这些边分别加入两个集合 :一个集合是外边集 ,为 $E_o=\{(v_0,v_1),(v_1,v_2),(v_2,v_3)\}$, 另一个集合为内边集 $E_w=\{(v_3,v_1),(v_3,v_0)\}$.

定理 2.2. 如果一个图仅由外边组成 ,那么它是一棵树 (无回路). 反之 ,如果一个图中至少含有一条内边 ,则它必含有回路 .

证明 :我们通过对边进行归纳证明外边形成树 .

如果一个图 g_0仅含有一条边 ,显然 ,g_0无回路 ,g_0是一棵树 .

设图 g_n由 n条外边组成且无回路 ,我们考虑加入一条新的外边的情况 .假设被加入的外边为 $e=(v_i,l_i,v_j,l_j,l_e)$, 并假定节点 v_j为新引入的节点 ,即 $v_j\notin V(g_n), e\notin E(g_n)$, 因此 ,节点 v_j与图 g_n之间仅有一条路 e与 v_i连通 .通过题设我们知道 ,在引入节点 v_j 之前 ,图 g_n中不存在回路 ,即由节点 v_i起始的路径集合 $P_k=\{p_k|(p_k$是 g_n中的路径 $)\wedge(p_k$起始于 $v_i)\}$(其中 ,k=1, …,(i- 1),(i+1), …,(n+1)) 中无回路 .到达 v_j的路径必经过 v_i,即 $P_j=\{p_j|p_j=p_k+e\}$, 由于 v_j与 g_n之间仅有一条路 e, 因此得出 $\forall p\in P_j$不是回路 .所以 ,g_{n+1}中无回路 .

现在我们考虑图 g 中存在至少一条内边的情况 .

设 $e=(v_i,l_i,v_j,l_j,l_e)$是内边 ,且图 g 在 e加入之前由外边组成 .由前面的证明知道 ,此时 g是树 ,所以 ,它的所有节点都是连通的 ,因此 ,必然存在一条路径 p起始于 v_i而终止于 v_j, 记为 $p_{(i,j)}$.$e\notin E(g_n)$且 e连通节点 v_i与 v_j. 因此 ,图 $g\cup e$中有回路 ,且该回路为 $(e+p_{(i,j)})$. □

在图挖掘过程中 ,影响计算效率的原因来自两个方面 :一方面是子图同构测试 ;另一方面是边的扩展 .通过定理 2.2, 我们可将图的挖掘过程分为两个步骤 :第 1 步生成频繁子树 .与图的同构测试相比 ,子树同构测试的时间复杂性更低 ;第 2 步由频繁子树进一步扩展为频繁子图 ,这时 ,频繁子树在扩展过程中不引入新的节点 ,边扩展的时间复杂性降低至 $O(n^2)$. 但是 ,将挖掘过程分为两步 ,我们需要面对的另一个问题是 ,在频繁子树向频繁子图扩展的过程中 ,能否生成完整的结果集 .

定理 2.3. 1) 一个图的先深搜索生成树的最小 DFS 编码是唯一的 .

2) 生成树的最小 DFS 编码确定 ,并给定频繁内边集 ,扩展之后的图也是唯一的 .

证明 :1) 树是无回路的图 .定理 2.1 保证了生成树的最小 DFS 编码不但存在 ,而且唯一 .

2) 假设图 g为最小 DFS 树 t所扩展 .令 $E\doteq E(g)-E(t)$ 为给定的频繁内边集 .同时 ,根据图中生成树的定义 ,$\forall v\in V(g)\Rightarrow v\in V(t)$, 即生成树 t包含图 g的所有节点 .因此 ,$\forall e\in E'$,$e=(v_i,l_i,v_j,l_j,l_e)$, 都能在树 t中找到对应的 $v_i\in V(t)$, $v_j\in V(t)$,即边 e的两个节点均在 t中出现 ,并且 ,由于我们考虑的是简单图 ,任意两节点间存在的边数不会大于 1, 所以 ,此对应关系为一一对应 .综上所述 ,DFS 树确定 ,且后向边集确定 ,得到的图也是唯一的 . □

定理 2.3 说明 ,最终结果集中的任一频繁子图都存在一棵频繁子树与之对应 ,在由树向图的扩展过程中 ,由于内边的逐条加入 ,会形成一个完整的搜索空间 .

2.2 FTGen

作为频繁子树挖掘算法 ,FTGen(frequent subtree generation) 的输入条件为图集 GD, 频繁子树 t(程序开始运行的时候是频率最高且 DFS 编码最小的边), 频繁边集 E.频繁边集 E 可通过对图集 GD 的扫描与边的统计得到 ,同时 ,E 中的边按频率的降序排列 .算法的详细定义如下 :

算法 1. FTGen.

输入 :图集 GD,频繁子树 t,频繁边集 E;

输出 :频繁子树集 T.

(1) if $t\neq\min(t)$ then return; /* 检查 t 是否具有最小 DFS 编码 */

(2) $E'\leftarrow\{e$ 是频繁外边 $\}$;

(3) for E 中的每条边

(4)　　E′←E′e;

(5)　　if (t?e)存在于图集中　;

(6)　　then t←t?e;

(7)　　if (T 中无　t 的同构子树　) then　T←T∪t;

(8)　　FTGen(GD,t,E,T);

(9) endfor;

算法在第　1 行判断频繁子树是否为最小　　DFS 编码 .若当前子树不是最小　　DFS 编码 ,则放弃操作　,以避免对重复生成的子树做进一步扩展　　.在生成频繁子树之前　　,算法在第　2 行找到子树　t 的所有频繁外边　,这样 ,可以保证新边加入后　,t 中依然无回路　.算法的第　4 行到第　8 行 ,循环处理子树的每一条外边　　,第　5 行检验当前外边　　e 能否加入　t 中 (扩展后的子树出现在图集　　GD 中 ,符号　“? ”的意义为将外边　　e 加入子树　t),并将这样的边与子树联接生成新的子树　.若无重复的子树生成　　,继续对　t 扩展 ,直至频繁外边集为空或　　T 中存在　t 的同构子树　.同时 ,将生成的频繁子树在第　7 行中加入到频繁子树的集合中　　.

FTGen　的挖掘结果是图集　　GD　中出现的频繁子树　　,储存在集合　T 中 .频繁子树将被用于进一步对边扩展以形成图　,同时 ,频繁子树作为图的一种　　,是最终挖掘结果的子集　　.

FTGen　主要的计算开销由两部分组成　　:一是子树同构的测试　　.虽然目前有很多关于子树同构测试的研究工作 [13-17], 但我们采用文献　[13] 中提供的方法　,将时间复杂性控制在　$O\left(\frac{n^{2.5}}{\log n}\right)$; 二是频繁外边的扩展　　,由于采用了先深搜索　,这部分的时间复杂性为　$O(2^n)$, 即算法总的时间复杂性为　$O\left(2^n \cdot \frac{n^{2.5}}{\log n}\right)$.

同时 ,定理　2.1 与定理　2.2 保证了算法的正确性　　.定理　2.1 确定了剪枝的基础　　,即通过　DFS 最小编码与图之间的一一对应　,对不符合最小　　DFS　编码的中间结果直接删除而不影响最终结果　　.定理　2.2　确定了外边、内边的扩展顺序与树和图之间的关系　　.我们在算法的挖掘过程中只增加外边　　,从而确保生成结果是树　　.

2.3　频繁子图挖掘算法　(GraphGen　)

算法 GraphGen用于挖掘图集中的频繁子图　　,其挖掘过程是先生成频繁子树　　,再由频繁子树扩展为频繁子图　　.其中 ,频繁子树挖掘算法采用本文算法　　1　描述的　FTGen. 由树向图扩展内边的特点是　　,随着边数的增加　,节点数保持不变　.根据这个特点　,我们在　GraphGen中将边扩展的时间复杂性降低至　　$O(n^2)$. 下面给出算法　GraphGen 的详细定义 :

算法　2. GraphGen.

输入 :图集　GD, 最小支持度阈值　　min_sup;

输出 :频繁子图集合　　FG.

(1)　　扫描图集并找到图集　　GD　中所有频繁边　;

(2)　　删除所有非频繁边　　;

(3)　　E←{ GD 中所有频繁边　};

(4)　　将　E 中的边按　DFS 编码顺序和频率的降序进行排列　　;

(5)　　T←NULL ; /* T 为频繁子树集合　*/

(6)　　t←e_1; /* E中的第一条边作　FTGen的初始值　*/

(7)　　FTGen(D,t,E,T);

(8)　　将集合　T 中的元素按节点数与　　DFS 编码顺序进行排序　　;

(9)　　FG ←T;

(10) for　T 中的每棵树

(11)　　g←t;

(12)　　E′←{e 是频繁边　,且　e 是内边　,并能在图集中找到　(g?e)};

(13) for E 中的每条边

(14) E′←E′∪e;

(15) g←g?e;

(16) if g≠min(g) then break;

(17) if FG 中无 g 的同构子图 then FG←FG∪g;

(18) endfor;

(19) endfor;

(20) return FG;

GraphGen 分为 3 个部分 ,算法 2 给出了这个算法的细节 .

算法在第 1 部分 (第 1 行~第 6 行)对图集 GD 进行预处理 .作为图挖掘的基础 ,必须从图集中提取出必要的信息 ,如频繁边集、 频繁节点集等 .在这个部分中 ,GraphGen 扫描图集 GD 并得到频繁边集 ,将频繁边集按频率递减与 DFS 编码值递增的顺序进行排列 ,供算法进一步计算 .

算法在第 2 部分 (第 7 行)进行频繁子树的挖掘工作 ,具体的挖掘过程见 2.2 节.

算法的第 3 部分 (第 8 行~第 19 行)是由树向图的扩展过程 .针对每一棵频繁子树 ,从频繁边集中找出能与之联接的内边 ,逐一加入到该树 ,从而形成频繁子图 .算法在第 12 行找到所有能与图 g 联接的频繁内边 ,并将这些边在第 13 行~第 17 行的子循环中加入到图 g 中.算法的执行结果为频繁子图集合 FG.

与 FTGen 相似 ,算法 GraphGen 主体的时间复杂性也分为两部分 :一部分是子图同构的时间复杂性 .由于子图同构的测试是 NP完全问题 ,算法在第 17 行进行的图同构测试 ,其时间复杂性为 $O(2^n)$;另一部分是扩展边的时间复杂性 .由于我们仅向频繁图中加入内边 ,通过两层循环完成 ,因此 ,这部分时间复杂性为 $O(n^2)$.边的每一次扩展 ,都需要检验所生成的图是否与结果集的元素重复 ,因此 ,算法总的时间复杂性为 $O(2^n\ n^2)$.

在本算法中 ,仅考虑在稳定的图集中进行频繁子图挖掘 .因此 ,图集 GD 及 GD 中的频繁边集都是确定的 .由定理 2.3 可知 ,GraphGen 的挖掘结果集是完整的 .

由于算法 1 与算法 2 是顺序完成的 ,因此 ,算法的整体时间复杂性为 $O\left(2^n \cdot \frac{n^{2.5}}{\log n}\right)$.

表 3 给出了 FTGen 和 GraphGen 与其他算法时间复杂性的比较值 .可以看出 ,本算法的优点是将整体挖掘过程分为两个部分 .在挖掘频繁子树的部分 ,我们的算法在子图同构的计算上节省了时间 ;而在由树向图扩展的过程中 ,我们在扩展边的过程中节省了时间 ,使整体时间复杂性与其他算法相比提高了 $O(\sqrt{n} \cdot \log n)$ 倍.

Table 3 Complexity of graph mining algorithms

表 3 图挖掘算法的时间复杂性分析

	扩展边的复杂性	同构测试复杂性	总复杂性
gSpan	$O(2^n)$	$O(2^n)$	$O(2^n \cdot n^2)$
CloseSpan	$O(2^n)$	$O(2^n)$	$O(2^n \cdot n^2)$
FFSM	$O(m^4 n \cdot n^2)$	$O(m^2)$	$O(n^3 \cdot n^2)$
FTGen	$O(2^n)$	$O\left(\frac{n^{2.5}}{\log n}\right)$	$O\left(2^n \cdot \frac{n^{2.5}}{\log n}\right)$
GraphGen	$O(n^2)$	$O(2^n)$	$O(n^2 \cdot n^2)$

2.4 算法的优化

算法 2 的第 3 部分 ,在边的扩展方式上较之前的算法进行了较大的改进 ,但仍然需要进行大量的子图同构验证 (第 17 行). 由于我们的算法是由频繁子树向图的扩展 ,因此对具有不同节点数的子树 ,不会产生相同的频繁子图 .基于这个特点 ,我们可以对算法 2 进行相应调整 ,使其在生成频繁子图的过程中 ,对同构子图的检验次数降至最低 .

算法 3. 对 GraphGen 同构测试部分的改进 .

(1) k←0; /* k 用于存储树的节点数 */

```
(2) for     T 中的每棵树    t
(3)         if  k<(t 中的节点数   ) then
(4)             k←t 的节点数  ;
(5)             FG←FG∪G; /* G 为临时频繁子图集合     */
(6)             G←Φ;
(7)       endif;
(8)         g←t;
(9)         E′←{e 是频繁内边   ,并能在图集中找到    (g?e)};
(10)   for      E 中的每条边
(11)            g←g?e;
(12)            if  g≠min(g) then break;
(13)            if  G 中无   g 的同构子图    then   G←G∪g;
(14)        endfor;
(15) endfor;
```

算法 3 是算法 2 中第 10 行~第 19 行循环部分的改进 .

算法 3 在第 3 行加入了对频繁子树集合 T 中子树节点数的检验 ,并将具有相同节点数的频繁子图放入中间集 G 中.频繁子树集合 T 是按照节点数增加的顺序排列的 .在生成频繁子图的过程中 ,如果子树的节点数相同 ,则直接在 G 中检验重复子图即可 ,如果子树的节点数在下一次循环中增加 ,则将 G 中频繁子图并入结果集 FG 并置空 ,加入重新生成的频繁子图 .通过上述办法 ,可最大限度的降低用于同构子图检验的时间 .

3　实验结果与分析

实验的运行环境为单 CPU PC 机,CPU 是主频为 3.2G 的 Intel Pentium IV 超线程处理器 ,内存为 512M, 硬盘为 120G,操作系统是 Windows XP Professional SP2 CN. 实验分别采用模拟数据和真实数据 .模拟数据来自 Yan 等人提供的数据模拟器 ,而真实数据则采用 DTP(developmental therapeutics program) 提供的化合物集合 .

实验从算法的执行效率与最终结果集尺寸两方面将 GraphGen 与 gSpan 和 FFSM 进行比较与分析 .针对算法执行效率的分析 ,我们将分别改变支持度和频繁边集的大小来观察算法的运行结果 ,而且,也通过改变图集的规模对算法的效率进行考查 .通过比较最终结果集大小 ,可以从另一个侧面证明算法的正确性 .

3.1　模拟数据集上的实验结果与分析

模拟数据集由数据模拟器进行模拟 ,该模拟器由 Yan 等人提供 ,表 4 中列举了该数据模拟器使用到的参数 .

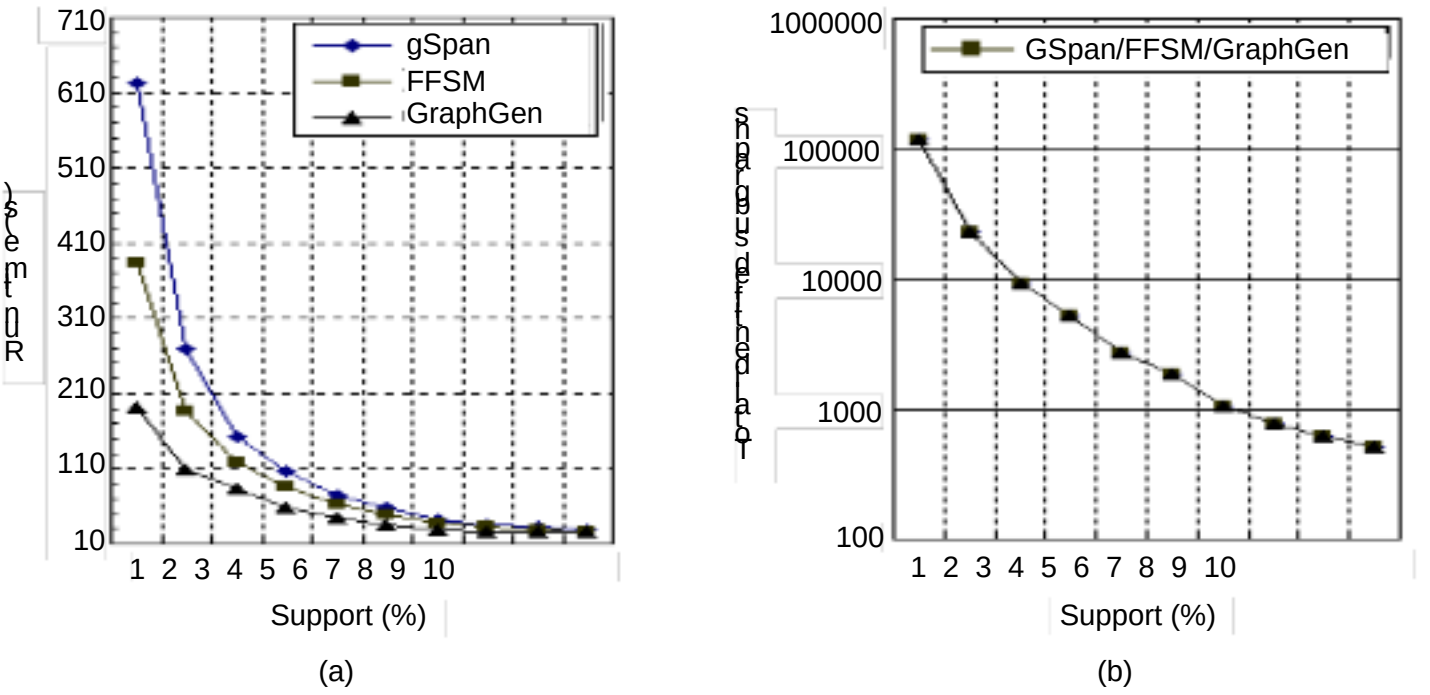
Table 4　Parameter of data generator

表 4　数据模拟器的可选参数

参数	意　义
D	生成图的总数
E	边的标号个数
i	频繁子图的平均尺寸
L	频繁子图种子个数 ,生成过程中会扩展或分割等
S	将频繁子图尽量多的放入每个生成图中 ,可互相覆盖
T	图的平均尺寸 (边数)
V	节点的标号个数
s	用于控制随机情况

在进行数据模拟的时候 ,我们采用类似 “D10kT30L200i11V4E4”的文件名来记录生成的数据 ,如 D10kT30L200i11V4E4 表示生成图的总数为 10 000 个(D10k), 图的平均尺寸为 30 条边 (T30), 频繁子图的种子个数为 200 个(L200), 频繁子图的平均尺寸为 11 条边 (i11), 节点的标号个数为 4 个(V4), 边的标号个数为 4 个(E4), 其他生成条件按默认值进行 .

图 3 中展示了对模拟数据 D10kE40I5T20L200 的试验结果 .横坐标表示支持度阈值 ,图 3(a)的纵坐标表示运行时间 ,单位为秒 ,图 3(b) 的纵坐标表示挖掘出频繁子图的个数 .在图 3(a)中我们看到 ,GraphGen 在运行时间上比 FFSM 和 gSpan 小得多 ,在支持度为 1%和 2%的时候 ,GraphGen 的效率较 gSpan 提高了 3~4 倍 .随着支持度的增加 ,3 个算法的运行时间趋于一致 ,这是由于支持度增加后频繁子图数量迅速降低降造成的 .当支持度超过 10%以后 ,算法的运行时间降低的比较缓慢 .图 3(b) 显示了挖掘算法得到的结果集的大小 ,从图中可以看出 ,这 3 个算法得到的结果集相等 .为较好地显示数量与支持度的关系 ,这个图的纵坐标使用了对数坐标 .

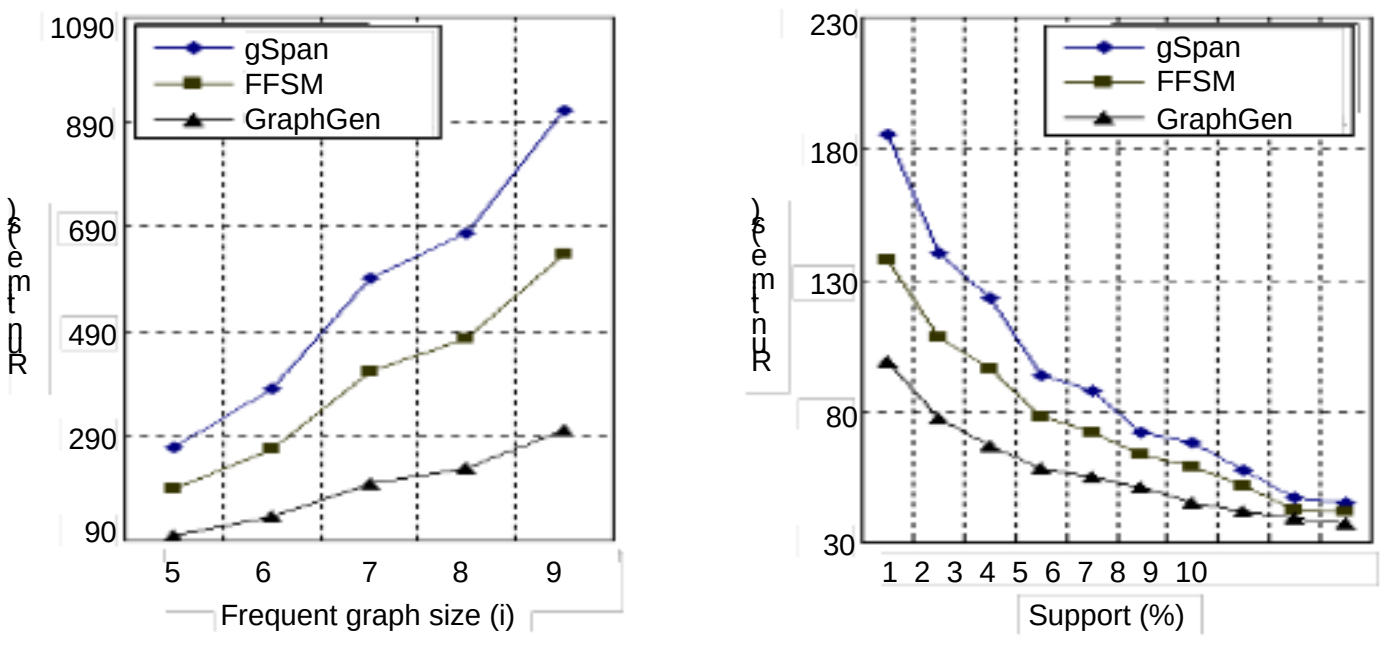


(a) (b)

Fig.3 Mining result of simulated data set

图 3 对模拟数据的挖掘结果

在图 4(a)中 ,我们通过变化所生成频繁子图的规模来对这 3 种算法进行比较 .试验方法是保持支持度为 2%不变 ,并通过改变数据集的模拟参数 i 来控制频繁子图的规模 ,分别将其边数的平均值控制在 5~9. 如图所示 ,随着频繁子图平均规模的扩大 ,gSpan 与 FFSM 的运行时间扩大的较快 ,而 GraphGen 则相对平稳 .图 4(b) 所示为模拟数据集 D100kV30I5T20L200 的挖掘结果 ,该实验使用了更大规模的数据集 D(100 000 个图),但对边的标号数量未做限制 .如图所示 ,虽然图集的规模变的更大 ,但由于边的标号个数降低 ,导致频繁边集数量下降 ,使算法的总体运行时间也随之降低 .同时 ,当支持度较小时 ,GraphGen 的效率依然高于 gSpan 三至四倍左右 .



(a) 频繁边集变化结果 (b) D100kV30I5T20L200

(a) 频繁边集变化结果 (b) D100kV30I5T20L200

Fig.4 Mining result of simulated data set

图 4 模拟数据挖掘结果

3.2　真实数据集上的实验结果与分析

实验的真实数据采用由美国国家癌症研究院(NCI)公布的实验数据,数据可从以下网址获得:http://dtp.nci.nih.gov/docs/aids/aids_data.html.DTP　　中的化合物可分为　　3　个部分　,分别是　CA(confirmed active), CM(confirmed moderately)　和　CI(confirmed inactive).　这个分类是这些化合物对　　HIV　病毒的作用　,其中,CA　含有 617 种化合物　,CM　含有　1 195　种化合物　,而　CI　中则含有　42 038　种化合物　.我们的实验使用了　　CA　与　CM　两类化合物,分别称为　DTP CA　和　DTP CM.DTP CA　中化合物节点数与边数的最大值分别是　　190　和　196.DTP CM　中化合物的最大节点数与边数分别为　　220 个节点和　234 条边.

我们将试验结果的比较通过图　　5 和图　6　给出.图　5　是使用　DTP CA　的试验结果　,左侧是　gSpan,FFSM　和 GraphGen　这　3 种算法运行时间与支持度阈值的关系曲线　　,右侧是　3 个算法结果集大小的比较　　.图　6　是使用　DTP CM　的实验结果　,与图　5 相同,左侧为　3 个算法运行时间与支持度阈值的关系曲线　　,右侧是　3 个算法结果集的大小.当支持度较高时运行时间相差较小　　,我们采用对数坐标进行描述　　,使数据曲线更加清晰　　.

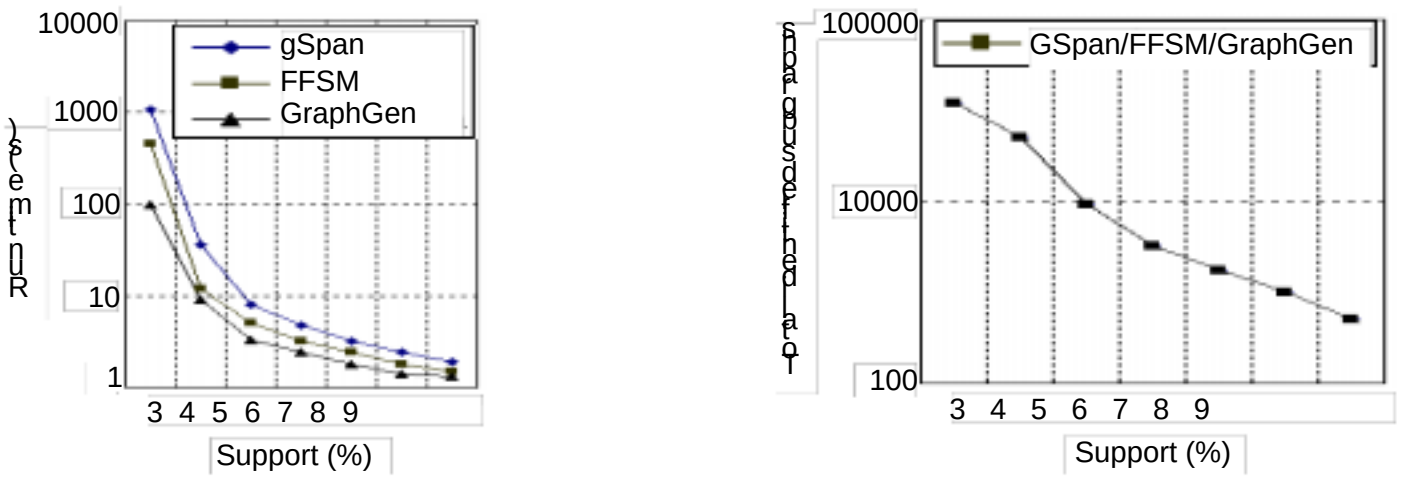


Fig.5　Mining result of DTP CA

图 5　DTP　CA 实验结果

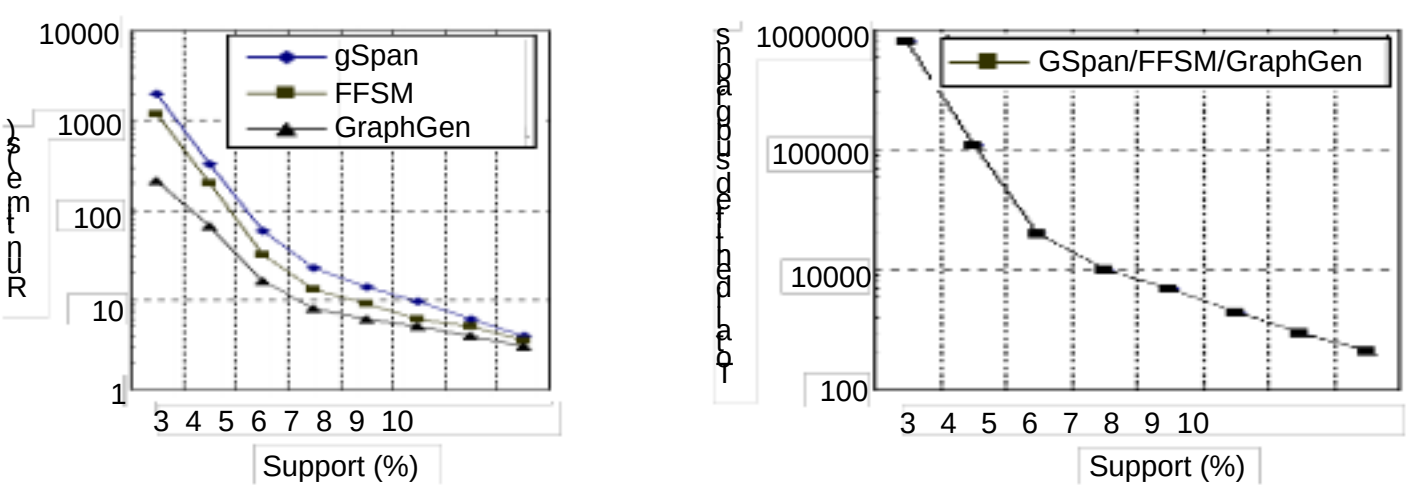


Fig.6　Mining result of DTP CM

图 6　DTP　CM 实验结果

从实验结果中可以看出　　,当支持度为　3%或 4%的时候,GraphGen 的执行效率远高于其他两种算法　　,这与我们的理论分析和前一小节中模拟数据的实验结果相一致　　.随着支持度阈值的增加　　,3　种算法的效率逐渐趋于一致　　,这是由于在支持度较高的情况下　　,从数据集中产生的频繁边集的数量也会迅速降低　　.在 DTP CM　数据集中,支持度升高至　10%的时候,算法产生的结果集接近支持度为　　3%时的　1/400.

通过对比模拟数据与真实数据的实验结果还可发现　　,当边的数量上升时　,算法的运行时间也急剧上升　　.虽然 DTP CA　和 DTP CM　的数据集规模均小于模拟数据的数据集　　,但运行时间却高于模拟数据　　,这是因为随着数据集平均边数的增加　,图的复杂程度也相应的增加　　.同时,进行图同构测试的时间复杂性为指数形式　　,导致了运行时间的急剧上升　.

4　结　论

本文提出了一种新的频繁子图挖掘算法　　GraphGen,与目前常用算法相比较　　,GraphGen　的优点在于将算法

的整体效率提高了 O(√n ?log n) 倍,这种提高来自于两个方面 :一方面 ,GraphGen 将子图同构问题部分地转化为子树同构问题 ,将时间复杂性由指数形式转化为多项式形式 ;另一方面 ,GraphGen 将图中的边分为两部分 ,这导致扩展边时的时间复杂性降低 .这两个方面恰恰是频繁子图挖掘算法中计算复杂性最高的部分 .通过不同数据的实验结果也证实了上述理论分析结果 .

References :

[1] Borgelt C, Berhold MR. Mining molecular fragments: Finding relevant substructures of molecules. In :Proc. of the ICDM 2002. 2002.

[2] Holder LB, Cook DJ, Djoko S. Substructures discovery in the subdue system. In: Proc. of the AAAI ’94 Workshop Knowledg Discovery in Databases. 1994. 169 - 180.

[3] Inokuchi A, Washio T, Okada T, Motoda H. Applying algebraic mining method of graph substructures to mutageniesis data analysis. In: Proc. of the PAKDD. 2000.

[4] Inokuchi A, Washio T, Okada T. An apriori-based algorithm for mining frequent substructures from graph data. In: Proc. of the PKDD 2000, LNAI 1910. 2000. 13 - 23.

[5] Kuramochi M, Karypis G. Frequent subgraph discovery. In: Proc. of the ICDM 2001. 2001.

[6] Yan Y Han J. gSpan: Graph-Based substructure pattern mining. In: Proc. of the 2002 Int ’l Conf. on Data Mining (ICDM 2002). Maebashi, 2002.

[7] Washio T, Motoda H. State of the art of graph-based data mining. In: Proc. of the SIGKDD 2003.

[8] Yan X, Han J. CloseGraph: Mining closed frequent graph patterns. In: Proc. of the 9th ACM SIGKDD Int ’l Conf. on Knowle Discovery and Data Mining (KDD2003). Washington, 2003.

[9] Han J, Wang W, Prins J. Efficient mining of frequent subgraphs in the presence of isomorphism. In: Proc. of the IEEE Int on Data Mining (ICDM 2003). 2003.

[10] Jin R, Wang C, Polshakov D, Parthasarathy S, Agrawal G. Discovering frequent topological structures from graph datasets. In: Proc. of the KDD 2005. Chicago, 2005. 606 - 611.

[11] Inokuchi A. Mining generalized substructures from a set of labeled graphs. In: Proc. of the IEEE Int ’l Conf. on Data M 2004), 0-7695-2142-8/04.

[12] Cohen M, Gudes E. Diagonally subgraphs pattern mining. In: Proc. of the DMKD 2004. Paris, 2004.

[13] Shamir R, Tsur D. Faster subtree isomorphism. In: Proc. of the IEEE ’97, 0-8186-8037-7/97.

[14] Ueda N, Aoki-Kinoshita KF, Yamaguchi A, Akutsu T, Mamitsuka H. A probabilistic model for mining labeled ordered trees: Capturing patterns in carbohydrate sugar chains. IEEE Trans. on Knowledge and Data Engineering, 2005,17(8):1051.

[15] Buss SR. Alogtime algorithms for tree isomorphism, comparison and canonization, In: Computational Logic and Proof Theory, Proc. of the 5th G?del Colloquium ’97, Lecture Notes in Computer Science #1289. Berlin: Springer-Verlag, 1997:33.

[16] Yang R, Kalnis P, Tung AKH. Similarity evaluation on tree-structured data. In: Proc. of the SIGMOD 2005. Baltimore, 2005.

[17] R ü ckert U, Kramer S. Frequent free tree discovery in graph data. In: Symp. on Applied Computing archive, Proc. of the 2004 ACM Symp. on Applied Computing. SESSION: Data Mining (DM). 2004. 564 - 570.

李先通 (1973－),男 ,黑龙江哈尔滨人 ,博士生 ,主要研究领域为频繁模式挖掘 .

李建中 (1950－),男 ,博士 ,教授 ,博士生导师 ,CCF 高级会员 ,主要研究领域为数据库 ,并行计算 .

高宏 (1966－),女 ,博士 ,教授 ,博士生导师 ,主要研究领域为数据库 ,数据仓库 ,计算生物学 .