维普资讯 http://www.cqvip.com

基于化合物库测试的gSpan算法

许荣斌,谢 莹,吴建国
(安徽大学 计算智能与信号处理教育部重点实验室,安徽 合肥 230039)

摘 要:gSpan算法是一种基于频繁图的数据挖掘算法。该算法基于无候选人产生的频繁子图,采用深度优先搜索策略挖掘频繁连接子图。由于其设计结构具有连续性以及无候选人产生,算法的性能得以提高,在执行速度上可以达到前人算法如FSG算法的15~100倍。基于化合物库Chemical_340测试发现,该算法能够以卓越性能有效挖掘频繁子图。该算法可以应用在搜索具有相同子结构的化合物研究中,对相关领域研究发展具有重要意义。

关键词:gSpan;化合物库;频繁子图;深度优先搜索
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2007)10-0058-03

The gSpan Algorithm Based on Compound-Library Testing

XU Rong-bin, XIE Ying, WU Jian-guo
(Ministry of Education Key Laboratory of Intelligent Computing & Signal Processing, Anhui University, Hefei 230039, China)

Abstract: Introduces a graph-based substructure pattern mining algorithm called gSpan, which discovers frequent substructures without candidate generation and adopts the depth-first search strategy to mine frequent connected subgraphs efficiently. Its performance is enhanced because the continuous design and non-candidate. When it is applied on the chemical compound-library Chemical_340, gSpan substantially outperforms previous algorithms such as FSG, sometimes by an order of magnitude. gSpan can be applied in the research of finding compounds with same substructure, it is very important to related areas.

Key words: gSpan; compound-library; frequent subgraphs; DFS

0 引 言

图形数据广泛存在于生活之中,如化学、生物等学科和国防等领域都大量使用图形数据,因此,对基于图的频繁子图挖掘算法的研究非常必要。基于图的数据挖掘算法提出时间并不长,但是由于图论作为数学的一个研究领域已经有很长的历史,所以频繁子图挖掘发展很快,并被广泛应用到许多领域之中,如通过频繁子图挖掘算法找出构成有毒物质的分子结构,以及通过对网站浏览日志的挖掘,分析出最频繁的浏览模式等。

目前,化合物结构在计算机中表达方式主要有两种类型:一类是线性表达式系统,此系统是应用简单编码规则对化合物的结构式进行编码,形成线性表达式。该系统较为简洁、唯一、单一,但是用其表达二维结构有一定困难。另一类是拓扑表达法,将化合物结构式看成图,将原子看成边,用关联矩阵来表示其结构式。通过Morgan排序等方式可以使它具有单一性。

收稿日期:2006-12-17
基金项目:国家科委创新基金资助项目(06C26213401229)
作者简介:许荣斌(1981-),男,安徽黄山人,硕士研究生,研究领域为嵌入式系统设计;吴建国,教授,博士生导师,研究领域为中文信息处理、嵌入式系统DA。

1 gSpan算法

1.1 一些概念

频繁子图(frequent subgraph):假定输入数据库 $G=\{G_i \mid i=0,1,\cdots,n\}$,给定一个最小支持度阈值minSup,规定:如果子图 g 与 G_i 子图同构,则 $o(g,G_i)=1$,否则 $o(g,G_i)=0$,$\delta(g,G)=\sum_{G_i\in G}o(g,G_i)$。如果 $\delta(g,G)\geqslant$ minSup,则 g 是一个频繁子图。

DFS词典序:假设 $Z=\{\text{code}(G,T) \mid T$ 是 G 的一个DFS树$\}$。如图1所示。假定在标号集合(L)中有一个线性序列$(\prec L)$,则 $\prec T$ 和 $\prec L$ 的词典组合是一个在集合 $E_T\times L\times L\times L$ 上的线性序列$(\prec e)$[1]。DFS词典序定义如下:如果 $\alpha=\text{code}(G_\alpha,T_\alpha)=(a_0,a_1,\cdots,a_m)$ 而且 $\beta=\text{code}(G_\beta,T_\beta)=(b_0,b_1,\cdots,b_n)$,$\alpha,\beta\in Z$,则 $\alpha\leqslant\beta$ 当且仅当下面条件成立:

(1) $\exists t$, $0\leqslant t\leqslant\min(m,n)$, $a_k=b_k$, $k<t$, $a_t\prec$

维普资讯 http://www.cqvip.com

$e\ b_t$

(2)$a_k = b_k, 0 \leqslant k \leqslant m$ 且 $n \geqslant m$

最小DFS代码:给定一个图 G,$Z(G) = \{\text{code}(G, T) \mid T$ 是G 的一个DFS树$\}$,基于DFS词典序,最小的为 $\min(Z(G))$,称为 G 的最小DFS代码。

mini Code:首先对图进行深度优先搜索,形成一棵DFS Tree(如图1所示),然后根据 $\prec T$ 来扫描该图,扫描边的顺序就构成了一个序列,称为 DFS Code,不同的 DFS Code 之间按照字典顺序排序。对这个图的所有DFS Code进行排序,找出最小 DFS Code 来唯一标识这个图,这个最小 DFS Code 被称为 mini Code。

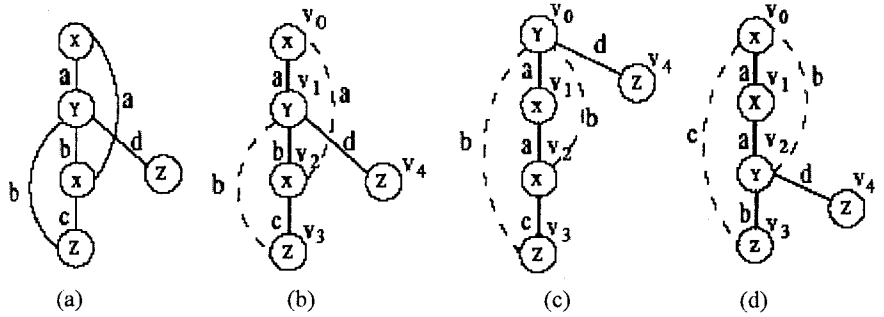

图 1　深度优先搜索树

空间树:将图的 mini Code 作为一个空间树的节点,对该节点进行最右路径扩展,把生成的新子图作为该节点的孩子,同一父节点的不同子节点之间是按照字典顺序排序,这样就生成了一棵空间树。如果将该空间树的所有非频繁子节点剪枝就形成了一棵频繁子图树。

Occurrence:如果子图 g 与输入数据库中图 G 具有子图同构关系,则 G 被称为 g 的 occurrence。

1.2　gSpan 算法思想

Kuramochi 等人曾研究出一种应用图邻近表示和边增长策略的挖掘算法——FSG 算法[1]。FSG 算法利用逐层演绎方式,是一种类 Apriori 算法[2]。而类 Apriori 算法在频繁子图挖掘应用中面临两个问题:

(1)候选人产生问题:与关联算法相比,FSG 产生候选人算法更复杂,代价更巨大;

(2)修剪主动错误问题:FSG算法中子图同构是一个 NP 完全问题,修剪主动错误代价巨大。gSpan 算法将深度优先搜索策略应用到频繁子图挖掘中,能够显著减少上文所提到的巨大开销。该算法提出 DFS 词典序和最小 DFS Code 两个技术,建立 canonical label 来支持 DFS 搜索,改善了上文提到的无候选人的频繁子图挖掘和主动错误修剪存在的问题。

假定存在标号集合 $\{A, B, C, \cdots\}$ 为顶点,$\{a, b, c, \cdots\}$ 为边。在算法 1 中,第一次循环将找到所有包含边 $A \xrightarrow{a} A$ 的频繁子图。第二次循环将找到所有包含 $A \xrightarrow{a} B$,但不包含任何 $A \xrightarrow{a} A$ 的频繁子图。循环重复执行直到找到所有的频繁子图。当这个过程循环执行时,数据库规模会缩减,而子图规模会增加。Subgraph_Mining 函数发现当支持度小于 minSup,或其代码不是最小代码时(即这个图和它的所有后代都已产生并都已被挖掘)即停止搜索,此递归函数能够找到所有频繁子图[3~5]。

gSpan 算法思想如下所述:

1)计算所有边的支持度,将所有非频繁边从输入数据库中删除,并以频繁边作为初始子图。

2)产生候选子图:对 k 频繁子图的 mini Code 的 DFS Tree 进行最右路径扩展,每次添加一条边,得到 k+1 候选子图。该方法经证明能够保证挖掘结果的完整性。

3)剪枝:如果 k+1 候选子图不是 mini Code 形式的,则认为该图是冗余的,从候选子图中删除。如图 2 所示。

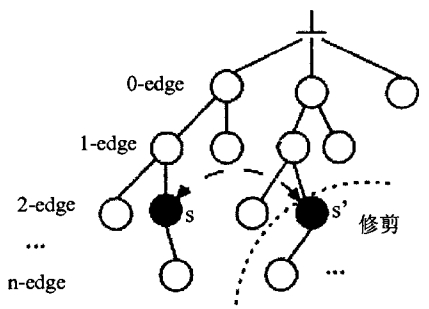

图 2　DFS Tree 搜索空间

4)每次在计算 k 频繁子图的支持度的时候,同时记录 k 频繁子图的所有 occurrence。这样,k+1 候选子图的支持度就可以通过对 k 频繁子图的所有 occurrence 进行最右路径扩展获得[6]。

5)缩减数据库:当某条频繁边的所有子节点都生成后,就将该边从输入数据库中删除,缩减输入数据库。

算法 1 GraphSet-Projection(D, S)

1:以频繁度排序 D 中的标号;

2:移出不频繁的定点和边;

3:重标号剩余的顶点和边;

4:$S^1 \leftarrow$ 所有 D 中频繁 1 - edge 图;

5:S^1 以 DFS 词典序进行排序;

6:$S \leftarrow S^1$;

维普资讯 http://www.cqvip.com

7:对每一个 $e \in S^1$ 的边循环操作
8:用 e 初始化 s,用包含 e 的图设置 $s.D$
9:Subgraph_ Mining(D,S,s);
10:$D \leftarrow D - e$;
11:如果 $|D| <$ minSup;
12:Break;
子过程 1 Subgraph_ Mining(D,S,s)
1:If $s! = \min(s)$
2:Return;
3:$S \leftarrow S \cup \{s\}$;
4:列举 D 中每一个图内的 s,计算其孩子数目;
5:for 每一个 c,c 是 s 的孩子 do
6:If support(c) >= minSup
7:$s \leftarrow c$;
8:Subgraph_ Mining(Ds,S,s);

2 基于化合物库 Chemica_340 性能测试

文中采用的实验化合物库属于一种稀疏数据库,其中包含 340 种化合物,24 种原子,66 种原子类型和 4 种类型的结合物。转换为拓扑表达法相当于每个图平均包含 27 个节点和 28 条边,其中最大的图包含 214 条边和 214 个顶点。化合物库将原子和结合物的类型作为 canonical label[7,8],文中采用化学化合物库 Chemical_340 对算法进行性能测试,应用 gSpan 算法挖掘库中频繁子图。测试 PC 机配备有 512MB 内存、Red Hat Linux 9 操作系统、Intel Pentium IV。性能测试运行界面如图 3 所示。

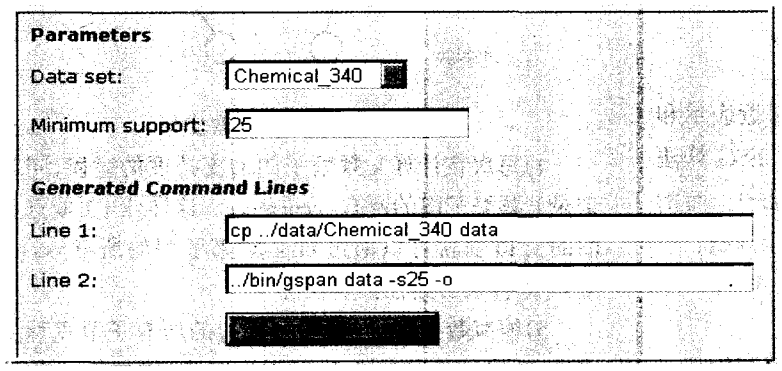

图 3 挖掘频繁子图运行界面

部分代码如下:

```
require 'chem'
# Load
puts "Load molecules"
mols=[]
1.upto(100) do |i|
  filename="/home/tanaka/share/data/kegg/ligand/mol/C%
05d.mol"% i
  mols.push Chem.open-mol(filename) if File.exist?(filename)
  print"."
end
puts
puts "Load completed!"
# Calculate with gSpan
filename="temp/temp.gspan.#{Process.pid}.0"
filename.succ! while File.exist?(filename)
filename=filename +".fp"
Chem.save(mols, filename)
system("gSpan #{filename} -o -s32")
freqs=Chem.open_mol("#{filename}.fp")
# Save Image
puts
puts "Now save images in temp/"
mols.each_with_index do |mol,i|
m=mol.match_by_ullmann(freqs[10])
  if m
    m.each do |index|
      mol.nodes[index].visible=true
    end
    # need rmagick or use pdf
    mol.save(File.join("temp","mol_#{i}.png"))
  end
End
```

运行结果如图 4 所示。gSpan 算法以 0.44 秒的耗时,25% 的支持度,在 Chemical_340 化合物库中挖掘到 1142 个频繁子图。

3 结 论

文中描述了一种基于频繁子图挖掘的 gSpan 算法。该算法利用深度优先搜索策略构建词典序的 canonical label,大大减少了冗余候选子图的产生,并避免了大量重复扫描数据库。gSpan 算法设计结构具有连续性以及无候选人产生,降低了其空间复杂度和时间复杂度。实验可证明 gSpan 算法性能比前人算法更优越,例如在执行速度上可以达到 FSG 算法的 15~100 倍。基于化合物库 Chemical_340 测试发现,该算法能够以卓越性能有效挖掘频繁子图,可应用在挖掘具有相同子结构的化合物研究中,如未知化合物毒性预测等。对相关领域研究发展具有重要意义。

下一步,将进行 gSpan 算法在未知化合物毒性预测方面的应用研究工作。

(下转第 64 页)

维普资讯 http://www.cqvip.com

化算法对于光滑的物体表面重建效果较理想。

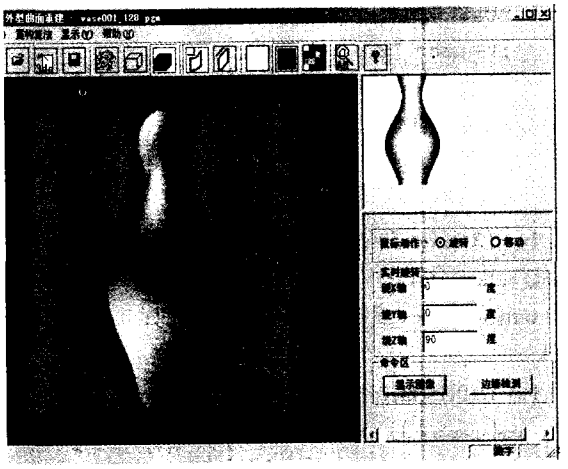

图 4 系统界面及重建效果

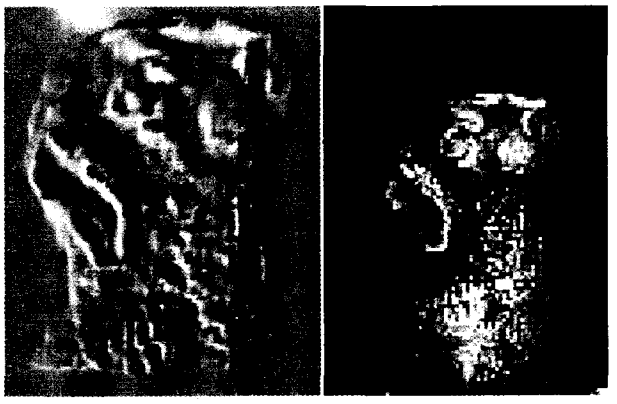

图 5 真实图像的重建效果
(左为重建后的模型,右为图像)

4 结论与展望

对 SFS 算法进行了研究分析,并提出一种利用 SFS 进行曲面外形曲面的重建的方法,在此基础上,利用 VC 和 OpenGL 开发了一个重建平台,可以实现一般曲面的重建。该系统基本可以快速地重建出物体的外形,或者先重建出各个部件,然后导入 3D Max 等其它成熟的三维造型软件中进行编辑修改,最后组装成整机。但该系统也存在一些不足,所以提出以下几点展望:1) 系统中采用的重建算法有待进一步改进,以实现任意光照下的灰度图像的三维重建。2) 提高重建精度,特别是重建误差太大,需要进一步研究。

参考文献:

[1] Zhang R, Tsai P S, Cryer J, et al. Shape from Shading: A Survey[J]. IEEE Transactions on PAMI, 1999, 21(8): 690 - 706.

[2] Ragheb H, Hancock E R. Surface radiance correction for shape from shading[J]. Pattern Recognition, 2005, 38: 1574 - 1595.

[3] Horn B, Brooks M. The Variational Approach to Shape from Shading[J]. Computer Vision, Graphics and Image Processing, 1986, 33(2): 174 - 208.

[4] 廖 熠,赵荣椿. 从明暗恢复形状(SFS)的几类典型算法分析与评价[J]. 中国图象图形学报, 2001, 6A(10): 953 - 960.

[5] 费广正,卢丽丹,陈立新. 可视化 OpenGL 程序设计[M]. 北京:清华大学出版社, 2001: 36 - 37.

(上接第 60 页)

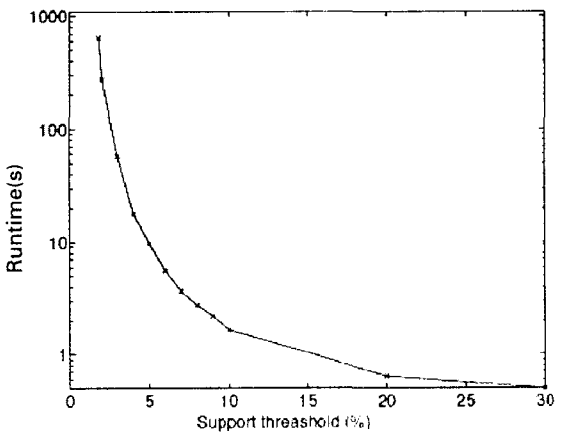

图 4 gSpan 算法性能曲线图

参考文献:

[1] Yan Xifeng, Han Jiawei. gSpan: Graph - based substructure pattern mining[R]. Madrid: Department of Computer Science, URJC, 2002.

[2] Agrawal R, Srikant R. Fast algorithms for mining association rules[M]. New York: IBM Almaden Research Center, 1994: 487 - 499.

[3] Kuramochi M, Karypis G. Frequent subgraph discovery[C]// ICDM2001. California: [s. n.], 2001: 313 - 320.

[4] Asai T, Kawasoe S, Kawasoe S. Efficient substructure discovery from large semistructured data[C]//SDM2002. Arlington, VA, USA: SLAM, 2002: 225 - 227.

[5] Pei J, Han J. PrefixSpan: Mining sequential patterns efficiently by prefix - projected pattern growth[C]//ICDE2001. Neuss: [s. n.], 2001: 215 - 224.

[6] SUN L, ZHANG X. Efficient Frequent Pattern Mining on Web Logs[C]//APWeb2004. Hangzhou: [s. n.], 2004: 533 - 542.

[7] Zaki M J. Efficiently mining frequent trees in a forest[C]// KDD2002. Edmonton: [s. n.], 2002: 135 - 139.

[8] Cormen T H, Leiserson C E, Rivest R L, et al. Introduction to Algorithms[M]. 2nd Edition. Massachusetts: MIT Press, 2001: 12 - 14.