Q-CFIsL:挖掘频繁闭项集并构建其格的快速算法

李学明[1] 余 春[1] 张 贺[1] 江 泓[2]
(重庆大学计算机学院 重庆 400044)[1] (Nebraska-Lincoln大学计算机科学与工程系 美国林肯)[2]

摘 要 提出了一种快速挖掘频繁闭项集并构建其格的算法Q-CFIsL。该算法引入了*preC*(*X*)的概念,使用*preC*(*X*)加快了包容检测和建格的速度。实验表明,对于真实数据集以及合成数据集,Q-CFIsL的性能都优于当前最新的同类算法CHARM-L。

关键词 关联规则,频繁闭项集,频繁闭项集格

Q-CFIsL: A Fast Algorithm for Mining Closed Frequent Itemsets and Building their Lattice

LI Xue-ming[1] YU Chun[1] ZHANG He[1] JIANG Hong[2]
(School of Computing, Chongqing University, Chongqing 400044, China)[2]
(Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, USA)[2]

Abstract A fast algorithm Q-CFIsL for mining closed frequent itemsets and building their lattice was presented. It introduced *prec*(*X*), which can make the subsumption cheching and latiice building more efficiently. Experimental results show that Q-CFIsL outperforms the currently latest similar algorithm CHARM-L.

Keywords Association rule, Closed frequent itemset, Closed frequent itemset lattice

1 引言

关联规则挖掘是数据挖掘中的重要研究方向之一,相关研究主要集中在两个方面:一是研究关联规则的语义,即规则的有用性;二是研究如何提高关联规则的挖掘效率。目前,大多数的研究主要集中在后者。

关联规则挖掘的传统算法主要分为两步:首先找出所有的频繁项集,然后由频繁项集产生满足最小置信度和最小支持度的规则。此类算法以Apriori[1]为代表。然而,实际应用中的某些数据集具有海量的频繁项集,要找出它们不但效率不高,甚至可能无法完成挖掘任务。解决这一问题有两种方案:一是通过挖掘最大频繁项集[2,8]来产生关联规则,但是最大频繁项集会造成信息丢失,故不可行;二是通过挖掘频繁闭项集来产生关联规则。对于给定的数据集,其产生的频繁闭项集远少于其产生的频繁项集;而且由频繁闭项集能得到所有的频繁项集,由频繁闭项集所产生的关联规则能得到所有的关联规则[6,7]。因此目前广泛采用的是第二种方案。

在用频繁闭项集产生关联规则时,需要找到它所有的子集。而现有的绝大多数算法只是将挖掘到的频繁闭项集简单地存储起来,要找子集需要较长的时间,从而降低了关联规则的挖掘效率。为了解决这个问题,可在挖掘频繁闭项集的同时,将项集之间的包含关系用格(lattice)保存起来[11]。

当前最有影响的频繁闭项集挖掘算法[16]有A-close[3],Closet[5],Closet+[9],FP-Close[15],CHARM[4],DCI-Closed[14],LCM[13]。它们通常分为两步:首先产生候选频繁闭项集,然后判断该项集是否闭合。然而它们均未构建频繁闭项集格。CHARM-L[11]是CHARM的拓展,是第一个挖掘频繁闭项集并构建频繁闭项集格的算法。其主要思想是:首先通过CHARM Properties[11]生成一个候选频繁闭项集X;然后在已挖掘到的频繁闭项集中找到X的所有超集,将其记为$s(X)$;最后判断是否存在$Y\in s(X)$使得$sup(X)=sup(Y)$,即包容检测(Subsumption Checking):如果存在,那么X不闭合,将其删除;如果不存在,那么将X加入到已挖掘的频繁闭项集中,并在$s(X)$中找到它的所有直接超集,分别与之建立联系,即建格。CHARM-L有几个不足之处:首先,得到$s(X)$需要进行大量的计算;其次,$|s(X)|$通常比较大,用其进行包容检测和建格效率不高。

为了克服上述不足,本文提出了快速挖掘频繁闭项集并建格的高效算法Q-CFIsL。其主要创新之处有:

①Q-CFIsL引入了*preC*的概念。通常,$|preC(X)|$远小于$|s(X)|$,使用$preC(X)$能够极大地提高包容检测以及建格的效率;

②在生成候选项集的同时,能快速地得到*preC*,从而保证了Q-CFIsL的高效性。

实验表明:Q-CFIsL在性能上明显优于CHARM-L,而且最小支持度越小,这种优势就越明显。

本文的其余部分组织如下:第2部分主要介绍基本概念;第3部分详细介绍preC的定义、性质以及如何快速得到preC;第4部分介绍Q-CFIsL算法并通过实例说明其工作流程;第5部分实验分析;最后是总结。

2 基本概念

设$I=\{i_1,i_2,\cdots,i_n\}$是一组项目的集合,$D=\{t_1,t_2,\cdots,$

到稿日期:2008-03-03 本文受重庆市自然科学基金(CSTC, 2007BB2178)资助。
李学明(1967-),男,副教授,主要研究领域为数据挖掘、神经网络、网络与网格计算,E-mail:lixuemin@cqu.edu.cn。

$t_n\}$是一组事务的集合。由 $k(k>0)$个项目组成的集合称为 k-项集(k-itemset),称 k 为该项集的长度。一条事务可用二元组$\langle tid,X\rangle$表示,其中 tid 是其标识,X 是其对应的项集。由标识组成的集合称为标识集($tidset$)。如果项集 $X\subseteq Y$,那么称事务$\langle tid,Y\rangle$支持 X。D 中支持 X 的事务数称为 X 的支持度,记为 $sup(X)$。支持度大于或等于最小支持度的项集称为频繁项集。图 1 给出示例数据集。

定义 1 对于项集 X,令 $t(X)=\{tid \mid tid$ 是 t 的标识,$t\in D$ 且 t 支持 $X\}$;对于标识集 Y,令 $i(Y)=\bigcap_{y\in Y} itemset(y)$,其中 $itemset(y)$表示 y 所对应的事务的项集。例如:$t(\{A,B,D\})=\{2,6\}$,$i(\{2,6\})=\{A,B,D\}\cap\{A,B,C,D,E\}=\{A,B,D\}$。简便起见,本文将用 ABD 表示$\{A,B,D\}$,用 26 表示$\{2,6\}$。

定义 2 对于项集 X,令 $c(X)=i(t(X))$。如果 $c(X)=X$,那么称 X 闭合,亦称 X 为闭项集。例如:$c(ABD)=i(t(ABD))=i(26)=ABD$,所以称 ABD 为闭项集。闭合的频繁项集称为频繁闭项集。称 $c(X)$为 X 的闭包。

定义 3 令 $CFIs$ 表示由 D 挖掘出来的所有频繁闭项集,那么称 $L=(CFIs,\leqslant)$为频繁闭项集格,其中"$\leqslant$"是 $CFIs$ 上的偏序关系。设最小支持度为 2,示例数据集的频繁闭项集格如图 2 所示。

Transcation	Itemset
1	ACDE
2	ABD
3	CDE
4	BDE
5	CE
6	ABCDE

(a) 水平形式

Item	Transcation
A	126
B	246
C	1356
D	12346
E	13456

(b) 垂直形式

图 1 示例数据集

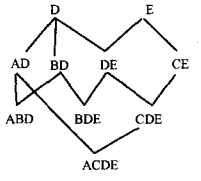

图 2 示例数据集的频繁闭项集格

对于闭项集 X,Y,如果 $X\subset Y$ 且不存在闭项集 Z 使得 $X\subset Z\subset Y$,那么称 Y 是 X 的直接闭超集。在频繁闭项集格中,项集的子节点即是它的直接频繁闭超集。因此建立频繁闭项集格也就是建立频繁闭项集与其直接频繁闭超集之间的联系。

定义 4 IT-Tree[4]是一种基于前缀的投影树,CHARM(-L)算法通过深度优先的顺序生成 IT-Tree,并同时得到所有的频繁闭项集。IT-Tree 中每个节点代表了一个项集及其对应的标识集,用 $X\times t(X)$ 表示。节点 $X\times t(X)$ 分别同其每个右兄弟 $Y\times t(Y)$进行交、并运算(项集相并、标识集相交),得其子节点,每个子节点均为候选频繁闭项集且有相同的前缀 X。交、并运算有如下性质(即 CHARM Properties):

(1) 如果 $t(X)=t(Y)$,那么 $c(X)=c(Y)=c(X\cup Y)$,这意味着可将 X 替换为 $X\cup Y$,也就是将项集 Y 并入 $X\times t(X)$及其子节点中,并将 $Y\times t(Y)$删除。

(2) 如果 $t(X)\subset t(Y)$,那么 $c(X)=c(X\cup Y)$,$c(X)\neq c(Y)$,这意味着可将 X 替换为 $X\cup Y$。

(3) 如果 $t(X)\supset t(Y)$,那么 $c(Y)=c(X\cup Y)$,$c(X)\neq c(Y)$,则生成 $X\times t(X)$的子节点 $X\cup Y\times t(X\cup Y)$,并删除 $Y\times t(Y)$。

(4) 如果 $t(X)\not\subseteq t(Y)\wedge t(X)\not\supseteq t(Y)$,那么 $c(X)\neq c(Y)\neq c(X\cup Y)$,则生成 $X\times t(X)$的子节点 $X\cup Y\times t(X\cup Y)$。

使用这 4 条性质能极大地减少候选项集的数量,从而提高频繁闭项集的挖掘速度。

如果候选项集 Y 在 CHARM Properties(1)、(3)或包容检测时被删除,那么将删除 Y 的节点记为 X_1;如果 X_1 被删除,那么将删除 X_1 的节点记为 X_2;…以此类推,直到 X_n 没被删除,则有 $c(Y)=c(X_1)=c(X_2)=\cdots=X_n$。如果在删除其中的某个节点时,只是将其标识为删除并建立链接,指向删除它的节点,那么通过删除链 $X_1\rightarrow X_2\rightarrow\cdots\rightarrow X_n$ 能快速地找到 Y 的闭包 X_n。

3 preC 的概念及其性质

定义 5 将项集 X 的 preC 定义为 $preC(X)=\{c(Xx_i)\mid x_i\in I-X\}$。

定理 1 对于项集 X,如果存在 $Y\in preC(X)$且 $sup(X)=sup(Y)$,那么 X 不闭合;反之亦然。

证明:因为 $Y\in preC(X)$,所以 $X\subset Y$,又因为 $sup(X)=sup(Y)$,所以 $c(X)=Y\neq X$,故 X 不闭合。反之,如果 X 不闭合,那么存在 $x\in I-X$ 且 $c(X)=c(Xx)$,取 $Y=c(Xx)$,故存在 $Y\in preC(X)$使得 $sup(X)=sup(Y)$。证毕。

定理 2 对于项集 X,其所有的直接频繁闭超集都在 $preC(X)$中。

证明:设 Y 是 X 的直接频繁闭超集且 $Y-X=x_1x_2\cdots x_n$,那么对 $x_i(1\leqslant i\leqslant n)$,有 $Xx_i\subset Y$,$c(Xx_i)\subseteq c(Y)=Y$。又因为 Y 是 X 的直接频繁闭超集,所以 $c(Xx_i)=Y$。由 $preC$ 的定义可知,Y 是 X 的 $preC$,即 $Y\in preC(X)$。证毕。

由定理 1 和定理 2 可知,可使用 $preC(X)$对候选频繁闭项集 X 进行包容检测以及建格。由定义 5 可知$|preC(X)|\leqslant|I-X|$,而$|s(X)|\leqslant 2^{|I-X|}$。且实验表明$|preC(X)|$远小于$|s(X)|$(如图 6 所示)。因此,使用 $preC$ 进行包容检测以及建格更为有效。

将节点 X 下的候选项集分别记为 $Xx_1,Xx_2,\cdots,Xx_n$,如图 3 所示。下面分 3 类情形阐述如何得到 Xx_i 的 $preC$。

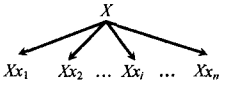

图 3 快速找寻

(1) 寻找 $c(Xxx_i)$,$x\in\bigcup_{j=1}^{i-1}x_j$。$Xx_i$ 是 Xx 的右兄弟,在 Xx 和 Xx_i 进行交、并运算时,

1)如果 Xx 已被标识为删除,那么通过删除链能快速地找到 $c(Xx)$。如果 $c(Xx)=c(Xxx_i)$,那么 $c(Xx)$即为所寻;否则在 $preC(c(Xx))$中找到 $c(Xxx_i)$,也就是在 $preC(c(Xx))$中找到包含 x_i 且长度最短的,即为 $c(Xxx_i)$。

2)如果 Xx 未被标识为删除且 $t(Xx)\subset t(Xx_i)$,那么将 Xx 替换成 Xxx_i。如果节点 Xxx_i 最终没被标识为删除,那么即为 $c(Xxx_i)$;否则通过删除链找到 $c(Xxx_i)$。

3)如果 Xx 未被标识为删除且 $t(Xx)\not\subseteq t(Xx_i)\wedge t(Xx)\not\supseteq t(Xx_i)$,那么将生成 Xx 的子节点 Xxx_i。如果节点 Xxx_i 最终没被标识为删除,那么即为 $c(Xxx_i)$;否则通过删除链找到 $c(Xxx_i)$。

(2) 寻找 $c(Xx_ix)$,$x\in\bigcup_{j=i+1}^{n}x_j$。$Xx$ 是 Xx_i 的右兄弟,在 Xx_i 和 Xx 进行交、并运算时,

1)如果 Xx 被标识为删除,那么通过删除链能快速地找

到 $c(Xx)$。如果 $c(Xx)=c(Xx_ix)$，那么 $c(Xx)$ 即为所寻；否则在 $preC(c(Xx))$ 中找到 $c(Xx_ix)$。

2)如果 Xx 未被标识为删除且 $t(Xx_i)\supset t(Xx)$，那么将生成 Xx_i 的子节点 Xx_ix。如果节点 Xx_ix 最终没被标识为删除，那么即为 $c(Xx_ix)$；否则通过删除链找到 $c(Xx_ix)$。

3)如果 Xx 未被标识为删除且 $t(Xx)\not\subseteq t(Xx_i)\wedge t(Xx)\not\supseteq t(Xx_i)$，那么将生成 Xx_i 的子节点 Xx_ix。如果节点 Xx_ix 最终没被标识为删除，那么即为 $c(Xx_ix)$；否则通过删除链找到 $c(Xx_ix)$。

(3) 寻找 $c(Xx_ix), x\in I-X-\bigcup_{j=1}^{n}x_j$。

该类情形通过 X 的 $preC$ 寻找 $c(Xx_ix)$。为了方便论述，将 $preC(X)$ 划分为两部分，分别记为 $preC_1(X)=\{c(Xx)\mid x\in\bigcup_{j=1}^{n}x_j\}$，$preC_2(X)=\{c(Xx)\mid x\in I-X-\bigcup_{j=1}^{n}x_j\}$。也就是说，$preC_1(X)$ 是 X 通过其子结点得到的那部分的 $preC(X)$，对应情形(2)中的后两种；$preC_2(X)$ 对应其余情形，包括情形(3)。显然，$preC_2(X)$ 中的每个 $c(Xx)$ 都对应着一个 $c(Xx_ix)$，如果 $c(Xx)=c(Xx_ix)$，那么即为所寻；否则，在 $preC(c(Xx))$ 中寻找 $c(Xx_ix)$。

4 Q-CFIsL 设计与实现

4.1 算法实现

根据定理 1 和定理 2，本文提出快速挖掘频繁闭项集并构建其格的算法 Q-CFIsL。其主要思想是：首先通过 CHARM Properties 生成一个候选频繁闭项集 X，同时使用上节所描述的方法快速得到 $preC(X)$；然后判断是否存在 $Y\in preC(X)$ 使得 $sup(X)=sup(Y)$：如果存在，那么 X 不闭合，将其标识为删除并建立链接指向删除它的节点；如果不存在，那么在 $preC(X)$ 中找到 X 的所有直接超集并分别与之建格。

Q-CFIsL 的伪代码如图 4 所示。它在建格的时候才通过 $nodeX$ 的子节点得到 $preC_1(nodeX)$，见代码 20－23 行。这样做有几个原因。首先，由 CHARM Properties(3)、(4)可知，通过子节点得到的 $preC$ 的支持度必定小于 $nodeX$，因此包容检测时不需要检测这部分 $preC$。其次，只有当 $nodeX$ 所有的子节点处理完之后才能通过它们得到 $preC_1(nodeX)$；最后，寻找 $nodeX$ 子节点的 $preC$ 时不需要 $preC_1(nodeX)$。$preC_2(nodeX)$ 由代码 47－54 得到，为了将情形(1)以及情形(2)中的第一种情形统一处理，Q-CFIsL 将这些情形中产生 $preC$ 的节点保存在 $protentialPreC$ 中，见代码 9,18,37,44。在包容检测时，如果 $nodeX$ 被删除，那么将不再在 $nodeX$ 分支搜索频繁闭项集，因此需将 $nodeX$ 的子节点标识为删除并建立链接指向它，以便其右兄弟通过删除链找寻 $preC$，见代码 55－60。

```
QCFIsL(D,minsup)
1. for each li∈I∧sup(li)≥minsup
2.     rootNode.children.Add(li×t(li))
3. QCFIsL_Extend(rootNode)
QCFIsL_Extend(rootNode)
4. for each nodeX∈rootNode.children{
5.     if (nodeX.flag! =DEL){
6.       for each nodeY∈nodeX.rightBrother
7.           if (nodeY.flag! =DEL)
8.               Charm_Property_L(nodeX,nodeY)
9.           else nodeX.protentialPreC.Add(nodeY)
10.      SubsumptionCheck(nodeX)
11.      if (nodex.flag! =DEL){
12.          QCFIsL_Extend(nodeX)
13.          BuildLattice(nodeX)
14.      }
15.    } else
16.      for each nodeY∈nodeX.rightBrother
17.          if (nodeY.flag! =DEL)
18.              nodeY.protentialPreC.Add(nodeX)
19. }
BuildLattice(nodeX)
20. for each child∈nodeX.children {
21.     while (child.flag==DEL) child=child.link
22.     nodeX.preC1.Add(child)
23. }
24. for each node∈nodeX.preC1∪node.preC2{
25.     if (node是直接频繁闭超集){
26.         nodeX.lChildren.Add(node)
27.         node.lParents.Add(nodeX)
28.     }
29. }
Charm_Property_L(nodeX,nodeY)
30. if (sup(nodeX.tids∩nodeY.tids)≥minsup) {
31.     if (nodeX.tids⊆nodeY.tids) {
32.         nodeX.items=nodeX.items∪nodeY.items
33.         for each child∈nodeX.children
34.             child.items=child.items∪nodeY.items
35.         if (nodeX.tids==nodeY.tids ){
36.             nodeY.flag=DEL , nodeY.link=nodeX
37.         } else nodeY.protentialPreC.Add(nodeX)
38.     } else {
39.         nodeZ.items=nodeX.items∪nodeY.items
40.         nodeZ.tids=nodeX.tids∩nodeY.tids
41.         nodeX.children.Add(nodeZ)
42.         if (nodeX.tids⊃nodeY.tids ){
43.             nodeY.flag=DEL , nodeY.link=nodeZ
44.         } else nodeY.protentialPreC.Add(nodeZ)
45.     }
46. }
SubsumptionCheck(nodeX)
47. for each node∈nodeX.protentialPreC∪nodeX.preC2{
48.     while (node.flag==DEL ) node=node.link
49.     if(nodeX.items⊂node.items)
50.       nodeX.preC2.Add(nodeY)
51.     else nodeX.preC2.Add(n)
52.     // n∈node.preC1∪node.preC2
53.     //∧n.length is minimal∧nodeX.items⊂n.items
54. }
55. if ∃ node∈nodeX.preC2∧nodeX.sup==node.sup {
56.     nodeX.flag=DEL , nodeX.link=node
57.     for each child∈nodeX.children
58.         child.flag=DEL,child.link=nodeX
59.     return
60. }
```

图 4 Q-CFIsL 算法

4.2 一个例子

下面以示例数据集为例，令最小支持度为 2，说明 Q-

CFIsL 的工作流程。图 5-1 为线型说明。

(1)算法初始化。生成根节点及其子节点,如图 5-2 所示。

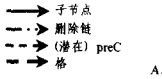

图 5-1 线型说明

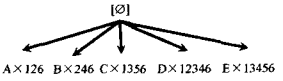

图 5-2 初始化算法

(2)处理 $A\times126$。得其子节点及其右兄弟节点的潜在 *preC*,如图 5-3 所示。

(3)处理 $ABD\times26$。分别与 $ACD\times16$,$ADE\times16$ 进行交并运算但未生成满足最小支持度的子节点。又因为 *preC*($ABD\times26$)为空,故无需对其建格。

(4)处理 $ACD\times16$。因为 $t(ACD)=t(ADE)$,所以将 $ACD\times16$ 替换成 $ACDE\times16$,将 $ADE\times16$ 删除且建立删除链指向 $ACDE\times16$。因为 *preC*($ABD\times26$)为空,故无需对其建格,如图 5-4 所示。

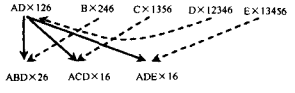

图 5-3 处理 $A\times126$

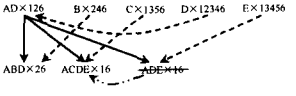

图 5-4 处理 $ACD\times26$

(5)程序返回到 $AD\times126$,对其建格。通过其子节点得到两个 *preC*:$ACDE\times16$ 和 $ABD\times26$,分别与其建格,如图 5-5 所示。

(6)处理 $BD\times246$。得其子节点 $BDE\times46$,如图 5-6 所示。

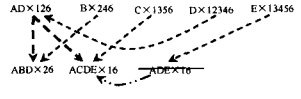

图 5-5 对 $AD\times126$ 建格

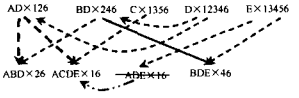

图 5-6 处理 $BD\times246$

(7)处理 $BDE\times46$。

(8)程序返回到 $BD\times246$,对其建格。$BD\times246$ 一共有两个 *preC*:$ABD\times26$ 和 $BDE\times46$。$BD\times246$ 分别与它们建格,如图 5-7 所示。

(9)处理 $C\times1356$ 分枝,见图 5-8 所示。

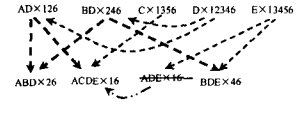

图 5-7 处理 $BD\times46$

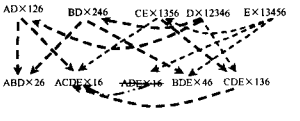

图 5-8 处理 $C\times1356$ 分枝

(10)处理 $D\times12346$ 分枝,如图 5-9 所示。

(11)处理 $E\times13456$ 分枝,如图 5-10 所示。

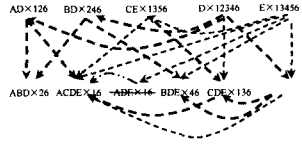

图 5-9 处理 $D\times12346$ 分枝

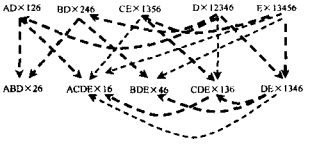

图 5-10 处理 $E\times13456$ 分枝

4.3 算法优化

为了提高使用 IT_Tree 挖掘频繁闭项集的性能,同 CHARM 一样,Q-CFIsL 使用了子节点动态排序以及 diffset 等优化策略。

所谓子节点动态排序,就是在第 9 行代码后加入 *sort_children*(*nodeX*),将新生成的子节点按照其支持度的大小从小到大重新排序。这样能减少 CHARM properties(3)、(4)的发生概率,从而减少分枝数,加快频繁闭项集的挖掘速度。

所谓 diffset 就是用子节点标识集与父节点标识集的差集来表示子节点的标识集。在 IT_Tree 中,每个节点代表了一个项集及其对应的标识集。然而这些标识集往往较大,要存储它们需要大量的内存。而且在进行交运算时,标识集越大运算量就越大。通过 diffset 能极大地减少标识集的大小,从而减少了内存的使用量和交运算的运算量,提高了算法的性能。关于 diffset 更详细的介绍,参见文献[10]。

5 实验及其结果分析

本文的实验是在一台 CPU 主频为 1.8GHz,内存为 1GB,操作系统为 Windows XP 的计算机上进行的。Q-CFIsL 用 C++实现。本文只选择与 CHARM-L 算法做比较,因为它是目前唯一挖掘频繁闭项集并同时建格的算法。CHARM-L 的原代码由算法的作者 Zaki 提供,它是一个 Linux 平台下的 C++程序。为了能让其在 Windows XP 下运行,使用了 cygwin.dll。本文的实验数据集 chess,mushroom,connect,pumsb 均来源于 http://www.cs.rpi.edu/zaki/software/。

实验表明,*preC*(X)的平均大小远小于 $s(X)$,并且随着最小支持度的减小,*preC*(X)的平均大小只有细微的变化,而 $s(X)$的平均大小将急剧增大(如图 6 所示第一列)。因此,使用 *preC* 进行包容检测以及建格效率更高。特别地,对于 chess,最小支持度为 45%时,Q-CFIsL 比 CHARM-L 约快 7 倍;对于 connect,最小支持度为 45%时,Q-CFIsL 比 CHARM-L 约快 8 倍;对于 mushroom,最小支持度为 0.1%时,Q-CFIsL 比 CHARM-L 约快 6 倍;对于 pumsb,最小支持度为 65%,Q-CFIsL 比 CHARM-L 约快 10 倍(如图 6 所示第

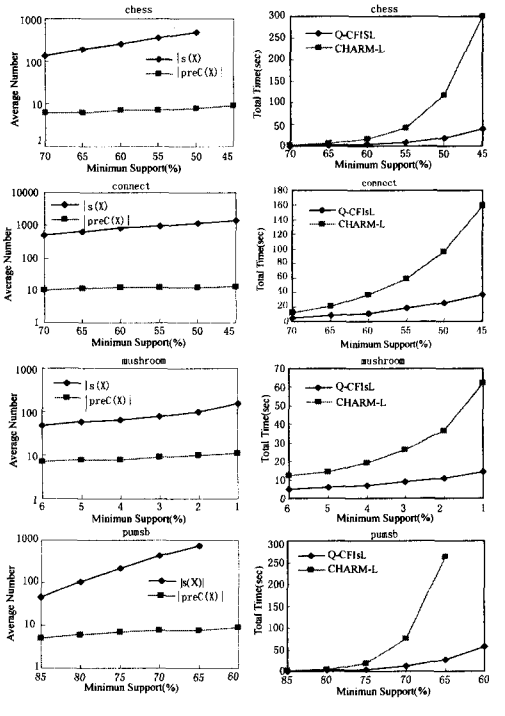

图 6 Q-CFIsL 性能分析

(下转第 193 页)

结束语 区域相关的匹配方法可以得到完整的视差图，但是在左右图像之间噪声、变形等情况下，匹配效果差[1]。本文提出了一种基于邻域差值变换的非参数立体匹配方法，这种方法是一种非参数的立体匹配方法，运算速度快，抗噪声、变形等能力强。同时和 rank 和 census 这两种非参数立体匹配方法比较，因为本文变换是同一扫描线上两个间隔一定距离窗口所有对应的像素灰度值作比较产生变换矩阵，而不像 rank 和 census 是窗口内的点和窗口中心点灰度值作比较产生变换矩阵[7]，避免了变换对中心像素点灰度值过分依赖的关系，这样在噪声情况下，本文算法匹配精度比传统的非参数变换 rank 和 census 方法提高了许多，具有很高的实用价值。

参 考 文 献

[1] Banks J, Bennamout M, Corke P. Non-parametric techniques for fast and robust stereo matching. IEEE Speech and Image Technologies for Computing and Telecommunications, 1997

[2] Ahlvers U, Zoelzer U, Rechmeier S. FFT-based disparity estimation for stereo image coding[C]// Proceedings 2003 International Conference of Image Processing. Barcelona, Spain, 2003(1): 761-764

[3] Moreau G, Fuchs P, Doncescu A, et al. Dense stereo matching-method using a quarter of wavelet transform[C]// Proceedings 2002International Conference of Image Processing. New York, USA Sept. 2002(1): 261-264

[4] Adjouadi M, Candocia F. A stereo matching paradigm based on the Walsh transformation [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence (S0162-8828), 1994, 16(12): 1212-1218

[5] Pagliari C L, Dennis T J. Stereo disparity computation in the DCT domain using genetic algorithms[C]//Proceedings 1997 International Conference of Image Processing. Washington DC, USA, 1997(3): 256-259

[6] Dunn U, Corke P. Real-time stereopsis using FPGAs // Proc. Workshop Field-Programmable Logic Applicat. London, U. K, Sept. 1997: 400-409

[7] Scharstein D, Szeliski R. A Taxonomy and Evaluation of Den-seTwo-Frame Stereo Correspondence Algorithms [J]. IJCV (S0920-5691), 2002, 47(1-3): 7-42

[8] Banks J, Bennamoun M. A constraint to improve the reliability of stereo matching using the rank transform// IEEE International Conference. Vol. 06, 1999

[9] Philippe L, John M. Robustness to noise of stereo matching[C] //Proceedings 12th International Conference of ImageAnalysis and Processing. Mantova, Italy, 2003: 606-611

[10] Banks J, Bennamout M. Reliablity Analysis of the Rank Transform for Stereo Matching. IEEE Transactions on Systems, Man, and Cybernetics-Prat B: Cybernetics, 2001, 31(6)

(上接第 178 页)

二列)。而且随着最小支持度的减少，Q-CFIsL 的性能优势越来越明显。

结束语 本文提出了挖掘频繁闭项集并同时建格的快速算法 Q-CFIsL。它引入了 preC 的概念，加快了挖掘频繁闭项集的速度，提高了建格的效率。实验表明，Q-CFIsL 在性能上优于 CHARM-L，而且最小支持度越小，Q-CFIsL 的优势就越明显。

参 考 文 献

[1] Agrawal R, srikant R. Fast Algorithms for Mining Association Rules in Large Databases// Proc. of 1994 International Conf. on Very Large Databases. 1994: 487-499

[2] Lin D-I, Kedem Z M. Pincer-search: A New Algorithm for Discovering the Maximum Frequent set. Lecture Notes in Computer Science, 1998, 1377: 103-119

[3] Pasquier N, Bastide Y, Taouil R, et al. Discovering frequent closed itemsets for association rules. Lecture Notes in Computer Science, 1998, 1540: 398-416

[4] Zaki M J, Hsiao C-J. CHARM: An Efficient Algorithm for Closed Association Rules Mining. Technical Report 99-10. Computer Science Department of Rensselaer Polytechnic Institute, 1999

[5] Pei J, Han J, Mao R. CLOsET: An efficient algorithm for mining frequent closed itemsets// Proc. sIGMOD Int'l Workshop Data Mining and Knowledge Discovery. 2000: 21-30

[6] Zaki M J. Generating Non-redundant Association Rules// Proc. of the 6th ACM sIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2000: 34-43

[7] Zaki M J. scalable Algorithms for Association Mining. IEEE Transactions on Knowledge and Data Engineering, 2000, 12(3): 372-390

[8] Burdick D, Calimlim M, Gehrke J. MAFIA: a Maximal Frequent Itemset Algorithm for Transactional Databases // Proc. of the 17th Int'l Conf. on Data Engineering. 2001: 443-452

[9] Wang J, Han J, Pei J. CLOsET+: Searching for the Best strategies for Mining Frequent Closed Itemsets // Conf. on KDD. 2003: 236-245

[10] Zaki M J, Gouda K. Fast Vertical Mining Using Diffsets// Proc. of the 9th ACM sIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2003: 326-335

[11] Zaki M J, Hsiao C-J. Efficient Algorithms for Mining Closed Itemsets and Their Lattice structure. IEEE Transactions on Knowledge and Data Engineering, 2005, 17(4): 462-478

[12] Angiulli F, Ianni G, Palopoli L. On the Complexity of Mining Association Rules. Technical Report IsI-CNR n10. Venezia, Italy, 2001

[13] Uno T, Kiyomi M, Arimura H. LCM ver. 3: Collaboration of Array, Bitmap and Prefix Tree for Frequent Itemset Mining // Proc. of the 1st Int'l Workshop on Open Source Data Mining: frequent pattern mining implementations. 2005: 77-86

[14] Lucchese C, Orlando s, Perego R. Fast and Memory Efficient Mining of Frequent Closed Itemsets. IEEE Transactions on Knowledge and Data Engineering, 2006, 18(1): 21-35

[15] Grahne G, Zhu J. Efficiently using prefix-trees in mining frequent itemsets // Proc. of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations. 2003

[16] Yahia S B, Hamrouni T, Nguifo E M. Frequent closed itemset based algorithms: A thorough structural and analytical survey. ACM sIGKDD Explorations Newsletter, 2006, 8(1): 93-104