一种基于粗糙集理论的快速并行属性约简算法

肖大伟[1] 王国胤[1,2] 胡 峰[1,2]
(重庆邮电大学计算机科学与技术研究所 重庆 400065)[1]
(西南交通大学信息科学与技术学院 成都 610031)[2]

摘 要 将并行计算的思想融入基于粗糙集理论的快速属性约简中,提出了一种基于粗糙集理论的快速并行属性约简算法。该算法在保证约简结果是Pawlak约简的情况下,将属性约简任务划分到多个处理器中同时处理,从而大大提高了属性约简的效率。仿真实验结果说明了该算法的高效性。

关键词 粗糙集,并行计算,属性约简,划分

Fast Parallel Attribute Reduction Algorithm Based on Rough Set Theory

XIAO Da-wei[1] WANG Guo-yin[1,2] HU Feng[1,2]
(Institute of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing 400065, China)[1]
(School of Information Science and Tecnology, Southwest Jiaotong University, Chengdu 610031, China)[2]

Abstract The idea of parallel computing was integrated into quick attribute reduction, and then a fast parallel attribute reduction algorithm based on rough set theory was proposed. The algorithm assigns the attribute reduction assignment to multi-processors to process simultaneously on the premise that its result is Pawlak reduction, which improves the reduction efficiency accordingly. Simulation experiment results prove the efficiency of the algorithm.

Keywords Rough set, Parallel computing, Attribute reduction, Partition

1 引言

粗糙集(Rough Set,RS)理论[1]由波兰学者Z. Pawlak教授于1982年提出,由于它能有效地处理和分析不精确、不一致、不完整等各种不完备信息,并能从中揭示潜在的规律,近年来在机器学习、数据挖掘等多个领域得到了广泛应用[2]。

属性约简是粗糙集理论的一个重要研究内容,许多学者已经对属性约简算法进行了大量的研究[3-6]。文献[3]给出了较好的启发函数,其属性约简算法的时间复杂度为$O(|C|^2\times|U|^2)$。文献[4]给出了基于条件信息熵的约简算法,其时间复杂度为$O(|C|^2\times|U|^2)$。文献[5]通过快速排序的方法计算正区域,将基于正区域的属性约简算法的时间复杂度降为$O(|C|^2\times|U|\times\log|U|)$。文献[6]采用了基数排序的方法计算正区域,得到了时间复杂度为$\max\{O(|C|\times|U|),O(|C|^2\times|U/C|)\}$的属性约简算法。然而不考虑决策表本身所占空间的情况下,这些算法的空间复杂度至少为$O(|C|\times|U|)$,有的甚至达到$O(|C|\times|U|^2)$。因此,这些算法对于海量数据的处理能力还不足。

随着数据挖掘中数据量的高速增长以及大规模并行计算机在数据挖掘中的应用,并行数据挖掘这一结合并行计算和数据挖掘的技术在社会各个方面得到了广泛的应用[7]。把并行计算应用于基于粗糙集的数据挖掘当中也是一个值得研究的方向,但目前还没有一种高效的并行属性约简算法。文献[8]采用了分治的思想计算属性核,文献[9]提出了基于分治法的属性序下的快速属性约简算法。本文采用文献[8,9]的分治思想结合并行计算对属性约简进行任务划分,把约简任务划分到多个处理器中同时处理,从而大大提高了属性约简算法的效率。该算法的时间复杂度为$O(1/K\times|C|^3\times|U|)$,其中$K$为参与运算的处理器数目,当$K=|C|$时,该算法的时间复杂度仅为$O(|C|^2\times|U|)$。本算法的空间复杂为$O(|U|+P\times|C|)$,其中$P=\max\{|V_i|1\leqslant i\leqslant|C|\}$,$|V_i|$表示第$i$个属性的不同属性值的个数,对于实际的决策表而言,$P$大部分情况下要远小于$|U|$。仿真实验结果说明了该算法的高效性,适合于处理大数据集。

2 相关概念

为了叙述方便,这里首先将粗糙集的基本概念作简单介绍。

定义1(决策表[2]) 一个决策表$S=\langle U,A=C\cup D,V,f\rangle$,其中$U$是对象的集合,也称为论域,$A=C\cup D$是属性集合,$C$和$D$分别称为条件属性和决策属性,$D\neq\emptyset$,$V$是属性值的集合,$f:U\times A->V$是一个信息函数,它指定了$U$中每个对象$x$的属性值。

定义2(不分明关系[2]) 给定决策表$S=\langle U,A=C\cup D,$

到稿日期:2008-05-30 本文受国家自然科学基金(No. 60573068,No. 60773113),新世纪优秀人才支持计划(NCET),重庆市自然科学基金(No. 2005BA2003),重庆市教委科学技术研究项目基金(KJ060517)资助。

肖大伟 硕士研究生,主要研究方向为基于粗糙集理论的并行数据挖掘,E-mail:xdwfree@126.com;王国胤 博士,教授,博士生导师,CCF会员,主要研究领域为模式识别、粗糙集、神经网络、粒计算;胡 峰 博士生,讲师,主要研究领域为粗糙集理论。

$V,f\rangle$，对于每个属性子集 $B\subseteq A$，定义了一个不分明关系 $IND(B)$，即 $IDN(B)=\{(x,y)|(x,y)\in U\times U,\forall_{b\in B}(b(x)=b(y))\}$。显然，不分明关系是一种等价关系。

定义 3(上、下近似集[2]) 给定决策表 $S=\langle U,A=C\cup D,V,f\rangle$，对于每个子集 $X\subseteq U$ 和不分明关系 B，X 的上、下近似集定义为：

$B^-(X)=\bigcup\{Y_i|Y_i\in U/IND(B)\wedge Y_i\cap X\neq\emptyset\}$

$B_-(X)=\bigcup\{Y_i|Y_i\in U/IND(B)\wedge Y_i\subseteq X\}$。

定义 4(相对正区域[2]) 设 U 为一个论域，P,Q 为定义在 U 上的两个等价关系簇，Q 的 P 正域记为 $POS_P(Q)$，并定义为：

$POS_P(Q)=\bigcup_{X\in IND(Q)}P_-(X)$。

定义 5(必要属性[2]) 设 U 为一个论域，P,Q 为定义在 U 上的两个等价关系簇，对于 P 中的任一属性 r，若 $POS_P(Q)=POS_{P-\{r\}}(Q)$，则称 r 为 P 中相对于 Q 不必要的，否则称 r 为 P 中相对于 Q 必要的。若 $\forall r\in P$ 都是 P 中相对于 Q 必要的，则称 P 为相对于 Q 独立的。

定义 6(相对约简[2]) 设 U 为一个论域，P,Q 为定义在 U 上的两个等价关系簇，若对于 P 的 Q 独立子集 $S\subset P$ 有 $POS_S(Q)=POS_P(Q)$，则称 S 为 P 的 Q 约简。

定义 7(属性核[2]) 决策表 $S=\langle U,A=C\cup D,V,f\rangle$ 的属性核定义为 $Core_D(C)=\bigcap_{R\in RED_D(C)}R$，其中，

$RED_D(C)=\{R|R$ 是 C 的 D 约简$\}$。

以下介绍本文中用到的几个并行计算的相关概念。

定义 8(散播[10]) 一种发送单个消息给所有进程的机制，该进程中的数据数组的每一个元素分别发送给各个进程。数组中的第 i 个单元被发送到第 i 个进程。

定义 9(归约[10]) 一种获得对象集合的机制，每个对象位于一个进程上，并将它们组合为位于一个进程之上的单个对象，或者组合它们，并使得组合值被留在每个进程之上。

定义 10(加速比系数[10]) 加速比系数 $S(n)$ 是衡量一个多处理机系统和一个单处理机性能的标准，定义为：$S(n)=$单处理机系统的执行时间/使用 n 个处理器的多处理机的执行时间。

3 基于粗糙集的并行属性约简算法

3.1 并行属性约简算法思想

根据相对约简的定义可以采用以下算法得到决策表的一个相对属性约简 R[11,12]。

算法 a 串行属性约简算法

输入：决策表 S=〈U,A=C∪D,V,f〉

输出：决策表 S 的一个相对属性约简 R

Stage1 属性扩张阶段

Step1 计算 $Core_D(C)$；

Step2 令 $R=Core_D(C)$，Attr_left= C/R；

Step3 $C_{select}=C_1(C_1\in$ Attr_left)；

For i=2 to |Attr_left| do

If $|POS_{R\cup\{C_i\}}(D)|>|POS_{R\cup\{C_{select}\}}(D)|$

then $C_{select}=C_i(C_i\in$ Attr_left)；

End For

Step4 $R=R\cup C_{select}$，Attr_left=Attr_left/C_{select}；

If $POS_R(D)=POS_C(D)$

then goto Stage2；

Else goto Stage1. Step3。

Stage2 属性收缩阶段

Step1 If $\exists C_{delete}\in R$ $POS_{R/\{C_{delete}\}}(D)=POS_C(D)$ then $R=R/\{C_{delete}\}$；

Step2 If $\forall r\in R POS_{R/\{r\}}(D)\neq POS_C(D)$

then Return R；

Else goto Stage2. Step1。

并行计算的核心是寻求并发性。Stage1 和 Stage2 中需要反复计算属性集的相对正区域，这是算法 a 效率低的主要原因，而对不同属性集的相对正区域的计算是相互独立的，因此，Stage1 和 Stage2 具有很好的并发性。对于 Stage1 考虑把属性集 Attr_left 划分到多个进程中，各进程同时对分配到的属性集执行 Stage1. Step3，主进程根据各进程的计算结果选择与 R 的并集的相对正区域最大的属性 C_{select} 添加到 R；对于 Stage2 可以采用同样的方法对 R 进行划分，同时在多个进程中检测冗余属性。

通过以上分析提出以下基于粗糙集理论的并行属性约简算法(以下称算法 b)。首先给出算法 b 的并行计算模型，如图 1、图 2 所示。

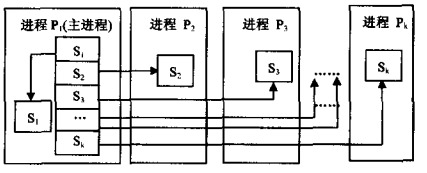

图 1 属性散播模型

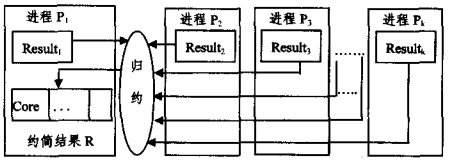

图 2 属性归约模型

图 1 和图 2 给出了算法 b 的并行计算模型，其中，属性散播模型中 S_i 表示属性集或某一属性，该模型对约简任务进行划分并平衡负载，属性归约模型中对各进程的运算结果进行归约并更新约简结果。算法 b 的具体描述如下：

设 K 为参与运算的处理器个数。其中，采用文献[8]的方法计算决策表的属性核和相对正区域。

算法 b 并行属性约简算法

输入：决策表 S=〈U,A=C∪D,V,f〉

输出：决策表 S 的一个相对属性约简 R

Stage1 属性扩张阶段

Step1 (串行处理阶段) 各进程计算决策表的属性核 $Core_D(C)$，令 R = $Core_D(C)$；主进程令 Attr_left= C/R；

Step2 (属性散播) 主进程令 Attr_left=$\bigcup_{i=1}^{k}S_i$，满足 $\forall S_i,S_j(i,j\in[1,k]\wedge i\neq j),||S_i|-|S_j||\leqslant 1\wedge S_i\cap S_j=\emptyset$。

将 S_1 分配给进程 P_1，S_2 分配给进程 P_2，…，S_k 分配给进程 P_k；

Step3 (各进程并行处理阶段) 设当前进程为 P_i，令 $C_{select_i}=C_1(C_1\in S_i)$；

For i=2 to $|S_i|$ do

If $|POS_{R\cup\{C_i\}}(D)|\geqslant|POS_{R\cup\{C_{select_i}\}}(D)|$ then $C_{select_i}=C_i(C_i\in S_i)$；End If

End For

Send(C_{select_i}, $|POS_{RU\{C_{select_i}\}}(D)|$, P_1);

Step4 (属性归约) 主进程 P_1 接收各进程计算结果,令 $T=\cup\{C_{select_i}|i\in[1,k]\}$, $C_{select}=C_{select_1}$ ($C_{select_1}\in T$);

For i=2 to |T| do

If $|POS_{RU\{C_{select_i}\}}(D)|\geqslant|POS_{RU\{C_{select}\}}(D)|$ then $C_{select}=C_{selcet_i}$ ($C_{select_i}\in T$); End If

End For

Step5 (更新约简结果) 各进程令 $R=R\cup C_{select}$, 主进程令 Attr_left = Attr_left/C_{select},

If $POS_R(D)=POS_C(D)$

then 各进程 goto Stage2;

Else 各进程 goto Stage1. Step2;

Stage2 属性收缩阶段

Step1 主进程令 Attr_text=$R/Core_D(C)$;

Step2 主进程令 low=1, high=K;

if K>|Attr_text| then high=|Attr_text|;

Step3 (属性散播) 主进程将属性 C_{low} 分配给进程 P_1, C_{low+1} 分配给进程 P_2, …, C_{high} 分配给进程 $P_{high-low+1}$, 其中 C_{low}, C_{low+1} … $C_{high}\in$ Attr_text。

Step4 (并行处理阶段) 设当前进程 P_i 在 Step3 中分配到的属性为 C_i, 令 $C_{delete_i}=-1$;

If $POS_{R/\{C_i\}}(D)=POS_C(D)$

then $C_{delete_i}=C_i$; End If

Send(C_{select_i}, P_1); //发送计算结果到 P_1

Step5 (属性归约) 主进程 P_1 令 $C_{delete}=C_{delete_1}$;

For i=2 to high−low+1 do

Receive(C_{delete_i}, P_i);

If $C_{delete_i}\neq-1\wedge C_{delete}=-1$

then $C_{delete}=C_{delete_i}$;

End For

Step6 (更新约简结果) If[1] $C_{delete}\neq-1$

各进程令 $R=R/\{C_{delete}\}$, 主进程令 Attr_text = Attr_text/$\{C_{delete}\}$,

各进程 goto Stage2. Step2; End If[1]

If[2] $C_{delete}=-1\wedge$ high=|Attr_text|

then Return R; End If[2]

If[3] $C_{delete}=-1\wedge$ high<|Attr_text|

then low=high+1, high=high+K;

if[3.1] high>|Attr_text| then high=|Attr_text|;

End if[3.1]

各进程 goto Stage2. Step3;

End If[3]

以下对 Stage1 和 Stage2 的任务划分和负载平衡方法进行举例说明。

设当前参与运算的进程数目 $K=3$, 在 Stage1 中需要处理的属性集 $Attr\_left=\{c_1,c_2,\cdots,c_7,c_8\}$, 按照 Stage1, Step2 的划分原则,进程 p_1 处理属性集 $S_1=\{c_1,c_2,c_3\}$, 进程 p_2 处理属性集 $S_2=\{c_4,c_5,c_6\}$, 进程 p_3 处理属性集 $S_2=\{c_7,c_8\}$。属性集 S_1,S_2,S_3 满足 $S_i\cap S_j=\varnothing\wedge||S_i|-|S_j||\leqslant1(i\neq j\wedge i,j\in\{1,2,3\})$。

设 Stage2 中当前参与运算的进程数目 $K=3$, 需要处理的属性集 $Attr\_text=\{c_1,c_2,\cdots,c_7,c_8\}$, 令指针 $low=1$, $high=3$ 分别指向要检测的第一个属性和最后一个属性。按照 Stage2, Step3 的划分原则进程 p_1 检测属性 c_1 是否为冗余属性,进程 p_2 检测属性 c_2 是否为冗余属性,进程 p_3 检测属性 c_3 是否为冗余属性。若本次运算中检测出冗余属性,则更新 R 和 $Attr\_text$, 令指针 $low=1$, $high=K$ 重新进行下一轮运算,否则,指针向前滑动 K 个距离,即令 $low=4$, $high=6$, 重新检测剩余的属性是否含有冗余属性。

由以上划分过程可以看出每个进程每次循环中处理的属性个数最大相差为 1,即任意两进程的任务量相当,这说明算法 b 具有很好的负载平衡性。

3.2 并行约简算法约简结果和时空复杂度分析

由算法 b 的终止条件可知,算法 b 的约简结果满足相对约简定义。对于 Stage1,若在一次循环中存在两个以上可以选择的属性,由于在不同的并行环境中主进程接收到这些属性的顺序不一样,主进程就会选择不同的属性加入 R。因此,对于这种情况约定算法 a 和算法 b 均选择位于决策表最左端的属性加入到 R。同理,若在一次循环中存在两个以上冗余属性,由于每次循环只能删除一个冗余属性,算法 a 每次循环删除的属性一定位于决策表的最左端,因此,约定算法 b 每次从多个冗余属性中选择位于决策表最左端的属性进行删除。以上约定保证了算法 a 和算法 b 的约简结果始终是相同的。

计算属性核的时间复杂度为 $O(|C|^2\times|U|)$[8],计算一次相对正区域的时间复杂度为 $O(|C|\times|U|)$[8]。算法 b 的时间复杂度 $T_p=T_{comp}+T_{comm}$, 其中 T_{comp} 为计算部分的时间, T_{comm} 为通信部分的时间。由算法 b 的约简过程易知,在有 K 个处理器参与运算的情况下,计算部分 T_{comp} 的时间复杂度为 $O(1/K\times|C|^3\times|U|)$。通信部分 T_{comm} 的时间复杂度为 $O(|C|)^2$。整个并行约简算法的时间复杂度为 $O(1/K\times|C|^3\times|U|)$。当 $K=|C|$ 时,算法 b 的时间复杂度仅为 $O(|C|^2\times|U|)$。

计算属性核和相对正区域的空间复杂度均为 $O(|U|+P\times|C|)$[8], 其中 $P=\max\{|V_i|\}$, $|V_i|$ 表示第 i 个属性的不同属性值的个数。所以,算法 b 的空间复杂度为 $O(|U|+P\times|C|)$。

4 仿真实验

实验分为三步,第一步测试算法 b 的约简效果;第二步测试两种算法的运行时间;第三步测试算法 b 的加速比系数。算法 a 和算法 b 均采用文献[8]中的算法计算属性核和相对正区域。最后两步中,每个数据集都测试 10 次,取平均值做为最终测试结果。

4.1 实验环境

在 windows 环境下采用 mpich. nt. 1. 2. 5 和 VC++开发工具在 5 台计算机上进行仿真实验。表 1 列出了实验的硬件环境。

表 1 仿真实验硬件环境

算法 a	windowsXp, CPU: p4 2.6, 内存 512.	
算法 b	windowsXp, CPU: p4 2.4, 内存 256.	3 台
	windowsXp, CPU: p4 2.6, 内存 512.	2 台

4.2 并行约简算法约简效果测试

从 UCI 数据库中选取 7 组数据集测试算法 b 的约简效果。对每一数据集随机抽取一半做为训练集,另一半做为测试集。其中,数据集 2 含有少量遗失属性值,采用取消补齐[2]的方法对其进行补齐,然后随机抽取 60%的数据作为原始数据集。采用基于属性重要性的离散化算法[2]对所有数据集

表 2　约简效果对比表

UCI 数据集	数据集大小	信息熵(N|P)		归纳约简(N|P)		算法 b(N|P)	
1. tic-tac-toe	958×9	8	82.4%	8	68.6%	8	78.4%
2. agaricus-lepiota	3386×22	12	62.1%	14	62.1%	12	62.1%
3. chess	3196×36	26	96.8%	31	96.9%	25	96.7%
4. zoo	101×16	6	90.1%	6	90.1%	4	94.1%
5. iris	150×4	2	90.6%	2	90.6%	2	90.6%
6. pima-indians-diabetes	768×8	5	68.4%	5	68.4%	5	68.4%
7. monks-problems	556×6	3	99.6%	6	93.1%	3	99.6%

中的连续属性进行离散化，与归纳属性约简算法[2]和基于信息熵的属性约简算法[2]的约简效果进行对比。本实验分为两步，第一步测试 3 种算法的属性约简结果，第二步在第一步的基础上采用归纳值约简算法[2]提取规则，然后采用少数优先策略测试每一规则集的正确识别率。表 2 列出了具体的测试结果。其中，N 表示约简结果中的条件属性个数，P 表示规则集的正确识别率，数据集大小用记录数目×条件属性数目表示。

仿真实验结果表明，数据集 5，6 的属性约简结果相同。对于数据集 1，2，3，4，7 而言，算法 b 的约简结果中的属性数目不大于另外两种算法，主要原因是算法 b 是根据当前的约简结果来计算属性重要性的，属于一种动态的贪心策略，而从表 2 可以看出，算法 b 的正确识别率和其它两种算法相当。

4.3　并行算法运行效率测试

从入侵检测数据集 KDDCUP99 中随机抽取 311029 条记录进行测试。每条记录含有 40 个条件属性和一个决策属性。然后从中随机选择 10%，20%，…，100%的记录生成新的数据集，分别用算法 a 和算法 b 对这 10 组数据集进行测试。表 3 和图 3 分别给出了两种算法的具体测试结果及对比曲线。

表 3　约简算法运行时间测试结果(单位：秒)

数据集	31102	62205	93308	124411	155514	186617	217720	248823	279926	311029
算法 a (T)	44.917	126.734	586.631	476.320	1096.923	1490.183	1987.058	2196.533	2372.541	2640.402
算法 b (T)	43.081	100.845	231.031	164.139	376.634	466.136	530.193	630.248	686.327	756.493

从图 3 可以看出，随机抽取的数据集的记录个数小于 62205 条时，算法 b 和算法 a 的运行时间相当，这主要是因为算法 b 的通信时间只与条件属性个数有关，当数据量比较小时，算法 b 的通信时间在整个约简时间中占的比例比较大。当数据集的记录个数足够大时两种算法的运行时间差别明显，当数据量达到最大时，由表 3 可知，算法 a 的运行时间大概需要四十多分钟，而算法 b 的运行时间只需要十分钟多一点。这充分说明了算法 b 的高效性。

4.4　加速比系数测试

随机生成 1×10^6 条记录，每条记录含有 15 个条件属性和一个决策属性，每条记录的条件属性和决策属性都在 0～9 上随机取值，然后选取 4.3 节实验中的最大数据集测试算法 b 的加速比系数。表 4 和图 4 分别给出了算法 b 的加速比系数随处理器数目变化的具体测试结果和曲线图。

表 4　加速比系数测试结果

处理器个数	自定义数据加速比	UCI 数据加速比
1	1	1
2	1.874	1.759
3	2.747	2.454
4	2.985	3.107
5	3.539	3.492

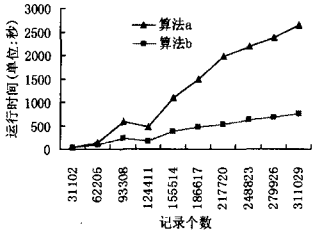

图 3　约简算法运行时间对比曲线

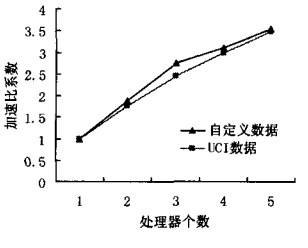

图 4　加速比系数随处理器数目变化曲线图

由图 4 可以看出，算法 b 的加速比系数与处理器数目几乎成线形关系变化。这充分说明了算法 b 具有很好的可扩展性和负载平衡性。

结束语　虽然粗糙集理论正日渐成熟，但实际应用还不够广泛。一个重要原因是：基于粗糙集理论的属性约简算法在大数据集下效率不高 。本文将并行计算的思想融入粗糙集的属性约简中，设计了时间复杂度为 $O(1/K\times|C|^3\times|U|)$，空间复杂度为 $O(|U|+P\times|C|)$($P=\max\{|V_i|\}$)的属性约简算法。该算法提高了属性约简的效率，适合于处理大数据集，并将推动粗糙集的实际应用 。基于粗糙集理论的并行离散化算法和并行值约简算法将是下一阶段的研究工作。

参 考 文 献

[1] Pawlak Z. Rough Set[J]. International Journal of Computer and Information Sciences, 1982, 11: 341-356

[2] 王国胤. Rough 集理论与知识获取[M]. 西安交通大学出版社, 2001

[3] Hu X H, Cercone N. Learning in Relational Database: A rough set approach [J]. Inter. J. of Computational Intelligence, 1995, 11(2): 323-338

[4] 王国胤，于洪，杨大春. 基于条件信息熵的决策表约简[J]. 计算机学报，2002，25(7)：759-766

[5] 刘少辉，盛球戬，吴斌，等. Rough 集理论高效算法的研究[J]. 计算机学报，2003，5(26)：524-529

[6] 徐章艳，刘作鹏，杨炳儒，等. 一个复杂度为 max{O(|C|×|U|，O(|C|²×|U/C|)}的快速属性约简算法 [J]. 计算机学报，2006，29(3)：391-399

[7] 刘华元，袁琴琴，王保保. 并行数据挖掘算法综述[J]. 电子科技，2006，01：65-68

[8] Hu Feng, Wang Guoyin, Xia Ying. Attribute Core Computation Based on Divide and Conquer Method[A]. RSEISP[C], 2007: 310-319

[9] 胡峰，王国胤. 属性序下的快速约简算法[J]. 计算机学报，2007，30(6)：1429-1435

[10] 李晓明. 数据并行计算：概念、模型与系统[J]. 计算机科学，2000，27(6)：1-5

[11] 苗夺谦，胡桂荣. 知识约简的一种启发式算法[J]. 计算机研究与发展，1999，36(6)：681-684

[12] 吴明芬，许勇，刘志明. 一种基于属性重要性的启发式约简算法[J]. 小型微型计算机系统，2007，28(8)：1452-1455