一种带局部回溯的属性约简算法*)

The Attribute Reduction Algorithm with Local Retrospect

侯丽珊　苗夺谦

(山西大学数学系　太原030006)

Abstract In this paper, the existing attribute reduction algorithms in rough sets are classified as members of TD-algorithms(Top Down); then a new reduction algorithm of BU(Bottom Up) is proposed; after the comparison to the ideas of the two groups of reduction algorithms, an effective attribute reduction is created through adding proper local retrospect to the traditional methods; at last an experiment is given to show the validity of this new technique.

Keywords Rough sets, Attribute reduction, SFS, SBS, Retrospect

1 引言

粗糙集理论自1982年由波兰科学家 Z. Pawlak 提出发展到现在，知识约简已经成为其研究的核心内容之一[5]。知识约简包括属性约简和属性值约简，属性约简是对整个知识库而言的，在保证信息量不减少的前提下，去掉冗余的属性；而属性值的约简是针对每条信息(决策)的描述而言的，在不影响对其分类或决策的能力的情况下，不考虑某些属性的取值。本文所提到的约简如果没有特别说明，特指属性的约简。一般来讲，约简是不唯一的，人们当然希望能够找到具有最少属性的约简，即最小约简。遗憾的是，求解最小约简是 NP 完全问题，任何非穷举的算法都不能保证所得结果是最优的，这时不得不放弃最优解而采取计算量小的次优约简算法。目前已有的知识约简算法有：X. H. Hu 算法、基于 Pawlak 属性重要性的约简算法、差别矩阵及其改进算法以及基于信息熵的约简算法，这些算法对 Pawlak 约简都是不完备的，只能找到包含约简的相对较小的属性子集。原因就是，一旦某个属性被选中，即使由于后面加入的属性使其变为冗余，也无法再把它剔除。为了避免上述缺点，本文提出一种带回溯的属性约简算法。在第1节中对粗糙集理论中传统的知识约简算法做了简要回顾，并将它们都纳于顺序前进法的体系之下，同时提出一种顺序后进知识约简算法；第2节在借鉴上一节思想的基础上，提出一种带回溯的属性约简新策略；第3节通过一个例子，对粗糙集理论中传统的约简算法和带回溯的约简算法进行了比较；最后一节给出了本文的结论及进一步研究的问题。

2 顺序前进法与顺序后退法

2.1 顺序前进法

这是最简单的自底向上的搜索方法，每次从未入选的特征属性集中选择一个属性，使得它与已入选的属性集组合在一起时的分类能力最强，直到分类能力与原表相等。"自底向上"法就是特征数从零逐步增加[1]。通过对粗糙集理论中现有的知识约简算法的分析，我们认为其都可以归入顺序前进法的范畴。

在决策表与信息系统上的约简算法基本类似，为叙述简洁，这里只以信息系统为例，对现有的属性约简算法进行总结：

Input 一个信息系统 $IS=\langle U,A,V,f\rangle$
Output $\{B\subseteq A \mid IND(B)=IND(A)\}$

1)求核 Co。
2)$B\Leftarrow Co$。
3)对$\forall\, c\in A-B$，计算，如果 $Sig(c,B)$，如果 $Sig(c',B)=\max\limits_{c\in C-B}\{Sig(c,B)\}$，
$B\Leftarrow B\cup\{c'\}$。
4)判断 $IND(B)=IND(A)$是否成立。
如果成立，则输出 B，结束；否则转3)。

现有的属性约简算法基本上都是从核出发，然后从余集中选择最重要的属性逐个添加，直到分类能力(信息量)与原信息系统相等。都是采用 SFS 的思路，所不同的只是在属性重要性函数的定义上，各种算法结合了自身的特点。

X. H. Hu 算法中：

$$Sig(c,B)=\frac{card(IND(B\cup\{c\}))-card(IND(B))}{card(IND(A))}$$

基于 Pawlak 属性重要性[4]的算法中：

$$Sig(c,B)=\frac{card(IND(B\cup\{c\}))-card(IND(B))}{cardU}$$

事实上，以上两个算法是完全等价的。因为在特征属性的选择中，函数的具体取值并不重要，我们所关心的只是由其决定的属性间的序关系。不难证明，它们所决定的序关系完全一致。

基于差别矩阵的算法中：

$Sig(c,B)=c$ 在差别矩阵 $M(IS)$中出现的频数。

改进的差别矩阵算法中：

$Sig(c,B)=c$ 在 M_i 中出现的次数，将差别矩阵 $M(IS)$中的元素按照其所含属性个数进行分类，$M_1,M_2,\cdots,M_m,M_i$ 为当前非空 $M_j(j=1,\cdots,m)$中下标最小的一个。

基于信息熵[3]的算法中：

$Sig(c,B)=H(B\cup\{c\})-H(B)=H(c\mid B)$，其中 $H(\cdot)$为熵函数。

这些算法对 Pawlak 约简都是不完备的，在很多情况下找到的只是包含约简的相对较小的属性子集。其原因就是，它们只考虑到所选属性与已入选属性之间的相关性，而未考虑未入选属性集之间的统计相关性，导致一旦某个属性被选中，即使由于后面加入的属性使其变为冗余，也无法再把它剔除。

克服这个缺陷，有一个办法就是约简过程中，每次不止增加一个属性，而是增加 r 个属性，其实这就是所谓的广义顺序前进法(Generalized Sequential Forward Selection, GSFS)。即每次从未入选的属性中选择出 r 个属性，使得加入这 r 个属性后分类能力相对最大。

*)本文得到国家自然科学基金(No. 60175016)、山西省青年科技基金(No. 20001001)和山西省留学归国基金资助。侯丽珊　硕士生，主要研究领域为粗糙集理论、人工智能。苗夺谦　博士，教授，主要研究领域为粗糙集理论、模式识别和人工智能。

GSFS 法可以克服 SFS 法不考虑未入选属性之间的统计相关性的缺点，有了一定的改进，比 SFS 法更可靠，但是每步都要计算 $C_{|C|-k}^{1}$ 个候补属性集的重要性，因而计算量变大了，此外它也无法剔除已入选的属性。

2.2 顺序后退法

与顺序前进法相反，这是一种自顶向下的方法。从整个属性集开始，每次剔除一个对划分没有贡献的属性，当然所剔除的属性必须使仍然保留的属性子集的分类能力不降低[1]。“自顶向下”法就是属性数从 $|A|$ 开始逐步减少，其中 A 为原始属性集[2]。SBS 的粗糙集理论属性约简算法可以描述为：

Input 一个信息系统 $IS=\langle U,A,V,f\rangle$
Output $\{B\subseteq A \mid IND(B)=IND(A)\}$
1)计算 $IND(A)$。
2)$B\Leftarrow A, C\Leftarrow A$。
3)对$\forall c\in C$，计算 $Sig(c,B)$。
$Sig(c',B)=\min_{c\in C-B}\{Sig(c,B)\}$。找出最不重要的属性(即最平凡的属性)$c'$。
4)判断 $IND(B\backslash\{c'\})=IND(A)$是否成立。如果成立，则 $B\Leftarrow B\backslash\{c'\}$，$C\Leftarrow C\backslash\{c'\}$；否则 $C\Leftarrow C\backslash\{c'\}$。
如果 $C=\Phi$，结束；否则转3)。

当然，函数 $Sig(c,B)$的定义视采用的具体策略的不同而不同，或者也可以沿用 X. H. Hu 算法、差别矩阵以及信息熵算法中的定义，只不过取其最小值即可。

由于顺序后退法的计算是在高维空间进行的，因此计算量比顺序前进法要大，是人们不愿意接受的。SBS 似乎并不是一种很好的算法，但是它却给了我们一个很有意义的启迪，能不能在传统的粗糙集理论的约简算法上加入回溯过程呢？

3 带局部回溯的约简算法

为避免传统算法中，属性一旦被选入(或剔除)就不能再剔除(或选入)的缺点，我们考虑在属性选择过程中加入局部回溯的策略。比如在第 k 步可先用 SFS 法逐个添加属性到 $k+l$ 个，然后再用 SBS 法剔去 $r(r<l)$个相对不重要的属性。

以基于差别矩阵的约简为例，算法描述如下：

Input 一个信息系统 $IS=\langle U,A,V,f\rangle$
Output $\{B\subseteq A \mid IND(B)=IND(A)\}$
1)构造差别矩阵 $M(IS)$。
2)求核 C_0。
3)$B\Leftarrow C_0$。
4)$M(IS)\Leftarrow M(IS)-\{c_{ij}\mid c_{ij}\cap B\neq\Phi\}$，$TM(IS)\Leftarrow M(IS)$
5)判断 $M(IS)=\Phi$?。
若真，输出 B，结束；否则
6)循环执行 l 次。
(1)$TM(IS)\Leftarrow TM(IS)-\{c_{ij}\mid c_{ij}\cap B\neq\Phi\}$
(2)判断 $TM(IS)=\Phi$?
若真，输出 B，结束。
(3)计算每个 $c\in A-B$ 的重要性。
$Sig(c,B)$=统计 c 在 $TM(IS)$中出现的次数。
$Sig(c',B)=\max_{c\in A-B}\{Sig(c,B)\}$，
$B\Leftarrow B-\{c'\}$。
7)循环执行 r 次。
计算每个 $c\in B$ 的重要性。
$Insig(c,B)=M(IS)$中与 $B-\{c\}$相交不空的差别元素的个数。
$Insig(c',B)=\max_{c\in B}\{Insig(c,B)\}$，$B\Leftarrow B-\{c'\}$。
8) 转4)。

注：上面的算法中，$TM(IS)$只是一个临时矩阵，$Sig()$是属性重要性的函数，$Insig()$是属性平凡性(即不重要性)的度量函数。

4 与传统约简算法的比较

本节将通过一个信息系统，如表1所示，对上节提出的带局部回溯策略的属性约简算法和传统的顺序前进的约简算法进行比较。

例 表1所示的信息系统中，共含18个对象，7个属性，分别为 a，b，c，d，e，f，g。

表1 一个信息表

U	a	b	c	d	e	f	g
1	1	1	1	1	1	1	1
2	0	0	1	1	1	1	1
3	2	2	2	2	2	2	2
4	3	2	3	2	2	2	2
5	4	4	4	4	4	4	4
6	5	4	4	5	4	4	4
7	6	6	6	6	6	6	6
8	6	7	6	6	7	6	6
9	8	8	8	8	8	8	8
10	8	8	9	8	8	9	8
11	10	10	10	10	10	10	10
12	10	10	10	11	10	10	11
13	12	12	12	12	12	12	12
14	13	13	12	13	12	12	12
15	14	14	14	14	14	14	14
16	14	15	14	14	14	15	14
17	16	16	16	16	16	16	16
18	16	16	17	16	16	16	17

首先用传统的约简算法对该信息系统进行约简。

差别矩阵方法：该信息表对应的差别矩阵

$M(IS)=\{ab, ac, ad, be, cf, dg, abd, bf, cg, abcdefg(144)\}$

144表示该差别元素在矩阵中出现的频数，缺省时属性出现次数为1。

由于差别元素 $abcdefg$ 的存在与否对每个属性的影响均相同，不会影响到属性重要性的排序，因此我们不妨只考虑

$M(IS)'=\{ab,ac,ad,be,cf,dg,abd,bf,cg\}$

显然核为空。

计算每个属性在差别矩阵中出现的频数：$P(a)=4$，$P(b)=4$，$P(c)=3$，$P(d)=3$，$P(e)=1$，$P(f)=2$，$P(g)=2$。属性 a 和 b 的频数相等，如果选 a，$B=\{a\}$，$M(IS)'=\{be,cf,dg,bf,cg\}$。

此时，$P(b)=2$，$P(c)=2$，$P(d)=1$，$P(e)=1$，$P(f)=2$，$P(g)=2$。如果选 b，$B=\{a,b\}$，$M(IS)'=\{cf,dg,cg\}$。

此时，$P(c)=2$，$P(d)=1$，$P(f)=1$，$P(g)=2$。选 c，$B=\{a,b,c\}$，$M(IS)'=\{dg\}$。

最后如果选为 d，$B=\{a,b,c,d\}$为约简。

事实上，$\{a,b,c,d\}$并不是 pawlak 意义下的约简，不具备独立条件，属性 a 是冗余的；而仅$\{b,c,d\}$就可以构成一个约简。

在属性选择过程中，有许多时候是有多个属性可供选择的，按照习惯常识我们取第一个。但如果选择了其它属性，结果又会是什么样呢？考虑所有的可能性，可以得到下面的约简结果。

表2

约简结果	出现概率
$\{b,c,d\}$	2/16
$\{a,b,c,d\}$	3/16
$\{a,b,c,g\}$	7/16
$\{a,b,f,g\}$	3/16
$\{b,c,d,g\}$	1/16

在25%的情况下得到的约简不独立。按照通常的习惯，用

(下转第140页)

事务进行哈希树中候补项集的出现频度计算时，亦先将事务中的相关项替换为新编号后再在哈希树中进行出现频度计算处理。在进行替换编号操作时，事务中没有出现在上述映射表中的项被忽略，因为它们对候补项集的出现频度计算是无用的。最后，在出现频度计算完成后，由哈希树输出 F_k 时，再用映射表将编号映射回原来的项。

由推理1可知，用上述动态建树方法构建哈希树时，可以保证不会产生溢出现象。这一方法大大节约了系统资源的使用，因为哈希表几乎没有被闲置的桶。例如，在上面所举例子中，如果 I 中所包含的项也为1000的话，那么同静态建树法相比，在每形成一个内部接点时，其哈希表所使用的空间将由静态法的1000个单元空间变为动态法的5个单元空间，仅为5‰。

3.3 两种建树法的比较

静态建树法与动态建树法各有其优势与不足。它们的共同特征是：

1. 哈希表大小和哈希函数的决定方法简单且实用。不需要针对不同的挖掘处理进行测试与调整。

2. 叶节点所允许保存的项集数量的最小值都可设定为一个，使得对哈希树的查找处理快速而有效。

3. 针对任何情形下的频繁项集查找处理，都能保证哈希树的构建一次成功，避免了挖掘处理因哈希树的构建失败而失败的出现可能。

除此之外，它们分别还拥有其各自的不同特征，从而导致它们适用于不同的情形的挖掘处理。

对于静态建树法，其哈希表的大小由数据库中所包含的不同项的总数决定。建树及对树处理时可以直接利用各个项在 I 中的下标值，处理方便迅速。其缺点是树中各个内部节点所用哈希表中太多的桶被闲置，造成大量系统资源的浪费。因此，静态建树法适用于数据库较小、库中所含总的不同项的数量较少且系统资源充足，保证有足够的内存供挖掘处理所使用的频繁项集挖掘情形。这是一个以空间换时间的建树方法。

而与静态建树法相对应，动态建树法是根据挖掘处理过程中实际需要处理的项动态地调整其哈希表的大小。这使得哈希树构建所占用的系统资源大大减小，各个内部节点所用哈希表中桶的利用率大大增加。但是它为此必须另外建立一个项与哈希表中所用代码之间的映射表，从而较静态建树法增加了额外的映射处理开销。这是一个以时间换空间的建树方法。它适用于大数据库、库中不同项多且系统资源及可用内存空间有限的频繁项集挖掘情形。

结束语 针对目前广泛使用的类 Apriori 频繁项集挖掘算法中所用哈希树构建上存在的问题，本文提出了两个建树方法--静态建树法和动态建树法，来确保挖掘过程中哈希树的构建简洁、有效并且一次成功。本文详细叙述了静态和动态建树方法，并证明了所提方法的正确性。在分析其各自的特征、优势与不足的基础上，给出了它们各自适用的挖掘处理情形。

参 考 文 献

1 Agrawal R, Srikant R. Fast Algorithms for Mining Association Rules. In VLDB'94. 487～499

2 Du X P, Kaneko K, Makinouchi A. Fast Algorithm to Find Frequent Itemsets for Mining of Association Rules. In IS'00. 408～414

3 Han J, Pei J, Yin Y. Mining Frequent Patterns without Candidate Generation. In: Proc. 2000 ACM-SIGMOD Int. Conf. on Management of Data, Dallas, TX, May 2000. 1～12

4 Agrawal R, Srikant R. Mining Sequential Patterns. In: ICDE'95. 3～14

5 Silverstein C, Brin S, Motwani R, Ullman J. Scalable Techniques for Mining Causal Structures. In: VLDB'98. 594～605

6 Bayardo R J. Efficiently Mining Long Patterns from Databases. In: SIGMOD'98. 85～93

7 Kamber M, Han J, Chiang J Y. Metarule-guided Mining of Multi-dimensional Association Rules Using Data Cubes. In: KDD'97. 207～210

8 Park J S, Chen M S, Yu P S. An Effective Hash-based Algorithm for Mining Association Rules. In: Proc. ACM SIGMOD Int. Conf. Management of Data, San Jose, CA, May 1995. 175～186

9 Du X P, Makinouchi A. Ameliorated Algorithm to Maintain Discovered Frequent Itemsets. Res. Rep. ISEE Kyushu University, 2001, 6(1): 19～24

（上接第128页）

X. H. Hu 的约简算法得到的结果同样是{a,b,c,d}；而基于信息熵的算法的约简结果为{a,b,c,g}，可以验证是独立约简，但显然不是最小约简（{b,c,d}才是最小约简）。

接下来利用本文给出的新算法计算约简。

l 和 r 的选取是本算法的关键。l 与 r 差距太大则体现不出该方法的特点，太小了则必须以巨大的计算量和存储量为代价；属性的个数以及核属性的个数都影响着 l 和 r 的取值。本例中我们取 $l=3, r=1$。仍考虑 $M(IS)'=\{ab,ac,ad,be,cf,dg,abd,bf,cg\}$ 显然核为空。

先用 SFS 法添加3个属性，则

$B=\{a,b,c\}$。

计算 a,b,c 的重要性。

$Insig(a,B)=card(\{ab,ac,be,cf,abd,bf,cg\})=7$

$Insig(b,B)=card(\{ab,ac,ad,cf,abd,cg\})=6$

$Insig(c,B)=card(\{ab,ac,ad,be,abd,bf\})=6$

相对来说，a 是最不重要的一个，所以 $B=\{b,c\}$，$M(IS)'=\{ad,dg\}=TM(IS)$。

$M(IS)'$ 中还剩3个属性，用 SFS 法计算它们出现的频数，$P(a)=1, P(d)=2, P(g)=2$。所以选 d，从而 $B\{b,c,d\}$，此时 $TM(IS)$ 为空。显然 B 就是约简，并且是最小约简。

在表1中，单个属性来看，a 是相当重要的，所有的算法都首先选择。随着属性的增加，后入选的属性的组合对于分类必不可少，同时可以取代 a 的作用，使其由最重要变为冗余。传统的算法对此无能为力，而本文提出的带局部回溯的约简策略则为解决这个问题提供了一种新思路。

结束语 本文提出了一种带局部回溯策略的属性约简新算法，克服了约简过程中属性一旦选入，即使变得冗余，也无法剔除出去的弊端，从而在更多的情况下可以找到最小约简。

回溯过程中 l 和 r 的选择还有待进一步研究，既要考虑未入选属性间的相关性，又不至于引起计算量过大，需要对二者合理兼顾；如果能在约简过程中动态地选择、调整，结果应该更为理想。这些都有待进一步的研究。

参 考 文 献

1 边肇祺，张学工，等. 模式识别. 清华大学出版社，2000. 202～204

2 于洪，杨大春，王国胤，等. 基于 Rough Set 理论的知识约简算法. 计算机科学，2001，28(5. 专刊)：31～34

3 苗夺谦，王珏. 粗糙集理论中知识粗糙性与信息熵关系的讨论. 模式识别与人工智能，1998，11(1)：34～40

4 Pawlak Z. Rough Set-Theoretical Aspects of Reasoning about Data. Kluwer Academic Publishers, Dorderecht, Boston, London, 1991

5 Polkowski L, Skowron A. Rough Sets: Perspectives, Rough Sets in Knowledge Discovery. In: Polkowski L, Skowron A, eds. Physica-Verlag, 1998. 1～27