文章编号: 1001-0920 (2011) 04-0495-06

一种高效的增量式属性约简算法

冯少荣, 张东站

(厦门大学 计算机科学系，福建 厦门 361005)

摘 要: 针对粗糙集中求属性核和属性约简存在的问题, 首先给出了改进的差别矩阵定义, 进而提出一种基于改进差别矩阵的核增量式更新算法, 用于解决对象动态增加情况下核的更新问题; 同时, 为了降低现有增量式属性约简算法的时间、空间复杂度, 提出一种不存储差别矩阵的高效属性约简算法, 用于处理对象动态增加情况下属性约简的更新问题. 理论分析及实验结果均表明了所提出算法的有效性和可行性.

关键词: 粗糙集；增量；属性约简；差别矩阵

中图分类号: TP311 **文献标识码:** A

Effective increment algorithm for attribute reduction

FENG Shao-rong, *ZHANG Dong-zhan*

(Computer Science Department，Xiamen University，Xiamen 361005，China．Correspondent：FENG Shao-rong, E-mail：shaorong@xmu.edu.cn)

Abstract: Aiming at some shortcomings of existing on computing attribute core and attribute reduction in rough sets, an improved discernibility matrix definition is introduced. By using this foundation, based on improved discernibility matrix, an incremental updating algorithm for computing core is proposed, which is mainly used to solve core updating when objects are dynamically increased and deleted. In order to decrease time and space complexity on the existence incremental attribute reduction algorithm, an effective algorithm for attribute reduction is proposed, which does not storage discernibility matrix. This algorithm is mainly used to process attribute reduction updating when objects are dynamically increased. Theoretical analysis and experimental results show the feasibility and effectiveness of the proposed algorithm.

Key words: rough set；increment；attribute reduction；discernibility matrix

1 引 言

知识约简是粗糙集理论[1]的核心内容之一. 知识约简分为属性约简和属性值约简, 即在保持信息系统分类或决策能力不变的前提下, 删除冗余属性和冗余属性值, 获得信息系统的分类或决策规则. 随着数据库系统中数据的不断增加, 属性约简相对于属性值约简更加有效, 它大大简化了数据库结构的复杂度, 提高了人们对隐含在数据库庞大数据量中各种信息的认识程度. 因此, 属性约简已成为目前粗糙集的研究热点之一.

决策表的属性约简通常是不唯一的, 人们希望能找到具有最少属性的约简, 即最佳约简. 然而, 找到一个最佳约简是一个NP-hard问题, 导致NP-hard的主要原因是属性的组合爆炸. 为解决这一问题, 通常采用启发式搜索方法求出最佳或次最佳约简[2-4]. 现有的属性约简大体上可分为3种: 基于差别矩阵或基于改进的差别矩阵的属性约简算法, 基于正区域的属性约简算法以及基于启发式的属性约简算法. 这些算法的共同特点是: 1) 利用属性的重要性作为启发式信息, 但并不完备[5-6], 不能保证一定能够得到约简[7]; 2) 这些算法大都是针对静态的信息系统或决策表, 不适合信息系统或决策表动态变化的情况. 目前, 有关属性约简的增量式算法并不多, 已有的动态增量属性约简算法其时间和空间复杂度较高. 文献 [8] 的增量式属性约简算法多次调用Core(P, U), 并且该算法存储了差别矩阵, 求核时需遍历差别矩阵, 其时间复杂度为$O(|U_1| \times (|U_1| + |U_2'|))$. 当决策表一致时, $U_1 = U$, 该属性约简算法的时间空间复杂度至少为$O(|U|^2)$. 因此文献 [8] 的方法具有较高的时间空间复杂度.

文献 [9] 的RedueBaseSig算法由于没有从求核

收稿日期: 2010-01-11；**修回日期:** 2010-06-15.

基金项目: 国家自然科学基金项目(50604012).

作者简介: 冯少荣(1964−), 男, 副教授, 博士, 从事并行分布数据库、数据挖掘等研究；张东站(1974−), 男, 副教授, 博士, 从事数据仓库、数据挖掘等研究.

出发, 在某些情况下, 获得的并不是属性的约简, 其中会包含冗余的属性. 其时间复杂度为 $\max\{O(|C|\times|U|\times\log|U|), O(|C|^2\times|U|)\}$.

文献 [10] 对 RedueBaseSig 进行了改进, 提出一种基于分布计数的基数排序方法的等价类划分算法. 对决策表采用分布计数的基数排序, 并按属性集 C 对决策表 S 排序, 处理逻辑较复杂. 该算法的时间复杂度为 $O(|C|^2\times|U|)$, 空间复杂度为 $O(|U|)$.

本文针对粗糙集中求属性核和属性约简问题, 在分析目前求核算法和属性约简算法[8]的基础上, 指出引起算法时间、空间复杂度高的原因, 提出了改进的增量式求属性核算法及高效的属性约简完备算法, 并进行了相关实验或实例验证. 理论分析表明, 本文算法是有效可行的; 计算示例表明该算法是正确的, 明显降低了时间、空间复杂度.

2 相关概念和理论

2.1 改进的差别矩阵定义

按照文献 [7] 中差别矩阵定义计算差别矩阵时, 当 $x_i\in U_1, x_j\in U_1$ 时, $m_{ij}=m_{ji}$. 文献 [7] 同时计算了 m_{ij} 和 m_{ji}, 而本文只需计算 m_{ij} 便能正确地求核, 所以对 m_{ji} 的计算是多余的. 当 $|U_1|$ 较大时, 这种多余的计算量很大, 特别当决策表一致时, 增加的不必要计算量是求核实际所需计算量的一倍. 因此, 当 $x_i\in U_1, x_j\in U_1$ 时, 通过限定 $i>j$, 只需计算 m_{ij}, 而不必计算与 m_{ij} 对称且相等的 m_{ji}. 下面给出改进的且与文献 [7] 等价的差别矩阵定义.

对于给定的信息系统 IS, 定义差别矩阵 $M=\{m_{ij}\}$, 其中

$$m_{ij}=\begin{cases}\{a\in C: f(x_i,a)\neq f(x_j,a)\},\\ \text{If } f(x_i,D)\neq f(x_j,D),\\ \quad x_i\in U_1,\ x_j\in U_1,\ i>j;\\ \{a\in C: f(x_i,a)\neq f(x_j,a)\},\\ \text{If } x_i\in U_1,\ x_j\in U_2';\\ \Phi,\ \text{others.}\end{cases}\tag{1}$$

这里 U_1, U_2, U_2' 同文献 [7].

2.2 改进的求核算法

针对动态增加的情况, 文献 [7] 中算法 2 具有较低的时间复杂度 $O(5\times|U_1|+3\times|U_2'|)$; 由于文献 [7] 计算核时使用了差别矩阵, 导致具有较高的空间复杂度 $O(|U_1|\times(|U_1|+|U_2'|))$. 特别当决策一致时, $U_1=U$, $U_2'=0$, 此时空间复杂度为 $O(|U|^2)$, 不能有效地处理大数据集. 为此, 本文给出文献 [7] 中算法 2 的一个改进算法, 它不需要存储差别矩阵, 具有较低的空间复杂度.

在改进的差别矩阵定义的基础上, 以及为改进文献 [7] 的算法 2 作准备, 首先给出新的求属性核的算法.

算法 1 求核算法

输入: U_1, U_2';

输出: DMSC(C).

```
begin
  for each x_i ∈ U_1 do
    for each x_j ∈ (U_1 ⋃ U_2') do
      m_ij = Φ; flage=0;
      if ((x_j ∈ U_1, f(x_i,D)! = f(x_j,D) and i > j)||x_j ∈ U_2')
        then
          for each a ∈ C do {
            flage++;
            if (f(x_i,a)! = f(x_j,a)) then m_ij = m_ij ⋃ a ;
            else if (flage >1) then break;
            }
          if (flage=1) then {
            if 存在 (m_ij, count)∈ DMSC(C) then
              从 DMSC(C) 中删除 (m_ij, count) 后
              插入 (m_ij, count+1);
            else 插入 (m_ij, 1) 到DMSC (C) 中;
          }
        }
end.
```

2.3 算法分析

因为当且仅当某个 m_{ij} 为单个属性时, 该属性属于核 Core(C,U), 所以当 m_{ij} 有 2 个或 2 个以上属性时, 它必不是核, 应停止该 m_{ij} 的计算. 将单个属性的 m_{ij} 保存至 DMSC(C).

3 改进的核增量式算法

3.1 算法思想

由于本文提出的算法不存储差别矩阵, 针对文献 [7] 的 3 种情况, 本文分别作如下处理:

1) x 与 $(U_1\bigcup U_2')$ 一致, 计算 x 与 $U_1\bigcup U_2'$ 间的 m_{ij}, 并按规则增加至 DMSC(C), $U_1=U_1\bigcup\{x\}$. 具体规则见下面的算法 2.

2) x 与 U_1 不一致时, 在 U_1 中找出与 x 不一致的 y, 计算 y 与 $U_1\bigcup U_2'$ 之间的 m_{ij}, 并将 DMSC(C) 中相应的 m_{ij} 按规则删除; $U_2'=U_2'\bigcup\{y\}$, $U_1=U_1-\{y\}$; 计算 y 与 U_1 之间的 m_{ij}, 按规则增加至 DMSC(C).

3) x 与 U_2' 不一致时, DMSC(C) 保持不变.

算法 2 改进的核增量式算法 (IOIUAC)

输入: 1) U_1, U_2', 由算法 1 得到的 DMSC(C);
2) 新增对象为 x.
输出: Core(C).
begin
if x 与 $(U_1 \bigcup U_2')$ 一致 then {
for each $x_i \in U_1 \bigcup U_2'$ do {
$m_{ij} = \Phi$, flage= 0;
if (($f(x_i, D)$!=$f(x, D)$ and $x_i \in U_1$)||$x_i \in U_2'$) then
for each $a \in C$ do {
flage++;
if $(f(x_i, a)! = f(x, a))$ then $m_{ij} = m_{ij} \bigcup a$;
else if (flage>1) then break;
}
if (flage= 1) then {
if 存在 (m_{ij}, count)∈DMSC(C) then 从 DMSC(C)中删除 (m_{ij}, count) 后插入 (m_{ij}, count+1);
else 插入 $(m_{ij}, 1)$ 到 DMSC(C) 中;
}
}
$U_1 = U_1 \bigcup \{x\}$;
}
else if x 与 U_1 不一致 then {
在 U_1 中找到与 x 不一致的对象 y;
for each $x_i \in U_1 \bigcup U_2'$ do { //此时的 $y \in U_1$
$m_{ij} = \Phi$; flage= 0;
if (($f(x_i, D)! = f(y, D)$ and $x_i \in U_1$)|| $x_i \in U_2'$) then
for each $a \in C$ do {
flage++;
if $(f(x_i, a)! = f(y, a))$ then $m_{ij} = m_{ij} \bigcup a$;
else if (flage> 1) then break;
}
if (flage= 1) then {
if 相应的 count>1 then 从 DMSC(C) 中删除 (m_{ij}, count) 后插入 (m_{ij}, count−1);
else 从 DMSC(C) 中删除 (m_{ij}, count);
}
}
$U_2' = U_2' \bigcup \{y\}; U_1 = U_1 - \{y\}$;
for each $x_i \in U_1$ do { //此时的 $y \in U_2'$
$m_{ij} = \Phi$; flage= 0;
for each $a \in C$ do {
flage++;
if $(f(x_i, a)! = f(y, a))$ then $m_{ij} = m_{ij} \bigcup a$;
else if (flage> 1) then break;
}
if (flage= 1) then {
if 存在 (m_{ij}, count)∈DMSC(C) then 从 DMSC(C) 中删除 (m_{ij}, count) 后插入 (m_{ij}, count+1);
else 插入 $(m_{ij}, 1)$ 到 DMSC(C) 中;
}
}
}
由 DMSC(C) 得到核 Core(C);
end.

3.2 算法分析

1) x 与 $(U_1 \bigcup U_2')$ 一致时, x 将加入 U_1 中, 文献 [7] 对差别矩阵增加了一行一列. 由上面的分析可知 $m_{ij} = m_{ji}$, 当 $x_i \in U_1, x_j \in U_1$ 时, 增加的列对于核计算而言是多余的, 所以, 算法 2 只需计算相应的行, 其计算量为 $|C| \times (|U_1| + |U_2'|)$. 判断 x 与 $(U_1 \bigcup U_2')$ 是否一致的时间为 $|U_1| + |U_2'|$, 故总的时间为 $2 \times |C| \times (|U_1| + |U_2'|)$.

2) x 与 U_1 不一致时, 计算 U_1 中与 x 不一致的对象 y, 删除 DMSC(C) 中 y 所在行的单个属性, 其计算量为 $|U_1| + |U_2'|$. 然后将 y 作为 U_2' 中的对象计算相应的 DMSC(C), 其计算量为 $|U_1|$. 判断一致性的时间为 $|C| \times (|U_1| + |U_2'|)$, 所以 x 与 U_1 不一致时总的时间为 $|C| \times (3 \times |U_1| + 2 \times |U_2'|)$. 故算法 2 的时间复杂度为 $O(|C| \times (3 \times |U_1| + 2 \times |U_2'|))$, 小于文献 [7] 中算法 2 的时间复杂度 $O(|C| \times (5 \times |U_1| + 3 \times |U_2'|))$.

文献 [7] 算法 2 的空间复杂度为 $O(|U_1| \times (|U_1| + |U_2'|))$, 本文算法 2 的空间复杂度为 $O(|C|)$, $|C|$ 为条件属性数. 因 $|C| \ll |U_1| + |U_2'|$, 故本文算法 2 较文献 [7] 中算法 2 的空间复杂度有显著的改善.

3.3 实验设计与结果

在内存为 1 024 MB, CPU 为 P IV 2.9 GHz, 操作系统为 Windows XP 的联想 PC 上, Eclipse 下 Java 实现了文献 [7] 中算法 2 及本文的算法 2. 记文献 [7] 中算法 2 为 OIUAC, 本文算法 2 为 IOIUAC. 利用 UCI 上所提供的蘑菇数据库进行实验, 该数据库有 8 124 个对象. 将蘑菇数据库视为决策表, 并进行以下两组实验:

实验 1　从 8 124 对象中选择 7 000 个对象作为基准决策表 (基准决策表的含义是指该决策表生成的

差别矩阵作为算法OIUAC和IOIUAC的输入), 从剩下的1 124个对象中依次选择200, 500, 800, 1 124个对象作为增量, 所得实验结果如图1所示.

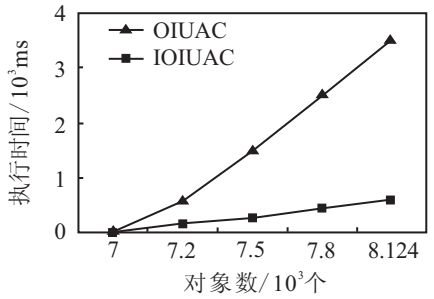

图 1 实验 1 算法的执行时间

实验 2 由蘑菇数据库生成8 000个对象, 其中不一致对象数为1 000. 从生成的8 000个对象中选择7 500个作为基准决策表, 从剩下的500个对象中依次选择100, 200, 300, 500个对象作为增量, 所得实验结果如图2所示.

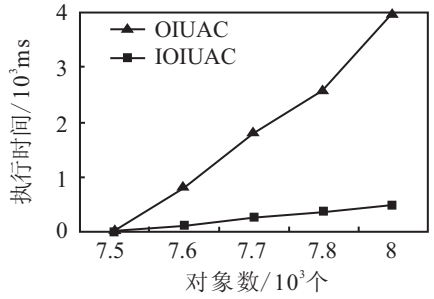

图 2 实验 2 算法的执行时间

3.4 实验分析

对于实验1而言, 8 124个对象为一致性对象. 算法OIUAC计算及遍历了差别矩阵相应的行和列; 而IOIUAC只需计算一行且优化了核属性计算算法, 所以IOIUAC的计算时间较少.

对于实验2, 当x与U_1不一致时, OIUAC需先遍历差别矩阵中与x相应的行和列, 并删除DMSC(C)中相关的核属性; 然后将该对象插入U_2, 并计算U_2中x相应的差别矩阵, 再将核属性插入DMSC(C). 此时IOIUAC的计算量明显少于OIUAC.

实验过程监测显示, 因为OIUAC需要存储差别矩阵, 随着对象数从7 200个增加到8 124个, 内存的使用增加很快, 从225 M增至285 M; 而IOIUAC一直保持稳定且只有较小的内存增加(只增加20 M左右). 所以, IOIUAC较OIUAC的另一重要优势为IOIUAC可以应对大数据集的挑战.

4 改进的属性约简完备算法及其分析

4.1 改进的属性约简完备算法

由文献[8]的定理4和定理5可知, 对于新增对象x, 若x与U_2一致或x与U_2中的某个对象不一致或x与U_1中的对象是P-一致的, 则P是一个属性约简. 因此新算法分为两种情况:

1) 新增对象x. 若x与U_2'一致或x与U_2'中的某个对象不一致或x与U_1中的对象是P-一致的, 则P是一个属性约简.

2) 其他情况. 由增量式核算法2计算得到的核开始, 重新计算属性约简.

算法 3 IIUAARI (improved incremental updating algorithm of attribute reduction for inserting)

输入: 1) U_1, U_2', 其定义同文献[7]; DMSC(C)为算法1的输出.

2) $P \in R(U)$; // $R(U)$为决策表的所有属性约简集, P为一个约简.

3) 新增对象为x.

输出: 一个属性约简$S \in R(U \bigcup \{x\})$.

begin

1) 运行算法2更新DMSC(C); //目的: 更新核

2) //依据x与U_1, U_2'中对象的一致和不一致, 进行属性约简更新

2.1) if x与U_2'一致或x与U_2'中的某个对象不一致或x与U_1中的对象是P-一致, then $S = P$;

2.2) else if x与U_1中的对象是P-不一致, then {

2.2.1) 由DMSC(C)得到核core;

2.2.2) 令$S =$ core;

if (S为空), then 在C中选最大SGF(a_i, S, D)取值的a_i, 令$S = a_i$;

2.2.3) 求$\mathrm{POS}_C(D)$, $\mathrm{POS}_S(D)$, $\mathrm{NEG}_S(D)$;

2.2.4) while $(\mathrm{POS}_C(D) \neq \mathrm{POS}_S(D))$ {

在C – S中选择最大SGF(a_i, S, D)取值的a_i; 如果这样的a_i有多个, 则在$(U - \mathrm{POS}_S(D))/\{a\}$中选择具有最小等价类个数的$a_i$; 若这样的$a_i$也有多个, 则任选一个;

$S = S + \{a_i\}$;

重新计算$\mathrm{POS}_S(D)$, $\mathrm{NEG}_S(D)$;

}

}

2.3) 输出一个属性约简$S \in R(U \bigcup \{x\})$;

end.

4.2 算法分析

算法3的时间复杂度分析如下:

步1) 的时间复杂度为$O(|C| \times (|U_1| + |U_2'|))$;

步2) 判断P-一致性为$O(|C| \times (|U_1| + |U_2'|))$;

步2.2) 为$O(U - \mathrm{POS}_P(D)) < O(|U|)$;

步2.2.1) 为$O(|C|)$;

步 2.2.3) 为 $O(|C|\times|U|)$;

步 2.2.4) 为 $O(|C|^2\times|U|)$.

算法 3 总的时间复杂度为 $O(|C|^2|\times U|)$.

由于算法 3 没有存储差别矩阵, 它较文献 [8] 的增量式属性约简算法的时间复杂度 $O(|C|^2\times|U|^2)$ 有显著的降低, 且与文献 [10] 的时空复杂度相当, 但文献 [10] 的求核算法只能用于求一致性决策表的核. 算法 3 充分利用已有的核, 能减少计算量, 且它适用于一致性、不一致性决策表的属性增量式约简. 当新增对象 x 与 U_2 中的某个对象不一致或 x 与 U_1 中的对象是 P- 一致时, 算法 3 的时间复杂度为 $O(|C|\times(|U_1|+|U_2'|))$, 能高效地求得属性约简.

4.3　计算示例

表 1 为二值数据表, 共有 5 个对象和 5 个属性, $C=\{C_1,C_2,C_3,C_4\}$ 为条件属性集, D 为决策属性.

表 1　二值数据表

对象	属　性				
	C_1	C_2	C_3	C_4	D
x_1	1	0	1	0	2
x_2	1	0	1	0	1
x_3	1	1	1	0	3
x_4	0	1	0	0	2
x_5	0	1	1	1	2

$U_1=\{x_3,x_4,x_5\}, U_2=\{x_1,x_2\}, U_2'=\{x_1\}$. 由文献 [7] 可得差别矩阵 M 为

$$\begin{array}{c} \\ x_3 \\ x_4 \\ x_5 \end{array}\begin{array}{c} \begin{array}{cccc} x_3 & x_4 & x_5 & x_1 \end{array} \\ \begin{bmatrix} \Phi & \{C_1,C_3\} & \{C_1,C_4\} & C_2 \\ \{C_1,C_3\} & \Phi & \Phi & \{C_1,C_2,C_3\} \\ \{C_1,C_4\} & \Phi & \Phi & \{C_1,C_2,C_4\} \end{bmatrix} \end{array}.$$

由文献 [7] 可得核为 $\{C_2\}$. 由定义 4 可知, $\{C_1, C_2\}$ 和 $\{C_2,C_3,C_4\}$ 为两个属性约简.

为进一步说明算法 3, 下面通过增加不同的对象来阐明对象插入后的属性约简更新情况.

1) 若新增对象 x 为 {1, 0, 1, 0, 3}, 则 x 与 x_1 不一致, 由算法 3 可知, 核和属性约简不改变.

2) 若新增对象 x 为 {1, 1, 1, 0, 1}, 则 x 与 x_3 不一致. 运用算法 3 的步 1) 运行算法 2 更新 DMSC(C). 若 x 与 U_1 中的对象不一致, 则转步 2.2) 和步 2.2.1), 得到核 core 为空. S 为空时, 计算 $\mathrm{SGF}(C_1,S,D)=2/6$, $\mathrm{SGF}(C_2,S,D)=0$, $\mathrm{SGF}(C_3,S,D)=1/6$, $\mathrm{SGF}(C_4,S,D)=1/6$. 在 C 中选择最大 $\mathrm{SGF}(a_i,S,D)$ 取值的 C_1, $S=\{C_1\}$, 此时 $\mathrm{POS}_C(D)=\mathrm{POS}_S(D)$, 所以 $S=\{C_1\}$ 为一属性约简.

3) 若新增对象 x 为 {0, 1, 0, 0, 3}, 则 x 与 x_4 不一致. 由算法 3 的步 1) 运行算法 2 以更新 DMSC(C). 若 x 与 U_1 中的对象不一致, 则转步 2.2) 和步 2.2.1), 得到核 core $=\{C_2\}$. $S=\{C_2\}$ 时, 计算 $\mathrm{SGF}(C_1,S,D)=1/6$, $\mathrm{SGF}(C_3,S,D)=0$, $\mathrm{SGF}(C_4,S,D)=1/6$, $(U-\mathrm{POS}_S(D))/\{C_1\}=2$, 选择 C_1 加入 S, 得 $S=\{C_1,C_2\}$. 重新计算 $\mathrm{SGF}(C_3,S,D)=1/6$, $\mathrm{SGF}(C_4,S,D)=1/6$, $(U-\mathrm{POS}_S(D))/\{C_1\}=2$, 选择 C_1 加入 S, $S=\{C_1,C_2,C_3\}$, 此时 $\mathrm{POS}_C(D)=\mathrm{POS}_S(D)$, 故 $S=\{C_1,C_2,C_3\}$ 为一属性约简.

4) 若新增对象 x 为{1, 1, 0, 1, 3}, 则 x 与原决策表中的对象 P- 一致, 由算法 3 可知, 核和属性约简不改变.

5) 若新增对象 x 为 {0, 1, 0, 1, 3}, 则 x 与原决策表中的对象一致. 由算法 3 的步 1) 运行算法 2 以更新 DMSC(C). 转步 2.2) 和步 2.2.1), 得到核 core $=\{C_2,C_3,C_4\}$. $S=\{C_2,C_3,C_4\}$, 此时 $\mathrm{POS}_C(D)=\mathrm{POS}_S(D)$, 所以 $S=\{C_2,C_3,C_4\}$ 为一属性约简.

综上所述, 算法 3 在各种情况下得到的约简与文献 [8] 中的算法相同. 本文的例子取自文献 [8].

4.4　实验结果及分析

选用 UCI 机器学习数据库中的 Mushroom 数据源, 在与 3.3 节相同的实验环境下分别实现了本文算法 IIUAARI 以及文献 [8-10] 中的约简算法 (简称为算法 A, 算法 B, 算法 C). 用 n 表示需要判断的实例数, m 表示原始条件属性数, r 表示约简后的条件属性数, t 表示约简时间 (算法的平均执行时间), $\eta=(m-r)/m$ 表示约简率, 实验结果如表 2 所示.

表 2　在 UCIMushroom 数据集上的约简算法实验结果

循环次数	n	m	IIUAARI			算法 A		
			r	t	η	r	t	η
第 1 次	214	18	14	1.412	0.222	14	1.634	0.222
第 2 次	155	19	13	1.056	0.316	17	1.314	0.105
第 3 次	8 124	22	17	5.863	0.227	21	6.012	0.045
第 4 次	3 190	20	14	3.735	0.3	17	4.112	0.15

循环次数	n	m	算法 B			算法 C		
			r	t	η	r	t	η
第 1 次	214	18	15	1.792	0.167	14	1.411	0.222
第 2 次	155	19	15	1.521	0.211	16	1.046	0.158
第 3 次	8 124	22	19	6.761	0.136	20	5.847	0.09
第 4 次	3 190	20	16	4.912	0.2	19	3.719	0.05

从表 2 可以看出, IIUAARI 在 t 上均优于算法 A 和算法 B, 与算法 C 的 t 基本一致, 但 IIUAARI的约简质量较高. 另外, 在几次循环实验中, IIUAARI 的约简效果比其他 3 种算法都好, 且通常可以找到最小或近似最小的约简. 表 2 给出了 4 种约简算法的 η, 结果表明, IIUAARI 在大大缩短 t 的同时, 约简率比其他 3 种算法明显提高, 且随着实例数的增加, IIUAARI 的约简效果更为显著.

5 结 论

已有的求核算法因为使用差别矩阵, 使得算法的空间复杂度较高. 为了降低空间复杂度, 本文在研究增量式求核算法的基础上, 提出了不存储差别矩阵的改进增量式求核算法, 并通过理论分析和实验表明了改进算法的有效性; 同时分析了已有的增量式属性约简算法效率低下的原因, 使用改进的增量式求核算法给出了一个高效的属性约简算法, 主要考虑对象动态增加情况下属性约简的更新问题. 该属性约简算法在动态求解核的基础上, 利用原有的属性约简有效地进行属性约简的增量式更新. 计算示例表明该算法是正确的, 较已有的属性约简增量式算法[8,11]更为高效.

本文的下一步工作是: 完善属性约简算法, 考虑对象删除情况下属性约简的更新问题.

参考文献(References)

[1] Pawlak Z. Rough sets[J]. Int J of Information and Computer Science, 1982, 11(5): 341-356.

[2] Wong S K M, Ziarko W. On optimal decision rules in decision tables[J]. Bulletin of Polish Academy of Sciences, 1985, 33(11/12): 663-676.

[3] Hu X H, Nick Cercone. Mining knowledge rules from databases: A rough set approach[C]. IEEE ICDE96. New Orleans, 1996: 96-105.

[4] Hu X H, Nick Cercone. Learning in relational databases: A rough set approach[J]. Int J of Computational Intelligence, 1995, 11(2): 323-338.

[5] 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.
(Wang G Y, Yu H, Yang D C. Decision table reduction based on conditional information entropy[J]. Chinese J of Computers, 2002, 25(7): 759-766.)

[6] Wang Jue, Wang Ju. Reduction algorithms based on discernibility matrix: The ordered attributes method[J]. J of Computer Science and Technology, 2001, 16(6): 489-504.

[7] 杨明. 一种基于改进差别矩阵的核增量式更新算法[J]. 计算机学报, 2006, 29(3): 407-413.
(Yang M. An incremental updating algorithm of the computation of a core based on the improved discernibility matrix[J]. Chinese J of Computers, 2006, 29(3): 407-413.)

[8] 杨明. 一种基于改进差别矩阵的属性约简增量式更新算法[J]. 计算机学报, 2007, 30(5): 815-822.
(Yang M. An incremental updating algorithm for attribute reduction based on improved discernibility matrix[J]. Chinese J of Computers, 2007, 30(5): 815-822.)

[9] 徐章艳, 杨炳儒. 一个基于决策表的快速属性约简算法[J]. 小型微型计算机系统, 2006, 25(5): 858-861.
(Xu Z Y, Yang B R. Quickly attribution reduction algorithm based on decision table[J]. J of Chinese Computer Systems,2006, 25(5): 858-861.)

[10] 葛浩, 李龙澍, 杨传健. 改进的快速属性约简算法[J]. 小型微型计算机系统, 2009, 30(2): 308-312.
(Ge H, Li L S, Yang C J. Improvement to quick attribution reduction algorithm[J]. J of Chinese Computer Systems, 2009, 30(2): 308-312.)

(上接第494页)

[11] 史成东, 陈菊红, 钟麦英. Downside-risk 测度下闭环供应链风险控制和利润分配机制研究[J]. 控制与决策, 2009, 24(11): 1693-1696.
(Shi C D, Chen J H, Zhong M Y. Risk controlling and profit distributing mechanism in closed-loop supply chain on theory of Downside-risk[J]. Control and Decision, 2009, 24(11): 1693-1696.)

[12] 叶飞, 李怡娜. 具有风险规避者加盟的供应链协作回购契约机制研究[J]. 工业工程与管理, 2006(2): 1-4.
(Ye F, Li Y N. Research on buy back contract mechanism for supply chain coordination with a risk-averse tetailer[J]. Industrial Engineering and Management, 2006(2): 1-4.)

[13] 于春云, 赵希男, 彭艳东, 等. 基于条件风险值理论的供应链优化与协调模型研究[J]. 中国管理科学, 2007, 15(3): 31-39.
(Yu C Y, Zhao X N, Peng Y D, et al. Study of supply chains optimization and coordination model based on conditional value-at-risk[J]. Chinese J of Management Science, 2007, 15(3): 31-39.)

[14] Uryasev S. Condition value-at-risk: Optimization algorithms and applications[J]. Financial Engineering News, 2000, 2(3): 1-5.

[15] Rockafellar R, Uryasev S. Optimization of conditional value-at-risk[J]. J of Risk, 2000, 2(3): 21-42.