文章编号: 1001-0920 (2012) 12-1769-07

一种基于 C-Tree 的属性约简增量式更新算法

杨 明, 吕 静

(南京师范大学 计算机科学与技术学院，南京 210046)

摘 要: 针对以往文献为克服基于差别矩阵的属性约简算法存储代价高的不足而提出的基于浓缩树 (C-Tree) 的高效属性约简算法仅考虑决策表不变的情况, 提出了一种基于 C-Tree 的属性约简增量式更新算法, 主要考虑对象动态增加情况下属性约简的更新问题. 该算法可通过快速更新 C-Tree, 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. 理论分析和实验结果表明, 所提出的算法是有效可行的.

关键词: 粗糙集；差别矩阵；浓缩树；核；属性约简；增量式更新

中图分类号: TP311 **文献标志码:** A

An incremental updating algorithm for attribute reduction based on C-Tree

YANG Ming, *LV Jing*

(School of Computer Science and Technology，Nanjing Normal University，Nanjing 210046，China. Correspondent: YANG Ming，E-mail：m.yang@njnu.edu.cn)

Abstract: To overcome the disadvantage of those attribute reduction algorithms based on the discernibility matrix with high space complexity, a compact storage structure called condensing tree(C-Tree) and corresponding efficient algorithms for attribute reduction are introduced in the existing reference, respectively. However, the mentioned algorithms in the reference only consider the case of the static decision table. Therefore, an incremental updating algorithm is proposed for attribute reduction based on C-Tree in the case of inserting, which only needs to modify the related nodes in the corresponding paths when updating the C-Tree. After dynamically computing a core, attribute reduction can be effectively updated by utilizing the old attribute reduction. Theoretical analysis and experiments show that the proposed algorithm is effective and feasible.

Key words: rough set；discernibility matrix；C-Tree；core；attribute reduction；incremental updating

1 引 言

粗糙集 (RS) 是一种新的处理不精确、不完全与不相容知识的数学理论[1], 近年来该理论在机器学习、数据挖掘及模式识别等多个领域得到了广泛的应用[2-3]. 作为有效的语义保持属性约简方法[4], 与主成分分析 (PCA) 相结合并用于模式识别, 提高了识别率[5], 因而受到了粗集研究者的重视. 现有的属性约简大体上可分为基于差别矩阵或在此基础上改进的属性约简算法[6-8]、基于正区域的属性约简算法[9-10]和基于启发式的属性约简算法[11-12], 其中差别矩阵模型下的属性约简方法因其简洁、高效备受关注. 然而, 对一个具有 N 个对象 m 个属性的决策表, 其差别矩阵的存储复杂度为 $O(N^2 * m)$, 因此当 N 或 m 较大时, 差别矩阵的存储代价较高.

为了有效降低差别矩阵的存储代价, 文献 [13] 在引入浓缩树 (C-Tree) 后, 提出了基于 C-Tree 的属性约简算法, 该算法将差别矩阵的各非空元素存储在浓缩树的相应路径上, 使得差别矩阵的大量非空元素压缩在浓缩树的相应路径上, 从而可有效地降低存储开销, 改进属性约简的性能. 但与众多属性约简算法一样, 文献 [13] 提出的算法主要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. 而现实世界是发展变化的, 决策表中的对象不断地在动态变化, 已得到的属性约简将可能不再有效, 这便需要对属性约简进行动态修改. 文献 [14] 给出了 C-Tree 的改进算法, 有效降低了存储代价. 此外, 现

收稿日期: 2011-07-20；**修回日期:** 2012-01-03.

基金项目: 国家自然科学基金项目(60873176, 61003116, 61272222)；江苏省自然科学基金重点重大专项项目(BK2011005)；江苏省自然科学基金项目(BK2011782, BK2010263).

作者简介: 杨明(1964−), 男, 教授, 博士, 从事数据挖掘、机器学习等研究；吕静(1978−), 女, 讲师, 从事模式识别、机器学习的研究.

有的属性约简更新算法[15-18]不能直接用于C-Tree结构下的属性约简更新, 且刘宗田教授提出属性最小约简的增量式算法主要考虑不带决策属性的信息系统情况下的属性约简[15]. 文献[16]的属性约简动态更新算法计算量大, 而王国胤教授等[17]提出的基于正区域的属性约简增量式更新算法改进了文献[15-16]的不足, 但未能有效利用更新的核. 为改进文献[18]的不足, 文献[17]提出一种基于差别矩阵的属性约简增量式更新算法, 该算法有效地利用了更新的核, 但其基于差别矩阵的存储结构, 因而空间存储代价较大.

为改进基于差别矩阵的属性约简更新方法存储代价大的不足, 本文提出了一种基于C-Tree的属性约简增量式更新算法, 主要讨论决策表动态增加情况下的属性约简. 该算法可通过快速更新C-Tree, 在动态求解核的基础上, 有效地利用原有的属性约简进行属性约简的增量式更新, 因而可有效改进属性约简的更新效率. 理论分析和实验结果表明, 本文提出的算法是可行有效的.

2 相关概念与结论

2.1 粗糙集概念

为节省篇幅, 仅介绍与属性约简及核有关的一些概念, 粗糙集的其他一些概念可参见文献[2-3].

决策表DT是一个四元组$\langle U, C\bigcup D, V, f\rangle$, 其中$U$是一组对象的非空有限集合. 设有$n$个对象, 则$U$可表示为$U=\{x_1, x_2, \cdots, x_n\}$; $C\bigcup D$是属性集合, C为条件属性集, D为决策属性集; $V=\bigcup\limits_{a\in(C\bigcup D)} V_a$, V_a为属性a的值域集; f是$U\times(C\bigcup D)\to V$的映射. 对$B\subseteq C\bigcup D$, 无差别关系IND(B)定义为$\{(x,y)\in U^2|\forall a\in B, f(x,a)=f(y,a)\}$, 通过IND($B$)将$U$划分为若干个类$E_i(1\leqslant i\leqslant|U/\text{IND}(B)|)$. 为便于叙述, 设条件属性集合$C$中有$m$个属性$C_1, C_2, \cdots, C_m$, 其值域为有限离散集合, 并用$|\cdot|$表示集合的基. 不失一般性, 假设仅有一个决策属性D, 其取值范围是$1, 2, \cdots, k$, 由D导出的等价类构成U的一个划分$\{\psi_1, \psi_2, \cdots, \psi_k\}$, 其中$\psi_i=\{x\in U: f(x,D)=i\}, i=1,2,\cdots,k$.

定义 1 设$X\subseteq U$为论域的一个子集, $P\subseteq C$, X的关于P的下近似为$\underline{P}X(U)=\{x\in U: [x(U)]_p\subseteq X\}$, 其中$[x(U)]_p$表示$U$中所有与$x$在关系IND($P$)下是等价的元素构成的集合.

定义 2 设$P\subseteq C$, 对划分$\{\psi_1, \psi_2, \cdots, \psi_k\}$的$P$近似精度为$\gamma_P(U)=\sum\limits_{i=1}^{k}|\underline{P}\psi_i(U)|/|U|$, 其中记$P(U)=\sum\limits_{i=1}^{k}|\underline{P}\psi_i(U)|$.

定义 3 设$P\subseteq C$, 若$\gamma_P(U)=\gamma_p(C)$, 且不存在$R\subset P$, 使得$\gamma_R(U)=\gamma_C(U)$, 则称P为C的一个(相对于决策属性D的)属性约简. 所有C的属性约简的交称为C的核(简称核), 记为Core(U).

为改进属性约简的效率, 文献[6-8,17]提出了基于差别矩阵的属性约简算法, 有关差别矩阵的定义及其核和属性约简等价求解策略如下.

定义 4 对于给定的决策表DT, 定义差别矩阵$M=\{m_{ij}\}$为

$$m_{ij}=\begin{cases}\{a\in C: f(x_i,a)\neq f(x_j,a)\},\ f(x_i,D)\neq f(x_j,D\};\\ \varnothing,\ \text{otherwise.}\end{cases}\tag{1}$$

基于定义4, 可得到差别矩阵结构下属性约简的求解策略: 对于给定的决策表DT, 其差别矩阵为M, $P(P\subseteq C)$为属性约简当且仅当M中的任意非空元素m_{ij}有$m_{ij}\bigcap P\neq\varnothing$, 且$\forall S\subset P$, 存在非空元素$m_{kl}\in M$使得$m_{kl}\bigcap S=\varnothing$.

性质 1 对给定的决策表DT, 其差别矩阵为M, 属性$a\in$Core(U)当且仅当$\{a\}$为差别矩阵M的非空元素.

2.2 浓缩树 (C-Tree)

为有效降低差别矩阵的存储代价, 保持差别矩阵中隐含的信息不丢失, 文献[13]引入了C-Tree及其生成算法.

定义 5 一棵浓缩树为满足以下3个条件的树型结构: 1) 它由一个标为"null"的根结点(用root表示)、作为根结点的孩子的属性前缀子树集合以及属性头表组成. 2) 属性前缀子树中的每一结点包含attribute-name, count, stcount, parent, child, node-link六个域. 其中: attribute-name记录属性名; count记录能到达该结点的路径所表示的非空元素的数目; stcount记录该结点的所有孩子结点的count值之和; parent和child分别表示某个结点指向父结点的指针和指向孩子结点集的头指针; node-link为指向C-tree中具有相同的attribute-name值的下一结点, 当下一个结点不存在时, node-link为null. 3) 属性头表的每一项包含3个域: attribute-name, frequency, head. 其中: frequency为属性名为attribute-name的属性在差别矩阵中出现的次数, head为指向C-tree中具有相同的attribute-name值的首结点的指针.

对于给定的某决策表DT, 建立C-tree的主要思路为: 1) 将属性集C的各属性按照某种排列次序(次序记为$\Re$)形成头表; 2) 将差别矩阵中的每个非空元素, 按组成相应的属性序列插入C-tree中, 且插入时

要求父子结点之间和孩子兄弟结点之间均保持次序$\Re$. C-tree的生成算法描述如下:

算法1　GenCT (Generation of C-Tree).

输入: DT$=\langle U, C\bigcup D, V, f\rangle$为一决策表, 其中$U$中含有$n$个对象, $C(C=\{C_1, C_2, \cdots, C_m\})$中含有$m$个条件属性, D为决策属性.

输出: 一棵C-tree.

Step 1: 条件属性集C的各属性按照某种排列次序排列, 记为$\Re$.

Step 2: 按照下列步骤建立C-Tree.

Step 2.1: 建立一棵C-Tree的根root, 并标记为"null", 指向root的指针记为T;

Step 2.2: 建立头表HT, 按照$\Re$依次记录各属性名attribute-name, 将其frequency及head分别置为0和null;

Step 2.3: for $i=1$ to $|U^{\rm new}|$ do

/* x_i, x_j为两个对象, */

for $j=1$ to $i-1$ do

if $f(x_i, D)\neq f(x_j, D)$ then

if $m_{ij}\neq\varnothing$ then {将m_{ij}中的各属性按照次序形成序列并记为$[a|R]$, a为首属性, R为其余的属性;

调用InsertCT($[a|R], T$)}.

/* 函数InsertCT($[a|R], T$)采用递归调用的方法, 若T有孩子P使得$P.{\rm attribute-name}=a$, 则T.stcount增1, P.count增1; 否则生成新的孩子结点P, P.count置1, P.stcount置0, 并按次序$\Re$将结点P插入T的孩子结点中, 并将父结点的stcount值增1; 同时, 通过node-link将该结点链接起来. */

Step 3: for 头表中的每个属性table[i].attribute-name do.

table[i].frequency=table[i].head指向的单链表中各结点的count值之和.

例1　表1为一决策表, 该决策表共有5个对象和4个属性, 条件属性为$C=\{C_1, C_2, C_3\}$, 决策属性为d.

表1　决策表

属性	对象			
	C_1	C_2	C_3	d
x_1	2	2	0	1
x_2	1	2	0	1
x_3	0	0	0	0
x_4	1	0	1	0
x_5	2	0	1	1

对于例1, 由定义4可得差别矩阵M(因M的对称性, 这里仅给出下三角), 即式(2). 由式(2)可知, M为5×5的矩阵, 其下三角非空元素为6个. 依据算法1, 由式(2)中差别矩阵的下三角非空元素生成的C-Tree如图1所示(除根结点外, 每个结点用attribute-name: count: stcount表示), 其中属性的排列次序$\Re$为C_1, C_2, C_3. 由图1可知, 表1的差别矩阵的非空元素被压缩到3条路径中, 路径的最大长度为3; 总的树结点数为6, 因而对差别矩阵的非空元素起到压缩作用.

$$P=\begin{array}{c} \\ x_1\\ x_2\\ x_3\\ x_4\\ x_5\end{array}\begin{array}{c}\begin{array}{ccccc} x_1 & x_2 & x_3 & x_4 & x_5\end{array}\\ \left[\begin{array}{ccccc} \varnothing & & & & \\ \varnothing & \varnothing & & & \\ \{C_1, C_2\} & \{C_1, C_2\} & \varnothing & & \\ \{C_1, C_2, C_3\} & \{C_2, C_3\} & \varnothing & \varnothing & \\ \varnothing & \varnothing & \{C_1, C_3\} & \{C_1\} & \varnothing\end{array}\right]\end{array}. \tag{2}$$

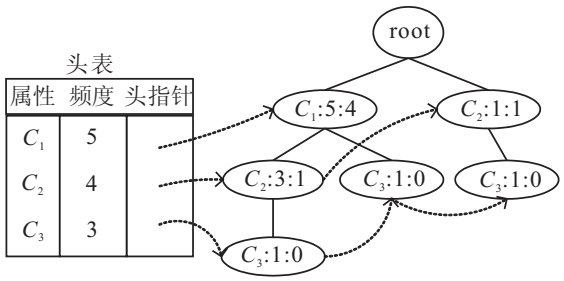

图1　例1的浓缩树$T^{\rm old}$

对于一个含有N个对象的决策表, 其差别矩阵M为$N\times N$的矩阵. 当N较大, M中含有大量非空元素时, M的存储代价较大, 影响属性约简求解性能, 且某些情况下使得基于差别矩阵的属性约简算法不再有效. 而对于一个大规模数据集, 其差别矩阵中必含有大量的重复或部分重复的非空元素, 这些非空元素将共享C-Tree的一条路径或一条子路径, 故可有效降低差别矩阵的存储代价, 同时保持差别矩阵的信息不丢失.

3　基于C-Tree的核和属性约简

为了便于讨论, 引入有效路径的定义, 给出C-Tree存储结构下核求解和属性约简的主要策略.

定义6　对于给定的决策表DT, 由算法1生成的C-Tree为T, N_s为T的某个结点, 若N_s.count>N_s.stcount, 则称从N_s结点沿着Parent指针到根结点root的孩子N_1构成的路径为一条有效路径L, 记为$L=\langle N_1, N_2, \cdots, N_s\rangle$; 路径$L$中的各结点对应的属性名构成的集合记为$S(L)$.

依据算法1和定义6, 基于C-Tree的核求解和属性约简算法可由下面的定理得到.

定理 1 对于给定的决策表DT, 若相应的差别矩阵为M, 由算法1生成的C-Tree为T, 则$a \in \mathrm{Core}(U)$当且仅当T中存在结点P使得$P.\mathrm{count} > P.\mathrm{stcount}$, 其中$P$为root的孩子结点.

证明 由性质1可知, $a \in \mathrm{Core}(U)$当且仅当$\{a\}$为差别矩阵M的非空元素, 而$\{a\}$为差别矩阵M的非空元素当且仅当T中存在root的孩子结点P使得$P.\mathrm{count} > P.\mathrm{stcount}$. □

由定理1可知, 在生成的C-Tree中, 仅需扫描根结点的孩子结点即可求得核, 因而由C-Tree可快速求解核. 对于例1, 由图1可知, root的孩子结点中, 只有结点"$C_1:5:4$"满足定理1的条件, 因此$\mathrm{Core}(U) = \{C_1\}$.

定理 2 对于给定的决策表DT, 若相应的差别矩阵为M, 则由算法1生成的C-Tree为T, 若$P(P \subseteq C)$为属性约简当且仅当下列条件均成立:

1) 对于T中的任意有效路径L, 有$S(L)\bigcap P \neq \varnothing$成立(满足该条件的$P$被简称为候选属性约简);

2) 对于任意$R \subset P$, T中必存在一条有效路径L_1使得$S(L_1)\bigcap R = \varnothing$成立.

证明 P是一个属性约简当且仅当P与M中的任意非空元素的交非空, 且$\forall R \subset P$, 存在M中的一个非空元素m使得$m\bigcap R = \varnothing$, 不妨设m对应的有效路径为L_1. P是一个属性约简, 则对于T中的任意一条有效路径$L = \langle N_1, N_2, \cdots, N_s\rangle$, 由算法1可知, 有$S(L)\bigcap P \neq \varnothing$, 而对$\forall R \subset P$, 有$S(L_1)\bigcap R = \varnothing$.

反之, 对于T中的任意一条有效子路径$L = \langle N_1, N_2, \cdots, N_s\rangle$, $S(L)\bigcap P \neq \varnothing$, 且对$\forall R \subset P$, T中必存在一条有效路径L_1使得$S(L_1)\bigcap P = \varnothing$成立, 则由算法1可知, P与M中的任意非空元素的交非空, 且存在M中的一个非空元素m使得$m\bigcap R = \varnothing$. 这里$m = S(L_1)$, 因而P是一个属性约简. □

定理2为基于C-Tree的属性约简算法的设计提供了理论依据. 依据定理2, 可设计出高效的属性约简算法, 如文献[13]提出的启发式属性约简算法, 其总体思路是: 1) 依据给定的决策表, 建立一棵C-Tree T; 2) 依据T求解核$\mathrm{Core}(U)$, $\mathrm{reduct} = \mathrm{Core}(U)$; 3) 快速修改$T$中使得$S(L)\bigcap \mathrm{reduct} \neq \varnothing$的有效路径$L$, 若$T$不空, 则进一步扩展重要属性$a$并将$a$加到reduct, 转3); 否则结束. 生成的C-Tree通常是一棵高度浓缩的树, 因此改进了求属性约简的效率.

针对例1, 采用上述算法, 由于$\mathrm{Core}(U) = \{C_1\}$, 快速消除了包含$C_1$的所有有效路径(因$C_1:5:4$为一棵子树的根结点, 由$C_1:5:4$出发的所有路径都包括$C_1$, 故可直接删除该子树), 仅有路径$C_2:1:1 \rightarrow C_3:1:0$保留, 因此进一步选择$C_2$或$C_3$即可快速得到属性约简$\{C_1, C_2\}$或$\{C_1, C_3\}$.

现有基于C-Tree的属性约简仅适合静态决策表的属性约简, 而现实世界是发展变化的, 决策表中的对象在不断动态变化, 已得到的属性约简将可能不再有效. 为此, 探索研究基于C-Tree的属性约简更新算法成了本文的主要目标.

4 基于C-Tree的属性约简增量式更新

限于篇幅, 本文侧重讨论一致决策表下的属性约简增量式更新, 有关不一致决策表下的C-Tree生成及相应属性约简更新将另文讨论. 为方便计, 设原给定的决策表为$\mathrm{DT}^{\mathrm{old}} = \langle U^{\mathrm{old}}, C\bigcup D, V, f\rangle$, 相应于$\mathrm{DT}^{\mathrm{old}}$的C-Tree为$T^{\mathrm{old}}$, 对新增对象构成的决策表为$\mathrm{dt}^{\mathrm{new}} = \langle U^{\mathrm{new}}, C\bigcup D, V, f\rangle$, $\mathrm{DT} = \langle U, C\bigcup D, V, f\rangle$, 其中$U = U^{\mathrm{old}}\bigcup U^{\mathrm{new}}$.

4.1 C-Tree的更新

一种简单的基于C-Tree的属性约简增量式更新方法是重新生成C-Tree并进行属性约简, 即重新生成基于$\mathrm{DT}(\mathrm{DT} = \mathrm{DT}^{\mathrm{old}}\bigcup \mathrm{dt}^{\mathrm{new}})$的C-Tree并进行属性约简. 显然, 这种方法的属性约简效率低. 为此, 如何在已生成的T^{old}和已求得的属性约简的基础上快速有效地生成新的C-Tree及相应的属性约简成了本节的主要目标.

给定决策表为$\mathrm{DT}^{\mathrm{old}}$, 若生成的C-Tree为$T^{\mathrm{old}}$, 则对新增对象构成的决策表$\mathrm{dt}^{\mathrm{new}}$, 仅需比较$U^{\mathrm{new}}$中不同类对象及$U^{\mathrm{new}}$和$U^{\mathrm{old}}$之间不同类对象; 若由此形成的可区分属性序列在$T^{\mathrm{old}}$中有相应的有效路径, 则将该属性序列插入$T^{\mathrm{old}}$, 否则插入$T^{\mathrm{new}}$; 将$T^{\mathrm{new}}$与$T^{\mathrm{old}}$合并得到对应DT的C-Tree T. 浓缩树T^{old}的更新算法UCT(Updating of Condensing Tree)描述如下.

算法 2 UCT.

输入: 1) $\mathrm{DT}^{\mathrm{old}} = \langle U^{\mathrm{old}}, C\bigcup D, V, f\rangle$为原决策表, 相应的C-Tree为$T^{\mathrm{old}}$, $\Re$为给定的属性序列; 2) $\mathrm{dt}^{\mathrm{new}} = \langle U^{\mathrm{new}}, C\bigcup D, V, f\rangle$为新增对象构成的决策表, 设$\mathrm{DT} = \langle U, C\bigcup D, V, f\rangle$, $U = U^{\mathrm{old}}\bigcup U^{\mathrm{new}}$.

输出: C-Tree T^{new}和T.

Step 1 生成T^{new}的头表和root结点, 初始化T^{new}.

Step 2 在新增对象之间进行比较, 并将生成的可区分属性序列插入T^{old}和T^{new}, x_i, x_j为两个对象.

for $i = 1$ to $|U^{\mathrm{new}}|$ do

for $j = 1$ to $i - 1$ do

if $f(x_i, D) \neq f(x_j, D)$ then

if $m_{ij} \neq \varnothing$ then {将m_{ij}中的各属性按照$\Re$次

序形成序列并记为 $[a|R]$, a 为首属性, R 为其余的属性;

if 在 $T^{\rm old}$ 中存在对应的有效路径 then

调用 InsertCT($[a|R], T^{\rm old}$);

else 调用 InsertCT($[a|R], T^{\rm new}$); }

Step 3　将新增对象与原决策表中的对象进行比较, 并将生成的可区分属性序列插入 $T^{\rm old}$ 和 $T^{\rm new}$, x_i, x_j 为两个对象.

for $x_i \in U^{\rm new}$ do

for $x_j \in U^{\rm old}$ do

if $f(x_i, D) \neq f(x_j, D)$ then

if $m_{ij} \neq \varnothing$ then {将 m_{ij} 中的各属性按照次序形成序列并记为 $[a|R]$, a 为首属性, R 为其余的属性;

if 在 $T^{\rm old}$ 中存在对应的有效路径 then

调用 InsertCT($[a|R], T^{\rm old}$);

else 调用 InsertCT($[a|R], T^{\rm new}$) }.

Step 4　for 头表中的每个属性 table[i].attribute-name do.

table[i].frequency = table[i].head 指向的单链表中各结点的 count 值之和.

Step 5　$T = {\rm merge}(T^{\rm old}, T^{\rm new})$. 将 $T^{\rm new}$ 的相关路径并入 $T^{\rm old}$ 中, 从而得到基于 DT 的 C-Tree T.

由算法 2 可知, UCT 无须进行 $U^{\rm old}$ 中不同类对象之间的比较, 而仅需比较 $U^{\rm new}$ 中不同类对象及 $U^{\rm new}$ 和 $U^{\rm old}$ 之间不同类对象. 若设 $N_1 = |U^{\rm old}|, N_2 = |U^{\rm new}|, m = |C|$, 则 UCT 的时间复杂度为 $O((N_1 + N_2) * N_2 * m)$, 当 $N_2 \ll N_1$ 时, UCT 的时间复杂度近似为 $O(N_1 * N_2 * m)$. 若不利用已得到的 $T^{\rm old}$, 重新运行算法 1 来生成基于 DT 的 C-Tree, 则相应的时间复杂度为 $O((N_1 + N_2) * (N_1 + N_2) * m)$. 可见, 新设计的 UCT 算法可有效改进 C-Tree 的更新效率.

4.2　基于 C-Tree 的属性约简增量式更新

为了从已给定的属性约简和更新后的 C-Tree 中快速得到新的属性约简, 引入下面的定理.

定理 3　原给定的决策表为 DT$^{\rm old}$, 由算法 1 生成的 C-Tree 为 $T^{\rm old}$; 新增对象构成的决策表为 dt$^{\rm new}$, 由算法 2 生成的两棵 C-Tree 为 $T^{\rm new}$ 和 T. 若 P 为由 $T^{\rm old}$ 得到的属性约简, 且对于 $T^{\rm new}$ 的任意所有有效路径 L 有 $S(L) \bigcap P \neq \varnothing$ 成立, 则 P 为决策表 DT 下的属性约简.

证明　若对于 $T^{\rm new}$ 的任意所有有效路径 L 有 $S(L) \bigcap P \neq \varnothing$ 成立, 则 P 是 DT 下的候选属性约简; 而对于 $\forall R \subset P$, 因 P 是 DT$^{\rm old}$ 下的属性约简, 故在 $T^{\rm old}$ 中必存在某个路径 L_1 使得 $S(L) \bigcap P = \varnothing$, 因此 P 为决策表 DT 下的属性约简. □

定理 4　原给定的决策表为 DT$^{\rm old}$, 由算法 1 生成的 C-Tree 为 $T^{\rm old}$; 新增对象构成的决策表为 dt$^{\rm new}$, 由算法 2 生成的两棵 C-Tree 为 $T^{\rm new}$ 和 T. 若 P 为由 $T^{\rm old}$ 得到的属性约简, 且在 $T^{\rm new}$ 中存在某条有效路径 L_1 使得 $S(L_1) \bigcap P = \varnothing$ 成立, 则 P 不是决策表 DT 下的属性约简.

证明　若在 $T^{\rm new}$ 中存在某条有效路径 L_1 使得 $S(L_1) \bigcap P = \varnothing$ 成立, 则将 $T^{\rm new}$ 并入 $T^{\rm old}$ 后得 T, L_1 必是 T 的一条有效路径, 因而 P 不是决策表 DT 下的属性约简. □.

原给定的决策表为 DT$^{\rm old}$, 由算法 1 生成的 C-Tree 为 $T^{\rm old}$; 新增对象构成的决策表为 dt$^{\rm new}$, 依据定理 3 和定理 4, 当决策表动态增加时, 属性约简更新的主要思路是: 1) 由算法 2 生成的两棵 C-Tree 为 $T^{\rm new}$ 和 T; 2) 若原属性约简 P 满足定理 3, 则 P 仍为属性约简; 否则, 通过 T 结构, 由定理 1 快速得到核 Core(U), 并依据文献 [13] 的启发式属性约简的思路, 由 Core(U) 逐步扩展可得到一个属性约简.

由上述分析, 基于 C-Tree 的属性约简更新算法描述如下.

算法 3　IUARCT(Incremental updating algorithm for attribute reduction based on C-Tree).

输入: 1) DT$^{\rm old}$ = $\langle U^{\rm old}, C \bigcup D, V, f\rangle$ 为原决策表, 相应的 C-Tree 为 $T^{\rm old}$, $\Re$ 为给定的属性序列; 2) DT$^{\rm new}$ = $\langle U^{\rm new}, C \bigcup D, V, f\rangle$ 为新增对象构成的决策表, 设 DT = $\langle U, C \bigcup D, V, f\rangle$, $U = U^{\rm old} \bigcup U^{\rm new}$; 3) P 为由 $T^{\rm old}$ 得到的属性约简.

输出: 一个属性约简 reduct.

Step 1: 生成 $T^{\rm new}$ 并更新浓缩树 $T^{\rm old}$, 由算法 2 生成 $T^{\rm new}$ 并更新 $T^{\rm old}$ 得到 T.

Step 2: 依据定理 3 和定理 4, 若 P 满足定理 3 的条件, 则 reduct = P, 转到 Step 4.

Step 3: 类似文献 [13], 有:

Step 3.1: 由定理 1 和 T 结构快速求得核 Core(U);

Step 3.2: reduct = Core(U);

Step 3.3: 由 T 结构, 逐步扩展重要属性直到得到一个属性约简 reduct.

Step 4: 结束.

由算法 3 可知, 在最坏的情况下, IUARCT 需要执行 Step 3. 与基于 C-Tree 的静态属性约简算法相比, 在采用相同的属性约简策略情况下, 基于 C-Tree 的动态属性约简节约的时间至少为 $O(N_1 * N_1 * m)$. 因此, 当 $N_1 \gg N_2$ 时, IUARCT 可有效提高属性约简的效率. 此外, 某些情况下, IUARCT 还可由 Step 2 直接得到属性约简, 因而将进一步提高属性约简更新的效率.

4.3 示例说明

为说明对象动态增加情况下, C-Tree 和属性约简的更新, 分以下两种情况进行讨论.

情况 1 对于例 1, 若增加两个对象 $x = (1,1,0,0)$ 和 $y = (0,1,1,1)$, 则由算法 2 可得更新后的浓缩树 T 如图 2 所示. 因 T^{new} 为空, 故原属性约简 $\{C_1, C_2\}$ 或 $\{C_1, C_3\}$ 也是对象动态增加后的属性约简.

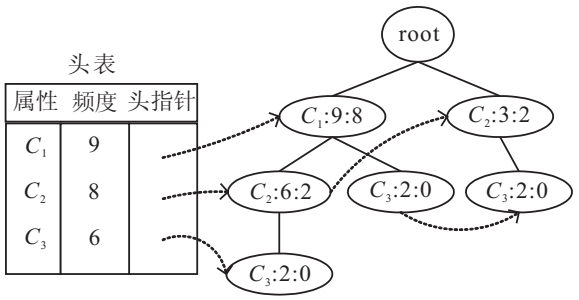

图 2 更新后的浓缩树 T

情况 2 对于例 1, 若增加一个对象 $x = (0,0,1,1)$, 则由算法 2 可得更新后的浓缩树 T, 如图 3 所示. 因 T^{new} 不为空, 且原属性约简 $\{C_1, C_2\}$ 与其有效路径上的属性不相交, 故由定理 4 可知, $\{C_1, C_2\}$ 不是对象动态增加后的属性约简. 而对于原属性约简 $\{C_1, C_3\}$, 因其与 T^{new} 的有效路径上的属性集有非空交, 故由定理 3 知, $\{C_1, C_3\}$ 为对象动态增加后的属性约简.

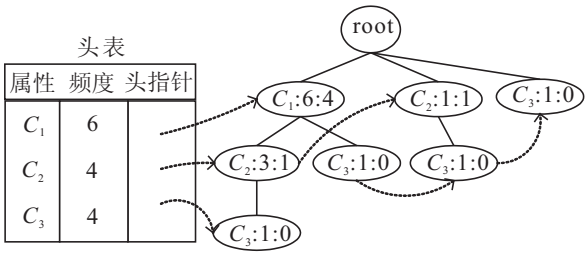

图 3 更新后的浓缩树 T

由以上分析可知, 算法 3 利用已生成的 C-Tree 和属性约简, 可有效提高属性约简的更新效率.

4.4 实验结果

因 C-Tree 更新是基于 C-Tree 的属性约简动态更新的关键步骤, 也是影响属性约简更新效率的重要因素, 所以, 侧重对 C-Tree 修改效率进行了测试. 为方便计, 文献 [13] 提出的 C-Tree 生成的静态算法被记为 old 算法. 在内存为 512 M, CPU 为 P1.73 GHz 的 PC 机上, 用 VC++6.0 实现 old 和 UCT 算法. 实验数据采用 http:// www.ics.uci.edu 上的蘑菇数据集 Mushroom (8 124 个对象, 22 个条件属性, 1 个决策属性, 2 个类别) 和人工合成数据集 Dataset(7 000 个对象, 23 个条件属性, 1 个决策属性, 条件属性和决策属性值均由 0～9 的数字随机生成). 为验证 C-Tree 更新算法的有效性, 分别从 Mushroom 和 dataset 两个数据集中随机抽取 100, 500, 1 000, 1 500, 2 000 个对象作为新增决策表 (dt^{new}) 中的对象集 (U^{new}), 剩下的对象作为原决策表 (DT^{old}) 中的对象集 (U^{old}), 实验结果如图 4 和图 5 所示.

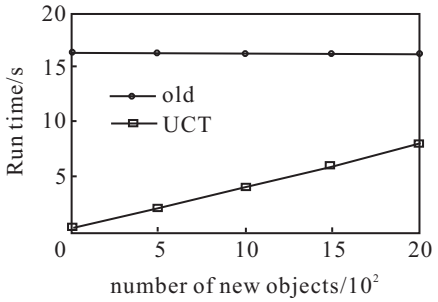

图 4 算法运行时间 (Mushroom)

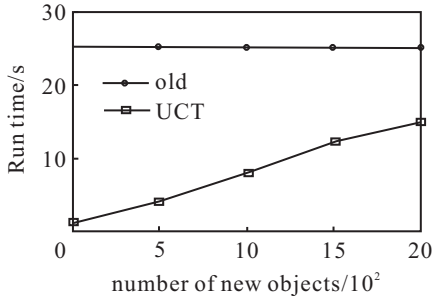

图 5 算法运行时间 (dataset)

由图 4 和图 5 可见, UCT 可有效提高 C-Tree 的更新效率, 而静态 C-Tree 生成算法 old 效率低. 特别地, 当增量决策表中的对象数相对较小时, UCT 可显著提高 C-Tree 的更新效率; 同时, 对新的 C-Tree, 在属性约简策略一致的情况下, IUARCT 用于属性约简上的代价低于文献 [13] 的相应算法, 因而 IUARC 有效地提高了属性约简的效率.

5 结 论

本文基于改进的高效差别矩阵压缩存储结构 C-Tree, 提出了一种基于 C-Tree 的属性约简增量式更新算法, 主要考虑对象动态增加情况下属性约简的更新问题. 该算法通过快速更新 C-Tree, 并利用原有的属性约简有效地提高属性约简的增量式更新效率. 相对于基于差别矩阵的属性约简增量更新方法, 新提出的算法仅需要增加或修改 C-Tree 的相关路径即可, 故为属性约简的快速更新提供了一条新的途径.

参考文献(References)

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

[2] Pawlak Z. Rough set approach to multi-attribute decision analysis[J]. European J of Operational Research, 1994,72(3): 443-459.

[3] 刘清. Rough集及Rough推理[M]. 北京: 科学出版社, 2001.
(Liu Q. Rough sets and rough reasoning[M]. Beijing: Science Press, 2001.)

[4] Jensen R, Shen Q. Semantics-preserving dimensionality reduction: Rough and fuzzy-rough-based approaches[J]. IEEE Trans on Knowledge and Data Engineering, 2004, 16(12): 1457-1471.

[5] Swiniarski R W, Skowron A. Rough set methods in feature selection and recognition[J]. Pattern Recognition Letters, 2003, 24(6): 833-849.

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

[7] 叶东毅. Jelonek属性约简算法的一个改进[J]. 电子学报, 2000, 28(12): 81-82.
(Ye D Y. An improvement to Jelonek's attribute reduction algorithm[J]. Acta Electronica Sinica, 2000, 28(12): 81-82.)

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

[9] 刘少辉, 盛秋戬, 吴斌, 等. Rough集高效算法的研究[J]. 计算机学报, 2003, 26(5): 524-529.
(Liu S H, Sheng Q J, Wu B, et al. Research on efficient algorithms for rough set methods[J]. Chinese J of Computers, 2003, 26(5): 524-529.)

[10] Guan J W, Bell D A. Rough computational methods for information systems[J]. Artificial Intelligences, 1998, 105(1/2): 77-103.

[11] 苗夺谦, 胡桂荣. 知识约简的一中启发式算法[J]. 计算机研究与发展, 1998, 36(6): 681-684.
(Miao D Q, Hu G R. A heuristic algorithm for reduction of knowledge[J]. J of Computer Research & Development, 1999, 36(6): 681-684.)

[12] 杨明. 决策表中基于条件信息熵的近似约简[J]. 电子学报, 2007, 35(11): 2156-2160.
(Ying M. Approximate reduction based on conditional information entropy in decision tables[J]. Acta Electronica Sinica, 2007, 35(11): 2156-2160.)

[13] Yang Ming, Yang Ping. A novel condensing tree structure for rough set feature selection[J]. Neurocomputing, 2008, 71(4/5/6): 1092-1100.

[14] Yang Ming, Yang Ping. A novel approach to improving C-Tree for feature selection[J]. Applied Soft Computing, 2010, 11(2): 1924-1931.

[15] 刘宗田. 属性最小约简的增量式算法[J]. 电子学报, 1999, 27(11): 96-98.
(Liu Z T. An incremental arithmetic for the smallest reduction of attributes[J]. Acta Electronica Sinica, 1999, 27(11): 96-98.)

[16] Orlowska M, Orlowski M. Maintenance of knowledge in dynamic information systems[C]. Intelligent Decision Support. Dordrecht: Kluwer Academic Publishers, 1992: 315-330.

[17] 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 the improved discernibility matrix[J]. Chinese J of Computers, 2007, 30(5): 815-822.)

[18] Hu F, Wang G Y, Huang H, et al. Incremental attribute reduction based on elementary sets[C]. Proc of the 10th Int Conf on Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing. Regina, 2005: 185-193.

(上接第1768页)

[15] 毕华, 梁洪力. 重采样方法与机器学习[J]. 计算机学报, 2009, 32(5): 862-877.
(Bi H, Liang H L. Resampling methods and machine learning[J]. Chinese J of Computers, 2009, 32(5): 862-877.)

[16] Liu Y, Yu X H. Combining integrated sampling with SVM ensembles for learning from imbalanced datasets[J]. Information Processing & Management, 2010, 47(4): 617-673.

[17] 王玲, 薄列峰, 焦李成. 密度敏感的半监督谱聚类[J]. 软件学报, 2007, 18(9): 2412-2422.
(Wang L, Bo L F, Jiao L C. Density-sensitive semi-supervised spectral clustering[J]. J of Software, 2007, 18(9): 2412-2422.)