Knowledge-Based Systems 24 (2011) 275–281

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

A rough set approach to feature selection based on power set tree

Yumin Chen [a,*], Duoqian Miao [b], Ruizhi Wang [b], Keshou Wu [a]

[a] Department of Computer Science and Technology, Xiamen University of Technology, 361024 Xiamen, PR China
[b] Department of Computer Science and Technology, Tongji University, 201804 Shanghai, PR China

ARTICLE INFO

Article history:
Received 8 January 2009
Received in revised form 22 July 2010
Accepted 20 September 2010
Available online 25 September 2010

Keywords:
Rough sets
Feature selection
Data mining
PS-tree
Reduction

ABSTRACT

Feature selection is viewed as an important preprocessing step for pattern recognition, machine learning and data mining. Traditional hill-climbing search approaches to feature selection have difficulties to find optimal reducts. And the current stochastic search strategies, such as GA, ACO and PSO, provide a more robust solution but at the expense of increased computational effort. It is necessary to investigate fast and effective search algorithms. Rough set theory provides a mathematical tool to discover data dependencies and reduce the number of features contained in a dataset by purely structural methods. In this paper, we define a structure called power set tree (PS-tree), which is an order tree representing the power set, and each possible reduct is mapped to a node of the tree. Then, we present a rough set approach to feature selection based on PS-tree. Two kinds of pruning rules for PS-tree are given. And two novel feature selection algorithms based on PS-tree are also given. Experiment results demonstrate that our algorithms are effective and efficient.

1. Introduction

Feature selection can be viewed as one of the most fundamental problems in the fields of pattern recognition, machine learning and data mining. The main aim of feature selection is to determine a minimal feature subset from a problem domain while retaining a suitably high accuracy in representing the original features [2]. In real world problems, feature selection is a must due to the abundance of noisy, irrelevant or misleading features [9]. By removing these factors, learning from data techniques can benefit greatly. As Liu pointed out in [11], the motivation of feature selection in data mining and machine learning is to reduce the dimensionality of feature space, improve the predictive accuracy of a classification algorithm, and improve the visualization and the comprehensibility of the induced concepts.

Rough set theory (RST), proposed by Pawlak in 1982 [16], is a mathematical tool to handle imprecision, uncertainty and vagueness. It has been widely applied in many fields such as machine learning [20], data mining [4], etc [12]. Rough set theory provides a mathematical tool to discover data dependencies and reduce the number of features contained in a dataset by purely structural methods.

Many rough set algorithms for feature selection have been proposed. The complete solution to find minimal reducts is to generate all possible reducts and choose one with minimal cardinality, which can be done by constructing a kind of discernibility function from the dataset and simplifying it [10]. Starzyk et al. proposed a strong equivalence to simplify discernibility functions [18]. Since it has been shown that the problem of minimal reduct generation is NP-hard and the problem of the generation of all reducts is exponential, many incomplete solutions have been proposed. The incomplete solutions include heuristic methods and stochastic methods. The heuristic methods employ an incremental hill-climbing (greedy) algorithm to select features [1,5,6,13,14,19,23]. The hill-climbing algorithms usually employ feature significance as heuristics. For instance, Hu and Cercone proposed a reduction algorithm using the positive-region-based feature significance as the guiding heuristic [5]. Susmaga focused on the topological aspects of rough set reducts, giving a discernibility method in reduction construct [19]. Miao and Hou developed a mutual-information-based reduction algorithm, using entropy-based feature significance [14]. Inuiguchi et al. proposed a variable-precision dominance-based rough set reduction approach [6]. However, hill-climbing methods often lead to a non-minimal feature combination, so there can be no guarantee of optimality.

Therefore, many researchers have shifted to stochastic methods for rough set feature selection. Wroblewski used genetic algorithms (GA) to find minimal reducts [24]. He combined a genetic algorithm with a greedy algorithm to generate short reducts. However, Wroblewski's method uses time-consuming operations and cannot assure that the resulting subset is really a reduct. ElAlami utilizes a genetic algorithm to find the optimal relevant features [3]. Zhai et al. proposed an integrated feature extraction approach

* Corresponding author at: Department of Computer Science and Technology, Xiamen University of Technology, 361024 Xiamen, PR China. Tel.: +86 0592 6291390.
E-mail address: cym0620@163.com (Y. Chen).

0950-7051/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2010.09.004

based on rough set theory and genetic algorithms [26]. Jensen and Shen proposed a method to find rough set reducts using ant colony optimization (ACO) [7,8]. Wang et al. developed a feature selection method based on rough sets and particle swarm optimization (PSO) [22]. Stochastic methods can provide a more robust solution as their global optimizing, but at the expense of increased computational effort.

In the paper, we define a novel tree structure called power set tree, which is an order tree representing the power set, and each possible reduct is mapped to a node in the PS-tree. We give two kinds of PS-tree-based rules for pruning unpromising parts of the search space. Two novel feature selection algorithms based on PS-tree are also given. One is a complete algorithm which can guarantee to find the minimal reduct. The other is a heuristic algorithm based on PS-tree. The performance of the first algorithm will be compared with that of the strong equivalence method. The performance of the second algorithm will be compared with that of traditional hill-climbing algorithms and stochastic algorithms.

The paper is organized as follows. In Section 2, we introduce some preliminaries in rough sets. Section 3 gives a formal definition of the PS-tree. Section 4 discusses PS-tree-based search strategies. Section 5 presents two kinds of pruning rules and two feature selection algorithms based on PS-tree. In Section 6, the search process is demonstrated by an example. Experimental results are given in Sections 7, and 8 concludes the paper.

2. Preliminary

This section recalls some preliminaries of RST that are relevant to this paper. Detailed description of the theory can be found in [16].

2.1. Preliminary of RST

The notion of information table has been studied by many authors as a simple knowledge representation method. Formally, an information table is a quadruple $\mathcal{I} = (U, A, V, f)$, where U is a nonempty finite set of objects, A is a nonempty finite set of features, V is the union of feature domains such that $V = \bigcup_{a \in A} V_a$ for V_a denotes the value domain of feature a, and f: $U \times A \to V$ is an information function which associates a unique value of each attribute with every object belonging to U, such that for any $a \in A$ and $x \in U$, $f(x,a) \in V_a$.

With any $B \subseteq A$, there is an associated indiscernibility relation $IND(B)$:

$$IND(B) = \{(x,y) \in U \times U | \forall a \in B, f(x,a) = f(y,a)\}. \quad (1)$$

The partition of U, induced by $IND(B)$ is denoted by $U/IND(B)$ and can be calculated as follows:

$$U/IND(B) = \otimes\{a \in B : U/IND(\{a\})\}, \quad (2)$$

where

$$R \otimes S = \{X \bigcap Y : \ \forall R \in X, \ \forall S \in Y, X \bigcap Y \neq \varnothing\}. \quad (3)$$

Given an information table $\mathcal{I} = (U, A, V, f)$, for any subset $X \subseteq U$ and equivalence relation $IND(B)$, the *B-lower* and *B-upper* approximations of X are defined, respectively, as follows:

$$B_*(X) = \{x \in U : [x]_B \subseteq X\}, \quad (4)$$

$$B^*(X) = \{x \in U : [x]_B \bigcap X \neq \phi\}. \quad (5)$$

Given any two subsets $P, Q \subset A$, which give rise to two equivalence relations $IND(P)$ and $IND(Q)$, the *P-positive*, *P-negative* and *P-boundary* regions of Q can be defined, respectively, as

$$POS_P(Q) = \bigcup_{X \in U/IND(Q)} P_*(X), \quad (6)$$

$$NEG_P(Q) = U - \bigcup_{X \in U/IND(Q)} P^*(X), \quad (7)$$

$$BND_P(Q) = \bigcup_{X \in U/IND(Q)} P^*(X) - \bigcup_{X \in U/IND(Q)} P_*(X). \quad (8)$$

An important issue in data analysis is discovering dependencies between features. Dependency can be defined in the following way. For any P, $Q \subset A$, P depends totally on Q, if and only if $IND(P) \subseteq IND(Q)$. That means that the partition generated by P is finer than the partition generated by Q. We say that Q depends on P in a degree $\mu_P(Q)$ $(0 \leqslant \mu_P(Q) \leqslant 1)$, if

$$\mu_P(Q) = |POS_P(Q)|/|U|. \quad (9)$$

If $\mu_P(Q) = 1$, then Q depends totally on P; if $0 < \mu_P(Q) < 1$, then Q depends partially on P; and if $\mu_P(Q) = 0$ then Q does not depend on P. Dependency degree $\mu_P(Q)$ can be used as heuristics in greedy algorithms to compute feature reduction.

Information table $\mathcal{I} = (U, A, V, f)$ is also called a decision table if $A = C \bigcup D$, and $C \bigcap D = \emptyset$, where C is the set of condition features, D is the set of decision features. The degree of dependency between condition and decision features, $\mu_C(D)$, is called the quality of classification.

The goal of feature reduction is to remove redundant features so that the reduced set provides the same quality of classification as the original. A reduct is defined as a subset R of the condition feature set C such that $\mu_R(D) = \mu_C(D)$ and $\forall B \subset R$, $\mu_B(D) \neq \mu_C(D)$. A subset $R_{sup} \subseteq C$ is called a super reduct, if $\mu_{R_{sup}}(D) = \mu_C(D)$. A given decision table may have many reducts, the set of all reducts is defined as

$$Red = \{R \subseteq C | \mu_R(D) = \mu_C(D), \ \forall B \subset R, \mu_B(D) \neq \mu_C(D)\}. \quad (10)$$

In rough set feature reduction, a reduct with minimal cardinality is called the minimal reduct, which can be defined as follows

$$R_{min} = \{R \in Red | \forall R' \in Red, |R| \leqslant |R'|\}. \quad (11)$$

2.2. Hill-climbing methods to feature selection

The hill-climbing methods usually employ dependency degree as heuristics. They start with the full feature set or an empty set and then adopt deletion or addition algorithms. The deletion and addition algorithms can be found in [1,5].

Given a decision table $\mathcal{DT} = (U, C \bigcup D, V, f)$, C and D are the condition feature set and decision feature set, respectively. A deletion algorithm starts with the full feature set. It is not efficient if the feature reduct is short since many features have been checked and eliminated.

Deletion algorithm for feature selection
Input: a decision table $\mathcal{DT} = (U, C \bigcup D, V, f)$
Output: a feature reduct R

(1) Let $R = C$, $Del = C$
(2) While $Del \neq \emptyset$ do
  (2.1) Select a feature $a \in Del$ according to heuristics, let $Del = Del - \{a\}$
  (2.2) If $\mu_{R-\{a\}}(D) = \mu_R(D)$, then $R = R - \{a\}$, else go to (3)
(3) Output R

The addition algorithm constructs a feature reduct from an empty set or the core, and adds features to the set until it becomes a feature reduct. In some cases, the algorithm cannot find a feature reduct that satisfies the strict reduction definition in Section 2.1, since the feature subset obtained may contain irrelevant features.

Addition algorithm for feature selection
Input: a decision table $\mathcal{DT} = (U, C \bigcup D, V, f)$
Output: a feature reduct R

(1) Let $R = \phi$, $Add = C$
(2) While $\mu_R(D) \neq \mu_C(D)$ and $Add \neq \phi$ do
  (2.1) Select a feature $a \in Add$ according to heuristics, let $Add = Add - \{a\}$
  (2.2) If $\mu_{R\cup\{a\}}(D) \neq \mu_C(D)$, then $R = R \cup \{a\}$, else go to (3)
(3) Output R

The order of features for deletion and addition is essential for feature reduction. Different heuristics may lead to different orders of features, and thus generate different reducts. The heuristics adopted in the current feature selection methods mainly include dependency-degree-based heuristics [5,15], entropy-based heuristics [14,25] and frequency-based heuristics [17,21].

3. PS-tree

Trees provide us an efficient way to solve many problems. The power set tree (PS-tree) is a tree structure to represent the power set in an order fashion. Since the PS-tree completely enumerates the subsets of a power set using a particular order, it can represent the search space of a particular feature selection problem.

Let $\mathcal{DT} = (U, C \bigcup D, V, f)$ be a decision table, where $C = \{c_1, \ldots, c_n\}$, P is the power set of C. PT is a PS-tree for $\mathcal{DT}$, which is a tree that satisfies the following properties:

(1) The root of PT is a tuple $\langle c_1, \ldots, c_n, n\rangle$, which means that the root has n elements $c_1, \ldots, c_n$ and n children.
(2) Let $PT_1, PT_2, \ldots, PT_n$ be the n children of root PT. PT_1 has $n-1$ elements and $n-1$ children; PT_2 has $n-1$ elements and $n-2$ children; …; PT_n has $n-1$ elements and zero children. Elements of PT_1 are inherited from PT by deleting the 1st element of PT; elements of PT_2 are inherited from PT by deleting the 2nd element of PT; …; elements of PT_n are inherited from PT by deleting the n element of PT. Each child of the root is also a PS-tree.
(3) The order of elements in PT_1 is unmodified. The orders of elements in $PT_2, \ldots, PT_n$ must be modified. The last element of PT_2 equals the deleted element of PT_1, while the order of other elements in PT_2 is the same as that of PT. The last two elements in PT_3 equal the deleted elements of PT_2 and PT_1, while the order of other elements in PT_3 is the same as that of PT. Similarly, the $n-1$ elements in PT_n equal the deleted elements of $PT_{n-1}, \ldots, PT_2, PT_1$.

Fig. 1 illustrates the PS-tree of set $\{a,b,c,d\}$. Expanded nodes enumerate all the subsets of $\{a,b,c,d\}$. Each node represents a possible solution of feature selection. For convenience, we label the deleted feature during the expanding process with letter on the top of tuple, and label the number of children for each node in the right of tuple.

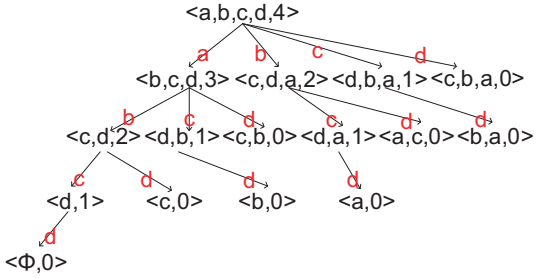

Fig. 1. The PS-tree for $\{a,b,c,d\}$.

The PS-tree can be used as a data structure for caching unordered sets, and as an effective means for checking whether a new set is subsumed by any one that is already cached. In our feature selection algorithms, we shall use the PS-tree for caching solutions and for checking the subsumption relation between two sets, which can efficiently reduce the time and space complexities of the algorithms.

4. PS-tree-based feature selection algorithm

Next, we give a PS-tree-based feature selection algorithm (also called PS-FS). The algorithm expands nodes according to some priority functions in an order fashion. Nodes along the tree's expanding fringe are kept in a priority queue and the next node to be expanded is obtained by virtue of the guiding heuristic.

Algorithm PS-FS
Input: a decision table $\mathcal{DT} = (U, C \bigcup D, V, f)$
Output: a feature reduct R

(1) Let $R = C$; Add C to *OPEN_NODES*; *feat_num* = $|C|$; *child* = $|C|$
(2) Repeat the following until *OPEN_NODES* is empty
  (2.1) Select the first node *NODE* in *OPEN_NODES*
  (2.2) Expand (*NODE*, *feat_num*, *child*)
  (2.3) Delete the node *NODE* from *OPEN_NODES*
  (2.4) Sort all the nodes in *OPEN_NODES* according to the heuristic
(3) Output R

Procedure Expand (S, S.feat_num, S.child)
(1) If $\mu_S(D) = \mu_C(D)$ and $\forall B \subset S$, $\mu_B(D) \neq \mu_C(D)$, then
  (1.1) If $|S| < |R|$, then $R = S$ and return
  (1.2) If $|S| \geqslant |R|$, then return
(2) If $\mu_S(D) = \mu_C(D)$, then
  (2.1) For $i = 1$ to *S.child* do
    (2.1.1) Select a feature $a_i \in S$ from sequence S, let $S_i = S - \{a_i\}$
    (2.1.2) S_i.*feat_num* = *S.feat_num* − 1
    (2.1.3) S_i.*child* = *S.child* − i
    (2.1.4) Modify the order of elements in S_i according to the definition of PS-tree
    (2.1.5) Add S_i to *OPEN_NODES*

The above algorithm simply implements a best-first search through sorting nodes by the heuristic. We can also adopt other expanding strategies, such as breadth-first search or depth-first search. Breadth-first search can be implemented by calling the expand procedure with a queuing function which puts the newly generated nodes at the end of the queue. And depth-first search can be implemented by calling the expand procedure with a queuing function which puts the newly generated states at the front of the queue.

Theorem 1. *If nodes are sorted by their cardinality in descending, then the PS-FS is complete. If nodes are sorted by their cardinality in ascending, then the PS-FS is also complete. Completeness means that we can guarantee to find all reducts or a minimal reduct.*

Proof

(1) In algorithm PS-FS, if nodes are sorted by their cardinality in descending, then it means that we adopt the breadth-first search strategy. It is well known that breadth-first search is complete. Hence, algorithm PS-FS is also complete.

(2) If nodes are sorted by their cardinality in ascending, then it means that we adopt the depth-first search strategy. Depth-first search is not complete generally. But in algorithm PS-FS, since all the possible solutions are represented by the PS-tree, and all the nodes in the PS-tree will be accessed, we can guarantee to find all reducts. Minimal reduct is a reduct that its cardinality is minimal. So, we can also find a minimal reduct from all reducts. Thus algorithm PS-FS is complete. □

5. Pruning rules

Due to its exponential size, it is impossible to completely explore the PS-tree. The rules of eliminating a branch of the search tree from consideration without examining the nodes in the branch are called pruning rules. The particular PS-tree-based pruning rules are called rotation and backtracking.

5.1. Rotation

The PS-tree is not a balancing tree as its left branches are larger than the right branches. Therefore, it is more efficient to prune the left branches. If a node is not a super reduct, then its children certainly are not reducts. By virtue of this property of PS-tree, we can prune unpromising parts of the search space. When an expanded node in the right of PS-tree is not a super reduct, it must be rotated to the left and pruned as a large branch. Simultaneously, the order of elements in the node should be modified according to the definition of PS-tree.

Fig. 2 illustrates the rotation pruning. Node $\{a,b,c\}$ is a leaf in the most right part of the left PS-tree, which is a small branch. Suppose $\{a,b,c\}$ is not a super reduct. Therefore, we can rotate it to the most left as a larger branch and prune it, which can improve the search efficiency greatly.

To add rotation pruning rule into feature reduction, we modify the previous PS-FS algorithm, and obtain a new algorithm, called PS-FS-I. PS-FS is a complete algorithm, which is an expensive method to search all the reducts and is only practical for small data sets. Therefore, in PS-FS-I algorithm we use the depth-first search strategy as adding rotation pruning and stop the search process when a reduct is found. The PS-FS-I algorithm is incomplete for finding minimal reduct, but can find the best solution by the sorting operation. The sorting operation can use heuristics to select the best solution.

Algorithm PS-FS-I
Input: a decision table $\mathcal{DT} = (U, C \bigcup D, V, f)$
Output: a feature reduct R

(1) Let $R = C$; add C to *OPEN_NODES*; let *feat_num* = $|C|$, *child* = $|C|$ and *Isreduct* = *false*
(2) Repeat the following until *Isreduct* = *true*
  (2.1) Select the first node *NODE* in *OPEN_NODES*
  (2.2) Expand (*NODE*, *feat_num*, *child*)
  (2.3) Delete the node *NODE* from *OPEN_NODES*
  (2.4) Sort all the nodes in *OPEN_NODES* according to the heuristic
(3) Output R

Procedure Expand (S, S.feat_num, S.child)
(1) If $\mu_S(D) = \mu_C(D)$ and $\forall B \subset S$, $\mu_B(D) \neq \mu_C(D)$, then let $R = S$, *Isreduct* = *ture* and return
(2) If $\mu_S(D) = \mu_C(D)$, then
  (2.1) For $i = 1$ to *S.child*
    (2.1.1) Select a feature $a_i \in S$ in sequence S, let $S_i = S - \{a_i\}$
    (2.1.2) Let S_i.*feat_num* = *S.feat_num* − 1
    (2.1.3) Let S_i.*child* = *S.child* − i
    (2.1.4) If $\mu_{S_i}(D) \neq \mu_C(D)$, then rotate S_i to the left of PS-tree to be pruned
    (2.1.5) Else rotate S_i to the right of PS-tree
  (2.2) Let $M_1, M_2, \ldots, M_m, M_{m+1}, \ldots, M_{child}$ be the nodes after rotation, where $M_1, M_2, \ldots, M_m$ are the pruned parts, and $M_{m+1}, \ldots, M_{child}$ are the non-pruned parts
  (2.3) Modify the order of elements in nodes of non-pruned parts according to the definition of PS-tree
  (2.4) For $j = m + 1$ to *S.child*
    (2.4.1) Add M_j to *OPEN_NODES*

The above algorithm is identical to PS-FS, except for the additions of rotation and the stop condition. The addition of rotation will improve the efficiency of search process. The PS-FS-I is incomplete for minimal reduct, as its stop condition only finding one reduct.

5.2. Backtracking

The PS-tree is an organization of the solution space so that it can be searched easily through either rotation method or backtracking method. Backtracking is a systematic way to search for the solution to a given problem. Once we have defined an organization for the solution space, this space is searched in a depth-first manner beginning at a start node. If the search process constructs a solution by depth-first manner, then backtracks to search for a more optimal solution. For example, in feature reduction problem, if we find a reduct by depth-first manner and regard it as the minimal reduct temporarily, then backtrack to its father's right brother and deem it as the current expansion node. Let r be the length of the temporary minimal reduct, t be the length of the expansion node and s be the number of children of the expansion node. If $t - s \geqslant r$, then prune the expansion node and back up to its father's right brother. The search terminates when we run out of live nodes to back up to.

Fig. 3 illustrates the backtracking. Suppose we have found node $\{b,a\}$ is a reduct by depth-first search and rotation pruning. We can

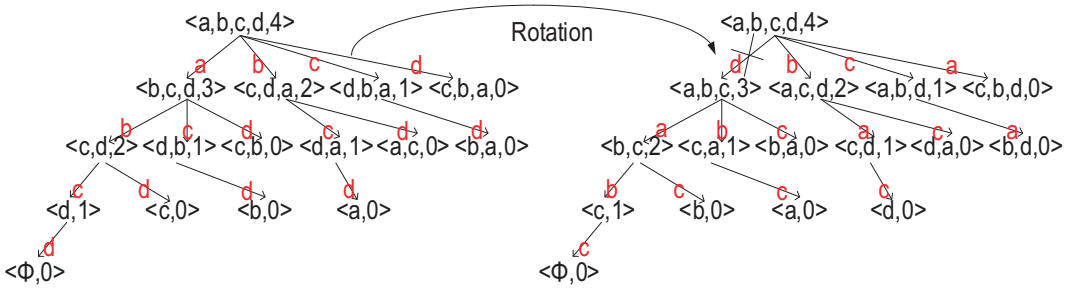

Fig. 2. Rotation.

regard node $\{b,a\}$ as the minimal reduct temporarily and backtracking to its father's right brother $\{a,c,d\}$. Node $\{a,c,d\}$ is labeled with two children (expand two children or two levels). If we expand two levels, the length of posterity of node $\{a,c,d\}$ will be 1, which is smaller than that of current minimal reduct. Therefore, node $\{a,c,d\}$ must be expanded. Consider its father's other two right brother $\{a,b,d\}$ and $\{c,b,d\}$. Node $\{a,b,d\}$ is labeled with one child (expand one child or one level). If we expand one level, the length of its child will be 2, which equals that of the current minimal reduct. So, node $\{a,b,d\}$ must be pruned. Node $\{c,b,d\}$ must also be pruned as it is a leaf and its length is more than that of the current minimal reduct.

Rotation method can prune left branches of the PS-tree. And backtracking method can prune the right branches of the PS-tree. Through rotation and backtracking, we can ignore those parts of the solution space which do not have the potential to lead to a solution.

To incorporate rotation and backtracking into feature reduction, we modify the expand procedure of PS-FS algorithm, and obtain a new algorithm, called PS-FS-C.

Algorithm PS-FS-C

Input: a decision table $\mathcal{DT} = (U, C \bigcup D, V, f)$

Output: a feature reduct R

(1) Let $R = C$; Add C to *OPEN_NODES*; Let *feat_num* = $|C|$, *child* = $|C|$
(2) Repeat the following until *OPEN_NODES* is empty
  (2.1) Select the first node *NODE* in *OPEN_NODES*
  (2.2) Expand (*NODE*, *feat_num*, *child*)
  (2.3) Delete the node *NODE* from *OPEN_NODES*
  (2.4) Sort all the nodes in *OPEN_NODES* according to the heuristic
(3) Output R

Procedure Expand (S, S.feat_num, S.child)

(1) If $\mu_S(D) = \mu_C(D)$ and $\forall B \subset S$, $\mu_B(D) \neq \mu_C(D)$, then
  (1.1) If $|S| < |R|$, then let $R = S$ and return
  (1.2) If $|S| \geqslant |R|$, then return
(2) If $\mu_S(D) = \mu_C(D)$, then
  (2.1) For $i = 1$–*S.child*
    (2.1.1) Select a feature $a_i \in S$ in sequence S, let $S_i = S - \{a_i\}$
    (2.1.2) Let S_i.*feat_num* = *S.feat_num* − 1
    (2.1.3) Let S_i.*child* = *S.child* − i
    (2.1.4) If $\mu_{S_i}(D) \neq \mu_C(D)$, then rotate S_i to the left of PS-tree to be pruned
    (2.1.5) Else rotate S_i to the right of PS-tree
  (2.2) Let $M_1, M_2, \ldots, M_m, M_{m+1}, \ldots, M_{child}$ be the nodes after rotation, where $M_1, M_2, \ldots, M_m$ are the pruned parts, and $M_{m+1}, \ldots, M_{child}$ are the non-pruned parts
  (2.3) Modify the order of elements in nodes of non-pruned parts according to the definition of PS-tree
  (2.4) For $j = m + 1$ to *S.child*
    (2.4.1) If M_j.*feat_num* − M_j.*child* < $|R|$, then add M_j to *OPEN_NODES*

The above algorithm is identical to PS-FS, except for the additions of rotation and backtracking. These changes will improve the efficiency of search process. The PS-FS-C is complete for minimal reduct, since the PS-FS is complete.

6. An example

Let $\mathcal{DT} = (U, C \bigcup D, V, f)$ be a decision table and $C = \{a,b,c,d,e\}$ be the condition feature set. Assume that all reducts are $\{b,c,e\}$ and $\{a,e\}$. The justification for a reduct is the definition of dependency degree. If a feature set B includes the reduct $\{b,c,e\}$ or $\{a,e\}$, then $\mu_B(D) = \mu_C(D)$. If a feature set B does not include the reducts $\{b,c,e\}$ and $\{a,e\}$, then $\mu_B(D) \neq \mu_C(D)$. So, we can use its including for justification.

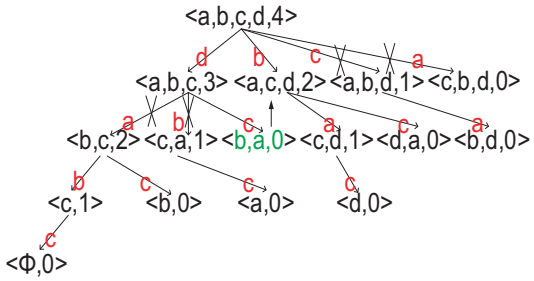

Fig. 3. Backtracking.

Initially, the minimal reduct $R = \{a,b,c,d,e\}$ and its length is 5. The following Figs. 4–7, respectively illustrate the search process. The start node is $\{a,b,c,d,e\}$.

We expand the start node by deleting a feature in sequence as illustrated in Fig. 4. The character on the top is the deleted feature of the node. The number in the right is the number of children of the node.

The last expanded node is $\{a,b,c,d\}$. We must prune it as its posterity is not a reduct. However, it is in the most right of the PS-tree. We can rotate it to the most left and then prune it. To guarantee the structure of PS-tree, inner order of nodes should be modified. The rotation pruning is illustrated in Fig. 5.

The other four children of the root include reducts, so each of them should be expanded. The node $\{b,c,d,e\}$ is expanded with three sub-nodes by deleting a feature in the sequence. Two children of node $\{b,c,d,e\}$ do not include reducts, so we prune them. The remainder child $\{e,c,b\}$ is a reduct. Moreover, its length is smaller than that of previous minimal reduct. Therefore, we update the minimal reduct $R = \{e,c,b\}$ and its length equals 3. The search process is illustrated in Fig. 6.

After finding a reduct by depth-first search and rotation pruning, we backtrack to its father's right brother $\{c,d,a,e\}$. Node

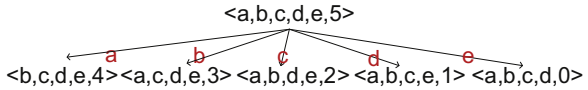

Fig. 4. Expand the root by deleting a feature in sequence.

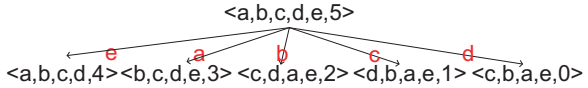

Fig. 5. Rotating node $\{a,b,c,d\}$ to the left.

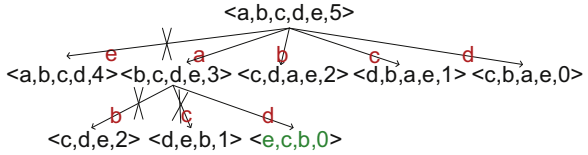

Fig. 6. Finding a reduct by depth-first search.

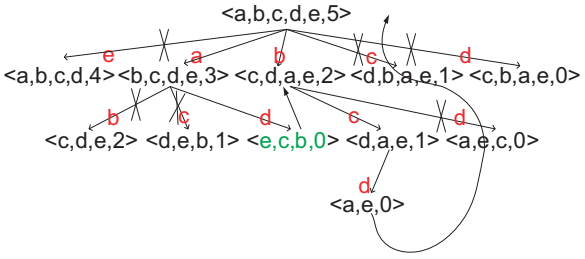

Fig. 7. The backtracking process.

$\{c,d,a,e\}$ is labeled with two children (expand two children or two levels). If we expand two levels, the length of posterity of node $\{c,d,a,e\}$ will be 2, which is smaller than that of the current minimal reduct. So node $\{c,d,a,e\}$ must be expanded. Consider its father's other two right brother $\{d,b,a,e\}$ and $\{c,b,a,e\}$. Node $\{d,b,a,e\}$ is labeled with one child (expand one child or one level). If we expand one level, the length of its child will be 3, which equals that of the current minimal reduct. So, node $\{d,b,a,e\}$ must be pruned. Node $\{c,b,a,e\}$ must also be pruned as its length is more than that of the current minimal reduct.

Next we search from the node $\{c,d,a,e\}$. It can be expanded two levels in a depth-first manner, and node $\{a,e\}$ is reached. Now, node $\{a,e\}$ is a reduct. Its length is 2, which is smaller than that of the current minimal reduct. Therefore, we regard node $\{a,e\}$ as the current minimal reduct and update the length of minimal reduct as 2. From the node $\{a,e\}$ we backtrack to its father's right brother $\{a,e,c\}$. But the length of node $\{a,e,c\}$ is more than that of the current minimal reduct. So, it must be pruned. From the node $\{a,e,c\}$ we backtrack to $\{d,b,a,e\}$, which is also pruned. Continuing in this way, we search the entire PS-tree. The best solution is found during the search is the optimal one. In the end, we find the minimal reduct, which is $\{a,e\}$. Fig. 7 shows the backtracking process.

7. Experimental results

Our method to feature reduction has two search strategies based on PS-tree, which induce two algorithms PS-FS-C and PS-FS-I. PS-FS-C can guarantee to find a minimal reduct, but the computation time is expensive. PS-FS-I can find a reduct quickly, but cannot guarantee to find a minimal reduct. Next, we shall compare our algorithms with the current algorithms to feature selection.

7.1. Comparison with the complete algorithm

Starzyk's strong equivalence algorithm [18] is complete, which can obtain all the reducts and then guarantee to find a minimal reduct. We have implemented a program in Visual C++ 6.0 to compare the efficiency of algorithm PS-FS-C with strong equivalence methods. A random number generator is used to provide uniformly distributed numbers to represent each feature of each instance. These values are multiplied by 8 and then the fractional parts are truncated. This results that all the feature values are integers between 0 and 8. The number of instances varies from 25 to 40 in steps of 5.

Fig. 8 illustrates how the run times increase with problem size using the strong equivalence algorithm. Note the abscissa is log10 of the run time. From Fig. 8 we can see that the computational time of strong equivalence algorithm is growing exponentially.

Fig. 9 shows the results using PS-FS-C algorithm. The number of features varies from 60 to 120 in steps of 20 and the number of instances varies from 60 to 120 in steps of 20.

From the above two figures, we can see that PS-FS-C algorithm can compute 120 features, while strong equivalence algorithm can compute only 40 features. Both PS-FS-C and strong equivalence algorithm are growing exponentially in the computational time. PS-FS-C algorithm is growing exponentially in step of 20 features, while strong equivalence algorithm is growing exponentially in step of 5 features.

7.2. Comparison with the incomplete algorithms

Hill-climbing algorithm is efficient but often finds a local solution. Ant colony optimization (ACO) is an effective global optimizing technique, which is a stochastic search method. ACO-based algorithm for feature selection can find more robust solution. We compare our algorithm PS-FS-I with the hill-climbing algorithm and ACO-based algorithm [7].

In order to compare PS-FS-I with the hill-climbing algorithm and ACO-based algorithm, we perform the experiments on nine publicly available data sets from UCI database (These data sets can be downloaded at http://www.ics.uci.edu).

The experimental results are summarized in Table 1. The leftmost column is the name of each data set. The 2nd and 3rd columns are instance numbers and feature numbers of the corresponding data set. The 4th, 6th, and 8th columns are the results of hill-climbing, ACO, and PS-FS-I algorithms, respectively. These results are the length of the reduct obtained by each algorithm. The 5th, 7th, and 9th columns are the results of hill-climbing, ACO, and PS-FS-I algorithms in run time, respectively. ACO algorithm is tested for 20 times. Therefore, its run time is the average time. For ACO algorithm, the number in parentheses denotes the times of test to achieve such a feature reduct.

From Table 1, we can see that ACO outperforms the hill-climbing with respect to the ability of finding optimal reducts, but for the expensive of computational time. And PS-FS-I outperforms the hill-climbing and ACO as respect to the ability of finding optimal reducts.

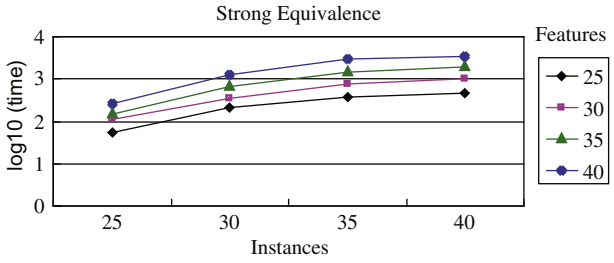

Fig. 8. Strong equivalence algorithm.

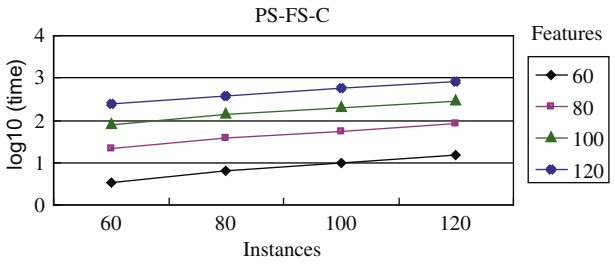

Fig. 9. PS-FS-C Algorithm.

Table 1
Experimental results compared with ACO-based algorithm.

Dataset	Instance	Feature	Hill-climbing	Run time (s)	ACO	Run time (s)	PS-FS-I	Run time (s)
Balance	200	5	4	<0.001	4	0.016	4	0.003
DNA-stalog	2000	61	10	21.703	10	134.203	10	102.325
Led24	200	25	12	0.094	$12^{(13)}13^{(7)}$	0.624	11	0.265
Lung	32	57	5	<0.001	$4^{(5)}5^{(10)}6^{(5)}$	0.421	4	0.109
Mushroom	8124	23	5	0.657	$4^{(13)}5^{(7)}$	23.391	4	15.236
Soy	47	36	2	<0.001	2	0.032	2	0.016
Vote	435	17	12	0.068	$10^{5}11^{7}12^{8}$	1.798	9	0.389
Wine	178	14	5	<0.001	$5^{(17)}6^{(3)}$	0.088	5	0.064
Zoo	101	17	6	<0.001	$5^{(18)}6^{(2)}$	0.089	5	0.016

8. Conclusions

This paper discussed the shortcomings of the conventional hill-climbing approaches to feature selection. These techniques often fail to find minimal reducts. On the other hand, stochastic methods can provide a more robust solution, but at the expense of increased computational effort. To solve these problems, we defined a novel structure called PS-tree which is efficient in finding minimal reducts. Based on the PS-tree, a complete algorithm for the generation of minimal reduction was proposed. In order to reduce the run time of complete algorithm, an incomplete algorithm was also given. Experimental results on synthetic and real data sets demonstrated the efficiency of our method to feature selection.

Acknowledgement

This work was supported by the National Natural Science Foundation of China under Grant Nos: 60903203, 60802042, 60970061.

References

[1] J.G. Bazan, H.S. Nguyen, et al., Rough set algorithms in classification problem, in: Rough Set Methods and Applications, Physica-Verlag GmbH Heidelberg, Germany, 2000, pp. 49–88.
[2] M. Dash, H. Liu, Feature selection for classification, Intelligent Data Analysis 1 (3) (1997) 131–156.
[3] M.E. ElAlami, A filter model for feature subset selection based on genetic algorithm, Knowledge-Based Systems 22 (5) (2009) 356–362.
[4] Q.L. Guo, M. Zhang, Implement web learning environment based on data mining, Knowledge-Based Systems 22 (6) (2009) 439–442.
[5] X. Hu, N. Cercone, Learning in relational databases: a rough set approach, Computation Intelligence: An International Journal 11 (1995) 323–338.
[6] M. Inuiguchi, Y. Yoshioka, Y. Kusunoki, Variable-precision dominance-based rough set approach and attribute reduction, International Journal of Approximate Reasoning 50 (2009) 1199–1214.
[7] R. Jensen, Q. Shen, Finding rough set reducts with ant colony optimization, in: Proceeding of 2003 UK Workshop Computational Intelligence, 2003, pp. 15–22.
[8] R. Jensen, Q. Shen, Semantics-preserve dimensionality reduction: rough and fuzzy-rough-based approaches, IEEE Transactions on Knowledge and Data Engineering 16 (2004) 1457–1471.
[9] R. Jensen, Combining rough and fuzzy sets for feature selection, Ph.D. Thesis, University of Edinburgh, 2005.
[10] J. Komorowski, Z. Pawlak, L. Polkowski, A. Skowron, Rough sets: a tutorial, in: Rough Fuzzy Hybridization, A New Trend in Decision-Making, Springer-Verlag Singapore Pte., Ltd., Singapore, 1999.
[11] H. Liu, H. Motoda, Feature Selection for Knowledge Discovery and Data Mining, Kluwer, Boston, 1998.
[12] J.S. Mi, W.Z. Wu, W.X. Zhang, Approaches to knowledge reduction based on variable precision rough set model, Information Sciences 159 (3–4) (2004) 255–272.
[13] J.S. Mi, Y. Leung, W.Z. Wu, Approaches to attribute reduction in concept lattices induced by axialities, Knowledge-Based Systems 23 (6) (2010) 504–511.
[14] D.Q. Miao, L. Hou, A comparison of rough set methods and representative inductive learning algorithms, Fundamenta Informaticae 59 (2–3) (2004) 203–219.
[15] M. Modrzejewski, Feature selection using rough sets theory, in: P.B. Brazdil (Ed.), Proceedings of the European Conference on Machine Learning, Vienna, Austria, 1993, pp. 213–226.
[16] Z. Pawlak, Rough sets, International Journal of Computer and Information Science 11 (5) (1982) 341–356.
[17] D. Slezak, Various approaches to reasoning with frequency based decision reducts: a survey, in: Rough Set Methods and Applications, Physica-verlag, Heidelberg, 2000, pp. 235–285.
[18] J. Starzyk, D.E. Nelson, K. Sturtz, Reduct generation in information systems, Bulletin of International Rough Set Society 3 (1998) 19–22.
[19] R. Susmaga, Reducts and constructs in attribute reduction, Fundamenta Informaticae 16 (61) (2004) 159–181.
[20] R.W. Swiniarski, A. Skowron, Rough set methods in feature selection and recognition, Pattern Recognition Letters 24 (2003) 833–849.
[21] J. Wang, J. Wang, Reduction algorithms based on discernibility matrix: the ordered attributes method, Journal of Computer Science and Technology 16 (2001) 489–504.
[22] X.Y. Wang, J. Yang, et al., Feature selection based on rough sets and particle swarm optimization, Pattern Recognition Letters 28 (2007) 459–471.
[23] X. Wang, W.X. Zhang, Relations of attribute reduction between object and property oriented concept lattices, Knowledge-Based Systems 21 (2008) 398–403.
[24] J. Wrblewski, Finding minimal reducts using genetic algorithms, In: Proceedings of Second Annual Join Conference on Information Sciences, Wrightsville Beach, NC. September 28–October 1, 1995, pp. 186–189.
[25] Y.Y. Yao, Information-theoretic measures for knowledge discovery and data mining, in: Entropy Measures, Maximum Entropy and Emerging Applications, Springer, Berlin, 2003, pp. 115–136.
[26] L.Y. Zhai et al., Feature extraction using rough set theory and genetic algorithms: an application for the simplification of product quality evaluation, Computers and Industrial Engineering 43 (2002) 661–676.