Soft Comput (2016) 20:2233–2243
DOI 10.1007/s00500-015-1638-0

METHODOLOGIES AND APPLICATION

Minimal attribute reduction with rough set based on compactness discernibility information tree

Yu Jiang · Yang Yu

Published online: 12 March 2015

Abstract Minimal attribute reduction plays an important role in rough set. Heuristic algorithms are proposed in literature reviews to get a minimal reduction and yet an unresolved issue is that many redundancy non-empty elements involving duplicates and supersets exist in discernibility matrix. To be able to eliminate the related redundancy and pointless elements, in this paper, we propose a compactness discernibility information tree (*CDI-tree*). The *CDI-tree* has the ability to map non-empty elements into one path and allow numerous non-empty elements share the same prefix, which is recognized as a compact structure to store non-empty elements in discernibility matrix. A complete algorithm is presented to address Pawlak reduction based on *CDI-tree*. The experiment results reveal that the proposed algorithm is more efficient than the benchmark algorithms to find out a minimal attribute reduction.

Keywords Rough set · Discernibility matrix · Minimal reduction · Compactness discernibility information tree

1 Introduction

Rough set theory, proposed by Pawlak in the 1980s, is a theory for the study of intelligent systems characterized by inexact, uncertain or vague information. It has found successful applications in such fields of artificial intelligence as machine learning, knowledge discovery, decision analysis, process control and pattern recognition (Slowinski 1992). It has become one of the flash points in the research area of information science.

Attribute reduction is one of the most important parts in rough set, which is defined as a process of deleting redundant attributes from larger set of condition attributes. As a consequence in Thangavel and Pethalakshmi (2009), up to now, many strategies for finding reducts have been investigated, such as discernibility matrix strategy (Chen et al. 2012; Zhou et al. 2014), positive region strategy (Qian et al. 2010; Jiang et al. 2011) and other strategies [information entropy (Zheng and Yan 2012; Jiang et al. 2015), Wasp Swarm Optimization (Huilian and Yuanchang 2012; Lustiana et al. 2013), hybrid genetic (Yuan 2014), Ant Colony Optimization (Majdi and Derar 2013; Lustiana et al. 2013)].

Discernibility matrix is a beautiful theoretical result for finding reducts in rough set, which was introduced in Skowron and Rauszer (1992). Both the rows and columns of the matrix correspond to the objects. An element $m_{i,j}$ of the matrix is the set of all attributes on which the corresponding two objects x_i and x_j have distinct values. Now, many researchers study attribute reduction based on discernibility matrix (or discernibility function) (Jiang et al. 2008; Yao and Zhao 2009; Chen et al. 2012; Zhou et al. 2014). Moreover, vast heuristic algorithms were proposed based on discernibility matrix. In Jiang et al. (2008), an elegant algorithm for finding a reduct based on the discernibility matrix and an order of attributes was introduced. In Yao and Zhao (2009), Yao and his colleagues introduced a method for constructing a minimal discernibility matrix whose elements were either the empty set or singleton set. The union of all elements in the minimal discernibility matrix produces a reduct. In Zhou et al. (2014), a quick attribute reduction algorithm

Communicated by V. Loia.

Y. Jiang (✉)
College of Software Engineering, Chengdu University of Information Technology, Chengdu 610225, China
e-mail: jiangyu@cuit.edu.cn

Y. Yu
College of Electric Information Engineering, Jiangsu University of Technology, Changzhou 213001, China

based on the improved discernbility matrix was introduced as well. During the process of finding reducts using the discernibility matrix, these algorithms have high cost of storage and heavy computing load. Since numerous redundancy non-empty elements involving duplicates and supersets exist in discernibility matrix. To eliminate the related redundancy elements, some elegant methods were proposed. In Jiang et al. (2008), a new discernibility matrix was introduced, which regards a decision-making class as a decision-making rule. It reduces the non-empty elements in the discernibility matrix. In addition, a novel condensing tree structure *C-Tree* and improving *C-Tree* were introduced in Yang and Yang (2008, 2011), respectively, which are order-tree based on the difference ordering strategy of condition attributes. Although they resolve the problem about the duplicates in the discernibility matrix and have lower space complexity as compared to the discernibility matrix, they never consider how to resolve the superset problem. Therefore, it is necessary to design a new method to store non-empty elements in discernibility matrix.

Generally speaking, researchers want to find not the set of all reducts, but a minimal reduct of a given decision table. Since a minimal reduct is a minimal subset of attributes that provides the same descriptive ability as the entire set of attributes (Yao and Zhao 2009). Thus, many heuristic methods for finding minimal reducts have been investigated (Vinterbo and Øhrn 2000; Jiang 2012; Ding et al. 2012; Dongyi and Zhaojiong 2014).

For example, in Vinterbo and Øhrn (2000), Vinterbo and Øhrn introduced a genetic algorithm for computing minimal hitting sets. In Jiang (2012), an algorithm for finding a minimal reduct based on attribute enumeration tree was also proposed. In Ding et al. (2012), Co-PSAR was introduced based on particle swarm optimization to find a minimal reduct. At the same time, an algorithm based on rough set and Wasp Swarm Optimization was also proposed in Huilian and Yuanchang (2012), which searches through the attribute space for finding a minimal reduct based on attribute significance. Moreover, in Dongyi and Zhaojiong (2014), an algorithm based on discrete artificial bee colony to find a minimal reduct was proposed as well.

In these methods, though heuristic algorithms for finding minimal reducts are effective, many of them can not always find a minimal reduct when decision tables are given and even the result is just a superset of a reduct sometimes.

With the above analysis, in this paper, a new data structure to store non-empty elements in the discernibility matrix, called compactness discernibility information tree (*CDI-tree* in short), is proposed. It is an extended order-tree based on the given order of condition attributes. The *CDI-tree* has the ability to eliminate duplicates and allow numerous non-empty elements share one path or the same prefix. It is recognized as a compact structure to store non-empty elements in discernibility matrix. To further eliminate supersets in discernibility matrix, *CDI-tree* incorporates some pruning strategies, such as core attribute pruning strategy, un-extension path strategy and truncation path (or deleting subtree) pruning strategy. Moreover, the time complexity of constructing a *CDI-tree* is $O(|C| * |U|^2)$. In particular, the space complexity of the *CDI-tree* is far less than $O(|C| * |U|^2)$ in most cases. To demonstrate the usefulness and flexibility of the *CDI-tree*, one heuristic algorithm for finding a minimal reduct is suggested. The approximately strategy of it is to involve deleting unimportant attributes in each iteration. The experiment results reveal that the proposed algorithm is more efficient than the benchmark algorithms to find out a minimal reduct. And the time complexity of this algorithm is $O(|C|^2 * |U|^2)$. At last, we prove in theory that the reduction found by this algorithm is a Pawlak reduction (by Pawlak, a reduction R is a Pawlak reduction, if and only if $\mathrm{POS}_R(D) = \mathrm{POS}_C(D)$, and $\forall a \in R$, $\mathrm{POS}_{R-\{a\}}(D) \neq \mathrm{POS}_C(D)$, Here, POS is the positive region).

The rest of this paper is structured as follows: in Sect. 2, we give basic concepts and attribute reduction related to rough set. In Sect. 3, we introduce the building process of *CDI-tree*. In Sect. 4, based on *CDI-tree* method, a heuristic algorithm is proposed to get a minimal reduct. At the same time, we prove in theory that the reduction found by this algorithm is a Pawlak reduction. In Sect. 5, we perform some experiments to show the usefulness of *CDI-tree*. Finally, we conclude the paper in Sect. 6.

2 Basic concepts of rough set theory

For the convenient description, we introduce some basic notions of information systems at first (Wong and Ziarko 1985; Pawlak 1991; Zhang et al. 2001).

Definition 1 A decision table (information table or information system) is an ordered quadruple $S = (U, C \cup D, V, f)$, where U is a non-empty finite set of objects, $C \cup D$ is a non-empty finite set of attributes, C denotes the set of condition attributes and D denotes the set of decision attributes. $C \cap D = \emptyset$. V is the union of attribute domains. $f : U \times (C \cup D) \to V$ is an information function which associates a unique value of each attribute with every object belonging to U. Let 'a' is an attribute in C, x_i is an object in U, then $f(x_i, a)$ denotes the value of object x_i in attribute 'a'. Table 1 lists a decision table, where $U = \{u_1, \ldots, u_6\}$, $C = \{a, b, c, d, e\}$ and $D = \{f\}$.

Given $|U| = n$, discernibility matrix of the decision table is a matrix which contains $n \times n$ elements. Each element of

Table 1 An example of decision table

U	*a*	*b*	*c*	*d*	*e*	*f*
u_1	Reject	Agree	Agree	Agree	Agree	Reject
u_2	Agree	Neutralism	Reject	reject	Reject	Accept
u_3	Agree	Reject	Reject	Reject	Reject	Reject
u_4	Agree	Reject	Reject	Neutralism	Reject	Accept
u_5	Neutralism	Reject	Neutralism	Reject	Reject	Reject
u_6	Neutralism	Agree	Neutralism	Agree	Reject	Reject

Table 2 Discernibility matrix corresponding to Table 1

	u_1	u_2	u_3	u_4	u_5	u_6
u_1	$\varnothing$	$\{a, b, c, d, e\}$	$\varnothing$	$\{a, b, c, d, e\}$	$\varnothing$	$\varnothing$
u_2		$\varnothing$	$\{b\}$	$\varnothing$	$\{a, b, c\}$	$\{a, b, c, d\}$
u_3			$\varnothing$	$\{d\}$	$\varnothing$	$\varnothing$
u_4				$\varnothing$	$\{a, c, d\}$	$\{a, b, c, d\}$
u_5					$\varnothing$	$\varnothing$
u_6						$\varnothing$

the discernibility matrix is defined as follows:

$$a^*(x_i, x_j) = \{a \in C | j > i \wedge f(x_i, a) \neq f(x_j, a) \wedge w(x, y) = 1)$$

Here, $\forall x, y \in U$, $w(x, y)$ is defined as

$$w(x, y) \begin{cases} 1 & x \in \mathrm{POS}_C(D) \wedge y \notin \mathrm{POS}_C(D) \\ 1 & x \notin \mathrm{POS}_C(D) \wedge y \in \mathrm{POS}_C(D) \\ 1 & x, y \in \mathrm{POS}_C(D) \wedge (x, y) \notin \mathrm{ind}(D) \\ 0 & \text{otherwise} \end{cases}$$

where ind(D) is an indiscernibility relation on U, and $\mathrm{POS}_C(D)$ is the positive region of D with respect to C.

Definition 2 Discernibility information, ***DI*** for short, is the non-empty element in discernibility matrix.

Definition 3 Discernibility set, ***Ds*** for short, is a set including all non-empty elements in discernibility matrix.

Definition 4 By ***DM***, if $a \in C \wedge \{a\} \in$ ***DM***, 'a' is called necessary attribute or core attribute. A set including all necessary attributes is called core and marked as ***Core(C)***.

According to above definitions, based on Table 2, ***Ds*** = $\{\{a,b,c,d,e\},\{a,b,c,d,e\},\{b\},\{a, b, c\},\{a, b, c, d\}, \{d\}, \{a, c, d\},\{a, b, c, d\}\}$, ***Core(C)*** = $\{b, d\}$.

Definition 5 A discernibility function f_A for a decision table A is a Boolean function of m Boolean variables $a'_1, \ldots, a'_m$ (corresponding to the attributes $a_1,\ldots,a_m$) defined as follows:

$$f_A(a'_1, \ldots, a'_m) = \wedge\{\vee a^*(x_i, x_j)|1 \leq j \leq i \leq n, a^*(x_i, x_j) \neq \varnothing\}$$

The set of all prime implicants of f_A determines the set of all reducts of A.

Example 1 The discernibility function for Table 1 is as follows:

$$\begin{aligned} f_A(a, b, c, d, e, f) = (a \vee b \vee c \vee d \vee e) \\ \times \wedge (a \vee b \vee c \vee d \vee e) \wedge b \\ \times \wedge (a \vee b \vee c) \wedge (a \vee b \vee c \vee d) \\ \times \wedge d \wedge (a \vee c \vee d) \wedge (a \vee b \vee c \vee d). \end{aligned}$$

So Table 1 has one reduction: $\{b, d\}$.

3 Compactness discernibility information tree: design and construction

Discernibility matrix is a beautiful theoretical result for finding out reducts in rough set. However, there are many redundancy non-empty elements involving duplicates and supersets in discernibility matrix, and all non-empty elements in discernibility matrix are employed to find out reducts, which leads to heavy computing load. Thus, this fact motivates our idea to develop a novel structure to eliminate the related redundancy elements.

In this section, with the help of some concept in Han et al. (2000), Yang and Yang (2008) and Chen et al. (2012), we introduce a compact data structure called compactness discernibility information tree to store non-empty elements in a discernibility matrix efficiently.

3.1 Compactness discernibility information tree

A compactness discernibility information tree (or *CDI-tree* in short) is a tree structure defined as follows:

1. It consists of one root labeled as "null", a set of condition-attribute-prefix subtree as the children of the root, and a condition-attribute-header table.
2. Each node in the condition-attribute-prefix subtree consists of five fields: *attribute-name*, *count*, *parent*, *children* and *node-link*, where *attribute-name* registers which condition attribute this node represents, *count* registers the number of discernibility information represented by the portion of path reaching this node, *parent* points to its parent node, *children* points to its all children, and *node-link* links to the next node in the *CDI-tree* carrying the same *attribute-name*, or null if there is none.
3. Each entry in the condition-attribute-header table consists of two fields: (1) *attribute-name* and (2) *head of node-link* (a pointer pointing to the first node in the *CDI-tree* carrying the *attribute-name*).

Based on this definition, we have the following *CDI-tree* construction algorithm.

```
Algorithm 1. constructSimpleCDI-tree(T) (Simple CDI-tree construction algorithm)
Input: A decision table T.
Output: CDI-tree.
constructSimpleCDI-tree(decision table T)
{
  (1). Create the root of a CDI-tree, TN, and label it as "null";
  (2). Let od be an order of condition attributes, which is obtained by choosing condition attributes
       from left to right in the decision table T;
  (3). Create the condition-attribute-header table HT[1..|C|], according to order od, get every condition
       attribute ca in turn and let its attribute-name and head of node-link be ca and null, respectively.
  (4). For each object pairs in decision table T do the following.
       Compute the discernibility information of each object pair by using formula (1). Select the attrib-
       utes in the discernibility information and sort them according to order od. Let the sorted discernibil-
       ity information be [b|B], where b is the first element and B is the remaining list. Call insert-
       Sub-tree([b|B],TN);
  (5). The function insertSub-tree([b|B],TN) is performed as follows:
     A. If (TN has a child N such that N.attribute-name = =b) {
            a) increase N's count by 1;
            b) if N is the leaf node then return; //un-extension strategy
            c) if b is the last attribute then delete the subtree of root N and remain root N; // truncation
            strategy
        }
        Else
            Create a new node N, with its count initialized to 1, its parent link linked to TN, and its
            node-link linked to the nodes with the same attribute-name via the node-link structure.
     B. If B is non-empty, call insertSub-tree(B, N).
}
```

Example 2 With the above algorithm, a *CDI-tree* can be constructed based on Table 1 as follows:

First of all, let ***od*** be the given order of condition attributes in Table 1: $\langle a, b, c, d, e\rangle$. This order is important since each path of the *CDI-tree* will follow this order. For each object pair in Table 1, compute its discernibility information using formula (1). Select the attributes in the discernibility information and sort them according to order ***od***.

Second, the first discernibility information $\{a, b, c, d, e\}$ leads to the construction of the first path or branch of the *CDI-tree*: $\langle a{:}1, b{:}1, c{:}1, d{:}1, e{:}1\rangle$ ($\langle a, b, c, d, e\rangle$ for short), in which each node is represented using the format "*attribute-name*: *count*". For the second discernibility information $\{a, b, c, d, e\}$, since its attribute list $\langle a, b, c, d, e\rangle$ is identical to the first one, the path is shared with the count of each node along the path incremented by 1. The third discernibility information $\{b\}$ leads to the construction of a new path of the *CDI-tree*:$\langle b{:}1\rangle$. For the fourth discernibility information $\{a, b, c\}$, since its attribute list $\langle a, b, c\rangle$ completely is contained in the existing path $\langle a, b, c, d, e\rangle$, thus all nodes behind node (c) must be deleted from path $\langle a, b, c, d, e\rangle$, the count of corresponding node along the path is incremented by 1. That is, the original path $\langle a{:}2, b{:}2, c{:}2, d{:}2, e{:}2\rangle$ is modified to $\langle a{:}3, b{:}3, c{:}3\rangle$. For the fifth discernibility information $\{a, b, c, d\}$, since its attribute list $\langle a, b, c, d\rangle$ completely contains the existing path $\langle a, b, c\rangle$, thus discernibility information $\{a, b, c, d\}$ is mapped to the existing path $\langle a, b, c\rangle$, the count of corresponding node along the path is incremented by 1. The sixth discernibility information $\{d\}$ leads to the construction of a new path of the *CDI-tree*:$\langle d{:}1\rangle$. For the seventh discernibility information $\{a, c, d\}$, because its attribute list

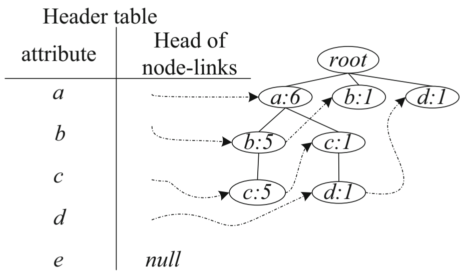

Fig. 1 The *CDI-tree* based on Algorithm 1 and Table 1

It is well known that core is the most important subset of condition attribute set C. Since none of its elements can be removed without affecting the classification power of attributes. If core is used to avoid mapping non-empty elements including core attributes into *CDI-tree*, there are better chances that more paths cannot be constructed if the core of the decision table is not empty, so that the efficiency of constructing *CDI-tree* can be improved.

With the above observations, we can improve Algorithm 1 based on core attributes, as shown in Algorithm 2.

Algorithm 2. *constructImprovingCDI-tree(T)* (*CDI-tree* construction algorithm with core attribute as heuristic information)

Input: A decision table *T*.

Output: *CDI-tree* and ***Core(C)***.

```
constructImprovingCDI-tree(T)
{
  (1). Create the root of a CDI-tree, TN, and label it as "null";
  (2). Let od be an order of condition attributes, which is obtained by choosing condition attributes from
       left to right in the decision table T;
  (3). Create the condition-attribute-header table HT[1..|C|], according to order od, get every condition
       attribute ca in turn and let its attribute-name and head of node-link be ca and null, respectively.
  (4). Create Core(C), and Core(C)=Ø;
  (5). For each object pairs in decision table T do the following.
       Compute the DI of each object pair by using formula (1). Select the attributes in the DI and sort
       them according to order od. Let the sorted DI be [b|B], where b is the first element and B is the
       remaining list.
  (6). If (DI∩Core(C) = =∅){
          If (the DI includes only one attribute $a_i$){
             Core(C)=Core(C)∪DI;
             Delete all paths including node ($a_i$) from CDI-tree;
          }
          Call insertSub-tree([b|B],TN);
       }
  (7). The function insertSub-tree([b|B],TN) is performed as follows:
     A. If (TN has a child N such that N.attribute-name = =b){
           a) increase N's count by 1;
           b) if N is the leaf node then return;
           c) if b is the last attribute then delete the subtree of root N and remain root N;
        }
     Else
        Create a new node N, with its count initialized to 1, its parent link linked to TN, and its node-link
        linked to the nodes with the same attribute-name via the node-link structure.
     B. If B is non-empty, call insertSub-tree(B, N).
}
```

$\langle a, c, d\rangle$ shares only the prefix $\langle a\rangle$ with the a-prefix subtree, a's count is incremented by 1, and one new node (c:1) is created and linked as a child of node (a:5). At the same time, another new node (d:1) is created and linked as a child of node (c:1). Repeat above procedure until the last discernibility information $\{a, b, c, d\}$ is inserted into the *CDI-tree*.

From Fig. 1, we can see that the *CDI-tree* has the ability to map non-empty elements into one path and allow a lot of non-empty elements share the same prefix. It is recognized as a compact structure to store non-empty elements in discernibility matrix.

According to Algorithm 2, a *CDI-tree* corresponding to Table 1 shows in Fig. 2.

Obviously, from Fig. 2, it shows that the cost of storage can be efficiently reduced as compared to Fig. 1. Moreover, some experimental results are designed in Sect. 5.

For any given decision table, if ***DIS*** is a set consisting of all paths in the *CDI-tree*, then $\boldsymbol{DIS} \subseteq \boldsymbol{DM}$ and $\forall\, \boldsymbol{DI} \in \boldsymbol{DM} - \boldsymbol{DIS}$, $\exists\, \boldsymbol{DI'} \in \boldsymbol{DIS} \wedge \boldsymbol{DI'} \subseteq \boldsymbol{DI}$, that is, the *CDI-tree* contains all discernibility information to find out reducts and eliminates the duplicates and supersets exiting in the discernibility matrix. Therefore, we can naturally deduce that the existing discernibility matrix-base methods can be used under *CDI-*

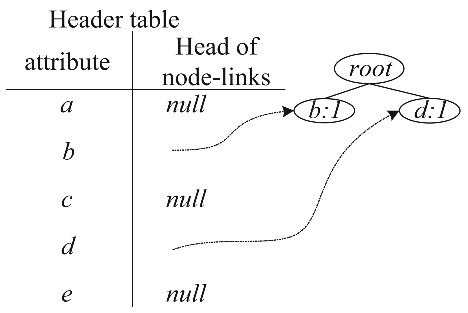

Fig. 2 The *CDI-tree* based on Algorithm 2 and Table 1

tree structure. To efficiently utilize the *CDI-tree* structure for attribute reduction, one *CDI-tree-based* rough set attribute reduction algorithm is developed in Sect. 4.

There are several important properties of the *CDI-tree* that can be derived from the *CDI-tree* construction process.

Definition 6 (*Core node*) For a given *CDI-tree*, if a path contains only one node, then this node is the core node.

Property 1. For a given *CDI-tree*, the attribute represented by core node is a core attribute.

Rational. If a path contains only one node in *CDI-tree*, then there must be an element in discernibility matrix containing only one condition attribute that is mapped to this path. According to the definition about core based on the discernibility matrix, the condition attribute represented by the *attribute-name* of core node is the core attribute.

Property 2. For a given *CDI-tree*, if R is the attribute set, which consists of all attributes represented by all of the root node's children, then $\forall A \in \boldsymbol{DM}$, $R \cap A \neq \varnothing$ (or $\mathrm{POS}_R(D) = \mathrm{POS}_C(D)$) holds.

3.2 Complexity analysis of compactness discernibility information tree

For a given decision table with $|U|$ objects and $|C|$ condition attributes, the number of non-empty elements in a discernibility matrix is $|U|^2$ at most. Let the number of different non-empty elements in a discernibility matrix be N. In the worst case, N is equal to $|U|^2$. By the *CDI-tree* construction process, we know that the number of all paths in the *CDI-tree* is N at most, and the number of nodes in a path is $|C|$ at most. So, the space complexity of the *CDI-tree* is $O(|C| * |U|^2)$ in the worst case. Moreover, in most cases, the nodes in the *CDI-tree* are far less than $|C| * |U|^2$, because numerous non-empty elements in the discernibility matrix share the same path or the same prefix.

During the *CDI-tree* construction process, the iteration times of it are $|U|^2$ at most, and the number of nodes inserting into *CDI-tree* and the number of nodes deleting from the *CDI-tree* are $|C|$ and N_i at most in each iteration, respectively. As analyzed above, the time complexity of building an *CDI-tree* is denoted as follows:

$$|C| * |U|^2 + (N_1 + \cdots + N^2_{|C|*|U|}) = |C| * |U|^2 + (N_1 + \cdots + N^2_{|C|*|U|}).$$

In addition, the number of nodes in the *CDI-tree* is $|C| * |U|^2$ at most. $N_1 + \cdots + N^2_{|C|*|U|}$ is equal to $|C| * |U|^2$ in the worst case. So, $|C| * |U|^2 + (N_1 + \cdots + N^2_{|C|*|U|}) = 2 * |C| * |U|^2$. The time complexity for building a *CDI-tree* is $O(|C| * |U|^2)$.

4 Attribute reduction algorithm based on compactness discernibility information tree

In this section, to efficiently utilize *CDI-tree* structure for attribute reduction, one *CDI-tree-based* attribute reduction algorithm is developed. This algorithm is to involve deleting unimportant attributes in each iteration.

4.1 Attribute significance

Attribute significance denotes the significance of attribute in decision table, which offers the powerful reference to the decision. The bigger the significance of attribute, the higher its position in the decision table, otherwise, the lower its position.

Definition 7 The attribute significance is defined as the number of each condition attribute symbol appearing in the *CDI-tree*, and denoted as $Sig(a)$.

According to Definition 7 and Fig. 1 $Sig(a) = 6$, $Sig(b) = 6$, $Sig(c) = 6$, $Sig(d) = 2$, $Sig(e) = 0$.

4.2 Minimal reduction algorithmic descriptions and pseudocode

In this subsection, we propose an algorithm for finding a minimal reduct based on *CDI-tree*. The approximately strategy of it is to delete a most unimportant attribute in each iteration, which guarantees that this algorithm can keep important attributes. At the same time, this algorithm deletes all paths including core node in every iteration as well.

According to the above description, the pseudocode for finding a minimal reduct based on *CDI-tree* is shown as follows:

Algorithm 3. (***M***inimal ***A***ttribute ***R***eduction ***A***lgorithm based on *CDI-tree*, ***MARACDI-tree*** in short)
Input: A decision table *T*.
Output: A minimal attribute reduct ***R***.

```
getMinimalReduction(decision table T)
{
    (1). Construct a CDI-tree based on Algorithm 2;
    (2). Create an empty set $R$;
    (3). Create empty sets $B$ and $P$ respectively;
    (4). If CDI-tree only includes the root, then output $R$ and return;
    (5). Create an empty set $Co$, and find core attribute in CDI-tree, and insert all core attributes into $Co$;
    (6). Let $B = Co$;
    (7). If $B \neq \varnothing$, goto (10);
    (8). For each attribute $a$ in $C-(R \cup P)$ , compute $sig(a)$;
    (9). Select most unimportant attribute $a$, and find all nodes whose attribute-name register attribute $a$,
         call mergeSub-tree(currentNode,parentNode), and let $P= P \cup \{a\}$; goto(4);
    (10). for each attribute $a$ in $B$, delete all branches containing attribute $a$; and $R=R \cup B$; goto (4).
}
```

The function ***mergeSub-tree***(***currentNode***, ***parentNode***) (***parentNode*** is the parent of ***currentNode***) is performed as follows:

(1) If ***currentNode*** is a leaf node, then delete the subtree of ***parentNode*** from *CDI-tree* and return.
(2) The entire subtree of ***currentNode*** is merged with the subtree of ***parentNode***, delete ***currentNode*** from *CDI-tree*, and return.

4.3 Completeness of algorithm

By the viewpoint of discernibility matrix, a reduction is a Pawlak reduction, if ①. $\forall A \in \boldsymbol{Ds}$, $A \cap R \neq \varnothing$, ②. $\forall a \in R$, $\exists A \in \boldsymbol{Ds}$, $A \cap (R-\{a\}) = \varnothing$. (Jue and Ju 2001).

Theorem 1 *If R is a reduction found out by Algorithm 3, then R is a Pawlak reduction.*

Proof According to Algorithm 3, a loop is constructed from step (4) to step (10) in this algorithm. $\boldsymbol{R} \cap P = \varnothing$ and $(\boldsymbol{R} \cup P) \subseteq C$. Suppose the number of the loop is i. Here, i is an integer ($|C| \geq i \geq 0$). $\boldsymbol{DIS}$ is a set, which consists of all paths in the *CDI-tree*. Let $\boldsymbol{DIS}^0,\ldots, \boldsymbol{DIS}^k,\ldots, \boldsymbol{DIS}^{i-1}, \boldsymbol{DIS}^i$ be a change list of the $\boldsymbol{DIS}$. Similarly, Let $\boldsymbol{R}^0,\ldots, \boldsymbol{R}^k,\ldots, \boldsymbol{R}^{i-1}$, $\boldsymbol{R}^i$ be a change list of the $\boldsymbol{R}$. Let $\boldsymbol{P}^0,\ldots, \boldsymbol{P}^k,\ldots, \boldsymbol{P}^{i-1}$, $\boldsymbol{P}^i$ be a change list of the $\boldsymbol{P}$. Here, $\boldsymbol{R}^0 = \varnothing$, $\boldsymbol{R}^i = \boldsymbol{R}, \boldsymbol{P}^0 = \varnothing$, $\boldsymbol{DIS}^0 \subseteq \boldsymbol{Ds}$ and $\forall \boldsymbol{DI} \in \boldsymbol{Ds}-\boldsymbol{DIS}^0$, $\exists \boldsymbol{DI'} \in \boldsymbol{DIS}^0 \wedge \boldsymbol{DI'} \subseteq \boldsymbol{DI}$. If $m \geq n$, then $\boldsymbol{R}^n \subseteq \boldsymbol{R}^m$ and $\boldsymbol{P}^n \subseteq \boldsymbol{P}^m$ hold. And (1) $\forall A \in \boldsymbol{DIS}^k$, $\exists A' \in \boldsymbol{DIS}^{k-1}$, $A \subseteq A'$ holds. (2) $\forall A \in \boldsymbol{DIS}^{k-1}$, if $\nexists A' \in \boldsymbol{DIS}^k$, $A \subseteq A'$ holds, then $|A|=1$ and $A \subseteq \boldsymbol{R}^{k-1}$ hold, that is, $A \cap \boldsymbol{R}^{k-1} \neq \varnothing$ holds. (3) $\forall \ A \in \boldsymbol{DIS}^k$, $A' \in \boldsymbol{DIS}^{k-1}$, if $A \subseteq A'$, then $A = A'$ or $|A|+1=|A'|$ holds. Moreover, if $A \subseteq A'$ and $|A| + 1 = |A'|$, then $(A'-A) \subseteq \boldsymbol{P}^{k-1}$and $(A'-A) \cap \boldsymbol{R}^{k-1} = \varnothing$ hold. In a word, $\forall A \in \boldsymbol{Ds}$, $A \cap R \neq \varnothing$, and $\forall a \in R$, $\exists A \in \boldsymbol{Ds}$, $A \cap (R-\{a\}) = \varnothing$ hold. Therefore, the reduction found out by Algorithm 3 is a Pawlak reduction. □

4.4 Time complexity analysis

According to the analysis in Sect. 3.2, we can observe that the number of nodes in *CDI-tree* is $|C| * |U|^2$ and the time complexity of building a *CDI-tree* is $O(|C| * |U|^2)$ in the worst case.

By Algorithm 3, its iteration times is $|C|$ at most, and the time complexity of computing attribute significance is $O(|C| * |U|^2)$, and the time complexity of ***mergeSub-tree*** (***currentNode***, ***parentNode***) is $O(|C| * |U|^2)$ in the worst case. Thus, the time complexity of Algorithm 3 is as follows: $O(|C| * |U|^2) + |C| * O(|C| * |U|^2 + |C| * O(|C| * |U|^2) + (N_1 + \cdots + N_{|C|})$.

Since, $N_1 + \cdots + N_{|C|}$ is equal to $|C| * |U|^2$ at most. Therefore, the time complexity of Algorithm 3 is $O(|C|^2 * |U|^2)$.

5 Experimental results and analysis

In this section, to demonstrate the usefulness and compactness of the *CDI-tree* method, we show some experimental comparisons. First of all, we perform an experiment to verify core attribute pruning strategy that plays an essential role in the *CDI-tree* construction process when the core of a given decision table is not empty. Second, we perform an experiment to compare the *CDI-tree* proposed in this paper with *C-Tree* proposed in Yang and Yang (2008) based on the same discernibility matrix and attribute order. At last, we perform another experiment to demonstrate the reduction results for finding out a minimal reduct between ***MARACDI-tree***, ***MinARA***, ***JohnsonReducer*** and ***SAVGeneticReducer***. Here, they are ***MARACDI-tree*** in this paper, ***MinARA*** in Jiang (2012), ***JohnsonReducer*** in Johnson (1974) and ***SAVGeneticReducer*** in Vinterbo and Øhrn (2000).

Table 3 The Detailed information of the data sets used in the comparison

Database	Abbreviation	$\|U\|$	$\|C\|$	$\|D\|$	$\|Core(C)\|$
Lenses	Lenses	24	4	1	4
Hayes roth	HR	132	4	1	3
SPECT	SPECT	269	44	1	0
Voting	Voting	435	16	1	7
Balance scale	BS	625	4	1	4
Tic-tac-toe	TTT	958	9	1	0
Chess	Chess	3196	36	1	27
Mushroom	MR	8124	22	1	0
Letter	Letter	20,000	15	1	4
Poker hand	PH	25,010	10	1	5
Shuttle	Shuttle	58,000	9	1	1
Connect4	Connect4	67,557	42	1	15

We choose some data sets from UCI database (http://www.ics.uci.edu/~mlearn/Machine-Learning.html) for the comparison. The detailed information shows in Table 3.

Here, $|U|$ is the number of objects of a decision table, $|C|$ is the number of the condition attribute, $|D|$ is the number of decision attribute, and $|Core(C)|$ is the number of core attribute of a decision table.

In addition, the experiments in Sects. 5.1 and 5.2 are conducted on a 3.2-Ghz Pentium® dual-core with 2 GB of memory running Microsoft Windows 7. However, the experiments in Sect. 5.3 are conducted on a 1.73-Mhz Pentium III with 768 MB of memory running Microsoft Windows XP, because ***JohnsonReducer*** and ***SAVGeneticReducer*** are implemented in **ROSETTA** and **ROSETTA** runs Microsoft Windows XP. All codes were compiled using Microsoft Visual Studio 2005.

5.1 Comparisons of constructing CDI-tree with and without core attribute pruning

It is well known that core is the most important subset of condition attribute C. None of its elements can be removed without affecting the classification power of attributes. In this part, we perform an experiment to verify core attribute pruning strategy playing an essential role in the *CDI-tree* construction process when the core of a given decision table is not empty.

During the *CDI-tree* construction process, core attribute pruning guarantees that the *CDI-tree* construction algorithm with core attribute pruning. So *CDI-tree* based on Algorithm 2 (with core attribute pruning) requires the less computational cost and less storage cost than *CDI-tree* based on Algorithm 1 (without core attribute pruning) as shown in Figs. 3 and 4. Clearly, the *CDI-tree* based on Algorithm 1 is equal to the *CDI-tree* based on Algorithm 2 if the core of a given decision table is empty.

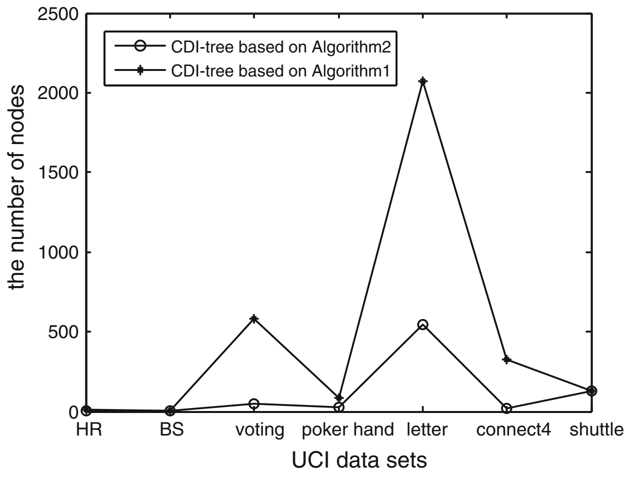

Fig. 3 The number of nodes in *CDI-tree* on some datasets

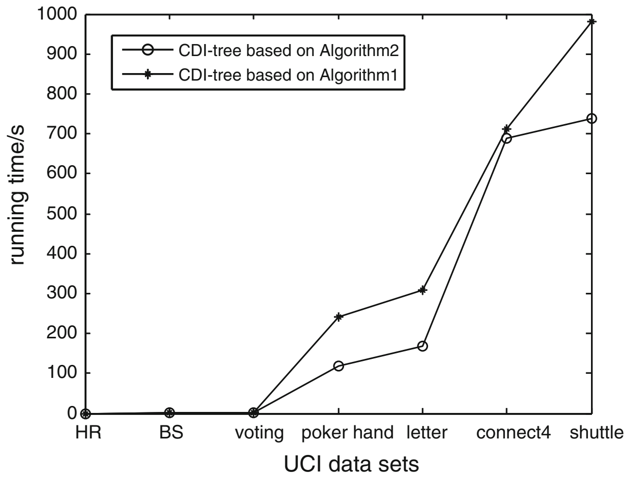

Fig. 4 The running time on some datasets

The experimental result in Fig. 3 shows that *CDI-tree* with core attribute pruning needs the less computational cost than *CDI-tree* without core attribute pruning. Since core attribute pruning guarantees that the *CDI-tree* construction algorithm does not construct all the paths including any core attribute. For instance, using letter, voting and connect4, the number of nodes in *CDI-tree* based on Algorithm 1 is approximately 4 times, 12 times and 12 times of the number of nodes in *CDI-tree* based on Algorithm 2, respectively.

In Fig. 4, we can see that *CDI-tree* with core attribute pruning is obviously faster than *CDI-tree* without core attribute pruning. For example, using poker hand, letter and shuttle, the running time of *CDI-tree* with core attribute pruning (or *CDI-tree* without core attribute pruning) is 118.872 (241.769), 168.885 (307.133) and 739.363 (980.352) respectively.

In summary, we can conclude that *CDI-tree* with core attribute pruning needs the less computational cost and less storage cost than *CDI-tree* without core attribute pruning.

Table 4 The experimental result based on *C-Tree* and *CDI-tree*

Database	$\|C\|$	$\|Core(C)\|$	$\|U\|$	The number of nodes		The construct time (s)	
				C-Tree	CDI-tree	C-Tree	CDI-tree
Chess	36	27	3196	2,665,537	51	12.855	2.278
Letter	15	4	20,000	32,528	545	361.016	168.885
Balance scale	4	4	625	16	5	0.109	0.046
Voting	16	7	435	16,746	50	0.125	0.078
Mushroom	22	0	8124	29,410	21,044	49.670	49.608
Tic-tac-toe	9	0	958	510	45	0.406	0.312
Poker hand	10	5	25,010	1024	23	273.047	118.872
Shuttle	9	1	58,000	389	126	1046.684	739.363
Connect4	42	15	67,557	413,100	21	698.678	688.788
Hayes roth	4	3	132	16	4	0.016	0.0

Table 5 The reduction results to find a minimal reduct.

Database	The cardinality of minimal reducts			
	$\|$ ***JohnsonReducer*** $\|$	$\|$***SAVGeneticReducer***$\|$	$\|$***MinARA***$\|$	$\|$***MARACDI-tree***$\|$
Lenses	4	4	4	4
Hayes roth	3	3	4	3
SPECT	3	3	3	2
Chess	29	29	29	29
Balance scale	4	4	4	4
Voting	9	9	9	9
Tic-tac-toe	8	8	8	7
Poker hand	7	7	7	6
Shuttle	Out of memory	Out of memory	Out of memory	4
Connect4	Out of memory	Out of memory	16	16

5.2 Comparison of CDI-tree and C-Tree

In this part, we experimentally demonstrate the running time and the storage cost in *CDI-tree* and *C-Tree* based on the same discernibility matrix and attribute order. As pointed in Yang and Yang (2008), the *C-Tree* is a compact tree structure, which may guarantee that less storage cost is required than the discernibility matrix. As compared with *CDI-tree*, *C-Tree* can restore all information coming from a discernibility matrix, that is, any non-empty element of a discernibility matrix can have a path in *C-Tree*. *CDI-tree* contains all information to find out reducts and integrates many pruning strategies, which can eliminate lots of redundancy elements and achieve compactness storage of the elements in the discernibility matrix. For instance, the non-empty elements, such as $\{a, b\}$, $\{a, b, c\}$, $\{a, b, c\}$, $\{a, b, c\}$, $\{a, b, d\}$ and $\{a, b, c, d\}$, share the same path $\langle a, b\rangle$ in the *CDI-tree*. Therefore, we naturally come to the conclusion that *CDI-tree* needs the less computational cost and less storage cost than *C-Tree*.

The experimental result in Table 4 shows that *CDI-tree* is a compact data structure, which also guarantees that the less computational cost and less storage cost are required than *C-Tree* based on same discernibility matrix and attribute order. For example, using *C-Tree* structure, chess, connect4, letter, voting and tic-tac-toe contain over 2.6×10^6, 4.1×10^5, 3.2×10^4, 1.6×10^4 and 5.1×10^2 nodes, respectively, while they only generate 51, 21, 545, 50 and 45 nodes by employing the *CDI-tree*, respectively. Furthermore, from Table 3, we can also observe that the running time of *C-Tree* is approximately 6 times, 3 times and 3 times of *CDI-tree* in chess, letter and poker hand, respectively. Therefore, it is obvious that the *CDI-tree* is more compactness than the *C-tree*, which can also guarantee that less computational cost is required.

5.3 Comparison of algorithms to find a minimal Reduct

In this part, we compare the method proposed in this paper (***MARACDI-tree***) with three common reduction methods for finding minimal reducts. They are ***MinARA*** algorithm based on attribute enumeration tree (Jiang 2012), ***JohnsonReducer*** algorithm and ***SAVGeneticReducer*** algorithm. ***JohnsonReducer*** invokes a variation of a simple greedy algorithm to compute a minimal reduct, as described by Johnson in Johnson (1974). The algorithm has a natural bias towards finding a single prime implicant of minimal length. ***SAVGeneticReducer*** is a genetic algorithm for computing minimal hitting sets, as described by Vinterbo and Øhrn (2000). ***JohnsonReducer*** and ***SAVGeneticReducer*** are implemented in **ROSETTA** that is a rough set toolkit for analysis of data. You can check out the **ROSETTA** homepage at http://www.idi.ntnu.no/~aleks/rosetta/ for information.

We experimentally compare these methods and summarize the results in Table 5. From Table 5, we can observe that ***MARACDI-tree***, ***MinARA***, ***JohnsonReducer*** and ***SAVGeneticReducer*** often find out a minimal reduct. Table 5 shows that ***MARACDI-tree***, ***JohnsonReducer*** and ***SAVGeneticReducer*** can find out a minimal reduct on "hayes roth", whereas ***MinARA*** does not. Similarly, ***MARACDI-tree*** can find out a minimal reduct on "SPECT", "tic-tac-toe", and "poker hand", whereas ***MinARA***, ***JohnsonReducer*** and ***SAVGeneticReducer*** cannot. In addition, from Table 5, we can also see that ***MinARA***, ***JohnsonReducer*** and ***SAVGeneticReducer*** do not suit for large data sets since their workloads are so heavy that they will be stopped because of out of memory in experimental software and hardware environments. In summary, it has been proven that finding a minimal reduct is NP-hard problem. Although we cannot prove in theory that our algorithm can find out a minimal reduct in any cases, from this section we draw a conclusion that ***MARACDI-tree*** is the best one to find out a minimal reduct based on UCI datasets.

6 Conclusions

Attribute reduction is the key research point of rough set theory that has been widely studied. An effective way for attribute reduction is to obtain the discernibility matrix. Unfortunately, a large number of redundant elements exist in discernibility matrix such as the same elements and the supersets. In this paper, we propose a *CDI-tree* recognized as a compact structure to store non-empty elements of a discernibility matrix, which can eliminate the related redundancies and pointless elements. The experimental results demonstrate that *CDI-tree* has the less computational cost and less storage cost than *C-Tree* has. A heuristic algorithm is proposed for minimal attribute reduction based on *CDI-tree*. The experimental performance using UCI datasets shows that the proposed algorithm could find the expected minimal attribute reduction. We plan to have tasks in the future work. First is to explore the methods to get the minimal number of nodes containing in *CDI-tree*, and the second is to prove in theory that the proposed minimal attribute reduction algorithm could find the minimal reduct for any decision tables.

References

Chen D, Zhao S, Zhang L et al (2012) Sample pair selection for attribute reduction with rough set. IEEE Trans Knowl Data Eng 24(11):2080–2093
Ding W, Wang J, Guan Z (2012) Cooperative extended rough attribute reduction algorithm based on improved PSO. J Syst Eng Electron 23(1):160–166
Feng JIANG, Sha-sha WANG, Jun-wei DU et al (2015) Attribute reduction based on approximation decision entropy. Control Decis 30(1):65–70
Han JW, Pei J, Yin YW (2000) Mining frequent patterns without candidate generation. Weidong C, Jeffrey F, Proceedings of the ACM SIGMOD conference on management of data. ACM Press, Dallas, pp 1–12
Huilian FAN, Yuanchang ZHONG (2012) A rough set approach to feature selection based on wasp swarm optimization. J Comput Inf Syst 8(3):1037–1045
Jiang Yu, Wang Xie, Ye Zhen (2008) Attribute reduction algorithm of rough sets based on discernibility matrix. J Syst Simul 20(14):3717–3720, 3725
Jiang Y (2012) Minimal attribute reduction for rough set based on attribute enumeration tree. Int J Adv Comput Technol 4(19):391–399
Jian-hua ZHOU, Zhang-yan XU, Chen-guang ZHANG (2014) Quick attribute reduction algorithm based on the improved discernibility matrix. J Chin Comput Syst 35(4):831–834
Jing S-Y (2014) A hybrid genetic algorithm for feature subset selection in rough set theory. Soft Comput 18(7):1373–1382
Johnson DS (1974) Approximation algorithms for combinatorial problems. J Comput Syst Sci 9:256–278
Jue W, Ju W (2001) Reduction algorithms based on discernibility matrix: the ordered attributes method. J Comput Sci Technol 16(6):489–504
Lakshmipathi Raju NVS, Dr MN, Seetaramanath D, Srinivasa Rao P et al (2013) Rough set based privacy preserving attribute reduction on horizontally partitioned data and generation of rules. Int J Adv Res Comput Commun Eng 2(11):4343–4348
Mafarja M, Eleyan D (2013) Ant colony optimization based feature selection in rough set theory. Int J Comput Sci Electron Eng 1(2):244–247
Nguyen T-T, Nguyen P-K (2013) Reducing attributes in rough set theory with the viewpoint of mining frequent patterns. Int J Adv Comput Sci Appl 4(4):130–138
Pawlak Z (1982) Rough sets. Int J Comput Inf Sci 11(5):341–356
Pawlak Z (1991) Rough set: theoretical aspects of reasoning about data. Kluwer Academic Publishers, Boston
Pratiwi L, Choo Y-H et al (2013) Immune ant swarm optimization for optimum rough reducts generation. Int J Hybrid Intell Syst 10(3):93–105
Qian YH, Liang JY, Pedrycz W, Dang CY (2010) Positive approximation: an accelerator for attribute reduction in rough set theory. Artif Intell 174:597–618
Skowron A, Rauszer C (1992) The discernibility matrices and functions in information systems. In: Slowinski R (ed) Intelligent decision support. Handbook of Applications and Advances of the Rough Sets Theory, Kluwer, Dordrecht
Slowinski R (1992) Intelligent decision support–handbook of applications and advances of the rough sets theory. Kluwer Academic Publishers, London

Thangavel K, Pethalakshmi A (2009) Dimensionality reduction based on rough set theory: a review. Appl Soft Comput 9:1–12

Vinterbo S, Øhrn A (2000) Minimal approximate hitting sets and rule templates. Int J Approx Reason 25(2):123–143

Wang GY, Zhao J, An JJ (2005) A comparative study of algebra viewpoint and information viewpoint in attribute reduction. Fundam Inf 68(3):289–301

Wong SKM, Ziarko W (1985) On optional decision rules in decision tables. Bull Polish Acad Sci 33(11/12):693–696

Xin-min TAO, Yan WANG, Jing XU et al (2012) Minimum rough set attribute reduction algorithm based on viruscoordinative discrete particle swarm optimization. Control Decis 27(2):259–265

Yang M, Yang P (2008) A novel condensing tree structure for rough set feature selection. Neurocomputing 71(4):1092–1100

Yang M, Yang P (2011) A novel approach to improving C-Tree for feature selection. Appl Soft Comput 11(2):1924–1931

Yao YY, Zhao Y (2009) Discernibility matrix simplification for constructing attribute reducts. Inf Sci 179(5):867–882

Ye D, Chen Z (2014) A new approach to minimum attribute reduction based on discrete artificial bee colony. Soft Comput. doi:10.1007/s00500-014-1371-0

Yu JIANG, Yin-Tian LIU, Chao LI (2011) Fast algorithm for computing attribute reduction based on bucket sort. Control Decis 26(2):207–212

Zhang WenXiu W, WeiZhi LJY et al (2001) Rough set theory and method. Beijing Science Press, Beijing

Zhang Q, Shen W (2014) Research on attribute reduction algorithm with weights. J Intell Fuzzy Syst 27(2):1011–1019

Zheng J, Yan R (2012) Attribute reduction based on cross entropy in rough set theory. J Inf Comput Sci 9(3):745–750

Zhou J, Miao D, Feng Q et al. (2009) Research on complete algorithms for minimal attribute reduction. In: Proceedings of the 4th international conference on rough sets and knowledge technology. Gold Coast, Australia, pp 152–159

Zhou J, Miao D, Feng Q, Sun L (2009) Research on complete algorithms for minimal attribute reduction. Rough sets and knowledge technology, volume 5589 of lecture notes in computer science. Springer, Berlin, pp 152–159