Knowl Inf Syst (2016) 46:599–628
DOI 10.1007/s10115-015-0824-9

REGULAR PAPER

# The logic transformations for reducing the complexity of the discernibility function-based attribute reduction problem

**Mehmet Hacibeyoglu · Mohammad Shukri Salman · Murat Selek · Sirzat Kahramanli**

Received: 30 December 2013 / Revised: 4 January 2015 / Accepted: 2 February 2015 /
Published online: 22 February 2015


**Abstract** The basic solution for locating an optimal reduct is to generate all possible reducts and select the one that best meets the given criterion. Since this problem is NP-hard, most attribute reduction algorithms use heuristics to find a single reduct with the risk to overlook for the best ones. There is a discernibility function (DF)-based approach that generates all reducts but may fail due to memory overflows even for datasets with dimensionality much below the medium. In this study, we show that the main shortcoming of this approach is its excessively high space complexity. To overcome this, we first represent a DF of $n$ attributes by a bit-matrix (BM). Second, we partition the BM into no more than $n-1$ sub-BMs (SBMs). Third, we convert each SBM into a subset of reducts by preventing the generation of redundant products, and finally, we unite the subsets into a complete set of reducts. Among the SBMs of a BM, the most complex one is the first SBM with a space complexity not greater than the square root of that of the original BM. The proposed algorithm converts such a SBM with $n$ attributes into the subset of reducts with the worst case space complexity of $\binom{n}{n/2}/2$.

**Keywords** Attribute reduction · Bit-matrix partitioning · CNF to DNF conversion · Computational complexity · Discernibility function · Set cover

M. Hacibeyoglu (✉)
Department of Computer Engineering, Necmettin Erbakan University, Konya, Turkey
e-mail: hacibeyoglu@konya.edu.tr

M. S. Salman
Department of Electrical and Electronic Engineering, Mevlana University, Konya, Turkey
e-mail: mssalman@mevlana.edu.tr

M. Selek
Technical Vocation School of Higher Education, Selcuk University, Konya, Turkey
e-mail: mselek@selcuk.edu.tr

S. Kahramanli
Department of Computer Education and Instructional Technologies Teaching,
Mevlana University, Konya, Turkey
e-mail: sirzat@selcuk.edu.tr

## 1 Introduction

An *information system* (IS) is usually represented as a triple $S = (U, A, d)$, where $U = \{u_1, u_2, \ldots, u_m\}$ is a set of uniformed *objects* ( *records, examples, instances, etc.*) referred to as a *dataset*, $A = \{a_1, a_2, \ldots, a_n\}$ is the set of *condition attributes* ( *features*), and $d$ is a *decision attribute* ( *class attribute*) of the dataset. The number of the objects of a dataset and the number of attributes describing this dataset are called the *size* and the *dimension* of the dataset, respectively. For instance, the dataset with $U = \{u_1, u_2, \ldots, u_m\}$ and $A = \{a_1, a_2, \ldots, a_n\}$ has a size of $m$ objects and a dimension of $n$ attributes.

One of the basic problems of ISs is the selection of a *minimal subset of attributes* (MSA) for a given dataset so that the selected MSA not only significantly reduces the dataset, but also provides a correct classification of objects within the reduced dataset.

In the *rough set theory*, every MSA is referred to as a *reduct*, the number of attributes composing a reduct is referred to as the *size* of the reduct, and the problem of finding the reducts for datasets is referred to as *attribute reduction* (AR) or *feature selection*, that is, one of the specific appearances of the *unweighted set cover* problem. In this study, the term *attribute reduction* is used.

AR provides many benefits for ISs such as: reducing the dimension and size of the dataset, simplifying the classification rules of the dataset with improving their classification efficiency, speeding up the data mining algorithms, reducing the amount of storage needed for ISs [1–7]. Due to these and other benefits, AR is widely used for *preprocessing* the datasets in ISs such as pattern recognition, knowledge discovery, data mining, information retrieval, computer vision, and bioinformatics [1,2,4,6,7].

From a methodological point of view, existing AR approaches are divided into two categories: *Wrapper approaches* (WAs) and *Filter approaches* (FAs). Since a WA selects an optimal reduct by minimizing the training error of the chosen classifier, it can achieve better classification accuracy for only particular classifiers. Moreover, the WAs are time-consuming. In contrast, the FAs evaluate attributes based on criteria independent of any classifier and find reducts by optimizing these criteria [5,6]. In this study, we deal only with FAs.

According to the *rough set theory*, once a reduct for a dataset has been computed, the rules classifying the dataset are easily constructed by overlaying it over the originating dataset and reading the values off [1,8]. But, unluckily, an optimal reduct is usually relative to a certain criterion, imposing some restrictions on the parameters such as the *reduct size,* the dataset *classification accuracy,* the *query response time,* the *memory capacity* required, and the *degree of inconsistency* of the reduced dataset [5,9–11]. The basic solution for locating an optimal reduct is to generate all possible reducts and retrieve those that meet the given criterion. Unfortunately, the solution to such problem is NP-hard [5,10–12].

In order to reduce the computational complexity of AR, most AR methods involve the heuristic criteria based on measures such as *distance, information, dependency* ( *correlation*), *learning model performance, consistency* and so on [7,13–15]. According to the first four criteria, an attribute $X$ is preferred over any other attribute $Y$ if $X$ induces a *greater difference* between the two-class conditional probabilities than $Y$; if the *information gain* from $X$ is greater than that from $Y$; or if the *association* between $X$ and the class $C$ is higher than the association between $Y$ and $C$; and/or if the *classification accuracy* provided by $X$ is higher than that by $Y$. As for the *consistency-based* heuristic, it selects such a minimum number of attributes that separates the dataset into consistent subsets of objects [10,13]. Unfortunately, every heuristic AR method generates a single reduct or a small number of possible reducts [2,7,13,14,16,17] with the risk of overlooking the optimal ones. Since heuristic AR methods

cannot guarantee the optimality of the reduct found [16,17], they estimate the efficiency of a reduct via the experiments on the dataset for which it was generated [10,18].

Above, it has been mentioned that the basic solution for locating an optimal reduct is to generate all possible reducts and retrieve those that meet the given criterion. But, unfortunately, there is only one such method developed by *Skowron* and *Rauszer* and named as the *discernibility function* (DF)-*based* AR [8,10,17,19–22]. It is based on the conversion of the DF of a dataset from the *conjunctive normal form* (CNF) to the *disjunctive normal form* (DNF). But since the CNF to DNF conversion is NP-hard, the DF-based AR is only applicable for simple datasets [8,10,14,17,19,23].

The analysis of CNF to DNF conversion processes shows that there are two main factors complicating it: the number of *prime implicants* (PIs) and the number of *redundant products* (RPs) generated during the process. Recall that a *product* is a conjunction of at least two logic variables. In the process of the CNF to DNF conversion, every product can be absorbed by the others or grow up to a member of the last result. While every product of the first type is called as a RP, every product of the second type is called as a PI. Recall that the aim of the CNF to DNF conversion for a dataset is to obtain the set of PIs of the DF belonging to the dataset and to consider each PI as one certain reduct of the dataset.

In [24], it has been shown that the *worst case time complexity* (WCTC) of the conventional CNF to DNF conversion is a square of its *worst case space complexity* (WCSC). Therefore, in this study, we are focusing only on reducing the WCSC of the mentioned conversion. Note that the WCSC of the CNF to DNF conversion is the maximum possible sum of the theoretically achievable numbers of PIs and RPs in this conversion. In [1] and [25], it has been shown that the number of PIs of the DNF of a function of $n$ variables may be no more than $\binom{n}{n/2}$. As for the number of RPs, usually, it is multiple times more than that of PIs [26]. In order to minimize the WCSC of the CNF to DNF conversion as many as possible, we first represent a DF by a $q \times n$ *bit-matrix* (BM) $M_1$ where the $BC_{t\in\{1,2,\ldots,q\}}$ and the attribute $a_{j\in\{1,2,\ldots,n\}}$ are represented by the row $R_t$ and the column $a_j$, respectively. For short, in sequel, instead of the phrase "the WCSC of the conversion of a BM to the DNF," we will use the phrase "WCSC of the BM".

We iteratively partition a BM $M_1$ into the SubBMs (SBMs) such that, firstly, the WCSC of the SBM processed in any iteration cannot exceed the square root of the WCSC of the conversion of the whole DF, and secondly, the number of the generated PIs in any iteration cannot exceed the limit of $\binom{n}{n/2}/2$. The second transformation is based on dividing the RPs generated during the conventional CNF to DNF conversion into two categories [26]: *preventable* RPs (PRPs) and *unpreventable RPs* (URPs). While every PRP can be pre-detected and avoided without generation, every URP can be detected and removed only after it is generated. Fortunately, the ratio between the numbers of UPRs and PRPs is usually very small so that it cannot affect the complexity class of the CNF to DNF conversion. Moreover, the *Algorithm_ILE* developed in Sect. 5 catches and discards each URPs once it is generated. Due to this property of the second transformation, none of the results (intermediate or end) generated during a CNF to DNF conversion can contain any RP. This transformation is also iterative and can reduce the space complexity of the CNF to DNF conversion up to $n-1$ times in each of iterations. We apply it for converting the SBMs produced by the first transformation to the appropriate PIs of the DF represented by the BM.

The paper is organized as follows. In Sect. 2, preliminaries to DF-based AR are given. In Sect. 3, the problem is stated and proved. In Sect. 4, unique properties of the approaches used for converting a BM into the appropriate set of PIs are exposed. In Sect. 5, the algorithm for converting a BM into the appropriate set of PIs and a numerical example demonstrating

the work of the algorithm in detail are given. In Sects. 6 and 7, experimental results and conclusions are given.

## 2 Preliminaries

As it is known, every DF is always represented in *conjunctive normal form* (CNF), which is a conjunction of different disjunctions of $1 \leq k \leq n-1$ attributes [1,8,21,22]. Each of such disjunctions is called a *clause.* The number of attributes composing a clause and the number of clauses composing a DF in CNF are called the *size of the clause* and the *size of the DF* (or the size of the CNF), respectively. For short, we denote the size of a set $X$ by $|X|$.

A *clause* $C_{ij}$ for a dataset with $n$ attributes and $m$ objects is defined as [1,8,19,22]:

$$C_{ij} = \vee_{k=1}^{n} a_k : a_k(u_i) \neq a_k(u_j) \tag{1}$$

where $a_k(u_i)$ and $a_k(u_j)$ are the values of the attribute $a_k$ for the objects $u_{i\in\{1,2,\ldots,m-1\}}$ and $u_{j\in\{i+1,i+2,\ldots,m\}}$, respectively. The DF for a dataset is formed as follows [1,8,10,19]:

$$DF = \wedge_{i=1}^{m-1} \wedge_{j=i+1}^{m} C_{ij} \tag{2}$$

This formula prescribes the logic multiplication (in sequel will be referred simply as multiplication) over all clauses derived from the same dataset. Usually, a DF contains many clauses absorbed (subsumed) by the other ones. We call such clauses as *Redundant Clauses* (RCs) and the ones are not absorbed by other clauses as *Prime Clauses* (PCs). In order to avoid superfluous operations and memory capacity related to RCs, before the computations in (2), the RCs should be removed. A DF without any RC, i.e., containing only PCs, is called a *minimal DF* and denoted by $DF_{\min}$. In [27], it is shown that for datasets with numbers of objects in the range from thousands to tens of thousands, the ratio between |DF| and $|DF_{\min}|$, usually, lies in the range from thousands to millions. Therefore, the algorithm obtaining the $DF_{\min}$, directly, from a dataset should compare each newly generated clause with the already existing ones and remove from them those absorbed by it, or discard it, if it is absorbed by any of them. Such an algorithm is given in [27].

*Example 1* Obtain the $DF_{\min}$ for the dataset in Table 1.

According to (1), the clauses for this dataset are to be as in Table 2. The original and minimized forms of the DF obtained from Table 2 are, respectively, as follows:

**Table 1** A dataset example (adopted from [10])

| $u$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $d$ |
|---|---|---|---|---|---|
| $u_1$ | 1 | 0 | 2 | 2 | 0 |
| $u_2$ | 0 | 1 | 1 | 1 | 2 |
| $u_3$ | 2 | 0 | 0 | 1 | 1 |
| $u_4$ | 1 | 1 | 0 | 2 | 2 |
| $u_5$ | 1 | 0 | 2 | 0 | 1 |

**Table 2** The clauses for the dataset given in Table 1

| $i, j$ | $C_{ij}$ | The status of $C_{ij}$ | $\mathrm{DF}_{\min}$ |
|---|---|---|---|
| 1,2 | $a_1 \vee a_2 \vee a_3 \vee a_4$ | Absorbed by $C_{1,3}$ | $\{C_{1,2}\}$ |
| 1,3 | $a_1 \vee a_3 \vee a_4$ | Absorbed by $C_{1,5}$ | $\{C_{1,3}\}$ |
| 1,4 | $a_2 \vee a_3$ | Prime clause | $\{C_{1,3}, C_{1,4}\}$ |
| 1,5 | $a_4$ | Prime clause | $\{C_{1,4}, C_{1,5}\}$ |
| 2,3 | $a_1 \vee a_2 \vee a_3$ | Absorbed by $C_{1,4}$ | $\{C_{1,4}, C_{1,5}\}$ |
| 2,4 | $a_1 \vee a_3 \vee a_4$ | Absorbed by $C_{1,5}$ | $\{C_{1,4}, C_{1,5}\}$ |
| 2,5 | $a \vee a_2 \vee a_3 \vee a_4$ | Absorbed by $C_{1,5}$ | $\{C_{1,4}, C_{1,5}\}$ |
| 3,4 | $a_1 \vee a_2 \vee a_4$ | Absorbed by $C_{1,5}$ | $\{C_{1,4}, C_{1,5}\}$ |
| 3,5 | $a_1 \vee a_3 \vee a_4$ | Absorbed by $C_{1,5}$ | $\{C_{1,4}, C_{1,5}\}$ |
| 4,5 | $a_2 \vee a_3 \vee a_4$ | Absorbed by $C_{1,5}$ | $\{C_{1,4}, C_{1,5}\}$ |

**Table 3** Description of the dataset by $a_2$ and $a_4$

| $u$ | $a_2$ | $a_4$ | $d$ |
|---|---|---|---|
| $u_1$ | 0 | 2 | 0 |
| $u_2$ | 1 | 1 | 2 |
| $u_3$ | 0 | 1 | 1 |
| $u_4$ | 1 | 2 | 2 |
| $u_5$ | 0 | 0 | 1 |

$$\begin{aligned}\mathrm{DF} &= C_{12}\wedge C_{13}\wedge C_{14}\wedge C_{15}\wedge C_{23}\wedge C_{24}\wedge C_{25}\wedge C_{34}\wedge C_{35}\wedge C_{45}\\ &= (a_1\vee a_2\vee a_3\vee a_4)\wedge(a_1\vee a_3\vee a_4)\wedge(a_2\vee a_3)\wedge a_4\wedge(a_1\vee a_2\vee a_3)\\ &\quad\wedge(a_1\vee a_3\vee a_4)\wedge(a_1\vee a_2\vee a_3\vee a_4)\wedge(a_1\vee a_2\vee a_4)\\ &\quad\wedge(a_1\vee a_3\vee a_4)\wedge(a_2\vee a_3\vee a_4)\\ \mathrm{DF}_{\min} &= C_{14}\wedge C_{15} = a_4\wedge(a_2\vee a_3)\end{aligned}$$

As it is seen from Table 2 and these expressions, while $|\mathrm{DF}| = 10$, $|\mathrm{DF}_{\min}| = 2$. Since a reduct of a DF is a product of the variables the DF depends on, we should obtain the DNF of $\mathrm{DF}_{\min}$ by expanding it as follows:

$$\mathrm{DF}_{\min} = a_4(a_2\vee a_3) = a_2a_4\vee a_3a_4$$

where each of the products $a_2a_4$ and $a_3a_4$ is a reduct of the dataset given in Table 1. The descriptions of the dataset by these reducts are given in Tables 3 and 4, respectively.

For simplicity and clarity of operations on the clauses, in [24,26,27], each clause, $C_{ij}$, given by (1) is represented by a *bit-clause* (BC) $B_{ij}$ defined as:

$$B_{ij} = (b_{ijk})_{k=1}^{n} \tag{3}$$

where $b_{ijk} = 1$ if $a_k(u_i) \neq a_k(u_j)$ and $b_{ijk} = 0$ otherwise, and $k$ is the position of the bit at hand in the $B_{ij}$. As it is seen from (2), for a dataset with $m$ objects, exactly $Q = m(m-1)/2$ BCs are generated. They compose a set

$$S = \{B_{ij}\},\ i = 1, 2, \ldots, m-1,\ j = i+1, i+2, \ldots, m \tag{4}$$

**Table 4** Description of the dataset by $a_3$ and $a_4$

| $u$ | $a_3$ | $a_4$ | $d$ |
|---|---|---|---|
| $u_1$ | 2 | 2 | 0 |
| $u_2$ | 1 | 1 | 2 |
| $u_3$ | 0 | 1 | 1 |
| $u_4$ | 0 | 2 | 2 |
| $u_5$ | 2 | 0 | 1 |

**Table 5** The BC-based representation of Table 2

| $i, j$ | $B_{ij}$ | The status of $B_{ij}$ | $S_{\min}$ |
|---|---|---|---|
| 1,2 | 1111 | Absorbed by $B_{1,3}$ | $\{B_{1,2}\}$ |
| 1,3 | 1011 | Absorbed by $B_{1,5}$ | $\{B_{1,3}\}$ |
| 1,4 | 0110 | Prime bit-clause | $\{B_{1,3}, B_{1,4}\}$ |
| 1,5 | 0001 | Prime bit-clause | $\{B_{1,4}, B_{1,5}\}$ |
| 2,3 | 1110 | Absorbed by $B_{1,4}$ | $\{B_{1,4}, B_{1,5}\}$ |
| 2,4 | 1011 | Absorbed by $B_{1,5}$ | $\{B_{1,4}, B_{1,5}\}$ |
| 2,5 | 1111 | Absorbed by $B_{1,5}$ | $\{B_{1,4}, B_{1,5}\}$ |
| 3,4 | 1101 | Absorbed by $B_{1,5}$ | $\{B_{1,4}, B_{1,5}\}$ |
| 3,5 | 1011 | Absorbed by $B_{1,5}$ | $\{B_{1,4}, B_{1,5}\}$ |
| 4,5 | 0111 | Absorbed by $B_{1,5}$ | $\{B_{1,4}, B_{1,5}\}$ |

Since every $B_{ij}$ is a bitwise representation of a certain propositional clause $C_{ij}$, a set $S$ has as many redundant BCs as the represented by its propositional DF with redundant clauses. For example, the BC-based representation of Table 2 is given in Table 5, where attributes and bit positions in BCs are associated by the following *attribute-bit* structure.

$$\textit{Attribute-bit-str} = \{a_1 : 1; a_2 : 1; a_3 : 1; a_4 : 1\} \tag{5}$$

In this structure, $a_k : 1$ means that, in BCs, the attribute $a_k$ is represented by a single bit in the $k$th position.

In Table 5, a redundant BC (RBC) is obtained as follows: Let $A$ and $B$ be different BCs. $A$ is absorbed by $B$ ($A$ is a RBC) if $A\&B = B$ and $B$ is absorbed by $A$ ($B$ is a RBC) if $A\&B = A$. If the result of $A\&B$ is neither $A$ nor $B$, then none of these BCs is a RBC. Based on this rule, we obtain from Table 5 that $S_{\min} = \{B_{1,4}, B_{1,5}\} = \{0110, 0001\}$. According to the structure in (5), this result should be interpreted as a set of reducts $a_2a_4$ and $a_3a_4$.

Note that, removing the RBCs from the set $S$ disorders the indices of the BCs remaining in $S_{\min}$. Therefore, for indexing the BCs in $S_{\min}$, we use a single variable $t = 1, 2, \ldots, q \leq Q$ and rewrite (1), (4) and (2), respectively, as follows:

$$B_t = (b_{tk})_{k=1}^{n} \tag{6}$$

$$S_{\min} = \{B_t\}_{t=1}^{q} \tag{7}$$

$$W = |_{t=1}^{q} E(B_t) \tag{8}$$

where $W$ is the bitwise representation of a *DF*, the "|" denotes the sign of the bitwise OR operation used for performing the AND operation on the pairs of BCs [24,26,27], and $E(B_t)$ is an operator expanding a $B_t$ into the set of associated *unit BCs* as follows:

$$E\,(B_t) = \{Pr_k\,(B_t)\,|b_{tk} = 1\} \tag{9}$$

where $Pr_k(B_t)$ is the *projection* of $B_t$ on the bit-position $k$ that gives a *unit BC* with a single 1 in the $k$th bit-position and 0's in all the others. For instance, $B_t = 1011001$, $E(B_t) = \{Pr_k(B_t)|b_{tk} = 1\} = \{1000000, 0010000, 0001000, 0000001\}$.

## 3 Problem statement

As discussed before, every DF is considered as a Boolean function where every variable represents an attribute of the dataset to which the DF belongs. Since every variable in a DF represents a certain attribute occurring only in the *positive* (uncomplemented) form, it may be simply considered as a *literal*. Therefore, in this study, the concepts of *attribute*, *variable*, and *literal* are interchangeably used.

Recall that according to DF-based AR, each reduct is obtained as one PI of the DF, and for obtaining the PIs of a DF, it should be converted from CNF to DNF [1,8,19,21]. As it is known, all CNF to DNF conversion algorithms are based on the *Nelson's* theorem [15,28–31] stating that "*all PIs of a Boolean function given in CNF can be obtained by straightforward multiplying its clauses and deleting all RPs from the result*" [29]. As it is easy to understand, the Nelson's CNF to DNF conversion is an *exhaustive process* during which no any minimization of intermediate results is done. We will refer to such a conversion as an *exhaustive conversion.* In the process of such conversion of a DF with $n$ variables, totally $\Omega_{EC}\,(n) = \prod_{i=1}^{q} w(c_i) \approx w(c)^q$ products are generated [24,30], where $q$ is the size of the DF, $w(c_i) < n$ is the size of the clause $C_i$, $w(c) < n$ is the average size per clause, and $\Omega_{EC}(n)$ is the space complexity of the exhaustive CNF to DNF conversion. On the other hand, the number of PIs of a logic function of $n$ variables may be as large as $3^n/n$ [29]. This is to say that the number of RPs generated in the DF to DNF conversion process may be as large as $R_{\max} = w(c)^q - 3^n/n$. But while this upper bound for $R_{\max}$ is valid for a generic Boolean function of $n$ variables, it is very surplus for a DF in which every attribute appears only in the positive form. Namely, every DF is a *completely monotonic function* (CMF). Such a specification of a DF is very beneficial because the number of PIs of a CMF of $n$ variables is limited by $\binom{n}{n/2}$ [1,25] which is much smaller than $3^n/n$ for all $n > 9$. That is, for a CMF, $R_{\max}$ may be as large as $w(c)^q - \binom{n}{n/2}$. The analysis of this formula shows that $w(c)^q$ reaches its maximum value at $w(c) = n/2$ and $q = \binom{n}{n/2}$. Namely,

$$R_{\max} = (n/2)^{\binom{n}{n/2}} - \binom{n}{n/2} \tag{10}$$

Moreover, a DF considered as a CMF may be specified in the binary space $\{0, 1\}^n$ of the complexity $2^n$ instead of the ternary one $\{0, 1, x\}^n$ with the complexity $3^n$ needed for specifying a generic function of $n$ variables. The computations by (10) show that even for $n = 7$, the $R_{\max}$ for a CMF could be as large as $10^{19}$. This is because with every increase of $n$ by 1, the exponent of $R_{\max}$ increases more than the double. For instance, the increment in $n$ from 9 to 10 causes an increment in the exponent of $R_{\max}$ from 82 to 176. While modern computers cannot afford a set of the size of the order of $10^{19}$, that would take place even at $n = 7$, they have to process the datasets with $n \gg 7$. Briefly speaking, the main factor complicating the CNF to DNF conversion is the excessively high WCSC of this problem that usually even exceeds the virtual memory capacity of modern computers. Therefore, reducing the WCSC of the mentioned conversion is one of the actual problems in logical data processing.

In order to reduce the WSCS of the CNF to DNF conversion, a number of attempts were made. In particular, the algorithm proposed by *Slagle and et al*., [32] represents a function in CNF by a semantic tree, considers each prime path in it as a PI of the function, and finds the PIs by a depth-first searching in the tree. Unfortunately, this algorithm usually performs superfluous operations as it would generate some PIs more than once and present some RPs as PIs [31]. Another search tree (ST)-based CNF to DNF conversion method is the *Thelen's* approach [33–35]. A *Thelen's* ST is built such that in its every level, one clause represented by the CNF is processed. Therefore, a ST representing a CNF of size $q$ has exactly $q$ levels. Every node in the level $i$ of the ST have as many outgoing arcs as many literals in the clause $C_i$ processed in this level, and each arc outgoing from every node of the level $i$ is labeled by one of literals of $C_i$. In such a ST, conjunction of all literals labeling the arcs at the path from the root to a node is considered as the product to be represented by this node. Therefore, each leaf node of such a ST represents a product that either a PI or a RP. In order to prevent the growth of the number of RPs, a ST is pruned in each of its levels by the following rules [34,35]:

**R1:** An arc is pruned if its predecessor node (product) contains the complement of the arc-literal,

**R2:** A clause is discarded if it contains a literal, which appears also in the predecessor node (product),

**R3:** An arc is pruned if another non-expanded arc on a higher level still exists, which has the same arc-literal.

The rule $R1$ has no meaning for DFs since, as a CMF, every DF always contains only the positive variables. In terms of switching functions, *Thelen's* approach begins with a root node representing a 1. In order to build each next level of a ST, the algorithm compares the next clause to be embedded into the ST with each of products represented by the nodes in the most recently built level. While the clause is multiplied by every product that does not contain any literal from it, it is discarded for every product that contains at least one literal from it (Rule 2). This process is iteratively continued until all clauses of the DF are processed (embedded into the ST). In result, the algorithm gives all possible PIs and some URPs that could not be recognized by rule $R2$. These URPs are removed from the result by rule $R3$. Note that both rules $R2$ and $R3$ realize the same *logic absorption law* given below.

$$\hat{P} \cap \hat{C} = \hat{P} \rightarrow P \wedge C = P \tag{11}$$

where $P$ is a product, $C$ is a clause and $\hat{P}$, and $\hat{C}$ are the sets of literals present in $P$ and $C$, respectively. Since (11) is the rule for the *irredundant logic expansion* (ILE) of a product $P$ by a clause $C$, we will refer to an approach based on this rule as an ILE. As it is stated in [35], in spite of iteratively pruning of the ST, the WCSC of the ILE is exponential in $n$. This is because of rule $R2$, according to which every clause to be embedded into a ST, should be compared with all already computed products, the number of which may be as large as $\binom{n}{n/2}$ that is of order of $2^n$. In [36], *Socher* proposed a $n \times q$ *binary matrix* (BM)-based approach where the row $k_{\in}\{1, 2, \ldots, n\}$ and the column $t \in \{1, 2, \ldots, q\}$ are labeled by the literal (attribute) $a_k \in A$ and by the BC $B_t \in S_{\min}$, respectively. In such a BM, the entry $(k, t)$ is always associated with the bit $b_{tk} = Pr_k(B_t)$. For example, the BM representation of the function $F = (a \vee c)(b \vee c)(b \vee d \vee f)(b \vee e \vee f)$ is to be as in Table 6.

The mentioned approach is based on the *divide-and-conquer* strategy according to which the given BM is processed iteratively and its remainder to be processed in the $i$th, $i \leq n-1$, iteration is denoted by $M_i$ (the original BM is denoted by $M_1$). According to this approach, in the $i$th iteration, a *split literal* $SL_i$ is chosen and the $M_i$ is partitioned into two sub-BMs

**Table 6** The BM representation of the function $F$

| | $B_1$ | $B_2$ | $B_3$ | $B_4$ |
|---|---|---|---|---|
| $a$ | 1 | 0 | 0 | 0 |
| $b$ | 0 | 1 | 1 | 1 |
| $c$ | 1 | 1 | 0 | 0 |
| $d$ | 0 | 0 | 1 | 0 |
| $e$ | 0 | 0 | 0 | 1 |
| $f$ | 0 | 0 | 1 | 1 |

(SBMs) so that while one of the SBMs, denoted by $D_i$, contains only those columns of the $M_i$ for which $SL_i = 0$, another SBM, denoted by $M_{i+1}$, contains all the rows of BM except the row labeled by the $SL_i$. The SBM $D_i$ is processed at once in order to obtain all PIs that can be derived from it. But the SBM $M_{i+1}$ is passed to the next iteration as the remainder of the BM $M_1$ to be processed in it. In sequel, we will call this method as the *sequential logic partitioning* (SLP). In short, a SLP-based algorithm discovers the PIs of a DF by generating all paths through the BM, discarding subsumed paths and considering each preserved path as a PI [31]. Since at each iteration of the SLP one SL is chosen, the number of iterations of a SLP-based algorithm cannot exceed the limit of $n - 1$. Unfortunately, this approach performs superfluous operations as it generates some part of products more than once and carries out some RPs [31,37]. In [37], an improvement of the SLP according to which the BM is partitioned into several cofactors is proposed. The subsets of PIs for all cofactors are obtained separately and then cross-concatenated. There are several other methods for CNF to DNF conversion reviewed in [31,37] but not so attractive for the conversion of a DF to DNF. In our opinion, the algorithms proposed in the abovementioned studies suffer from the following drawbacks making them not so convenient for converting a DF from CNF to DNF:

1. The algorithms have been developed for logic minimization in the space $\{0, 1, x\}^n$ that is much wider than the space $\{0, 1\}^n$ enough for DFs,
2. Every algorithm fixes all paths existing in the BM or ST, removes redundant paths from the result and considers each of remaining paths as a PI. But since the number of RPs for a function is usually much greater than that of PIs, the algorithm would generate so many RPs that may overflow the memory,
3. The computational complexities of the algorithms have not been satisfactorily estimated,
4. Search for PIs in a BM or in a ST is more difficult than their computation by switching functions techniques, where the PIs are calculated by logic transformations on the input cubes (vectors specified in the space $\{0, 1, x\}^n$).

The contributions of this paper include:

- We show that every DF is exactly a CMF and can be processed by much simpler way than when it is considered as a generic function,
- We show that in the SLP the number of iterations cannot exceed $n - 1$ and the most complex iteration is the first one with a space complexity no higher than the square root of that of the original BM,
- We show that the WCSC of the ILE applied is $\binom{n}{n/2}/2$,
- We show that SLP and ILE may be combined in a single algorithm so that the WCSC of the CNF to DNF conversion reduces from $\Omega_{EC}(n) = (n/2)^{\binom{n}{n/2}}$ to $\Omega_{ILE}(n-1) = \binom{n}{n/2}/2$,

where the $\Omega_{EC}(n)$ and $\Omega_{ILE}$ are WCSCs of the exhaustive and ILE-based CNF to DNF conversions, respectively.

## 4 Unique properties of SLP and ILE

In our opinion, SLP and ILE have some unique properties that allow us to combine them in a single algorithm more efficient than each of them alone. The unique properties of the SLP are as follows:

1. It iteratively (sequentially) partitions the BM so that at each iteration only one SBM with the WCSC much less than the original BM is processed,
2. The number of PIs generated in each iteration cannot exceed the half of the number of all PIs. This property is very useful, especially, in cases where $n$ is a big number and the number of PIs is close to the maximum possible one; that is almost exponential in $n$.

The unique property of the ILE is that it can *prevent* the generation of the vast majority of RPs by using Rule 2 given above. As for the remaining few RPs referred to above as URPs, they can be detected and deleted by the logic absorption law in (11) at the end of constructing every level of the ST being built. In our opinion, the ILE may be considered as an improvement of the *Nelson's* method in the sense that it, additionally, can avoid the generation of all PRPs. Therefore, we obtain the WCSC for the ILE via the WCSC of the *Nelson's* method.

### 4.1 The WCSC of the Nelson's method

As mentioned above, in the *Nelson's* method, the minimized DNF (disjunction of all PIs) of a Boolean function given in CNF is obtained by straightforward multiplying its clauses and removing all RPs from the result. If the clauses of a CNF are $C_1, C_2, \ldots, C_q$ then the *Nelson's* algorithm can be written as given in Fig. 1.

In order to convert a DF to the DNF, the *Algorithm_ON* (Fig. 1) multiplies all clauses of the DF together regardless of which there may be generated a large number of RPs. Above, such conversion has been referred to as exhaustive one and its WCSC has been obtained as $\Omega_{EC}(n) = \prod_{i=1}^{q} w(c_i)$. Since for the worst case $q = \binom{n}{n/2}$ and $\forall i \in \{1, 2, \ldots, q\},\ w(c_i) = n/2$, the WCSC of the *Algorithm_ON* is to be as:

$$\Omega_{ON}(n) = (n/2)^{\binom{n}{n/2}} \tag{12}$$

while processing such a big number of clauses is beyond the power of modern computers for $n \geq 7$, the number of PIs of a function of $n$ variables cannot exceed the number $q = \binom{n}{n/2} \ll \Omega_{EC}(n)$. However, the WCSC of the DF to DNF conversion can be reduced in a large scale if statement 2 in Fig. 1 to be included into the *For* loop as in Fig. 2.

In the iteration $t \in \{0, 1, 2, \ldots, q\}$ of the *Algorithm_IN* (Fig. 2), first the set of products $W$ is expanded by multiplying it with the clause $C_t$ and second all RPs present in $W$ are

**Fig. 1** The original Nelson's algorithm

```
Algorithm_ON(C1, C2 , ... , Cq)
    W=1
    1. For t=1 to q
           W=W∧ Ci
       End for
    2. Remove from the set W all elements
       absorbed by at least one other element in it
End
```

**Fig. 2** The improved Nelson's algorithm

```
Algorithm_IN(C1, C2 , ... , Cq)
    W=1
    For t=1 to q
        1. W=W ∧ Ci
        2. Remove from the set W all elements
           absorbed by at least one other element
           in it
    End for
End
```

removed. Namely, no RP is passed to the next $(t+1)$th iteration. According to the algorithm, every iteration begins with a products set $W$ of size $|W| \leq \binom{n}{n/2}$. Statement 1 increases the size of the set $W$ by a factor $n/2$ and statement 2 reduces it up to $|W| \leq \binom{n}{n/2}$. Therefore, WCSC of the *Algorithm_IN* is as follows:

$$\Omega_{IN}(n) = (n/2) \times \binom{n}{n/2} \tag{13}$$

As it is easy to see, the WCSC of the *Algorithm_ IN* (Fig. 2) is $R = \Omega_{ON}(n) / \Omega_{IN}(n) = (n/2)^{\binom{n}{n/2}} / ((n/2) \times \binom{n}{n/2})$ times less than that of the *Algorithm_ON*. For example, for $n = 4$, 5, 6, 7, 8, 9 and 10, $R = 5.3, 3.8 \times 10^2, 5.8 \times 10^7, 9.0 \times 10^{16}, 5.0 \times 10^{39}, 3.5 \times 10^{79}$ and $1.1 \times 10^{173}$, respectively. In spite of the reduction in these large scales, the WCSC of the *Algorithm_ IN* remains in the class of exponential complexity because $(n/2) \times \binom{n}{n/2}$ may be specified as $O(2^n)$, i.e.,

$$O(\Omega_{IN}(n)) = O(2^n) \tag{14}$$

However, fortunately, the space complexity of real tasks is much less than the mentioned WCSC, and hence, the conversion of a DF to DNF is applicable for datasets with several tens of attributes.

### 4.2 The WCSC of the ILE

Recall that according to the ILE, while a clause to be inserted (embedded) into the ST is multiplied by every product that does not contain any literal from it, it is neglected for every product that contains at least one literal from it (Rule R2). This process is continued iteratively until all clauses of the DF at hand are embedded into the ST. As it is shown in [26] and [34], rule $R2$ may detect and prevent the generation of most RPs in each level of the ST. But the number of RPs that cannot be detected by rule $R2$ remains not only unknown but also may grow from level to level. This problem can be avoided if rule $R3$ is applied not at the end of the algorithm but at the end of building each level of a ST.

In the worst case, the DF has $q = \binom{n}{n/2}$ clauses with $n/2$ literals per clause and each literal appears exactly in $\binom{n}{n/2}/2$ products of DNF of the DF. This is to say that in expansion (multiplication) of a products set $W$ with a clause $C$ by the ILE, the maximum possible number of expanded and unexpanded elements of the set $W$ are the same and are equivalent to $\binom{n}{n/2}/2$, i.e., the WCSC of the ILE is to be as

$$\Omega_{ILE}(n) = \binom{n}{n/2} \tag{15}$$

From (13) and (15), it is seen that the amount of intermediate products produced by an ILE-based algorithm is to be $n/2$ times less than that of the *Improved Nelson's* algorithm. This is to say that in many cases when *Improved Nelson's* algorithm can overflow the memory, an ILE-based algorithm can work easily. In spite of this, $\Omega_{ILE}(n)$ belongs to the same complexity class $O(2^n)$ as $\Omega_{ILE}(n)$. However, the ILE is applicable for datasets with dozens of attributes

because, firstly, the space complexities of the real AR tasks are usually much less than the mentioned WCSC and secondly, the SLP applied to the BM before the ILE may seriously reduce the work to do by it.

### 4.3 The WCSC of the SLP

Since the SLP was developed for solving the problems with a few variables such as theorem proving [36,37], it has not been subjected to a serious *computational complexity* analysis. But in our opinion, since this method may be applied to the processing of DFs with dozens of attributes, the mentioned analysis is necessary. For this aim, consider the BM structure given in [24] that is a 90 degrees right-rotated form of the conventional BM. In such a BM, every row and column is, respectively, labeled with one BC and one certain literal of the DF it represents. Since the location of such a BM in the memory is almost the same with its logic structure, it may be processed by a more easy way. Moreover, every BM of this structure may be subjected easily to the *commutative decomposition* according to which every BM can be partitioned into *smaller* and *independent SBMs* without losing its original features [38]. In our opinion, from the related approaches, the approach that most satisfies this criterion is the *Socher's* approach [36] summarized above (Sect. 3). In [24], two more important concepts for processing the BMs have been introduced. They are the *weight of an attribute* and *the weight of a BC* obtained as the numbers of 1's in the column and in the row labeled by the attribute and the BC, respectively. While the weights of the attributes are used for selecting the best suited SL for current iteration, the weights of the BCs are used for obtaining the space complexity of the exhaustive processing of BM. According to [22,24,34,36], as the SL for every iteration the attribute of the highest weight in the BM to be processed in this iteration is chosen. If there are more than one such attribute, any one of them may be chosen. An algorithm implementing the SLP is given in Fig. 3.

In the *Algorithm_SLP* (Fig. 3), $R_t$ is the $t$th row of the SBM $M_i$, $W_i$ is the subset of prime paths derived from the SBM $M_i$, and $W$ is the union of the subsets $W_1, W_2, \ldots, W_{i-1}$. In order to derive the SBM $D_i$ from the SBM $M_i$, first the $M_i$ is considered as a set and second the $D_i$ is derived from it by the statement $A2$.

As it is easy to see from Fig. 3, the *Algorithm_SLP* processes the original BM $M_1$ iteratively so that, in the $i$th iteration, all paths containing the $SL_i$ (split literal for the $i$th iteration) but not containing any literal from the set $\{SL_I\}_{I=1}^{i-1}$ are generated (see the statement A3), where $i \in \{1, 2, \ldots, n-1\}$. This is to say that in the $i$th iteration, the algorithm processes a BM with $n - i + 1$ literal including the $SL_i$. Generally speaking, the algorithm begins with a BM of

***Algorithm_SLP ($M_1$)***
*i=1*
***Until $M_i$ is a zero row***
A1. *Select the split literal $SL_i$ for $M_i$*
A2. *$D_i$ ={$R_t$: $R_t \in M_i$ and $R_t(SL_i)$=0}*
A3. *$M_{i+1}$ =$M_i$ with the column $SL_i$ cleared and the redundant rows (BCs absorbed by other ones) removed*
A4. *Obtain the set $W_i$ of all paths in $D_i$*
A5. *Remove all redundant paths from $W_i$*
A6. *Concatenate each path in $W_i$ with the $SL_i$*
A7. *i=i+1*
***End Until***
A8. *$W=\{W_j\}_{j=1}^{i-1}$ ; Consider each element of W as a PI of the DF*
*End*

**Fig. 3** The algorithm of the SLP

$n$ literals, reiterates with removing one literal in every iteration from the BM and ends when there a SBM $M_i = 0$ (a SBM with single zero row) appears. This is to say that for a BM with $n$ columns (literals) the *Algorithm_SLP* will do at most $n-1$ iterations. But the number of BCs in the SBM $D_i$ processed in the $i$th iteration and the number of PIs generated in the same iteration may be as large as $\binom{n-i}{(n-i)/2}$. It is easy to see that with each increment in the value of $i$ by 1, the value of $\binom{n-i}{(n-i)/2}$ decreases approximately by the half. In other words, the WCSC of the $(i+1)$th iteration of the *Algorithm_SLP* is to be approximately two times less than that of $i$th iteration.

As mentioned above, in the $i$th iteration, the BM $M_i$ is partitioned into the SBMs $D_i$ and $M_{i+1}$. While the $M_{i+1}$ is passed to the next iteration, the $D_i$ is processed at once to find PIs derivable from it.

In the $i$th iteration, the algorithm *Algorithm_SLP* (Fig. 3) converts from CNF to DNF a SBM $D_i$ with $n-i+1$ nonzero columns (literals) and no more than $\binom{n-i}{(n-i)/2}$ rows (BCs). Since $max\left\{\binom{n-i}{(n-i)/2}\right\} = \binom{n-i}{(n-i)/2}$, among the SBMs $D_1, D_2, D_3, \ldots, D_{n-1}$ processed in the 1st, 2nd, …, $(n-1)$th iterations of the algorithm, respectively, the one with the highest WCSC is the $D_1$. Therefore, below we will focus only on the WCSC of the 1st iteration of the *Algorithm_SLP*.

As it is well known, the most complex BM representing a DF with $n$ literals is a BM with $q = \binom{n}{n/2}$ rows, $n$ columns, exactly $w(c) = n/2$ 1's per row and exactly $\binom{n}{n/2}/2$ 1's per column [1,24,25]. The WCSC of the exhaustive conversion of such a BM to DNF is as of *Algorithm_ON* (Fig. 2), i.e.,

$$\Omega_{EMC_1}(n) = \Omega_{ON}(n) = (n/2)^{\binom{n}{n/2}} \tag{16}$$

As mentioned above in the first iteration of a SLP-based algorithm, to DNF is converted not the whole of the original BM $M_1$ but only its SBM $D_1 = \{R_t | R_t \in M_1 \text{ and } SL_1(R_t) = 0\}$, where $R_t$ is the label of the row representing the $BC_t$. Since in such a BM each column (attribute) has exactly $\binom{n}{n/2}/2$ 0's, the SBM $D_1$ will have exactly $\binom{n}{n/2}/2$ rows with the following WCSC of the exhaustive conversion,

$$\Omega_{ECD_1}(n) = (n/2)^{\binom{n}{n/2}/2} = \sqrt{\Omega_{EMC_1}(n)} \tag{17}$$

Namely, the SLP reduces the WCSC of the exhaustive processing of a BM at least by $\sqrt{\Omega_{EMC_1}(n)}$ times. The efficiency of this reduction increases almost in a quadratic behavior with every increase in the value of $n$ by 1. For instance, for $n = 7$, 8, 9 and 10, while the orders of the values of $\Omega_{EMC_1}(n)$ are $10^{19}$, $10^{42}$, $10^{82}$ and $10^{176}$, the orders of the values of $\Omega_{ECD_1}(n)$ are $10^{9}$, $10^{21}$, $10^{41}$ and $10^{88}$, respectively. In spite of this, the space complexity of the mentioned conversion still remains very high for all $n \geq 8$. However, the SLP has the following advantages that may be considered as a base for developing more efficient algorithms: 1) The WCSC of the approach cannot exceed the square root of that of the BM in whole, 2) The number of PIs that may be generated in one iteration of the approach cannot exceed the half of the maximum possible number of all PIs, 3) The number of iterations of the approach cannot exceed $n$-1.

## 5 The combined SLP-ILE approach

As mentioned above, the SLP allows us to reduce the WCSC of the exhaustive processing of a BM from $\Omega_{EMC_1}(n) = (n/2)^{\binom{n}{n/2}}$ to $\Omega_{ECD_1}(n) = \sqrt{\Omega_{EMC_1}(n)}$ and the maximum

possible number of simultaneously generated PIs from $\binom{n}{n/2}$ to $\binom{n-1}{(n-1)/2} \approx \binom{n}{n/2}/2$. As for ILE, it can reduce the WCSC of the mentioned processing from $\Omega_{ECD_1}(n) = \sqrt{\Omega_{EMC_1}(n)}$ to $\Omega_{ILE}(n) = \binom{n}{n/2}/2$. That is, the SLP and ILE may be combined together such that the WCSC of CNF to DNF conversion of a DF can be reduced from $(n/2)^{\binom{n}{n/2}}$ to $\binom{n}{n/2}/2$. In order to do this, it is sufficient to replace the statements $A4$ to $A6$ of the *Algorithm_SLP* (Fig. 3) with a procedure implementing the ILE. But since the most efficient system of operations for processing a BM is the *bitwise-logic* [16,24], we should implement the ST-based ILE in terms of this logic.

### 5.1 The bitwise implementation of the ILE

In our opinion, the *ILE* is based on the *commutative decomposition* principle and *logic absorption law*. According to this principle, a function is partitioned into smaller and independent sub-functions without losing its original features [38]. For example, the logic expression $f$ consisting of a multiplication of a product $P$ by a clause $C = (x_i \vee x_j \vee \cdots \vee x_h)$ may be expressed as follows:

$$f = P\left(x_i \vee x_j \vee \cdots \vee x_h\right) = x_i P \vee x_j P \vee \cdots \vee x_h P \tag{18}$$

where $i < j < h < n$. In order to explain the application of the logic absorption law to a function like (18), let us to denote $x_i P, x_j P, x_k P, \ldots, x_h P$ by $F_i \vee F_j \vee F_k \vee \cdots \vee F_h$, respectively. For every $F_{g\in\{i,j,k,\cdots,h\}}$, rule $R2$ (Sect. 3) may be rewritten as:

$$\text{if } x_g \in \hat{P} \text{ then } F_g = P \text{ else } F_g = x_g P \tag{19}$$

where $\hat{P}$ is the set of literals present in $P$. The bitwise representation of (19) is as follows:

$$\text{if } BP_i \& B_t \neq 0 \text{ then } F_g = BP_i \text{ else } F_g = BP_i | E(B_t) \tag{20}$$

where $BP_i$ is a *bit-product*, $B_t$ is a BC, $E(B_t)$ is the set of unit BCs generated according to (8) and "|" is the sign of *bitwise OR* operation used for bitwise ANDing the BPs and unit BCs. Note that the BP representation of a product $P$ contains 1's and 0's in the positions of the literals present and not present in $P$, respectively.

Equation (20) says that "*if* $BP_i$ & $B_t \neq 0$ then $BP_i$ *may be preserved* (*as a candidate to be a PI*) *without any operation on it, else it must be expanded by the* $B_t$". Based on this corollary, our main procedure given by (8) and to be realized by the ILE can be expressed as the following system of equations:

$$V_{t1} = \{BP : BP \in V_{t-1} \, and \, BP \& B_t \neq 0\}; V_{t2} = V_{t-1} - V_{t1} \tag{21}$$

$$E(B_t) = \{b_{tk} : b_{tk} = Pr_k(B_t) \, and \, b_{tk} = 1\} \tag{22}$$

$$Q = V_{t2} | E(B_t) \tag{23}$$

$$V_{t3} = \left\{h : h \in Q \, and \, \exists z \in V_{t1} : z \& h = z\right\} \tag{24}$$

$$V_t = V_{t1} \cup V_{t3}. \tag{25}$$

As it can be seen from (21–25), while the set $V_{t1}$ consists of such elements from the set $V_{t-1}$ are to be included into the set $V_t$ without any expansion, the set $V_{t2}$ contains those elements from $V_{t-1}$ are to be expanded by $B_t$ (multiplied by $E(B_t)$) and then included into the set $V_t$. This is to say that, no element from $V_{t1}$ may contain more literals than any element from the set $V_{t3}$. In the other words, some elements from $V_{t3}$ may be absorbed by some elements from $V_{t1}$ but not vice versa, i.e., the $V_{t3}$ may contain redundant BPs to instantly discard that each new element generated for $V_{t3}$ should be controlled, whether it is absorbed by any element

**Fig. 4** The bitwise algorithm for the ILE

```
Algorithm_ILE (Di)
  H1. V0={B0}
  For t=1 to qi // qi the number of rows of the Di
      H2. Vt1 = {P:P ∈ Vt-1 and P&Bt≠0}; Vt2= Vt1 - Vt-1
      H3. E(Bt)={btk:btk=Prk (Bt) and btk=1}; Vt3=∅
      H4. For k=1 to |Vt2|
          H5. For s=1 to |E(Bt)|
              H6. hks = gk |us  //gk∈ Vt2 and us ∈ E(Bt)
              H7. If ∃ z:z∈ Vt1 and z&hks = z
                  Then Vt3= Vt3∪hks
          End for
      End for
      H8. Vt = Vt1 ∪Vt3
  End for
  Return (Wi =Vt)
End
```

```
Algorithm_SLP_ILE (M1)
  Order the rows M1 by the weight of the rows; i=1
  Until Mi is a zero bit-string
      A1.1. Select SLi for Mi
      A1.2. Form a unit BC B0 with a 1 in the position of SLi and 0's in all others positions
      A2. Di = B0 ∪ {Rt: Rt ∈Mi and Rt (SLi)=0 }
      A3. Mi+1 =Mi with the column SLi cleared and the redundant (absorbed) rows removed
      Call Algorithm_ILE (Di) // Output Wi
      A7. i=i+1; End Until
      A8. W ={Wj}, j=1 to i-1
      A9. Consider each element of W as a PI of the DF
End
```

**Fig. 5** The *Algorithm_SLP_ILE*

from $V_{t1}$ or not. That is, the set of all non-redundant BPs generated in the $t$th level of a ST will be obtained and included into the set of prime BPs by (24) and (25), respectively. Equations (21–25) can be implemented by the *Algorithm_ILE* (Fig. 4), where $B_0$ is the unit BC with a 1 in the position of the $SL_i$ and 0's in all other positions. In the algorithm, the contribution of a SL has been realized indirectly via the $B_0$. This allows us to process every BM (or SBM) by only parallel bitwise operations on its rows.

In the *Algorithm_ILE* (Fig. 4), statement $H1$ initiates the set $V_0$, statement $H2$ implements (21), statement $H3$ implements (22) and initiates the set $V_{t3}$, statements $H4$ to $H7$ implement (23) and (24) with discarding each unpreventable BP once it is generated. Statement $H8$ implements (25).

## 5.2 The SLP-ILE algorithm

As mentioned above, the SLP iteratively (sequentially) partitions a BM to SBMs so that the WCSC of a SBM processed in any iteration is much less than that of the original BM. Since in this approach, every SBM is transformed into the appropriate subset of PIs by conventional CNF to DNF conversion, the processing of SBMs with high dimensions may overflow the computer memory. As mentioned above, the main factor complicating CNF to DNF conversion is that, besides the essential products, there are usually numerous RPs are generated. We can avoid the generation of these RPs by replacing the statements $A4$ to $A6$ in the *Algorithm_SLP* (Fig. 3) with the *Algorithm-ILE* (Fig. 4) as shown in Fig. 5.

As mentioned before, the most complex SBM generated by the statements *A1.1*, *A1.2*, *A2*, and *A3* of the *Algorithm_SLP_ILE* is the SBM $D_1$ with $n-1$ literals except the SL. According to (15), this SBM is processed by the *Algorithm_ILE* with the WCSC $\binom{n-1}{(n-1)/2} \approx \binom{n}{n/2}/2$. Albeit this WCSC is almost exponential in $n$, the probability of occurrence of a function with this WCSC is as small as $1/((n-1)! + n^{n-2})$, where $((n-1)! + n^{n-2})$ is the number of all possible minimal forms of a function of $n$ variables [39]. In particular, this probability becomes less than 0.001 even at $n = 6$ and decreases more than $2n$ times with every increment of $n$ by 1.

*Example* For an example, we select the dataset *Diabet* [40] with $m = 768$ objects and $n = 8$ attributes denoted by $a_1, a_2, \ldots, a_8$. This dataset was preferred due to that it is the smallest of the available to us real datasets suitable for demonstrating all details of the proposed algorithm.

For the dataset *Diabet* were generated $Q = 294568$ BCs from which at most $q = \binom{n}{n/2} = 70$ are to be prime BCs. But in this example, fortunately $q = 15 < 70$. Namely, there 294553 BCs have been removed as those absorbed by 15 prime BCs (PBCs) denoted by $R_1, R_2, \ldots, R_{15}$ and represented as the BM $M_1$ given in Table 7.

In the BM $M_1$ (Table 7), the elements of the extra row $w(a)$ and extra column $w(R)$ contain the *weights* of the attribute $a \in \{a_1, a_2, \ldots, a_8\}$ and BC $R \in \{R_1, R_2, \ldots, R_{15}\}$, respectively. The $w(a)$ and $w(R)$ are obtained as the number of 1's present in the column and in the row labeled by $a$ and $R$, respectively. For ease of processing the BM, its rows are ordered by the value of the $w(R)$.

The space complexity of the BM $M_1$ is $\Omega_{EMC_1}(n) = \prod_1^{15} w(R_i) = 2.8 \times 10^9$. The PIs for the dataset are obtained by the *Algorithm_SLP_ILE* as follows:

**The 1st Iteration of the *Until* Statement:** i = 1, input is $M_1$

A1.1: Since in the BM $M_1$ (Table 7) $max\{w(a_j)\}_{j=1}^{8} = w(a_7) = 10$, $SL_1 = a_7$
A1.2: $B_0 = 00000010$

**Table 7** The BM $M_1$ representing the set of PBCs for the dataset *Diabet*

| | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | $a_7$ | $a_8$ | w(R) |
|---|---|---|---|---|---|---|---|---|---|
| $R_1$ | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 3 |
| $R_2$ | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 3 |
| $R_3$ | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 3 |
| $R_4$ | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 3 |
| $R_5$ | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 4 |
| $R_6$ | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 4 |
| $R_7$ | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 4 |
| $R_8$ | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 4 |
| $R_9$ | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 5 |
| $R_{10}$ | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 5 |
| $R_{11}$ | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 5 |
| $R_{12}$ | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 5 |
| $R_{13}$ | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 6 |
| $R_{14}$ | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 6 |
| $R_{15}$ | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 6 |
| w(a) | 8 | 8 | 9 | 8 | 6 | 9 | 10 | 8 | |

**Table 8** The SBMs $D_1$ (a) and $M_2$ (b)

| | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | $a_7$ | $a_8$ | |
|---|---|---|---|---|---|---|---|---|---|
| (a) | | | | | | | | | |
| $B_0$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| $B_1$ | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | |
| $B_2$ | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | |
| $B_3$ | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | |
| $B_4$ | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | |
| $B_5$ | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | |
| | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | $a_7$ | $a_8$ | w(R) |
| (b) | | | | | | | | | |
| $R_1$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 2 |
| $R_2$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 2 |
| $R_3$ | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 2 |
| $R_4$ | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 2 |
| $R_5$ | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 3 |
| $R_6$ | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 3 |
| $R_7$ | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 3 |
| $R_{10}$ | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 4 |
| $R_{11}$ | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 4 |
| $R_{12}$ | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 4 |
| w(a) | 4 | 4 | 5 | 4 | 3 | 5 | 0 | 4 | |

A2: $D_1 = B_0 \cup \{R_t : R_t \in M_1 \; and \; R_t(SL_1 = 0 = \{B_0, R_8, R_9, R_{13}, R_{14}, R_{15}\}$ that is re-denoted as $\{B_0, B_1, B_2, B_3, B_4, B_5\}$, respectively (Table 8a)

A3: $M_2 = M_1$ with the column $SL_1 = a_7$ cleared and the redundant rows $(R_8, R_9, R_{13}, R_{14}, R_{15})$ removed (Table 8b)

The space complexity of the SBM $D_1$ to be processed in the first iteration is $\Omega_{ECD_1}(n) = \prod_0^5 w(B_i) = 1 \times 4 \times 5 \times 6 \times 6 \times 6 = 4320 \ll \Omega_{ECD_1}(n) = \prod_0^5 w(B_t) = 1 \times 4 \times 5 \times 6 \times 6 \times 6 = 4320 \ll \sqrt{\Omega_{EMC_1}(n)} = \sqrt{2.8 \times 10^9} = 52915$. But according to (15), the *Algorithm_SLP_ILE* will process the $D_1$ with the WCSC $\Omega_{ILE}(n = 8, i = 1) \leq \binom{8-1}{(8-1)/2} = 35$.Since $\Omega_{ECD_1}(n = 8, i = 1) > \forall \Omega_{ECD_i}(n - i + 1, i > 1)$ and $\Omega_{ILE}(n = 8, i = 1) > \forall \Omega_{ILE} > (n - i, i > 1)$, we will estimate only the first iteration of the *Until* statement of the algorithm.

**Start of the Algorithm_ILE ($\mathbf{D_i}$) at i = 1**. Since the SBM $D_1$ has 5 rows, the *Algorithm_ILE* $(D_1)$ will do 5 iterations.

H1: $V_0 = \{B_0\} = \{00000010\}$

**The 1st iteration:** t = 1, inputs are $V_0$ and $B_1$

H2:$V_{11} = \{P : P \in V_0 \; and \; P\&B_1 \neq 0\} = \emptyset$; $V_{12} = V_0 - V_{11} = \{00000010\}$
H3:$E(B_1) = \{Pr_k(B_1) : b_{1k} = 1\} = \{10000000, 01000000, 00100000, 00000100\}$
H4-H7:$V_{13} = V_{12}|E(B_1) = \{10000010, 01000010, 00100010, 00000110\}$
H8:$V_1 = V_{11} \cup V_{13} = V_{13}$

**The 2nd iteration**: t = 2, inputs are $V_1$ and $B_2$

H2:$V_{21} = \{P : P \in V_1 \; and \; P\&B_2 \neq 0\} = \{01000010, 00100010, 00000110\}$; $V_{22} = V_1 - V_{21} = \{10000010\}$
H3:$E(B_2) = \{Pr_k(B_2) : b_{2k}1\} = \{01000000, 00100000, 00010000, 00000100, 00000001\}$
H4-H7:$V_{23} = V_{22}|E(B_2) = \{10010010, 10000011\}$//Here were generated and discarded 3 URPs: 11000010, 10100010 and 10000110.
H8:$V_2 = V_{21} \cup V_{23} = \{01000010, 00100010, 00000110, 10010010, 10000011\}$

**The 3rd iteration:** t = 3, inputs are $V_2$ and $B_3$

H2:$V_{31} = \{P : P \in V_2 \; and \; P\&B_3 \neq 0\} = \{01000010, 00100010, 10010010, 10000011\}$; $V_{32} = V_2 - V_{31} = \{00000110\}$
H3:$E(B_3) = \{Pr_k(B_3) : b_{3k} = 1\} = \{10000000, 01000000, 00100000, 00010000, 00001000, 00000001\}$
H4-H7:$V_{33} = V_{32}|E(B_3) = \{10000110, 00010110, 00001110, 00000111\}$ //Here were generated and discarded 2 URPs: 01000110 and 00100110
H8:$V_3 = V_{31} \cup V_{33} = \{01000010, 00100010, 10010010, 10000011, 10000110, 00010110, 00001110, 00000111\}$

**The 4th iteration**: t = 4, inputs are $V_3$ and $B_4$

H2:$V_{41} = \{P : P \in V_3 \; and \; P\&B_4 \neq 0 = \{00100010, 10010010, 10000011, 10000110, 00010110, 00001110, 00000111\}$; $V_{42} = V_3 - V_{41} = \{01000010\}$
H3:$E(B_4) = \{Pr_k(B_4) : b_{4k} = 1\} = \{10000000, 00100000, 00010000, 00001000, 00000100, 00000001\}$
H4-H7:$V_{43} = V_{42}|E(B_4) = \{11000010, 01010010, 01001010, 01000100, 01000011\}$ // Here was generated and discarded 1 URP: 01100010
H8:$V_4 = V_{41} \cup V_{43} = \{00100010, 10010010, 10000011, 10000110, 00010110, 00001110, 00000111, 11000010, 01010010, 01001010, 01000110, 01000011\}$

**The 5th iteration:** t = 5, inputs are $V_4$ and $B_5$

H2:$V_{51} = \{P : P \in V_4 \; and \; P\&B_5 \neq 0 = \{10010010, 10000011, 10000110, 00010110, 00001110, 00000111, 11000010, 01010010, 01001010, 01000110, 01000011\}$; $V_{52} = V_4 - V_{51} = \{00100010\}$
H3:$E(B_5) = \{Pr_k(B_5) : b_{5k} = 1\} = \{10000000, 01000000, 00010000, 00001000, 00000100, 00000001\}$
H4-H7:$V_{53} = V_{52}|E(B_5) = \{10000110, 01000110, 00100110, 00010110, 00001110, 00000111\}$
H8:$V_5 = V_{51} \cup V_{53} = \{10010010, 10000011, 10000110, 00010110, 00001110, 00000111, 11000010, 01010010, 01001010, 01000110, 01000011, 10100010, 01100010, 00110010, 00101010, 00100110, 00100011\}$
H9:$W_1 = V_5$; **The end of the *Algorithm_ ILE***

While in this iteration of the *"Until"* statement, conventionally $\sum_{t=0}^{4} V_t \times E(B_{t+1}) = 1 \times 4 + 4 \times 5 + 5 \times 6 + 8 \times 6 + 12 \times 6 = 174$ multiplications with the space complexity $max\{4, 20, 30, 48, 72, 17\} = 72$ had to be performed, where the number 17 is the size of the set $W_1$, the algorithm performs only $\sum_{t=1}^{5} V_{t2} \times E(B_t) = 1 \times 4 + 1 \times 5 + 1 \times 6 + 1 \times 6 + 1 \times 6 = 27$ multiplications with the space complexity $max\{4, 5, 6, 8, 12, 17\} = 17$.

**Table 9** The submatrices $D_2$ (a) and $M_3$(b)

| | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | $a_7$ | $a_8$ | |
|---|---|---|---|---|---|---|---|---|---|
| (a) | | | | | | | | | |
| $B_0$ | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | |
| $B_1$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | |
| $B_2$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $B_3$ | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $B_4$ | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | |
| $B_5$ | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | |
| | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | $a_7$ | $a_8$ | w(R) |
| (b) | | | | | | | | | |
| $R_1$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 2 |
| $R_2$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 2 |
| $R_3$ | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 2 |
| $R_4$ | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 2 |
| $R_5$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 |
| $R_6$ | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 2 |
| $R_7$ | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| $R_{11}$ | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 3 |
| $R_{12}$ | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 3 |
| w(a) | 3 | 4 | 0 | 3 | 2 | 4 | 0 | 4 | |

**The 2nd Iteration of the *Until* Statement:** i = 2, input is $M_2$

A1.1: Since, in the BM $M_2$ (Table 8b), $max\{w(a_j)\}_{j=1}^{8} = w(a_3) = w(a_6) = 5,\ SL_2 = a_3;\ A1.2 : B_0 = 00100000$
A2:$D_2 = B_0 \cup \{R_t : R_t \in M_2\ and\ R_t(SL_2) = 0\} = \{B_0,\ R_1, R_2, R_3, R_4, R_{10}\}$ that are re-denoted as $\{B_0,\ B_1, B_2, B_3, B_4, B_5\}$, respectively (Table 9a)
A3:$M_3 = M_2$ with the column $SL_2 = a_3$ cleared and the redundant (absorbed) rows removed (Table 9b).

**Start of the Algorithm_ ILE ($D_i$) at i = 2**. Since the SBM $D_2$ has 5 rows, the Algorithm_ ILE ($D_2$) will do 5 iterations.

H1:$V_0 = \{00100000\}$

**The 1st iteration**: t = 1, inputs are $V_0$ and $B_1$

H2:$V_{11} = \{P : P \in V_0\ and\ P\&B_1 \neq 0\} = \emptyset;\ V_{12} = V_0 - V_{11} = \{00100000\}$
H3:$E(B_1) = \{Pr_k(B_1) : b_{1k} = 1\} = \{00000100, 00000001\}$
H4-H7:$V_{13} = V_{12}|E(B_1) = \{00100100, 00100001\}$
H8:$V_1 = V_{11} \cup V_{13} = V_{13}$

**The 2nd iteration:** t = 2, inputs are $V_1$ and $B_2$

H2:$V_{21} = \{P : P \in V_1\ and\ P\&B_2 \neq 0\} = \{00100001\};\ V_{22} = V_1 - V_{21} = \{00100100\}$
H3:$E(B_2) = \{Pr_k(B_2) : b_{2k} = 1\} = \{01000000, 00000001\}$

H4-H7:$V_{13} = V_{22}|E(B_2) = \{01100100\}$//Here was generated and discarded 1 URP: 00100101
H8:$V_2 = V_{21} \cup V_{23} = \{00100001, 01100100\}$

**The 3rd iteration:** t = 3, inputs are $V_2$ and $B_3$

H2:$V_{31} = \{P : P \in V_2 \text{ and } P\&B_3 \neq 0\} = \{01100100\}$; $V_{32} = V_2 - V_{31} = \{00100100\}$
H3:$E(B_3) = \{Pr_k(B_3) : b_{3k} = 1\} = \{01000000, 00000100\}$
H4-H7:$V_{33} = V_{32}|E(B_3) == \{01100001, 00100101\}$
H8:$V_3 = V_{31} \cup V_{33} = \{01100100, 01100001, 00100101\}$

**The 4th iteration:** t = 4, inputs are $V_3$and $B_4$

H2:$V_{41} = \{P : P \in V_3 \text{ and } P\&B_4 \neq 0\} = \{01100100, 01100001\}$; $V_{42} = V_3 - V_{41} = \{00100101\}$
H3:$E(B_4) = \{Pr_k(B_4) : b_{4k} = 1\} = \{01000000, 00010000\}$
H4-H7:$V_{43} = V_{42}|E(B_4) = \{00110101\}$//Here was generated and discarded 1 URP 01100101;
H8:$V_4 = V_{41} \cup V_{43} = \{01100100, 01100001, 00110101\}$

**The 5th iteration:** t = 5, inputs are $V_4$ and $B_5$

H2:$V_{51} = \{P : P \in V_4 \text{ and } P\&B_4 \neq 0\} = \{01100100, 00110101\}$; $V_{52} = V_3 - V_{41} = \{01100001\}$
H4:$E(B_5) = \{Pr_k(B_5) : b_{5k} = 1\} = \{10000000, 00010000, 00001000, 00000100\}$
H5-H8:$V_{53} = V_{52}|E(B_5) = \{11100001, 01110001, 01101001\}$ // Here was generated and discarded 1 URP: 01100101
H9:$V_5 = V_{51} \cup V_{23} = \{01100100, 00110101, 11100001, 01110001, 01101001\}$
H10:$W_2 = V_5$; **The end of the *Algorithm_ ILE***

To complete this example, two more iterations of the *Until* statement with $i = 3$, and $i = 4$ are to be performed. At $i = 3$, first the SBM $M_3$ (Table 9b) is partitioned by $SL_3 = a_2$ (instead of $a_2$ could be used either $a_6$ or $a_8$) into the SBMs $D_3$ and $M_4$ (Table 10), and then $D_3$ is converted into the set of PBPs $\boldsymbol{W}_3 = \{01000101, 11010100, 11001100, 11010001, 11001001\}$.

At $i = 4$, first the SBM $M_4$ (Table 10b) is partitioned by $SL_4 = a_1$ (instead of $a_1$ could be used either $a_4$ or $a_6$ or $a_8$) into the SBMs $D_4$ and $M_5$ (Table 11), and then $D_4$ is converted into the set of PBPs $W_4 = \{10010101\}$.

Since the SBM $M_5$ consist of a single zero row, the *Algorithm_SLP_ILE* is terminated with:

**A8**. $W = \{W_j\}_{j=1}^{4} = \{10010010, 10000011, 10000110, 00010110, 00001110, 00000111, 11000010, 01010010, 01001010, 01000110, 01000011, 10100010, 01100010, 00110010, 00101010, 00100110, 00100011, 01100100, 00110101, 11100001, 01110001, 01101001, 01000101, 11010100, 11001100, 11010001, 11001001, 10010101\}$;

**A9.** *Consider each element of W as a PI of the DF:* $W_{PI} = \{a_1a_4a_7, a_1a_7a_8, a_1a_6a_7, a_4a_6a_7, a_5a_6a_7, a_6a_7a_8, a_1a_2a_7, a_2a_4a_7, a_2a_5a_7, a_2a_6a_7, a_2a_7a_8, a_1a_3a_7, a_2a_3a_7, a_3a_4a_7, a_3a_5a_7, a_3a_6a_7, a_3a_7a_8, a_2a_3a_6, a_3a_4a_6a_8, a_1a_2a_3a_8, a_2a_3a_4a_8, a_2a_3a_5a_8, a_2a_6a_8, a_1a_2a_4a_6, a_1a_2a_5a_6, a_1a_2a_4a_8, a_1a_2a_5a_8, a_1a_4a_6a_8\}$.

## 6 Experimental results

To estimate the performance of the proposed method, we compare the results generated by it with the results generated by RSES (Rough Set Exploration System), which is a

**Table 10** The submatrices $D_3$ (a) and $M_4$ (b)

| | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | $a_7$ | $a_8$ | |
|---|---|---|---|---|---|---|---|---|---|
| (a) | | | | | | | | | |
| $B_0$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| $B_1$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | |
| $B_2$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $B_3$ | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $B_4$ | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | |
| $B_5$ | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | |
| | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | $a_7$ | $a_8$ | w(R) |
| (b) | | | | | | | | | |
| $R_2$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| $R_3$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| $R_4$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| $R_7$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| w(a) | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | |

**Table 11** The submatrices $D_4$ (a) and $M_5$ (b)

| | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | $a_7$ | $a_8$ | |
|---|---|---|---|---|---|---|---|---|---|
| (a) | | | | | | | | | |
| $B_0$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| $B_1$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $B_2$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $B_3$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | |
| | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ | $a_7$ | $a_8$ | w(R) |
| (b) | | | | | | | | | |
| $R_7$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| w(a) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

unique exhaustive search-based AR program generating all reducts. In the experiments, we used a target machine with an Intel Core I7 3840QM@2.80 GHz processor and 16 GB memory, running on Microsoft Windows 8 Ultimate Edition OS. For experiments, we chose datasets with different characteristics such as: the number of attributes, the number of classes, the number of distinct values of the attributes, the types of attribute values (symbolic, numeric and mixed) and the number of objects (instances). We loaded these datasets into the memory without any preprocessing. The results of experiments measured by *Microsoft's process explorer utility* show that our algorithm has important advantages over RSES in terms of both the memory used and time elapsed. As the examples in Table 12, the results of processing of 23 datasets from the UCI repository [40] are given. Processing some of these datasets by RSES was failed due to overflow of memory. The

**Table 12** The amounts of the memory used, the CPU-times elapsed, and the number of reducts generated by SLP-ILE and RSES approaches

| Data set | Number of attributes/objects/classes | Amount of the memory (in MBs) used by RSES ($M_1$) / SLP-ILE ($M_2$) | | | CPU time (in seconds) elapsed by RSES ($T_1$) / SLA_ILA ($T_2$) | | | Number of reducts generated | |
|---|---|---|---|---|---|---|---|---|---|
| | | $M_1$ | $M_2$ | $M_1/M_2$ | $T_1$ | $T_2$ | $T_1/T_2$ | RSES | SLP-ILE |
| Diabet | 8/768/2 | 0.83 | **0.27** | 3.07 | 0.67 | **0.47** | 1.43 | 28 | 28 |
| Whitewinequality | 11/4898/7 | 1.62 | **0.70** | 2.32 | 12.36 | **2.80** | 4.42 | 127 | 127 |
| Redwinequality | 11/1599/6 | 0.99 | **0.33** | 3.01 | 1.33 | **0.31** | 4.24 | 227 | 227 |
| Pokerhand | 11/1025010/10 | 278.12 | **29.35** | 9.47 | 202.68 | **80.96** | 2.50 | 8 | 8 |
| Lymn | 18/148/4 | 0.66 | **0.32** | 2.04 | 3.75 | **0.03** | 120.97 | 424 | 424 |
| Vehicle | 18/846/4 | 1.49 | **0.32** | 4.60 | 40.19 | **0.22** | 183.50 | 1413 | 1413 |
| Hepatitis | 19/155/2 | 1.23 | **0.39** | 3.18 | 59.27 | **0.03** | 1911.77 | 694 | 694 |
| Twonorm | 20/7400/2 | 2.75 | **1.81** | 1.52 | 33.73 | **13.88** | 2.43 | 189 | 189 |
| Thyroid | 21/7200/3 | 2.34 | **0.26** | 8.89 | 15.80 | **1.78** | 8.87 | 24 | 24 |
| Mushroom | 22/8124/2 | 2.72 | **1.01** | 2.69 | 48.11 | **9.23** | 5.21 | 292 | 292 |
| Statlog(Gcd) | 24/1000/2 | **2.40** | 2.95 | 0.81 | 115.36 | **0.28** | 410.53 | 2424 | 2424 |
| Wall-Following Robot Navigation | 24/5456/4 | 5.02 | **1.57** | 3.19 | 2024.58 | **9.70** | 208.65 | 5175 | 5175 |
| Fars | 29/100968/8 | 27.72 | **22.21** | 1.24 | 11247.27 | **3177.8** | 3.54 | 8 | 8 |
| Chess | 36/3196/2 | 680.42 | **0.82** | 825.75 | 23.58 | **2.24** | 10.55 | 4 | 4 |
| Soybean | 35/683/19 | 1001.08 | **0.39** | 2593.47 | 453.13 | **0.63** | 725.00 | 2370 | 2370 |
| Spambase | 57/4601/2 | 1764.13 | **3.56** | 495.26 | 78.31 | **7.23** | 10.83 | 120 | 120 |
| Ionosphere | 34/351/2 | >1857.13 | **0.39** | >4811.22 | >7142.73 | **0.41** | 17592.94 | No Result (Due to Failure) | 417 |
| Dermatology | 34/366/6 | >1895.32 | **10.50** | >180.51 | >1229.75 | **231.83** | >5.30 | | 41922 |

**Table 12** continued

| Data set | Number of attributes/objects/classes | Amount of the memory (in MBs) used by RSES ($M_1$) / SLP-ILE ($M_2$) | | | CPU time (in seconds) elapsed by RSES ($T_1$) / SLA_ILA ($T_2$) | | | Number of reducts generated | |
|---|---|---|---|---|---|---|---|---|---|
| | | $M_1$ | $M_2$ | $M_1/M_2$ | $T_1$ | $T_2$ | $T_1/T_2$ | RSES | SLP-ILE |
| Anneal | 38/798/6 | >1903.00 | **0.45** | >4238.31 | >394.78 | **0.20** | >1944.73 | | 5759 |
| Cylinderbands | 40/540/2 | >1857.78 | **8.48** | >218.97 | >21544.17 | **34.99** | >615.81 | | 43907 |
| Kddcup | 41/494020/23 | >1800.00 | **150.3** | >11.97 | >409875.14 | **83258** | >4.88 | | 607 |
| Spectf Heart | 44/267/59 | >1800.00 | **6.73** | >267.46 | >149582.81 | **48.81** | >3064.47 | | 11556 |
| Sonar | 60/208/2 | >1800.00 | **7.48** | >240.64 | >404788.14 | **21.03** | >19247.21 | | 86872 |

**Table 13** The size and the attributes of reducts generated by AR approaches

| Data set | Size/attributes of the reducts generated by | | | |
|---|---|---|---|---|
| | SLP-ILE | CBA | CA | WA |
| Diabet | **4/a**$_1$, **a**$_2$, **a**$_3$, **a**$_8$ | **4/a**$_2$, **a**$_6$, **a**$_7$, **a**$_8$ | 8/ a$_1$, a$_2$, a$_3$, a$_4$, a$_5$, a$_6$, a$_7$, a$_8$ | 5/ a$_1$, a$_2$, a$_3$, a$_6$, a$_7$ |
| Fars | 23/a$_1$, a$_2$, a$_3$, a$_4$, a$_5$, a$_6$, a$_7$, a$_8$, a$_9$, a$_{10}$, a$_{11}$, a$_{12}$, a$_{13}$, a$_{14}$, a$_{15}$, a$_{16}$, a$_{17}$, a$_{18}$, a$_{19}$, a$_{20}$, a$_{24}$, a$_{25}$, a$_{26}$ | 7/a$_3$, a$_6$, a$_{10}$, a$_{12}$, a$_{24}$, a$_{25}$, a$_{29}$ | 23/a$_1$, a$_2$, a$_3$, a$_4$, a$_5$, a$_6$, a$_7$, a$_8$, a$_9$, a$_{10}$, a$_{11}$, a$_{12}$, a$_{13}$, a$_{14}$, a$_{15}$, a$_{16}$, a$_{17}$, a$_{18}$, a$_{19}$, a$_{20}$, a$_{24}$, a$_{25}$, a$_{26}$ | **1/a**$_4$ |
| Ionesphere | **4/a**$_1$, **a**$_4$, **a**$_5$, **a**$_6$ | 14/ a$_1$, a$_3$, a$_4$, a$_5$, a$_6$, a$_7$, a$_8$, a$_{14}$, a$_{18}$, a$_{21}$, a$_{27}$, a$_{28}$, a$_{29}$, a$_{34}$ | 7/ a$_5$, a$_6$, a$_8$, a$_{13}$, a$_{22}$, a$_{27}$, a$_{34}$ | 5/ a$_4$, a$_5$, a$_{12}$, a$_{14}$, a$_{32}$ |
| Kddcup | 10/ a$_3$, a$_5$, a$_6$, a$_{24}$, a$_{26}$, a$_{33}$, a$_{34}$, a$_{36}$, a$_{37}$, a$_{40}$ | 10/ a$_3$, a$_5$, a$_6$, a$_{24}$, a$_{26}$, a$_{33}$, a$_{34}$, a$_{36}$, a$_{37}$, a$_{40}$ | 10/ a$_1$, a$_3$, a$_5$, a$_6$, a$_{12}$, a$_{33}$, a$_{35}$, a$_{36}$, a$_{37}$, a$_{38}$ | **1/a**$_2$ |
| Lymn | 9/ a$_2$, a$_3$, a$_6$, a$_7$, a$_9$, a$_{11}$, a$_{14}$, a$_{16}$, a$_{17}$ | 11/ a$_1$, a$_2$, a$_3$, a$_8$, a$_9$, a$_{10}$ a$_{11}$, a$_{12}$, a$_{14}$, a$_{15}$, a$_{16}$ | 9/ a$_1$, a$_2$, a$_3$, a$_6$, a$_9$, a$_{11}$, a$_{14}$, a$_{15}$, a$_{16}$ | **7/a**$_3$, **a**$_8$, **a**$_9$, **a**$_{10}$, **a**$_{13}$, **a**$_{14}$, **a**$_{16}$ |
| Pokerhand | 7/ a$_1$, a$_2$, a$_4$, a$_5$, a$_6$, a$_8$, a$_{10}$ | **1/a**$_1$ | 4/ a$_2$, a$_3$, a$_4$, a$_6$ | 4/ a$_2$, a$_3$, a$_4$, a$_6$ |
| Redwinequality | **4/a**$_2$, **a**$_5$, **a**$_{10}$, **a**$_{11}$ | **4/a**$_2$, **a**$_7$, **a**$_{10}$, **a**$_{11}$ | 8/ a$_1$, a$_2$, a$_3$, a$_5$, a$_7$, a$_8$, a$_{10}$, a$_{11}$ | **4/a**$_2$, **a**$_7$, **a**$_{10}$, **a**$_{11}$ |
| Sonar | 3/ a$_{11}$, a$_{36}$, a$_{45}$ | 19/ a$_2$, a$_4$, a$_5$, a$_9$, a$_{10}$, a$_{11}$, a$_{12}$, a$_{13}$, a$_{21}$, a$_{28}$, a$_{36}$, a$_{44}$, a$_{45}$, a$_{47}$, a$_{48}$, a$_{49}$, a$_{51}$, a$_{52}$, a$_{58}$ | 15/ a$_1$, a$_4$, a$_5$, a$_9$, a$_{11}$, a$_{12}$, a$_{20}$, a$_{35}$, a$_{37}$, a$_{45}$, a$_{46}$, a$_{47}$, a$_{51}$, a$_{52}$, a$_{54}$ | **2/a**$_{12}$, **a**$_{19}$ |
| Spectf | 5/ a$_1$, a$_{15}$, a$_{23}$, a$_{37}$, a$_{43}$ | **2/a**$_{27}$, **a**$_{43}$ | **2/a**$_{27}$, **a**$_{43}$ | **2/a**$_{37}$, **a**$_{43}$ |
| StatlogGCD | 13/ a$_1$, a$_3$, a$_5$, a$_6$, a$_7$, a$_8$, a$_9$, a$_{11}$, a$_{14}$, a$_{16}$, a$_{17}$, a$_{20}$, a$_{24}$ | **5/a**$_1$, **a**$_2$, **a**$_3$, **a**$_7$, **a**$_{21}$ | 14/ a$_2$, a$_3$, a$_4$, a$_5$, a$_6$, a$_7$, a$_9$, a$_{10}$, a$_{11}$, a$_{15}$, a$_{16}$, a$_{17}$, a$_{20}$, a$_{21}$ | 8/ a$_1$, a$_3$, a$_5$, a$_9$, a$_{15}$, a$_{16}$, a$_{17}$, a$_{23}$ |
| Thyroid | 6/ a$_3$, a$_8$, a$_{10}$, a$_{11}$, a$_{17}$, a$_{21}$ | **5/a**$_8$, **a**$_{13}$, **a**$_{17}$, **a**$_{18}$, **a**$_{21}$ | 9/ a$_1$, a$_2$, a$_3$, a$_8$, a$_{17}$, a$_{18}$, a$_{19}$, a$_{20}$, a$_{21}$ | 10/ a$_1$, a$_2$, a$_3$, a$_9$, a$_{10}$, a$_{17}$, a$_{18}$, a$_{19}$, a$_{20}$, a$_{21}$ |

**Table 13** continued

| Data set | Size/attributes of the reducts generated by | | | |
|---|---|---|---|---|
| | SLP-ILE | CBA | CA | WA |
| Vehicle | 5/ $a_1$, $a_6$, $a_8$, $a_{10}$, $a_{14}$ | 11/ $a_4$, $a_5$, $a_6$, $a_7$, $a_8$, $a_9$, $a_{11}$, $a_{12}$, $a_{14}$, $a_{15}$, $a_{16}$ | 18/ $a_1$, $a_2$, $a_3$, $a_4$, $a_5$, $a_6$, $a_7$, $a_8$, $a_9$, $a_{10}$, $a_{11}$, $a_{12}$, $a_{13}$, $a_{14}$, $a_{15}$, $a_{16}$, $a_{17}$, $a_{18}$ | **2**/$\mathbf{a}_{11}$, $\mathbf{a}_{18}$ |
| Whitewinequality | 5/ $a_2$, $a_3$, $a_5$, $a_8$, $a_{11}$ | 6/ $a_2$, $a_3$, $a_5$, $a_6$, $a_8$, $a_{11}$ | 11/ $a_1$, $a_2$, $a_3$, $a_4$, $a_5$, $a_6$, $a_7$, $a_8$, $a_9$, $a_{10}$, $a_{11}$ | **2**/$\mathbf{a}_2$, $\mathbf{a}_{11}$ |
| Wall-Following Robot Navigation | **3**/$\mathbf{a}_2$, $\mathbf{a}_{15}$, $\mathbf{a}_{20}$ | 6/ $a_{13}$, $a_{14}$, $a_{15}$, $a_{18}$, $a_{19}$, $a_{20}$ | 7/ $a_2$, $a_{12}$, $a_{14}$, $a_{15}$, $a_{18}$, $a_{19}$, $a_{20}$ | 4/ $a_5$, $a_{10}$, $a_{15}$, $a_{24}$ |

**Table 14** The classification accuracies and the elapsed CPU-Times for classification of reducts generated by AR approaches

| Dataset name | Classification accuracies of reducts generated by | | | | CPU time elapsed for classification with the reduct generated by | | | |
|---|---|---|---|---|---|---|---|---|
| | SLP-ILE | CBA | CA | WA | SLP-ILE | CBA | CA | WA |
| Diabet | **0.751** | 0.741 | 0.688 | 0.742 | **0.19** | **0.19** | 0.34 | 0.22 |
| Fars | **0.767** | **0.767** | **0.767** | 0.415 | 3317.75 | 2294.15 | 3551.10 | **1186.41** |
| Ionesphere | **0.912** | 0.840 | 0.863 | 0.860 | **0.04** | 0.11 | 0.06 | 0.05 |
| Kddcup | 0.992 | 0.990 | **0.995** | 0.568 | 57662.43 | 57812.45 | 57598.45 | **2667.25** |
| Lymn | 0.818 | 0.723 | 0.730 | **0.824** | 0.02 | 0.02 | 0.02 | **0.01** |
| Redwinequality | **0.579** | 0.538 | 0.548 | 0.538 | **0.80** | **0.80** | 1.42 | **0.80** |
| Pokerhand | **0.732** | 0.507 | 0.604 | 0.604 | 97648.95 | **33204.76** | 75698.83 | 75582.21 |
| Sonar | **0.731** | 0.582 | 0.591 | 0.558 | 0.02 | 0.06 | 0.04 | **0.01** |
| Spectf | **0.236** | 0.154 | 0.154 | 0.109 | 0.04 | **0.02** | **0.02** | **0.02** |
| StatlogGCD | **0.739** | 0.708 | 0.690 | 0.732 | 0.71 | **0.34** | 0.77 | 0.46 |
| Thyroid | **0.979** | 0.970 | 0.943 | 0.941 | 17.92 | **16.69** | 28.19 | 30.25 |
| Vehicle | **0.735** | 0.621 | 0.680 | 0.615 | 0.28 | 0.53 | 0.83 | **0.16** |
| Whitewinequality | **0.514** | 0.492 | 0.463 | 0.508 | 9.03 | 10.39 | 17.91 | **4.69** |
| Wall-following robot navigation | **0.905** | 0.892 | 0.862 | 0.855 | **9.14** | 12.75 | 14.43 | **9.14** |

results of experiments are given in Table 12, where "> X" denotes the expression "*greater than X*".

As seen from Table 12, our method requires much less amount of memory and much less processing time than RSES, especially, for medium- and large-sized datasets. For example, for the dataset *Sonar* with 60 attributes, 208 objects and 2 classes, while RSES failed after 404788.14 seconds with no result due to memory overflow at about 1800MB of the intermediate results, our program used only 7.48 MB of the memory and generated all of 86872 reducts in only 21.03 seconds.

For evaluating the comparative performance of the reducts generated by the SLP-ILE approach, we also generated reducts for the 11 datasets given in Table 12 by the widely used AR approaches such as *Correlation-based* approach (CBA), *Consistency* approach (CA) and *Wrapper* approach (WA) [41]. The sizes and the attributes of the reducts generated by the mentioned AR approaches are given in Table 13.

Note that the shortest reduct is not always the reduct with the best classification accuracy [10,42]. Therefore, in Table 13, column of SLP-ILE are not the sizes of the shortest reduct but the sizes of the reduct with the best classification accuracies.

For evaluating the classification accuracies of the reducts generated by the SLP-ILE, CBA, CA, and WA, we used the Euclidian distance-based K-nearest neighborhood (K-nn) classification algorithm with k = 9. The classification results of the datasets given in Table 13 with 10-fold cross-validation are given in Table 14.

As it is seen from Table 14, the reducts generated by SLP-ILE for 12 of 14 datasets provide significant higher classification accuracies than those generated by other AR approaches. Moreover, the CPU time elapsed by SLP_ILE is usually lower than other AR approaches.

## 7 Conclusions

At present, the generation of the complete set of reducts for a dataset can be done only by the exhaustive search program RSES that, unfortunately, is very time-consuming and suffers from frequent memory overflows even for the datasets of medium dimensionality. The proposed algorithm is based on a sequential partitioning of the BM of a DF into the SBMs and preventing the generation of redundant products during the transformation of a SBM into PIs (reducts) of the DF. Although this approach reduces the WCSC of the DF-based AR in a large scale, it still remains theoretically exponential in $n$. This is, mainly, due to that the mentioned WCSC is evaluated for the DFs containing all $\binom{n}{n/2}$ combinations of $n$ attributes with uniformly distributed frequencies. But since the space complexity of the proposed algorithm is rapidly decreased with increasing the imbalance in weights of columns in the BM, it may be quite useful for problems of tens and even of hundreds of attributes with seriously imbalanced distributions of frequencies in DFs. This is confirmed by the analysis of results of processing numerous datasets borrowed from practice and from the UCI repository. The proposed approach also allows us to process the original BM in parallel by first partitioning it into the SBMs $D_1$, $D_2$, …, $D_{k\leq n-1}$, and then separately processing them and uniting the separate obtained results. In this respect, if some SBM is found to be so large that cannot be processed by the computer used, it may be processed as a pseudo-original BM with uniting the result with that of the rest of processing of the original BM. It should be noted that, the proposed approach reduces not only the WCSC of DF to DNF conversion in a very large scale, but it also reduces the WCTC of this conversion from square of WCSC to somewhat near to it. Moreover, when it is to find a single reduct, it may be found simply by considering every $D_i$ as $M_{i+1}$ and taking each SL as a component of the reduct. We believe that the proposed approach could be an efficient solution for the *unweighted set cover problem* used for solving wide range of discrete optimization problems in knowledge and data engineering.

## References

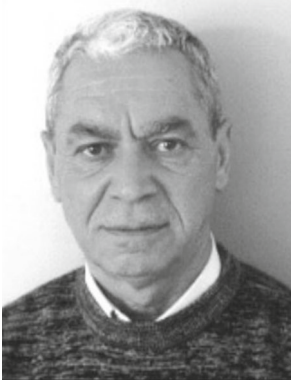

1. Komorowski J, Pawlak Z, Polkowski L, Skowron A (1999) Rough-fuzzy hybridization: rough sets: a tutorial. A new trend in decision making. Springer, Berlin
2. Wang XY, Yang J, Teng XL, Xia WJ, Jensen R (2007) Feature selection based on rough sets and particle swarm optimization. Pattern Recogn Lett 28(4):459–471
3. Shang WQ, Huang HK, Zhu HB, Lin YM, Qu YL, Wang ZH (2007) A novel feature selection algorithm for text categorization. Expert Syst Appl 33(1):1–5
4. Matsumoto Y, Watada J (2009) Knowledge acquisition from time series data through rough sets analysis. Int J Innov Comput Inf Control 5(12B):4885–4897
5. Javed K, Babri HA, Saeed M (2012) Feature selection based on class-dependent densities for high-dimensional binary data. IEEE Trans Knowl Data Eng 24(3):465–477
6. Yang SH, Hu BG (2012) Discriminative feature selection by nonparametric Bayes error minimization. IEEE Trans Knowl Data Eng 24(8):1422–1434
7. Zhao Z, Wang L, Liu H, Ye JP (2013) On similarity preserving feature selection. IEEE Trans Knowl Data Eng 25(3):619–632
8. Skowron A, Rauszer C (1992) The discernibility matrices and functions in information systems. ICS PAS Report 1/91, Technical University of Warsaw, pp 1–44
9. Maji P, Pal SK (2010) Feature selection using f-information measures in fuzzy approximation spaces. IEEE Trans Knowl Data Eng 22(6):854–867
10. Jensen R, Shen Q (2004) Semantics-preserving dimensionality reduction: rough and fuzzy-rough-based approaches. IEEE Trans Knowl Data Eng 16(12):1457–1471
11. Thangavel K, Pethalakshmi A (2009) Dimensionality reduction based on rough set theory: a review. Appl Soft Comput 9(1):1–12

12. Gao BJ, Ester M, Xiong H, Cai JY, Schulte O (2013) The minimum consistent subset cover problem: a minimization view of data mining. IEEE Trans Knowl Data Eng 25(3):690–703
13. Hall MA, Holmes G (2003) Benchmarking attribute selection techniques for discrete class data mining. IEEE Trans Knowl Data Eng 15(6):1437–1447
14. Liu H, Yu L (2005) Toward integrating feature selection algorithms for classification and clustering. IEEE Trans Knowl Data Eng 17(4):491–502
15. Qu GZ, Hariri S, Yousif M (2005) A new dependency and correlation analysis for features. IEEE Trans Knowl Data Eng 17(9):1199–1207
16. Chen WC, Tseng SS, Hong TP (2008) An efficient bit-based feature selection method. Expert Syst Appl 34(4):2858–2869
17. Wang F, Liang JY, Dang CY (2013) Attribute reduction for dynamic data sets. Appl Soft Comput 13(1):676–689
18. Yan J, Zhang BY, Liu N, Yan SC, Cheng QS, Fan WG, Yang Q, Xi WS, Chen Z (2006) Effective and efficient dimensionality reduction for large-scale and streaming data preprocessing. IEEE Trans Knowl Data Eng 18(3):320–333
19. Skowron A (1990) The rough sets theory and evidence theory. Fundam Inf 13:245–262
20. Øhrn A, Komorowski J, Skowron A, Synak P (1998) The design and implementation of a knowledge discovery toolkit based on rough sets: the ROSETTA system. In: Polkowski L, Skowron A (eds) Rough sets in knowledge discovery. Physica Verlag, Heidelberg, pp 376–399
21. Jensen R, Shen Q (2007) Rough set based attribute selection: a review. http://cadair.aber.ac.uk/dspace/bitstream/handle/2160/490/JensenShen.pdf?sequence=3. Accessed 12 Dec 2013
22. Wang J, Wang J (2001) Reduction algorithms based on discernibility matrix: the ordered attributes method. J Comput Sci Technol 16(6):489–504
23. Mafarja M, Abdullah S (2013) Record-to-record travel algorithm for attribute reduction in rough set theory. J Theor Appl Inf Technol 49(2):507–513
24. Kahramanli S, Hacibeyoglu M, Arslan A (2011) Attribute reduction by partitioning the minimized discernibility function. Int J Innov Comput Inf Control 7(5A):2167–2186
25. Procaccia AD, Rosenschein JS (2006) Exact VC-dimension of Monotone formulas. Neural Inf Process Lett Rev 10(7):165–168
26. Hacibeyoglu M, Basciftci F, Kahramanli S (2011) A logic method for efficient reduction of the space complexity of the attribute reduction problem. Turk J Electr Eng Comput Sci 19(4):643–656
27. Kahramanli S, Hacibeyoglu M, Arslan A (2011) A Boolean function approach to feature selection in consistent decision information systems. Expert Syst Appl 38(7):8229–8239
28. Nelson JR (1955) Simplest normal truth functions. J Symb Log 20(2):105–108
29. Malik AA, Brayton RK, Newton AR, Sangiovannivincentelli A (1991) Reduced offsets for minimization of binary-valued functions. IEEE Trans Comput Aided Des Integr Circuits Syst 10(4):413–426
30. Miltersen PB, Radhakrishnan J, Wegener I (2005) On converting CNF to DNF. Theor Comput Sci 347 (1–2):325–335
31. Vorwerk K, Paulley GN (2002) On implicate discovery and query optimization. In: Proceedings of international database engineering and applications symposium, 2002
32. Slagle JR, Chang CL, Lee RCT (1970) New algorithm for generating prime implicants. IEEE Trans Comput C–19(3):304–310
33. Thelen B (1981) Investigations of algorithms for computer-aided logic design of digital circuits. University of Karlsruhe, Karlsruhe
34. Bieganowski J, Karatkevich A (2005) Heuristics for Thelen's prime implicant method. Scheda Inf 14: 125–135
35. Karatkevich A, Bieganowski J (2004) Detection of deadlocks and traps in petri nets by means of Thelen's prime implicant method. Int J Appl Math Comput Sci 14(1):113–121
36. Socher R (1991) Optimizing the clausal normal form transformation. J Autom Reason 7:325–336
37. Shiny AK, Pujari AK (1998) Computation of prime implicants using matrix and paths. J Logic Comput 8(2):135–145
38. Lee TT, Lo TY, Wang JF (2006) An information-lossless decomposition theory of relational information systems. IEEE Trans Inf Theory 52(5):1890–1903
39. Sasao T, Butler JT (2001) Worst and best irredundant sum-of-products expressions. IEEE Trans Comput 50(9):935–948
40. Machine Learning Repository. http://archive.ics.uci.edu/ml. Accessed 13 Dec 2013
41. Selvakuberan K, Indradevi M, Rajaman R (2008) Combined feature selection and classification—a novel approach for the categorization of web pages. J Inf Comput Sci 3(2):83–89
42. Hacibeyoglu M, Arslan A, Kahramanli S (2013) A hybrid method for fast finding the reduct with the best classification accuracy. Adv Electr Comput Eng 13(4):57–64

**Mehmet Hacibeyoglu** is currently Assistant Professor at the Computer Engineering Department of the Konya Necmettin Erbakan University (Konya, Turkey). He received the B.Sc., M.Sc., and Ph.D. Degrees in Computer Engineering from the Selcuk University (Konya, Turkey) in 2003, 2006, and 2012, respectively. His research interests are system administration in Unix and Linux, machine learning, feature selection, logic circuits, meta heuristic algorithms to the nesting problems, information security, and data mining.

**Mohammad Shukri Salman** received the B.Sc., M.Sc., and Ph.D. Degrees in Electrical and Electronics Engineering from the Eastern Mediterranean University (EMU), in 2006, 2007, and 2011, respectively. From 2006 to 2010, he was a teaching assistant of Electrical and Electronics Engineering department at EMU. In 2010, he joined the Department of Electrical and Electronic Engineering at European University of Lefke (EUL) as a senior lecturer for the Department. Since 2011, he worked as an Assist. Prof. Dr. in the Department of Electrical and Electronics Engineering, Mevlana (Rumi) University, Turkey. He served as a general chair, program chair, and a TPC member for many international conferences. He has supervised 6 M.Sc. Theses and currently supervising 4 Ph.D. Theses. His research interests include signal processing, adaptive filters, image processing, and communications systems.

**Murat Selek** received the M.Sc. Degree in electronics engineering from the Selcuk University, Turkey, in 1997 and the Ph.D. degree from the Selcuk University, Turkey, in 2007. He is head of the department of electronics and automation, Higher School of Vocational and Technical Sciences, Selcuk University. His research interests are in the fields of image processing, infrared thermography, autofocusing, remote control and telecommunications.



**Sirzat Kahramanli** was born in Fizuli, Azerbaijan, in 1944. He received his Ph.D. degree in computer science from Azerbaijan Technical University, Baku in 1983. From 1968 to 1993, he worked in the department of Computer Systems at the same university. From 1993 to 2011, he was with the department of Computer Engineering at Selcuk University, Konya, Turkey. From 2011 to 2014, he served as a Professor with the department of Computer Engineering at Mevlana (Rumi) University, Konya, Turkey. Since 2014, he worked as a Professor with the Department of Computer Education and Instructional Technologies Teaching at Mevlana (Rumi) University, Konya, Turkey. His research interests include switching theory and data processing.