Breadth search strategies for finding minimal reducts: towards hardware implementation

Mateusz Choromański¹ · Tomasz Grześ¹ · Piotr Hońko¹

Received: 27 December 2018 / Accepted: 5 March 2020 / Published online: 19 March 2020
© The Author(s) 2020

Abstract

Attribute reduction, being a complex problem in data mining, has attracted many researchers. The importance of this issue rises due to ever-growing data to be mined. Together with data growth, a need for speeding up computations increases. The contribution of this paper is twofold: (1) investigation of breadth search strategies for finding minimal reducts in order to emerge the most promising method for processing large data sets; (2) development and implementation of the first hardware approach to finding minimal reducts in order to speed up time-consuming computations. Experimental research showed that for software implementation blind breadth search strategy is in general faster than frequency-based breadth search strategy not only in finding all minimal reducts but also in finding one of them. An inverse situation was observed for hardware implementation. In the future work, the implemented tool is to be used as a fundamental module in a system to be built for processing large data sets.

Keywords Attribute reduction · Minimal reducts · Breadth search strategy · Hardware implementation · Field programmable gate arrays

1 Introduction

Feature selection, especially attribute reduction [26], seems to be a more essential data preprocessing task than ever before. In the Big Data era, reducing a data set, even by one feature/attribute, may significantly influence the data size or/and time needed for processing it. In spite of the fact that reducing as many attributes as possible of a large data set may be expensive itself, benefits that come from further processing a much smaller data set can be incomparably greater.

Attribute reduction has extensively been investigated from a theoretical and practical viewpoint (e.g. [2, 4, 11, 19, 20, 30, 33, 37, 38, 40, 56]) in rough set theory

[26, 28] that is viewed as a mathematical tool to deal with inconsistent data. In spite of the existence of a rich literature, attribute reduction is still a hot topic; a new area of its application and new methods of its improvement are constantly discovered (e.g. [3, 5, 14, 15, 22, 25, 39]).

Due to the scope of this paper, the below subsections provide an overview of software and hardware rough set-based approaches for finding reducts.

1.1 Software approaches for finding reducts

A variety of algorithms have been developed in rough set theory for finding reducts of an information system or decision table. They can be categorized into the following general groups: discernibility matrix-based approach [4, 11, 19, 30, 44, 53, 55], positive region-based approach [7, 9, 13, 26, 29, 50, 52], and information entropy-based approach [10, 23, 24, 31, 47, 48].

Algorithms of the first group use an auxiliary structure, called a discernibility matrix, which is constructed based on the data table. Each cell of the matrix shows for a given pair of objects the attributes for which the objects are

✉ Piotr Hońko
p.honko@pb.edu.pl
Mateusz Choromański
m.choromanski@pb.edu.pl
Tomasz Grześ
t.grzes@pb.edu.pl

¹ Faculty of Computer Science, Białystok University of Technology, Wiejska 45A, 15-351 Białystok, Poland

different. Any minimal subset of attributes that includes at least one attribute of each cell is a reduct.

The second group of algorithms operates on the positive region of a decision table. The region includes objects that are consistent for a given subset of attributes. A minimal attribute subset that preserves the consistency is a reduct.

The last group of algorithms evaluates a subset of attributes using an information entropy measure. Information entropy shows the discernibility power of a given attribute subset. That with the highest quality under a given measure is chosen as a reduct.

An important branch of attribute reduction is finding minimal reducts. Finding such a reduct enables to construct the smallest representation of data (in terms of the number of attributes) that preserves the original level of objects discernibility. However, finding minimal reduct is a more complex task than finding any reduct, namely it is proved to be an NP-hard problem [30].

One of general approaches is to limit the problem of finding all reducts to those whose cardinality is smallest. The solution is rather simple since it can rely on filtering all reducts or on controlling the process of generation of reducts so that at least some reducts to be prior known non-minimal ones are not generated. The drawback of this approach is its complexity which is convergent with that of the approach for finding all reducts.

Another more general approach is based on well-known breadth search strategy. An empty set of attributes, alternatively the set consisting of core attributes, is iteratively increased by one attribute each time. If a given subset is not a reduct, the previously added attribute is removed and another one not used so far is added to the subset. This approach guarantees to check only subsets with the cardinality not higher than that of the minimal reducts. However, the solution may be expensive if the number of all attributes that describe the data is relatively high.

A modification of the above approach relies on the use frequency function that determines the order in which the attributes are to be added to a subset. Namely, the attribute most frequently appearing in the discernibility matrix is considered as the first one.

One can also find more concrete approaches for finding one or all minimal reducts.

A genetic algorithm was adapted in [49] for finding a minimal reduct. Each subset of the attribute set is an individual and is represented by a bit string where '1' ('0') means that the attribute appears (does not appear) in the subset. The fitness function is defined based on the subset cardinality and the number of rows in the discernibility matrix that are covered by the attribute subset. The best individual from the last generation is returned as a minimal reduct. The approach can be fast only if the stop criterion is

easy to reach (e.g. a low number of generations), and it does not guarantee that the found reduct is always minimal.

A similar, yet simpler solution to that from [49] was proposed in [45]. The authors cast the problem of finding minimal reducts in the framework of particle swarm optimization. Thanks to that complex operators such as crossover and mutation are not needed, namely only primitive and simple mathematical operators are required. It was experimentally verified that the proposal is computationally less expensive in terms of run-time and memory compared with GA approach.

To find a minimal reduct, the authors of [1] transform the data set into the binary integer programming (BIP) model and attribute reduction is seen as a satisfiability problem. To solve the problem using the BIP model, a branch and bound algorithm is applied. An experimental verification showed that the approach significantly decreases the number of rules that can be generated based on the obtained reducts.

In [12], the problem of finding the minimal rough set reducts is reformulated in a propositional satisfiability framework (SAT). Clauses of features in conjunctive normal (CNF) form are generated from the data set. Clauses are satisfied if after assigning true values to their all variables the formula is true in the data set. The task is to find the smallest number of such features so that the CNF formula is satisfied. The SAT problem is solved in this approach using the Davis–Logemann–Loveland algorithm (DPLL). The approach was experimentally compared with the one based on positive region reduct evaluation. The proposed approach is more time-consuming but in contrast to the reference one always returns minimal reducts.

Particle swarm optimization (PSO) technique is adapted in [46] to find minimal rough set reducts. Particles from PSO correspond to attributes. Particles position is a binary bit string of the length equal to the total attribute number. Each bit of a string defines if an attribute selected (1) or not selected (0). Each position corresponds to an attribute subset. Finding of minimal reducts is done according a standard PSO algorithm. The approach is a relatively fast and guarantees to always find minimal reducts.

A modification of a positive region-based attribute reduction approach is used for finding a minimal reduct in [51]. The attribute reduction process is controlled by the threshold that is the minimal number of attributes needed to distinguish all objects in the data set.

A combination of rough set theory and fish swarm algorithm was used in [34] to find a minimal reduct. The attribute core computed from the discernibility matrix is iteratively incremented by one attribute until a minimal reduct is found. Sets of non-core attributes with the same cardinality are encoded into integers as the individuals of the fish swarm algorithm. Attribute dependency is applied

to calculate the fitness values of the attribute subsets. In experiments, the proposal obtained in general a better performance than approaches from [12, 17] in terms of the accuracy of the found minimal reduct.

1.2 Hardware approaches for finding reducts

Hardware support of rough set-based data processing has a long history and is strictly connected with the rough set theory beginnings. The idea of a sample processor that generates classification rules from decision tables was described by Pawlak in [27]. In the proposed solution, the decision is made after the sequence of calculations of the factors describing the quality of a decision (strength, certainty, coverage). The structure of the processor allows using either the digital or the analog arithmometer. Other early researches include the construction of a rough sets processor based on the cellular networks, presented by Lewis, Perkowski, and Jozwiak in [21] as well as a concept of a hardware device capable of minimizing the large logic functions created on the basis of discernibility matrix, developed by Kanasugi and Yokoyama [17].

One of the first real systems for supporting the rough set data processing was described in [16]. Authors presented the design and implementation of the rough set processor and showed the high acceleration in computations: the proposed processor was ten times faster than PC, despite the clock frequency lower about 70 times. In [35], the FPGA-based implementation of the rough set methods for the technology research of fault diagnosis is presented. The FPGA-based hardware is used among others for data reduction and rule extraction. The same authors in [36] showed hardware system designed for the discretization of the continuous attributes. Authors proposed an algorithm that combines the advantages of the FPGA and the combined attributes dependency degree. All these solutions achieved a high acceleration compared to the software implementations and proved that FPGA-based implementations are currently one of the most significant research problems.

Hardware implementation of the reduct generation problem is described in [43]. Authors designed and implemented a reduct calculation block that utilizes the binary discernibility matrix and put them into the rough set processor used for the robotics application.

Real applications of the FPGA-based rough set method data processing systems are described in [41, 42]. In [41], the rough set coprocessor is described. Authors used the dual port RAM as well as pipelining to increase the efficiency thus making the solution suitable for the real-time applications. For the reduct generation, there is a discernibility matrix used. In [42], there is an intelligent system for medical applications described. The system was

tested for breast cancer detection and achieved a high acceleration compared to the general-purpose processor even though the much lower clock frequency.

A previous work of one of the authors on the hardware implementations of the rough sets methods was introduced in [6], where the system for calculating the core and the superreduct is described. The core and reduct are calculated from the discernibility matrix obtained from the decision table stored in FPGA. An other superreduct generating device was described in [18]. None of the described previously solutions allow calculating all the reducts.

The goal of this paper is to provide an efficient approach for finding minimal reducts. The contribution of this work can be summarized as follows:

1. Experimental investigation of the performance of two breadth search strategies for attribute reduction, i.e. blind- and frequency-based approaches, using their software implementation.
2. Design and implementation of hardware versions of both strategies using an FPGA device.
3. Verification how much the performance of attribute reduction using the strategies can be improved in terms of run-time when software implementation is replaced with its hardware equivalent.

The novelty of this work is to provide the first hardware design and implementation of an algorithm for finding minimal reducts. The key finding of this study is that the speed up factor of the hardware approach compared with the software one is over 20 for both strategies used for computing minimal reducts. This work can be treated as preliminary investigation for designing a powerful tool for finding minimal reducts in bigger data sets.

The remaining part of this work can be outlined as follows. Firstly, algorithms of blind- and frequency-based breath search strategies are defined (Sect. 2.1) and investigated using software implementation as well as their important properties are discovered (Sect. 3.1). Secondly, FPGA architectures of the chosen approaches are proposed (Sect. 2.2), implemented and previously found properties for software implementation are verified for the hardware approach (Sect. 3.2). Finally, important differences between the two strategies and between the two implementations are discussed (Sect. 4).

2 Finding minimal reducts using breadth search strategies

This section provides details of software and hardware approaches for finding minimal reducts using blind- and frequency-based breath search strategies.

Before moving to the approaches, the basic notions that regard attribute reduction will be introduced.

To store the data to be processed, a decision table is used.

Definition 1 [26] A decision table is a pair $DT = (U, A \cup \{d\})$, where U is a non-empty finite set of objects, called the universe, A is a non-empty finite set of condition attributes, and $d \notin A$ is the decision attribute.

Each attribute $a \in A \cup \{d\}$ is treated as a function $a : U \rightarrow V_a$, where V_a is the value set of a .

To compute reducts, the discernibility matrix of the decision table can be used.

Definition 2 [30] The discernibility matrix $(c_{x,y}^d)$ of a decision table $DT = (U, A \cup \{d\})$ is defined by $c_{x,y}^d = \{a \in A : a(x) \neq a(y), d(x) \neq d(y)\}$ for any $x, y \in U$.

A reduct is defined as follows.

Definition 3 A subset $B \subseteq A$ is a reduct in a decision table $DT = (U, A \cup \{d\})$ if and only if

1. $\forall x, y \in U, d(x) \neq d(y) \exists a \in c_{x,y}$,
2. $\forall C \subseteq B \exists x, y \in U, d(x) \neq d(y) \forall a \in C a \notin c_{x,y}$.

A reduct with the minimal cardinality is a minimal one.

Example 1 Given a decision table $DT = (U, A \cup \{flu\})$ of patients who are suspected to be sick with flu, where $A = \{temperature, headache, weakness, nausea\}$.

$U \setminus A \cup \{flu\}$	Temperature	Headache	Weakness	Nausea	Flu
1	Very high	Yes	Yes	no	Yes
2	Normal	No	No	No	No
3	High	No	No	No	No
4	Normal	No	Yes	No	Yes
5	Normal	No	Yes	No	No
6	High	Yes	No	Yes	Yes
7	Very high	No	No	No	No
8	Normal	Yes	Yes	Yes	Yes

We obtain the following discernibility matrix $(c_{x,y})$ (For simplicity's sake, the attributes names are abbreviated to their first letters and only the part of the matrix under the diagonal is shown since the matrix is symmetric.).

$$\begin{pmatrix} \emptyset & \{t, h, w\} & \{t, h, w\} & \emptyset & \{t, h\} & \emptyset & \{h, w\} & \emptyset \\ & \emptyset & \emptyset & \{w\} & \emptyset & \{t, h, n\} & \emptyset & \{h, w, n\} \\ & & \emptyset & \{t, w\} & \emptyset & \{h, n\} & \emptyset & A \\ & & & \emptyset & \emptyset & \emptyset & \{t, w\} & \emptyset \\ & & & & \emptyset & A & \emptyset & \{h, n\} \\ & & & & & \emptyset & \{t, h, n\} & \emptyset \\ & & & & & & \emptyset & A \\ & & & & & & & \emptyset \end{pmatrix}.$$

We obtain the following set of reducts $\{\{h, w\}, \{t, w, n\}\}$, where the set of minimal reducts is $\{\{h, w\}\}$.

2.1 Software approach

The blind version of breadth search-based strategy (see Algorithm 1) starts with computing the discernibility matrix and attribute core. Next, all combinations of the smallest cardinality (i.e. each combination consists of the core attributes and one additional attribute) are checked for being a reduct. The searching is interrupted after finding the required number of reducts or after checking all combinations and finding at least one reduct. If no reduct is found, all combinations with the cardinality greater by one (i.e. each combination is constructed based on a combination from the previous cardinality level by adding another attribute) are checked in the same way. Combinations are constructed according to alphanumeric order, thanks to this each new combination is not a repetition of any previously generated.

Algorithm 1: GenerateMinReducts

Data: $DT = (U, A \cup \{d\})$ – a decision table; k – the number of minimal reducts to be found (0 – all reducts);

Result: MR – a set of first k minimal reducts found;

```

begin
  MR := ∅;
  DM := computeDiscMatrix(DT);
  C := computeCore(DM);
  S := ∅;
  while MR = ∅ do
    if S = ∅ then S = {C};
    else S := computeAllCombs(A, S);
    foreach s ∈ S do
      if isReduct(DM, S) then
        MR = MR ∪ {S};
        if |MR| = k then break;
      end
    end
  end
end

```

Algorithm 1 uses the following functions:

1. $computeDiscMatrix(DT)$ computes the discernibility matrix for the decision table DT ;
2. $computeCore(DM)$ computes the attribute core for the discernibility matrix DM .
3. $computeAllCombs(A, S)$ computes for each attribute combination $S \in \mathcal{S}$ its all combinations (by adding one attribute) that have not been checked so far.
4. $isReduct(DM, S)$ checks if the attribute combination S is a reduct for the discernibility matrix DM .

The frequent version of breadth search-based strategy (see Algorithm 2) starts with the same computations as the blind one, i.e. the discernibility matrix and attribute core. Next, for each attribute that is not included in the core, its occurrence frequency in the discernibility matrix is computed. Combinations with the smallest cardinality (i.e. the core attributes plus one attribute) are constructed starting with the most frequent attribute and finishing after using all attributes or the l most frequent attributes (l defined by a user). The combinations are verified for being a reduct in the same way as by the blind version. Combinations of a higher cardinality level are constructed based on those from the previous level by adding one attribute chosen according the occurrence frequency order. One should stress that this way of attribute combination construction does not guarantee that only unique combinations are generated. Hence, each new combination is first compared to other combinations of the same cardinality level that have been generated so far.

Algorithm 2: *GenerateMinReductsFreq*

Data: $DT = (U, A \cup \{d\})$ – a decision table; k – the number of minimal reducts to be found (0 – all reducts); l – the number of first frequent attributes to be checked (0 – all reducts);

Result: MR – a set of first k minimal reducts found;

begin

$MR := \emptyset$;

$DM := \text{computeDiscMatrix}(DT)$;

$C := \text{computeCore}(DM)$;

$\mathcal{S} := C$; // the set of attribute combinations to be checked

while $MR = \emptyset$ **do**

$\mathcal{S}' := \emptyset$; // the set of attribute combinations checked so far

foreach $S \in \mathcal{S}$ **do**

$order := \text{sort}(\text{computeAttrFreq}(DM, A \setminus S))$;

// the attributes ordered descending according to their frequency

if $l = 0$ **then** $l' = |order|$; **else** $l' = l$;

$i := 0$;

while $i < l'$ **do**

$S' = S \cup order[i]$;

if $S' \in \mathcal{S}'$ **then continue**;

if $\text{isReduct}(DM, S')$ **then**

$MR = MR \cup \{S'\}$;

if $|MR| = k$ **then break**;

end

$i := i + 1$;

$\mathcal{S}' := \mathcal{S}' \cup \{S'\}$;

end

end

$\mathcal{S} := \mathcal{S}'$;

end

end

Compared with Algorithm 1, additional function employed by Algorithm 2 is $\text{computeAttrFreq}(DM, B)$ that computes for each attribute from B its frequency in the discernibility matrix DM .

One should note that due to introducing the l parameter Algorithm 2 is in fact a combination of breadth and depth search strategies. Namely,

- $l = 0$ —breadth search strategy (the beam is all attributes);

- $l = 1$ —depth search strategy (the beam is one attribute);
- $l > 1$ —limited breadth search strategy (the beam limited to l attributes), extended depth search strategy (the beam extended to l attributes);

2.2 Hardware approach

The below subsections describe the used hardware, the data set, and the implemented blocks for finding minimal reducts.

2.2.1 Used hardware

The project was implemented on Intel Arria V SoC (5ASTFD5K3F40I3) which is 28 nm FPGA with 462K logic elements and integrated with ARM Cortex-A9 dual-core processor clocked at 1.05 GHz. In our solution, the embedded processor was not used, the Nios II softcore processor was utilized instead. The choice was made because of easier debugging on registry and ALU levels. Nios II was clocked at 50 MHz, which gave a better chance to observe the differences between the solutions.

Software used for VHDL compilation, synthesis and generation of configuration files was Quartus Prime 17.1.0 Build 590 10/25/2017 SJ Standard Edition. C code was compiled with Eclipse Mars.2 Release (4.5.2) Build 20160218-0600 with GCC compiler version 4.8.3.

2.2.2 Used data and its structure

For this project, the authors used data about children with insulin-dependent diabetes mellitus (type 1) [32]. The data consist of 107 cases. Each case has 12 attributes that in the binary representation make up a total of 16-bit word:

- **Sex**—1 bit
- **Age of disease diagnosis**—2 bits
- **Disease duration**—2 bits
- **Appearance diabetes in the family**—1 bit
- **Insulin therapy type**—1 bit
- **Respiratory system infections**—1 bit
- **Remission**—1 bit
- **HbA1c**—2 bits
- **Hypertension**—1 bit
- **Body mass**—2 bits
- **Hypercholesterolemia**—1 bit
- **Hypertriglyceridemia**—1 bit

There is one 1-bit decision attribute:

- **Microalbuminuria**

Data are divided into two separate sets:

- **Positive**—with 56 cases that have microalbuminuria (decision attribute value equals to ‘1’)
- **Negative**—with 51 cases that does not have microalbuminuria (decision attribute value equals to ‘0’)

The full database is included in ‘Appendix’.

Since this is a preliminary investigation on finding minimal reducts using an FPGA device, a database that can be fit in the memory was used. In Intel Arria V SoC, the size of data set is limited by number of logic elements and routing channels. However, processing data sets of an arbitrary size can be possible by adapting a data decomposition method for an FPGA device. This issue is to be a direction of the future work.

2.2.3 Implemented hardware blocks

The architecture of the proposed solution is presented in Fig. 1.

Full_system_wrapper is the top level file which contains the *freq_wrapper* and *nios* components, as well as connect them together. This block can be treated as the whole system and has two input ports:

- **clk**—50 MHz clock input for processor (*nios* module);
- **reset**—reset signal for resetting all the components.

This block has no output ports.

The whole system is composed of the following components:

- **nios**—Nios II/f (fast) processor instance used for controlling the execution of the algorithm. The main aim of this block is to find the reduct using application written in C and compiled using Eclipse Mars.2 Release (4.5.2) Build 20160218-0600 with GCC compiler version 4.8.3. It utilizes the remaining blocks of the system while executing the program to increase the speed: receives the attribute core calculated by FPGA and

according to it sends reduct candidates until ISRED signal changes the value to ‘0’. This block has four input ports:

- **clk**—50 MHz clock input;
- **reset**—signal for resetting the processor;
- **freqOut00...freqOut11**—16-bit word with frequency of each attribute;
- **CORE**—provides the attribute core from *freq_wrapper* component;
- **ISRED**—signal for indicating that the signal sent from MASK output is a reduct.

This block has one output port:

- **MASK**—sends to *freq_wrapper* a reduct candidate based on calculated *CORE* given on input. This signal is also used in other components.

• **freq_wrapper**—component in which attributes frequencies are calculated and sorted using the bitonic sort algorithm (presented in Fig. 2) from the most to the least frequent ones. Each result is represented by a 16-bit vector, where the first 4 bits contain the attribute number and the remaining 12 bits are the number of occurrences. *freq_wrapper* contains *DMBlockWrapper* component that is responsible for calculating the core and reduct. This block has one input port:

- **MASK**—reduct candidate provided from *nios* component. This block has 14 output ports:
 - **freqOut00...freqOut11**—16-bit word with frequency of each attribute;
 - **CORE**—calculated core provided from *DMBlockWrapper*;
 - **ISRED**—sends single bit which informs that the candidate passed from *nios* is a reduct.

Fig. 1 Solution architecture

The architecture of *freq_wrapper* block is shown in Fig. 3. As mentioned before, this component consists of one block *DMBlockWrapper* described below:

- **DMBlockWrapper**—wrapper for components that are used to calculate the core and reduct. It consists of *DM_block*, *Core_block*, 2856 generated *isZero* components and connects them together. When a reduct is found, then it sends to *freq_wrapper* 1-bit signal '0'. *DMBlockWrapper* component contains also positive and negative data sets described in Sect. 2.2.2 which is passed to *DM_block* module in *DT_pos* and *DT_neg* signals. This block has one input port:

- **MASK**—reduct candidate provided from *nios* component.

This block has three output ports:

- **Disc_matrix**—the array that contains the calculated discernibility matrix;
- **CORE**—sends the calculated core to *freq_wrapper* where it is passed to *nios* component
- **ISRED**—sends 1-bit signal, which indicates that vector given on MASK input is reduct or not to *freq_wrapper* where it is passed to *nios* component.

The connection among components inside the wrapper is shown in Fig. 4. The blocks that create *DMBlockWrapper* component are described below:

- **DM_block**—component that creates the discernibility matrix using the data given from generated 2856 *DM_comp* modules. This block has three input ports:
- **MASK**—reduct candidate provided from *nios* component;
- **DT_pos**—provides a 56-element array of 16-bit word positive class data set from *DMBlockWrapper* component;

Fig. 2 Bitonic sort schema

Fig. 3 *Freq_wrapper* block architecture

- **DT_neg**—provides a 51-element array of 16-bit word negative class data set from *DMBlockWrapper* component.

This block has one output port:

- **DM**—sends an array of 2856 12-bit words (which contains the whole discernibility matrix) to *DMBlockWrapper*

- **Core_block**—component where the core is calculated. This block consists of generated 2856 *DM_sing* components, which creates a cascade of elements. Connections between blocks are shown in Fig. 5. This block has one input port:

- **DM**—2856-element array of 12-bit words which contains whole discernibility matrix provided from *DMBlockWrapper*

This block has one output port:

- **CORE**—sends to *DMBlockWrapper* the core calculated from provided the discernibility matrix.

- **isZero**—component that looks for '0' in given 12-bit word by performing OR operation on every single bit from the provided vector. the result is passed to *DM_sing*, where it is used to determine whether the new discernibility matrix entry should be equal to '0' or not. This block has one input port:

- **inval**—12-bit word that is the result of comparison of two values from positive and negative data sets.

This block has one output ports:

- **res**—single bit which is '0' only if the signal given on input equals to '0'.

As mentioned before, *DM_block* consists of multiple generated *DM_comp* component, which is described below:

- **DM_comp**—component that compares positive and negative data value determined by decisive bit (positive values are assigned to persons with microalbuminuria). The comparison is made by XORing positive with negative values and then conjunction-joining it with given mask. *DM_comp* block uses *isZero* component for looking for any '0' in the result of the previous XORing and conjunction-joining. The result of these operations is single discernibility matrix entry. If *isZero* component

Fig. 4 DMBlockWrapper schema **Fig. 5** Connections inside Core_block

does not find any '0', the value entered to discernibility Matrix is equal to '0'. This block has three input ports:

- **MASK**—12-bit word that is used to attribute masking provided from *nios* component;
- **DT_pos**—decision table entry from the positive class data set (16-bit word) from *DMBlockWrapper* component;
- **DT_neg**—decision table entry from the negative class data set (16-bit word) from *DMBlockWrapper* component.

This block has one output port:

- **DM_entry**—sends the calculated entry for discernibility matrix (12-bit word) to *DM_block*.

As shown in Fig. 5, *Core_block* consists of multiple generated *DM_sing* component described below:

- **DM_sing**—component in which every discernibility matrix entry is checked whether it is singleton or not. It takes two values: the previous value from cascade and the value from the discernibility matrix. It checks whether the provided value from DM is a singleton or not. If the value from DM is a singleton, then OR operation with the cascade value is performed and the result is pushed to output. Otherwise, the cascade value is pushed to output. This block has two input ports:

- **prev**—provides 12-bit word from cascade;

- **DM**—provides 12-bit DM entry. This block has one output port:
- **res**—sends the 12-bit word result of OR operation or the cascade value to *Core_block*.

There are also minor blocks included in the system which are used just as logic containers:

- **rMux**—module in which all attributes are counted for the sorting purposes;
- **sort**—module used for the bitonic sort algorithm and responsible for the comparison of two given values. The schema of this block is presented in Fig. 2.
- **isSingleton**—module that checks whether the value given on input is a singleton or not.

2.2.4 Used resources

The project was built using the Quartus Prime by Altera. The compilation reports show the described below utilization of the FPGA resources:

- Logic utilization (in ALMs): 115,492 / 176,160 (9%)
- Total registers: 12763
- Total pins: 11 / 876 (<1%)
- Total block memory bits: 18,452,928 / 23,367,680 (36%)
- Total DSP Blocks: 13 / 1,090 (<1%)

The utilization of the FPGA resources is low. The project needed less than 10% of the adaptive logic modules (ALMs). It should be noted that the project uses the soft-core processor, which needs some resources (about 3553 of ALM was used to implement Nios II processor). Therefore, the total utilization can be even decreased after switching to embedded processor.

2.2.5 Implementation

All hardware components for calculating the discernibility matrix and the core and for checking candidates whether they are reducts or not were written in VHDL language.

Hardware implementation is supported by the application written in C language, which on the basis of data provided by the hardware determines the most likely candidate for a reduct.

The discernibility matrix is calculated in *DM_Block* which consists of many comparators, which compares values of two objects from the decision tables. The decision table is passed from *DMBlockWrapper* component, where both: negative and positive class were declared. Calculation results are passed back to *DMBlockWrapper*, where it is used to calculate the core for given data.

The core is calculated in *Core_Block* which consists of a cascade of singleton checkers. *Core_Block* is a combinational circuit and does not need clock signal, so calculation time depends only on the propagation time of FPGAs logic blocks.

The calculated core is also used as a first mask, which zeroizes already used attributes in the discernibility matrix. After zeroizing, *freq_wrapper* calculates for each attribute how many times it is used, and then sort these data from the most to least frequent ones. Sorted data are sent with the calculated core and *ISRED* flag (which informs whether the current mask is reduct or not) to Nios II processor instance.

In Nios, C application based on frequency data, types most likely candidate for reduct and send it back to *freq_wrapper* component as a mask. Mask which was sent from C application is checked for a reduct by FPGA. Also zeroizing and frequency calculation is made again.

New frequencies with *ISRED* flag are sent back to Nios processor which types the next candidate. The process is repeated until all minimal reducts are found.

3 Experiments

This section describes experimental research that concerns finding minimal reducts using software and hardware implementation of the chosen breadth search-based algorithms.

3.1 Software implementation

The software implementation was tested on six binary classification problem data sets (see Table 1). One of them (*diabetes*) was taken from [32] (see Sect. 2.2.2), the remaining ones from the UCI Repository (<https://archive.ics.uci.edu/ml/index.php>). The approach was implemented

Table 1 Characteristics of databases

sym	db	attr	obj	min red	red card
D1	diabetes	12	107	3	9
D2	australian credit appr.	14	690	5	3
D3	breast cancer	10	699	9	3
D4	parkinson	28	1040	2	1
D5	pima ind. diab.	8	768	12	3
D6	sonar, mines versus rocks	60	208	8	1

sym—the symbol of a database used in Tables 2, 3, and 4; db—database's name; attr, obj, min red—the number of attributes, objects, minimal reducts, respectively; red card—the cardinality of minimal reducts

in C++ and tested using a standard PC (Intel Core i7-7700, 3.6 GHz, 32 GB RAM, Windows 10).

In spite of the fact that the data sets are small, they are big and complex enough to observe essential differences between the blind (BBFS) and frequency (FBFS) based breadth search strategies employed for finding minimal reducts. However, to obtain more reliable results, the computations for each data set (i.e. the discernibility matrix and minimal reducts) were repeated 1000 times and the total run-time was provided as the final outcome.¹

Table 2 shows run-time expressed in seconds for finding first minimal reduct, all minimal reducts, and *l* minimal reducts where *l* is the number of all minimal reducts for a give data set. Results for FBFS that are better than for BBFS are written in bold. Analogously in the remaining tables.

One can prior state that FBFS is a less efficient approach for finding all minimal reducts since it has to check the same number of attribute combinations as BBFS and additionally to compute frequency of attributes occurring in the discernibility matrix. The goal of this experiment (i.e. finding all minimal reducts by both strategies) is then to show how big the computational overhead is for FBFS compared to BBFS. The remaining experiments (i.e. finding one minimal reduct and the concrete number of reducts) show how much the overhead can be balanced thanks to processing attribute according to its frequency. As the results show only for 3 out of 12 cases (the run-time written in bold), it was possible to totally balance the overhead.

A better performance of BBFS may be caused by the following factors:

1. BBFS in fact processes more attribute combinations, but a lot of them can be verified negatively in a very short time. Namely, often only a few cells from the

¹ The tested data sets are too small to check the result for one run only (the run-time is often lower than 0.001). The used number of repetitions, i.e. 1000, which was experimentally determined, is big enough to precisely measure the run-time of one run for smaller data sets.

discernibility matrix need to be used to check that a given attribute combination cannot be a reduct.

2. FBBS does not generate all attribute combinations, but instead it checks possible repetitions of them. In other words, processing attributes according to their frequency does not guarantee that all combinations to be checked are unique.

The goal of the next experimentation is to check how the attribute order influences the run-time. To this end, for each of the chosen data sets (*D1*, *D2*, and *D5*), 20 random attribute orders were tested. Table 3 shows the average run-time and the standard deviation (the first row for each data set). For *D1* and *D2*, the average run-time obtained for a random attribute order is clearly worse compared with that for the original attribute order. For *D3*, the results are similar, which means that the original attribute order is representative for this database in terms of the run-time for computing minimal reducts.

The reason why the attribute order may influence the run-time lies in the way how a candidate for a minimal reduct is compared to a discernibility matrix cell. Namely, the corresponding attributes are compared until the common attribute is found or all attributes are checked.

To illustrate this issue, consider the below two exemplary pairs of a candidate and cell. In the second pair, the order of attributes is changed. For the first pair, we need

only one iteration to find a common attribute, whereas for the second pair we need as many as six iterations.

a	b	c	d	e	f	g	h	b	c	d	e	f	a	g	h
1	0	1	0	1	0	0	0	0	1	0	1	0	1	0	0
1	1	0	0	0	1	1	0	1	0	0	0	1	1	1	0

To reduce the influence of the attribute order, a candidate for a reduct is expressed as a list of the positions of the attributes occurring in the combination. To compare a candidate to a cell, it is enough to check only attributes whose positions are given in the list corresponding to the candidate. For example, for the candidate from the first and second pair, we obtain the following lists $\langle 0, 2, 4 \rangle$ and $\langle 1, 3, 5 \rangle$, respectively. We need up to three iterations to check if the candidate and cell has a common attribute.

The results obtained using the improved comparison method are also shown in Table 3 (the second row for each database). This modification not only reduces the influence of the attribute order (i.e. the standard deviation values are clearly smaller) but also shortens the run-time. One can observe that results are definitely more stable for FBFS. The reason is that attribute frequency does not depend on the order in which attributes occur in the data set.

The goal of the last experimentation is to check how much the run-time can be shortened when a limited version of FBFS is used. In the preliminary experiments, for each data set the minimal number of the most frequent attributes that guarantee to find all minimal reducts was found.

Table 4 compares results obtained for BBFS, FBFS, and limited FBFS. All the strategies use the improved comparison method and were run on the data sets with original

Table 2 The run-time (in sec.) of finding minimal reducts using blind- and frequency-based strategies (1000 repetitions)

db	blind			freq		
	1	all	all*	1	all	all*
D1	0.654	0.872	0.773	0.934	1.779	1.003
D2	23.413	33.991	32.476	22.975	47.334	29.752
D3	16.448	28.32	24.247	35.976	86.001	41.584
D4	50.355	56.065	55.501	70.647	71.773	71.505
D5	13.189	22.296	22.246	22.202	48.361	33.36
D6	3.011	3.979	3.882	3.397	4.54	3.772

blind—blind breadth first strategy; freq—frequency-based breadth first strategy; 1—one minimal reduct, all—all minimal reducts, all*—all minimal reducts (the number known a priori)

Table 3 The average run-time (in sec.) and standard deviation of finding minimal reducts using blind- and frequency-based strategies for data with attributes randomly ordered

db	blind		freq	
	1	all*	1	all*
D1	0.760 (0.095)	0.943 (0.107)	1.063 (0.071)	1.167 (0.067)
	0.707 (0.068)	0.897 (0.086)	1.003 (0.036)	1.049 (0.046)
D2	24.018 (2.586)	36.838 (5.316)	24.964 (1.778)	36.506 (5.022)
	22.579 (1.533)	31.211 (2.074)	22.527 (0.383)	28.103 (0.463)
D5	12.978 (1.005)	20.181 (1.858)	22.542 (0.772)	33.054 (1.898)
	12.535 (0.961)	18.323 (0.893)	18.081 (0.654)	24.764 (0.659)

Table 4 The run-time (in sec.) of finding minimal reducts using blind-, frequency-, and limited frequency-based strategies

db	blind	freq	ltd freq
D1	0.796	1.021	0.869
D2	30.778	27.952	25.875
D3	23.5	34.242	27.183
D4	51.002	62.386	62.385
D5	19.26	25.566	21.681
D6	3.077	3.241	3.249

attribute order. The run-time obtained by limited FBFS can in general be significantly shortened compared with FBFS but not with BBFS. This experimentation shows that even if it could be possible to a priori find the minimal number of the most frequent attributes that guarantee to find all minimal reducts, BBFS is still a better choice for most of data sets.

3.2 Hardware implementation

For experimental research, an application for computing the core and minimal reducts was written in C language. It works in two modes: a standalone software and a support for the hardware system described in Sect. 2.2.5. The main difference between the two modes is the transfer of the logic responsible for computing the core and reducts to the FPGA. In fact, a standalone mode can be shortly described as software simulation of the algorithm implemented in hardware. Every calculation was performed 10, 100, 1000, and 10000 times in order to obtain the most precise time result. All experiments were performed on the data set described in Sect. 2.2.2.

In the beginning, tests were performed by using C application in standalone mode, launched on Nios II processor which uses a 50 MHz clock. The purpose of these tests was to examine the average time needed to calculate the core and the reducts with ‘clean’ C application before coupling it with FPGA circuit.

The goal of the first experiment is to check how much time takes the calculations when using the BBFS. Table 5 shows run-time for finding all minimal reducts for the given data set and time needed to perform one execution.

The second experiment checks the time needed for the calculations when using the frequency-based breadth search strategy. Results presented in Table 6 compared to the previous test show that strategy based on frequencies of every attribute is about 1.38 times faster and thus better.

After determining the reference times, C application was run as the support for the created hardware system. Next experiments were performed exactly like the previous two, but this time calculations were done by FPGA circuit.

In the third experiment, the BBFS was used. The results presented in Table 7 show run-time and the average time

Table 5 Reduct computation using implementation written in C launched on Nios II—blind strategy

Times calculated	Time in (s)	Time in (s) per one execution
10	0.657	0.0657
100	6.572	0.06572
1000	65.728	0.065728
10000	657.272	0.0657272

Table 6 Reduct computation using implementation written in C launched on Nios II—FBFS

Times calculated	Time in (s)	Time in (s) per one execution
10	0.732	0.0732
100	7.327	0.0732
1000	73.275	0.073275
10000	732.749	0.0732749

Table 7 Reduct computation using FPGA to calculate core and check reduct correctness—blind strategy

Times calculated	Time in (s)	Time in (s) per one execution
10	0.006	0.0006
100	0.059	0.00059
1000	0.590	0.000590
10000	5.898	0.0005898

Table 8 Reduct computation using FPGA to calculate core and check reduct correctness—FBFS

Times calculated	Time in (s)	Time in (s) per one execution
10	0.005	0.0005
100	0.048	0.00048
1000	0.482	0.000482
10000	4.817	0.0004817

needed for one execution. Comparing the result with the software solution, it can be concluded that the execution time is almost the same in hardware and software implementations.

The goal of the last experiment is to check how much time is taken to calculate the core and the reducts when using the frequency-based breadth search strategy. The results presented in Table 8 show that this strategy gives almost the same values like the BBFS.

The results presented in Tables 7 and 8 show that if combinational logic is used to solve the problem, it does not matter what kind of algorithm is used, because everything depends mostly on signal propagation time.

For test purposes, C application in standalone mode was run on PC with Intel Core i7-770 @ 3.60 GHz and 32 GB RAM.

As can be seen in Tables 9 and 10, the results are different than those from FPGA and just like in C++ application the BBFS was a little bit faster than the strategy based on frequencies. The obtained results are also about 3.5 times (in the case of BBFS) and about 2.7 times (in the case of FBFS) better than results from FPGA. It should be noticed, that PCs clock is $\frac{\text{clk}_{\text{PC}}}{\text{clk}_{\text{FPGA}}} = \frac{3600\text{MHz}}{50\text{MHz}} = 72$ times faster than FPGA clock source, so if we take that difference,

Table 9 Reduct computation using PC to calculate core and check reduct correctness—BBFS

Times calculated	Time in (s)	Time in (s) per one execution
10	0	0
100	0.015622	0.00015622
1000	0.177181	0.000177181
10000	1.696734	0.0001696734

Table 10 Reduct computation using PC to calculate core and check reduct correctness—FBFS

Times calculated	Time in (s)	Time in (s) per one execution
10	0	0
100	0.015652	0.00015652
1000	0.187467	0.000187467
10000	1.880553	0.0001880553

the results are much better—the speed up factor is about 21 for the BBFS and about 28 for FBFS.

4 Conclusion

We have defined and investigated two algorithms realizing breadth search strategies-based approaches for finding minimal reducts, i.e. blind- and frequency-based versions. We have also designed and implemented the first FPGA architectures for finding minimal reducts. Based on the experimental research reported in this paper, we can formulate good practices of application of the type of breadth search strategy as well as the type of its implementation. The most important remarks on the both strategies and implementations are given below.

1. Software implementation.

BBFS in general is more efficient than FBFS not only for finding all minimal reducts but also for finding the first minimal reduct. The main reasons of that are the following.

- In contrast to FBFS, BBFS only checks unique attribute combinations for being reducts.
- For data sets with a higher number of attributes, BBFS checks, in fact, a higher number of attribute combinations compared with FBFS, but a lot of them can quickly be verified as negative (non-reducts).

2. Hardware implementation.

In contrast to the software implementation, for an FPGA, FBFS is more efficient. This is because most of the blocks in the hardware implementation are the combinational logic ones. Consequently, they calculate the results almost instantly (only data propagation time delays the calculation).

The great benefit of the FPGA is the possibility to use the hardware sorting block; thus, the FBFS is more efficient than BBFS.

The research done in this paper is a fundamental step towards developing a system for efficient reduction of large data sets. The experimental research has provided an answer which breadth search strategy and which combination of software and hardware implementations are more suitable for finding minimal reducts. The next two steps are the following.

- Make the hardware implementation more flexible so that it can be used for data sets with arbitrary number of attributes. It can be done by adapting a vertical data decomposition approach (see, e.g. [54]).
- Make the hardware implementation scalable so that it can be used for data sets with arbitrary number of objects. The horizontal data decomposition approach proposed in [8] is to be used for this purpose.

Acknowledgements This work was supported by the Grant S/WI/1/2018 of the Polish Ministry of Science and Higher Education.

Compliance with ethical standards

Conflict of interest All authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

Appendix

The diabetes database, divided into both classes, is given in Tables 11 and 12.

Table 11 Diabetes—cases with microalbuminuria

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	d
f	1	0	No	KIT_IIT	Yes	No	0	Yes	3–97	No	No	Yes
f	2	0	No	KIT_IIT	Yes	No	1	No	3–97	No	No	Yes
f	1	0	Yes	KIT	Yes	No	1	No	3–97	Yes	No	Yes
m	2	0	No	KIT_IIT	Yes	No	1	Yes	3–97	No	No	Yes
f	2	1	No	KIT_IIT	Yes	No	0	Yes	3–97	No	No	Yes
m	2	0	No	KIT	Yes	Yes	1	No	3–97	No	No	Yes
m	1	1	No	KIT_IIT	No	No	0	No	3–97	Yes	No	Yes
m	3	0	No	KIT	Yes	No	1	Yes	3–97	No	No	Yes
f	1	2	Yes	KIT_IIT	Yes	No	1	No	3–97	Yes	No	Yes
f	1	0	No	KIT_IIT	Yes	No	2	No	3–97	Yes	No	Yes
m	1	0	No	KIT	No	No	1	No	3–97	No	No	Yes
m	1	1	No	KIT	No	No	2	No	3–97	Yes	Yes	Yes
f	1	1	No	KIT	Yes	No	1	No	3–97	No	No	Yes
f	1	0	No	KIT	No	No	0	No	< 3	No	No	Yes
f	1	1	No	KIT_IIT	Yes	Yes	1	No	3–97	No	No	Yes
f	1	1	No	KIT	Yes	No	1	No	3–97	No	No	Yes
m	1	0	No	KIT	Yes	No	1	No	3–97	No	No	Yes
f	2	0	No	KIT	Yes	No	1	No	3–97	No	No	Yes
f	1	1	No	KIT_IIT	No	Yes	1	No	3–97	No	No	Yes
f	1	1	Yes	KIT_IIT	Yes	Yes	1	No	3–97	No	No	Yes
m	2	0	No	KIT_IIT	No	No	1	Yes	3–97	No	No	Yes
f	1	1	No	KIT_IIT	Yes	No	0	No	3–97	Yes	No	Yes
m	1	1	No	KIT	Yes	Yes	1	Yes	3–97	Yes	No	Yes
m	1	0	No	KIT_IIT	Yes	No	0	Yes	3–97	No	No	Yes
f	1	1	Yes	KIT	Yes	No	2	Yes	3–97	Yes	Yes	Yes
f	2	0	No	KIT_IIT	Yes	No	1	No	> 97	Yes	No	Yes
f	3	0	No	KIT_IIT	No	No	1	No	3–97	No	No	Yes
m	1	2	No	KIT	Yes	No	1	Yes	3–97	No	No	Yes
f	1	1	No	KIT_IIT	Yes	No	2	No	3–97	Yes	No	Yes
f	0	0	No	KIT	Yes	No	0	No	< 3	No	No	Yes
f	0	0	No	KIT	Yes	No	0	No	3–97	Yes	No	Yes
f	0	2	No	KIT_IIT	No	No	1	No	3–97	No	No	Yes
m	1	1	Yes	KIT	Yes	No	1	Yes	3–97	Yes	Yes	Yes
f	1	1	No	KIT	No	No	2	No	> 97	Yes	Yes	Yes
f	2	0	Yes	KIT_IIT	Yes	No	2	No	< 3	Yes	Yes	Yes
m	2	0	No	KIT	Yes	No	0	Yes	3–97	No	No	Yes
m	2	0	No	KIT_IIT	Yes	No	1	No	3–97	Yes	Yes	Yes
m	0	0	No	KIT_IIT	Yes	No	0	No	3–97	No	No	Yes
m	2	1	Yes	KIT	No	No	0	No	3–97	No	No	Yes
f	1	1	No	KIT_IIT	Yes	Yes	2	Yes	3–97	No	No	Yes
f	1	1	No	KIT_IIT	Yes	No	0	No	> 97	Yes	No	Yes
m	1	0	No	KIT_IIT	Yes	No	1	No	< 3	Yes	No	Yes
f	1	0	No	KIT_IIT	Yes	No	2	No	3–97	Yes	Yes	Yes
m	1	1	No	KIT	No	No	0	No	3–97	No	No	Yes
m	1	1	No	KIT_IIT	Yes	Yes	1	No	3–97	Yes	No	Yes
m	1	0	No	KIT_IIT	Yes	No	1	No	3–97	No	No	Yes
f	1	1	Yes	KIT	Yes	No	2	No	3–97	Yes	Yes	Yes
m	1	1	No	KIT	Yes	No	0	No	< 3	No	No	Yes

Table 11 (continued)

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	d
f	2	1	No	KIT	Yes	No	0	No	> 97	No	No	Yes
f	0	0	No	KIT	Yes	No	0	No	3–97	No	No	Yes
m	1	1	No	KIT	Yes	No	2	No	3–97	Yes	Yes	Yes
f	1	1	No	KIT	Yes	No	0	No	> 97	Yes	No	Yes
f	2	1	Yes	KIT	Yes	No	0	No	3–97	Yes	No	Yes
f	2	0	No	KIT	Yes	No	0	Yes	< 3	No	No	Yes
f	2	0	No	KIT_IIT	No	No	0	No	3–97	No	No	Yes
m	1	1	No	KIT	Yes	No	2	Yes	3–97	Yes	Yes	Yes

Table 12 Diabetes—cases without microalbuminuria

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	d
m	0	0	No	KIT	Yes	No	2	No	3–97	No	No	No
m	2	0	Yes	KIT	Yes	No	0	No	3–97	No	No	No
f	3	0	Yes	KIT_IIT	No	No	0	No	3–97	No	No	No
m	1	0	No	KIT	Yes	No	0	No	3–97	No	No	No
m	0	1	Yes	KIT	Yes	No	2	No	3–97	No	No	No
f	1	0	No	KIT	Yes	No	1	No	3–97	No	No	No
f	2	0	No	KIT	Yes	Yes	0	No	3–97	No	No	No
m	3	0	No	KIT_IIT	No	No	0	No	3–97	No	No	No
m	2	1	No	KIT_IIT	Yes	No	0	Yes	3–97	No	No	No
m	1	0	No	KIT	No	No	2	No	3–97	No	No	No
m	2	1	No	KIT	Yes	Yes	1	Yes	3–97	No	No	No
m	1	0	Yes	KIT_IIT	Yes	No	1	No	3–97	Yes	No	No
f	3	0	No	KIT	No	Yes	0	No	3–97	No	No	No
f	0	0	No	KIT	No	No	1	No	3–97	No	No	No
m	2	0	No	KIT_IIT	Yes	No	0	No	3–97	No	No	No
m	1	0	Yes	KIT_IIT	No	No	1	No	3–97	No	No	No
m	0	2	No	KIT_IIT	No	No	1	No	3–97	No	No	No
m	1	0	Yes	KIT_IIT	No	No	0	No	3–97	Yes	No	No
m	0	0	Yes	KIT_IIT	Yes	No	1	No	3–97	No	No	No
m	1	0	No	KIT	No	No	0	No	< 3	Yes	No	No
m	1	0	No	KIT	Yes	Yes	0	No	3–97	No	No	No
f	1	0	Yes	KIT_IIT	Yes	No	0	No	3–97	No	No	No
m	0	0	No	KIT	No	No	0	No	3–97	No	No	No
f	0	1	No	KIT	Yes	No	1	No	< 3	No	Yes	No
f	1	0	No	KIT_IIT	No	No	1	Yes	3–97	No	No	No
m	0	1	Yes	KIT	Yes	No	1	No	3–97	No	No	No
m	1	0	No	KIT	Yes	No	0	No	3–97	No	No	No
m	1	1	Yes	KIT_IIT	No	No	1	No	3–97	No	No	No
f	1	1	Yes	KIT	No	No	0	No	3–97	No	No	No
m	1	0	No	KIT_IIT	Yes	No	1	No	< 3	Yes	Yes	No
f	0	1	No	KIT	No	No	0	No	3–97	No	No	No
m	1	1	No	KIT	Yes	No	1	No	3–97	No	No	No
m	1	0	Yes	KIT_IIT	Yes	No	1	No	3–97	Yes	No	No
m	0	1	Yes	KIT	No	No	1	No	3–97	Yes	Yes	No

Table 12 (continued)

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	d
m	2	0	Yes	KIT	Yes	No	0	No	3–97	No	No	No
m	1	1	No	KIT	Yes	Yes	0	No	3–97	No	No	No
m	0	0	Yes	KIT	Yes	No	0	No	3–97	No	No	No
f	0	1	Yes	KIT	Yes	No	0	No	3–97	No	Yes	No
f	0	1	No	KIT	Yes	No	1	No	3–97	No	No	No
m	2	1	No	KIT	No	No	1	No	3–97	No	No	No
f	2	1	No	KIT	Yes	No	1	No	3–97	No	No	No
f	1	2	No	KIT	Yes	No	2	No	3–97	Yes	Yes	No
m	0	1	Yes	KIT	Yes	No	0	No	3–97	No	No	No
f	1	1	No	KIT	Yes	No	0	Yes	3–97	No	No	No
m	0	0	No	KIT	Yes	No	0	No	3–97	Yes	No	No
m	3	1	No	KIT	Yes	No	0	No	3–97	Yes	No	No
m	0	2	No	KIT_IIT	No	No	0	Yes	3–97	No	No	No
m	0	1	Yes	KIT	Yes	No	2	No	3–97	Yes	Yes	No
m	2	1	No	KIT_IIT	Yes	Yes	1	Yes	3–97	No	No	No
m	0	0	No	KIT	Yes	No	0	No	3–97	No	No	No
f	0	0	No	KIT	Yes	No	1	No	3–97	Yes	No	No

The full attribute names are the following:

- a_1 = Sex,
 a_2 = Age of disease diagnosis,
 a_3 = Disease duration,
 a_4 = Appearance diabetes in the family,
 a_5 = Insulin therapy type,
 a_6 = Respiratory system infections,
 a_7 = Remission,
 a_8 = HbA1c,
 a_9 = Hypertension,
 a_{10} = Body mass,
 a_{11} = Hypercholesterolemia,
 a_{12} = Hypertriglyceridemia,
 d = Microalbuminuria.

References

- Bakar AA, Sulaiman MN, Othman M, Selamat MH (2002) Propositional satisfiability algorithm to find minimal reducts for data mining. *Int J Comput Math* 79(4):379–389
- Chen D, Zhao S, Zhang L, Yang Y, Zhang X (2012) Sample pair selection for attribute reduction with rough set. *IEEE Trans Knowl Data Eng* 24(11):2080–2093
- Czolombitko M, Stepaniuk J (2016) Attribute reduction based on mapreduce model and discernibility measure. In: Saeed K, Homenda W (eds) *Computer information systems and industrial management: 15th IFIP TC8 international conference. CISIM 2016*, Vilnius, Lithuania, Sept 14–16, 2016, Proceedings. Springer, Cham, pp 55–66
- Degang C, Changzhong W, Qinghua H (2007) A new approach to attribute reduction of consistent and inconsistent covering decision systems with covering rough sets. *Inf Sci* 177(17):3500–3518
- Dong Z, Sun M, Yang Y (2016) Fast algorithms of attribute reduction for covering decision systems with minimal elements in discernibility matrix. *Int J Mach Learn Cybern* 7(2):297–310
- Grześ T, Kopczyński M, Stepaniuk J (2013) FPGA in rough set based core and reduct computation. In: Lingras P, Wolski M, Cornelis C, Mitra S, Wasilewski P (eds) *Rough sets and knowledge technology*. Springer, Berlin, pp 263–270
- Grzymala-Busse J (1991) *An algorithm for computing a single covering*. Kluwer Academic Publishers, Berlin, p 66
- Hońko P (2016) Attribute reduction: a horizontal data decomposition approach. *Soft Comput* 20(3):951–966
- Hu Q, Yu D, Liu J, Wu C (2008) Neighborhood rough set based heterogeneous feature subset selection. *Inf Sci* 178(18):3577–3594
- Hu Q, Yu D, Xie Z, Liu J (2006) Fuzzy probabilistic approximation spaces and their information measures. *IEEE Trans Fuzzy Syst* 14(2):191–201
- Hu X, Cercone N (1995) Learning in relational databases: a rough set approach. *Comput Intell* 11(2):323–338
- Jensen R, Shen Q, Tuson A (2005) Finding rough set reducts with SAT. In: *Rough sets, fuzzy sets, data mining, and granular computing, 10th international conference, RSFDGrC 2005*, Regina, Canada, Aug 31–Sept 3, 2005, proceedings, part I, pp 194–203
- Jia X, Liao W, Tang Z, Shang L (2013) Minimum cost attribute reduction in decision-theoretic rough set models. *Inf Sci* 219:151–167
- Jiang Y, Yu Y (2016) Minimal attribute reduction with rough set based on compactness discernibility information tree. *Soft Comput* 20(6):2233–2243
- Jing F, Yunliang J, Yong L (2017) Quick attribute reduction with generalized indiscernibility models. *Inf Sci* 397–398:15–36
- Kanasugi A, Matsumoto M (2007) Design and implementation of rough rules generation from logical rules on FPGA board. In: Kryszkiewicz M, Peters JF, Rybinski H, Skowron A (eds) *Rough*

- sets and intelligent systems paradigms. Springer, Berlin, pp 594–602
17. Kanasugi A, Yokoyama A (2001) A basic design for rough set processor. In: Proceedings of the annual conference of JSAI, pp 65–65
 18. Kopczyński M, Grześ T, Stepaniuk J (2014) FPGA in rough-granular computing: reduct generation. In: Proceedings of the 2014 IEEE/WIC/ACM international joint conferences on web intelligence (WI) and intelligent agent technologies (IAT)—volume 02, WI-IAT '14, Washington, DC, USA. IEEE Computer Society, pp 364–370
 19. Kryszkiewicz M (1998) Rough set approach to incomplete information systems. *Inf Sci* 112(1–4):39–49
 20. Kryszkiewicz M (2001) Comparative study of alternative type of knowledge reduction in inconsistent systems. *Int J Intell Syst* 16:105–120
 21. Lewis T, Perkowski M, Jozwiak L (1999) Learning in hardware: architecture and implementation of an FPGA-based rough set machine. In: Proceedings 25th EUROMICRO conference. Informatics: theory and practice for the New Millennium, vol 1, pp 326–334
 22. Li F, Yang J (2016) A new approach to attribute reduction of decision information systems. In: Qin Y, Jia L, Feng J, An M, Diao L (eds) Proceedings of the 2015 international conference on electrical and information technologies for rail transportation: transportation. Springer, Berlin, pp 557–564
 23. Liang J, Mi J, Wei W, Wang F (2013) An accelerator for attribute reduction based on perspective of objects and attributes. *Knowl Based Syst* 44:90–100
 24. Liang J, Xu Z (2002) The algorithm on knowledge reduction in incomplete information systems. *Int J Uncertain Fuzziness* 10(1):95–103
 25. Liu G, Hua Z, Chen Z (2017) A general reduction algorithm for relation decision systems and its applications. *Knowl Based Syst* 119:87–93
 26. Pawlak Z (1991) Rough sets, theoretical aspects of reasoning about data. Kluwer Academic, Dordrecht
 27. Pawlak Z (2004) Elementary rough set granules: toward a rough set processor. In: Pal SK, Polkowski L, Skowron A (eds) Rough-neural computing: techniques for computing with words. Cognitive Technologies. Springer, pp 5–14
 28. Pawlak Z, Skowron A (2007) Rudiments of rough sets. *Inf Sci* 177(1):3–27
 29. Qian Y, Liang J, Pedrycz W, Dang C (2010) Positive approximation: an accelerator for attribute reduction in rough set theory. *Artif Intell* 174(9–10):597–618
 30. Skowron A, Rauszer C (1992) The discernibility matrices and functions in information systems. In: Intelligent decision support. Springer, Amsterdam, pp 331–362
 31. Ślęzak D (2002) Approximate entropy reducts. *Fundam Inform* 53(3,4):365–390
 32. Stepaniuk J (1999) Rough set data mining of diabetes mellitus data. *Lect Notes Comput Sci* 1906(Supplement C):457–465
 33. Stepaniuk J (2008) Rough-granular computing in knowledge discovery and data mining. Studies in computational intelligence, vol 152. Springer, Berlin
 34. Su Y, Guo J (2017) A novel strategy for minimum attribute reduction based on rough set theory and fish swarm algorithm. *Comput Int Neurosci* 2017:6573623:1–6573623:37
 35. Sun G, Qi X, Zhang Y (2011) A FPGA-based implementation of rough set theory. In: 2011 Chinese control and decision conference (CCDC), pp 2561–2564
 36. Sun G, Wang H, Lu J, He X (2013) A FPGA-based discretization algorithm of continuous attributes in rough set. Applied mechanics and materials, vol 278–280. Trans Tech Publications, Zurich
 37. Swiniarski R (2001) Rough sets methods in feature reduction and classification. *Int J Appl Math Comput Sci* 11(3):565–582
 38. Swiniarski RW, Skowron A (2003) Rough set methods in feature selection and recognition. *Pattern Recognit Lett*. 24(6):833–849
 39. Teng S-H, Lu M, Yang A-F, Zhang J, Nian Y, He M (2016) Efficient attribute reduction from the viewpoint of discernibility. *Inf Sci* 326:297–314
 40. Thi VD, Giang NL (2013) A method for extracting knowledge from decision tables in terms of functional dependencies. *Cybern Inf Technol* 13(1):73–82
 41. Tiwari K, Kothari A (2015) Design and implementation of rough set co-processor on FPGA. *Int J Innov Comput Inf Control* 11(2):641–656
 42. Tiwari K, Kothari A (2016) Design of intelligent system for medical applications using rough set theory. *Int J Data Min Model Manag* 8(3):279–301
 43. Tiwari KS, Kothari AG (2011) Architecture and implementation of attribute reduction algorithm using binary discernibility matrix. In: 2011 international conference on computational intelligence and communication networks, pp 212–216
 44. Wang C, He Q, Chen D, Hu Q (2014) A novel method for attribute reduction of covering decision systems. *Inf Sci* 254:181–196
 45. Wang X, Yang J, Peng N, Teng X (2005) Finding minimal rough set reducts with particle swarm optimization. In: Ślęzak D, Wang G, Szczuka M, Dütsch I, Yao Y (eds) Rough sets, fuzzy sets, data mining, and granular computing. Springer, Berlin, pp 451–460
 46. Wang X, Yang J, Teng X, Xia W, Jensen R (2007) Feature selection based on rough sets and particle swarm optimization. *Pattern Recogn Lett* 28(4):459–471
 47. Wei W, Liang J, Qian Y, Wang F, Dang C (2010) Comparative study of decision performance of decision tables induced by attribute reductions. *Int J Gen Syst* 39(8):813–838
 48. Wei W, Liang J, Wang J, Qian Y (2013) Decision-relative discernibility matrices in the sense of entropies. *Int J Gen Syst* 42(7):721–738
 49. Wroblewski J (1995) Finding minimal reducts using genetic algorithms. In: Proceedings of the second annual join conference on information sciences, pp 186–189
 50. Xie J, Shen X, Liu H, Xu X (2013) Research on an incremental attribute reduction based on relative positive region. *J Comput Inf Syst* 9(16):6621–6628
 51. Xu N, Liu Y, Zhou R (2008) A tentative approach to minimal reducts by combining several algorithms. In: Advanced intelligent computing theories and applications. With aspects of contemporary intelligent computing techniques, 4th international conference on intelligent computing, ICIC 2008, Shanghai, China, Sept 15–18, 2008, Proceedings, pp 118–124
 52. Yao Y, Zhao Y (2008) Attribute reduction in decision-theoretic rough set models. *Inf Sci* 178(17):3356–3373
 53. Ye D, Chen Z (2002) A new discernibility matrix and the computation of a core. *Acta Electron Sin* 30(7):1086–1088
 54. Ye M, Wu C (2010) Decision table decomposition using core attributes partition for attribute reduction. In: 5th international conference on computer science and education (ICCSE), vol 23. IEEE, pp 23–26
 55. Zhang W-X, Mi J-S, Wu W-Z (2003) Approaches to knowledge reductions in inconsistent systems. *Int J Intell Syst* 18(9):989–1000
 56. Zhang X, Mei C, Chen D, Li J (2013) Multi-confidence rule acquisition oriented attribute reduction of covering decision systems via combinatorial optimization. *Knowl Based Syst* 50:187–197

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.