Expert Systems With Applications 42 (2015) 9612–9619

Contents lists available at ScienceDirect

Expert Systems With Applications

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

A fast hardware software platform for computing irreducible testors

Vladimir Rodríguez-Diez[a], José Francisco Martínez-Trinidad[b], Jesús Ariel Carrasco-Ochoa[b], Manuel Lazo-Cortés[b], Claudia Feregrino-Uribe[b], René Cumplido[b,*]

[a] *Universidad de Camaguey, Carretera Circunvalación Norte Km. 5, Camaguey, 74650, Cuba*
[b] *Instituto Nacional de Astrofísica, Óptica y Electrónica, Tonanzintla, Puebla, 72840, Mexico*

ARTICLE INFO

Keywords:
Feature selection
Testor theory
Hardware architecture
FPGA

ABSTRACT

Among the systems involved with data and knowledge that give answers, solutions, or diagnoses, based on available information; those based on feature selection are very important since they allow us to solve important tasks into pattern recognition and decision making areas. Feature selection consists in finding a minimum subset of attributes that preserves the ability to discern between objects from different classes. Testor theory is a convenient way to solve this problem since a testor is defined as a subset of attributes that can discern between objects from different classes; and an irreducible testor is a minimal subset with this property. However, the computation of these minimal subsets is a problem whose space complexity grows exponentially regarding the number of attributes. Therefore, in the literature, several hardware implementations of algorithms for computing testors, which take advantage of the inherent parallelism in the evaluation of testor candidates, have been proposed. In this paper, a new fast hardware software platform for computing irreducible testors is introduced. Our proposal follows a pruning strategy that, in most cases, reduces the search space more than any other alternative reported in the literature. The experimental results show the runtime reduction achieved by the proposed platform in contrast to other state-of-the-art hardware and software implementations.

1. Introduction

Feature selection for supervised classification consists in identifying those attributes that provide relevant information for the classification process. This procedure does not only reduce the computational cost of the classification process by eliminating superfluous information, but in some cases it could even provide better classification accuracy. Testor theory can be used for feature selection as shown in (Martínez-Trinidad & Guzmán-Arenas, 2001; Ruiz-Shulcloper, 2008). A testor is defined as a subset of attributes that can discern objects from different classes. An irreducible testor is a subset of attributes such that every attribute is indispensable for satisfying the testor condition. Finding all irreducible testors has been proven to be an NP-hard problem (Skowron & Rauszer, 1992).

Feature selection is also a key aspect in expert and intelligent systems. Martínez, León, and García (2007) used testor theory to propose a new case-based approach for developing intelligent teaching-learning systems. Irreducible testors are computed in (Medina, Martínez, García, Chávez, & García, 2007) to select problem attributes for an intelligent tutoring system which has the capacity of adapting its interaction to the user's specific needs. Recently, Torres et al. (2014) used testors for determining risk factors associated with transfusion related to acute lung injuries. Another approach to feature selection based on rough set reducts has been found to have a close relationship with irreducible testors (Lazo-Cortés, Martínez-Trinidad, Carrasco-Ochoa, & Sánchez-Díaz, 2015). Therefore, expert systems can be benefited from rough set methods, especially for feature selection by means of reducts, as shown by Yahia, Mahmod, Sulaiman, and Ahmad (2000).

Recently, there is an increasing popularity of architectures based on Field Programmable Gate-Array (FPGA) for solving complex computational problems. Several hardware software platforms based on this technology have been reported (Compton & Hauck, 2002; Pocek, Tessier, & DeHon, 2013). In these platforms, the software component handles those tasks less suited for hardware implementation, and it is also responsible of configuring the FPGA, as well as handling communication with the hardware component. The hardware component, on the other hand, performs those operations with a high parallelism degree.

* Corresponding author. Tel.: +52 222 2663100x8225; fax: +52 222 2663152.
E-mail addresses: vladimir.rdguez@gmail.com (V. Rodríguez-Diez), fmartine@ccc.inaoep.mx (J.F. Martínez-Trinidad), ariel@inaoep.mx (J.A. Carrasco-Ochoa), mlazo@inaoep.mx (M. Lazo-Cortés), cferegrino@inaoep.mx (C. Feregrino-Uribe), rcumplido@inaoep.mx (R. Cumplido).

http://dx.doi.org/10.1016/j.eswa.2015.07.037
0957-4174/© 2015 Elsevier Ltd. All rights reserved.

In spite of advances in the theoretical aspects of computing irreducible testors (Djukova, 2005; Kudryavtsev, 2006; Martínez-Trinidad & Guzmán-Arenas, 2001), there are no hardware implementations reported aside from (Cumplido, Carrasco, & Feregrino, 2006; Rojas, Cumplido, Carrasco-Ochoa, Feregrino, & Martínez-Trinidad, 2007, 2012). In the first work, an FPGA-based brute force approach for computing testors was proposed (Cumplido et al., 2006). This first approach did not take advantage of dataset characteristics to reduce the number of candidates to be tested; thus all 2^n combinations of n attributes have to be tested. Then, in (Rojas et al., 2007) a hardware architecture of the BT algorithm (Ruiz-Shulcloper, Aguila-Feros, & Bravo-Martínez, 1985) for computing irreducible testors was implemented. This algorithm uses a candidate pruning process for avoiding many unnecessary candidate evaluation, reducing the number of verifications of the irreducible testor condition. These two previous works computed a set of testors on the FPGA device whilst irreducible condition was evaluated afterwards by the software component in the hosting PC. Thus Rojas, Cumplido, Carrasco-Ochoa, Feregrino, and Martínez-Trinidad (2012) proposed a hardware software platform for computing irreducible testors that implemented the BT algorithm, as in Rojas et al. (2007), but it also included a new module that eliminates most of the non irreducible testors before transferring them to a host software application for final filtering. One disadvantage of these approaches is the huge amount of data that must be transferred to the PC. Consequently, in Rodríguez et al. (2014) we proposed a modification to this platform in order to compute irreducible testors in the hardware component.

Several hardware accelerations have been recently reported for feature selection in rough set theory. Authors of Grze, Kopczynski, and Stepaniuk (2013), Kopczynski, Grze, and Stepaniuk (2014) presented an FPGA based platform for computing a reduct. Tiwari, Kothari, and Shah (2013) presented various algorithms for attribute reduction using concepts of rough set theory and implemented the Quick Reduct algorithm in a hardware fashion. Then, Tiwari and Kothari (2014) presented a thorough survey on hardware implementation of rough set algorithms. These hardware implementations aim to compute a single reduct. Jensen, Tuson, and Shen (2014) addressed the lack of guarantees for finding an attribute subset with minimal cardinality, which is the main drawback of these approaches.

In this paper, we present an efficient hardware software platform for computing irreducible testors based on the CT-EXT algorithm proposed in (Sánchez-Díaz & Lazo-Cortés, 2007). The main contribution of this work is the design and implementation of a hardware architecture that traverses the search space in a different order than that presented in (Rodríguez et al., 2014; Rojas et al., 2007, 2012). This new strategy evaluates less candidate subsets than previous architectures, which results in shorter runtime. In comparison to the software versions of CT-EXT (Sánchez-Díaz & Lazo-Cortés, 2007; Sánchez-Díaz, Piza-Davila, Lazo-Cortés, Mora-González, & Salinas-Luna, 2010), our proposal evaluates a candidate every clock cycle, which leads to a faster execution. The runtime gain of our new hardware software platform is demonstrated throughout experiments over synthetic datasets.

The rest of this paper is structured as follows. Section 2 introduces the CT-EXT algorithm. Section 3 describes the proposed architecture. Evaluation of the proposed platform and a discussion of the experimental results are presented in Section 4. Finally, Section 5 shows our conclusions and some directions for future work.

2. CT-EXT algorithm

CT-EXT is one of the fastest algorithms for computing irreducible testors reported in the state of the art (Piza-Davila, Sánchez-Díaz, Aguirre-Salado, & Lazo-Cortes, 2015; Sánchez-Díaz & Lazo-Cortés, 2007; Sánchez-Díaz et al., 2010). In order to describe this algorithm we introduce some definitions and notations.

Let *TM* be a training matrix with k objects described through n attributes of any type $R = \{x_1, \dots, x_n\}$ and grouped in r classes. Let *DM* be the binary pairwise comparison matrix, called dissimilarity matrix (0=similar, 1=dissimilar), obtained by means of attribute by attribute comparisons of every pair of objects from *TM* belonging to different classes. *DM* has m rows and n columns, where usually $m > > k$.

As it is known, *DM* commonly contains redundant rows, so algorithms for computing irreducible testors frequently work on the submatrix called basic matrix (*BM*); which is obtained from *DM* by eliminating redundant rows. For obtaining *BM* from *DM*, absorption laws are applied. Rows in *BM* are called basic rows.

Let T be a subset of attributes, T is a testor of *BM* if the attributes in T do not form a zero row in *BM*. It means that every row in *BM* has at least a 1 in those columns corresponding to attributes belonging to T. We say that a testor T is an irreducible testor if all the proper subsets of T are not testors.

We can interpret an irreducible testor as a subset of attributes being jointly sufficient and individually necessary to differentiate every pair of objects belonging to different classes.

During the search, CT-EXT follows the idea that an attribute contributes to a subset T (candidate to be an irreducible testor) if after adding this attribute to T, the attributes in T form less zero rows in *BM* than the amount of zero rows before adding the attribute. This idea is used for pruning the search space.

The following proposition, introduced and proved in (Sánchez-Díaz & Lazo-Cortés, 2007), constitutes the basis for the CT-EXT algorithm.

Proposition 1. *Given $T \subseteq R$ and $x_j \in R$ such that $x_j \notin T$. If x_j does not contribute to T, then $T \cup \{x_j\}$ cannot be a subset of any irreducible testor.*

Algorithm 1 shows the pseudocode of CT-EXT, a detailed explanation of this algorithm can be seen in (Sánchez-Díaz & Lazo-Cortés, 2007). The function SortBM(*BM*) sorts the basic matrix as follows. Randomly select one of the rows of *BM* with the fewest number of 1's. The selected row goes first and all columns in which it has a 1 are moved to the left.

The function Evaluate(*BM, T*) returns three values: *testor, irreducible* and *zero_rows*. *testor* is TRUE if the set T is a testor of *BM* and FALSE otherwise. *irreducible* is TRUE if the set T is an irreducible testor and FALSE otherwise. *zero_rows* is the amount of zero rows of T. The function LastOne(T) returns the position of the rightmost element in the set T.

Let us consider the basic matrix of Table 1, with $m = 3$ (rows) and $n = 5$ (attributes). After the ordering step we obtain the matrix shown in Table 2. Table 3 illustrates the application of the CT-EXT algorithm over this ordered basic matrix.

Table 1
Basic matrix for the example.

x_0	x_1	x_2	x_3	x_4
1	0	0	1	1
0	1	1	0	1
1	1	0	0	1

Table 2
Ordered basic matrix obtained from the matrix of Table 1.

x_0	x_3	x_4	x_1	x_2
1	1	1	0	0
0	0	1	1	1
1	0	1	1	0

Algorithm 1 CT-EXT algorithm.

```
Input: $BM$ - basic matrix with $m$ rows and $n$ columns.
Output: $TT$ - set of irreducible testors.
$TT \leftarrow \{\}$
$j \leftarrow 0$                          ▷ first attribute from $BM$ to be analyzed
$BM \leftarrow \mathrm{SortBM}(BM)$
while $BM[0, j] \neq 0$ do
    $T \leftarrow \{X_j\}$                 ▷ current attribute subset
    $testor, irreducible, zero\_rows \leftarrow \mathrm{Evaluate}(BM, T)$
    if $testor = TRUE$ then
        if $irreducible = TRUE$ then       ▷ $T$ is an irreducible testor
            $TT \leftarrow TT \cup T$
    else
        $i \leftarrow j + 1$
        while $i < n$ do
            $T \leftarrow T \cup \{X_i\}$
            $zero\_rows\_last \leftarrow zero\_rows$
            $testor, irreducible, zero\_rows \leftarrow \mathrm{Evaluate}(BM, T)$
            if $zero\_rows = zero\_rows\_last$ then
                $T \leftarrow T \setminus \{X_i\}$     ▷ attribute $X_i$ does not contribute
            else
                if $testor = TRUE$ then
                    if $irreducible = TRUE$ then
                        $TT \leftarrow TT \cup T$
                    $T \leftarrow T \setminus \{X_i\}$
                    $zero\_rows \leftarrow zero\_rows\_last$
            if $i = n - 1$ then
                $k \leftarrow \mathrm{LastOne}(T)$
                if $k = i$ then
                    $T \leftarrow T \setminus \{X_k\}$
                    $k \leftarrow \mathrm{LastOne}(T)$
                if $k \neq j$ then
                    $T \leftarrow T \setminus \{X_k\}$
                    $testor, irreducible, zero\_rows$
                    $\leftarrow \mathrm{Evaluate}(BM, T)$
                    $i \leftarrow k + 1$
                else
                    $i \leftarrow i + 1$
            else
                $i \leftarrow i + 1$
    $j \leftarrow j + 1$
```

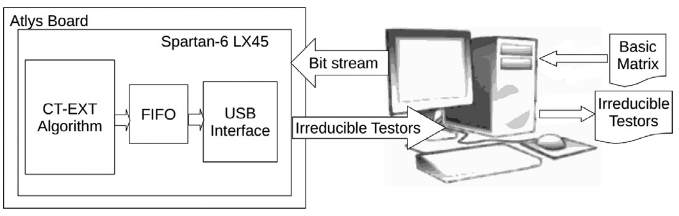

Fig. 1. Proposed hardware software platform.

3. Proposed platform

Since the problem of computing all irreducible testors has exponential complexity regarding the number of attributes; our proposed platform, as well as any other existing algorithm, has exponential complexity. A common stage to all algorithms for computing irreducible testors is the verification of each candidate combination over the basic matrix. This is an intrinsic parallel operation that a hardware implementation could take advantage of. The time complexity of evaluating a candidate for the testor condition is $O(nm)$ and for the irreducible condition is $O(n^2m)$; where n is the number of attributes and m is the number of rows in the basic matrix. In the proposed hardware component, these conditions are simultaneously evaluated in a single clock cycle. Moreover, the main novelty in this work relays on the candidate generator module. This new module implements the lexicographical total order (Sánchez-Díaz & Lazo-Cortés, 2007) as in the CT-EXT algorithm. This traversing order allows our proposal to evaluate less candidates than those evaluated by other hardware architectures reported in the literature; reducing, in this way, the execution time.

The proposed platform is shown in Fig. 1. The platform comprises a host PC and an Atlys board populated with a FPGA Spartan-6 device (Digilent, 2013); which are connected through a USB cable. A custom developed software application, running in the PC, handles all the processes needed to create the bitstream file to configure the FPGA device. The custom architecture implemented in the FPGA carries out all the calculations needed to generate the testors and sends the results to the PC where the users can then analyze the results. A detailed description of all the platform components are given below.

3.1. Hardware architecture

In the hardware architecture, an attribute subset is handled as an n-tuple, using a positional representation for all the n attributes of a basic matrix BM. Given a subset T, its n-tuple representation has a 1 in the corresponding position j for each $x_j \in T$ and 0 otherwise. The

Table 3
CT-EXT algorithm example.

T	Evaluate (BM, T)	x_j	$T \cup \{x_j\}$	Evaluate $(BM, T \cup \{x_j\})$	Comments
$\{\}$	F, F, ∞	x_0	$\{x_0\}$	F, F, 1	$\{x_0\}$ is not a testor. Add a new attribute.
$\{x_0\}$	F, F, 1	x_3	$\{x_0, x_3\}$	F, F, 1	x_3 does not contribute. Eliminate x_3. Add a new attribute.
$\{x_0\}$	F, F, 1	x_4	$\{x_0, x_4\}$	T, F, 0	x_4 contributes, $\{x_0, x_4\}$ is testor but is not irreducible. Eliminate x_4. Add a new attribute.
$\{x_0\}$	F, F, 1	x_1	$\{x_0, x_1\}$	T, T, 0	x_1 contributes, $\{x_0, x_1\}$ is an irreducible testor. It is saved ($TT = \{\{x_0, x_1\}\}$). Eliminate x_1. Add a new attribute.
$\{x_0\}$	F, F, 1	x_2	$\{x_0, x_2\}$	T, T, 0	x_2 contributes, $\{x_0, x_2\}$ is an irreducible testor. It is saved ($TT = \{\{x_0, x_1\}, \{x_0, x_2\}\}$). Eliminate x_2. Add a new attribute.
$\{x_0\}$	All attributes tested. Eliminate x_0 and add a new one.				
$\{\}$	F, F, ∞	x_3	$\{x_3\}$	F, F, 2	$\{x_3\}$ is not a testor. Add a new attribute.
$\{x_3\}$	F, F, 2	x_4	$\{x_3, x_4\}$	T, F, 0	x_4 contributes, $\{x_3, x_4\}$ is testor but is not irreducible. Eliminate x_4. Add a new attribute.
$\{x_3\}$	F, F, 2	x_1	$\{x_3, x_1\}$	T, T, 0	x_1 contributes, $\{x_3, x_1\}$ is an irreducible testor. It is saved ($TT = \{\{x_0, x_1\}, \{x_0, x_2\}, \{x_3, x_1\}\}$). Eliminate x_1. Add a new attribute.
$\{x_3\}$	F, F, 2	x_2	$\{x_3, x_2\}$	F, F, 1	x_2 contributes but $\{x_3, x_2\}$ is not a testor. Add a new attribute.
$\{x_3\}$	All attributes tested. Eliminate x_3 and add a new one.				
$\{\}$	F, F, ∞	x_4	$\{x_4\}$	T, T, 0	$\{x_4\}$ is an irreducible testor. It is saved ($TT = \{\{x_0, x_1\}, \{x_0, x_2\}, \{x_3, x_1\}, \{x_4\}\}$). Eliminate x_4. Add a new attribute.
$\{\}$	F, F, ∞	$\{x_1\}$			x_1 has a 0 in first row of BM, Algorithm finishes. $TT = \{\{x_0, x_1\}, \{x_0, x_2\}, \{x_3, x_1\}, \{x_4\}\}$

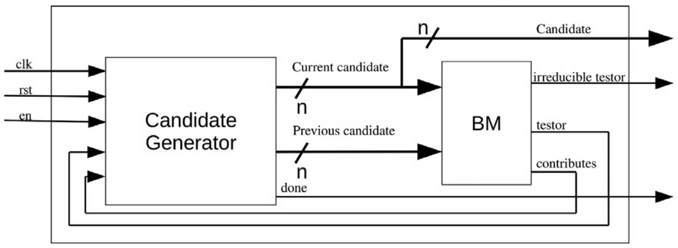

Fig. 2. CT-EXT architecture.

process of deciding whether an n-tuple is a testor of *BM* involves comparing the candidate against each one of the *BM*'s rows. For software-only implementations, this is a big disadvantage, especially for large matrices with many rows. The proposed hardware architecture exploits the parallelism inherent in the CT-EXT algorithm and evaluates whether a candidate is an irreducible testor, or not, in a single clock cycle. The hardware implementation of CT-EXT is composed of two modules, the *BM* module and the candidate generator module, as shown in Fig. 2.

The *BM* module stores the input matrix and includes all the logic needed to decide whether an n-tuple is a testor. The candidate generator module produces the candidates (n-tuples) to be evaluated by the *BM* module. In order to calculate the next candidate according to the CT-EXT algorithm, the architecture feedbacks the evaluation result of the previous candidate to the generator module; this drastically reduces the number of candidates tested and consequently the number of iterations needed by the algorithm.

The *BM* module is composed of M sub-modules named *row i*, as shown in Fig. 3. Each *row i* module contains a row (n bits) of the *BM* matrix and the logic needed to perform a testor evaluation. To decide whether an n-tuple is a testor, a bitwise AND operation is performed between the value stored in each *row i* module and the current candidate, as shown in Fig. 4. If at least one bit of the AND operation is TRUE, then the output *Testor*of that particular *row i* sub-module will be TRUE. The same operation is performed over the previous candidate. If the output *Testor* is different from the output *Contributes* for any *row i* sub-module, it means that the current candidate reduces the amount of zero rows regarding the previous candidate and then, the output *Contributes* from the *BM* module becomes TRUE. If the output *Testor* of all *row i* sub-modules is TRUE, then the output *Testor* of the *BM* module will be TRUE, which means that the candidate is a testor of *BM*.

In order to verify the irreducible condition, an *N to N Decoder* receives as input the result of the AND operation between the

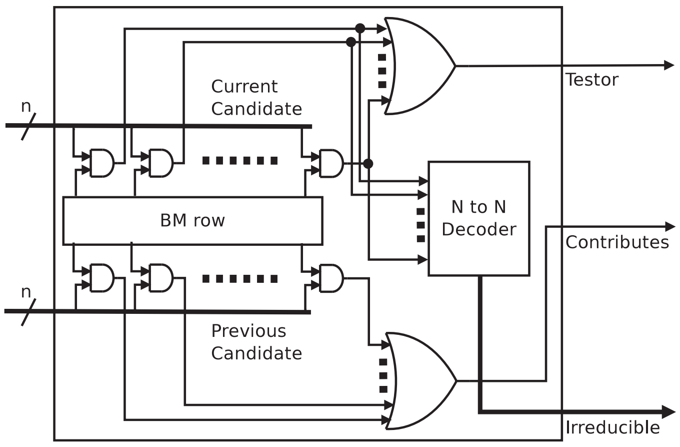

Fig. 4. *BM* row.

Table 4
An example of irreducible testor.

Cand. $\{x_0, x_1\}$					Decoder output				
x_0	x_3	x_4	x_1	x_2	x_0	x_3	x_4	x_1	x_2
1	0	0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0
Candidate =					1	0	0	1	0

current candidate and the corresponding *BM* row. The output from the *N to N Decoder* repeats the input when there is only one bit set to 1, and returns zero otherwise. For those rows with only one bit having a 1 after ANDed with the candidate, the attribute in the position of that bit is indispensable if the candidate is a testor. According to definition of irreducible testor, every attribute must be indispensable.

Taking as example the ordered basic matrix of Table 2. In Table 4 the irreducibility of $\{x_0, x_1\}$ is evaluated while the same is done for $\{x_0, x_4\}$ in Table 5. Left rows show the result of the AND operation between each row of *BM* and the candidate, while those rows in the right show the decoder output taking as input its corresponding left row. In the last row, the result of an OR operation over all above bits is shown. According to our previous explanation, the candidate $\{x_0, x_1\}$ is an irreducible testor given that the result of the OR operation is equal to the candidate itself; while candidate $\{x_0, x_4\}$ is not. This can be corroborated in Table 3.

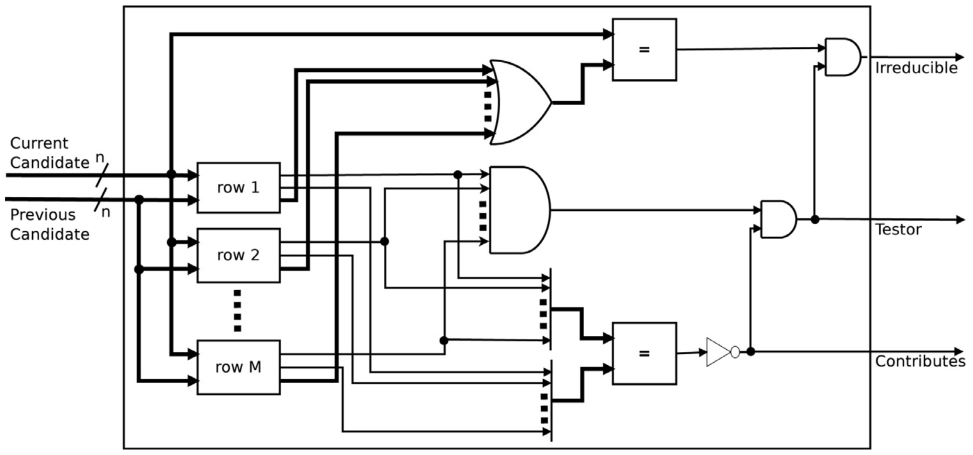

Fig. 3. *BM* module.

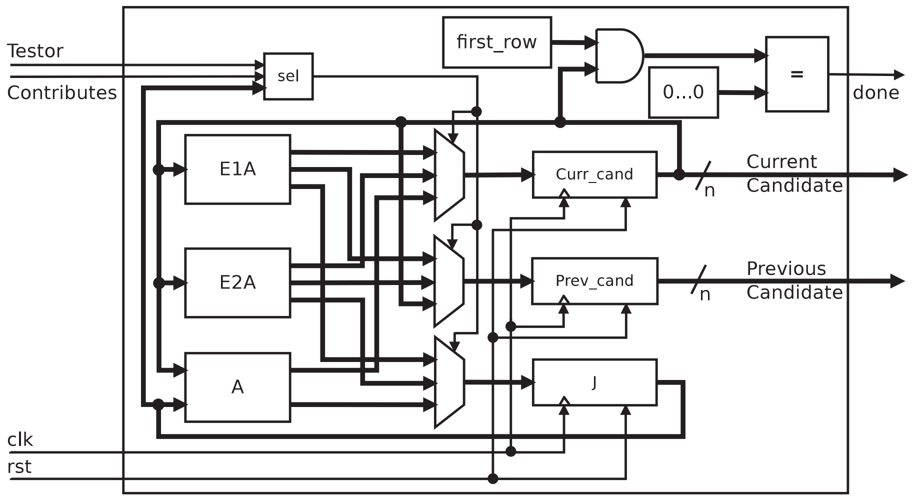

Fig. 5. Candidate generator module.

Table 5
An example of not irreducible testor.

Cand. $\{x_0, x_4\}$					Decoder output				
x_0	x_3	x_4	x_1	x_2	x_0	x_3	x_4	x_1	x_2
1	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0
1	0	1	0	0	0	0	0	0	0
Candidate $\neq$					0	0	1	0	0

Table 6
Candidate generator selector.

Priority	Condition	Registers update
1	$J = J_{\max}$ ($J_{\max}$ = max value of J)	$Curr\_cand \leftarrow$ E2A $Prev\_cand \leftarrow$ E2A $J \leftarrow$ E2A
2	Contributes = 0 or testor = 1	$Curr\_cand \leftarrow$ E1A $Prev\_cand \leftarrow$ E1A $J \leftarrow$ E1A
3	Contributes = 1 or testor = 0	$Curr\_cand \leftarrow$ A $Prev\_cand \leftarrow$ $Curr\_cand J \leftarrow$ A

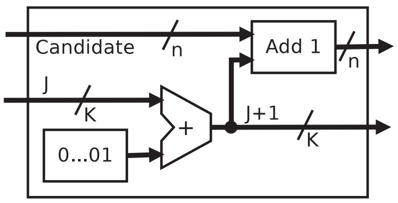

Fig. 6. Submodule A.

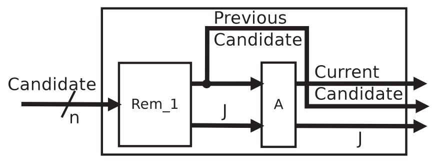

Fig. 7. Submodule E1A.

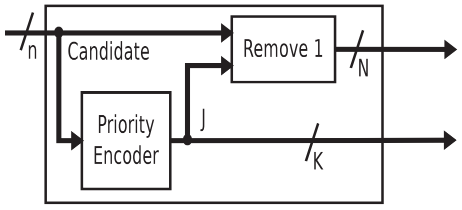

Fig. 8. Submodule *Rem*_1.

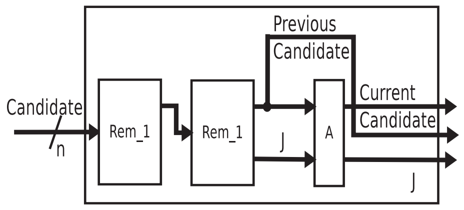

Fig. 9. Submodule E2A.

The candidate generator module (Fig. 5) uses the feedback from the *BM* module to calculate the next candidate to be evaluated. The candidate generator module consists of three registers for holding the current candidate (*Curr_cand*), the previous candidate (*Prev_cand*) and the last added attribute (*J*). These registers values are updated by the modules EA1, EA2 and A.

Depending on the combination of the input values, the outputs E1A, E2A or A are used for updating the records. Table 6 shows how the records are updated according to the values of *Testor, Contributes* and *J* inputs. This operation is computed by the module *sel* shown in Fig. 5.

The submodule *A*, shown in Fig. 6, assigns 1 to the next attribute at the right of the last bit with value 1 in the input candidate. The outputs of the submodule *A* are the new candidate and $J + 1$.

The submodule *E1A*, shown in Fig. 7, comprises the *Rem*_1 (Fig. 8) and *A* submodules. The submodule *Rem*_1 deletes the last attribute added to the input candidate. This action is performed by a priority encoder which locates the last bit with value 1 in the input candidate. *Rem*_1 outputs represent the previous candidate and the index of the deleted bit. These outputs are connected to the corresponding inputs of the submodule *A*, in order to add an attribute in the corresponding position. Finally, the outputs of *E1A* represent the new candidate to be evaluated, the previous candidate and the index where the new attribute was added to the current candidate.

Finally, the submodule *E2A* removes the last two attributes from the input candidate, and then adds the following corresponding attribute. This operation is performed by means of two *Rem*_1 submodules and an *A* submodule, as shown in Fig. 9.

In order to check if the execution of the CT-EXT algorithm has finished, the result of an AND operation between the current candidate

```
3
5
0 1 0 0 1
1 1 0 0 0
1 0 0 1 1
```

Fig. 10. Input file format.

and the first row of the basic matrix is compared to the null n-tuple $(0, \ldots, 0)$, as shown in the upper right corner of Fig. 5. If the result of this comparison is TRUE, then the output *done* is activated because any further candidate will not satisfy the testor condition over the first row of *BM* (line 6 of Algorithm 1).

3.2. The FPGA-based board

The Atlys board from Digilent (Digilent, 2013) was selected as the prototyping board. This board is a development and prototyping platform based on a Xilinx Spartan-6 LX45 FPGA, speed grade −3. The Atlys board supports device programming and simplified user-data transfer at a maximum rate of 48 MB/s, over a single USB connection.

The communication between the host PC and the FPGA uses the Digilent Synchronous Parallel Interface (DSTM) protocol (Digilent, 2010). Irreducible testors n-tuples, computed by the proposed architecture, are buffered within a FIFO in order to be split into bytes. These bytes are then buffered into a double clocked FIFO (Xilinx, 2012) to be read from the PC. This last FIFO ensures the output interface operation at 48 MHz, as required by the DSTM protocol.

3.3. Software description

The software component allows the user to provide the basic matrix in a plain text file following the format shown in Fig. 10. The software component is responsible for programming the FPGA device and communicating with the board during irreducible testor computation.

First, the basic matrix is reorganized by setting one of the rows with the minimum amount of ones as the first row and swapping columns in such a way those with a 1 in the first row appear on the left.

Using the sorted basic matrix, a VHDL file is generated and the synthesis and optimization process is started. This way, the optimization stage takes advantage of the basic matrix data to minimize the FPGA resource utilization. Then, the programming file for the FPGA device is generated.

On the running stage, the software component interacts with the hardware architecture. First, the device is programmed with the bitfile obtained from the previous stage. Then, the hardware architecture starts computing irreducible testors. The software component keeps pulling through a USB port for new irreducible testors in the output FIFO until the *done* signal is activated in the FPGA.

As a result of the sorting process, the order of attributes in the basic matrix is altered as can be seen comparing Tables 1 and 2. Consequently, the irreducible testors calculated in the FPGA must be codified according to the order of columns in the original basic matrix. This task is performed by the software component and then the results are written to the output file.

4. Evaluation and discussion

In order to show the performance of the proposed platform, it was compared against a software implementation of the CT-EXT algorithm (Sánchez-Díaz & Lazo-Cortés, 2007) and the BT hardware platform previously reported in (Rodríguez et al., 2014);[1] which is the most recent hardware implementation for computing irreducible testors reported in the literature.

Either CT-EXT or BT hardware implementations are capable of evaluating a candidate per clock cycle. If both architectures are running at the same frequency, as it will be the case in our experiments, there are two reasons for differences in running time. The first one is the time taken for reorganization of basic matrix, which is a more complex process in BT, although it can be neglected as shown in (Rojas et al., 2012). The second and the most relevant, is the amount of candidates to be evaluated.

Regarding to the software implementation, the CT-EXT hardware platform has two disadvantages. Firstly, VHDL code is generated for each *BM* data and a process of synthesis must be executed previously to executing the algorithm; while this is unnecessary in the software version of CT-EXT. Secondly, the software will be running in a PC at a frequency of 3.10 GHz while FPGA architecture will run at 50 MHz.

These disadvantages make the hardware approach useful (faster) under two conditions. First, the number of candidates to be evaluated is big enough to overcome the synthesis overhead. Second, the dimensions of the *BM* are big enough to provide a considerable speed up of the candidate evaluation process. Although the hardware architecture could be designed for a fixed maximum matrix size and receive the *BM* through the USB port, by doing this, the size of the problem which can be solved would be significantly reduced. The synthesis process comprehend an optimization of the design, taking advantage of the *BM* data distribution for the reduction of the generated hardware configuration. The number of operations for the evaluation of a single candidate, in the software approach, is proportional to the number of rows and it is directly related to the number of columns in the *BM*. Using this approach it is possible to achieve a significant reduction in the processing time, even if operating at a much lower clock frequency, by evaluating a candidate on each clock cycle.

With these points in mind and in order to show the usability of the proposed platform, three kinds of basic matrices were randomly generated. Each type containing different percentage of 1's:

1. Very-low density matrices: approximately 8%.
2. Low density matrices: approximately 33%.
3. Medium density matrices: approximately 45%.

Higher density matrices were discarded because they do not constitute a computationally expensive problem, as stated by Rojas et al. (2012). Here after, we will be referring to these three sets of matrices by its approximate density of 1's.

For our experiments, 30 basic matrices of different sizes were randomly generated. A random number generator was used to generate rows, which are filtered for the minimum and maximum number of 1's allowed. In this way the desired density was controlled. If accepted, the row is verified as basic against the saved rows. Basic rows are saved until the desired number of rows is reached.

For the hardware platforms, we measure the runtimes including the time for the following stages: *BM* input parsing and VHDL code generation, synthesis process, and irreducible testor computation (with the hardware component running at 50 MHz). The number of rows for each type of matrices is conditioned by the dimensions of the biggest matrix that may be synthesized at the desired running frequency. All experiments are performed using an Intel(R) Core(TM) i5-2400 CPU @ 3.10 GHz for software executions and an Atlys board, powered by a Spartan-6 LX45 FPGA device, for the hardware components. Figs. 11–13 show graphics of the runtime (in hours) for the three types of basic matrices.

The proposed CT-EXT hardware platform (CTH) results were taken as reference for axis limits in Figs. 11–13. Slowest executions of the CT-EXT software implementation (CTS) are not shown in order to keep clarity in the figures. The hardware platform for BT (BTH) was not able to meet the constrain of 50 MHz clock frequency for some matrices and running it at a lower frequency produces longer

[1] The source code of the three implementations and the basic matrix generator, as well as the matrices used for the experiments, can be downloaded from http://ccc.inaoep.mx/˜ariel/CTHW/.

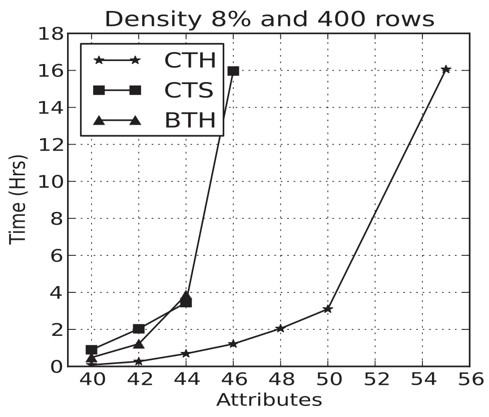

Fig. 11. Total runtime for density 8%.

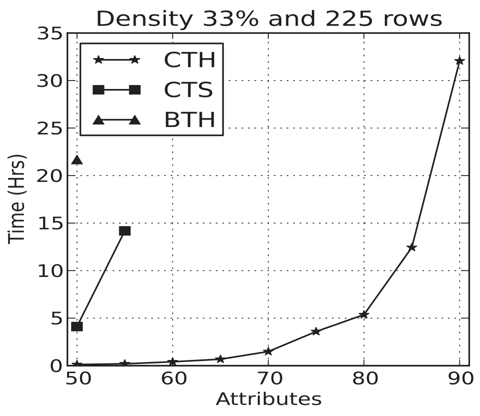

Fig. 12. Total runtime for density 33%.

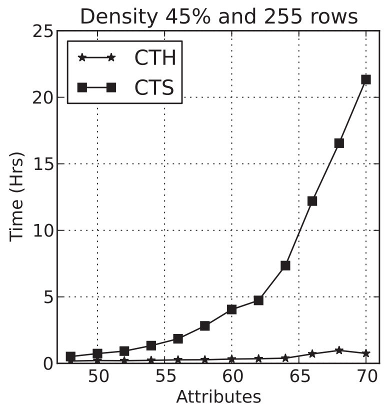

Fig. 13. Total runtime for density 45%.

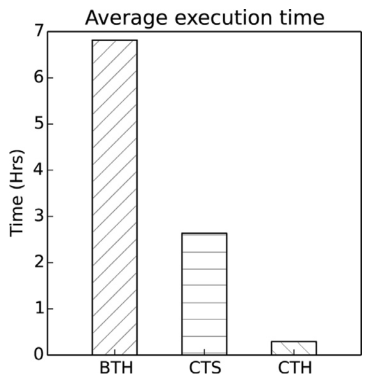

Fig. 14. Average execution time of CTH, CTS and BTH for various type of matrices.

Table 7
Processing time in seconds (broken down for each stage) for 400 × 40, 400 × 42 and 400 × 44 very-low density matrices.

Dimensions	400 × 40		400 × 42		400 × 44	
Stage	CTH	BTH	CTH	BTH	CTH	BTH
Load and file generation	0.05	0.07	0.05	0.06	0.06	0.06
Synthesis process	253	656	401	564	452	612
Algorithm execution	300	1071	970	3826	2031	13,311
Total time	554	1727	1372	4390	2484	13,924
CTS total time	3238		7320		12,420	

Table 8
Synthesis summary of resource utilization for *BM* with 8% density on an Spartan-6 LX45 FPGA device.

Dimensions	400 × 40		400 × 44	
Algorithm	BT	CT-EXT	BT	CT-EXT
Slices	1398 (20%)	983 (14%)	1554 (22%)	1209 (17%)
6-input LUTs	4010 (14%)	2806 (10%)	4475 (16%)	3004 (11%)
Flip-Flops	832 (1%)	852 (1%)	876 (1%)	938 (1%)
Max clock freq	80.44 MHz	179.58 MHz	84.56 MHz	173.24 MHz

Table 9
Synthesis summary of resource utilization for *BM* with 33% density on an Spartan-6 LX45 FPGA device.

Dimensions	225 × 50		225 × 55	
Algorithm	BT	CT-EXT	BT	CT-EXT
Slices	1381 (20%)	1554 (22%)	1455 (21%)	1562 (22%)
6-input LUTs	3769 (13%)	4315 (15%)	4135 (15%)	5026 (18%)
Flip-Flops	949 (1%)	980 (1%)	1002 (1%)	1039 (1%)
Max clock freq	87.46 MHz	155.40 MHz	85.27 MHz	156.35 MHz

execution time; which fall out of the limits of the figures. We were able to run the three platforms for 4 matrices of different types. Fig. 14 shows the average execution time of each platform for these 4 cases.

Rojas et al. (2012) stated that the time for computing irreducible testors does not only depend on the size and density of the *BM*. This assertion is illustrated by the matrices with 68 and 70 attributes respectively, in Fig. 13. Although these two matrices have a similar density, the larger matrix requires a shorter execution time.

Table 7 shows the runtime for each stage of the data flow, for 400 × 40, 400 × 42 and 400 × 44 very-low density matrices. This table shows that the synthesis time becomes less significative regarding the total time when the problem size increases. From this table, it can also be seen that the main difference between the BT and CT-EXT hardware implementations is the runtime of the irreducible testor computation process. The main reason for this difference is the number of candidates evaluated by each algorithm. This is an explanation about why the BT hardware implementation is slower than the software version of CT-EXT for the largest matrix in Table 7.

Tables 8 and 9 summarize the FPGA resource utilization on our prototyping board. The maximum operation frequency from Tables 8 and 9 shows that usually the CT-EXT implementation is potentially faster than the modified BT implementation. Resource utilization is directly related to *BM* dimensions, its density and to a lesser extent to data organization.

As it was shown in the previous section, the proposed hardware platform provides higher processing performance than the software implementation of the CT-EXT algorithm for the matrices used in our experimentation. This behavior is possible because the hardware component of the proposed platform is capable of testing whether a candidate is a testor of a *BM* in a single clock cycle, independently of the number of columns and rows, whereas the software implementation runtime will significantly increase for matrices with a large number of rows.

Experiment results show that the proposed platform beats the software implementation of the CT-EXT algorithm, with ratios of around **one order** of magnitude. However, for large enough datasets this improvement could be significantly higher, as can be inferred from Fig. 13.

5. Conclusions

In this work, we present the design and implementation of a new hardware software platform for computing all irreducible testors in a dataset. Unlike most of the existing hardware architectures for feature selection, our proposal computes all the minimal subsets of attributes that preserve the classification accuracy of the original dataset. The good performance of our hardware implementation, compared to the software approach, is feasible due to the high level of parallelism implicit in the candidate evaluation process of the CT-EXT algorithm; which can be efficiently implemented on an FPGA. This proposed architecture offers an alternative to previous hardware implementations; being faster in most of the cases, by evaluating less candidates to be irreducible testors.

Experiments also showed that the proposed platform uses fewer hardware resources and it is able to run at a higher clock frequency than previous hardware implementations. This characteristic allows processing larger matrices, since the maximum size of the problem that can be solved in a hardware architecture is conditioned by its resource utilization. Our approach enables the application of testor theory methods in larger classification and decision-making problems than it was possible before.

Even though our platform can process larger basic matrices than previous ones, its resource utilization determines the maximum size of the basic matrix that can be solved (this is, indeed, the main limitation of our proposal), for this reason, the search for new algorithms that could be efficiently implemented on an FPGA constitutes the main direction for future work. Improvements, such as testing two or more candidates at the same time, are still unexplored and would be evaluated in further studies.

References

Atlys Board Reference Manual. Digilent, Inc.
Compton, K., & Hauck, S. (2002). Reconfigurable computing: A survey of systems and software. *ACM Computing Surveys (CSUR), 34*(2), 171–210.
Cumplido, R., Carrasco, A., & Feregrino, C. (2006). On the design and implementation of a high performance configurable architecture for testor identification. In *Lectures notes on computer science: vol. 4225* (pp. 665–673).
Digilent Synchronous Parallel Interface (DSTM) Programmer's Reference Manual. Digilent, Inc.
Djukova, E. V. (2005). On the number of irreducible coverings of an integer matrix. *Computational Mathematics and Mathematical Physics, 45*, 903–908.
Grze, T., Kopczynski, M., & Stepaniuk, J. (2013). FPGA in rough set based core and reduct computation. In *Proceedings of the 8th international conference on rough sets and knowledge technology* (pp. 263–270).
Jensen, R., Tuson, A., & Shen, Q. (2014). Finding rough and fuzzy-rough set reducts with SAT. *Information Sciences, 255*, 100–120.
Kopczynski, M., Grze, T., & Stepaniuk, J. (2014). FPGA in rough-granular computing: Reduct generation. In *Proceedings of the IEEE/WIC/ACM international joint conferences on web intelligence (WI) and intelligent agent technologies (IAT) 2014* (pp. 364–370).
Kudryavtsev, V. B. (2006). Test recognition theory. *Discrete Applied Mathematics, 16*, 319–350.
LogiCORE IP FIFO Generator v9.2. Xilinx Inc.
Lazo-Cortés, M. S., Martínez-Trinidad, J. F., Carrasco-Ochoa, J. A., & Sánchez-Díaz, G. (2015). On the relation between rough set reducts and typical testors. *Information Sciences, 294*, 152–163.
Martínez, N., León, M., & García, Z. (2007). Features selection through FS-testors in case-based systems of teaching-learning. In *Proceedings of the MICAI 2007: Advances in artificial intelligence* (pp. 1206–1217).
Martínez-Trinidad, J. F., & Guzmán-Arenas, A. (2001). The logical combinatorial approach to pattern recognition an overview through selected works. *Pattern Recognition, 34*, 741–751.
Medina, D., Martínez, N., García, Z., Chávez, M., & García, M. (2007). Putting artificial intelligence techniques into a concept map to build educational tools. In *Proceedings of the nature inspired problem-solving methods in knowledge engineering* (pp. 617–627).
Piza-Davila, I., Sánchez-Díaz, G., Aguirre-Salado, C. A., & Lazo-Cortes, M. S. (2015). A parallel hill-climbing algorithm to generate a subset of irreducible testors. *Applied Intelligence, 42*(4), 622–641.
Pocek, K., Tessier, R., & DeHon, A. (2013). Birth and adolescence of reconfigurable computing: a survey of the first 20 years of field-programmable custom computing machines. In *Proceedings of the highlights of the first twenty years of the IEEE international symposium on field-programmable custom computing machines* (pp. 3–19).
Rodríguez, V., Martínez, J. F., Carrasco, J. A., Lazo, M. S., Cumplido, R., & Feregrino-Uribe, C. (2014). A hardware architecture for filtering irreducible testors. In *Proceedings of the 2014 international conference on reconfigurable computing and FPGAs (Reconfig)* (pp. 1–4).
Rojas, A., Cumplido, R., Carrasco-Ochoa, J. A., Feregrino, C., & Martínez-Trinidad, J. F. (2007). FPGA based architecture for computing testors. In *Lectures notes on computer science: vol. 4881* (pp. 188–197).
Rojas, A., Cumplido, R., Carrasco-Ochoa, J. A., Feregrino, C., & Martínez-Trinidad, J. F. (2012). Hardware-software platform for computing irreducible testors. *Expert Systems with Applications, 39*, 2203–2210.
Ruiz-Shulcloper, J. (2008). Pattern recognition with mixed and incomplete data. *Pattern Recognition and Image Analysis, 18*(4), 563–576.
Ruiz-Shulcloper, J., Aguila-Feros, L., & Bravo-Martínez, A. (1985). BT and TB algorithms for computing all irreducible testors. *Revista Ciencias Matemáticas, 2*, 11–18.
Sánchez-Díaz, G., & Lazo-Cortés, M. (2007). CT-EXT: An algorithm for computing typical testor set. In *Lecture notes in computer science: vol. 4756* (pp. 506–514).
Sánchez-Díaz, G., Piza-Davila, I., Lazo-Cortés, M., Mora-González, M., & Salinas-Luna, J. (2010). A fast implementation of the CT-EXT algorithm for the testor property identification, *6438*, 92–103.
Skowron, A., & Rauszer, C. (1992). The discernibility matrices and functions in information systems. In *Handbook of applications and advances of the rough sets theory* (pp. 331–362).
Tiwari, K., & Kothari, A. (2014). Design and implementation of rough set algorithms on FPGA. A survey. *International Journal of Advanced Research in Artificial Intelligence, 3*(9), 14–23.
Tiwari, K., Kothari, A., & Shah, R. (2013). FPGA implementation of a reduct generation algorithm based on rough set theory. *International Journal of Advanced Electrical and Electronics Engineering (IJAEEE), 2*(6), 55–61.
Torres, M., Torres, A., Cuellar, F., Torres, M., Ponce de León, E., & Pinales, F. (2014). Evolutionary computation in the identification of risk factors. Case of TRALI. *Expert Systems with Applications, 41*(3), 831–840.
Yahia, M. E., Mahmod, R., Sulaiman, N., & Ahmad, F. (2000). Rough neural expert systems. *Expert Systems with Applications, 18*(2), 87–99.