A Hardware Architecture for Filtering Irreducible Testors

Vladímir Rodríguez, José F. Martínez, Jesús A. Carrasco,
Manuel S. Lazo, René Cumplido and Claudia Feregrino Uribe
Instituto Nacional de Astrofísica, Óptica y Electrónica, México

***Abstract*—Feature selection in pattern recognition is a problem whose space complexity grows exponentially regarding the number of attributes in a dataset. There are several hardware implementations of algorithms for overcoming this complexity. These hardware architectures relay on a software component for filtering irreducible features subsets, which is a computationally complex task. In this paper, a new hardware module for the filtering process is presented. The main advantage of this new architecture is that no additional time is required for hardware execution whilst the software component is no longer needed. Experimental results show that the runtime magnitude order for software is the same as for hardware in some cases. The proposed architecture is algorithm independent and may lead to smaller hardware realizations than previous architectures.**

I. Introduction

Nowadays, the feature selection problem in pattern recognition is usually characterized by a large number of attributes. The feature selection process consists in identifying those attributes that provide relevant information for the classification process. Testor Theory is an effective way to deal with feature selection [11]. A testor is defined as a subset of attributes capable of discerning objects from different classes. An irreducible testor is a testor such that every attribute is indispensable for satisfying the testor condition. The problem of finding all irreducible testors has been proven NP-hard [10].

Reconfigurable computing using Field Programmable Gate-Array (FPGA) has become a powerful alternative to handle complex computational problems. An FPGA implementation of Testor Theory algorithms allows us to evaluate a candidate subset of attributes, over the whole basic matrix, in a single clock cycle. This kind of parallelization is not feasible in other type of technology as GPU since it would require some special handling of shared memory.

The first implementation of Testor Theory algorithms on FPGA [2] was a brute force approach for computing testors. In this first approach all the 2^n combinations of n attributes are tested in order to select only those which are testors. Afterwards, in [8] a hardware implementation of the Bottom-Top (BT) algorithm for computing irreducible testors was introduced. The BT algorithm uses a candidate pruning process for avoiding unnecessary candidate evaluation, reducing this way the number of iterations needed. These two previous platforms computed a set of testors on the FPGA device whilst irreducible condition was evaluated afterwards by the software component in a hosting PC. In [9] a hardware-software platform for computing irreducible testors implementing the BT algorithm, as in [8], was presented. This last architecture also included a new module that eliminates most of the non irreducible testors before transferring them to a host software application for final filtering.

Main disadvantages of these approaches are the huge amount of data that must be transferred to the PC and the extra cost of the final filtering stage in the software component. For this reason, in this paper, we develop a hardware module for eliminating all non irreducible testors on the FPGA device, reducing the amount of data that must be transferred to the PC. This modification incurs no delay in the FPGA runtime.

II. Basic concepts

The concept of testor for pattern recognition was introduced by [4]. They defined a testor as a subset of features that allows differentiating objects from different classes. Testors are quite useful, especially when object descriptions contain both numeric and non-numeric features, and maybe they are incomplete (mixed incomplete data) [7].

Let TM be a training matrix with k objects described through n features of any type $\{x_1, \ldots, x_n\}$ and grouped in r classes. Let DM be a Boolean dissimilarity matrix (0 = similar, 1 = dissimilar), obtained from a feature by feature comparison of every pair of objects from TM belonging to different classes. DM has m rows and n columns, where usually $m >> k$.

Let T be a subset of attributes. Testors and irreducible testors are formally defined as follows:

Definition 1: T is a testor if and only if when all features (columns) are eliminated from DM, except those from T, there is not any row of DM with only 0's. T is an irreducible testor if none of its proper subsets is a testor.

In defintion 1, if there is not any row of DM with only 0's it means that there is not a pair of objects from different classes that are similar on all the features of T, that is, a testor T allows differentiating objects from different classes. From definition 1, if T is a testor then, any superset of T is a testor too.

The number of rows in DM could be too large, therefore a strategy to reduce this matrix, without losing relevant information for computing irreducible testors, is eliminating redundant rows [6].

Definition 2: Let $f = [f_1, f_2, \ldots, f_n]$ and $f' = [f'_1, f'_2, \ldots, f'_n]$ be two rows of DM, f is a sub-row of f' if for each column $j = 1, 2, \ldots, n$; $f_j \leq f'_j$ and for at least one index, the inequality is strict.

Definition 3: A row f of DM is a basic row if no row of the matrix DM is a sub-row of f.

Definition 4: The basic matrix of DM, denoted as BM, is the sub-matrix of DM formed only by the basic rows (without repetitions).

Let $TT(DM)$ and $TT(BM)$ be the sets of all irreducible testors of DM and BM respectively, it is relatively easy to prove that $TT(DM) = TT(BM)$, see [5]; i.e., we can conclude that usually DM contains redundant information. In addition, the construction of BM from DM is a very fast process, then we work over BM.

Definition 5: Let f_i be a row of BM, f_i is a zero row of $S \subseteq R$, and we denote it as $f_i{}^0(S)$, if $\forall x_p \in S$, $f_i[p] = BM[i,p] = 0$. We denote as $\Sigma_S f^0$ the amount of zero rows of S.

Definition 6: In terms of BM, a testor $T \subseteq R$ is a feature set such that there are no zero rows of T in BM.

Definition 7: Let f_i be a row of BM, f_i is a typical row of $S \subseteq R$ regarding x_q, being $x_q \in S$, and we denote it by $f_i{}^1(S,q)$ if $f_i[q] = BM[i,q] = 1$, and $\forall x_p \in S$, $x_p \neq x_q$, $f_i[p] = BM[i,p] = 0$.

Definition 8: In terms of BM, $T \subseteq R$ is an irreducible testor if T is a testor and $\forall x_j \in T, \exists f_i{}^1(T,j)$.

This means that for each feature in an irreducible testor, there exists a row in the sub matrix of BM associated to T, having a 1 in the position corresponding to that feature, and 0 in all remaining positions (if any column of T is eliminated, at least one zero row will appear, and the testor property would not be fulfilled).

III. Proposed architecture

Hardware implementations of Testor Theory algorithms have a common module holding the basic matrix data, which is responsible for this verification. Our proposed architecture consists in modifying this module which can be ported to any algorithm implementation. We will introduce the new architecture in the top of the platform presented in [9].

In the hardware platform, a feature 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. Fig. 1 shows the original platform for testors computation as in [9]. This architecture consists of three main modules: the *Candidate Generator*, the BM and the *Dismiss Testors*.

The *Candidate Generator* module handles the sequence of candidates to be evaluated. In order to calculate the next candidate, this module receives some feedback information from the BM module. The *Candidate Generator* module is specific and its internal design is beyond the goal of this paper. The BM module handles the process of deciding whether an n-tuple is a testor of BM, comparing the candidate against each one of the BM's rows. Since our modifications take place in this module, we will expose it in detail afterwards. Finally, the *Dismiss Testors* module filters most non irreducible testors

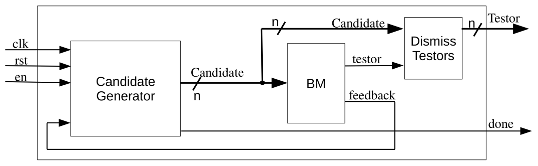

Fig. 1. Previous Architecture.

by checking inclusion of every new testor in each register of an output FIFO [9].

The modified architecture for finding irreducible testors is shown in Fig. 2. It can be seen that the *Dismiss Testors* module is no longer needed since we have a signal indicating whether the current candidate is an irreducible testor or not.

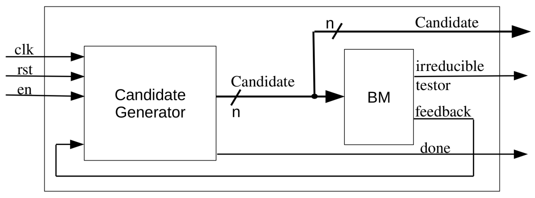

Fig. 2. Proposed Architecture.

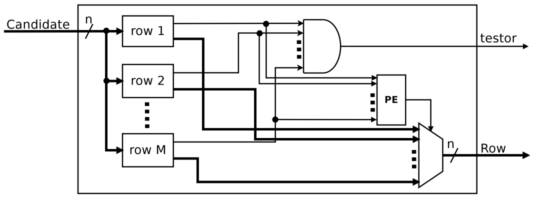

Fig. 3. Original BM module.

The original 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 basic matrix and the logic needed to perform testor evaluation. To decide whether an n-tuple is a testor, a bitwise AND operation is performed between the constant 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 is also TRUE. If the output $testor$ of all *row i* sub-modules is TRUE, then the output *testor* of the BM module is TRUE, which means that the candidate is a testor of BM. A priority encoder is used to select the n-tuple holding the data corresponding to the upper row of BM which does not satisfy the testor condition. This information is used by the *Candidate Generator* module given that the current candidate is not a testor [9].

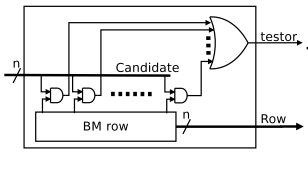

Fig. 4. Original BM row.

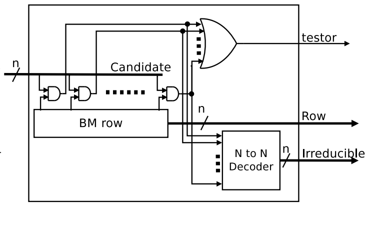

Fig. 5. Proposed BM row.

In our proposed architecture, an *N to N Decoder* is introduced into each *row i* module, as shown in Fig. 5. This new component receives as input the result of the AND operation between the 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 the null n-tuple $(0, ..., 0)$ 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 8, every attribute in a testor must be indispensable to be an irreducible testor. Fig. 6 shows the modified BM module for the evaluation of irreducible testors. Two operations are added to this module in order to verify the condition stated in definition 8. First, a bitwise OR operation is performed among the output $Irreducible$ of all *row i* submodules. The result of this operation has a 1 in the positions corresponding to each indispensable attribute in the current candidate. This value is then compared to the current candidate, and the output *irreducible testor* is TRUE given that this comparison holds equality and the output $testor$ of the BM module is TRUE.

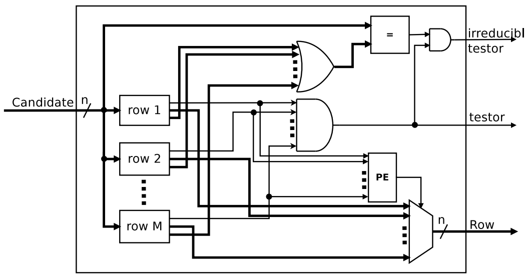

Fig. 6. Proposed BM module.

TABLE I. BASIC MATRIX

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

Lets us take for example the basic matrix shown in Table I. We are going to illustrate the operation of the proposed architecture using two testors for this basic matrix. First we will evaluate the candidate $\{x_0, x_1\}$ which is an irreducible testor. Secondly, the candidate $\{x_0, x_1, x_2\}$ will be evaluated. This last attribute set is a superset of $\{x_0, x_1\}$ and thus, it is not an irreducible testor.

TABLE II. AN EXAMPLE OF IRREDUCIBLE TESTOR

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

Left rows of Tables II and III show the result of the AND operation between each row of BM and the corresponding candidate. Rows in the right side show the decoder output taking as input its corresponding left row. In the last row, the result of an OR operation over all above n-tuples 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 the candidate $\{x_0, x_1, x_2\}$ is not.

TABLE III. AN EXAMPLE OF NOT IRREDUCIBLE TESTOR

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

IV. EVALUATION AND DISCUSSION

The proposed architecture allows sending only irreducible testors from the FPGA device to the host PC. This modification eliminates the final filtering stage in the software component of the hardware-software platform [9]. The main draw back of our proposed modification is its dependence with basic matrix dimensions which could lead to a larger hardware realization when the number of rows is much bigger than the number of columns in the basic matrix.

Transferring huge amount of data from FPGA device to the host PC could saturate the communication channel, causing the main candidate evaluation process to work intermittently. Lets take for instance a middle size problem with 64 attributes (represented by 8 bytes) evaluating candidates at 50MHz. A peak situation where several consecutive candidates are testors, leads to a data rate of 400MB/s. Even though there is an output FIFO, this data rate will eventually saturate our 48MB/s channel [3]. Transferring only irreducible testors to the host PC, reduces drastically the probability of channel saturation.

The final filtering stage in the software component needed in the original platform [9] checks every pair of testors received from the FPGA. Testors which are superset of any other testor are eliminated as they do not satisfy the irreducibility condition stated in definition 1. We can establish the lower boundary of the computational complexity for this process as $N(N-1)/2$, where N is the number of irreducible testors in the basic matrix.

Table IV shows the runtime, including the testors computation and the final filtering stage, for some basic matrices obtained from real data. For this purpose eight standard datasets from the UCI Repository of Machine Learning [1] were used. Columns in Table IV show the dataset name, the number of candidates tested by the BT algorithm, the number of irreducible testors found, the runtime for the main algorithm execution in the FPGA device and the final filtering stage in the host PC. For these runtime calculations, a frequency of 3.6GHz was used for software execution and 50MHz for the FPGA architecture.

For most of datasets shown in Table IV the processing time taken for FPGA and PC executions are of the same order of magnitude. For those basic matrices with a large number of irreducible testors, the final filtering stage could be even more expensive than the main testors computation, as is the case for the dataset labelled *german*. The total runtime for the previous architecture is the sum of the FPGA and the PC runtime.

TABLE IV. ALGORITHM EXECUTION AND IRREDUCIBLE TESTORS FILTERING STAGE RUNTIMES FOR REAL DATASETS

Dataset	Tested Candidates	Irreducible Testors	FPGA runtime (μs)	PC runtime (μs)
liverdisorder	16	9	0.32	0.04
zoo	20	7	0.40	0.02
krvskp	36	4	0.72	0.01
shuttle	38	19	0.76	0.17
tic-tac-toe	44	9	0.88	0.04
australian	330	44	6.60	0.95
lymphography	802	75	16.04	2.77
german	16921	846	338.42	357.44

Our proposed platform will require only the FPGA runtime. Although finding all irreducible testors for these datasets does not constitute a complex computational problem, they serve to show our point.

The original architecture includes a module for the elimination of most non irreducible testors, called *Dismiss Testors*. This module consists of a predefined number of registers which store previously calculated testors. New testors are compared against each previously stored testor removing any superset. At every iteration, the remaining testors are shifted to the bottom of this FIFO while the new ones are introduced on top. Outgoing testors from the *Dismiss Testors* module are transferred to the host PC for final filtering. The more registers in the FIFO, the higher the percentage of non irreducible testors eliminated. However, the number of registers impacts directly on hardware requirements. Experimental results in [9] suggest that the use of 16 registers provides a good balance between non irreducible testors elimination percentage and hardware utilization.

Tables V and VI show the hardware utilization of the original and the proposed architectures for two different basic matrices with 100 rows and 50 attributes each. Resource utilization in the original architecture is shown for 8, 16 and 32 registers in the *Dismiss Testors* module. The basic matrices have 12% and 45% density of 1's for Tables V and VI respectively.

TABLE V. FPGA RESOURCE UTILIZATION FOR SPARTAN-6 LX45 FOR N = 50 AND M = 100 (DENSITY 12%)

Resources	8 registers	16 registers	32 registers	Proposed Architecture
Slices	648	703	1043	599
Flip-Flops	1313	1715	2520	814
LUTs	1689	2040	2515	1530

TABLE VI. FPGA RESOURCE UTILIZATION FOR SPARTAN-6 LX45 FOR N = 50 AND M = 100 (DENSITY 45%)

Resources	8 registers	16 registers	32 registers	Proposed Architecture
Slices	701	814	880	895
Flip-Flops	1335	1736	2548	937
LUTs	1914	2162	2725	2439

The *Dismiss Testors* module has a fixed size in the FPGA realization once the number of registers and attributes is given. This fact is reflected in the high stability shown in the resource utilization for two basic matrices with different density of 1's, as it can be seen in Tables V and VI. The proposed architecture does, on the other hand, benefits from the logic optimization accomplished by the synthesis process. The goal of synthesis is to provide the smallest possible implementation of the design while meeting timing and power constraints. This means that the hardware utilization of the proposed module depends on the structure of data values in the basic matrix. The high reduction achieved by optimization in the low density matrix, explains that the proposed architecture is smaller than the original architecture, even with 8 registers, in Table V.

V. CONCLUSION

The hardware architecture presented in this work allows us to compute irreducible testors on the FPGA device. This characteristic implies a shorter runtime, avoiding the final filtering stage needed in previous implementations. The resource utilization of the proposed architecture depends on the basic matrix dimensions. This characteristic could be a drawback for basic matrices with a large number of rows. This new module is, however, sensitive to the logic optimization accomplished by the synthesis process. We found that optimization may lead, in some cases, to smaller hardware realizations than previous architectures.

ACKNOWLEDGMENT

This work was partly supported by the National Council of Science and Technology of Mexico (CONACyT) through the project grants CB2008-106366 and SEP-2010-158135.

REFERENCES

[1] Bache, K., Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[2] Cumplido, R., Carrasco, A. and Feregrino, C. (2006). On the Design and Implementation of a High Performance Configurable Architecture for Testor Identification. Lectures Notes in Computer Science, 4225, 665-673.

[3] Digilent Synchronous Parallel Interface (DSTM) Programmer's Reference Manual. Digilent, Inc.

[4] Dmitriev, A. N., Zhuravlev, Y. I. and Krendeliev, F. P. (1966). About Mathematical Principles of Objects and Phenomena Classification. Diskretni Analiz, 7, 3-17.

[5] Lazo-Cortés, M., Ruiz-Shulcloper, J. (1995). Determining the feature relevance for non-classically described objects and a new algorithm to compute typical fuzzy testors. Pattern Recognition Letters, 16(12), 1259-1265.

[6] Lazo-Cortés, M., Ruiz-Shulcloper, J., and Alba-Cabrera, E. (2001). An Overview of the Evolution of the Concept of Testor. Pattern Recognition, 34, 753-762.

[7] Martínez-Trinidad, J. F. and Guzmán-Arenas, A. (2001). The Logical Combinatorial Approach to Pattern Recognition an Overview through Selected Works. Pattern Recognition, 34, 741-751.

[8] Rojas, A., Cumplido, R., Carrasco-Ochoa, J. A., Feregrino, C. and Martínez-Trinidad, J. F. (2007). FPGA Based Architecture for Computing Testors. Lectures Notes in Computer Science, 4881, 188-197.

[9] Rojas, A., Cumplido, R., Carrasco-Ochoa, J. A., Feregrino, C. and Martínez-Trinidad, J. F. (2012). Hardware-software platform for computing irreducible testors. Expert Systems with Applications, 39, 2203 - 2210.

[10] Skowron, A. and Rauszer, C. (1992). The discernibility matrices and functions in information systems. Handbook of Applications and Advances of the Rough Sets Theory, 331-362.

[11] Santos, J., Carrasco, A., and Martínez, J. F. (2004). Feature selection using typical testors applied to estimation of stellar parameters. Computación y Sistemas (CyS), 8(1).