A Fast Implementation for the Typical Testor Property Identification Based on an Accumulative Binary Tuple

Guillermo Sanchez-Diaz 1, Manuel Lazo-Cortes 2, Ivan Piza-Davila 3

1 Universidad Autonoma de San Luis Potosi
Dr. Manuel Nava no 8,
San Luis Potosi, San Luis Potosi, 78290, Mexico
E-mail: guillermo.sanchez@uaslp.mx

2 Universidad de las Ciencias Informaticas
Carr. de San Antonio de los Baños Km. 2.5, Torrens,
Havana, 19370, Cuba
E-mail: mlazo@cedai.com.cu

3 Instituto Tecnologico y de Estudios Superiores de Occidente
Periferico Sur Manuel Gomez Morin 8585,
Tlaquepaque, Jalisco, 45604, Mexico
E-mail: hpiza@iteso.mx

Received 1 November 2011
Accepted 13 August 2012

Abstract
In this paper, we introduce a fast implementation of the CT EXT algorithm for testor property identifica-
tion, that is based on an accumulative binary tuple. The fast implementation of the CT EXT algorithm
(one of the fastest algorithms reported), is designed to generate all the typical testors from a training ma-
trix, requiring a reduced number of operations. Experimental results using this fast implementation and
the comparison with other state-of-the-art algorithms that generate typical testors are presented.

Keywords: feature selection, tipycal testors, pattern recognition, logical combinatorial pattern recognition.

1. Introduction

Feature selection is a significant task in supervised classification and other pattern recognition areas. This task consists of identifying those features that provide relevant information for the classification process. In Logical Combinatorial Pattern Recognition (Ref. 1,2), feature selection is solved by using Testor Theory (Ref. 3). Yu. I. Zhuravlev introduced the concept of testor to pattern recognition problems (Ref. 4). He defined a testor as a set of features that does not confuse objects descriptions belonging to different classes. This concept has been extended and generalized in several ways (Ref. 3).

The concept of testor, has also been used by V. Valev, under the name of non-reducible descriptors (NRD) (Ref. 5,6,7,8,9).

The compute of all typical testors is a procedure which have exponential complexity (Ref. 10). Different approaches have been developed, including: genetic algorithms (Ref. 11), and distributed computing (Ref. 12). Recently, implementations for
feature-selection algorithms over FPGA-based embedded systems have been developed (Ref. 13,14).

Typical testors have been widely used to evaluate the feature relevance (Ref. 15) and as support sets in classification algorithms (Ref. 16). In text mining, they have also been used for text categorization (Ref. 17) and document summarization (Ref. 18).

Unlike other methods for feature selection, typical testors have no confusion errors, due to they do not confuse objects of different classes.

But even though the application of these techniques, the running time of existing algorithms continues to be unacceptable owing to several problems which are dependent mainly, of the number of features of training matrix.

The present paper introduces a fast implementation of the CT_EXT algorithm, using a binary accumulative tuple, which simplify the search of feature combinations which fulfill the testors property.

The algorithm CT_EXT was selected for incorporating the accumulative tuple, because it is one of the fastest algorithms reported in the state of the art (Ref. 19,20).

The classic concept of testor, in which classes are assumed to be both hard and disjointed, is used. The comparison criteria used for all features are Boolean, regardless of the feature type (qualitative or quantitative). The similarity function used for comparing objects demands similarity in all features. These concepts are formalized in the following section.

Preliminary results of this algorithm were presented in (Ref. 21), but this version incorporates the following: a) explains in detail the work from V. Valev and collaborators who have made important contributions to the field of typical testors. b) the article published in (Ref. 21) introduces Typical Testors quiet briefly. This new version incorporates an entire 3-page section (Section 2), with the formalization of Typical Testors by means of: two tables, ten definitions, five remarks, one example, three matrices, three theorems, and two proofs. c) the article introduced in (Ref. 21) required barely a couple of paragraphs to introduce state-of-the-art algorithms useful to find all the typical testors. This version incorporates an entire section (Section 3) to talk about these algorithms, including its limitations.

d) this version includes a proposition and its proof, in Section 4. e) section 4 dedicates three pages to a detailed example that compares three kinds of algorithms useful to find typical testors. f) some parts of section 5 was rewritten. g) section 6 (Experiments) was entirely rewritten, and finally, h) a key idea to extend the proposed algorithm in a near future was added in Conclusions Section.

2. Typical testors

Let \( TM = \{O_1, O_2, \ldots, O_m\}, TM \subseteq U \) (where \( U \) is the universe set of objects), be a training matrix containing \( m \) objects, described in terms of \( n \) features \( R = \{x_1, x_2, \ldots, x_n\} \) and distributed into \( c \) classes \( \{K_1, K_2, \ldots, K_c\} \). Each feature \( x_i \in R \) takes values in a set \( M_i, i = 1, \ldots, n \). A comparison criterion of dissimilarity \( D_j : M_i \times M_i \rightarrow \{0, 1\} \) is associated to each \( x_i \) (0=similar, 1=dissimilar).

Each object \( O \in TM \) has associated a c-tuple of membership, which describes the belonging of the object \( O \) to the classes \( K_1, \ldots, K_c \). This c-tuple of membership is denoted by \( \alpha(O) \). Then, \( \alpha(O) = (\alpha_1(O), \ldots, \alpha_c(O)) \), where \( \alpha_i(O) = 1 \) means that \( O \in K_i \) and \( \alpha_i(O) = 0 \) means that \( O \not\in K_i \) (Ref. 22). Table 1 shows the general scheme of a training matrix.

| Objects | Features | c-tuple of belonging |
|---------|----------|----------------------|
| \( O_1 \) | \( x_1(O_1) \) | \( \alpha_1(O_1), \ldots, \alpha_c(O_1) \) |
| \( \ldots \) | \( \ldots \) | \( \ldots \) |
| \( O_p \) | \( x_1(O_p) \) | \( \alpha_1(O_p), \ldots, \alpha_c(O_p) \) |
| \( O_q \) | \( x_1(O_q) \) | \( \alpha_1(O_q), \ldots, \alpha_c(O_q) \) |
| \( \ldots \) | \( \ldots \) | \( \ldots \) |
| \( O_m \) | \( x_1(O_m) \) | \( \alpha_1(O_m), \ldots, \alpha_c(O_m) \) |

Some of the following definitions were taken from: (Ref. 3,23,22,6,7)

**Definition 1.** If some feature subset \( T \subseteq R \) does not confuse any two descriptions of objects belonging to different classes of training matrix, then \( T \) is called testor (descriptor).

**Definition 2.** If in a certain testor \( T \), after removing any attribute \( x_i \in R \), \( T \) confuses some two de-
scriptions of objects belonging to different classes of training matrix, then \( T \) is called typical testor (non-reducible descriptor (NRD)), and denoted by TT.

That means that each TT cannot be reduced any more (Ref. 23, 6). Thus each TT is of minimal length. Besides, each TT may distinguish any object description from the descriptions of the objects belonging to the remaining classes (Ref. 24, 23).

**Remark 1.** Both Ruiz-Shulcloper and Valev define typical testors (NRD) as properties of objects. However, for Valev, each object of the training matrix has associated a set of TT (NRD). On the other hand, for Ruiz-Shulcloper, all objects in each class of the training matrix have associated a set of TT (NRD) too.

In order to illustrate the above definitions, we present the follow example, which was taken from (Ref. 9)

**Example 1.** Patients are characterized as suffering from strep throat or flu depending on the presence or absence of a combination of the following symptoms: sore throat \((x_1)\), cough \((x_2)\), cold \((x_3)\), and fever \((x_4)\). Let \(K_1\) denote the class of patients suffering from strep-throat, and \(K_2\), the class of patients suffering from flu. The training matrix (shown in table 2) consist of information pertaining to seven patients, the first two in \(K_1\), and the last five in \(K_2\).

The information pertaining to each patient is represented as a row in the training matrix. A 1 in a particular column represents the presence of the corresponding symptom, and a 0 represents the absence of that symptom. Note that \(O_1, O_2 \in K_1\), and \(O_3, \ldots, O_7 \in K_2\).

In the training matrix of patients, the set of features \(\{x_1, x_2, x_4\}\) is a testor. Also, the set \(\{x_1, x_4\}\) is a typical testor of this training matrix.

We can observe that if we handle only the features of the typical testor \(\{x_1, x_4\}\), the descriptions of patients belonging to different classes are not confused on the training matrix.

This would mean that, based on the symptoms and diagnosis of the patients, only with the information about sore throat and fever is sufficient to diagnose a patient with strep throat or with flu. For this particular training matrix, the presence of sore throat and the absence of fever are sufficient to diagnose a patient with strep throat.

Let us consider the problem of obtained the TT set of a TM.

**Definition 3.** The dissimilarity matrix (denoted by DM) of the objects \(O_i \in TM\), is a Boolean matrix, where their rows are obtained by feature comparison (using dissimilarity comparison criteria D) of every pair of objects from TM belonging to different classes.

**Remark 2.** If the objects under comparison are not similar in terms of a feature, a value of 1 is assigned in the corresponding row and column of DM. If this is not the case, a value 0 is assigned.

The DM of the objects of the training matrix of patients was obtained for all the features (e.g. \(s = 1, 2, 3, 4\)), using the comparison criterion \(D_s\) shown in (2). The DM built is the follow:

\[
DM = \begin{pmatrix}
1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 \\
1 & 1 & 0 & 1
\end{pmatrix}
\]

(1)

The first row of the DM above was obtained from comparing \(O_1\) and \(O_3\). In the same way, the second row was obtained comparing \(O_1\) and \(O_4\), the third
row by the comparison of \( O_1 \) and \( O_2 \), and so on. Finally, last row was obtained by the comparison of \( O_2 \) and \( O_7 \).

\[
D_s(x_i(O_i), x_j(O_j)) = \begin{cases} 
1 & \text{if } x_i(O_i) \neq x_j(O_j) \\
0 & \text{otherwise} 
\end{cases} 
\] (2)

**Remark 3.** It is computationally faster to work with the DM instead their belonging TM. Because, in order to create the DM, the comparison between two arbitrary objects of TM is performed only once, and the DM is a Boolean matrix.

**Remark 4.** Notice that the number of rows in DM is

\[
m' = \sum_{i=1}^{c-1} \sum_{j=i+1}^{c} \text{card}(K_i) \ast \text{card}(K_j) 
\] (3)

where \( \text{card}(K_i) \) denotes the number of objects in the class \( K_i \).

Let \( p \) and \( q \) be two rows of DM.

**Definition 4.** The row \( p \) is a subrow of \( q \) if:

\[ \forall j [q_j = 0 \Rightarrow p_j = 0] \text{ and } \exists i [p_i = 0 \Rightarrow q_i = 1] \]

**Definition 5.** A row \( p \) of DM is called basic if no row in DM is a subrow of \( p \).

**Definition 6.** A matrix containing all the basic rows of DM (without repetitions) is called a basic matrix, and denoted by BM.

The BM obtained of the DM (1) is the following:

\[
BM = \begin{pmatrix} 
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 
\end{pmatrix} 
\] (4)

Only rows 7 and 8 of DM (1) are basic, and then BM (4) is composed only by these rows.

**Remark 5.** The typical testor set of a TM can be obtained by using DM or BM. A theorem showed in (Ref. 23) proves that the set of all typical testors generated from the use of DM or BM is the same. And it is shown as follows:

Let \( \tau(DM) \) be the set of all typical testors of a training matrix TM that uses its belonging dissimilarity matrix DM. And let \( \tau(BM) \) be the set of all typical testers of the same the training matrix TM that uses its corresponding basic matrix BM.

**Theorem 1.** \( \tau(DM) = \tau(BM) \)

Commonly, algorithms used for computing typical testers make use of BM instead of DM, due to the substantial reduction of rows (see remark 4)

Now, the characterization of a typical testor working with the basic matrix is presented.

**Definition 7.** Columns \( j_1, j_2, \ldots, j_d \) of an arbitrary matrix \( A = a[i,j]; i = 1, \ldots, s, j = 1, \ldots, n \) form a covering if there is not row \( p, p = 1, \ldots, s \) of the matrix \( A \) such that \( a[p,j_q] = 0, \) for \( q = 1, \ldots, d \)

Definition 7 means that a subset of columns of a matrix form a covering, if in this subset of columns, no row has only zeros.

Let matrix \( Z \) be created from a subset of columns of the basic matrix BM, generated from TM.

**Theorem 2.** (Ref. 23) If the columns \( j_1, \ldots, j_d \) of the matrix \( Z \) form a covering of BM, then the set \( T = \{x_{j_1}, \ldots, x_{j_d}\} \) is a testor of TM.

**Proof.** As the columns \( j_1, \ldots, j_d \) form a covering of BM, by definition 7, any row of the matrix \( Z \) contains only zeros. Then, every row of \( Z \) has at least a 1. By definitions 3 and 6, and remark 2 all the objects compared are not similar. Therefore, no object of TM described by the feature set \( T = \{x_{j_1}, \ldots, x_{j_d}\} \) is confused. Then, by definition 1, \( T \) is a testor.

Theorem 2 means that a testor is a subset of features \( T = \{x_{i_1}, \ldots, x_{i_s}\} \) of TM for which a whole row of zeros does not appear in the remaining submatrix of BM, after eliminating all the columns corresponding to the features in \( R \setminus T \).

**Definition 8.** Two elements \( a[i_1, j_1] \) and \( a[i_2, j_2] \) belonging to the basic matrix BM are called compatible elements, if:

(i) \( a[i_1, j_1] = a[i_2, j_2] = 1 \), for \( i_1 \neq i_2 \) and \( j_1 \neq j_2 \).

(ii) \( a[i_1, j_2] = a[i_2, j_1] = 0 \).

**Definition 9.** Elements \( a[i_1, j_1], a[i_2, j_2], \ldots, a[i_d, j_d] \) are called a sequence of compatible elements (SCE), if:

(i) for \( d = 1, a[i_1, j_1] = 1 \),
(ii) for $d > 1$, each pair of elements is a pair of compatible elements.

Definition 9 means that the matrix $Z$ formed by the rows $i_1, \ldots, i_d$ and columns $j_1, \ldots, j_d$ comprises only one unit in each row and each column.

Definition 10. The number of compatible elements $d$ of a SCE is called a rank of this SCE and it is denoted by $SCE^d$

Example 2. The matrix $Z$ formed by the rows 1 and 2, and columns 1 and 4 belonging to BM (4), which form a $SCE^2$ is the following:

$$Z = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Theorem 3. (Ref. 23) If the set $TT = \{x_{j_1}, \ldots, x_{j_p}\}$ is a testor of $TS$ (generated by columns $j_1, \ldots, j_d$ of matrix $Z$, which form a covering of BM), and rows $i_1, \ldots, i_d$ of $Z$, whose elements $a[i_1, j_1], \ldots, a[i_d, j_d]$ form a $SCE^d$, then the set $TT = \{x_{j_1}, \ldots, x_{j_p}\}$ is a typical testor of $TS$.

Proof. As the elements $a[i_1, j_1], \ldots, a[i_d, j_d]$ form a $SCE^d$, by definition 10, $SCE^d$ contains $d$ compatible elements. Then, each row $i_s$, $s = 1, \ldots, d$ of matrix $Z$, comprises a single column with only one unit, and the remaining columns contain zeros. This means that, if an arbitrary column $j_p$, $p = 1, \ldots, d$ of matrix $Z$ is removed, then a row whose columns contains only zeros is generated. But, by definition 7, columns $j_1, \ldots, j_{p-1}, j_{p+1}, \ldots, j_d$ do not form a covering of BM. Then, TT no longer meets the ownership of testor. Therefore, the testor TT cannot be reduced, and by definition 2, TT is a typical testor.

Theorem 3 means that TT is a typical testor if there is no proper subset of TT that meets the testor condition.

Columns $j_1, j_3, j_4$ of BM (4) form a covering of this matrix, and then, the feature set $T = \{x_1, x_3, x_4\}$ is a testor of TM of patients. However, $T$ is not a typical testor, because it cannot be formed in DM a $SCE^d$, with $d = 2$. But columns $j_1, j_4$ form a covering of DM, and there are elements $a[1, 4], a[2, 1]$ which form a $SCE^d$, with $d = 2$. Therefore, the feature set $T = \{x_1, x_4\}$ is a typical testor of TM of patients.

3. Algorithms for computing typical testors

In general, finding all typical testors of a training matrix is the equivalent of conducting an exhaustive search of all feature subsets. As the number of rows, columns or classes increases, the runtime required by this procedure increases too until it becomes impossible to compute. Clearly, this is a very inefficient procedure for cases in which the number of features is not too large (Ref. 25, 26).

Subsequently, several algorithms to generate the typical testor set have been proposed (Ref. 24, 27, 28, 8, 29, 19, 20). All of these algorithms can be classified into groups: internal-type and external-type algorithms.

Internal-type algorithms are based in the combinatorial construction of sequence of compatible elements (SCE) in Basic Matrix, subsequently to verify if the corresponding columns of previous combination satisfies covering property. Or these same steps in reverse order. Some examples of these algorithms are: CC (Ref. 26) and CT (Ref. 27).

These algorithms have the disadvantage of generating many repetitions of feature combinations.

External-type algorithms directly construct feature combinations to analyze from an empty subset of features. New attributes are then added to the same combination in order to verify if: a) they form a $SCE^d$, and b) the columns satisfies the covering property. Whereas internal-type algorithms start with the entire set of features, and generates several subsets as a result of removing some features subsequently, to verify the properties listed above.

Besides, external-type algorithms do not generate all possible feature combinations as the exhaustive-search algorithms. Instead, these algorithms employ the idea of jumping over possible feature combinations using different properties of testors because these feature combinations cannot be typical testors. Examples of this kind of algorithms include the following: BT and TB (Ref. 24), LEX (Ref. 29), Asaithambi (Ref. 8), CT,EXT (Ref. 19) and BR (Ref. 20).

In both kind of algorithms, the calculation of all typical testors takes exponential time (Ref. 22).

In addition with above facts about external algorithms, LEX, CT,EXT and BR incorporate into
the feature set under construction only those features that allow the construction of sequence of compatible elements (SCE) in Basic Matrix (LEX and BR), to verify subsequently if the columns of this combination satisfy the covering property. On the other hand, they incorporate only those features that can satisfy the covering property (CT_EXT), to verify subsequently if a sequence of compatible elements (SCE) exists in the combination constructed.

As alternative approaches, embedded systems based on a FPGA architecture were developed (Ref. 13,14). These architectures are able to evaluate if a feature combination fulfills the testor or typical testor property in a single clock cycle, using an exhaustive algorithm, or the external-type algorithm BT. In each step, this architecture needs the same time to process a matrix of N columns, independently of the number of rows.

The fast implementation of CT_EXT simplifies the search and construction of a feature combination which satisfies the typical testor property.

4. Definitions and propositions of fast implementation of CT_EXT algorithm

The bases of the fast implementation of the CT_EXT algorithm are presented below.

Follow the notation used in (Ref. 20), let \( V = (a_1, a_2, \ldots, a_u) \) be a binary \( u \)-tuple of elements, \( a_i \in \{0, 1\} \), \( i = 1, \ldots, u \). The column corresponding to a feature \( x_i \in BM \) is a binary \( u \)-tuple, denoted by \( V_{x_i} \), whose cardinality is the number of rows in BM.

The logical operations on binary tuples are defined as follows:

\[
(a_1, a_2, \ldots, a_u) \lor (b_1, b_2, \ldots, b_u) = (a_1 \lor b_1, a_2 \lor b_2, \ldots, a_u \lor b_u)
\]

\[
(a_1 \lor b_1, a_2 \lor b_2, \ldots, a_u \lor b_u)
\]

\[
\neg(a_1, a_2, \ldots, a_u) = (\neg a_1, \neg a_2, \ldots, \neg a_u)
\]

\[
\bigoplus(a_1, a_2, \ldots, a_u) = (a_1 \lor a_2 \lor \cdots \lor a_u)
\]

\[
(1, \ldots, 1) \text{ and } (0, \ldots, 0) \text{ represent binary tuples in which all elements are one or zero, respectively.}
\]

The notation \( L = [x_{i_1}, \ldots, x_{i_s}] \), \( x_{i_s} \in R \) is used to represent an ordered list of features. A list \( L \) that does not contain features is denoted as \([\ ]\) (empty list).

We call the length of a list \( L \), denoted as \( len(L) \), to the number of features in the list. All basic operations of the set theory (difference, intersection, subset or sublist) can be defined on ordered lists of features in a similar way.

We denote the concatenation between ordered lists of features with the + symbol.

**Definition 11.** Let \( L = [x_{i_1}, \ldots, x_{i_s}] \) be a feature list. We call accumulative mask of \( L \), denoted as \( am_L \), to the binary tuple in which the \( i^{th} \) element is 1 if the \( i^{th} \) row in BM has at least a 1 in the columns corresponding to the features of \( L \), and it is 0 otherwise.

**Definition 12.** Let \( L = [x_{i_1}, \ldots, x_{i_s}] \) be a feature list. We call contribution mask of \( L \) in feature \( x_{i_q} \in L \), denoted as \( cm_{L,x_{i_q}} \), to the binary tuple in which the \( i^{th} \) element is 1 if the \( i^{th} \) row in BM has only a 1 in the column corresponding to the feature \( x_{i_q} \), and none in the columns belonging to the remaining features, and it is 0, otherwise.

Notice that the cardinal of both \( am_L \) and \( cm_{L,x_{i_q}} \) is the number of rows in BM.

**Example 3.** Let \( L_1 = [x_2] \), \( L_2 = [x_1, x_4] \), \( L_3 = [x_1, x_4, x_5] \) and \( L_4 = [x_1, x_2, x_3, x_4] \). The accumulative and contribution masks in the features \( x_2, x_4, x_5, x_4 \) are the following: \( am_{L_1} = (0, 0, 1, 1) \), \( am_{L_2} = (1, 0, 1, 0) \), \( am_{L_3} = (1, 1, 1, 0) \), \( am_{L_4} = (1, 1, 1, 1) \); \( cm_{L_1,x_2} = (0, 0, 1, 1) \), \( cm_{L_2,x_4} = (0, 0, 1, 0) \), \( cm_{L_3,x_5} = (0, 1, 0, 0) \), \( cm_{L_4,x_4} = (0, 0, 0, 0) \).

\[
BM = \begin{pmatrix}
 x_1 & x_2 & x_3 & x_4 & x_5 \\
 1 & 0 & 0 & 0 & 0 \\
 0 & 0 & 1 & 0 & 1 \\
 0 & 1 & 0 & 1 & 1 \\
 0 & 1 & 1 & 0 & 0
\end{pmatrix}
\]

**Proposition 4.** Let \( L = [x_{i_1}, \ldots, x_{i_s}] \) be a feature list and \( x_{i_q} \notin L \). The accumulative mask of the list \( L + [x_{i_q}] \) is calculated as follows:

\[
am_{L+[x_{i_q}]} = am_L \lor V_{x_{i_q}}
\]
where \( V_{x_iq} \) denotes the column corresponding to feature \( x_{iq} \) \( \in \) BM

**Proof.** We have two cases. a) the binary u-tuple \( am_L \) in the \( i^{th} \) element has a 0, and \( V_{x_iq} \) in its same \( i^{th} \) element has a 1. To perform the operation \( am_L \lor V_{x_iq} \), this u-tuple in its \( i^{th} \) element will now have a 1. b) both the binary u-tuple \( am_L \) and \( V_{x_iq} \) in its \( i^{th} \) element has a 0. In this case, to perform the operation \( am_L \lor V_{x_iq} \), this u-tuple in its \( i^{th} \) element maintains the value of 0.

For the case in which u-tuple \( am_L \) in the \( i^{th} \) element has a 1, the value of \( V_{x_iq} \) in its \( i^{th} \) element is irrelevant, because when performing the operation \( am_L \lor V_{x_iq} \), this u-tuple in its \( i^{th} \) element retains the value of 1.

Thus, the u-tuple \( am_{L+\{x_i\}} \) meets definition 11. \( \square \)

Table 3. Value table for contribution mask

| \( B_1 \) | \( B_2 \) | \( \neg B_1 \) | \( \neg B_1 \land B_2 \) |
|---|---|---|---|
| 1  | 1  | 0  | 0  |
| 1  | 0  | 0  | 0  |
| 0  | 1  | 1  | 1  |
| 0  | 0  | 1  | 0  |

**Proposition 6.** A feature list \( L = [x_{i_1}, \cdots, x_{i_s}] \) is a testor if and only if \( am_L = (1, \cdots, 1) \)

**Proof.**

\((\Rightarrow)\) If \( L = [x_{i_1}, \cdots, x_{i_s}] \) is a testor, thus by definition 1, the feature set \( \{x_{i_1}, \cdots, x_{i_s}\} \) does not confuse any two descriptions of objects belonging to different classes of training matrix. In terms of a DM derived of their belonging TM, by remark 3, when two objects are not similar in terms of a feature, a value of 1 is assigned in the corresponding row and column of DM. This means that for a testor, there are no row has only zeros in columns \( i_1, \cdots, i_s \) of DM.

By definition 11, the accumulative mask of L applied over DM is \( am_L = (1, \cdots, 1) \).

\((\Leftarrow)\) As each element of the u-tuple \( am_L \) has a 1, by definition 11, each row of BM has at least a 1 in the columns corresponding to the features of L. This means that columns \( i_1, \cdots, i_s \) form a covering of BM.

Thus, by theorem 2, the set of features \( x_{i_1}, \cdots, x_{i_s} \) belonging to L, is a testor. \( \square \)

**Definition 13.** Let \( L = [x_{i_1}, \cdots, x_{i_s}] \) be a feature list and \( x_{iq} \in R, x_{iq} \notin L \). A row \( p \) in BM is a typical row of \( x_{iq} \) as regards of L, if it has a 1 in the column corresponding to \( x_{iq} \) and zero in all the columns corresponding to the features in \( L \setminus x_{iq} \).

**Proposition 7.** A feature list \( L = [x_{i_1}, \cdots, x_{i_s}] \) is a typical testor if and only if \( L \) is a testor and for every feature \( x_{iq} \in L \) there is at least a typical row of \( x_{iq} \) with respect to \( L \).

**Proof.**

\((\Rightarrow)\) If \( L = [x_{i_1}, \cdots, x_{i_s}] \) is a typical testor thus, by definition 2, L is a testor which can not be removed any attribute \( x_{iq}, q = 1, \cdots, s \).

Once again, in terms of a DM derived of their belonging TM, by remark 3, when two objects are not similar in terms of a feature, a value of 1 is assigned in the corresponding row and column of DM.

This means that for a typical testor, for each \( x_{iq} \in L \), exists at least one row in DM such that it has a 1 in the column corresponding to \( x_{iq} \) and zero in all the columns belonging to remaining features of L.

By definition 13, for each \( x_{iq} \in L \) there is at least
Definition 14. Let \( L = [x_i_1, \cdots, x_i_k] \) be a feature list. We say that row \( p \) in BM, is a zero-row of \( L \), denoted by \( p_0 \), if it has a 0 in all the columns corresponding to the features of \( L \).

Definition 15. Let \( L = [x_i_1, \cdots, x_i_k] \) be a feature list and \( x_i_q \in R, x_i_q \notin L \). We denote the number of zero rows of \( L \) by \( L \). We say that \( x_i_q \) contributes with \( L \) if and only if \( \sum_{L} x_i_q \) does not contribute with \( L \), and \( x_i_q \) does not contribute with \( L \).

Proposition 8. Let \( T \subseteq R \) and \( x_j \in T \). If \( x_j \) does not contribute to \( T \), then \( T \cup \{ x_j \} \) cannot generate any typical testor.

Proposition 9. Let \( T \subseteq R, Z \subseteq R, Z \neq \emptyset \). If \( T \) is a testor, then \( T \cup Z \) is a testor as well, but it is not a typical testor.

Now, for the fast implementation of the CT_EXT algorithm we have the following propositions and theorems.

Proposition 10. The feature \( x_i_q \in R, x_i_q \notin L \) contributes with \( L \) if and only if

\[
\bigoplus cm_{L+[x_i_q]} x_i_q \neq 0 \tag{12}
\]

Proof. By definition 15, a feature \( x_i_q \) contributes with \( L \), if the number of zero rows decreases considering \( L + [x_i_q] \) instead of only \( L \). By proposition 5, equation (12) has a value of 1, if and only if, at least a 1 appears in contribution mask of \( L + [x_i_q] \). And this occurs only if the number of rows of zeros decreases when the contribution mask of this list is calculated. Therefore, feature \( x_i_q \) contributes with \( L \), if and only if equation (12) takes a non-zero value.

The CT-EXT algorithm (Ref. 19) has the following theoretical bases.

Propositions 8 and 9, are re-written in terms of Propositions 10, 6 and 7, in the following way.

Theorem 11. Let \( L = [x_i_1, \cdots, x_i_k] \) be a feature list and \( x_i_q \in R, x_i_q \notin L \). If \( x_i_q \) does not contribute with \( L \), then \( L + [x_i_q] \) cannot generate any typical testor.

Proof. As \( x_i_q \) does not contribute with \( L \), this means that:

a) there is not typical row of \( x_i_q \) with respect to \( L + [x_i_q] \). Thus, \( L + [x_i_q] \) does not satisfy proposition 7;

b) it exists the \( p^q \) th element in the accumulative mask of \( L + [x_i_q] \) which has a 0. And then, the \( p^q \) th row in BM has only zeros in the columns belonging to the features of \( L + [x_i_q] \). Thus, \( L + [x_i_q] \) does not satisfy propositions 6 and 7.

Therefore, \( L + [x_i_q] \) cannot generate any typical testor.

Theorem 12. Let \( L = [x_i_1, \cdots, x_i_k] \) and \( W \subseteq R, W \neq \emptyset \) be a feature lists. If \( L \) is a testor, then \( L + W \) is a testor too, but it is not a typical testor.

Proof. As \( L \) is a testor, then \( am_L = (1, \cdots, 1) \). Then, no feature of \( W \) contributes with \( L \). Thus, by Theorem 11, \( L + W \) is not a typical testor.

And, by proposition 4, the accumulative mask of \( L + W \) is the same as that of \( L \). Then, \( L + W \) is a testor too.

Example 4. In this example, we present a comparison between the procedures of three algorithms that finds typical testors, to determine whether a subset of features is a typical-testor: a) BT (Ref. 24,30), b) CT_EXT (Ref. 19), and c) the proposed fast implementation (Ref. 21). The first one finds typical testors in a defined search space. The second, incorporates the contribution of an feature in a previous combination of attributes, to verify if it fulfills the typical-testor property. Finally, the last uses the accumulative binary tuple, to verify the same property in a feature combination.
This example takes the BM (9), and two combinations of features: \{x_1, x_2, x_3\} and \{x_1, x_4, x_5\}, checking if any of them satisfy the property of being typical-testor.

The space search with five features of algorithm BT is as follows:

\[
\begin{array}{cccccc}
  x_1 & x_2 & x_3 & x_4 & x_5 \\
 0 & 0 & 0 & 0 & 1 \\
 0 & 0 & 0 & 1 & 0 \\
 0 & 0 & 0 & 1 & 1 \\
 0 & 0 & 1 & 0 & 0 \\
 0 & 0 & 1 & 1 & 0 \\
 0 & 1 & 0 & 0 & 1 \\
 0 & 1 & 0 & 1 & 0 \\
 0 & 1 & 1 & 0 & 0 \\
 0 & 1 & 1 & 1 & 1 \\
 0 & 0 & 0 & 0 & 1 \\
 0 & 0 & 0 & 1 & 0 \\
 0 & 0 & 1 & 0 & 0 \\
 0 & 0 & 1 & 1 & 0 \\
 0 & 1 & 0 & 0 & 0 \\
 0 & 1 & 0 & 1 & 0 \\
 0 & 1 & 1 & 0 & 0 \\
 0 & 1 & 1 & 1 & 0 \\
 0 & 1 & 1 & 1 & 1 \\
\end{array}
\]

Following this procedure, it leads us to analyze the following combination of attributes: \{1, 0, 0, 1, 1\} (\{x_1, x_4, x_5\}), and to check whether columns 1, 4 and 5 form a covering of MB. For this particular case, this combination does not comply with Theorem 2, therefore, it is not a testor.

After analyzing the above combination, the BT algorithm considers the following combinations of attributes in the search space: \{1, 0, 1, 0, 0\} (\{x_1, x_3\}), \{1, 0, 1, 0, 1\} (\{x_1, x_3, x_5\}), \{1, 0, 1, 1, 0\} (\{x_1, x_3, x_4\}), \{1, 1, 0, 0, 0\} (\{x_1, x_2\}), \{1, 1, 0, 0, 1\} (\{x_1, x_2, x_5\}), \{1, 1, 0, 1, 0\} (\{x_1, x_2, x_4\}), \{1, 1, 0, 1, 1\} (\{x_1, x_2, x_4, x_5\}), reaching the following combination of features: \{1, 1, 1, 0, 0\} (\{x_1, x_2, x_3\}), and checking whether columns 1, 2 and 3 form a covering of MB. This combination of features complies with Theorem 2 and then, verifies if rows of BM forms a sequence of compatible elements (see Theorem 3). As this combination of features complies with Theorem 3, then it is a typical-testor.

The algorithm BT continues until reaching and verifying the combination \{1, 1, 1, 1, 1\} (\{x_1, x_2, x_3, x_4, x_5\}), finishing at this point.

\textbf{Algorithm CT\_EXT}

Starting at \([x_1]\), the feature list \(L = [x_1]\) is generated. Then, the number of zero rows in BM, considering \(L = [x_1]\), is counted; for this case, it results \(\Sigma_L p_0 = 3\) (see definition 14).

Following the order of search for the next feature that will contribute with \(L\), the algorithm now considers \(L + [x_2]\). Likewise, it counts the number of zero rows with \(L + [x_2]\), resulting the following: \(\Sigma_{L+[x_2]} p_0 = 1\). As \(\Sigma_{L+[x_2]} p_0 \leq \Sigma_L p_0\), feature \(x_2\) contributes with \(L\), then it is updated \(L = [x_1, x_2]\) (see definition 15).

As \(\Sigma_L p_0 \neq 0\), columns 1 and 2 do not form a covering of BM (see Theorem 2), so the algorithm continues the search of next feature that will contribute with \(L\), now considering the feature list as \(L + [x_3]\). Then, it counts the number of zero rows with \(L + [x_3]\), resulting: \(\Sigma_{L+[x_3]} p_0 = 0\). As \(\Sigma_{L+[x_3]} p_0 < \Sigma_L p_0\), feature \(x_3\) contributes with \(L\), then it is updated: \(L = [x_1, x_2, x_3]\). As \(\Sigma_L p_0 = 0\) columns 1, 2 and 3 form a covering of BM, thus \{x_1, x_2, x_3\} is a testor of BM. Since the rows of BM
considering this combination of features, form a sequence of compatible elements, then it is a typical testor.

Following this procedure, it lead us to search the next features that will contribute with L, to generate the lists: \( L = [x_1, x_2, x_3] \), \( L = [x_1, x_3] \), \( L = [x_1, x_3, x_4] \), and \( L = [x_1, x_3, x_5] \), and to reach the list: \( L = [x_1] \), considering \( x_4 \) as the next feature to analyze. Likewise, it counts the number of zero rows with \( L + [x_4] \), resulting: \( \sum_{L+[x_4]} p_0 = 2 \). As \( \sum_{L+[x_4]} p_0 < \sum_L p_0 \) feature \( x_4 \) contributes with \( L \), and it is updated \( L = [x_1, x_4] \).

As \( \sum_L p_0 \neq 0 \) then columns 1 and 4 do not form a covering of BM; so, the algorithm continues the search of the next feature that will contribute with \( L \), now considering the feature list as \( L + [x_5] \). Then, it counts the number of zero rows with \( L + [x_5] \), resulting: \( \sum_{L+[x_5]} p_0 = 1 \). As \( \sum_{L+[x_5]} p_0 < \sum_L p_0 \) feature \( x_5 \) contributes with \( L \), which is updated as \( L = [x_1, x_4, x_5] \). But \( \sum_L p_0 \neq 0 \), so columns 1, 4 and 5 do not form a covering of BM; however, there are not more features that may contribute to the list \( L \). In this case, \( L \) is not a testor.

The algorithm CT_EXT continues until reach and verifies the list \( L = [x_1, x_5] \), finishing on this point.

**Fast implementation proposed**

Like the CT_EXT algorithm, the fast implementation of CT_EXT starts at \( [x_1] \), so the feature list \( L = [x_1] \). Then, the accumulative mask of \( L \) is generated as follows (see definition 11): \( am_L = (1,0,0,0) \). But \( am_L \neq (1,1,1,1) \), therefore \( L \) is not a testor (see Proposition 6).

**Remark 6.** Illustratively in this example, the accumulative mask of \( L \) is shown as a binary tuple. However, it was implemented bitwise in the algorithm by means of 32-bit unsigned integers. For example, in the case of \( am_L = (1,0,0,0) \), the implementation in hexadecimal format is \( amL = 8000h \), which is equivalent to decimal format \( amL = 2^{31} = 2147483648 \). This is because the operations OR, AND and NEG are performed bitwise among unsigned integers variables.

Following the order of search for the next feature that will contribute with \( L \), the algorithm now considers \( L + [x_2] \). Then, the contribution mask of \( L + [x_2] \) is generated as follows: (see Proposition 5) \( cm_{L+[x_2],x_2} = (0,0,1,1) \).

As \( cm_{L+[x_2],x_2} \neq 0 \), feature \( x_2 \) contributes with \( L \), and the accumulative mask of \( L + [x_2] \) is calculated as follows: \( am_{L+[x_2]} = (1,0,1,1) \) (see Proposition 4). The feature list \( L \) is updated \( L = [x_1, x_2] \). As \( amL \neq (1,1,1,1) \) then \( L \) is not a testor.

The procedure of searching the next feature that will contribute with \( L \) is continues; for this case, this feature is \( x_3 \). Then, the contribution mask of \( L + [x_3] \) is: \( cm_{L+[x_3],x_3} = (0,1,0,0) \).

As \( cm_{L+[x_3],x_3} \neq 0 \), feature \( x_3 \) contributes with \( L \), and the accumulative mask of \( L + [x_3] \) is: \( am_{L+[x_3]} = (1,1,1,1) \). Feature list \( L \) is updated: \( L = [x_1, x_2, x_3] \). Since \( amL = (1,1,1,1) \) then \( L \) is a testor. After that, it is verified whether for each feature \( x_1, x_2, x_3 \in L \), there is at least one typical row in BM (see proposition 7). For this case, \( L = [x_1, x_2, x_3] \) is a typical testor.

Just like the algorithm CT_EXT, the fast implementation leads us to search the next features that will contribute with \( L \), producing the lists: \( L = [x_1, x_2, x_3] \), \( L = [x_1, x_3] \), \( L = [x_1, x_3, x_4] \), and \( L = [x_1, x_3, x_5] \), reaching list \( L = [x_1] \), and considering \( x_4 \) as the next feature to analyze. The accumulative mask of \( L \) is generated \( amL = (1,0,0,0) \). Then, the contribution mask of \( L + [x_4] \) is generated \( cm_{L+[x_4],x_4} = (0,0,1,0) \).

The procedure of searching the next feature that will contribute with \( L \) is continues; for this case, this feature is \( x_5 \). Then, the contribution mask of \( L + [x_5] \) is: \( cm_{L+[x_5],x_5} = (0,1,0,0) \).

As \( cm_{L+[x_5],x_5} \neq 0 \), feature \( x_5 \) contributes with \( L \), and the accumulative mask of \( L + [x_5] \) is: \( am_{L+[x_5]} = (1,1,1,1) \). Feature list \( L \) is updated \( (L = [x_1, x_4, x_5]) \). Since \( amL \neq (1,1,1,1) \) then \( L \) is not a testor.

However, there are not more features that may contribute to the list \( L \). In this case, \( L \) is not a testor.

The fast implementation continues until reaching and verifying list \( L = [x_1, x_5] \), finishing at this point.

The above example shows the procedure capable to determine whether a subset of features is or not a testor. The algorithms that do not use accumulative tuples (BT and CT_EXT), always verify whether
the columns associated with the subset of features form a covering on the basic matrix. Unlike them, the fast implementation proposed performs a bit-level involve only bit-level operations (AND, OR, NOT), whereas the algorithms described before verify whether a subset of features is a testor by crossing the rows of MB.

5. The fast implementation of the CT-EXT algorithm

The algorithm CT-EXT performs a search of features subsets on the Basic Matrix obtained from the Training Matrix. It generates feature combinations that contribute to decreasing the number of objects belonging to different classes that are confused.

In general, The algorithm CT-EXT works as follows. First, it reorders the rows and columns of the Basic Matrix, because CT-EXT is an algorithm which uses the same lexicographic total order as LEX (Ref. 29) and BR (Ref. 20). The CT-EXT algorithm incrementally, generates feature combinations by reducing step by step the number of objects belonging to different classes that are confused. Subsequently, CT-EXT verifies whether the generated combinations are typical testors. As well as LEX and BR, CT-EXT rules out those feature combinations that can generate a testor which is not a typical testor, preserving only those candidates capable of generating a typical testor. If a testor is generated, all its consecutive supersets (in the lexicographic order previously introduced into the power set of features) are not analyzed. They are skipped because these feature combinations are testors, but not typical testors.

Now, the fast implementation of the algorithm CT-EXT -based on addition and comparison operations used in CT-EXT- is replaced by simple logical Boolean operations (OR, AND and NOT), to verify if a feature \( x_j \) contributes to a list \( L \) (see Definition 15 and Proposition 10, of which Theorems 11 and 12 are derivated) then it is verified if the list \( L \) satisfies the testor property.

Thus, the number of operations carried out by CT-EXT is significantly reduced in the fast implementation proposed. Therefore, the run-time of the algorithm is improved significantly. Facilitates Description of the fast implementation of algorithm CT-EXT

Input: BM (Basic Matrix)

Output: TT (set of all typical testors)

- **Step 1**: Order rows and columns in BM.- The row of BM with the minimum number of 1’s, is set as the first row of BM. The columns of BM are sorted from left to right, each having 1 at the first row and each subsequent column having 0 at the first row of BM. The order of the columns into each group (with the same value of 1 or with the same value of 0) is irrelevant.

- **Step 2**: Initialize.- Let \( TT = \emptyset \) be an initially empty set, \( T = [] \) be an initially empty list, which is the current feature combination, and \( j = 1 \) be the index of the first feature of BM to be analyzed.

- **Step 3**: Add a new feature of first row of BM.- If \( x_j \) has 1 at the first row of BM, then concatenated \([x_j]\) to \((T = T + [x_j])\) and go to step 5. Otherwise, the algorithm finishes here (no new feature combination will generate a typical testor, because all these feature combinations have a zero row).

- **Step 4**: Evaluate the new feature.- Concatenate the list \([x_j]\) that contains feature \( x_j \) to the current list \( T \) \((T = T + [x_j])\), and verify whether this new feature contributes to the current combination (Proposition 10). If \( x_j \) does not contributes with \( T \), then go to step 6.

- **Step 5**: Verify testor property.- Verify whether list \( T \) is a testor (Proposition 6). If so, verify whether if \( T \) is a typical testor (Proposition 7). If \( T \) is a typical testor, then \( T \) is added to the set \( TT \) \((TT = TT \cup \{T\})\). Otherwise, go to step 7.

- **Step 6**: Remove the last feature processed.- Remove the list containing the last feature processed \( x_j \) from \( T \) \((T = T \backslash [x_j])\). If \( x_j \) does not contribute to \( T \), then no combination containing \( T \) is verified (Theorem 11); in such a case, go to step 7. Otherwise, if list \( T \) was a testor, then no consecutive concatenated list of \( T \) is processed (Theorem 12). If \( T \) is empty, then \( j = j + 1 \), and go to step 3.
Step 7: Select a new feature to analyze. Select the next feature that was not concatenated in the current combination. If \( j < n \) then \( j = j + 1 \), and go to step 4. Otherwise, go to step 6.

6. Experiments

In order to evaluate the performance of the fast implementation of CT\_EXT algorithm using the binary accumulative structure, a comparison with four algorithms reported in the literature (BT, CT, LEX and CT\_EXT) was made. The first algorithm selected is a classical external type algorithm, which uses the last reported algorithm that incorporates several improvements in performance (Ref. 30). The second algorithm is a classical internal type algorithm (Ref. 31). The third algorithm, LEX, is reported with a very well runtime among classical algorithms (Ref. 29). Finally the algorithm CT\_EXT, which is one of the fastest algorithms, is reported too (Ref. 19).

Please notice that comparisons with BR algorithm (Ref. 20) are not shown. This is because we were not endowed with the source code of the BR implementation as we kindly requested to the authors, but only with an application which is unable to read previously-defined DM and BM; instead, it works with randomly-generated ones.

We have employed several BM with different dimensions to compare the runtimes of the algorithms. Two of them were taken from real medical diagnosis problems. In table 4, the experimental results obtained with the algorithms are shown.

In the table, the size of the basic matrix is denoted as \( \text{rows} \times \text{columns} \); AL denotes the algorithm; LX denotes the algorithm LEX; CX represents the algorithm CT\_EXT; FI denotes the fast implementation of the CT\_EXT; and finally, TT denotes the number of typical testors found by each algorithm. The experiments were conducted in a Pentium IV, with 2Ghz, and 1 Mbyte of RAM. The runtime are measured in seconds.

| AL    | 10x34 | 20x38 | 209x32 | 209x47 | 269x42 |
|-------|-------|-------|--------|--------|--------|
| BT    | 14    | 105   | >43200 | >43200 |
| CT    | 0     | 0     | 39     | 8026   | 38691  |
| LX    | 0     | 0     | 14     | 1799   | 2530   |
| CX    | 0     | 0     | 3      | 483    | 928    |
| FI    | 0     | 0     | 0      | 72     | 120    |
| TT    | 935   | 2,436 | 6,405  | 184,920| 302,066|

In addition, we evaluate the performance of the algorithm CT\_EXT and the fast implementation proposed. We handled four real databases, obtained from UCI Machine Learning Repository (Ref. 32). These databases are the following: Zoo database; Mushroom database; Chess (King-Rook vs. King-Pawn), denoted by Kr\_vs\_Kp; and Molecular Biology (Promoter Gene Sequences), denoted by Promoters. In table 5, experimental results obtained from these databases are shown.

In the case of Promoters database, several basic sub-matrices were calculated from the original basic matrix because of its large size: 2761 rows x 57 columns. The third and fourth columns contain, respectively, the runtime of algorithm CT\_EXT and the runtime of the fast implementation. For table 5, the notation used is the following: \( M_{\text{rows} \times \text{columns}} \) which denotes the data base in use and its dimensions; BM-S denotes the size (rows x columns) of the basic matrix obtained from its belonging training matrix; CX-T and FI-T represent, respectively, the runtime of algorithm CT\_EXT and the fast implementation; finally, TT denotes the number of typical testors found.

| DB-TM                  | BM-S            | CX-T | FI-T | TT       |
|------------------------|-----------------|------|------|----------|
| Zoo\_101x17            | 14x17           | 0    | 0    | 34       |
| Kr\_vs\_Kp\_35         | 120x35          | 295  | 58   | 8464     |
| Mushroom\_122x22       | 30x22           | 0    | 0    | 292      |
| Promoters\_106x57      | 100x57          | 16   | 2    | 77,467   |
| Promoters\_106x57      | 250x57          | 167  | 13   | 257,189  |
| Promoters\_106x57      | 300x57          | 940  | 47   | 726,700  |
| Promoters\_106x57      | 1000x57         | 14,979| 199  | 1,490,107|

In table 5, we can observe that the fast implementation proposed achieves important reductions in runtime: among 80% and 98% regarding the al-
algorithm CT\_EXT.

In order to have a clear idea about the behavior of the algorithm CT\_EXT and its fast implementation, we have shown: a) the runtimes (in seconds) using a 57-column Basic Matrix, varying the number of rows from 100 to 1000, in Figure 1, and b) the runtime using a 50-row Basic Matrix, varying the number of columns from 25 to 100, in Figure 2.

7. Conclusions

In this paper, a fast implementation of the algorithm CT\_EXT that facilitates the identification of all typical testors from a training matrix was suggested.

The fast implementation -developed in this work over the algorithm CT\_EXT algorithm- is feasible to be adapted to other algorithms which calculated the typical-testor set from a training matrix.

The main contribution of the fast implementation is to keep the information of each new feature to be processed, which allows to quickly verify whether a feature combination fulfills the testor property, this is due to the fact that the algorithm proposed uses simple logical operations, implemented bitwise.

Based on experimental results, we can conclude that the fast implementation proposed improves the performance of the CT\_EXT algorithm.

The fast implementation proposed, does not contemplate the inclusion of noise in BM, because a sensitivity analysis (as is presented in (Ref. 33)) must be performed, which is beyond the scope of the paper.

Finally, our work will continue in adapting the fast implementation to a Field Programmable Gate Array (FPGA) to perform the calculation of all the typical testors.

References

1. J. Martinez-Trinidad and A. Guzman-Arenas, “The Logical Combinatorial approach for pattern recognition. An overview through selected Works,” Pattern Recognition, 34, 4, 741–751 (2001).
2. J. Ruiz-Shulcloper and M. Abidi, “Logical Combinatorial Pattern Recognition: A Review”, Recent Research Developments in Pattern Recognition, S. Pandalai, Ed., Transworld Research Networks, Kerala, India, 133-176 (2002).
3. M. Lazo-Cortes and J. Ruiz-Shulcloper and E. Alba-Cabrera, “An Overview of the evolution of the concept of testor”, Pattern Recognition, 34, 4, 753–762 (2001).
4. A. Dmitriev and I. Zhuravlev and F. Krendeliev, “About mathematical principles and phenomena classification”, Diskretni Analiz, Rusia, 7, 3–15, (1966).
5. V. Valev and Y. Zhuravlev, “Integer-valued problems of transforming the training tables in k-valued code in pattern recognition problems”, Pattern Recognition, 24, 4, 283–288 (1991).
6. V. Valev and P. Radeva, “On the determining of non-reducible descriptors for multidimensional pattern recognition problems”, Pattern Recognition and Image Analysis, 3, 3, 258–265 (1993).
7. V. Valev, “Construction of Boolean classification rules and their applications in computer vision problems”, Machine Graphics and Vision, 5, 2, 5–23 (1996).
8. A. Asaithambi and V. Valev, “Construction of all non-reducible descriptors”, Pattern Recognition, 37, 9, 1817–1823 (2004).
9. V. Valev and B. Sankur, “Generalized non-reducible descriptors”, Pattern Recognition, 37, 9, 1809–1815 (2004).
10. V. Valev and A. Asaithambi, “On computational complexity of non-reducible descriptors”, Proc. of the IEEE Int. Conf. on Information Reuse and Integration, 208–211 (2003).
11. G. Sanchez-Diaz and M. Lazo-Cortes and O. Fuentes-Chavez, “Genetic algorithm for calculating typical testors of minimal cost”, Proc. of the Iberoamerican Symposium on Pattern Recognition, SIARP 99, 207–213 (1999).
12. G. Sanchez-Diaz and M. Lazo-Cortes and J. Garcia-Fernandez, “Parallel and distributed models for calculating typical testors”, Proc. of the Iberoamerican workshop on Pattern Recognition, 135–140 (1997).
13. R. Cumplido and J. Carrasco-Ochoa and C. Feregrino, “On the design and implementation of a high performance configurable architecture for testor identification”, Proc. XI Iberoamerican Conference on Pattern Recognition, 665–673 (2006).
14. A. Rojas and R. Cumplido and J. Carrasco-Ochoa and C. Feregrino and J. Martinez-Trinidad, “FPGA-based architecture for computing testors”, Proc. XII Iberoamerican Conference on Pattern Recognition, 188–197 (2007).
15. M. Ortiz-Posadas and M. Martinez-Trinidad and J. Ruiz-Shulcloper, “A new approach to differential diagnosis of diseases”, International Journal of Biomedical Computing, 40, 3, 179–185 (2001).
16. L. De la Vega-Doria and A. Carrasco-Ochoa and J. Ruiz-Shulcloper, “Fuzzy KORA-W algorithm”, Proc. of the 6th European Conf. on Intelligent Techniques and Soft Computer, Germany, 1190–1194 (1998).
17. A. Pons-Porrata and R. Gil-Garcia and R. Berlanga-Llavori, “Using Typical Testors for Feature Selection in Text Categorization” Proc. XII Iberoamerican Conference on Pattern Recognition, 643–652 (2007).
18. A. Pons-Porrata and J. Ruiz-Shulcloper and R. Berlanga-Llavori, “A Method for the Automatic Summarization of Topic-Based Clusters of Documents”, Proc. VIII Iberoamerican Conference on Pattern Recognition, 596–603 (2003).
19. G. Sanchez-Diaz and M. Lazo-Cortes, “CT_EXT: an external escale algorithm for generated typical testers”, Proc. XII Iberoamerican Conference on Pattern Recognition, 506–514 (2007).
20. A. Lias-Rodriguez and A. Pons-Porrata, “BR: A new method for computing all typical testors”, Proc. XIV Iberoamerican Conference on Pattern Recognition, 443–440 (2009).
21. G. Sanchez-Diaz and I. Piza-Davila and M. Lazo-Cortes and M. Mora-Gonzalez and J. Salinas-Luna, “A fast implementation of the CT_EXT algorithm for the testor property identification”, Proc. Mexican International Conference on Artificial Intelligence, 92–103 (2010).
22. J. Ruiz-Shulcloper, “Pattern Recognition with Mixed and Incomplete Data”, Pattern Recognition and Image Analysis, 18, 4, 563–576 (2008).
23. J. Ruiz-Shulcloper and A. Guzman-Arenas and J. Martinez-Trinidad, “Logical combinatorial pattern recognition approach”, Centro de Investigacion en Computacion, IPN, Mexico City, Mexico (1999).
24. J. Ruiz Shulcloper and M. Bravo and F. Aguila, “Algorithms BT and TB for calculating all typical tests”, Revista Ciencias Matematicas, Cuba, 6, 2, 11–18 (1982).
25. J. Ruiz Shulcloper and R. Pico, “Mathematical modeling of discriminate anomalies AGE perspective to phosphoric rocks of sedimentary genesis”, Revista Ciencias Matematicas, Cuba, 13, 2, 159–171 (1992).
26. E. Cheremesina and J. Ruiz Shulcloper, “Methodological questions of the application of the mathematical models of pattern recognition to soft formalized knowledge zones”, Revista Ciencias Matematicas, Cuba, 13, 2, 93–108 (1992).
27. A. Bravo, “Algorithm CT for calculating the typical testors of k-valued matrix”, Revista Ciencias Matematicas, Cuba, 4, 2, 123–144 (1983).
28. L. Aguila-Feroz and J. Ruiz-Shulcloper, “Algorithm CC for the elaboration of k-valued information on pattern recognition problems”, Revista Ciencias Matematicas, Cuba, 5, 3, 89–101 (1984).
29. Y. Santiesteban Alganza and A. Pons Porrata, “LEX: A new algorithm for calculating typical testors”, Revista Ciencias Matematicas, Cuba, 21, 1, 85–95 (2003).
30. G. Sanchez-Diaz and M. Lazo-Cortes, “Modifications to BT algorithm for improving its run time execution”, Revista Ciencias Matematicas, Cuba, 20, 2, 129–136 (2002).
31. G. Sanchez-Diaz, “Developing and implementing efficient algorithms (batch and parallel) for calculating typical testors of a basic matrix”, Master Thesis, Autonomous University of Puebla, Puebla, Mexico (1997).
32. UCI-Machine-Learning-Repository, “Website of the UCI Machine Learning Repository, University of California”, http://archive.ics.uci.edu/ml (2008).
33. J. Carrasco-Ochoa, “Sensitivity Analysis in Logical-
A Fast Implementation for Typical Testor Property Identification

Combinatorial Pattern Recognition”, *Revista Computacion y Sistemas, Mexico*, 6, 1, 62–66 (2002).