Pattern Recognition Letters 80 (2016) 46–51

Contents lists available at ScienceDirect

Pattern Recognition Letters

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

Generating synthetic test matrices as a benchmark for the computational behavior of typical testor-finding algorithms☆

Eduardo Alba-Cabrera[a], Salvador Godoy-Calderon[b,*], Julio Ibarra-Fiallo[a]

[a] Colegio de Ciencias e Ingenierías, Universidad San Francisco de Quito (USFQ), Diego de Robles y Vía Interoceánica, Quito, Ecuador
[b] Centro de Investigación en Computación (CIC), Instituto Politécnico Nacional (IPN), Av. Juan de Dios Bátiz, Esq. Miguel Othón de Mendizábal. Col. Nueva Industrial Vallejo, Ciudad de México, México

ARTICLE INFO

Article history:
Received 4 September 2015
Available online 7 May 2016

Keywords:
Feature selection
Testor theory
Typical testor algorithms

ABSTRACT

Each typical testor-finding algorithm has a specific sensibility towards the number of rows, columns or typical testors within its input matrix. In this research a theoretical framework and a practical strategy for designing test matrices for typical testor-finding algorithms is presented. The core of the theoretical framework consists on a set of operators that allow the generation of basic matrices with controlled dimensions and for which the total number of typical testors is known in advance. After presenting the required theoretical foundation, and the logic for measuring a testor-finding algorithm's computational behavior, the proposed strategy is used to assess the behavior of three well-known algorithms: *BT*, *LEX*, and *FastCTExt*. Unexpected behaviors, observed during the test experiments, are analyzed and discussed, revealing previously unknown characterizations of the tested algorithms that neither a complexity analysis, nor a random experimentation protocol could have revealed beforehand.

1. Introduction

When dealing with supervised classification problems in pattern recognition, two common tasks are: (1) the determination of the informational relevance for each feature used to describe the objects under study, and (2) the selection of class representative objects from a supervision sample. Testor Theory [6], provides a solid framework under which both tasks can be tackled, positioning itself as a source of some of the most useful feature selection techniques. Typical testors play an important role when dealing with feature selection tasks [5,18] and have been used in solving practical problems like diagnosis of diseases [11], text categorization [12], document summarization [13] and document clustering [8].

The problem of finding the set of all typical testors in a basic matrix is an old problem that has had an important development during the last ten years. To support this statement, consider the number of papers presenting new algorithms related to this problem, for example [3,4,9,10,17]. All those algorithms are generally referred to as Typical Testor-finding Algoritms (*TTA*).

There are two classes of *TTA*: deterministic and meta-heuristic. Deterministic algorithms guarantee that they will find all typical testors in a given problem at the expense of an exponential time complexity. On the other hand, meta-heuristic algorithms have no guarantee to find all the typical testors in a given problem, but they are feasible to be used on extremely large search spaces where the time complexity of deterministic algorithms is simply unacceptable [16].

The complexity of deterministic *TTA* has not been sufficiently studied. This lack of sufficient study can be regarded as the cause of why most published works introducing new *TTA* fail, in the opinion of the authors of this paper, to properly justify their selection of basic matrices for comparative performance experimentation between different algorithms. On one hand, since the number of matrices selected for experimentation is considerably low, the obtained results lack statistical significance. On the other hand, by not using a specific strategy for comparatively testing algorithms, the characteristic behavior of each algorithm in the presence of certain stereotypical phenomena is not captured.

Fortunately a convenient strategy for selecting matrices for algorithm testing is certainly viable. In [1], a feasible strategy for generating test matrices was sketched for the first time, and in [2] it was used to benchmark some *TTA*. In the generated test matrices the set of typical testors can be determined in advance. This property allows the assessment of the computational behavior for the implementation of any deterministic *TTA*, as well as the validation of the answer completeness of any meta-heuristic *TTA*. Since both, the amount of typical testors and their length can be

☆ This paper has been recommended for acceptance by Eckart Michaelsen.

* Corresponding author. Tel.: +52 55 5729 6000x56553; fax: +52 5556681250.
E-mail address: sgodoyc@cic.ipn.mx (S. Godoy-Calderon).

http://dx.doi.org/10.1016/j.patrec.2016.04.020
0167-8655/

preset, generated test matrices can be targeted for studying some specific computational behavior by varying only one parameter at a time. For example, we can consider the exponential increase in the number of matrix rows with only a linear increase in the number of typical testors, or a linear increase in the number of matrix columns, resulting in a polynomial growth of the number of typical testors.

In this paper, we worked along two main directions. First, we significantly extended the previously presented theoretical framework to allow for the generation of a whole new set of test matrices that is more flexible and versatile. Second, we show how those matrices can be used to study the behavior of a *TTA* in the presence of specific phenomena. We also selected three previously published *TTA*, tested them against carefully selected test matrices, and discussed the obtained results.

The rest of this paper is structured as follows. In Section 2, the theoretical background for the generation of test matrices is set. Section 3, presents the *TTA* selected for experimentation, as well as some of their known properties. The complete set of experiments with all three *TTA* is presented in Section 4. Finally, we draw some important conclusions for new algorithm developers.

2. Theoretical background

Several, if not all of the research works in Testor Theory, handle a matrix that holds the information about the comparison of objects belonging to different classes within a certain supervision sample. That matrix is called a comparison matrix. Since using boolean comparison functions for constructing this matrix is a common practice, the result is a Boolean matrix. The comparison matrix is known as a difference matrix (*DM*) when each entry 0 means that there is a pair of objects, within the supervision sample, with the exact same value in the feature corresponding to the column of that entry, and each entry 1 means that the value, for the corresponding feature, is dissimilar in those objects.

Let $\mathcal{R}_{DM} = \{a_1, \ldots, a_m\}$ and $\mathcal{C}_{DM} = \{x_1, \ldots, x_n\}$ be the set of rows and the set of columns of a *DM*, respectively. $T \subseteq \mathcal{C}_{DM}$ is called a testor in *DM* if the submatrix $DM|_T$, obtained by eliminating from *DM* all columns not in the subset *T*, does not have any row composed exclusively by entries 0. Also, *T* is called a typical testor (irreducible testor) if no subset of *T* can be found to be also a testor in *DM*. According to the previous definition, a typical testor is a minimal set of features capable of describing all objects in the supervision sample, without causing confusion among those belonging to different classes.

A row r_p within a difference matrix is considered as sub-row of another row r_q if the following two conditions hold: each position of r_p holds a value less than or equal to the value in r_q at the same position, and there is at least one position where r_p has a value strictly less than the corresponding one in r_q. A row r_p in a difference matrix *A*, is called a *basic row* if it has no sub-rows within the same matrix.

To reduce a difference matrix, and take advantage of the last definition, for each *DM*, a basic matrix (*BM*) can be constructed which contains all and exclusively the basic rows from that *DM*. Moreover, since a *BM* has equal or less rows than its original *DM*, and it has been demonstrated that the set of all typical testors is exactly the same in both matrices, a great majority of testor-finding algorithms work on the *BM* instead of the *DM* [7,14].

Let $A = [a_{ij}]_{m\times n}$ and $B = [b_{ij}]_{m'\times n'}$ be two basic matrices. Then three crucial operators on pairs of matrices *A* and *B* ($\theta(A, B)$, $\gamma(A, B)$, and $\varphi(A, B)$) can be defined as follows:

1. The $\theta(A, B)$ operation produces a new matrix where each row in *A* is left-concatenated with each row in *B*, consequently having $m \times m'$ rows (the product of the number of rows in *A* and *B*), and also having $n + n'$ columns.
2. The $\gamma(A, B)$ operation creates a new matrix which has matrix *A* on its upper-left corner, followed by zeroes on all columns of *B*, and also has the *B* matrix on its lower-right corner, preceded by zeroes on all columns of *A*.
3. Finally, the result of a $\varphi(A, B)$ operation is a new Boolean matrix obtained by concatenating *A* and *B* if they have the same number of rows. The resulting matrix has exactly the same number of rows of *A* and *B*, but it has $n + n'$ columns (the sum of the number of columns from *A* and *B*).

Here are the formal specifications for all the above operations:

$$\theta(A,B) = \begin{bmatrix} a_{11\ldots}a_{1n} & b_{11\ldots}b_{1n'} \\ \vdots & \vdots \\ a_{11\ldots}a_{1n} & b_{m'1\ldots}b_{m'n'} \\ \vdots & \vdots \\ a_{m1\ldots}a_{mn} & b_{11\ldots}b_{1n'} \\ \vdots & \vdots \\ a_{m1\ldots}a_{mn} & b_{m'1\ldots}b_{m'n'} \end{bmatrix} \tag{1}$$

$$\gamma(A,B) = \begin{bmatrix} a_{11\ldots}a_{1n} & 0 \\ \vdots & \vdots \\ a_{m1\ldots}a_{mn} & 0 \\ \vdots & \vdots \\ 0 & b_{11\ldots}b_{1n'} \\ \vdots & \vdots \\ 0 & b_{m'1\ldots}b_{m'n'} \end{bmatrix} \tag{2}$$

and if $m = m'$ then

$$\varphi(A,B) = \begin{bmatrix} a_{11\ldots}a_{1n} & b_{11\ldots}b_{1n'} \\ & \vdots \\ a_{m1\ldots}a_{mn} & b_{m1\ldots}b_{mn'} \end{bmatrix} \tag{3}$$

The most important property of the φ, θ and γ operators is that, when applied to basic matrices, the resulting matrix is also basic, since they preserve the portion of the matrix that guarantees that rows are incomparable. Also, if their arguments are all basic matrices, then all three operators are associative. As a consequence, we will write $\varphi^N(A)$ to represent the resulting matrix of applying the φ operator over *N* matrices *A* (with $\varphi^1(A) = A$); we write $\theta^N(A)$ to represent the result of applying the θ operator *N* times consecutively (with $\theta^1(A) = A$); and we write $\gamma^N(A)$ when applying γ *N* times over matrix *A* (with $\gamma^1(A) = A$).

Now, let $\mathcal{C}_A = \{x_1, \ldots, x_n\}$ be the set of columns in basic matrix *A*, and let $x_j \in \mathcal{C}_A$. We will write $[x_j]_N$ to denote the class of all columns in $\varphi^N(A)$ exactly equal to x_j. In other words, $[x_j]_N = \{x_j, x_{j+n}, \ldots, x_{j+(N-1)n}\}$. Given $S \subseteq \mathcal{C}_A$ and $S = \{x_{j_1}, \ldots, x_{j_s}\}$, $[S]_N$ will denote the family of all subsets of columns from $\varphi^N(A)$ that can be obtained by replacing one or more columns in *S* with any other column in the same class, that is, $[S]_N = [x_{j_1}]_N \times \cdots \times [x_{j_s}]_N$. Then it is easy to verify that $|[S]_N| = N^{|S|}$.

Therefore, if *A* and *B* are basic matrices such that the sets $\Psi^*(A)$ and $\Psi^*(B)$ of all typical testors in matrices *A*, and *B* are known, then the next three propositions establish how the sets $\Psi^*(\varphi^N(A))$, $\Psi^*(\theta(A, B))$, and $\Psi^*(\gamma(A, B))$ can be analytically obtained:

Proposition 1. $\Psi^*(\varphi^N(A)) = \{[T]_N \mid T \in \Psi^*(A)\}$.

Proposition 1 states that the set of typical testors in a matrix *A*, concatenated *N* times with itself, is exactly the set of all classes of typical testors in *A*. This proposition can be proved by observing

Table 1
Matrices $\varphi^N(B)$, $\theta^N(\theta(A, B))$ and $\gamma^N(A)$.

	$\varphi^N(B)$			$\theta^N(\theta(A, B))$			$\gamma^N(A)$		
N	Rows	Columns	$\lvert\Psi^*\rvert$	Rows	Columns	$\lvert\Psi^*\rvert$	Rows	Columns	$\lvert\Psi^*\rvert$
1	4	5	4	16	10	8	4	5	4
2	4	10	18	256	20	16	8	10	4^2
3	4	15	48	4096	30	24	12	15	4^3
...	...	...	...	...	...	...	...	...	...
N	4	$5N$	$N + 2N^2 + N^3$	16^N	$10N$	$8N$	$4N$	$5N$	4^N

that, with the exception of the order of the columns, the submatrices in $\varphi^N(A)$ that form the elements of $[T]_N$ are always identical to $A|_T$. This is, because a column in T can only be replaced by another from the same class, and therefore all elements in $[T]_N$ are by definition, typical testors. Now, let us make the assumption that some typical testor $S = \{x_{j_1}, \ldots, x_{j_s}\}$ exists outside $\{[T]_N | T \in \Psi^*(A)\}$. If so, there must be at least one column of S that is not part of A. Let us replace all columns in S with the column in A within the same equivalence class. Then we would have a contradiction, because the resulting submatrix must determine a typical testor, which must be in A, and therefore S must be in $[T]_N$.

Proposition 2. $\Psi^*(\theta(A, B)) = \Psi^*(A) \cup \Psi^*(B)$.

Proposition 2 states that the set of typical testors in $\theta(A, B)$ is exactly the union of all typical testors in A, and all typical testors in B. In order to prove Proposition 2, it is enough to observe that if we identify in $\theta(A, B)$ the columns from A and from B, and we eliminate repeated rows in each set, we end up with the original matrices A and B. Therefore, the set of typical testors in A and B is preserved in $\theta(A, B)$. Also, we cannot find any testor in $\theta(A, B)$ with columns of both matrices, because if the selected columns were testors in A or in B, then we would be constructing supersets of testors, and they would not be typical (irreducible) any more. On the other hand, if the selected columns were not testors, the following reasoning applies:

Let S_A and S_B be sets of non-testor columns from A and B respectively. Let also a be the row from A with zeros in the columns of S_A, and let b be the row from B with zeros in the columns of S_B. Since the row $[ab]$ is in $\theta(A, B)$ the submatrix of $\theta(A, B)$, formed by $S_A \cup S_B$ must also have a row with zeros, and therefore it cannot be a testor.

Proposition 3. $\Psi^*(\gamma(A, B)) = \{T_A \cup T_B \mid T_A \in \Psi^*(A) \wedge T_B \in \Psi^*(B)\}$.

Finally, Proposition 3 states that the set of typical testors in $\gamma(A, B)$ is the set of all testors formed as the union of a testor in matrix A, and a testor in matrix B. To prove Proposition 3, note that, since both matrices have been added with several zero rows, then it is not possible to find a testor made up exclusively with columns from the A-side of the matrix, or exclusively with columns from the B-side of it. Any testor in $\gamma(A, B)$ must include columns from both sides of the new matrix. Now let T be a typical testor in $\gamma(A, B)$, then the A-side columns in it must have been typical testors in A, as the B-side columns in it must have been typical testors in B.

So, we have three important results, stated in the following corollaries:

Corollary 1. $|\Psi^*(\varphi^N(A))| = \sum_{T \in \Psi^*(A)} N^{|T|}$

Corollary 2. $|\Psi^*(\theta(A, B))| = |\Psi^*(A)| + |\Psi^*(B)|$

Corollary 3. $|\Psi^*(\gamma(A, B))| = |\Psi^*(A)||\Psi^*(B)|$

Example 1. Let

$$A = \begin{array}{c} \begin{matrix} x_1 & x_2 & x_3 & x_4 & x_5 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \end{array} \text{ and } B = \begin{array}{c} \begin{matrix} x_6 & x_7 & x_8 & x_9 & x_{10} \end{matrix} \\ \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix} \end{array}$$

We can verify the respective sets of all typical testors to be $\Psi^*(A) = \{\{x_1, x_2, x_3\}, \{x_1, x_2, x_5\}, \{x_1, x_3, x_4\}, \{x_1, x_3, x_5\}\}$, and $\Psi^*(B) = \{\{x_6\}, \{x_8, x_{10}\}, \{x_7, x_9, x_{10}\}, \{x_7, x_8\}\}$.

Note that A has 4 typical testors of length 3. By using Corollary 1 above, we know that $|\Psi^*(\varphi^3(A))| = 4 * 3^3 = 108$. Likewise, since B has one typical testor of length 1, two of length 2, and one of length 3, we can establish, using the same corollary, that $|\Psi^*(\varphi^3(B))| = 3^1 + 2 * 3^2 + 3^3 = 48$. We can also determine how many typical testors to expect in $\theta(A, B)$, and $\gamma(A, B)$ using Corollaries 2 and 3: $|\Psi^*(\theta(A, B))| = 8$ and $|\Psi^*(\gamma(A, B))| = 16$.

In Table 1, the effect (on the number of rows, columns and typical testors) of applying some operator combinations over matrices A and B from Example 1 is shown.

As it can be seen, the generated matrices preserve the same number of rows, while the number of columns increase linearly; but the resulting number of typical testors grows according to a cubic polynomial. If one wishes to assess the effect of an exponential growth in the number of rows while sustaining a linear behavior of both the number of columns and the number of typical testors, then one would only need to use the test matrix $\theta^N(\theta(A, B))$ with A and B being exactly those from Example 1. The resulting behavior of such test is shown in Table 1. Table 1 also shows the use of operator γ^N over the matrix A. In this case, we can see that the number of rows and columns increase linearly; but the resulting number of typical testors grows exponentially. In the next section, some deterministic *TTA* to be tested against a selection of generated test matrices will be reviewed.

3. Reviewed algorithms

Deterministic *TTA* can be classified in two sets: external and internal. External *TTA* always set an order to test the power set of all columns in the basic matrix. Following that order, each subset of columns is tested in order to determine if it is a typical testor or not. However, the test process is not an exhaustive search over the power set. Some properties of each tested subset allow the *TTA* to infer which other successive subsets, following the established order, cannot be typical testors, and therefore it is not worth to test them. The act of bypassing the test of some subsets of columns is commonly referred to as jumping. In general, the order selected to traverse the power set of columns, along with the magnitude of the jumps (e.g. the number of subsets not tested), and the specific procedure applied over a subset to test if it is a typical testor or not, determine the behavior of an external *TTA*. On the other hand, the strategy of internal *TTA* lies on selecting, in an iterative fashion, the entries in the basic matrix, and use them to construct typical testor candidates.

Internal *TTA* are not considered for experimentation in this article, due to their notoriously low performance reported on several state-of-the-art works [3,17]. However, three external *TTA* were selected for showing how the φ, θ, and γ operators can be used to assess the computational behavior of any *TTA*. These algorithms are *BT* [15], *LEX* [3], and *FastCTExt* [17].

3.1. The BT algorithm

The order in which the *BT* algorithm traverses the power set of all columns in a basic matrix is called the *natural order*. This order maps each column subset (of length m) to a binary string (also of length m), which is then interpreted as the binary representation of a natural number. Accordingly, an exhaustive traversal of any power set should begin with a string full of zeros, representing the empty set of columns associated with the natural number zero. The exhaustive traversal ends with a string full of ones, representing the full set of columns associated with the natural number $2^m - 1$.

Once the natural order is set over the power set of all columns in a basic matrix, the first element is tested to find if it is a testor or not. Each time a column subset is tested, its properties are used to determine how many of the next subsets cannot be typical testors, and therefore, can be jumped (i.e. not tested). Depending on the inherent structure in the basic matrix, the jumps performed can be of different lengths. Of course, the amount and length of the jumps performed heavily influence the computational behavior of the algorithm.

3.2. The LEX algorithm

Similar to *BT*, the *LEX* algorithm also sets an order on the power set of columns in the basic matrix. However, the *LEX* algorithm uses a different order called the *lexicographic order*. Another fundamental difference between these two algorithms is the fact that *LEX* follows its established order only to select the initial column subset for the process of constructing typical testor candidates. Each time a new column is added to a candidate, its corresponding submatrix is checked to ensure that it has no row formed exclusively by zeros (i.e. the candidate is a testor). In case it does not have such a row, then the candidate is extended by adding to it those columns that can turn it into a typical testor. Then, the modified candidate is acknowledged to be a typical testor and a jump is performed, in the same style as in the *BT* algorithm.

In order to improve the algorithm's general performance, the basic matrix is subjected to an internal column reordering process, aiming at configuring it, as similar as possible, to a lower triangular matrix. That reordering ensures that the jumps performed by the algorithm will be of the maximum possible length.

3.3. The CT_EXT and FastCTExt algorithms

CT_EXT follows the same order used by the *LEX* algorithm to test the power set of columns in the basic matrix, and it also constructs typical testor candidates. The main difference lies in the criteria used for adding a new column to the candidate. *CT_EXT* does not add columns to complete the candidate as a typical testor but only as a testor, and the columns added are known as compatible sets. Because of this, after constructing a candidate as a testor, but not necessarily typical, the algorithm must check if it is typical or not afterwards. In [17], a fast implementation of this algorithm, named *FastCTExt*, was presented. Since the reported performance of this implementation is better than the one from the original algorithm, it was selected for all the experiments reported in this research.

Table 2
Relative efficiency of the tested algorithms against the φ operator, applied to identity matrix I_5.

N	Matrix	Rows	Cols	$\lvert\Psi^*\rvert$	LEX (%)	FastCTExt (%)	BT (%)
2	$\varphi^2(I_5)$	5	10	32	13.91	12.17	10.03
4	$\varphi^4(I_5)$	5	20	1024	17.73	17.31	0.51
6	$\varphi^6(I_5)$	5	30	7776	19.72	19.54	0.01
8	$\varphi^8(I_5)$	5	40	32,768	20.87	20.77	0.0056
10	$\varphi^{10}(I_5)$	5	50	100,000	21.61	21.54	

4. Sample testing of *TTA* with generated test matrices

As an example of how the proposed operators can be used for *TTA* testing purposes, we designed specific experiments to assess the computational behavior of each algorithm when facing different phenomena. For each test, a custom-designed test matrix was created with some specific combination of the θ, γ, and φ operators, applied to matrices A and B, as well as on some identity matrices.

Since all *TTA* traverse the power set of basic matrix columns in search for typical testors, a rather fair way to compare their performances is to compare the number of tested column subsets against the previously known number of typical testors found in the problem. In order to do so, for each experiment below, we registered the number of column subsets tested by the *TTA* (labeled as HITS), and calculated the ratio of typical testors found over the total number of tested column subsets (labeled as EFFICIENCY). In the ideal case a *TTA* would test only those columns subsets which are typical testors, and would have an efficiency of 100%. A non-ideal algorithm will always test more column subsets than the number of typical testors it discovers, and so, its efficiency will always be lower than 100%. This evaluation strategy has two advantages over a more traditional complexity analysis: first, there is a theoretical guarantee that a deterministic *TTA* will always test the exact same number of column subsets when feed with the same initial basic matrix; second, in the case of heuristic algorithms that do not behave the same way on each run, the method is also applicable. All the following experiments were run on an Intel i7 processor, with 4GB in RAM. However, since the ultimate goal of this work is to show the usefulness of the proposed set of operators, and not to rigorously test each algorithm, absolute execution times as well as the hardware platform, are not relevant.

The first set of experiments was designed with successive powers of the φ operator, applied to identity matrices of size 5, denoted I_5. Each one of those operations generates a basic matrix with 5 rows, but with a linearly increasing number of columns, and with a number of typical testors equal to the selected power of φ raised to the power of 5, which is the dimension of the matrix. All three algorithms were tested against each matrix, and Table 2 summarizes the obtained results.

Just as several research works have previously reported [3,17], Table 2 seems to confirm the low performance of the *BT* algorithm. However, Table 2 also shows how dramatic its decrease in performance can be under some special circumstances. Notably, while the efficiency of the *LEX* and *FastCTExt* algorithms slightly increase as the number of columns and typical testors also increase, the *BT* algorithm behaves exactly the opposite way. Now, the fundamental premise of this work is that, without using a practical test strategy capable of generating a wide diversity of phenomena, it is not possible to identify performance bottlenecks and special cases where a different processing technique is needed. As concrete evidence of this premise, the reader should consider the next set of experiments, where the three studied algorithms are compared using simple identity matrices. Clearly, an identity matrix contains

Table 3
Relative efficiency of the tested algorithms using identity matrices of different sizes.

N	Matrix	Rows	Cols	$\|\Psi^*\|$	LEX (%)	FastCTExt (%)	BT (%)
1	I_5	5	5	1	16.67	6.25	16.67
2	I_{10}	10	10	1	9.09	0.20	9.09
3	I_{15}	15	15	1	6.25	0.01	6.25
4	I_{20}	20	20	1	4.76	0.0002	4.76
5	I_{25}	25	25	1	3.85	0.000006	3.85

Table 4
Relative efficiency of the tested algorithms against the θ operator.

N	Matrix	Rows	Cols	$\|\Psi^*\|$	LEX (%)	FastCTExt (%)	BT (%)
1	$\theta^1(A)$	4	5	4	36.36	36.36	44.44
2	$\theta^2(A)$	16	10	8	7.55	5.19	8.60
3	$\theta^3(A)$	64	15	12	1.31	0.84	1.56
4	$\theta^4(A)$	256	20	16	0.21	0.12	0.31
5	$\theta^5(A)$	1024	25	20	0.03	0.02	0.04

Table 5
Relative efficiency of the tested algorithms against the φ operator.

N	Matrix	Rows	Cols	$\|\Psi^*\|$	LEX (%)	FastCTExt (%)	BT (%)
1	$\varphi(\theta(A, B))$	16	10	8	5.23	4.30	8.51
2	$\varphi^2(\theta(A, B))$	16	20	50	1.89	0.88	0.56
3	$\varphi^3(\theta(A, B))$	16	30	156	1.00	0.26	0.04
4	$\varphi^4(\theta(A, B))$	16	40	356	0.62	0.14	0.002
5	$\varphi^5(\theta(A, B))$	16	50	680	0.42	0.06	0.001

only one typical testor, regardless of its dimension. The results of this second set of experiments are summarized in Table 3.

Surprisingly enough, the *BT* and *LEX* algorithms behave with the exact same efficiency, far beyond that of the *FastCTExt* algorithm. Such unexpected result can undoubtedly be attributed to the jumping mechanism shared by both algorithms. Although the *LEX* and the*FastCTExt* algorithms use the same order over the power set of basic matrix columns (the lexicographic order), their jumping mechanisms are very different, and so is their efficiency. The *FastCTExt* search for compatible sets severely complicates its ability to identify the only typical testor found within each experiment.

Moreover, it can be argued that the alleged low performance of the *BT* algorithm cannot be explained by the intrinsic properties of the natural order it uses. To test that hypothesis, we provide the next set of experiments, where successive powers of the θ operator, over the *A* matrix, are used to test the relative efficiency of all three algorithms. See Table 4.

From Table 4 it can be observed that when the number of rows grows exponentially, and the number of columns and testors grow linearly, the *BT* algorithm performs slightly better than the other two algorithms. However, *BT*'s sensibility to an increase in the number of columns seems to be much less than its sensitivity to an increase in the number of typical testors. As an example of this, consider the next set of experiments (shown in Table 5), where the three tested algorithms are faced with a specific combination of the φ and θ operators.

Table 6
Relative efficiency of the tested algorithms against the γ operator.

N	Test matrix	Rows	Cols	$\|\Psi^*\|$	LEX (%)	FastCTExt (%)	BT (%)
1	$\gamma(\theta(A, B))$	16	10	8	5.22	4.30	8.512
2	$\gamma^2(\theta(A, B))$	32	20	64	0.35	0.20	0.08
3	$\gamma^3(\theta(A, B))$	48	30	512	0.02	0.01	0.001
4	$\gamma^4(\theta(A, B))$	64	40	4096	0.003	0.001	

Although *BT's* efficiency was higher than those from the other two algorithms during the first experiment, it is also true that its efficiency rapidly falls as the number of typical testors increases. Table 5 shows that the same behavior, in a lesser magnitude, can also be observed for *FastCTExt*, and *LEX's* efficiency stands as the best for that set of experiments. Roughly the same phenomenon can be observed in Table 6, where a very similar combination of operators is used, only this time with γ and θ.

Finally, for the last set of experiments all three algorithms were faced with a test matrix designed with a combination of all three operators and with identity matrices (See Table 7). Just as it occurred in Tables 2 and 5, when the number of basic matrix rows is kept constant *LEX's* behavior emerges as the best performance. Table 7 also strengthens the hypothesis that an increase in the number of typical testors within the basic matrix, particularly an exponential increase, is the phenomenon that most considerably affects the performance of all thee algorithms, with *BT* as the most sensitive one.

5. A practical strategy for benchmarking

As mentioned in the introduction section of this paper, there is no commonly used strategy for experiment design when testing a new *TTA*. Making an effort to validate their proposed algorithms, several papers have chosen to experiment with real data sets (usually from the repository at UCI). Their first concern seem to be the performance of the *TTA* as the dimension of the input basic matrix grows. Papers like [3,4,9,16] and [17] all show one or more experiments result tables where a few data sets from the UCI repository are selected, the execution time of several *TTA* is compared, and the total number of typical testors found is reported. However, the criteria for selecting test matrices is not explained and seems random at best. Besides, experiment results analyzed during the previous section seem to suggest that a No-Free-Lunch kind of reasoning is in order; no particular *TTA* will always behave as the best option regardless the nature of the basic matrix it receives as input. In this regard, practically all previous works end up concluding that their proposed *TTA* have better performance than all others, but they never explicitly show experiment cases where their *TTA* falls behind other algorithms. Since those data sets are supervision samples, there is no unified methodology for pre-processing the data and preparing the final basic matrix that will be input to the *TTA*. Moreover, there is no record showing the "correct" set of typical testors for any of those data sets, consequently there is no correct proof of the completeness of the *TTA* result. All the above discussion support the claim that it is not enough to experiment with real data sets when testing a new *TTA*, and it is not enough

Table 7
Relative eficiency of the tested algorithms using combination of operators.

N	Test matrix	Rows	Cols	$\|\Psi^*\|$	LEX (%)	FastCTExt (%)	BT (%)
1	$\varphi(\gamma^2(\theta(I_2, I_3)))$	12	10	4	0.97	0.80	2.64
2	$\varphi^2(\gamma^2(\theta(I_2, I_3)))$	12	20	144	0.65	0.30	0.07
3	$\varphi^3(\gamma^2(\theta(I_2, I_3)))$	12	30	1296	0.48	0.14	0.003
4	$\varphi^4(\gamma^2(\theta(I_2, I_3)))$	12	40	6400	0.36	0.07	
5	$\varphi^5(\gamma^2(\theta(I_2, I_3)))$	12	50	22,500	0.30	0.04	

to randomly select the test matrices. Finally, the No-Free-Lunch assumption suggests the prudence of identifying at least one family of input matrices that will give a hard time to the proposed *TTA*.

A practical strategy for testing any *TTA* can be drawn from the results presented in this research. We propose that the design of experiments for testing *TTA* place particular care on the structural phenomena of the input matrices. The θ, γ, and ϕ operators provide a feasible way to generate the exact kind of phenomena tested. For example, consider the following test cases shown in this article:

- Tables 1, 2, 5 and 7 show experiments with basic matrices that show a linear increase in the number of columns. This type of matrices usually imply a polynomial grow in the number of typical testors and can be generated by applying the ϕ operator over the same known original matrix. The degree of that polynomial depends on the maximum typical testor length in the original matrix.
- Tables 1 and 5 show experiments with basic matrices that show an exponential increase in the number of rows. These matrices can be generated with the θ operator and imply a very slow linear increase in the number of typical testors. An example of this family of experiments can be found in Tables 1 and 5.
- Tables 1 and 6 show experiments with basic matrices that show an exponential grow in the number of typical testors. This phenomenon is generated with the γ operator.

The above examples constitute only one possible testing scheme among an infinite range of context circumstances that a particular *TTA* may face during its operation. Other operator combinations may provide some specific testing conditions while retaining the certainty that the number and structure of the set of typical testors is always known in advance.

6. Conclusions and recommendations

We have presented a practical, and solidly-grounded-on-theory tool that allows a researcher to test a *TTA* against specific predesigned phenomena within the input basic matrix. The combination of θ, φ and γ operators, in conjunction with sufficiently studied small basic matrices, turns out to be a versatile tool for generating almost any conceivable phenomenon a researcher could wish a *TTA* to be capable to tackle. The behavior observed during those tests can potentially yield enough information for identifying performance bottle-necks, and help design the appropriate fine-tuning procedures.

Deterministic *TTA*, as previously discussed, are guaranteed to find the complete set of typical testors in a basic matrix. The order in which a *TTA* traverses the search space ultimately determines its behavior and general performance. However, *TTA* sensitivity to specific phenomena, such as the growth in rows, columns, or typical testors within the basic matrix is, in general, not sufficiently assessed.

Meta-heuristic *TTA*, on the other hand, are pseudo-random search procedures that, by nature, do not offer enough guarantees about the completeness of the resulting typical testors set. In order to validate these kind of *TTA*, the proposed course of action is to test the *TTA* against sufficiently studied basic matrices (those for which the total number of typical testors is known in advance). Matrices that satisfy such requirements are generally small ones, not suited for serious testing purposes. The method herein proposed allows the generation, based on one original matrix, of increasingly larger matrices, with any desired dimensions, for which the total number of typical testors is always known.

As the presented experimental sets suggest, neither the order in which a *TTA* traverse the power set of basic matrix columns, nor the specific jumping mechanism it uses, are enough to accurately predict its computational behavior. Moreover, none of the currently known deterministic *TTA* have been tested against sufficiently large basic matrices. In the case of meta-heuristic *TTA* the situation is slightly worse; not only there is an extremely low number of published meta-heuristic *TTA*, but since meta-heuristic algorithms are designed to work over a huge search space, there has also been insufficient test procedures over those algorithms. If *TTA* are to be seriously used on real applications it is advised to custom fine-tune each algorithm for the specific kind of phenomenons it will most likely face in the context of the intended application.

In conclusion, *TTA* testing, under realistic conditions, appears to be a valuable asset for advancing the general state-of-the-art in pattern recognition by providing practical means for testing both deterministic and meta-heuristic *TTA*, fulfilling the current gap between theoretical developments and practical implementations.

Acknowledgments

This research was supported by Small Grants from Universidad San Francisco de Quito USFQ, Ecuador by Instituto Politécnico Nacional and CONACyT, México, particularly through the project Grant SIP-20160721.

References

[1] E. Alba, D. Guilcapi, J. Ibarra, New strategies for evaluating the performance of typical testor algorithms, Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, Springer, 2012, pp. 813–820.
[2] E. Alba-Cabrera, J. Ibarra-Fiallo, S. Godoy-Calderon, A theoretical and practical framework for assessing the computational behavior of typical testor-finding algorithms, Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, Springer, 2013, pp. 351–358.
[3] Y.S. Alganza, A.P. Porrata, Lex: a new algorithm for calculating typical testors, Rev. Cienc. Mat. 21 (1) (2003) 85–95.
[4] G. Diaz-Sanchez, I. Piza-Davila, G. Sanchez-Diaz, M. Mora-Gonzalez, O. Reyes–Cardenas, A. Cardenas-Tristan, C. Aguirre-Salado, Typical testors generation based on an evolutionary algorithm, in: Proceedings of the International Conference on Intelligent Data Engineering and Automated Learning-IDEAL 2011, Springer, 2011, pp. 58–65.
[5] M. Lazo-Cortés, J. Ruiz-Shulcloper, Determining the feature relevance for non–classically described objects and a new algorithm to compute typical fuzzy testors, Pattern Recognit. Lett. 16 (12) (1995) 1259–1265.
[6] M. Lazo-Cortes, J. Ruiz-Shulcloper, E. Alba-Cabrera, An overview of the evolution of the concept of testor, Pattern Recognit. 34 (4) (2001) 753–762.
[7] M.S. Lazo-Cortés, J.F. Martínez-Trinidad, J.A. Carrasco-Ochoa, G. Sanchez-Diaz, On the relation between rough set reducts and typical testors, Inf. Sci. 294 (2015) 152–163.
[8] F. Li, Q. Zhu, Document clustering in research literature based on NMF and testor theory, J. Softw. 6 (1) (2011) 78–82.
[9] A. Lias-Rodríguez, A. Pons-Porrata, Br: a new method for computing all typical testors, Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, Springer, 2009, pp. 433–440.
[10] A. Lias-Rodriguez, G. Sanchez-Diaz, An algorithm for computing typical testors based on elimination of gaps and reduction of columns, Int. J. Pattern Recognit. Artif. Intell. 27 (08) (2013) 1350022.
[11] M. Ortiz-Posadas, J. Martínez-Trinidad, J. Ruiz-Shulcloper, A new approach to differential diagnosis of diseases, Int. J. Biomed. Comput. 40 (3) (1996) 179–185.
[12] A. Pons-Porrata, R. Gil-García, R. Berlanga-Llavori, Using typical testors for feature selection in text categorization, Progress in Pattern Recognition, Image Analysis and Applications, Springer, 2007, pp. 643–652.
[13] A. Pons-Porrata, J. Ruiz-Shulcloper, R. Berlanga-Llavori, A method for the automatic summarization of topic-based clusters of documents, Progress in Pattern Recognition, Speech and Image Analysis, Springer, 2003, pp. 596–603.
[14] A. Rojas, R. Cumplido, J.A. Carrasco-Ochoa, C. Feregrino, J.F. Martínez-Trinidad, Hardware–software platform for computing irreducible testors, Expert Syst. Appl. 39 (2) (2012) 2203–2210.
[15] J. Ruiz-Shulcloper, M. Bravo, F. Aguila, Algoritmos BT y TB para el cálculo de todos los tests tıpicos, Rev. Cienc. Mat. 6 (2) (1985) 11–18.
[16] G. Sanchez-Diaz, G. Diaz-Sanchez, M. Mora-Gonzalez, I. Piza-Davila, C.A. Aguirre-Salado, G. Huerta-Cuellar, O. Reyes-Cardenas, A. Cardenas-Tristan, An evolutionary algorithm with acceleration operator to generate a subset of typical testors, Pattern Recognit. Lett. 41 (2014) 34–42.
[17] G. Sanchez-Diaz, M. Lazo-Cortes, I. Piza-Davila, A fast implementation for the typical testor property identification based on an accumulative binary tuple, Int. J. Comput. Intell. Syst. 5 (6) (2012) 1025–1039.
[18] R.A. Vázquez, S. Godoy-Calderón, Using testor theory to reduce the dimension of neural network models, Res. Comput. Sci. 28 (2007) 93–103.