FINDING TYPICAL TESTORS BY USING AN EVOLUTIONARY STRATEGY

EDUARDO ALBA-CABRERA
ROBERTO SANTANA
ALBERTO OCHOA-RODRIGUEZ
MANUEL LAZO-CORTES

Institute of Cybernetics, Mathematics and Physics
Calle E # 309 e/ 13 y 15, Vedado, c/p 10400, C. de La Habana, Cuba
Telef. (537)32-6748, 32-5026
FAX: (537)33-3373
{ealba, rsantana, ochoa, mlazo}@cidet.icmf.inf.cu

Abstract

The concept of testors appeared in the middle of the fifties. Testors and particularly typical testors, have been used in feature selection and supervised classification problems. Deterministic algorithms have usually been used to find typical testors. In this paper a new approach to find typical testors of a basic matrix is described. This approach is based on the application of the Univariate Marginal Distribution Algorithm as the kernel of an optimization strategy. The behavior of this algorithm is at least as well as the simple Genetic Algorithms with uniform crossover for the same kind of problems, but it is simpler and less costly in computational terms. Several experiments confirm the validity of this approach.

1. INTRODUCTION

The concept of testor appeared in the middle of the fifties [2], associated to the problem of detection of faults in electrical circuits.
Afterwards, it was applied to supervised classification problem in Geology [4]. In this case the concept of testor was adapted to determine the informational relevance of features that describe the set of objects. Up to date, the development of testor theory has strongly been connected to these two kind of problems: the detection of faults in electrical circuits [3], and the classification and recognition of objects [8], [13].
This paper deals in the second kind of problems and focus on the computation of the typical testors of a Boolean matrix [13]. This problem has usually been solved by deterministic algorithms (*DA*) that find all typical testors when the matrices are not large [14]. However, in some applications for large matrices, the use these *DA*s is impossible because their efficiency is very poor. The motivations for this work are the combinatorial nature of the testor calculus problem, and the successes obtained in the application of the evolutionary algorithms (*EA*) in the solution of combinatorial problems.
First attempt of an *EA* application to calculate the typical testors was published in [15]. In that paper, a Genetic Algorithm (*GA*) [5], [6] with one point crossover, and heuristic mutation and selection operators was used. The *EA* we employ in this paper is the Univariate Marginal Distribution Algorithm (*UMDA*) [10]. The behavior of this algorithm is at least as well as the simple *GA* with uniform crossover for the same kind of problems, but it is simpler and less costly in computational terms. In addition, we consider an improved objective function, which evaluates more accurately the typical testor candidates.
At first, we introduce the concept of typical testor for Boolean matrices. Then, we analyze the *DA*s reported to calculate the set of all typical testors and we propose a *UMDA* to obtain "as many as possible" typical testors from a basic matrix. By the end, we discuss several experiments in order to show the performance of the introduced *UMDA*.

2. THE CONCEPTS OF TESTOR AND TYPICAL TESTOR

Let U be a collection of objects, these objects are described by a set of n features and are grouped into l classes. By comparing feature to feature each pair of objects belonging to different classes, we obtain a matrix $M = [a_{ij}]_{mxn}$ where $a_{ij} \in \{0,1\}$, and m is the number of pairs. $a_{ij} = 0$ (1) means that the objects of pair denote by i are similar (different) in the feature j. Let $I = \{k_1,...,k_m\}$ be the set of the rows of M and $J = \{j_1,...,j_n\}$ the set of labels of its columns (features). Let $T \subseteq J$, $M_{/T}$ is the matrix obtained from M eliminating all columns not belonging to the set T.

Definition 1.- A set $T = \{j_{i_1}, \ldots, j_{i_s}\} \subseteq J$ is a *testor* of *M* if no zero row in $M_{/T}$ exists.

Definition 2.- The feature $j_{i_r} \in T$ is *typical* with respect to (wrt) *T* and *M* if $\exists q$, $q \in \{1,\ldots,m\}$ such that $a_{k_q j_{i_r}} = 1$ and for $s > 1$ $a_{k_q j_{i_p}} = 0$, $\forall p$, $p \in \{1,\ldots,s\}$ $p \neq r$.

Definition 3.- A set *T* has the property of typicality wrt a matrix *M* if all features in *T* are typical wrt *T* and *M*.

Proposition 1.- A set $T = \{j_{i_1}, \ldots, j_{i_s}\} \subseteq J$ has the property of typicality wrt matrix *M* if and only if an identity matrix can be obtained in $M_{/T}$, by eliminating some rows.

Definition 4.- A set $T = \{j_{i_1}, \ldots, j_{i_s}\} \subseteq J$ is denominated *typical testor* of *M* if it is a testor and it has the property of typicality wrt *M*.

Let ***a*** and ***b*** be two rows from *M*.

Definition 5.- We say that $\boldsymbol{a} < \boldsymbol{b}$ if $\forall i$ $a_i \leq b_i$ and $\exists j$ such that $a_j \neq b_j$.

Definition 6.- ***a*** is a *basic row* from *M*, if there is not any row less than ***a*** in *M*.

Definition 7.- The *basic matrix* of *M* is the matrix *M'* only containing all different basic rows of *M*.

Proposition 2.- The set of all typical testors of *M* is equal to the set of all typical testors from the basic matrix *M'*.

Let $\psi^*(M)$ be the set of all typical testors of the matrix *M*.

According to proposition 2, to obtain the set $\psi^*(M)$ it is very convenient to find the matrix *M'*, and then, to calculate the set $\psi^*(M')$. Taking into account that *M´* has equal or less number of rows than *M*, the efficiency of the algorithms should be better with *M'* than with *M*. In fact, all *DA*s described in this paper work on *M'*. The set $\psi^*(M')$ of all typical testors from a Boolean matrix *M'* allows the determination of the informational relevance of features [8], the reduction of the dimension of objects descriptions and the use of these testors as support sets for partial comparison between objects [13].

3. DETERMINISTIC ALGORITHMS TO CALCULATE $\psi^*(M')$

Trivial Algorithm

In order to determine $\psi^*(M')$, we can use a very simple algorithm verifying whether or not each subset of *J* fulfills the properties of testor and typicality. Let *m* be the number of rows in *M'*. It can be observed that the number of operations is approximately

$$c = \left(\binom{n}{1} + 2\binom{n}{2} + 3\binom{n}{3} + \ldots + n\binom{n}{n}\right)m$$

$$m2^n < c < mn2^n.$$

Therefore, the basic need of searching more efficient algorithms.

Yablonskii's algorithm

Yablonskii's algorithm [3] uses the language of Formal Logic. A conjunctive normal form (*cnf*) with *m* elementary disjunctions is created. Disjunctions are associated one to one to rows of *M'* by joining all column labels that have unitary elements in that row via disjunctive operator. Then, starting from the *cnf*, a disjunctive normal form (*dnf*) is obtained by opening the parenthesis and considering column labels as Boolean variables. Afterwards, some algorithm to find the *reduced dnf* is applied. The sets of column labels associated to each elementary conjunction of reduced *dnf* are typical testors of *M'*.

Algorithm BT

In order to apply the algorithm *BT* [14], the set $\wp(J)$ of all subsets of column labels of *M´* is ordered. This total order is defined by the number associated to the binary characteristic vector of each element of $\wp(J)$. This algorithm, as well the trivial one, verifies the fulfillment of the testor and typicality properties but, as a difference, it does not verify all the elements of $\wp(J)$. The algorithm *BT* "jumps" over some sets of $\wp(J)$, i.e. it does not consider several successors of the verified set because it is possible to know that they are not typical testors.

Algorithm REC

The algorithm *REC* [14] is based on the same principles of algorithm *BT*, but the "jumps" are different.

Algorithm CC

The algorithm *CC* [14] proposes a different approach. The elements of the set $\wp(J)$ are not verified but the internal structure of matrix *M'* (positions of the zero and one entries) is analyzed. *CC* finds all unitary submatrices of *M'* that contain as many rows as possible (maximal subsets of columns having the property of typicality). Then, the property of testor is verified for each one of these sets.

Algorithm CT

This algorithm [14], as well as the *CC*, starts from the internal structure of matrix *M'*, but as a difference , it finds all *complete sets*. A complete set is a set of columns that is testor and contains a submatrix with the following characteristics:

1.- Dominant diagonal of ones.

2.- Zeros in the bottom triangular.

Algorithm YYC

The algorithm *YYC* [14] also analyzes the internal structure of the matrix by checking the fulfillment of the properties. It is carried out by adding a row each time and updating the set of all typical testors up to the current row. The algorithm *YYC* combines ideas from both algorithms: Yablonskii's algorithm and *CC*.

All the above described *DA*s calculate the set $\psi^*(M')$. These algorithms are not feasible for large matrices in terms of time. This is consequence of the exponential nature of the problem. Therefore, it would be convenient to find a subset of the set $\psi^*(M')$ in a viable time. This subset gives us the possibility for the application of the testor's tools when a large *M'* is generated.

4. POPULATION BASED SEARCH METHODS THAT USE SELECTION AND THE PROBLEM OF CALCULATING A SUBSET OF THE SET $\psi^*(M')$

Population Based Search Methods that use Selection (*PBSMS*) do the search of solutions using a set of points instead of a single one. Starting from a set of initial points (called population), these algorithms select a set of promising points and generate a new population. This process is repeated until a stop condition is satisfied.

The *GA*s belong to the *PBSMS* class. In *GA*s, new populations of points (usually called chromosomes or individuals) are created by applying a set of genetic operators to the population of selected individuals. Classical genetic operators comprise, besides Selection, the Crossover and Mutation ones. In crossover two individuals are picked from the selection set, and two offspring are created by recombining the genetic information contained in their parents. Mutation applies by modifying the values (alleles) of variables (genes) in the chromosomes.

*GA*s have shown to be very effective when applied to a wide set of optimization problems, nevertheless they have experienced difficulties in optimizing functions with nonlinear interacting variables [11].

As we stated before, a *GA* has already been used for calculating a subset of the set $\psi^*(M')$. We will use a different kind of *PBSMS* and an improved objective function.

It has been stated [11] that a good search strategy for *PBSMS* is to generate new points with a similar probability distribution to that existing in the selected set. This is:

$$P(x,t+1) \approx P^s(x,t)$$

The class of *PBSMS*, which estimates a probability distribution of points in the selected set and uses this information to generate new points is called Estimation of Distribution Algorithms (*EDA*). Although *EDA* have shown a better performance than the *GA*s, the estimation of distributions is also a difficult problem [7], [12]. A particular case is the Univariate Marginal Distribution Algorithm (*UMDA*), which has a very simple probabilistic model. This reason, added to the above mentioned *EDA* advantages over *GA* determined our choice.

Another point is that the stochastic attribute of these methods allows finding different solutions each time the algorithm is executed. Diverse solutions can be found in the same population when they are combined with appropriated niching methods [9].

It is clear, that given a matrix *M*', multiple typical testors can be found, this fact can lead the *PBSMS* in different directions (sometimes opposite) during the search. This would require more time to reach one of the solutions.

When there are typical testors very hard to find, we are also in presence of isolation.

Finally, another characteristic of the typical testor problem is that most of the information about the problem structure is stored in the basic matrix. It is an open question the influence of the number of testors, their length and the internal structure of basic matrix in the complexity of the search.

5. OPTIMIZATION APPROACH FOR FINDING A SUBSET OF THE SET $\psi^*(M')$

Before introducing the *UMDA* we present its components.

Representation: Vectors (chromosomes) are binary strings of length n (number of columns of the analyzed basic matrix) and they represent the characteristic vectors of the subsets of features.

Initial population: It is a set of N randomly chosen characteristic vectors.

Selection: Truncation selection with a truncation parameter t_c, $0<t_c<1$

Each chromosome $x_k = (x_{k1},...,x_{kn})$; $k = 1,...,N$; $x_{ki} \in \{0,1\}$; $i = 1,...,n$ is evaluated using the following objective function:

$$f(x_k) = \alpha \frac{t(x_k)}{m} + (1-\alpha)\frac{p(x_k)}{\sum_{v=1,..,n} x_{kv}},\ 0 < \alpha < 1$$

where $t(x_k)$ is the number of rows of *M'* that contain some unitary element in the columns of T_{x_k}, T_{x_k} is the set which characteristic vector is x_k, $p(x_k)$ is the number of typical features wrt T_{x_k} and *M'*. α is a weighting coefficient.

Notice that for any chromosome x_k, $0 \le f(x_k) \le 1$. If the function *f* reaches the maximal value 1 for a chromosome x_k, then T_{x_k} is a typical testor. It is held if and only if any row of *M'* has at least one unitary element and all the features of the set T_{x_k} are typical. The higher the value of $f(x_k)$, the nearer to the fulfillment of the typicality and testor properties the set T_{x_k} wrt the matrix *M'* is. The previous *GA* [15] does not consider this information. A very simple function is taken as the objective function of the *GA* that takes only three values: typical testor, no testor and no typical testor. This function does not consider the fact that the sets can be near of (or far from) the fulfillment of the testor and typicality properties. α is parameter that controls the trade-off between typicality and testor properties.

We describe the *UMDA* in Fig 1. *UMDA* exploits the additive genetic variance mainly. Nevertheless it has been applied to several problems where there are interactions among the variables. The class of evolutionary optimization algorithms that make use of univariate distributions comprises also the Probabilistic Based Incremental Learning (PBIL) [1].

UMDA

1. Set $t \leftarrow 1$ The initial population is generated.
2. Select $k \le N$ points according to a selection method. Compute the marginal frequencies $p^s_i(x_i,t)$ of the selected set.
3. Generate *N* new points according to the distribution $p(x,t+1) = \prod_{i=1}^{n} p^s{}_i(x_i,t)$. Set $t \leftarrow t+1$
4. If the termination is not met, go to step 2.

Figure 1. Univariate Marginal Distribution Algorithm

The termination criteria for the *UMDA* can be a low fitness variance in the population (indicating an early convergence to suboptimal values), a fixed number of generations, or the appearance of the desired solution in the population. In our experiments, after the optimum has been found we allow the algorithm to execute during *q* additional generations in order to explore the local space around the optimum found. Otherwise, the algorithm stops when no typical testor has been found after a maximum number of generations (*Maxgen*). In Fig. 2 we observe how the *UMDA* is inserted in the general optimization algorithm. Similarly to the previous one, we use as the stop condition of the optimization algorithm the number of iterations, the elapsed time or the number of found testors.

1. Start from an empty list of typical testors *TTLIST*. Iter ← 1
2. While not Stop Condition
3. Execute *UMDA*
4. For all solutions in the final population
5. If the solution is not in the *TTLIST* add it to the list
6. Iter ← Iter + 1

Figure 2. Optimization approach for the typical testor problem

6. EXPERIMENTS AND RESULTS

The first experiment focus on a comparison between the performances of the *UMDA* and the simple *GA* published in [15], based on the time spent to compute the same number of typical testors. As in [15], all experiments were carried out on a PC with a Pentium 150Mhz processor.

Given a basic matrix that determines the number of variables in our codification of the optimization approach, and the *UMDA*, we can completely specify the algorithm to use with a tuple of 6 parameters {*Iter, popsize, Truncation, Maxgen*, α, *q*}. The parameters correspond, respectively, to the maximal number of iterations of the optimization algorithm, the population size of the evolutionary algorithm, the parameter of the truncation selection, the maximum of generations allowed, parameter alpha of the objective function, and the additional number of generations after one optimum is reached.
The used set of parameters was {no defined, 100, 0.15, 15, 0.1, 3}. The stop condition was the number of typical testors that were found by the algorithm (see Table I).
A collection of four basic matrices described in [15] was tested. The results are shown in Table I. In all cases the efficiency of the *UMDA* is much better than the simple *GA*. Besides, the number of evaluations of the *UMDA* is significantly less than the simple *GA* one.

Table I. Comparison of the simple *GA* and the *UMDA* performances.

Matrices	Typical testors found	Simple *GA*		*UMDA*	
		Time	Evaluations	Time	Evaluations
40x42	655	241 s	1 400 000	19 s	142 500
269x42	318	1043 s	5 000 000	28 s	89 800
209x47	1967	1816 s	5 000 000	114 s	706 900
1215x105	105	19 033 s	22 500 000	1 020 s	336 700

Table II shows the comparison between *UMDA* and *DA*s. In [14] was experimentally stated that the most efficient *DA* is the algorithm *CT*. Therefore, in all the experiments with the *DA*s this algorithm was used. For the first three matrices, the number $|\psi^*(M')|$ is known, because the *DA* calculates the set $\psi^*(M')$ in a relative small time. It can be appreciated that the *UMDA* calculates in the same time around half of the $\psi^*(M')$. For the remaining three matrices, the number $|\psi^*(M')|$ is unknown because the *DA* works more than two days and does not finish. In the Table II, the number of typical testors found by *UMDA* in three hours is shown. In this experiment, as in the previous one, the same set of *UMDA* parameters for all matrices except the last was used. Taking into account the dimension of the last matrix the population size was increased to 200.

Table II. Comparison of the *DA* and the *UMDA* performances

Matrices	Total num. of typical testors	Det. Algorithm Time	Typical testors found by *UMDA*
40x42	8963	49 s	2 047[1]
80x42	32 277	1722 s	16 254[2]
110x42	65 299	4820 s	36 683
269x42	?	+ 2 days	(in 3 hours) 71 448
209x47	?	+ 2 days	(in 3 hours) 65143
1215x105	?	+ 2 days	(in 3 hors) 2421

In other experiments we studied the dynamics of the algorithm, the influence for the search of both, the parameters and the characteristics of the specific instances of the problem.
We use a matrix corresponding to a real problem with 42 variables and compare 16 different algorithms which are determined by the following set of parameters: {10000, [100, 200, 300, 400], 0.15, 15, [0.2, 0.4, 0.6, 0.8], 3}. Numbers in brackets indicate the different values used in the experiments for the population size and the parameter α.
Table III shows the number of different typical testors found for each combination. It can be appreciated that the choice of parameter α influences the quality of the search. In opposition to expectations, by augmenting the population size we do not achieve an increment in the number of solutions. This could be explained because with a smaller population size we have a stronger selection pressure when the parameter of the truncation selection is fixed as the case in Table III is.

Table III . Influence of alpha and population size in the number of typical testors found by *UMDA*.

[1] This number corresponds to the median in ten applications of the *UMDA*.
[2] This number corresponds to the median in three applications of the *UMDA*.

α Popsize	0.2	0.4	0.6	0.8
100	3989	4434	5009	5606
200	3936	4163	4585	5257
300	3770	3896	4418	4983
400	3704	3784	4325	4826
Tot.	15399	16277	18337	20672
	6186	6524	7110	7697

We also evaluate the behavior of the algorithm analyzing the number of evaluations needed to reach an optimum. Here we divided the number of evaluations made by the algorithm by the number of testors found (considering repetitions) during the search. Table IV shows the values for the algorithms. In comparison to the expected number of evaluations needed by a random walk algorithm ($\approx 2^{42}/2^{14}=2^{28}$) all the entries show a more efficient performance. Nevertheless we believe these values can be improved by looking for the optimal set of parameters of the algorithm.

Table IV. The ratio between the number of evaluations and the number of found typical testors.

α Popsize	0.2	0.4	0.6	0.8
100	875.4	710.3	532.3	428.3
200	1235.7	1078.4	931.5	872.3
300	1628.6	1495.3	1292.8	1274.9
400	2033.2	1899.6	1672.5	1644

Figure 3 shows the distribution of the length for all the typical testors found for different values α. We can see that the choice of α does not influence only the number of testors found but also their length. When $\alpha = 0.8$ we have a higher probability of finding testors of short length, this relation is changed when $\alpha = 0.2$.

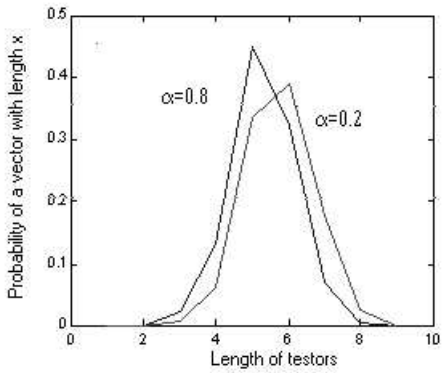

Figure 3. Distribution of the length for all the typical testors found for different values α.

CONCLUSIONS

A new approach to calculate typical testors of a basic matrix is described. This approach is based on the application of the *UMDA* as the kernel of the optimization strategy.

The superior performance of the proposed *UMDA* over the simple *GA* reported in [15] is experimentally demonstrated. The *UMDA* calculates the same number of typical testors as the reported *GA* in a fewer number of evaluations and spending significantly less time.

This performance can be a consequence of the following points:

- the *UMDA* advantages over *GA* inherent to its internal optimization scheme
- the introduced objective function, that allows evaluating any subset of the whole set of features considering its proximity to the fulfillment of the properties of testor and typicality. This fact is very important for the selection step of the *UMDA*.

- the global optimization scheme in which the *UMDA* is inserted. For instance, by allowing the *UMDA* runs *q* generations after the first optimum has been found, we can get other solutions without using mutation and crossover operators.

Further planned research includes the application of hybrid strategies, combining traditional *DA*s and evolutionary algorithms, and the study of the convenience of applying different *EDA* to calculate typical testors of a basic matrix.

REFERENCES

[1] Baluja S. Population-Based incremental learning: A method for integrating genetic search based function optimization and competitive learning(Tech Rep. No. CMU-CS-94-163) Pittsburg, PA: Carnegie Mellon University, 1994.

[2] Cheguis, I. A. and Yablonskii, S. V. About testors for electrical outlines. Uspieji Matematicheskij Nauk 4, (66) pp. 182-184 Moscow (In Russian), 1955.

[3] Cheguis, I. A. and Yablonskii, S. V. Logical Methods for controlling electrical systems. Trudy Matematicheskava Instituta imeni V. A. Steklova LI, pp. 270-360. Moscow (In Russian), 1958.

[4] Dmitriev, A. N.; Zhuravlev, Yu. I. and Krendeleiev, F. P. On the mathematical principles of patterns and phenomena classification. Diskretnyi Analiz 7. pp. 3-15. Novosibirsk, Russia (In Russian), 1966.

[5] Goldberg D. E. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, 1989.

[6] Holland J. H. Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, Michigan, 1975.

[7] Larrañaga P., Etxeberria R., Lozano J. A., Peña J. M. Optimization by learning and simulation of Bayesian and Gaussian networks. Technical Report EHU-KZAA-IK-4/99 Intelligent Systems Group Dept. of Computer Science and Artificial Intelligence. University of the Basque Country. December 1999. Also at http://www.sc.ehu.es/isg.

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

[9] Mahfoud S. W. Niching methods for genetic algorithms, IlliGAL Report No.95001. University of Illinois at Urbana-Champaign , 1995 also at ftp://ftp-illigal.ge.uiuc.edu/pub/papers/IlliGALs/95001.ps.Z

[10] Mühlenbein, H. The equation for response to selection and its use for prediction, Evolutionary Computation 5, pp. 303-346, 1998.

[11] Muhlenbein, H., Mahnig, T., Ochoa, A.: Schemata, Distributions and Graphical Models in Evolutionary Optimization, Journal of Heuristic, Vol. 5, No. 2, 1999.

[12] Pelikan M, Goldberg D. E. and Lobo F. A survey of optimization by building and using probabilistic models, IlliGAL Report No.99018. University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, 1999.

[13] Ruiz-Shulcloper, J., Lazo-Cortés M., Alba-Cabrera E. An overview of the concept of testor. Pattern Recognition Journal. Pergamon Press, 2000. (in press)

[14] Sánchez Díaz G. Program and Design of Efficient algorithms to calculate typical testors from a basic matrix. M.Sc Thesis BUAP. 1997. (In Spanish)

[15] Sánchez-Díaz, G., Lazo-Cortés M., Fuentes-Chávez O. Genetic Algorithm to calculate minimal typical testors. Proceedings of the IV Iberoamerican Symposium on Pattern Recognition, pp. 207-214, 1999. (In Spanish)