YYC: a fast performance incremental algorithm for finding typical testors

Eduardo Alba-Cabrera(1), Salvador Godoy-Calderon(2), Julio Ibarra-Fiallo(1), Fernando Cervantes-Alonso

April 26, 2015

(1) Departamento de Matemáticas, Universidad San Francisco de Quito, Colegio de Ciencias e Ingeniería,

Diego de Robles y Vía Interoceánica, Quito, ECUADOR.

(2) Centro de Investigación en Computación (CIC), Instituto Politécnico Nacional (IPN),

Av. Juan de Dios Bátiz, Esq. Miguel Othón de Mendizábal. Col. Nueva Industrial Vallejo. D.F., MÉXICO

(3) CITRIX Systems,

851 West Cypress Creek Road, Fort Lauderdale, FL 33309, USA

ealba@usfq.edu.ec;sgodoy@cic.ipn.mx;jibarra@usfq.edu.ec;fernando.cervantes@citrix.com

Abstract

The last few years have seen an important increase in research publications dealing with external algorithms, while internal ones have been almost forgotten or modified to behave as external on the basis of their alleged poor performance. In this research we present a new internal typical testor-finding algorithm called YYC that incrementally calculates typical testors for each row of the basic matrix by searching for compatible sets. The experimentally measured performance of this algorithm stands out favorably in problems where other external algorithms typically show very low performance.

Keywords: feature selection, testor theory, typical testor algorithms, YYC algorithm.

1 Introduction

Testor Theory [6], has repeatedly proven itself as one of the most useful tools for feature selection problems in pattern recognition[5, 16]. Typical testors have been used in solving a wide range of practical problems like diagnosis of diseases [9], text categorization [11], document summarization [10] and document clustering [7]. Testor theory has had an important growth over the past ten years, particularly regarding the development of new algorithms, like [4, 8, 14, 15]. All those algorithms are generally refereed to as Typical Testor-finding Algorithms (TTA). Interestingly, in almost all those cases researchers propose external algorithms. There are even cases of internal algorithms which were adapted to operate externally, as is the case for [14].

It is common knowledge that typical testor-finding algorithms follow two main strategies. On one hand, external algorithms induce an order over the power set of features used to describe objects in some previously specified domain, and then some logical properties of that order are used to optimally search for typical testors. On the other hand, internal algorithms do not test the power set of features in the domain; their strategy lies in iteratively selecting some entries from the basic matrix induced by the studied domain, and use them to construct typical testor candidates.

The performance of both, external and internalTTA is experimentally measured by counting the number of feature subsets tested by the algorithm when applied over a specific problem, and comparing that quantity with the total number of typical testors to be found on that same problem. An ideal TTA would test only those feature subsets which are typical testors, however it is common knowledge that no search procedure has better performance than an iteratively constructive algorithm. For external TTA efficiency heavily depends on the order in which the power set of features is traversed, along with the magnitude of its *jumps* (i.e. the number of subsets not tested according to the established order). On the contrary, for internal TTA efficiency depends on how many elements from the basic matrix it combines to construct a testor candidate.

In this paper we present a new internal TTA, named YYC (*Yablonski* and *Compatible sets*), that identifies the set of typical testors implied by each row of the basic matrix. Once the set of typical testors from the first row of the basic matrix is calculated, the algorithm successively updates that set by adding new feature combinations from each new row, and by removing those feature combinations that cease to be typical testors when combined with the information from a new row in the basic matrix. The procedure that determines whether a particular feature combination is to be updated relies on the identification of compatible sets (see Theoretical background). We also propose a fast procedure for finding those compatible sets and include it within the YYC algorithm.

The rest of this paper is structured as follows. In Section 2, the theoretical background for typical testors, compatible sets, and the YYC algorithm is set. Section 3, presents the proposed procedure for finding compatible sets, as well as the YYC algorithm in detail. Section 4, outlines the experiments carried out comparing YYC's performance versus some external TTA, and the analysis of the yielded results. Finally, we draw some relevant conclusions in Section 5.

2 Theoretical background.

Practically all published researches in Testor Theory, work with a Boolean matrix that holds all the information about the comparison of objects belonging to different classes within a supervision sample; that matrix is called a difference matrix (DM). Each column of the DM represents a specific feature perceived from all objects under study, and each row holds the set of feature comparison values for a pair of objects. In such a comparison matrix an entry 0 means that the two objects being compared have a similar value in one feature, and an entry 1 means that the value, for the corresponding feature, is dissimilar in each compared object.

Let $\mathcal{R}_{DM} = \{r_1, ..., r_m\}$ and $C_{DM} = \{x_1, ...x_n\}$ be the set of rows and columns of a difference matrix, 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, doesn't have any row composed exclusively by zeros. A testor is then a set of features capable of discriminating all objects in a supervision sample without causing confusion among those belonging to different classes. When that set of features is also minimal with respect to the inclusion criterion, then it is called a typical testor.

Two rows from $\mathcal{R}_{DM}$, r_p and r_q, are said to be incomparable if $(\exists i)\,[r_{pi} \geq r_{qi}] \wedge (\exists j)\,[r_{qj} \geq r_{pj}]$. A row in is called a basic row if it is incomparable with any other row in $\mathcal{R}_{DM}$. 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 [12].

Within a BM a set of elements are said to form a compatible set if, under some row and column rearrangement, those elements shape into an identity matrix. When some BM elements are known to form a compatible set, then their corresponding subset of columns, form a typical testor in that BM (typicality condition) iff the sub-matrix defined by those columns has no row exclusively formed by zeros (testor condition).

The algorithm herein proposed (YYC), which is explained in the next section, heavily relies on the idea that a row-by-row analysis of the BM provides two significant advantages: on one hand, the set of typical testors can be initialized with the typical testors found on the first row, which are extremely easy to find since each column with a value 1 is a typical testor for that row. On the other hand, for each successive BM row added to the analysis, there are only two possible options: either each previously known typical testor is preserved by virtue of some value 1 in any of the columns that conform it (i.e. maintains the testor condition), or it must be combined with some other columns to fulfill the testor condition. These cases are handled by the YYC algorithm in the most efficient possible way; the BM is analyzed and the set $\psi^*$of all typical testors is incrementally updated for each new row of the BM. When a previously known typical testor looses its property of being testor, the algorithm searches for other columns (typical testor candidates) that could be combined with it, and that would preserve its quality of being a typical testor. However, the typical testor candidates are only selected from those columns of the most recent basic matrix row that have a value 1. That way the search space is reduced and not all possible combinations of columns have to be tested, just those with a real chance of enhancing the previously known set of typical testors.

3 The YYC Algorithm

As stated before the fundamental idea underlying theYYC algorithm is that, instead of analyzing the whole basic matrix in order to determine the set ψ^* of all typical testors within it, that process can be break down into an incremental one that, during each iteration i, calculates the set of all typical testors embedded within the first i rows of the basic matrix.

Following that idea, the YYC algorithm starts by extracting the typical testors implied by the first row of the basic matrix. From that point on, each new row r_i of the BM that is analyzed, triggers an update on the known set of typical testors. If a subset of columns was previously known to be a typical testor, and the new BM row has, at least one 1 in any of those columns, then by definition the sub-matrix defined by those columns do not have any row exclusively formed by zeros, and therefore it is still a testor (a typical testor). Else, if the new row shows only zeros in the columns of a previously known typical testor, then some new columns must be included in order to preserve its testor condition. The candidate columns to be included can only be those with a value 1 on the new BM row. Including any 1-valued column to the column subset will certainly preserve its testor condition, however, to also be typical the sub-matrix determined by those columns and rows must, under some row and column rearrangement, include a compatible set. Algorithm 1 shows the pseudo-code for the YYC algorithm.

Algorithm 1: YYC

input : a basic matrix BM with $bm \geq 2$ rows.

output : the set $\psi^*$of all typical testors embedded in that matrix.

```
Initialize ψ* = ∅
Read BM's first row (r_1)For each column x_j , such that r_1[x_j] = 1, Add {x_j}to ψ*
For each row r_i, i = 2..bm do
    Initialize ψAux = ∅
    For each τ_j ∈ ψ* do
    If (∃x_p ∈ τ_j)[r_i[x_p] = 1] then
        Add τ_j to ψAux
    else
        For each x_p ∈ r_i such that r_i[x_p] = 1 do
            If FindCompatibleSet(τ_j, x_p) [See Algorithm2] (Hits) then
              Add τ_j ∪ {x_p}to ψAux
    Let ψ* = ψAux
Return ψ* as the final answer
```

The critical performance-related step in Algorithm 1 is the search for compatible sets. Several different algorithms can be used for that goal; however, in order to do so efficiently, we propose a specific procedure (see Algorithm 2). We define the following functions: $HSum(row)$ and $VSum(col)$ which respectively return the horizontal sum of the elements in one specific row of the basic matrix, and the vertical sum of the elements in one column of the same matrix. Using the above functions we propose the following algorithm:

Algorithm 2: $FindCompatibleSet(\tau, x_p)$
input : a subset of columns known to be a typical testor up to the last BM row.
\ \ \ \ \ \ x_p the column id of an element from the new BM row which has value 1.
output : TRUE if a compatible set can be found within the sub-matrix defined by $\tau \cup x_p$, FALSE otherwise.

```
Define RefSM as the sub-matrix of BM defined by the columns in τ ∪
{x_j}, and the current read rows.
Evaluate Condition1 to be TRUE iff |{r_s ∈ RefSM, HSum(r_q) = 1}| ≥ |τ ∪ {x_p}|
Redefine RefSM = {r_s ∈ RefSM, HSum(r_s) = 1}
Evaluate Condition2 to be TRUE iff (∀x_k ∈ RefSM)[VSum(x_k) = 1]
Return the truth value of the Boolean expression [Condition1ANDCondition2] as the final answer
```

Undoubtedly, several different procedures can be defined to find if a sub-matrix defined by some columns in BM contain an identity matrix. Considering its one-pass nature, Algorithm 2 was the selected choice for that task.

4. Comparative performance testing

For experimentally assessing the performance of the YYC algorithm we comparatively test it against some external algorithms, namely LEX [15], $FastCTExt$ [14], and BT [13]. For each experiment, a custom-designed basic matrix was designed, following the method described in [3], and whose complete set of typical testors was known a priori. Following the same method, we asses the efficiency of any TTA by comparing the number of tested column subsets against the previously known number of typical testors found in the problem. Consequently, the efficiency of a TTA is the ratio of the number of typical testors to be found in that problem, and the number of column subsets tested by the TTA (labeled as $Hits$). In an 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%. Algorithm 1 marks when the $Hits$ counter is to be incremented while running the YYC algorithm. All of the following experiments are summarized in tables showing the number of rows, columns, and typical testors to find (labeled TT), as well as the number of registered hits and the resulting efficiency for each tested TTA. Please note that, the method used for assessing the performance of a TTA is completely independent from the hardware platform it runs on. Nevertheless, we proclaim that all the following experiments were run on an Intel i7 processor, with 4GB in RAM, and with a GNU/Linux operating system.

Finding the only typical testor embedded in an identity matrix has always proven to be a formidable challenge for almost all TTA. For that reason experiment number one, tested the compared TTA against variable size identity matrices. Each one of those runs has only one typical testor embedded into a matrix of successive bigger sizes. All four algorithms were tested against each matrix, and Table 1 summarizes the obtained results.

N	Rows	Cols	TT	LEX		FastCTExt		BT		YYC	
				Hits	Efficiency	Hits	Efficiency	Hits	Efficiency	Hits	Efficiency
1	5	5	1	6	16.67%	16	6.25%	6	16.67%	4	25.00%
2	10	10	1	11	9.09%	512	0.20%	11	9.09%	9	11.11%
3	15	15	1	16	6.25%	16384	0.01%	16	6.25%	14	7.14%
4	20	20	1	21	4.76%	524288	0.0002%	21	4.76%	19	5.26%
5	25	25	1	26	3.85%	16777216	0.000006%	26	3.85%	24	4.17%

Table 1. Comparative performance test against identity matrices.

Table 1 clearly shows how the YYC algorithm slightly outperforms the LEX andBT algorithms whose performance turns out to be the exact same, while the $FastCTExt$ algorithm shows a surprisingly low perform. Both, the initial reorder of the basic matrix, and the use of Algorithm 2 can be regarded as the reasons behind YYC's higher efficiency during this experiment.

Experiment number two set test matrices with a constant number of rows, a successive polynomial increase in the number of columns, but an exponential increase in the number of typical testors to find. Table 2 summarizes experiment two's results

N	Rows	Cols	TT	LEX		FastCTExt		BT		YYC	
				Hits	Efficiency	Hits	Efficiency	Hits	Efficiency	Hits	Efficiency
1	16	10	8	153	5.23%	186	4.30%	94	8.51%	76	10.53%
2	16	20	50	2648	1.89%	5666	0.88%	8861	0.56%	1056	4.73%
3	16	30	156	15635	1.00%	59536	0.26%	444459	0.04%	5652	2.76%
4	16	40	356	57311	0.62%	246700	0.14%	19762606	0.002%	19984	1.78%
5	16	50	680	160001	0.42%	1153242	0.06%	75336732	0.001%	54700	1.24%

Table 2. Performance test with exponential growth in the number of typical testors.

The performance of all compared algorithms decreases while the number of typical testors to find increases. Notably the YYC algorithm shows the lowest decreasing efficiency rate. After maintaining the number of rows constant during the last experiment, it seems just fair to perform another experiment with just the opposite scenario: an exponential (quadratic) growth in the number of rows of the test matrix, albeit just a polynomial increase in the number of columns and typical testors to find. Table 3 summarizes the results for this experiment.

N	Rows	Cols	TT	LEX		FastCTExt		BT		YYC	
				Hits	Efficiency	Hits	Efficiency	Hits	Efficiency	Hits	Efficiency
1	4	5	4	11	36.36%	11	36.36%	9	44.44%	9	44.44%
2	16	10	8	106	7.55%	154	5.19%	93	8.60%	90	8.89%
3	64	15	12	918	1.31%	1431	0.84%	767	1.56%	684	1.75%
4	256	20	16	7618	0.21%	13538	0.12%	5194	0.31%	4128	0.39%
5	1024	25	20	70278	0.03%	82456	0.02%	49395	0.04%	22365	0.09%

Table 3 Performance test with exponential growth in the number of rows.

This experiment clearly shows that all TTA, regardless of whether they are internal or external, have a hard time finding typical testors within this type of test matrix, but again, YYC shows a slightly lower decrease rate in efficiency.

For the last experiment we wanted to test the hypothesis that, since the the YYC algorithm processes only those basic matrix elements with value 1, its efficiency could depend on the density of the basic matrix. In order to test that hypothesis a single general model for a basic matrix, with 4 rows and from 5 to 100 columns, was modified to change its density and then tested with the YYC algorithm. Table 4 shows the results of the experiment.

N	Rows	Cols	$Density = 0.3$			$Density = 0.4$			$Density = 0.6$			$Density =$		
			TT	Hits	Efficiency	TT	Hits	Efficiency	TT	Hits	Efficiency	TT	Hits	E
1	4	5	2	5	40.00%	4	7	57.14%	4	9	44.44%	8	11	
5	4	25	750	1525	49.18%	500	675	74.07%	180	525	34.29%	200	275	
10	4	50	11000	22100	49.77%	4000	5200	76.92%	1210	3600	33.61%	800	1100	
15	4	75	54000	108225	49.90%	13500	17325	77.92%	3840	11475	33.46%	1800	2475	
20	4	100	168000	336400	49.94%	32000	40800	78.43%	8820	26400	33.41%	3200	4400	

Table 4. Performance test with different matrix densities.

As Table 4 shows, the behavior of YYC's performance is not clearly related to the basic matrix density. While in some instances of the experiment, the efficiency of the YYC algorithm increases along the number of columns and typical testors, in some others it behaves on the opposite way. There is even the case of the last three columns in table 4, where the efficiency kept constant, disregarding the hypothesis of direct dependency on the basic matrix density.

5. Conclusions

We have presented a new internal typical testor-finding algorithm with two highlighted features: first, instead of analyzing the whole basic matrix it incrementally analyzes one row at a time. The sought set of all typical testors is initialized with the typical testors embedded in the first row of the basic matrix and then, it is updated with each new basic matrix row the algorithm receives. Second, by processing only those basic matrix elements with value 1, and rearranging all basic matrix rows prior to starting the search procedure, the YYC algorithm achieves better efficiency than the other tested external TTA.

By proceeding with its incremental strategy, and by taking advantage of the compatible set concept, YYC strictly tests for those column combinations that show the highest possibility to conform a typical testor, unlike external algorithms whose ordering over the power set of columns severely limits their performance. We also proposed an efficient procedure to find a compatible set within the reference sub-matrix determined by the previously known typical testors and the elements with value 1 within the next basic matrix row. This procedure not only reduces the search space, but also takes advantage of the local properties of a compatible set to establish a one-pass efficient algorithm.

Four experiments were designed and run to comparatively test the performance of the YYC algorithm against some widely used external TTA. Each experiment targeted a different scenario regarding the number of rows, columns, and typical testors to be found. Also, the last experiment aimed at discovering a possible relationship between the density of the basic matrix and the performance of the YYC algorithm. Evidently, all possible structural characteristics for the initial basic matrix are not accounted for by the presented experiments, so a general best-performance claim is not appropriate. Nevertheless, experimental experience with previous TTA, both internal and external, seems to show that there is no TTA that can run on any basic matrix configuration and always yield the best possible performance.

In conclusion, this paper strengthens the idea that research on testor theory is far from done. During its early days internal TTA dominated the scene. Later, on the argument of a better performance, external TTA stood out. Now we look again at some properties of the internal TTA that can open new directions and trends in testor theory that still has to fill the gap between theoretical developments and practical implementations.

References

[1] Alba-Cabrera, E., Santana, R., Ochoa-Rodriguez, A., Lazo-Cortes, M.: Finding typical testors by using an evolutionary strategy. Proceedings of the V Ibero American Symposium on Pattern Recognition., 267-278. (2000)

[2] Alba-Cabrera E., Guilcapi, D., Ibarra-Fiallo, J.: New strategies for evaluating the performance of typical testor algorithms. In: Lecture Notes in Computer Science, Volume 7441, 813-820 (2012)

[3] Alba-Cabrera E., Ibarra-Fiallo J., Godoy-Calderon, S.: A Theoretical and Practical Framework for Assessing the Computational Behavior of Typical Testor-Finding Algorithms. In: Lecture Notes in Computer Science, Volume 8258, pp. 351–358 (2013)

[4] Diaz-Sanchez, G., Piza-Davila, I., Sanchez-Diaz, G., Mora-Gonzalez, M., Reyes- Cardenas, O., Cardenas-Tristan, A., Aguirre-Salado, C.: Typical Testors Generation Based on an Evolutionary Algorithm. Lecture Notes in Computer Science. 936, 58- 65 (2011)

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

[6] Lazo-Cortes, M., Ruiz-Shulcloper, J., Alba-Cabrera, E.: An overview of the evolution of the concept of testor. Pattern Recognition 34 (4), 753–762 (2001)

[7] Li, F., Zhu, Q.: Document clustering in research literature based on NMF and testor theory. Journal of Software. 6 (1), 78-82 (2011)

[8] Lias-Rodriguez, A., Pons-Porrata, A.: BR: A New Method for Computing All Typical Testors. Lecture Notes in Computer Science, Volume 5856, 433-440 (2009)

[9] Ortiz-Posadas M, Martinez-Trinidad F., Ruiz-Shulcloper J.: A new approach to differential diagnosis of diseases. International Journal of Biomedical Computing, Vol. 40, No. 3, 179-185 (2001).

[10] Pons-Porrata A., Ruiz-Shulcloper J., Berlanga-Llavori R.: A method for the automatic summarization of topic-based clusters of documents, Proc. VIII Iberoamerican Conference on Pattern Recognition, LNCS 2905, Springer, 596-603 (2003).

[11] Pons-Porrata, A., Gil-Garcia, R., Berlanga-Llavori, R.: Using Typical Testors for Feature Selection in Text Categorization. Rueda, L., Mery, D., Kittler, J. (eds.) CIARP 2007. LNCS, vol. 4756, 643–652. Springer, Heidelberg (2007)

[12] Rojas, A., Cumplido, R., Carrasco-Ochoa, J. A., Feregrino, C., Martínez-Trinidad, J. Fco.: Hardware-software platform for computing irreducible testors. Expert Systems With Applications 39, no. 2, 2203-2210, (2012).

[13] Ruiz-Shulcloper, J.; Bravo, M.; Aguila, F.: Algoritmos BT y TB para el cálculo de todos los tests típicos. Revista Ciencias Matemáticas, 6 (2) (1982)

[14] Sanchez-Diaz G., Lazo-Cortes M., Piza-Davila I.: A fast implementation for the typical testor property identification based on an accumulative binary tuple. International Journal of Computational Intelligence Systems, Vol. 5, No. 6, (2012)

[15] Santiesteban-Alganza, Y., Pons-Porrata, A.: LEX: A new algorithm for calculating typical testors. Revista Ciencias Matematicas, 21 (1), 85-95, (2003)

[16] Vazquez-R., Godoy-Calderon S.: Using testor theory to reduce the dimension of neural network models. Special Issue in Neural Networks and Associative Memories. 28, 93-103 (2007)