# YYC: A Fast Performance Incremental Algorithm for Finding Typical Testors*

Eduardo Alba-Cabrera[1], Julio Ibarra-Fiallo[1],
Salvador Godoy-Calderon[2], and Fernando Cervantes-Alonso[3]

[1] Universidad San Francisco de Quito, Colegio de Ciencias e Ingeniería,
Diego de Robles y Vía Interoceánica, Quito, Ecuador
[2] Instituto Politécnico Nacional-Centro de Investigación en Computación (CIC)
Av. Juan de Dios Bátiz, Esq. Miguel Othón de Mendizábal. Col. Nueva
Industrial Vallejo. D.F., México
[3] CITRIX Systems
{ealba,jibarra}@usfq.edu.ec, sgodoyc@cic.ipn.mx,
fernando.cervantes@citrix.com
http://www.usfq.edu.ec,
http://www.cic.ipn.mx,
http://www.citrix.com

**Abstract.** The last few years have seen an important increase in research publications dealing with external typical testor-finding 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 the currently analized set of basic matrix rows by searching for compatible sets. The experimentally measured performance of this algorithm stands out favorably in problems where other external algorithms show very low performance. Also, a comparative analysis of its efficiency is done against some external typical testor-finding algorithms published during the last few years.

**Keywords:** Feature selection, Testor Theory, typical testor algorithms.

## 1 Introduction

Testor Theory [5], has repeatedly proven itself as one of the most useful tools for feature selection problems in pattern recognition [4]. Typical testors have been used for solving a wide range of practical problems like diagnosis of diseases [8], text categorization [10], document summarization [9] and document clustering [6]. Testor theory has had an important growth over the past ten years, particularly regarding the development of new algorithms, like [3,7,13,14]. All

* This research was supported by the Program of Collaboration Grants from Universidad San Francisco de Quito, Ecuador. The third author is also grateful for the finantial support by Instituto Politécnico Nacional and CONACyT, México, particularly through project SIP 20141492.

E. Bayro-Corrochano and E. Hancock (Eds.): CIARP 2014, LNCS 8827, pp. 416–423, 2014.

those algorithms are generally refereed to as Typical Testor-finding Algorithms ($TTA$).

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. Interestingly, in almost all published algorithms during the last ten years, researchers have proposed external algorithms. There are even cases of internal algorithms which were adapted to operate externally, as is the case for [13].

The performance of both, external and internal $TTA$ 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. 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 $YYC$ ($Yablonsky$ & *Compatible sets*)[1], that progressively identifies the set of all typical testors contained within a range of basic matrix rows comprasing from the first and up to the current row. implied by each row of the basic matrix. 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 supervised 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

[1] In honor of *Sergey Yablonsky*, who published the first studies of testor theory (for more information see [5]), and the term *compatible sets*, which are crucial for the proposed algorithm.

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 those compared objects.

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 supervised 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.

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 $DM$ [11].

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: 1) 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. 2) 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. preserves its property of being a testor), 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 the $YYC$ algorithm is that, instead of analyzing the whole basic matrix in order to determine the set $\psi^*$ 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 value 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 $BM$
*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) then (Hit)
                  Add τ_j ∪ {x_p} to ψAux
     Let ψ* = ψAux
Return ψ*
```

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 function: *Sum*(<*vector*>), were *vector* can be either a row or a column fron the $BM$, which returns the sum of elements in the provided vector. Using this function we propose the following algorithm:

Algorithm 2: $FindCompatibleSet\,(\tau,\ x_p)$
*input* : $\tau$ 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_p},
       and the current read rows.
Evaluate Condition1 to be TRUE iff |{r_s ∈ RefSM, Sum (r_q) = 1}| ≥ |τ ∪ {x_p}|
Redefine RefSM = {r_s ∈ RefSM, Sum (r_s) = 1}
Evaluate Condition2 to be TRUE iff (∀x_k ∈ RefSM) [Sum (x_k) = 1]
Return the truth value of the Boolean expression [Condition1 AND Condition2]
```

Undoubtedly, several different procedures can be defined to find if a submatrix 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 $LEX$ [14], $FastCTExt$ [13], and $BT$ [12]. For each experiment, a custom basic matrix was designed, following the method described in [2], 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 feature 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 feature subsets tested by the $TTA$ (labeled as $Hits$). 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.

Table 1 clearly shows how the $YYC$ algorithm slightly outperforms the $LEX$ and $BT$ algorithms whose performance turns out to be exactly the 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 the higher efficiency of $YYC$ 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

**Table 1.** Comparative performance test against identity matrices

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

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

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

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

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

increase in the number of typical testors to find. Table 2 summarizes experiment two's results

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.

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 $YYC$ 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.

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

**Table 4.** Performance test with different matrix densities

| Rows | Cols | *Density* = 0.3 | | | *Density* = 0.4 | | | *Density* = 0.6 | | | *Density* = 0.7 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | TT | Hits | Efficiency | TT | Hits | Efficiency | TT | Hits | Efficiency | TT | Hits | Efficiency |
| 4 | 5 | 2 | 5 | 40.00% | 4 | 7 | 57.14% | 4 | 9 | 44.44% | 8 | **11** | **72.73%** |
| 4 | 25 | 750 | 1525 | 49.18% | 500 | **675** | **74.07%** | 180 | 525 | 34.29% | 200 | 275 | 72.73% |
| 4 | 50 | 11000 | 22100 | 49.77% | 4000 | **5200** | **76.92%** | 1210 | 3600 | 33.61% | 800 | 1100 | 72.73% |
| 4 | 75 | 54000 | 108225 | 49.90% | 13500 | **17325** | **77.92%** | 3840 | 11475 | 33.46% | 1800 | 2475 | 72.73% |
| 4 | 100 | 168000 | 336400 | 49.94% | 32000 | **40800** | **78.43%** | 8820 | 26400 | 33.41% | 3200 | 4400 | 72.73% |

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: incremental row-by-row analysis of the basic matrix, and high efficiency. The 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. Also, by processing only those basic matrix elements with value 1, 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. In: Proceedings of the V Ibero American Symposium on Pattern Recognition, pp. 267–278 (2000)
2. 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: Ruiz-Shulcloper, J., Sanniti di Baja, G. (eds.) CIARP 2013, Part I. LNCS, vol. 8258, pp. 351–358. Springer, Heidelberg (2013)
3. 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. In: Yin, H., Wang, W., Rayward-Smith, V. (eds.) IDEAL 2011. LNCS, vol. 6936, pp. 58–65. Springer, Heidelberg (2011)
4. Lazo-Cortes, R.-S.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)
5. 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)
6. Li, F., Zhu, Q.: Document clustering in research literature based on NMF and testor theory. Journal of Software 6(1), 78–82 (2011)
7. Lias-Rodríguez, A., Pons-Porrata, A.: BR: A New Method for Computing All Typical Testors. In: Bayro-Corrochano, E., Eklundh, J.-O. (eds.) CIARP 2009. LNCS, vol. 5856, pp. 433–440. Springer, Heidelberg (2009)
8. Ortiz-Posadas, M., Martinez-Trinidad, F., Ruiz-Shulcloper, J.: A new approach to differential diagnosis of diseases. International Journal of Biomedical Computing 40(3), 179–185 (2001)
9. Pons-Porrata, A., Ruiz-Shulcloper, J., Berlanga-Llavori, R.: A Method for the Automatic Summarization of Topic-Based Clusters of Documents. In: Sanfeliu, A., Ruiz-Shulcloper, J. (eds.) CIARP 2003. LNCS, vol. 2905, pp. 596–603. Springer, Heidelberg (2003)
10. Pons-Porrata, A., Gil-García, R.J., Berlanga-Llavori, R.: Using typical testors for feature selection in text categorization. In: Rueda, L., Mery, D., Kittler, J. (eds.) CIARP 2007. LNCS, vol. 4756, pp. 643–652. Springer, Heidelberg (2007)
11. 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(2), 2203–2210 (2012)
12. 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)
13. 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 5(6) (2012)
14. Santiesteban-Alganza, Y., Pons-Porrata, A.: LEX: A new algorithm for calculating typical testors. Revista Ciencias Matematicas 21(1), 85–95 (2003)